Base de datos

Mejorar el desempeño de las consultas

Mecanismos

- No son reglas, se trata de técnicas que permiten, en la mayoría de los casos, mejorar el rendimiento de las consultas.

DISTINCT

- Sólo debe usarse si sabemos que la query puede devolver duplicados.

TOP

- Reduce la salida de la consulta.
- Ej:
  - SELECT TOP 100 fname, lname FROM customers
    WHERE state = 'mo'

  - SELECT TOP 10 PERCENT fname, lname FROM customers
    WHERE state = 'mo'

ROWCOUNT

- Variable que acota la salida.
- SET ROWCOUNT anula la palabra clave TOP de la instrucción SELECT si el número de filas es inferior.
- SET ROWCOUNT se omite en las instrucciones INSERT, UPDATE y DELETE

Índices

- Un índice es una estructura de datos que permite acceder a diferentes filas de una misma tabla a través de un campo (o campos clave).

Índices

- Los índices se utilizan para mejorar el rendimiento de las operaciones sobre una tabla.

- En general mejoran el rendimiento las SELECT y empeoran (mínimamente) el rendimiento de los INSERT, UPDATE y los DELETE.

Clasificación

- Índice único: basado en una columna de tabla cuyos valores son únicos.
- Índice primario (caso particular de índice único): basado en una clave primaria de una tabla. También llamado índice físico.
- Índice secundario: basado en columnas con valores no únicos.

Cuando crearlos

- La columna es usada frecuentemente en la cláusula WHERE o en condiciones de unión o *join*.
- La columna contiene un amplio rango de valores.
- La columna contiene un gran número de valores nulos (ya que no se almacenan).
- Dos o más columnas son usadas juntas con frecuencia en la cláusula WHERE o en condiciones de unión o *join*.
- Si la tabla es grande y se espera que la mayoría de las consultas recuperen poca cantidad de filas.

Cuando no crearlos

- Cuando la tabla es pequeña.
- Las columnas no son frecuentemente usadas como una condición en las consultas.
- La tabla se actualiza con frecuencia. Si tiene una o más índices en una tabla, las sentencias DML que acceden a la tabla, toman relativamente más tiempo, debido al mantenimiento de los índices.

Operadores en cláusula WHERE

- Rendimiento en orden decreciente:
  1. =
  2. >, >=, <, <=
  3. LIKE
  4. <>

WHERE

- Consideraciones:
  - Un literal único en lugar de varios usado al lado de un operador
  - Un nombre de columna o de parámetro al lado de un operador
  - Una expresión multi-operando al lado de un operador
  - Un número único exacto al lado de un operador
  - Un número único no exacto al lado de un operador (date, time)
  - Datos de caracteres, Null

WHERE SARGABLE

- WHERE NON-SARGABLE(Search Argument):
  - Adicionalmente las expresiones que incluyen una función sobre una columna, expresiones que tienen la misma columna en ambos lados de un operador o comparaciones contra una columna.
- WHERE SARGABLE:
  - compara una columna con una constante, puede aprovecharse la funcionalidad de los INDICES.

WHERE SARGABLE

- Ej:
  - WHERE substring(firstname,1,1) = ‘m’
  - WHERE firstname LIKE ’m%’

WHERE SARGABLE

- La función actúa directamente sobre la columna y el INDICE no puede usarse:
  - SELECT member_number, first_name, last_name
    FROM members
    WHERE DATEDIFF(yy,datofbirth,GETDATE()) > 21
- La función ha sido separada de la columna y puede usar un INDICE:
  - SELECT member_number, first_name, last_name
    FROM members
    WHERE dateofbirth < DATEADD(yy,-21,GETDATE())

WHERE SARGABLE

- Las cláusulas WHERE que usen el operador NOT son todas NON-SARGABLE, pero pueden ser reescritas para eliminar dicho operador, por ejemplo:
  - WHERE NOT column_name > 5
  - WHERE column_name <= 5

EXISTS

- Rendimiento en orden decreciente:
  1. Usar EXISTS o NOT EXISTS
  2. Usar IN
  3. Realizar un LEFT OUTER JOIN y chequear por una condición NULL
  4. Evitar not in

SELECT COUNT (*)

- IF EXISTS es más rápido que SELECT COUNT (*) porque en el momento que dicha QUERY encuentra el registro finaliza inmediatamente, mientras que COUNT(*) debe contar todas las filas.

SELECT COUNT (*)

- Usando SELECT COUNT (*):
  - IF (SELECT COUNT(*) FROM table_name WHERE column_name = 'xxx')
- Usando IF EXISTS (mucho más rápido):
  - IF EXISTS (SELECT * FROM table_name WHERE column_name = 'xxx')

LIKE

- LIKE ‘%m’ tiene peor rendimiento que LIKE ‘m%’ por causa de los índices.

AND

- En las condiciones que se establezcan en la cláusula WHERE al menos una de ellas debería basarse en una columna lo más selectiva posible y que sea parte de un INDICE.

- Si al menos uno de los criterios de búsqueda en la cláusula WHERE no es altamente selectivo, consideraremos añadir INDICES para todas las columnas referenciadas en la cláusula WHERE.

AND concatenados

- Localizaremos la expresión menos probable de suceder y la pondremos en primer lugar de la expresión AND. De este modo si una expresión AND es falsa la cláusula finalizará inmediatamente ahorrando tiempo

- Si ambas partes de una expresión AND son iguales o tienen igual peso, y son falsas, pondremos la menos compleja primero. De este modo si es falsa la expresión se realizará menos trabajo para evaluar la expresión.

BETWEEN antes que IN

- SELECT customer_number, customer_name
  FROM customer
  WHERE customer_number in (1000, 1001, 1002, 1003, 1004)

- Es menos eficiente que la siguiente QUERY:
  - SELECT customer_number, customer_name
    FROM customer
    WHERE customer_number BETWEEN 1000 and 1004

OR

- Si hay sólo un INDICE pero ninguna otra columna está en la cláusula WHERE entonces la primera versión no usará ningún INDICE y se realizará un TABLE SCAN.

OR

- SELECT employeeID, firstname, lastname
  FROM names
  WHERE dept = 'prod' or city = 'Orlando' or division = 'food'

- Este ejemplo puede ser reescrito usando un UNION ALL en lugar de un OR, tal y como muestra el ejemplo:
  - SELECT employeeID, firstname, lastname FROM names
    WHERE dept = 'prod'
    UNION ALL
    SELECT employeeID, firstname, lastname FROM names
    WHERE city = 'Orlando'
    UNION ALL
    SELECT employeeID, firstname, lastname FROM names
    WHERE division = 'food'

ORDER BY

- Mantener el número de filas a ordenar al mínimo, haciendo esto mediante la devolución de aquellas filas que son absolutamente necesarias.
- Mantener el número de columnas a ordenar al mínimo. En otras palabras, no ordenar columnas no requeridas.
- Mantener el ancho (tamaño físico) de las columnas a ordenar al mínimo
- Ordenar columnas con tipos de datos numéricos en lugar de tipos de datos carácter

GRUOP BY

- Recomendaciones:
  - Mantener el número de filas a devolver por la QUERY tan pequeño como sea posible
  - Mantener el número de agrupaciones tan limitado como sea posible
  - No agrupar columnas redundantes
  - Si hay un JOIN en la misma SELECT que tiene un GROUP BY, intentar reescribir la QUERY usado una SUBQUERY en lugar de usar un JOIN. Si es posible hacer esto, el rendimiento será mejor. Si se tiene que usar un JOIN, intentaremos hacer el GROUP BY por columna desde la misma tabla que la columna o columnas sobre la cual la se usa la función.
  - Consideraremos el añadir un ORDER BY a la SELECT que ordena por la misma columna que el GROUP BY. Eso puede producir que el GROUP BY tenga mejor rendimiento.

Tablas derivadas

- En lugar de usar tablas temporales, usaremos tablas derivadas para mejorar el rendimiento de nuestra QUERY. Esto funciona de la siguiente forma:
  - SELECT num_Customer, dt_Date FROM (SELECT num_Customer, dt_Date, nom_Customer FROM Customers)

MAX y MIN

- Si existe un índice en la columna agregada.
- Para MIN, para la búsqueda para en la primera fila que califica.
- Para MAX, va directamente al final del índice para ubicar la última fila.
- La optimización no se aplica si:
  - La expresión dentro del MAX o MIN es distinta de una columna.
  - La columna dentro del MAX o MIN no es la primera columna del índice.
  - Existe otra función agregada en la consulta.
  - Existe una cláusula GROUP BY.

- Adicionalmente, la optimización del MAX no es aplicada si existe una cláusula WHERE. Entre más selectivo el WHERE, más lenta la consulta.

Joins y Tipos de Datos

- Cuando se hace un join entre dos tablas utilizando columnas de distintos tipos de datos, una de las columnas debe ser convertida al tipo de dato de la otra.
- La columna cuyo tipo es menor en jerarquía es la que es convertida.
- El optimizador de consultas no puede utilizar un índice sobre la columna convertida.