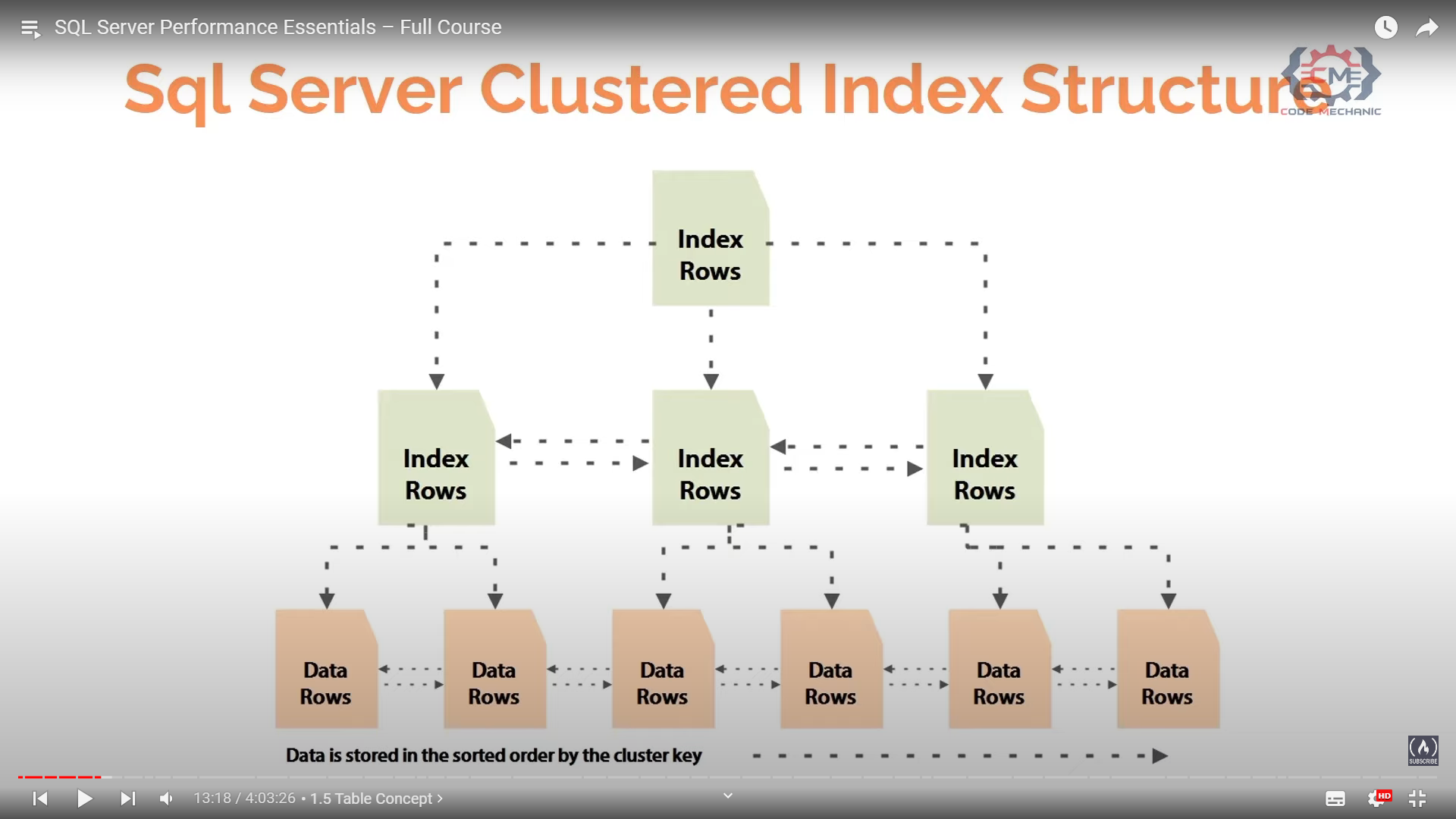
SQL Server Performance

La performance es importante en el desarrollo de proyectos que involucran una base de datos, ya que la misma tiene impacto en la:

* escalabilidad: La performance impacta en la experiencia del usuario. Si el sistema es lento para un entorno mono-usuario, no es viable escalarlo a un entorno multi-usuario, limitando la aplicabilidad.
* reducción de costos
* velocidad de ejecución

Estructura de tablas e índices



En el caso de índices Clustered, vimos que los nodos hoja directamente constituyen las páginas en memoria donde está almacenada la tabla, ya que el índice responde al propio ordenamiento de la tabla dado el atributo utilizado.

De esta forma a nivel implementación, un índice Clustered conlleva un archivo Sorted. Y, por lo tanto, esto facilita las operaciones de búsqueda en base al atributo de ordenación.

En el caso de indices unclustered, estos están hechos sobre atributos para los cuales los datos originales no estan ordenados físicamente. Por lo tanto, en las hojas del índice tendremos los valores del atributo en base al cual fue construido el índice y los punteros a memoria del dato con ese valor.

Execution Plan o Plan de ejecución

Dada una query a realizar a la DB, a esta se le va a asignar una plan de ejecución que refiere a la secuencia de operaciones que realizará el DBMS para resolver dicha consulta.

El DBMS nos va a brindar información sobre el costo de cada operación, con lo cual tendremos una noción relativa de cuales operaciones son las más demandantes y qué algoritmos está utilizando para implementar dichas operaciones.

Veamos un ejemplo con la consulta sobre los cursos que realizó el estudiante 29717.

El flujo de la consulta se da en forma “serial” de derecha a izquierda. Donde a la derecha veremos las tablas que son accedidas para resolver la consulta. Este proceso ocurre realmente con cierto grado de paralelización[[1]](#footnote-0), que es determinado por el DBMS. Sin embargo, a nivel análisis de optimización podemos hacerlo como serial.

Luego, debemos hacer foco en las operaciones que representen un porcentaje mayor en nuestro costo de ejecución. Determinar qué parte de la consulta se esta implementando y cómo se esta haciendo.

Para ello usamos el “predicado” de esta operación y luego analizamos las métricas como:

Número de filas a leer

Costo de i/o

Costo de CPU

Costo estimado de operación

Para comparar entre distintos planes de ejecución, debemos tener presente el Costo estimado del Arbol que se informa al final del plan de ejecución. Ya que si bien es un valor relativo, siempre se relativiza respecto a la misma operación o referencia.

\*Explicar paralelización\*

\*Explicar inner loop join\* : como vemos indica que recorre todas las filas de la relación superior y obtener los valores correspondientes en la tabla inferior. Que, en este caso, esa busqueda se hace mediante un índice clustered, ya que la tabla Co está ordenada en función el atributo de junta.

Y podemos pararnos sobre las flechas de entrada a cada operación para determinar el número estimado de filas que ingresan a la misma.

NOTA: Observen que el número de filas que se mueven entre operaciones no necesariamente corresponde a la salida. Ya que este plan de ejecución se ESTIMA en función de estadísitcas que el DBMS almacena de la estructura de las tablas.

Lo cual nos lleva a la siguiente sección: Qué son las estadísticas?

Estadísticas

Las estadísitcas son la información de la que se vale el DBMS para estimar los costos de las query. Para poder observar estas estadísticas usamos:

Ahora vamos a generar nuestras estadísticas en relación a IO y TIME/CPU.

Las estadísticas pueden actualizarse manualmente o se actualizan automáticamente dado un período de tiempo o trigger-based conditions. Claramente, las estadísticas comenzarán a diferir cada vez más de los datos reales en memoria cuantas más actulizaciones, deletes, inserts, etc se hagan desde la última actualización de las estadísticas.

Podemos utilizar la pestaña de “Messages” una vez ejecutada la consulta para detectar cuantas lecturas se han realizado REALMENTE al momento de ejecutar la consulta para poder compararlo con lo indicado por el plan estimado (LOGICAL READS → lecturas de páginas en disco)

CPU time → tiempo de CPU que tomaría ejecutar la consulta si se realizara 100% serial (1 único core)

Elapse time → tiempo que realmente requiere ejecutar la consulta. Si el Elapse time no coincide con el CPU time, nos indica que se ha aplicado algun grado de paralelización en la ejecución.

Nuevamente, el número de Logical reads asociadas a una operación nos habla del costo de la misma. Es decir, que también nos señala donde debemos hacer foco a fin de optimizar una dada consulta.

OPTIMIZACION DE CONSULTA

Claramente, el DBMS tiene la inteligencia suficiente para hacer este analisis que nosotros estamos haciendo, ya que todas nuestras conclusiones las estamos haciendo en base s información que la misma DB nos brinda. Por ende, si volvemos a pedirle un plan de ejecución para la consulta, veremos que nos indica que tenemos un indica faltante que tiene un gran impacto sobre la performance de nuestra consulta. Incluso, podemos decirle que nos brinde más información sobre este indice que nos está faltando.

Si nosotros consideramos que este índice puede ser de utilidad dado que esta misma estructura de consulta puede darse frecuentemente, entonces podemos crearlo.

Una vez creado el índice, volvemos a pedir un plan de ejecución para nuestra consulta y observamos las diferencias.

No solo cambia el arbol de ejecución, sino que veamos cómo se reduce el Costo estimado del árbol y el numero de Logical reads y el CPU / Elapsed time

Ejemplo 2:

El problema con este plan de ejecución es que se está armando todas las combinaciones de cursos con TermCode SP2016 antes de analizar si es NULL.

Si armamos un índice, como nos sugiere el DBMS, veremos que cambiamos el plan y reducimos de 24 a 8 el costo de ejecución. Sin embargo, el problema no es ese sino que la consulta está planteada en forma ineficiente.

La solución es cambiar nuestra consulta a una equivalente pero más apropiada.

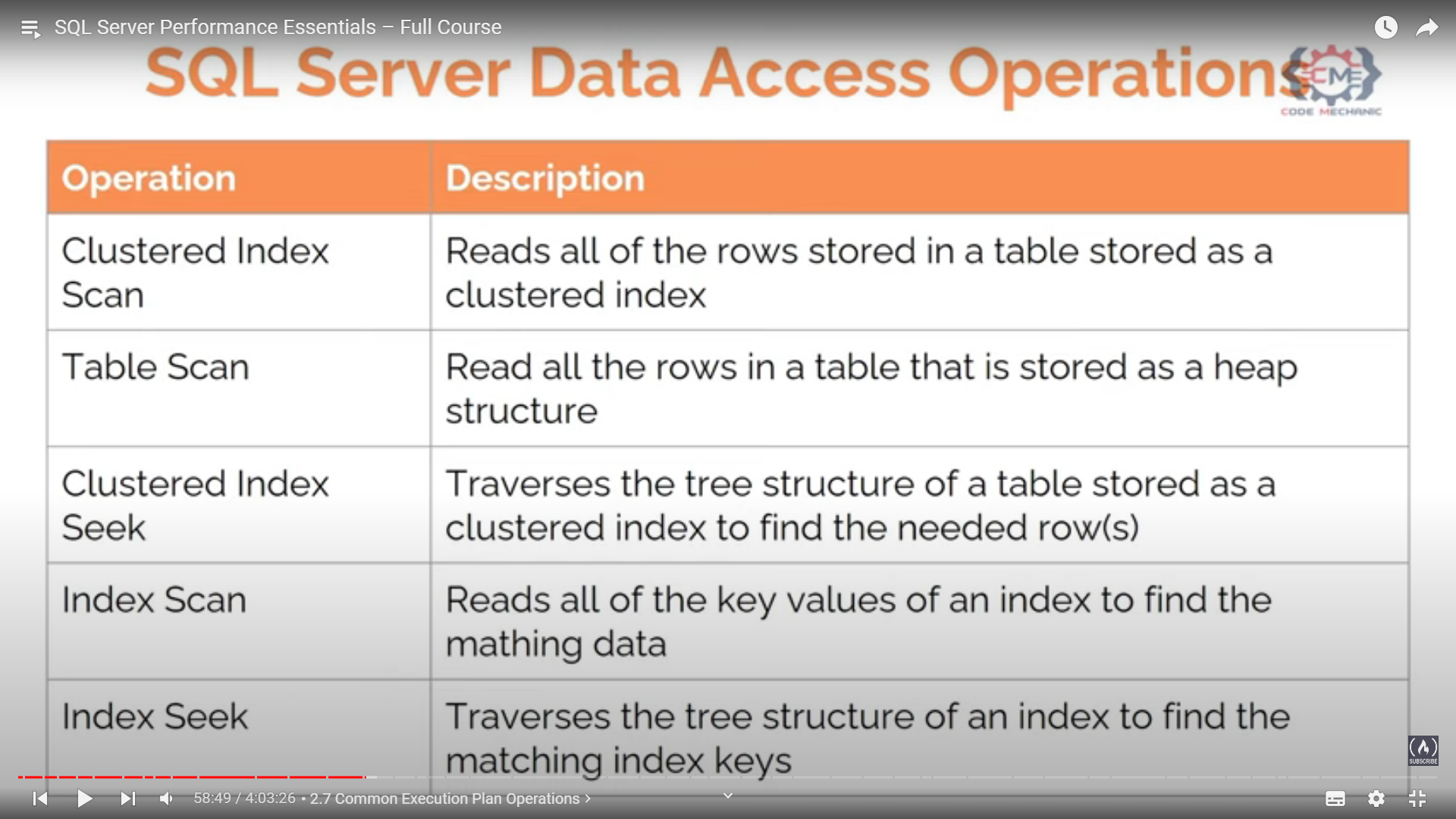
Incluso podemos cambiar esa ejecución volviendo a crear el indice sugerido

\*Mostrar el HASH search\*

\*Mostrar el TOP\* que significa que solo necesito tomar el valor superior. Es decir, solo necesito devolver si hay 0 o 1 o más en cada Curso. Con eso me alcanza para resolver el NOT EXIST

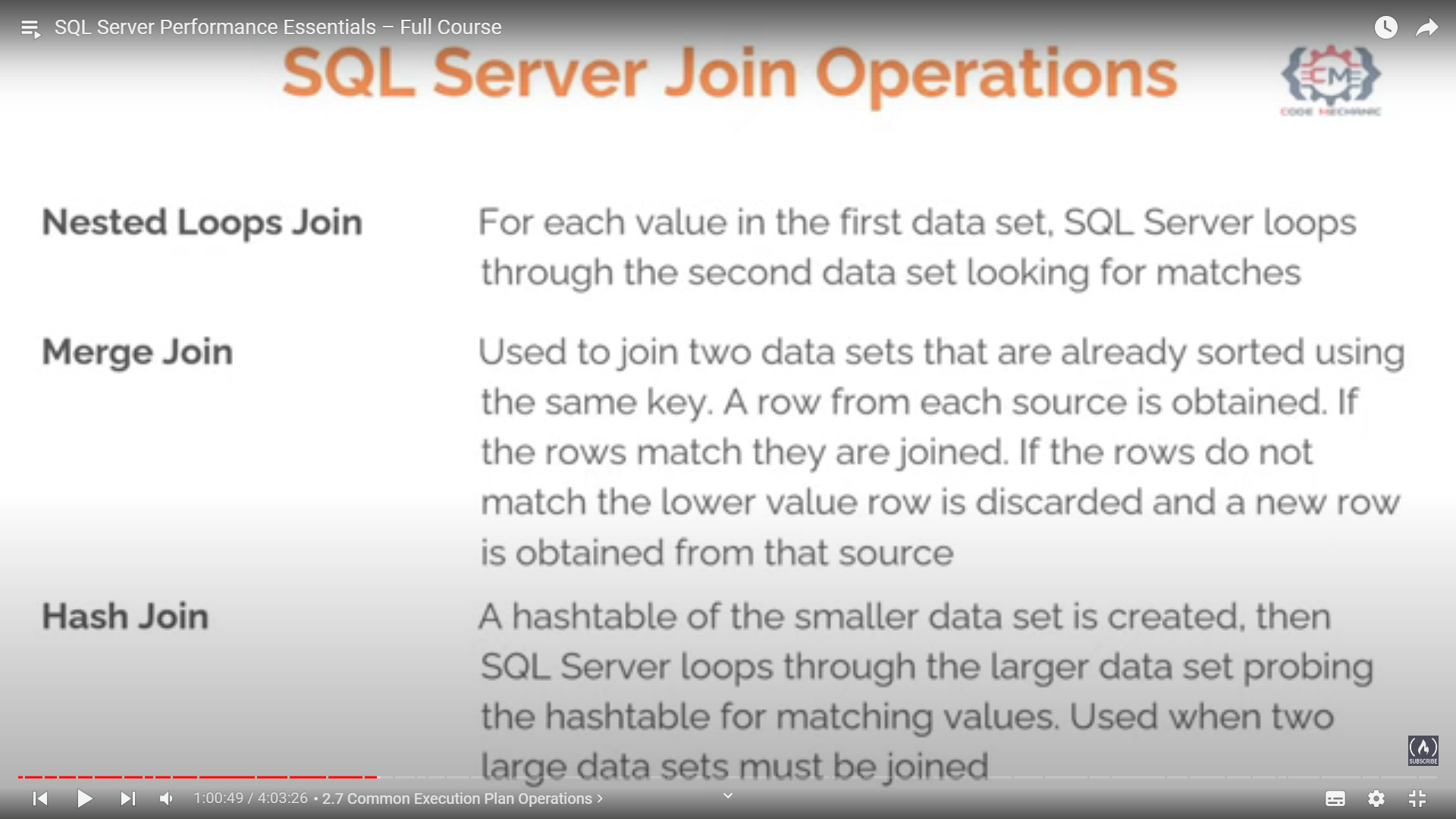
\*Mostrar el índice que sugiere generar SQL\* Básicamente nos está sugiriendo replicar toda la información pero ordenada de cierta forma. Lo cual, no es muy óptimo a nivel almacenamiento.

RESUMEN de OPERACIONES COMUNES:



scan = malo

seek = bueno



Hash nos dice que no hay indices utiles para la junta. En general tambien conviene restringir el nro de filas sobre la relación que se usa para generar la tabla. A veces suele indicar que se obvió alguna FK en el diseño.

PARTE 2:

CREANDO INDICES

Índices clustered: Están asociados a la estructura con la que se almacena la información en memoria. Es único

Índices non-clustered: aquellos que se construyen sobre un atributo o conjunto de atributos (generalmente, no los del clustered) y en los hojas almacenan punteros a memoria donde encontrar las entradas que cumplen con la condición.

Operaciones:

Scan: Se lee todas las hojas del indice en busca de las entradas que cumplan la condición establecida.

Seek: Bajamos por el indice para encontrar los nodos ojos que cumplen pero utilizando las condiciones de la estructura del indice, no escaneando

Cómo elegimos sobre qué atributos crear un índice? Básicamente, analizando la aplicación y determinando aquellas operaciones que se repiten y requieren hacerse de forma óptima.

En base a esto se definen dos tipos de columnas:

* Atributos que aparecen en nuestras clauslas Where: Las clausulas where determinan cómo vamos a encontrar las entradas a nuestra tabla que cumplen las condiciones requeridas
* FK columns: El otro caso de atributos para los cuales vamos a querer generar ínidices son aquellos atributos que son clave foranea de otra tabla. Ya que los vamos a utilizar en las operaciones de junta. Y también nos importa para obtener información en forma transversal a tablas.  
  Piensen en un estudiante que se logea con su ID. Seguramente aparezca en una tabla Cursada. Aunque la tabla sea sobre [Estudiante,MAteria] Indirectamente, tendremos de esa tabla todas las filas que son de ese estudiante y sus notas

El orden de las columnas o atributos importa? Si. Ya que define cómo estarán ordenadas las entradas del índice. En base a esto, el DBMS determina si un índice puede usarse para una consulta y cómo puede utilizarlo

SELECTIVIDAD DE LOS INDICES:

LA selectividad de los índices es el 2do factor determinante en el úso de los mismos. Básicamente, cuantas menos filas sean apuntadas por cada hoja del indice, mejor. Ya que eso reduce considerablemente los accesos a disco para cada condición.

Si la selectividad es demasiado baja, entonces puede que sea conveniente NO usar el indice, ya que me conviene directamente escanear la tabla que accederla multiples veces.

LIKE operator: Tener en cuenta que para poder utilizar los indices a veces necesitamos un mínimo de selectividad en nuestra consulta.

CONSULTAS CON FUNCIONES

PArte 3: Dynamic management Views

https://github.com/iCodeMechanic/Essentials-of-Sql-Server-Performance-for-Every-Developer/blob/master/Exercise%20Files/Module%201/Part%204/Links.txt

DMVs son vistas que brindan informaciòn sobre la actividad en la DB. No todas las DVMs estan accesibles, se requiere un nivel de “Server state” porque no solo tienen información sobre las estadisticas de uso, sino tambien información sobre estructura el sistema.

Query1: utiliza dm\_exec\_sessions para informanos:

* sobre los clientes que están conectados a nuestra DB
* el estado: activos, inactivos, etc
* Nos permite saber si alguna de las sesiones está consumiendo demasiados recursos

https://raw.githubusercontent.com/iCodeMechanic/Essentials-of-Sql-Server-Performance-for-Every-Developer/master/Exercise%20Files/Module%204/Part%202/DMV.txt

Query2: Averiguar qué operaciones se están ejecutando en un momento dado. Esto nos permite identificar qué sentencia puede estar haciendo al sistema lento en un momento dado.

Esta consulta hace:

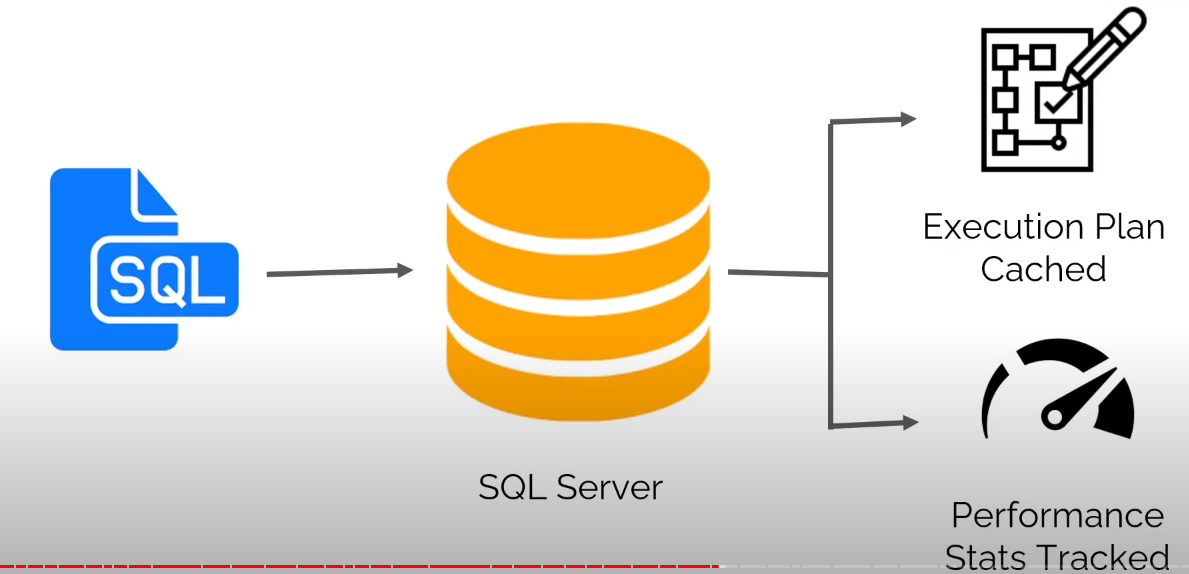
* TRae las sesiones y las request de cada una
* Luego, nos trae el plan de ejecuciòn para esta request
* Y nos indica si está siendo blockeada por otra sentencia en ejecución

Ynos muestra:

* información sobre el estado de la sentencia
* Si pertenece a un stored procedure
* informaciòn sobre el cliente de la consulta
* información sobre el tiempo y actividad en la sentencia
* información sobre la sentencia que la está blockeando si existiera

https://raw.githubusercontent.com/iCodeMechanic/Essentials-of-Sql-Server-Performance-for-Every-Developer/master/Exercise%20Files/Module%204/Part%203/DMV.txt

Query 3: encontrar sentencias eficientes y sentencias poco eficientes



* traemos las estadísticas de las queries y las juntamos con sus textos y planes de ejecución asociados
* Traemos la info sobre:
  + la sentencia específica
  + sus estadísticas de consumo de recursos
* Ordenamos la información según estemos buscando sentencias eficientes o que generen bottlenecks

https://raw.githubusercontent.com/iCodeMechanic/Essentials-of-Sql-Server-Performance-for-Every-Developer/master/Exercise%20Files/Module%204/Part%204/DMV.txt

Query 5: Obteniendo sugerencias de índices de las estadísticas de SQL.

Vimos que al ejecutar consultas, si SQL server consideraba que existían índices que podrían optimizar la consulta, nos lo sugería.

Esto no solo ocurre al realizar una única consulta, sino que podemos obtener sugerencias de SQL desde sus estadísiticas.

Literalmente, del funcionamiento de la DB nos genera un listado de “index faltantes”

Esta vista nos marca:

* Atributos que vamos a evaluar por igualdad y por rango
* columnas a incluir en las hojas
* las veces que podríamos haberlo usado para generar index seek and index scans
* El estimado % de optimización que se percibe al generar el índice

Pero recuerden que no es optimo generar todos lo índices, ya que es tendencia de SQL buscar un índice para optimizar toda Query ejecutada.

Y si creamos todos estos índices sin pararnos a analizarlos, seguramente llegaríamos a un estado de sobre-indexación de la DB.

https://raw.githubusercontent.com/iCodeMechanic/Essentials-of-Sql-Server-Performance-for-Every-Developer/master/Exercise%20Files/Module%204/Part%205/DMV.txt

Query 6: Identificar índices que no se usan

Vease, índices que se actualizan mucho más de las veces que se utilizan.

En general esto puede ocurrir porque las columnas/atributos escogidos no son de utilidad para sentencias o porque no son suficientemente selectivos como para usarlos en lugar de un scan.

<https://raw.githubusercontent.com/iCodeMechanic/Essentials-of-Sql-Server-Performance-for-Every-Developer/master/Exercise%20Files/Module%204/Part%206/DMV.txt>

Parte 4: Buenas prácticas

Parameterized SQL: cuando mandamos sentencias a ejecutar en SQL desde una app en C#, C++, python, Java, etc, lo mejor es generarlo en forma parametrizada declarando las variables como inputs (usando @). De esta forma SQL puede optimizarla y guardar un plan de ejecución para la misma, si se vuelve a ejecutar.

Tener en cuenta que es muy similar a la sintaxis de generar la query en forma de string pero tiene beneficios a nivel de performance.

En entornos multi usuario, esto puede llegar a representar que consultas que llegan desde app se ejecuten hasta 10 veces más rápido.

Parameterized SQL vs Stored procedures: Básicamente, la performance es la misma. PERO hay ventajas en el uso de Stored procedures, tales como:

* Mayor seguridad en el manejo de los datos
* Control de actualizaciones en la DB

Commit impact on performance: En general, las sentencias vienen con auto-commit por defecto. Pero esto trae ciertos problemas:

* Demasiados write en log
* Imposibilidad de correr scripts en forma eficaz. Ya que de haberse ocurrido una carga incompleta previamente, la misma generará errores debido al intento de duplicar data existente.

1. Paralelizar consiste en dividir la tarea de lectura en pequeños bloques de lectura que ocurren en simulataneo y cuyos resultados parciales se concatenan para obtener el resultado final. [↑](#footnote-ref-0)