ISIL APRENDE HACIENDO

SQL Server 2012

Optimización de Consultas

Temas

1 Introducción

Uno de los temas más importantes en el desarrollo de bases de datos es el tiempo de procesamiento y ejecución. Cada día se necesita procesar mayor cantidad de datos y obtener de manera más rápida y precisa la información.

Muchos de los problemas de rendimiento se deben entre otras cosas al hardware, al software, al motor de base de datos y por sobre todo al diseño, índices y mala formulación de consultas SQL.

En este documentos trataremos sobre la formulación de consultas, que siguiendo algunas recomendaciones veremos que se puede mejorar el tiempo de respuesta de nuestro motor de BD significativamente.

2 Diseño de la base de datos

2.1 Las tablas

- Las tablas normalizadas permiten reducir al mínimo el espacio ocupado por nuestra base de datos y permiten asegurar la consistencia de la información al mismo tiempo que son muy rápidas para la realización de transacciones, pero generan un mayor tiempo a la hora de consultarlas ya que se deben realizar generalmente la unión de varias tablas, por lo que en caso de necesidad de altas velocidades de respuesta con grandes volúmenes de datos un modelo desnormalizado es más que conveniente teniendo en cuenta todas las implicancias del caso.
- Ajustar al máximo el tamaño de los campos ayuda a no desperdiciar espacio.
- Eliminar todo campo que no sea de utilidad ya que por más que no contenga datos genera retrasos.

2.2 Los índices

- Los índices son campos que permiten la búsqueda a partir de dicho campo a una velocidad notablemente superior. Sin embargo cuentan con la desventaja que hacen más lenta la actualización, carga y eliminación de los registros ya que por cada modificación en la tabla se deberá modificar también el índice, además se debe tener en cuenta el hecho de que los índices también ocupan espacio en disco. Es por esto que no es factible indexar todos los campos de la base y se hace necesario seleccionarlos cuidadosamente. Cabe destacar que por defecto las tablas no contienen índices por lo que la introducción de estos puede llegar a producir mejoras de más del 100% en algunos casos.
- Los campos que se recomiendan indexar son:
  - ✓ Claves Primarias

- ✓ Claves Foráneas
- ✓ Campos por los cuales se realizaran búsquedas
- ✓ Campos por los cuales se va a ordenar

- Siempre conviene indexar tablas con gran cantidad de registros y que van a ser consultadas frecuentement.

3 Donde escribir las sentencias

- Siempre es preferible utilizar consultas almacenadas dentro del motor de base de datos y no dentro de la aplicación, una consulta almacenada en un procedimiento almacenado por ejemplo, se ejecuta mucho más rápido y directamente sobre el motor que cualquier consulta externa.
- Muchas de las aplicaciones sobre todo de reporting permiten unir y realizar los joins y consultas directamente en la herramienta produciendo una baja de performance realmente considerable. Lo correcto y más eficiente seria generar la consulta en un SP con todos los joins y guardar el resultado en una tabla temporal, de donde posteriormente el reporte solo deberá mostrar los datos sin necesidad de trabajarlos primero. Dependiendo de los joins que intervengan y los registros involucrados se pude mejorar drásticamente el rendimiento.

4 Acerca de la formulación de las consultas

A la hora de ejecutar una consulta SQL la forma en que esta es expresada afecta directamente al motor de BD, pequeños cambios pueden significar la ganancia de muchos segundos o minutos que el usuario debe esperar al momento de ejecutar la consulta. Algunas recomendaciones son:

- No utilizar **SELECT *** porque el motor debe leer primero la estructura de la tabla antes de ejecutar la sentencia.
- Seleccionar solo aquellos campos que se necesiten, cada campo extra genera tiempo extra.

```
SET STATISTICS TIME ON;
GO

SELECT *
FROM curso c
join CursoProgramado cp on c.IdCurso = cp.IdCurso
join profesor p on cp.IdProfesor = p.IdProfesor
join Matricula m on cp.IdCursoProg = m.IdCursoProg
join Alumno a on m.IdAlumno = a.IdAlumno
WHERE cp.IdCiclo = '2011-07' and cp.IdProfesor = 'P036'
```

```
order by 1,3;
GO

Tiempo de análisis y compilación de SQL Server:
   Tiempo de CPU = 0 ms, tiempo transcurrido = 0 ms.

(41 filas afectadas)

 Tiempos de ejecución de SQL Server:
   Tiempo de CPU = 0 ms, tiempo transcurrido = 57 ms.

SELECT
 c.IdCurso, c.NomCurso, cp.Horario, cp.Vacantes, cp.Matriculados,
 p.IdProfesor, p.NomProfesor, a.IdAlumno, a.NomAlumno, m.Promedio
FROM curso c
join CursoProgramado cp on c.IdCurso = cp.IdCurso
join profesor p on cp.IdProfesor = p.IdProfesor
join Matricula m on cp.IdCursoProg = m.IdCursoProg
join Alumno a on m.IdAlumno = a.IdAlumno
WHERE cp.IdCiclo = '2011-07' and cp.IdProfesor = 'P036'
order by 1,3;
GO

Tiempo de análisis y compilación de SQL Server:
   Tiempo de CPU = 0 ms, tiempo transcurrido = 0 ms.

(41 filas afectadas)

 Tiempos de ejecución de SQL Server:
   Tiempo de CPU = 0 ms, tiempo transcurrido = 1 ms.
```

- Utilizar **Inner Join**, **left join** , **right join**, para unir las tablas en lugar del **where**, esto permite que a medida que se declaran las tablas se vallan uniendo mientras que si utilizamos el **where** el motor genera primero el producto cartesiano de todos los registros de las tablas para luego filtrar las correctas, un trabajo definitivamente más lento.
- Especificar el alias de la tabla delante de cada campo definido en el **select**, esto le ahorra tiempo al motor de tener que buscar a que tabla pertenece el campo especificado.
- Evitar el uso de Cast. Y formulas dentro de las consultas, cada formula y casteo retrasan el motor considerablemente.
- El orden de ubicación las tablas en la cláusula from deberían ir en lo preferible de menor a mayor según el número de registros, de esta manera reducimos la cantidad de revisiones de registros que realiza el motor al unir las tablas a medida que se agregan.

5 Mas sobre las consultas

Tres puntos en particular a tener en cuenta y entender por qué mejoran las consultas:

- Combinaciones en where Vs. Combinaciones en el from
- Combinaciones externas (en el where y en el from)
- Query performance

5.1 Combinaciones en el where Vs. Combinaciones en el from

Debido al orden de procesamiento de una consulta multitabla el motor de BD se ve beneficiado si las combinaciones se resuelven en el FROM a través de INNER JOIN o alguna de sus variantes (LEFT OUTER JOIN, RIGHT OUTER JOIN). Esto se debe a que el motor al procesar una consulta SQL lo hace en este orden:

- Realiza el producto cartesiano de las tablas en el FROM
- Si existe cláusula WHERE aplica la condición de búsqueda
- Para las filas restantes calcula el valor de cada elemento en la lista del SELECT
- Si existe un SELECT DISTINCT elimina las filas duplicadas
- Si existe un ORDER BY ordena el resultado

Al ser el primer paso el producto cartesiano de las tablas que se encuentran listadas en el FROM, dependiendo del número de columnas de cada tabla y la cantidad de registros en cada una de ellas, se genera un resultado que puede llevarle demasiado tiempo al motor de BD, y luego tiene que aplicar la condición existente en la cláusula WHERE para reducir el número de registros resultante del producto cartesiano. Si realizamos la combinación directamente en el FROM evitamos utilizar la cláusula WHERE para dicho propósito haciendo mucho más eficiente la consulta, ya que al momento de realizar el producto cartesiano estamos diciéndole al motor qué registros queremos de ese producto. La sintaxis sería la siguiente:

```
SELECT
  c.IdCurso, c.NomCurso, cp.Horario, cp.Vacantes, cp.Matriculados,
  p.IdProfesor, p.NomProfesor, a.IdAlumno, a.NomAlumno, m.Promedio
FROM curso c
join CursoProgramado cp on c.IdCurso = cp.IdCurso
join profesor p on cp.IdProfesor = p.IdProfesor
join Matricula m on cp.IdCursoProg = m.IdCursoProg
join Alumno a on m.IdAlumno = a.IdAlumno
```

Como vemos no se necesita la cláusula WHERE para realizar las combinaciones de las tablas, si sería necesaria la cláusula WHERE si se necesita filtrar los resultados de dicha consulta:

```
SELECT
  c.IdCurso, c.NomCurso, cp.Horario, cp.Vacantes, cp.Matriculados,
  p.IdProfesor, p.NomProfesor, a.IdAlumno, a.NomAlumno, m.Promedio
FROM curso c
join CursoProgramado cp on c.IdCurso = cp.IdCurso
join profesor p on cp.IdProfesor = p.IdProfesor
join Matricula m on cp.IdCursoProg = m.IdCursoProg
join Alumno a on m.IdAlumno = a.IdAlumno
WHERE cp.IdCiclo = '2011-07'
and cp.IdProfesor in ('P016','P013','P005','P010')
order by 1,3;
GO
```

5.2 Combinaciones externas

Al realizar combinaciones internas o INNER JOIN entre dos tablas, los registros que no se combinen en ambas tablas son eliminados de la lista de resultados, si se necesita incluir las filas que no se combinan de una o ambas tablas se debe realizar combinaciones externas. Por ejemplo:

MARCA

	idmarca	nombre
1	1	TOYOTA
2	2	HONA
3	3	DAEWOO
4	4	KIA
5	5	HYUNDAI
6	6	VOLVO
7	7	NISSAN

AUTO

	idauto	modelo	anio	idmarca
1	1	RACER ETI	1995	3
2	2	BMW i8 Concept Spyder	2010	NULL
3	3	Concept Style Coupe	2011	NULL
4	4	Yaris	2011	1
5	5	Corolla	2005	1
6	6	Civic	2006	2

Existen registros en la tabla **MARCA** que no se encuentran en la tabla **AUTO**, si se realiza una combinación interna esas marcas no formarían parte del conjunto de resultados.

```
select *
from marca m
join auto a on m.idmarca = a.idmarca;
```

	idmarca	nombre	idauto	modelo	anio	idmarca
1	3	DAEWOO	1	RACER ETI	1995	3
2	1	TOYOTA	4	Yaris	2011	1
3	1	TOYOTA	5	Corolla	2005	1
4	2	HONA	6	Civic	2006	2

Al realizar una combinación externa se puede recuperar todo el conjunto de resultados, las combinaciones externas dependen del orden en el que se listen las tablas:

```
select *
```

```
from marca m
left join auto a on m.idmarca = a.idmarca;
```

	idmarca	nombre	idauto	modelo	anio	idmarca
1	1	TOYOTA	4	Yaris	2011	1
2	1	TOYOTA	5	Corolla	2005	1
3	2	HONA	6	Civic	2006	2
4	3	DAEWOO	1	RACER ETI	1995	3
5	4	KIA	NULL	NULL	NULL	NULL
6	5	HYUNDAI	NULL	NULL	NULL	NULL
7	6	VOLVO	NULL	NULL	NULL	NULL
8	7	NISSAN	NULL	NULL	NULL	NULL

5.3 Query performance

Buenas prácticas para mejorar nuestras consultas SQL

- Realizar las combinaciones en el FROM
- Tener en cuenta el orden en el que listamos las tablas en la cláusula FROM
- Evitar el uso de funciones T-SQL en el WHERE
- Evitar el uso de SELECT *
- Evitar el SELECT DISTINCT y el ORDER BY
- Evitar las subconsultas correlacionadas

6 Comparaciones entre tablas padre e hija

Es bastante común realizar comparaciones entre tablas padre e hija, existen tres formas de realizar dichas comparaciones que a continuación presento con un ejemplo y que sirven para encontrar si hay registros padre que no tiene filas relacionadas en la tabla hija.

Supongamos que queremos encontrar los profesores que no han sido programados en el ciclo 2012-05.

- **Caso 1: Usando NOT EXISTS**

```
SET STATISTICS TIME ON;
GO

SELECT *
FROM profesor p
WHERE NOT EXISTS (
  SELECT 1 FROM cursoprogramado cp
  WHERE cp.IdCiclo = '2012-05'
  AND cp.IdProfesor = p.IdProfesor);
```

- **Caso 2: Usando LEFT JOIN**

```
SET STATISTICS TIME ON;
GO

WITH cp(idcursoprog,  idprofesor, idcurso)
AS (
  SELECT idcursoprog, idprofesor, idcurso
  FROM CursoProgramado WHERE IdCiclo = '2012-05'
)
SELECT *
FROM profesor p
LEFT JOIN cp ON p.IdProfesor = cp.IdProfesor
WHERE cp.idprofesor is null;
```

- **Caso 3: Usando NOT IN**

```
SET STATISTICS TIME ON;
GO

SELECT * FROM profesor
WHERE IdProfesor NOT IN (
  SELECT IdProfesor
  FROM CursoProgramado
  WHERE IdCiclo = '2012-05'
  AND IdProfesor is not null );
```

En cada caso las consultas anteriores devuelven el mismo resultado. Pero ¿cuál de ellas tiene mejor rendimiento?. Asumiendo que todo lo demás es igual, la versión que tiene mejor rendimiento es la primera y la última es la que peor rendimiento. En otras palabras la variación NOT EXISTS es la consulta más eficiente.

Esto es en general, quiere decir que el resultado puede variar por ejemplo por los índices.

7 Acelerando las consultas

Es frecuente escuchar a muchos usuarios que se quejan de consultas lentas, en apartados anteriores se ha visto dos aspectos principales, primero el diseño de la base de datos y luego la elaboración de la consulta.

Por lo tanto, podemos afirmar que la consulta está bien diseñada, pero necesitamos acelerar su ejecución todavía, todavía no tiene el rendimiento esperado.

Para acelerar las consultas en SQL Server y en cualquier motor de base de datos debemos usar apropiadamente los índices.

Las columnas que se aconseja indexar son:

- Las que son clave primaria y/o foránea.
- Aquellas que se usan frecuentemente en búsquedas de rangos de valores con BETWEEN.
- Aquellas que se usan frecuentemente en ordenaciones con ORDER BY.
- Aquellas que se usan frecuentemente para combinar tablas o JOIN
- Aquellas que se usan frecuentemente en agrupaciones con GROUP BY

Tampoco se trata de indexar todos las columnas, porque se estaría penalizando el rendimiento de la base de datos en las inserciones, actualizaciones y borrados por la cantidad de índices a mantener. De lo que se trata es de seleccionar las consultas que más se usan en nuestra aplicación todos los días y coger las más importantes y fijarse en la parte WHERE.

Es importante tener presente que no se debería indexar columnas con pocos valores como true, false, o 1, 2, 3 por ejemplo porque tienen poca selectividad y una alta densidad de valores habitualmente. También decir que solo puede haber un índice agrupado por tabla y hasta 249 índices no agrupados.

Otras consideraciones a tener en cuenta:

- Usar procedimientos almacenados.
- Usar la opción FILLFACTOR al crear el índice para evitar divisiones de página en las modificaciones si nuestra base de datos sufre muchas modificaciones.
- Revisar la fragmentación de los índices para desfragmentarlos si es necesario.