

Introducción a optimización del rendimiento de las consultas

Cuando una aplicación envía una consulta a SQL Server, el optimizador de consultas del motor de base de datos la compila y genera un plan de ejecución, almacena el plan en la memoria caché, y luego lo ejecuta. Para analizar el rendimiento de las consultas podemos revisar sus planes de ejecución y manipular el modo en que el motor de base de datos los ejecuta.

¿Cómo trabaja el optimizador de consultas de SQL Server?

El optimizador de consultas recibe como entradas: la consulta, las definiciones de las tablas e índices, y las estadísticas de distribución de los datos, y entrega como salida el **plan de ejecución** de la consulta.

Para una consulta puede haber muchos planes de ejecución. El optimizador de consultas es un optimizador basado en el costo, por lo que de todos los planes posibles elige el de menor costo. En ocasiones una consulta compleja puede tener muchísimos planes de ejecución posibles, en este caso, el optimizador utiliza algoritmos complejos para seleccionar el plan de ejecución de menor costo posible.

¿Cómo se procesa una consulta?

El plan de ejecución tiene los siguientes elementos básicos:

- Secuencia en la que se tiene acceso a las tablas.
- Métodos a utilizar para extraer los datos de las tablas: **table scan** (recorre toda la tabla), **index seek** (utiliza un índice para seleccionar ciertas filas), **index scan** (recorre todo el índice)

SQL Server procesa la consulta según los pasos siguientes:

1. El analizador examina la instrucción SELECT y la divide en unidades lógicas como palabras clave, expresiones, operadores e identificadores.
2. Se genera un árbol de secuencia de la consulta que muestra los pasos lógicos necesarios para obtener el formato del conjunto de resultados a partir de los datos de origen.
3. El optimizador de consulta analiza las diferentes formas de acceso a los datos y selecciona la secuencia que entrega los resultados en la forma más rápida y usando la menor cantidad de recursos. Se actualiza el árbol de secuencia de la consulta que constituye lo que se conoce como plan de ejecución.
4. El motor relacional ejecuta el plan de ejecución solicitándole los datos al motor de almacenamiento.
5. El motor relacional procesa los datos y se los entrega a la aplicación cliente.

Analizar una consulta revisando el gráfico del plan de ejecución

Plan de ejecución estimado: analiza la consulta y muestra un costo estimado de su ejecución.

Plan de ejecución real: ejecuta la consulta y muestra el costo real de su ejecución.

Nota: antes de empezar a ejecutar los ejemplos de este documento, iniciaremos una traza para hacerle seguimiento a las operaciones que ejecutemos. Para ello:

1. Inicie **SQL Server Profiler**.
2. Menú **Archivo, Nuevo seguimiento**, iniciar sesión en el servidor.
3. Ventana **Propiedades de seguimiento**, ficha **General**.
4. **Nombre de seguimiento:** *Traza_Total*.
5. **Usar la plantilla:** **TSQL**
6. Casilla **Guardar en el archivo:** marcada, especificar nombre y ubicación del archivo de traza.
7. Clic en **Ejecutar** para iniciar la traza.

A partir de este momento, todas las instrucciones SQL que ejecutemos quedarán registradas en la traza para su posterior análisis.

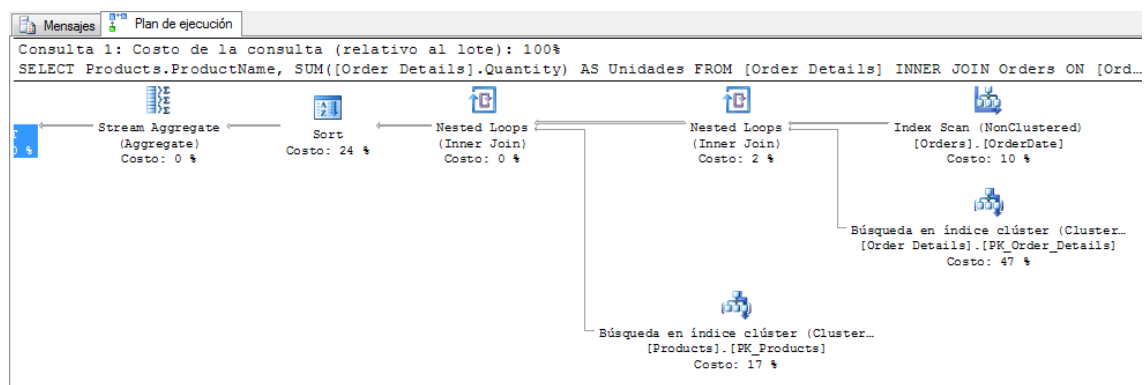
Ejemplo

En su ventana de consulta de SQL Server Management Studio escriba la siguiente instrucción:

```
SELECT Products.ProductName,  
       SUM([Order Details].Quantity) AS Unidades  
FROM [Order Details] INNER JOIN Orders  
     ON [Order Details].OrderID = Orders.OrderID  
INNER JOIN Products  
     ON [Order Details].ProductID = Products.ProductID  
WHERE MONTH(Orders.OrderDate) = 8  
      AND YEAR(Orders.OrderDate) = 1996  
GROUP BY Products.ProductName  
go
```

En el menú **Consulta** seleccione **Mostrar plan de ejecución estimado** o haga clic en el botón correspondiente de la barra de herramientas.

Para ver el plan de ejecución, haga clic en la ficha **Plan de ejecución** del panel de resultados. La salida del gráfico del plan de ejecución se lee de derecha a izquierda y de arriba a abajo.






Tenga en cuenta las siguientes directrices para interpretar el plan de ejecución:






- Cada ícono representa un nodo del árbol de secuencia y especifica los operadores físico y lógico utilizados para ejecutar esa parte de la consulta.
- Los nodos que tienen el mismo nodo principal se dibujan en la misma columna.
- El ancho de la flecha es proporcional al número de filas afectadas.

Información mostrada en un nodo

Elemento	Descripción
Operación física	Operación física utilizada. Si se muestra en color rojo significa que el optimizador generó una advertencia.
Operación lógica	Operador lógico que representa la operación física.
Tamaño de fila estimado	Tamaño de la fila creada por el operador, en bytes.
Costo de E/S estimado	Costo estimado de E/S para la operación.
Costo de CPU estimado	Costo estimado de CPU para la operación.
Costo de operador estimado	Costo del optimizador de consultas para ejecutar la operación. Se muestra entre paréntesis.
Costo de subárbol estimado	Costo total del optimizador para ejecutar esta operación y todas las anteriores en el mismo subárbol.
Número de filas estimado / Número de filas	Número de filas producida por el operador.

Operadores

Operador	Descripción
Table Scan 	Especifica que el optimizador ha recorrido la tabla. Se presenta cuando la tabla es un heap (tabla sin índice clustered) o cuando la tabla es pequeña y leerla directamente produce menos carga que leerla usando el índice. Las tablas grandes no deberían leerse vía Table Scan.
Clustered Index Scan / Nonclustered Index Scan 	Especifica que el optimizador ha recorrido el índice clustered / índice nonclustered para recuperar las filas de la tabla.
Clustered Index Seek / Nonclustered Index Seek 	Especifica que el optimizador ha ejecutado una búsqueda indexada para recuperar filas específicas de la tabla.

Nested Loops 	<p>Especifica que el optimizador ha ejecutado un join. El bucle externo lee cada una de las filas de la tabla externa; el bucle interno busca filas coincidentes en la tabla interna para cada una de las filas de la tabla externa. Es eficaz cuando la tabla externa es pequeña, y la tabla interna es grande y está indexada.</p>
Merge Join 	<p>Especifica que el optimizador ha ejecutado un join. Se presenta con conjuntos moderados de datos ordenados. Incluso si no están ordenados, el optimizador puede concluir que es más rápido ordenar los datos y luego combinarlos antes que ejecutar un Nested Loops.</p>
Hash Match 	<p>Especifica que el optimizador ha ejecutado un join. Se presenta con grandes volúmenes de datos. Una de las tablas se utiliza para generar la tabla de valores hash, y la segunda tabla para sondear la tabla hash buscando valores coincidentes.</p>
Stream Aggregate 	<p>Especifica que el optimizador ha ejecutado una agregación por una o varias columnas. La data debe estar ordenada por dichas columnas por lo que previamente, de ser necesario, ejecutará una operación Sort.</p>
Sort 	<p>Especifica que el optimizador ha ordenado todas las filas.</p>

Mostrar el plan de ejecución con una opción SET

Ejecute en su ventana de consultas el siguiente código:

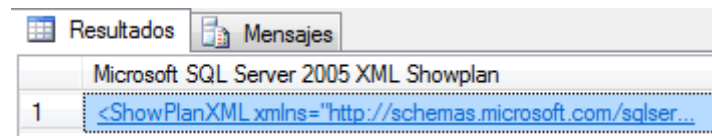
```
USE Northwind
go

SET SHOWPLAN_XML ON
go

SELECT Products.ProductName,
       SUM([Order Details].Quantity) AS Unidades
FROM [Order Details] INNER JOIN Orders
  ON [Order Details].OrderID = Orders.OrderID
INNER JOIN Products
  ON [Order Details].ProductID = Products.ProductID
WHERE MONTH(Orders.OrderDate) = 8
      AND YEAR(Orders.OrderDate) = 1996
GROUP BY Products.ProductName
go

SET SHOWPLAN_XML OFF
go
```

Se muestra lo siguiente en el panel de resultados:



Haga clic sobre el enlace para ver el plan de ejecución almacenado en un documento XML.

La característica "buscar índices que faltan"

Esta característica utiliza objetos de administración dinámica y los planes de ejecución para identificar los índices que faltan en la base de datos y cuya implementación puede mejorar el rendimiento de las consultas.

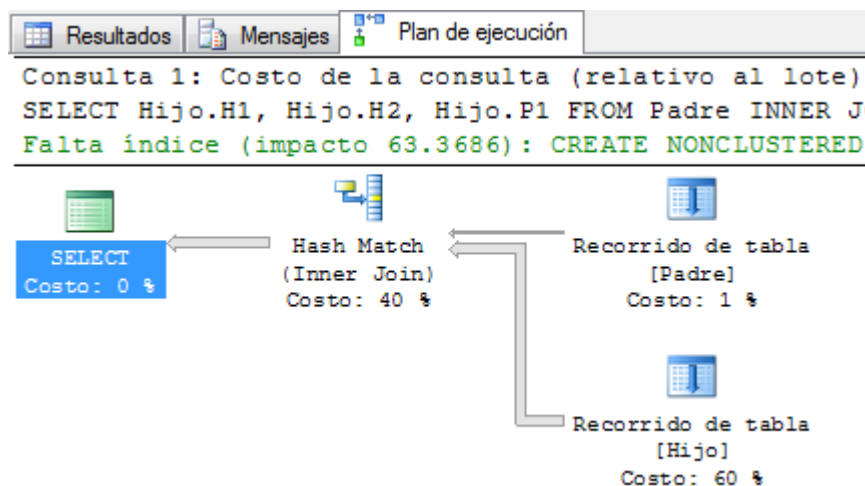
Cuando el optimizador de consultas procesa un requerimiento, identifica cuáles son los mejores índices para una condición de filtro específica. Si estos índices no existen, genera un plan de ejecución de menor calidad y registra la información de los índices que faltan en objetos de administración dinámica que podemos consultar para decidir si se implementan o no dichos índices.

Ejemplo

Abra y ejecute el script **CreaBD_Pruebas**. Este script crea la base de datos **Pruebas** con 2 tablas sin restricciones: tabla **Padre** y tabla **Hijo**. En la tabla **Padre** inserta 100 registros, y en la tabla **Hijo** inserta 100000 registros relacionados con los registros de la tabla **Padre**.

Ejecute la siguiente consulta, cuyo rendimiento deseamos mejorar, de modo que se vea su plan de ejecución:

```
SELECT Hijo.H1, Hijo.H2, Hijo.P1
FROM Padre INNER JOIN Hijo
      ON Padre.P1 = Hijo.P1
WHERE Padre.P1 > 50 AND Padre.P1 < 70
go
-- costo: 0,512594
```



Observe que en **Padre** e **Hijo** el acceso a los datos es mediante **Table Scan**.

Estableciendo la integridad referencial entre Padre e Hijo

Ejecute las siguientes instrucciones para definir la relación entre **Padre** e **Hijo**:

```
ALTER TABLE Padre

    ADD PRIMARY KEY(P1)

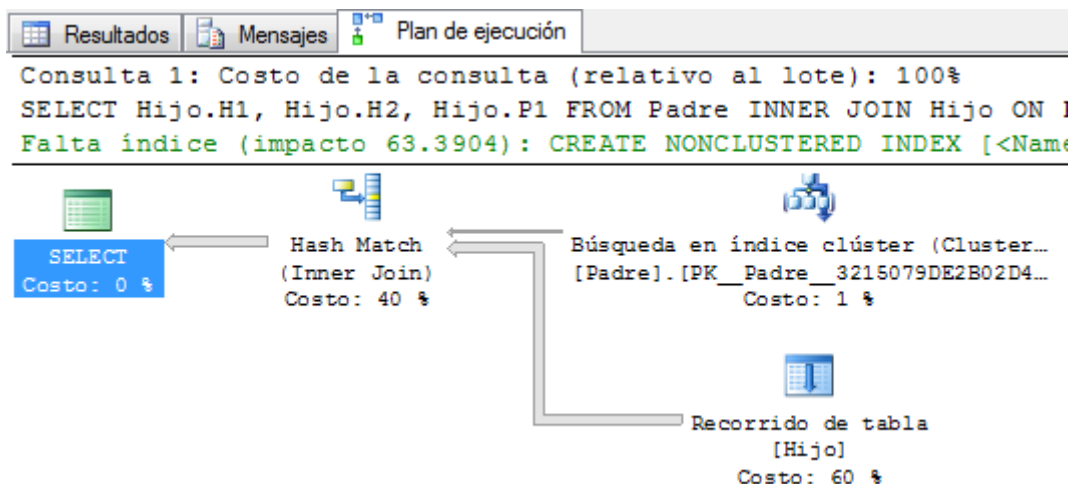
go

ALTER TABLE Hijo
    ADD FOREIGN KEY(P1)
    REFERENCES Padre

go
```

Ejecute nuevamente la consulta inicial mostrando el plan de ejecución para ver si el rendimiento ha mejorado.

```
SELECT Hijo.H1, Hijo.H2, Hijo.P1
FROM Padre INNER JOIN Hijo
    ON Padre.P1 = Hijo.P1
WHERE Padre.P1 > 50 AND Padre.P1 < 70
go
-- costo: 0,512418
```



El costo de la consulta casi no ha variado, pero ahora se accede a la tabla **Padre** mediante un **Clustered Index Seek**.

Consulta que devuelve los índices que faltan ordenados de mayor a menor impacto en el rendimiento

La siguiente consulta lee el objeto de administración dinámica **sys.dm_db_missing_index_group_stats** y muestra los índices que faltan.

```
SELECT group_handle, user_seeks, user_scans,
    avg_total_user_cost, avg_user_impact
FROM sys.dm_db_missing_index_group_stats
ORDER BY avg_total_user_cost * avg_user_impact *
    (user_seeks + user_scans) DESC;

go
```

	group_handle	user_seeks	user_scans	avg_total_user_cost	avg_user_impact
1	68	1	0	0,512417851851852	98,31
2	70	1	0	0,512417851851852	63,39

- **group_handle:** identifica al grupo del índice que falta.
- **user_seeks:** número de búsquedas indexadas iniciadas por consultas de usuarios para las que se podría haber utilizado el índice.
- **user_scans:** número de recorridos de índice iniciados por consultas de usuarios para los que se podría haber utilizado el índice.
- **avg_total_user_cost:** costo medio de las consultas de los usuarios.
- **avg_user_impact:** beneficio porcentual medio que se podría obtener si se implementa el índice.

Consulta que muestra los detalles de columna para el índice que falta

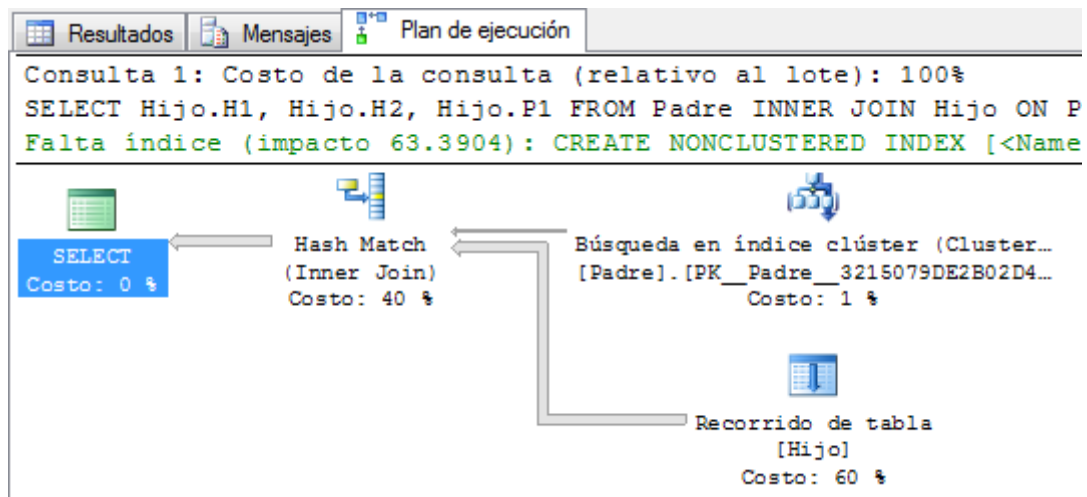
La siguiente consulta lee los objetos de administración dinámica **sys.dm_db_missing_index_group_stats**, **sys.dm_db_missing_index_groups**, y **sys.dm_db_missing_index_details** para el índice cuyo **group_handle** es **68**, y muestra los detalles del índice.

```
SELECT migs.group_handle, mid.*
FROM sys.dm_db_missing_index_group_stats AS migs
INNER JOIN sys.dm_db_missing_index_groups AS mig
    ON (migs.group_handle = mig.index_group_handle)
INNER JOIN sys.dm_db_missing_index_details AS mid
    ON (mig.index_handle = mid.index_handle)
WHERE migs.group_handle = 68;
```

	group_handle	index_handle	database_id	object_id	equality_columns	inequality_columns	included_columns	statement
1	68	67	18	261575970	[P1]	NULL	[H1], [H2]	[Pruebas].[dbo].[Hijo]

- **index_handle:** identifica al índice.
- **database_id:** identifica la base de datos a la que pertenece la tabla en la que falta el índice.
- **object_id:** identifica la tabla en la que falta el índice.
- **equality_columns / inequality_columns:** columnas del filtro.
- **included_columns:** columnas requeridas por la consulta que pueden incluirse en el índice que falta.
- **statement:** nombre de la tabla en la que falta el índice.

Ver el plan de ejecución antes de crear el índice



Según el plan de ejecución, el costo de la consulta es 0,512418.

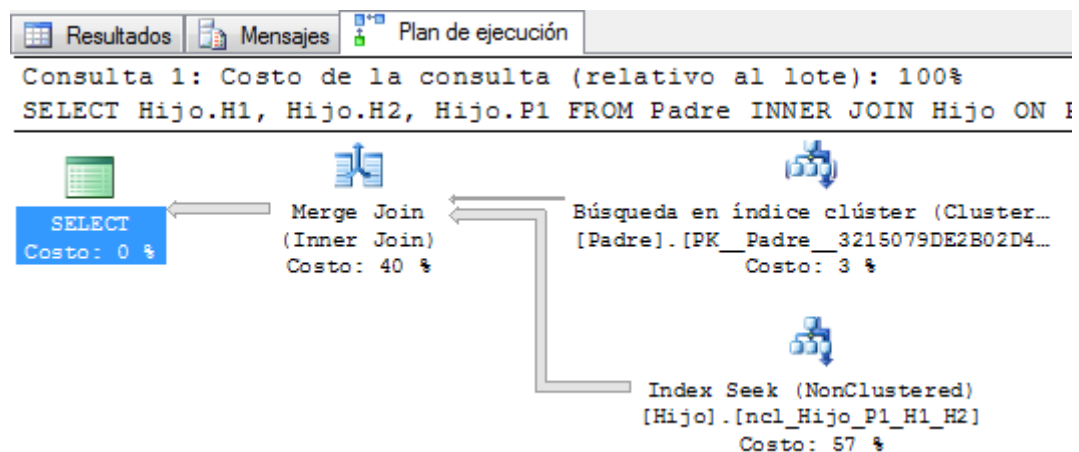
Creando el índice que falta

Ejecute la siguiente instrucción para crear el índice que falta:

```
CREATE NONCLUSTERED INDEX ncl_Hijo_P1_H1_H2
    ON Hijo(P1, H1, H2)
go
```

Ver el plan de ejecución después de crear el índice

```
SELECT Hijo.H1, Hijo.H2, Hijo.P1
FROM Padre INNER JOIN Hijo
    ON Padre.P1 = Hijo.P1
WHERE Padre.P1 > 50 AND Padre.P1 < 70
go
-- costo: 0,118185
```



Observe que el rendimiento ha mejorado bastante, el acceso a las tablas **Padre** e **Hijo** es mediante **Clustered Index Seek** y **Nonclustered Index Seek** respectivamente, y que la combinación es un **Merge Join** en razón de que los dos conjuntos de datos están ordenados.

Analizando el tiempo de ejecución de las consultas

En este punto, haremos uso de la instrucción SET STATISTICS TIME para habilitar las estadísticas de tiempo y analizar el tiempo que le toma al motor de datos ejecutar una consulta.

Codifique cada uno de los ejercicios siguientes en una ventana query separada y ejecute las instrucciones una por una para ver mejor los mensajes entregados por el sistema.

Ejercicio 1: Combinación de tablas sin integridad referencial ni índices

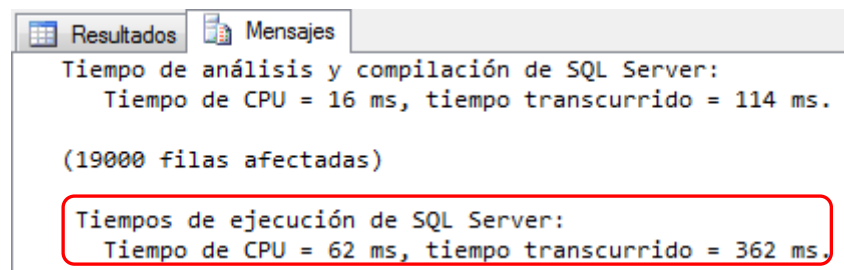
Ejecute el script **CreaBD_Pruebas.sql** que crea la base de datos sin restricciones. Luego,

```
USE Pruebas
go

SET STATISTICS TIME ON
go

-- tablas sin integridad referencial y sin índices
SELECT Hijo.H1, Hijo.H2, Hijo.P1
FROM Padre INNER JOIN Hijo
    ON Padre.P1 = Hijo.P1
WHERE Padre.P1 > 50 AND Padre.P1 < 70
go

SET STATISTICS TIME OFF
go
```



Ejercicio 2: Combinación de tablas sin integridad referencial ni índices usando SELECT *

En este ejercicio usaremos sentencias DBCC para limpiar el caché de planes de ejecución y el buffer de datos antes de ejecutar la consulta, de modo que su rendimiento no se vea afectado por la ejecución anterior de consultas similares.

- **DBCC FREEPROCCACHE:** limpia el caché de planes de ejecución.
- **DBCC DROP CLEANBUFFERS:** limpia el buffer de datos.
- **CHECKPOINT:** escribe en el disco la data confirmada presente en el buffer de datos.

Estas instrucciones no deben ejecutarse en bases de datos en producción. Se utilizan para pruebas de diagnóstico.

```

USE Pruebas
go

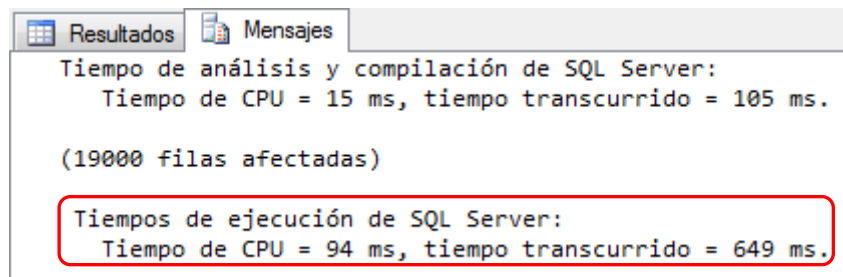
-- limpiando los buffers en la memoria
DBCC FREEPROCCACHE
go
CHECKPOINT
go
DBCC DROPCLEANBUFFERS
go

SET STATISTICS TIME ON
go

-- tablas sin integridad referencial y sin índices
SELECT Hijo.*
FROM Padre INNER JOIN Hijo
      ON Padre.P1 = Hijo.P1
WHERE Padre.P1 > 50 AND Padre.P1 < 70
go

SET STATISTICS TIME OFF
go

```



Observe que la consulta `SELECT *` de este ejercicio entrega los mismos datos que la consulta del ejercicio anterior, pero consume más recursos. Esto se debe a que el motor de datos debe leer primero la estructura de la tablas antes de ejecutar la sentencia.

Ejercicio 3: Combinación de tablas con integridad referencial

```

USE Pruebas
go

ALTER TABLE Padre
      ADD PRIMARY KEY(P1)
go

ALTER TABLE Hijo
      ADD FOREIGN KEY(P1)
      REFERENCES Padre
go

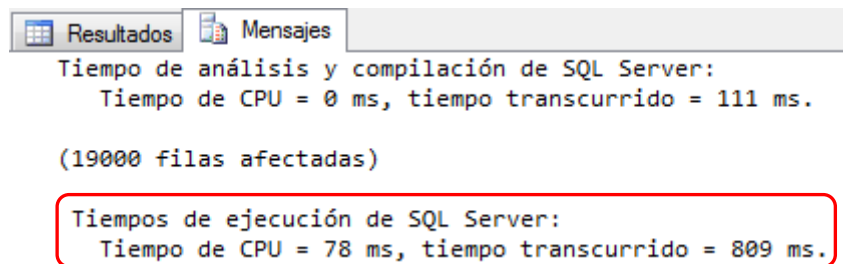
```

```
-- limpiando los buffers en la memoria
DBCC FREEPROCCACHE
go
CHECKPOINT
go
DBCC DROPCLEANBUFFERS
go

SET STATISTICS TIME ON
go

-- tablas con integridad referencial
SELECT Hijo.H1, Hijo.H2, Hijo.P1
FROM Padre INNER JOIN Hijo
    ON Padre.P1 = Hijo.P1
WHERE Padre.P1 > 50 AND Padre.P1 < 70
go

SET STATISTICS TIME OFF
go
```



Observe que en este caso, la integridad referencial no mejora el rendimiento de la consulta. La integridad referencial es básicamente una herramienta para garantizar la consistencia de los datos.

Ejercicio 4: Combinación de tablas con integridad referencial e índice en la clave foránea

```
USE Pruebas
go

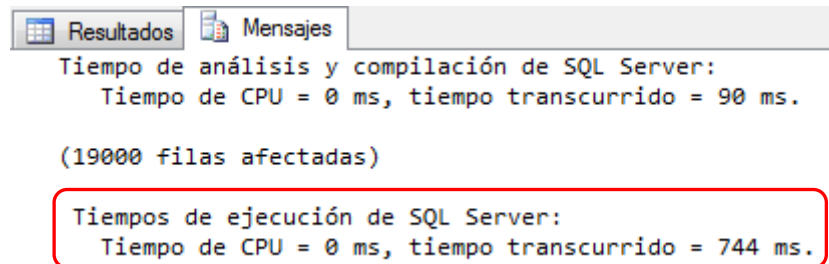
CREATE NONCLUSTERED INDEX ncl_Hijo_P1_H1_H2
    ON Hijo(P1, H1, H2)
go

-- limpiando los buffers en la memoria
DBCC FREEPROCCACHE
go
CHECKPOINT
go
DBCC DROPCLEANBUFFERS
go

SET STATISTICS TIME ON
go
```

```
-- tablas con integridad referencial e índices
SELECT Hijo.H1, Hijo.H2, Hijo.P1
FROM Padre INNER JOIN Hijo
      ON Padre.P1 = Hijo.P1
WHERE Padre.P1 > 50 AND Padre.P1 < 70
go

SET STATISTICS TIME OFF
go
```



Observe que el tiempo de CPU se reduce de manera notable.

Ejercicio 5: Combinación definida en la cláusula WHERE

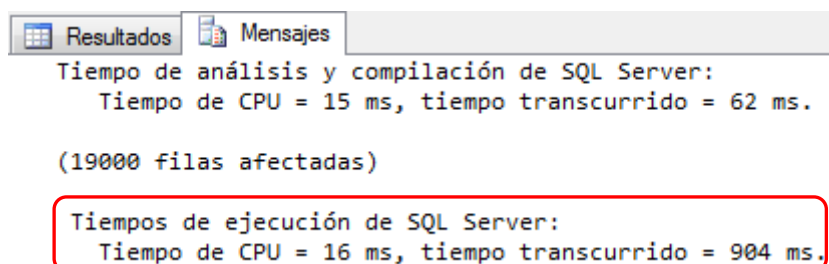
```
USE Pruebas
go

-- limpiando los buffers en la memoria
DBCC FREEPROCCACHE
go
CHECKPOINT
go
DBCC DROPCLEANBUFFERS
go

SET STATISTICS TIME ON
go

-- combinación definida con WHERE
SELECT Hijo.H1, Hijo.H2, Hijo.P1
FROM Padre, Hijo
WHERE Padre.P1 = Hijo.P1
      AND Padre.P1 > 50 AND Padre.P1 < 70
go

SET STATISTICS TIME OFF
go
```



Cuando la combinación se especifica en la cláusula WHERE, el motor de datos ejecuta el producto cartesiano de las tablas a combinar, y luego filtra las filas según la condición de la combinación; es por ello que consume mucho más tiempo.

Ejercicio 6: Comparando datos de 2 tablas

En este ejercicio utilizaremos la base de datos **MarketPERU** y ejecutaremos una consulta que liste los productos que se encuentran en la tabla **Producto**, pero no se encuentran en la tabla **Guia_Detalle**.

Obtendremos el conjunto de resultados con una consulta OUTER JOIN, luego con una subconsulta, y finalmente con una subconsulta correlacionada. No olvide limpiar la memoria antes de ejecutar cada una de las consultas.

```
USE MarketPERU
go

-- limpiando los buffers en la memoria
DBCC FREEPROCCACHE
go
CHECKPOINT
go
DBCC DROPCLEANBUFFERS
go

SET STATISTICS TIME ON
go

-- LEFT OUTER JOIN
SELECT Producto.idProducto, Producto.nombre
FROM Producto LEFT OUTER JOIN Guia_Detalle
    ON Producto.idProducto = Guia_Detalle.idProducto
WHERE Guia_Detalle.cantidad IS NULL
go

-- Subconsulta
SELECT idProducto, nombre
FROM Producto
WHERE idProducto NOT IN (
    SELECT idProducto FROM Guia_Detalle)
go

-- Subconsulta correlacionada
SELECT Producto.idProducto, Producto.nombre
FROM Producto
WHERE NOT EXISTS (
    SELECT Guia_Detalle.idProducto FROM Guia_Detalle
    WHERE Producto.idProducto = Guia_Detalle.idProducto)
go

SET STATISTICS TIME OFF
go
```

El siguiente cuadro muestra los tiempos de ejecución:

Tipo de consulta	Tiempo de CPU	Tiempo transcurrido
LEFT OUTER JOIN	0	4
Subconsulta (NOT IN)	16	17
Subconsulta correlacionada (NOT EXISTS)	16	11

Por lo general, un JOIN tiene mejor rendimiento que una subconsulta.

Conclusiones

De los ejercicios desarrollados podemos inferir las siguientes recomendaciones:

- No usar SELECT *.
- Seleccionar solo las columnas que se requieren ya que cada columna adicional requiere tiempo extra de procesamiento.
- Utilice índices para las columnas relacionadas y para las columnas en las que se ejecuta búsqueda.
- Use los operadores JOIN para definir las combinaciones en vez de especificarlas en la cláusula WHERE.
- Prefiera el uso de JOIN antes que subconsultas.

Por otro lado, tenga en cuenta que si bien los índices pueden optimizar el rendimiento de las consultas, hacen que el tiempo requerido para las operaciones de actualización de datos e inserción y eliminación de filas sea mayor.