Comprender el Execution Plan SQL

Una de las tareas más básicas que debe realizar al ajustar las sentencias SQL es determinar las causas del problema de rendimiento, y para hacerlo, lo más probable es que necesite analizar el plan de ejecución de la sentencia problemática.

En este artículo, cubro los conceptos básicos de los planes de ejecución, con el objetivo de ayudar a las personas nuevas en el ajuste de SQL a comprender qué son y cómo se pueden usar como parte de un proceso de ajuste de SQL.

¿Qué es el Execution Plan?

Puede pensar en un plan de ejecución como la combinación de pasos utilizados por la base de datos para ejecutar una instrucción SQL. En un plan de ejecución, puede ver la lista de pasos que la base de datos tendrá que realizar para ejecutar la declaración, junto con el costo asociado con cada paso, lo cual es muy útil porque puede ver cuáles de los pasos están utilizando más recursos, y

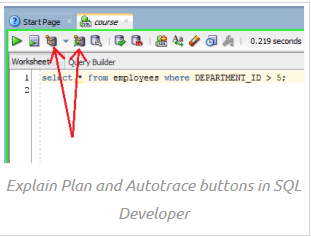
Por lo tanto, probablemente tomar más tiempo para completar.

¿Cómo muestro el Execution Plan?

El método utilizado para mostrar un plan de ejecución y la forma en que se mostrará pueden variar según la herramienta que esté utilizando.  
  
Si está utilizando SQL \* Plus, puede establecer la variable del sistema AUTOTRACE en ON o TRACEONLY, de esta manera:  
  
SET AUTOTRACE TRACEONLY;  
  
Esto hará que SQL \* Plus muestre el plan de ejecución para la consulta que está ejecutando.  
También puedes hacer algo como esto:



Si está utilizando SQL Developer, que es la herramienta más común, simplemente puede presionar la tecla F10 (para explicar el plan) o F6 (para autotrace), o simplemente hacer clic en el botón apropiado en la barra de herramientas de la hoja de trabajo:



Explain plan VS. Autotrace

Explain plan y el Autotrace producen resultados similares (ambos muestran un plan de ejecución), pero hay dos diferencias clave:

El plan que se muestra mediante el plan de explicación es el que la base de datos "piensa" o predice que usará para ejecutar la consulta, mientras que el plan que se muestra mediante autotrace es el plan que realmente se utilizó para ejecutar la consulta (al menos ese es el caso cuando usando autotrace en SQL Developer), y es por eso que **en algunos casos puede obtener diferentes planes para la misma consulta dependiendo de cómo la genere, por eso es importante como se escribe la consulta**.

Además del plan de ejecución, el trazado automático muestra algunas estadísticas o métricas sobre cada uno de los pasos realizados para ejecutar la consulta, lo que puede ser muy útil para investigar problemas de rendimiento.

Es por es preferible usar el autotrace sobre el explain plan en la mayoría de las situaciones. La única desventaja del autotrace podría ser que realmente ejecuta la declaración, por lo que si desea ver el plan de ejecución de una consulta que demora mucho tiempo en ejecutarse, la obtendrá mucho más rápido utilizando el plan de explicación.

La documentación dice que para usar la utilidad autotrace necesita los privilegios SELECT\_CATALOG\_ROLE y SELECT CUALQUIER DICCIONARIO, pero en realidad solo necesita SELECT\_CATALOG\_ROLE.

El Optimizer

El software o proceso en la base de datos que se encarga de generar planes de ejecución para cada declaración que necesita ejecutarse se llama "El Optimizador". Generalmente genera más de un plan de ejecución para cada declaración, y luego, trata de estimar cuál de los planes candidatos sería más eficiente.

Para hacer esta estimación, el optimizador observa las numerosas estadísticas que la base de datos recopila sobre la base de datos y los objetos involucrados en la declaración, como:

* el número de filas en las tablas involucradas
* el tamaño promedio de las filas
* el número de valores distintos en columnas, E / S
* uso de CPU
* etc.

Luego asigna un costo a cada uno de los pasos del plan. Este costo es un número que representa el uso estimado de recursos para cada paso, y no se mide en segundos, ni el tiempo de CPU, ni los bytes de memoria necesarios, ni nada más. Es solo un número o una unidad interna que se utiliza para poder comparar diferentes planes.

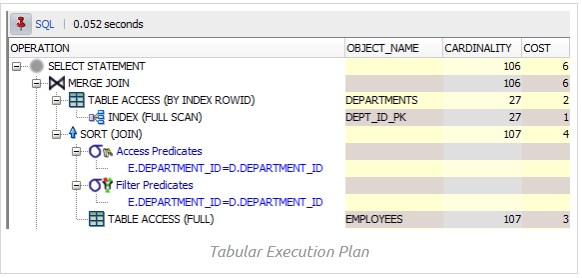
Después de que el optimizador haya estimado el costo de cada uno de los planes candidatos, elige el que tenga el costo total más bajo, por lo que también se le llama optimizador "basado en el costo". Entonces, por ejemplo, si hay 2 planes de ejecución para una consulta, y uno de ellos incluye 15 pasos con un costo total de 30, y hay otro plan candidato con solo 5 pasos pero un costo total de 40, entonces el primero se usa porque tiene el costo más bajo.

Además de estimar los costos y elegir el mejor plan de ejecución para cada declaración, el optimizador también puede decidir "transformar" una consulta en una declaración diferente que produciría los mismos resultados de una manera más eficiente.

Tipos de Operaciones en un Execution Plan

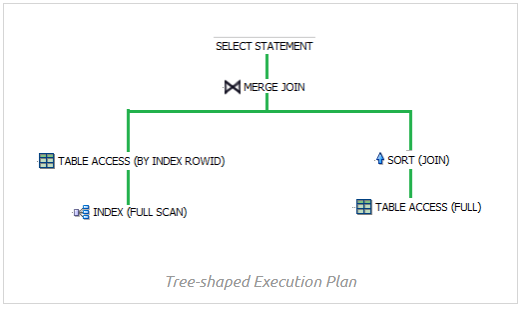
Las operaciones que ve en un plan de ejecución se pueden clasificar en 2 tipos principales: las que acceden o recuperan datos y las que manipulan los datos recuperados por otra operación.

Aquí hay una imagen de cómo SQL Developer muestra los planes de ejecución:



Aunque vemos el plan en forma de tabla, un plan de ejecución es en realidad un árbol, que se atraviesa de arriba hacia abajo, de izquierda a derecha para mostrarlo tal como lo vemos en la imagen.

Así es como se vería este plan si se mostrara como un árbol:



Análisis de las imágenes

Las imágenes de arriba, las operaciones se muestran en el orden opuesto al que se realizan. En el caso de este árbol, los datos se leen primero de los índices y tablas apropiados (las hojas del árbol), y cuando ambos conjuntos de resultados están listos, se unen (la operación MERGE JOIN). En el caso de la representación tabular, las operaciones más profundas son las que se ejecutan primero, con varios hilos de ejecución, de estos hilos, solo se coge el de coste más alto.

La primera línea que vemos es la línea que representa la declaración completa que se está evaluando, por lo que el costo que vemos asociado con esa línea es el costo total para la ejecución de la declaración.

En general, el costo que se muestra para cada operación representa la suma de los costos de sus hijos. En este ejemplo, como puede ver en la imagen del árbol anterior, el ACCESO A LA MESA para la tabla de departamentos y el SORT (JOIN) son los elementos secundarios directos de MERGE JOIN, por lo que el acceso a la tabla tiene un costo de 2 y el orden tiene un costo de 4, la combinación tiene un costo total de 6.

A partir de la segunda línea, vemos las operaciones que realizará la base de datos para obtener los resultados deseados. Cada operación tiene un costo asociado, por lo tanto, esas líneas que no tienen un costo, no son operaciones sino solo detalles informativos sobre la operación de la que forman parte (en el caso del plan en la imagen, los predicados de acceso y filtro son solo detalles sobre la operación SORT).

Las operaciones que acceden o recuperan datos son aquellas que tienen un nombre de objeto asociado, que es la fuente desde donde recuperan los datos (una tabla, un índice, etc.). En el caso del plan en la imagen, los objetos de los que se están recuperando datos son las tablas DEPARTMENTS y ​​EMPLOYEES, y el índice DEPT\_ID\_PK.

La columna de cardinalidad muestra una estimación del número de filas que saldrán de cada una de las operaciones, por lo que, en este caso, el optimizador estimó que se recuperarían 107 filas de la tabla de empleados y 27 de la tabla de departamentos, para producir un Conjunto de resultados finales de 106 filas.

***Operaciones que recuperan filas (Access Paths)***

Como se mencionó, algunas operaciones recuperan filas de fuentes de datos, y en esos casos, la columna object\_name muestra el nombre de la fuente de datos, que puede ser una tabla, una vista, etc.

Sin embargo, el optimizador puede optar por utilizar diferentes técnicas para recuperar los datos en función de la información que tiene disponible de las estadísticas de la base de datos. Estas diferentes técnicas que se pueden usar para recuperar datos generalmente se denominan rutas de acceso, y se muestran en la columna de operaciones del plan, generalmente encerradas entre paréntesis.

A continuación se muestra una lista de las rutas de acceso más comunes con una pequeña explicación de ellas (fuente). <https://docs.oracle.com/database/121/TGSQL/tgsql_optop.htm#TGSQL229>

Full Table Scan

Un Full Table Scan completo lee todas las filas de una tabla y luego filtra aquellas filas que no cumplen con los criterios de selección (si hay una). Los escaneos de tabla completa no son necesariamente algo malo. Hay situaciones en las que un escaneo completo de la tabla sería más eficiente que recuperar los datos utilizando un índice, debido a que el abuso de índices en una tabla puede ser perjudicial.

Table Access by Rowid

Un rowid es una representación interna de la ubicación de almacenamiento de datos (el índice del registro en memoria de la base de datos). El rowid de una fila especifica el archivo de datos y el bloque de datos que contiene la fila y la ubicación de la fila en ese bloque. La ubicación de una fila especificando su rowid es la forma más rápida de recuperar una sola fila porque especifica la ubicación exacta de la fila en la base de datos.  
  
En la mayoría de los casos, la base de datos accede a una tabla por rowid después de una exploración de uno o más índices.

Index Unique Scan

Un Index Unique Scan, devuelve como máximo 1 rowid y, por lo tanto, después de un escaneo único de índice, normalmente verá un acceso a la tabla por rowid (si los datos deseados no están disponibles en el índice). Los escaneos únicos de índice se pueden usar cuando un predicado de consulta hace referencia a todas las columnas de un índice único, utilizando el operador de igualdad. Where Indice = ‘algo’

Index Range Scan

Index Range Scan es una exploración ordenada de valores, y generalmente se usa cuando un predicado de consulta hace referencia a algunas de las columnas iniciales de un índice, o cuando por alguna razón se puede recuperar más de un valor utilizando una clave de índice. Estos predicados pueden incluir operadores de igualdad y no igualdad (=, <.>, Etc.).

Index Full Scan

Index Full Scan lee todo el índice en orden y puede usarse en varias situaciones, incluidos los casos en los que no hay un predicado, pero ciertas condiciones permitirían que el índice se use para evitar una operación de Sorting separada.

Index Fast Full Scan

Un Index Fast Full Scan lee los bloques de índice en orden no ordenado, tal como existen en el disco. Este método se usa cuando todas las columnas que la consulta necesita recuperar están en el índice, por lo que el optimizador usa el índice en lugar de la tabla.

Index Join Scan

Un Index Join Scan es una combinación hash de múltiples índices que juntos devuelven todas las columnas solicitadas por una consulta. La base de datos no necesita acceder a la tabla porque todos los datos se recuperan de los índices.

Operaciones que manipulan datos

Para las operaciones que recuperan datos de la base de datos, hay algunos otros tipos de operaciones que puede ver en un plan de ejecución, que no recuperan datos, sino que operan con datos recuperados por alguna otra operación. Las operaciones más comunes en este grupo son las clases y las uniones.

Sort

Un Sort se realiza cuando las filas que salen del paso deben devolverse en un orden específico. Esto puede ser necesario para cumplir con el orden solicitado por la consulta o para devolver las filas en el orden en que la próxima operación necesita que funcionen como se esperaba, por ejemplo, cuando la siguiente operación es un sort merge join.

Joins

Cuando ejecuta una consulta que incluye más de una tabla en la cláusula FROM, la base de datos necesita realizar una operación de unión, y el trabajo del optimizador es determinar el orden en el que deben unirse las fuentes de datos, y el mejor método de unión para utilizar para producir los resultados deseados de la manera más eficiente posible.  
  
Ambas decisiones se toman en base a las estadísticas disponibles.  
  
Aquí hay una pequeña explicación de los diferentes métodos de combinación que el optimizador puede decidir usar:

Nested Loops Joins

Cuando se utiliza este método, para cada fila del primer conjunto de datos que coincide con los predicados de una sola tabla, la base de datos recupera todas las filas del segundo conjunto de datos que satisfacen el predicado de unión. Como su nombre lo indica, este método funciona como si hubiera 2 anidados para bucles en un lenguaje de programación procesal, en el que para cada iteración del bucle externo se recorre el bucle interno para encontrar las filas que satisfacen la condición de un join.

Como puede imaginar, este método de combinación no es muy eficiente en conjuntos de datos grandes, a menos que se pueda acceder de manera eficiente a las filas en el conjunto de datos interno (a través de un índice).

En general, las uniones de bucles anidados funcionan mejor en tablas pequeñas con índices en las condiciones del join.

Hash Joins

La base de datos utiliza una combinación hash para unir conjuntos de datos más grandes. En resumen, el optimizador crea una tabla hash (estructura de datos índice / clave valor) A partir de uno de los conjuntos de datos (generalmente el más pequeño) utilizando las columnas utilizadas en la condición de unión como la clave, y luego escanea el otro conjunto de datos aplicando el misma función hash para las columnas en la condición de unión para ver si puede encontrar una fila coincidente en la tabla hash construida a partir del primer conjunto de datos.

Realmente no necesita comprender cómo funciona una tabla hash. En general, lo que necesita saber es que este método de combinación se puede usar cuando tiene una combinación equitativa, y que puede ser muy eficiente cuando el conjunto de datos más pequeño se puede guardar completamente en la memoria.

En conjuntos de datos más grandes, este método de combinación puede ser mucho más eficiente que un bucle anidado.

Sort Merge Joins

Un Sort Merge Join es una variación de un Nested Loop Join. La principal diferencia es que este método requiere que las 2 fuentes de datos se ordenen primero, pero el algoritmo para encontrar las filas coincidentes es más eficiente.

Este método generalmente se selecciona cuando se unen grandes cantidades de datos cuando la unión usa una condición de desigualdad, o cuando una unión hash no podría colocar la tabla hash para uno de los conjuntos de datos completamente en la memoria.