Unidad 3

Bases de datos relacionales – aspectos avanzados – parte 1

Bases de Datos Aplicada

v1.0 – Agosto 2023

Universidad Nacional de La Matanza

DIIT
Departamento de Ingeniería e Investigaciones Tecnológicas

Contenido

- Qué son los planes de ejecución, cómo entenderlos.
- Tipos de índices, cómo se crean y cómo los usa el DBMS.
- Qué son las estadísticas y cómo las usa el DBMS.
- Cómo optimizar consultas para mejorar el rendimiento.

Al finalizar deberías ser capaz de…

- Interpretar un plan de ejecución a un nivel básico.
- Optimizar consultas y código SQL.
- Crear y optimizar índices.
- Generar estadísticas.

El plan de ejecución

Describe el **conjunto de operaciones** que el motor de ejecución debe realizar **para obtener los datos** requeridos en una consulta.

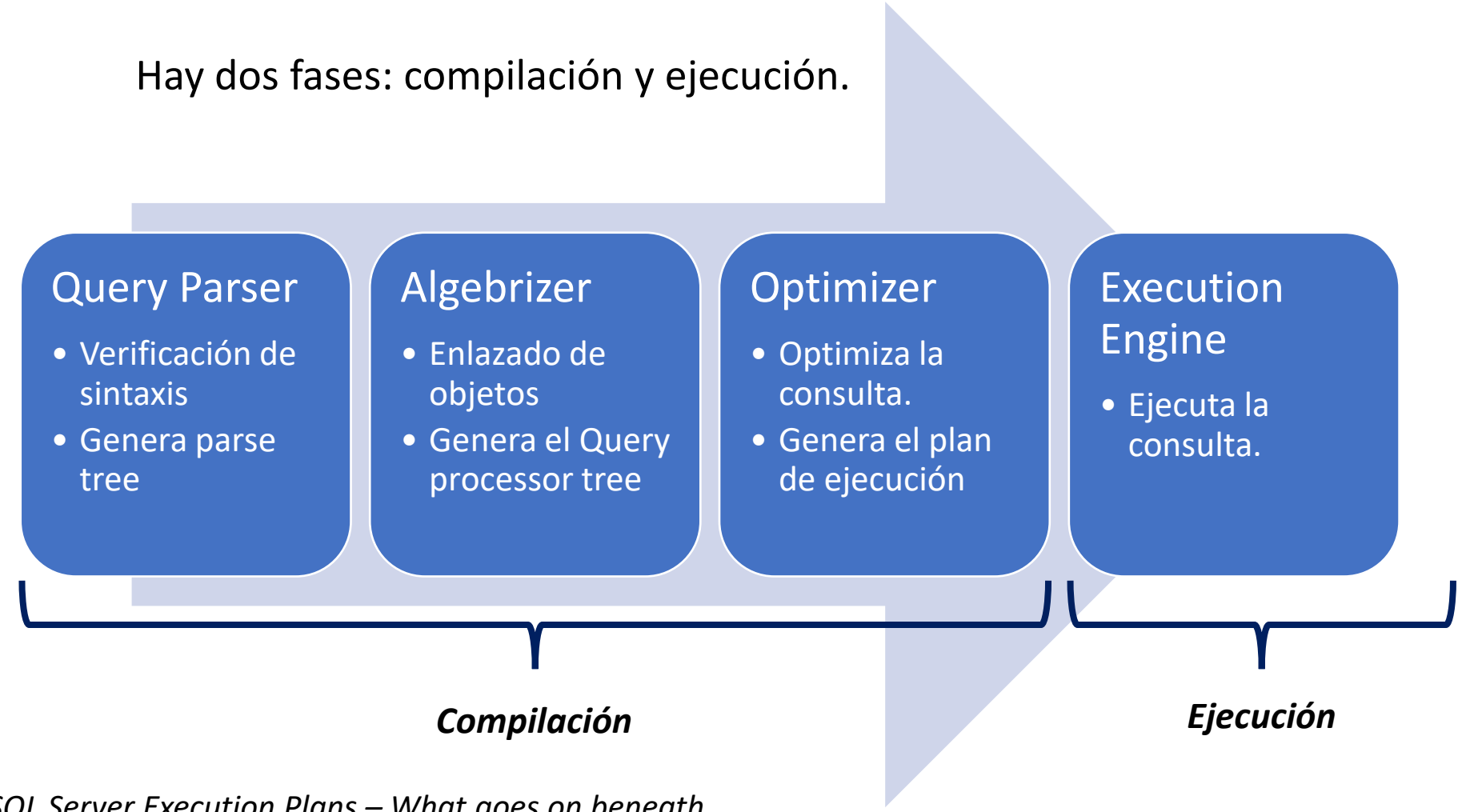
Detalla las tablas e índices **que** se accederán, **en qué orden**, de qué **manera** (algoritmo), en qué punto se calculan las funciones de agregado, ordenamiento, estimaciones tenidas en cuenta, reservas de memoria, etc.

Son un excelente punto de partida para la **resolución de problemas de performance.**

Fuente: SQL Server Execution Plans – What goes on beneath the Surface with your queries – 3rd edition – Grant Fritchey

Qué ocurre cuando se ejecuta una consulta

Hay dos fases: compilación y ejecución.



Fuente: SQL Server Execution Plans – What goes on beneath the Surface with your queries – 3rd edition – Grant Fritchey

Qué ocurre cuando se ejecuta una consulta

El plan de ejecución obtenido se almacena en una **caché** del motor.

La próxima vez que se ejecute la misma consulta, primero se verifica si existe el plan en caché, en ese caso se *reutiliza*.

Query parser: verifica sintaxis y produce un parse tree (estructura que representa los pasos lógicos necesarios para ejecutar la consulta).

Query binding: se resuelven los nombres de objeto, identifica los tipos de datos, determina la ubicación de funciones de agregado.

Los SP postergan (deferr) la resolución de nombres.

¿Qué errores se pueden generar en cada paso?

Fuente: SQL Server Execution Plans – …

Qué ocurre cuando se ejecuta una consulta

El **Algebrizer** produce un *query processor tree* (binario) que se pasa al **Query Optimizer**. Se incluye un **hash** del mismo.

El hash se utiliza para verificar si la consulta está en caché.

- De ser así se utiliza la versión compilada en caché.
    - Salvo que se haya modificado alguna estructura (p/e índice) o se hayan actualizado las estadísticas desde la creación del plan.
- Caso contrario el **Query Optimizer** analiza las distintas alternativas para resolver la consulta buscando la de menor costo.
    - Utiliza **heurística**: busca la mejor alternativa hasta un punto.

Fuente: SQL Server Execution Plans – …

Qué ocurre cuando se ejecuta una consulta

El **Optimizador de Consultas** (no el desarrollador) decide cómo debe ejecutarse una consulta.

Para obtener buena **performance**… debemos escribir SQL **eficiente**.

- Utilizar el enfoque de conjuntos para resolver problemas.
- La menor cantidad de sentencias posible.
- Con solo los datos necesarios.
- Evitar consultas *ad hoc* (parámetros concatenados) en favor de SP y consultas preparadas.

También podemos lograrlo manteniendo estadísticas exactas y actualizadas y promoviendo la reutilización de planes.

Fuente: SQL Server Execution Plans – …

Plan estimado y plan real (actual)

- Solo hay un plan de ejecución para una *query* en un momento dado.
- El plan estimado es el que genera el Optimizador o el que se encuentra en caché para la query enviada.
- El plan real incluye la información de tiempo de ejecución.
- La carga del servidor en el momento de la ejecución puede modificar el plan empleado.
- Si se crean y usan tablas temporales en la misma query el plan estimado no se podrá generar.

Fuente: SQL Server Execution Plans – …

Qué mirar en un plan de ejecución

Se lee de arriba a la derecha… hacia la izquierda y abajo.

Aprenda a identificar los Operadores.

¿Hay table scans?

¿Coinciden las estimaciones con lo real?

Observe que el plan estimado muestra los mismos costos que el real.

¿Dónde se producen los flujos de datos más grandes?

¿Se utiliza swap (page spills)?

¿Cuánta memoria se reserva para la consulta? (Primer op. a la izquierda)

¿Hay operadores en warning?

¿Se logra un nivel de optimización FULL?

Puede ser TRIVIAL, FULL, Timeout, Memory Limit Exceeded.

Fuente: SQL Server Execution Plans – …

Índices

Recurso/estructura para **acceder rápidamente** a ciertos datos.

Se aplican a tablas, incluso a un subconjunto de campos.

En algunos RDMBS / versiones se pueden aplicar a vistas.

Hay bases de datos que tienen un orden dado:

Diccionarios

Directorio telefónico (páginas blancas y amarillas) ¿iguales?

Estructuras de datos **redundantes**: generan copias de campos (excepto el índice clúster).

El RDBMS siempre utilizará el índice que le permita reducir al máximo el ámbito de la búsqueda.

Índices clúster

El índice CLÚSTER (o de agrupamiento) es el que determina en qué orden se guardan los registros en la tabla.

- Solo puede haber uno por tabla.
  - Si la tabla no tiene índice clúster se le denomina heap (montón).
- Puede ser compuesto por varios campos.
- Aunque no se índique la cláusula UNIQUE solo acepta valores únicos.
- Las búsquedas por rango se benefician especialmente de ellos.
- Se puede generar con la designación de un campo como primary key.
  - Pero para indicar clave compuesta o sentido de ordenamiento se hace en forma explícita.

Índices no clúster

Generan una **copia** de los campos indexados (y los incluidos) manteniendo el orden según el criterio de indexación.

En cada inserción/borrado/modificación se afectará también.

Se generan con la sentencia

```
CREATE NONCLUSTERED INDEX nombreIndice ON
esquema.tabla (campos criterio);
```

Si una consulta hace referencia a campos del índice puede que no se requiera acceder a la tabla para resolverla.

Verifíquelo haciendo un SELECT solo de los campos del índice (incluidos también) con un ORDER BY de los campos indexados.

Siempre incluirá una referencia al índice clúster (aunque no lo indiquemos).

¿Qué hace a un índice **eficaz**?

Índices no clúster

Selectividad

¿Qué atributo de los árboles indexaría?

¡Depende de la consulta que quiera hacer!

Florecimiento de árboles en el dosel del bosque lluvioso de Perú. (Foto de R. Butler)

Índices de cobertura

En los índices NO CLÚSTER podemos agregar **campos incluidos.**

- No se utilizan para indexar (no se ordena en base a ellos).
- Pueden mejorar mucho la performance si todos los campos requeridos están incluidos.
  - ¡No se necesitaría acceder a la tabla!
- Cuando un índice contiene todos los campos referenciados por una columna se dice que cubre la consulta (índice de cobertura).
- El orden de los campos incluidos (non-key) no afecta la performance.
- No siempre es la mejor solución (p/e si siempre lee todos los campos)
  - *¿Ve alguna relación con la !recomendación "SELECT *" ?*

Índices: fill factor

Fill factor: factor de relleno. Atributo de cada índice que se define al crearlo.

Cuando *se llena* una estructura de índice se genera una nueva y se enlaza a la anterior, eso se denomina "*Page Split*".

"Page Split" implica:

- Agregar una nueva página.
- Mover la mitad de los datos a la nueva página.
- Marcar como inválidos los datos movidos en la página anterior.
- Actualizar las referencias de páginas existentes para que apunten a la nueva.

Dejar un espacio libre en cada página (hoja) del índice reduce los page splits.

- Si se insertan entradas adicionales (o un campo ocupa más lugar en un update) se usa ese espacio disponible.

Índices: fill factor

Reducir los *page split* mejora la performance.

¡Pero **mantener las páginas incompletas la empeora**!

- Tendrá bases de datos más grandes
- Mayor uso de memoria (menos datos entrarán en ella)
- Las consultas demorarán más (más *spills*).
- Backups y verificaciones DBCC demorarán más.
- Regeneración de índices demorará más.

¿Cuál es el mejor valor?

No aplique el mismo default a todos los índices.

- Si la clave del índice es siempre creciente: 100% (default) es bueno
- Si la clave no es siempre creciente: evalúe la fragmentación de los índices (más fragmentado es peor) y reduzca el *fill factor* gradualmente.

Optimización de índices

El desarrollador de DB genera índices de acuerdo a las **consultas** que realizará.

- Mantener **actualizado** cada índice supone un costo de I/O.
  - ¿Qué pasa si el campo clave de varios índices no clúster se actualiza frecuentemente?
- ¿Se utilizan todos los índices?
- ¿Se han generado de forma óptima?
- ¿Se necesitan índices adicionales? ¿Valen la pena?
  - Determine no solo las consultas lentas sino también su frecuencia.

Optimización de índices

- ¿Hay índices CLÚSTER sobre campos NO ÚNICOS?
  - Para salvarlos el DBMS agrega un *uniquefier* de 4 bytes.
- ¿Se podrían agregar campos INCLUDE para mejorar performance?
- ¿Hay campos INCLUDE que rara vez se acceden?
- ¿Se podrían mover al INCLUDE campos indexados que no se usan en las búsquedas?
- ¿Se podría generar un índice no clúster con la cláusula UNIQUE (en campos no clave)?
- ¿Es apropiado el FILL FACTOR?

Optimización de índices

- Orden lógico y físico dejan de estar alineados con *updates, deletes*...
- En algunos DBMS **regenerar** los índices mejora su desempeño.

Determine cuánto tiempo puede estar offline una tabla.

Rebuild vs **reorganize**

Diferentes operaciones que reducen la fragmentación de los índices.

Rebuild: Impacto alto (bloquea la tabla). Genera de nuevo el índice, puede modificar el FILL FACTOR. Aconsejada para fragmentación alta (>30%).

Reorganize: Impacto bajo. Aconsejada para fragmentación baja (<30%).

- Si hay una ventana considerable use rebuild (cuidado con *mirroring*).
- No tiene sentido ejecutar ambas.
- Siempre mantenga un plan de mantenimiento.

Estadísticas

Información generada y mantenida automáticamente(*) por el DMBS acerca de la **distribución de los valores** de las tablas (histogramas). *"Data about the data".*

El optimizador de consultas las utiliza en la generación de planes de ejecución.

Obtiene de ellas la **cardinalidad**: una estimación de la cantidad de filas.

Se crean **automáticamente** en base a campos indexados.

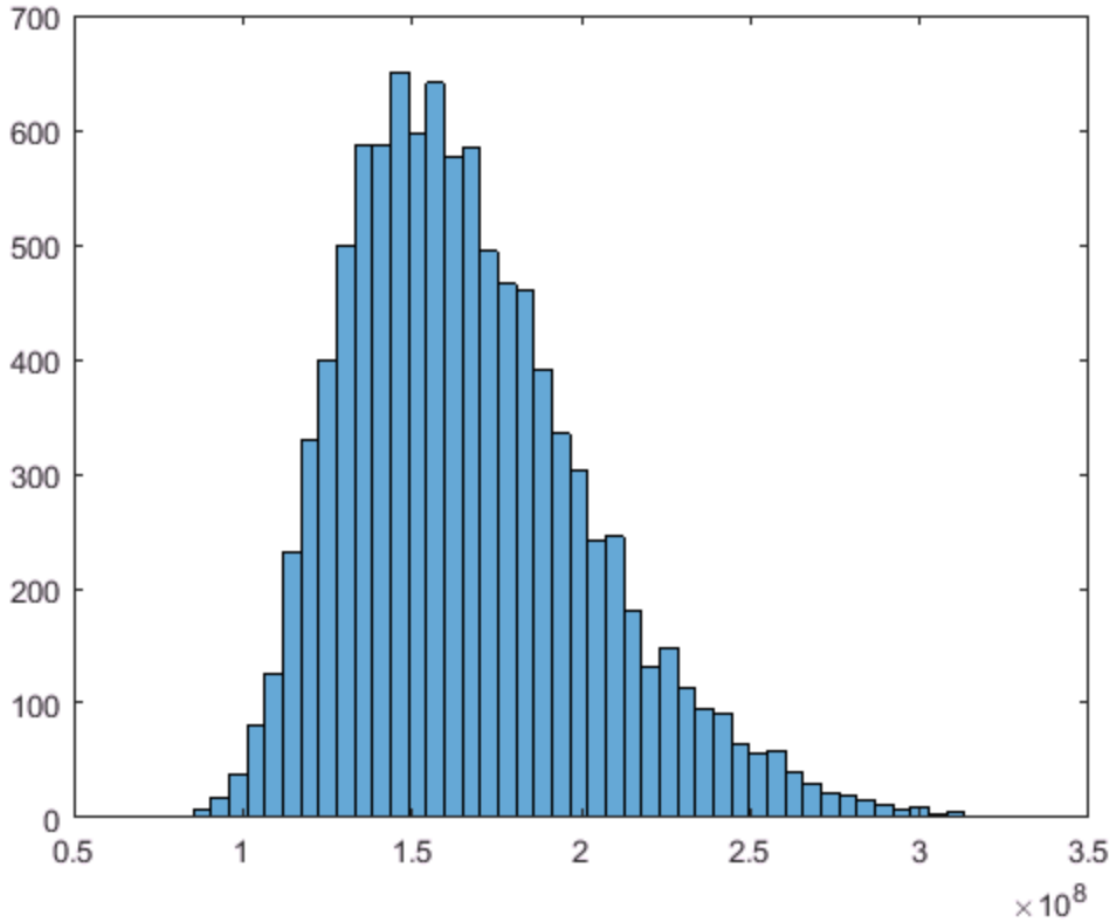
- Por defecto no se muestrean todas las filas sino un subconjunto.
- Se puede forzar el muestreo de todas las filas de la tabla.

Constan de:

- **Encabezado**: información general de un conjunto de estadísticas dado.
- **Grafo de densidad**: Selectividad, singularidad (*uniqueness*) de los datos.
- **Histograma**: Cuenta tabulada de las ocurrencias de valores particulares de hasta 200 puntos elegidos por representar toda la tabla.

Fuente: https://learn.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-ver16

Estadísticas



Histograma: Representación gráfica de una distribución de frecuencias por medio de barras, cuyas alturas representan las correspondientes frecuencias. (RAE)

L-W-Doyle, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

Estadísticas

Puede mejorarse la performance creando estadísticas **manualmente** si…

- Lo sugiere el *Database Engine Tuning Advisor*.
- El predicado de la consulta (ver plan) contiene múltiples columnas correlacionadas que no están en el mismo índice.
- La consulta aplica a un subconjunto de datos (estadísticas filtradas).

Existe un SP para actualizar todas las estadísticas de una DB:

```
EXEC sp_updatestats;
```

También podemos actualizar para una tabla, por ejemplo con muestreo de toda la tabla:

```
UPDATE STATISTICS ddbba.Venta
    WITH FULLSCAN
```

¡Cuidado! Esta sentencia consume muchos recursos.

Fuente: https://learn.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-ver16

Estadísticas

Actualización de estadísticas **sincrónica** (default)

- Recomendada para operaciones que cambian la distribución de los datos.
- Truncado, inserción en lote (bulk), actualización de muchas filas (bulk).
- Garantiza uso de estadísticas actualizadas.
- Si no están actualizadas primero se actualizan y luego se compila y ejecuta la query.

Actualización de estadísticas **asincrónica**

- El *query optimizar* utiliza las estadísticas incluso si no están actualizadas.
- Se ejecutan frecuentemente las mismas consultas (o similares).
- El tiempo de respuesta puede ser más predecible.
- Se producen *timeouts* por demora en compilación por demora en actualización de estadísticas.

Fuente: https://learn.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-ver16

Optimización de consultas

Cuanto más compleja, más esfuerzo para optimizarlas.

- ¿Hay relaciones o tablas de sobra? Observe las subconsultas. Minimice los SELECT.
- ¿Se usan los WHERE y JOIN aprovechando los índices?
- ¿Podemos generar CTEs para facilitar el mantenimiento?

Verifique el plan de ejecución.

- ¿Hay algun table scan? Evalúe creación de índices.
- ¿Alguna tabla es un heap? Cree índice clúster.
- ¿Hay mucha diferencia entre filas estimadas y reales? Estadísticas.
- Reduzca o elimine el uso de tablas temporales.
- Las tablas variables pueden tener **un** índice clúster (aprovéchelo).
- ¡Cuidado con el NOLOCK! Obtendrán lecturas sucias.
- Mida las ejecuciones por SP para optimizar selectivamente.

Optimización de consultas

- Verifique los tipos de datos empleados (abuso de float).
  - Realice el testing correspondiente antes de modificar en productivo.
  - No todos los lenguajes tienen tipos de datos 100% equivalentes.
- ¿Realmente necesita UNICODE?
  - Sino, puede usar char/varchar en lugar de nchar/nvarchar.
    - Reducirá espacio ocupado (disco, memoria, backups) a la mitad.
- ¿Realmente necesita VARchar/nVARchar?
  - Si el campo no es realmente tan variable use CHAR/NCHAR.
    - Mejora la performance (no necesita calcular largo del campo).
    - ¡Cuidado! Rellena con espacios al final del texto.
- Examine la calidad del plan de ejecución: ¿requiere partir la consulta en otras más simples?

Optimización de consultas

- Las tablas variables NO tienen estadísticas.

  Siempre se asume una fila (dependiendo la versión).

  Las tablas temporales SÍ.

  Realizar JOINs sobre tablas temporales (especialmente si son grandes) tendrá mejor performance.

- Cursores.

  Elimínelos. Tecnología caduca.

  Si no se pueden eliminar, optimícelos (solo de lectura, forward, ciérrelos).

- Utilice SPs o consultas parametrizadas en lugar de literales hardcodeados.
  - Facilita la reutilización de planes en caché.

Fuente: https://learn.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-ver16

Mi consulta tarda mucho…

Sospechosos de siempre: consulta pobremente generada, red, recursos del servidor, coincidencia con tareas de backup.

Habiendo descartado lo demás:

- Actualizar estadísticas
- Actualizar estadísticas con FULLSAMPLE.
- Regenerar índices.
- Crear índices o modificar índices existentes.
- ¿Se necesita una estadística filtrada? Campos con distribución MUY desigual son candidatos.
- Si en SSMS o entorno de desarrollo (VS) funciona rápido y en el cliente no, comparar **configuración de entorno**.
- ¿Demasiado normalizado?
- ¿Puedo mejorar el HW?

https://www.sqlservercentral.com/articles/sql-server-stored-procedures-and-set-options

Tablas temporales

1. ¡No abuse! No siempre son el mejor recurso.
2. La base tempdb debe estar correctamente *tuneada* (ubicación, crecimiento).
3. Solo inserte las columnas y filas necesarios.
4. Compare la performance de SELECT INTO e INSERT INTO y use la mejor.
5. Elimínelas tan pronto deje de usarlas. No deje esto al motor.
6. *tempdb* es un recurso compartido. *Compartir es de nene bueno.*
7. Genere índices **si está seguro** de que se requerirán.
8. Las tablas variables sufren menos por bloqueos, son seguras y los rollback no las afectan. Ideales para pocos datos.

Rowstore vs Columnstore

- Las tablas siempre se almacenaron por fila (*rowstore*)
- Pero al realizar operaciones de agregado normalmente nos enfocamos en determinadas **columnas**.
- Generar una tabla almacenada por columna (*columnstore*) puede mejorar muchísimo las consultas, especialmente las de agregado/ventana.
- Los índices para almacenado por columna son ideales en tablas grandes del datawarehouse.
  - Se comportan mejor si son solo de consulta.

¿Dudas?

DIIT

Departamento de Ingeniería e Investigaciones Tecnológicas

Universidad Nacional de La Matanza

Bibliografia

TCL
Silberschatz, A., Korth, F., Sudarshan, S.: Fundamentos de Base de Datos, cuarta edicion capitulo 15-16-17
SET TRANSACTION ISOLATION LEVEL (Transact-SQL)
https://learn.microsoft.com/es-es/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-ver16
Transacciones Distribuidas
https://learn.microsoft.com/es-es/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-ver16