**Diseño Físico**

**Documentación del diseño físico:**

Para los requerimientos funcionales 9 y 10 se utilizaron índices sobre las llaves primarias para optimizar las consultas debido a que, solo se requiere filtrar en cada paso por el id que es la llave primaria de cada tabla. Por lo tanto, no fue necesario crear otros índices ya que no se afectaría el resultado en costos de la consulta. De igual forma, para el caso de las tablas que poseen llaves primarias compuestas, fue de gran utilidad la creación automática de índices por Oracle por cada una de las columnas que conforman la llave primaria para que el orden en el que se filtre no afecte el costo de la consulta.

Para los requerimientos 11 y 12 si fue necesaria la creación de índices que optimicen las consultas debido a que se requiere filtrar la información dependiendo de columnas tales como el nombre de una localidad, los requerimientos que satisface un sitio, el estado de una boleta, la fecha y hora de una función, entre otras. Los índices que se crearon fueron:

• Sobre la columna NOMBRE de la tabla LOCALIDAD

• Sobre la columna REQUERIMIENTOS de la tabla REQUERIMIENTOS\_SITIO

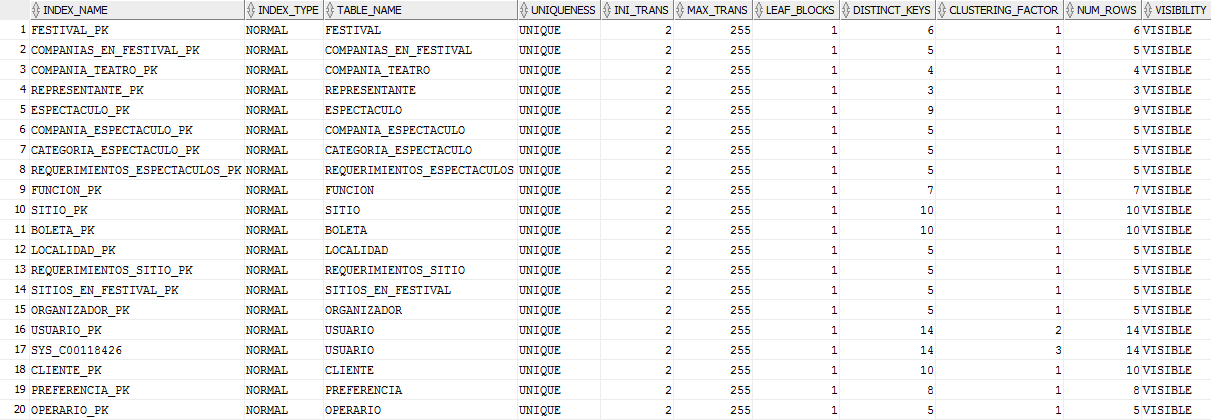
• Sobre la columna ESTADO de la tabla BOLETA

• Sobre la columna FECHA\_HORA de la tabla FUNCION

En precisión, los índices que afectan a ambas consultas son el que se coloca sobre el nombre de la tabla localidad y el que se coloca sobre el estado de la tabla boleta. Se justifica el uso de estos dos índices, porque en ambas consultas se realizan filtros con la instrucción WHERE para encontrar valores en las columnas de los índices que cumplan los requerimientos de cada consulta en específico. En cuanto a los dos índices restantes, estos solo afectan el rendimiento de la consulta 11. Esto sucede dado que solo en la consulta 11 se revisan los valores de los requerimientos que satisface un sitio y la “fecha\_hora” de una función. Por lo tanto, el uso de estos 4 índices está justificado desde el punto de vista de los requerimientos de consulta 11 y 12.

El tipo de índice que se utilizo fue el de árboles B que es el tipo de índice standard utilizado por Oracle. Este tipo de índice es de gran utilidad para nuestras consultas debido a que para la filtración de información se usan las llaves primarias que aseguran un alto nivel de selectividad, característica que promueve el uso de los arboles B. De igual forma, se utilizó el subtipo “índices ascendentes” para almacenar los datos de las columnas afectadas por los índices en orden ascendente. Por otra parte, se manejaron índices primarios y secundarios para las tablas. Los índices primarios se utilizaron en las llaves primarias de las tablas y los secundarios se usaron en los índices creados para las columnas seleccionadas. Con respecto a costos de almacenamiento y mantenimiento, Oracle utiliza una llave de compresión “key compression” para comprimir los valores guardados por los índices y agrupar los datos repetidos, lo que se encarga de reducir enormemente el costo de almacenamiento utilizado por los índices. Por último, Oracle también ofrece una característica llamada “Oracle partitioning” que permite subdividir las tablas y los índices en particiones para reducir los costos de almacenamiento en grandes cantidades de datos, mejorar el rendimiento de las búsquedas y acceder a los datos con un nivel más fino de granularidad.

La siguiente imagen muestra los índices creados automáticamente por Oracle para nuestro usuario (ISIS2304A231720):



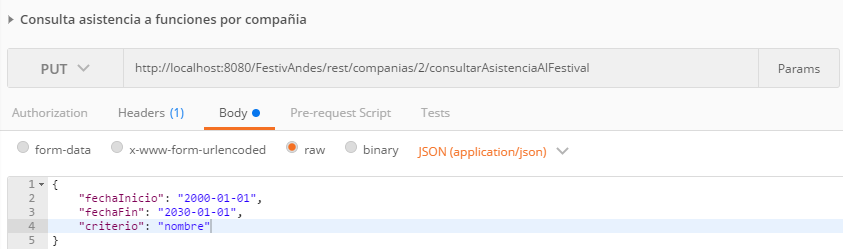
Estos índices creados hacen referencia a las llaves primarias de cada tabla creada. Fueron creados por Oracle debido a que las consultas, de manera global, se suelen hacer por la llave primaria. Además, la llave al ser única, no tiene valores repetidos lo que le da una alta selectividad en los datos que mejora el rendimiento de las consultas.

Los índices creados automáticamente por Oracle mejoran el rendimiento de los requerimientos funcionales debido a que cuando se quiere filtrar por la llave primaria de una tabla o cuando la condición que se debe cumplir en un join incluye está llave, los costos y el tiempo de ejecución se disminuyen altamente.

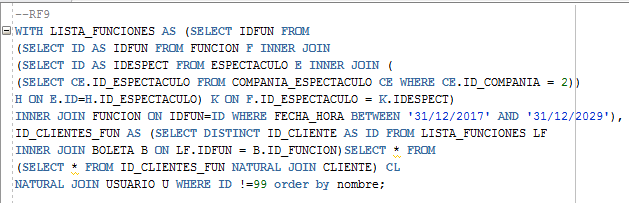
**Documentación del análisis realizado:**

**Requerimiento funcional 9**

El escenario de pruebas entrega un resultado que incluye una totalidad de 2 tuplas que cumplen con el rango de fechas y el criterio designado por el usuario para la compañía seleccionada. Estos son los datos utilizados para la consulta:



La sentencia SQL que responde al requerimiento es la siguiente:



Para la distribución de los datos en este requerimiento, se pueden utilizar otros escenarios de prueba en donde se cambie la compañía, el rango de fechas y el criterio para ver cómo se cambia el resultado.

Se recomienda utilizar los siguientes parámetros:

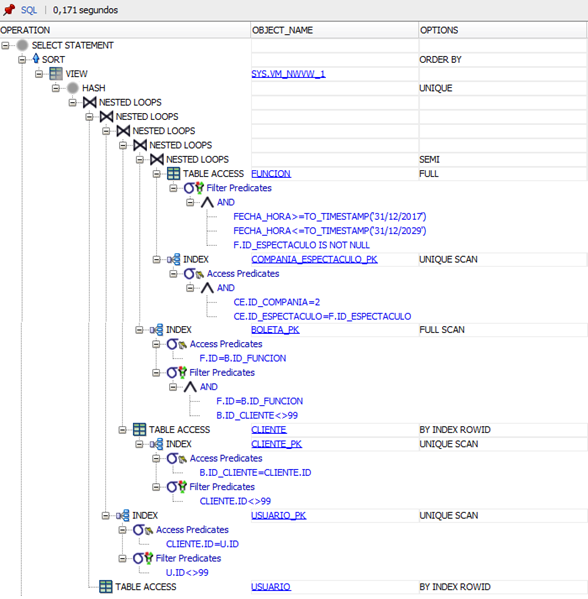
Compañía: 4

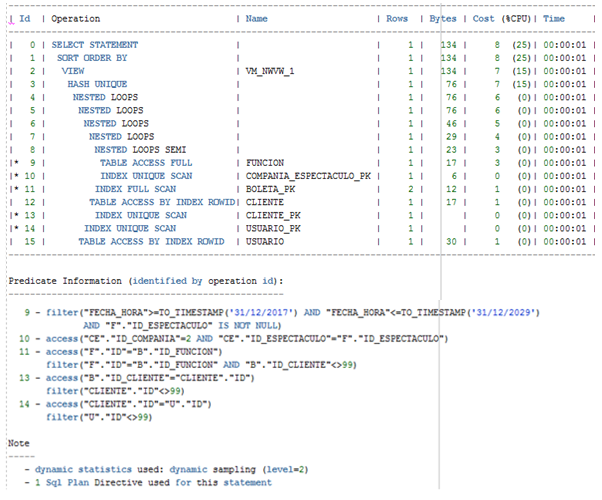
Fechas: 2017-01-01 al 2017-12-31

Criterio: Rol

El plan de consulta obtenido en Oracle muestra la utilización de los índices creados automáticamente para el acceso a la información de manera más eficiente. Esto se evidencia al buscar la compañía por el id dado o al comparar dos llaves primarias en dos tablas que tienen relación. De igual forma, se utilizan una gran cantidad de Nested Loops para comparar cada una de las tuplas y poder llevar a cabo el respectivo join. El resultado final de cada join se almacena en una tabla de hash con llaves únicas para el acceso a la información debido al uso del distinct en la sentencia SQL. Luego, se nota un sort que se hace para organizar las tuplas que cumplen las condiciones previas por el criterio asignado. Finalmente se hace uso del SELECT STATEMENT para mostrar los valores en el query que entregaron sus hijos.

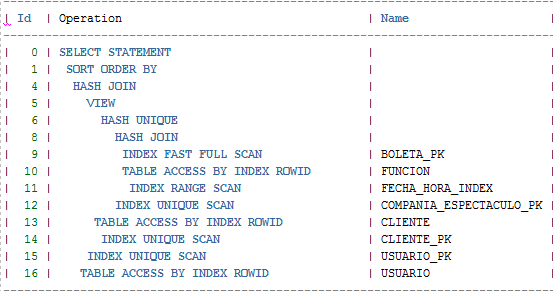
El plan de ejecución y el tiempo obtenido se muestra en la siguiente imagen:





**Análisis de eficiencia**

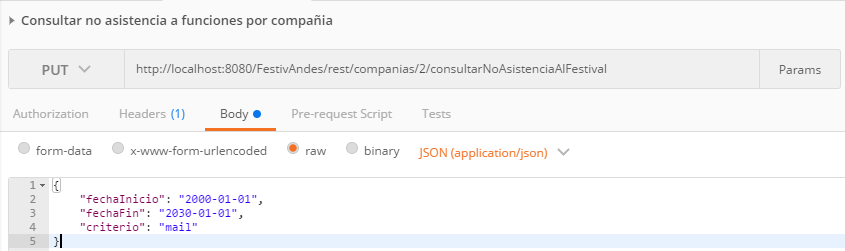
**Plan de ejecución diseñado:**

****

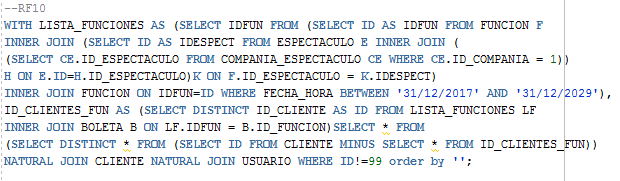
EL plan de ejecución que creamos para este requerimiento funcional utiliza los índices para el recorrido parcial de las tablas y el acceso directo la información de la misma manera que el creado por SQLDevelopper. Sin embargo, en nuestro plan de ejecución utilizamos en remplazo de los Nested loops, hash joins para realizar los inner joins de la consulta debido a que para grandes cantidades de datos el costo del hash join crece de manera lineal con una buena función de hash mientras que el nested loop crece de manera exponencial.

**Requerimiento funcional 10**

El escenario de pruebas utilizado incluye una cantidad de # datos por cada una de las 6 tablas utilizadas. De estos datos, el resultado solo incluye una totalidad de # tuplas que cumplen con el rango de fechas y el criterio designado por el usuario para la compañía seleccionada. Estos son los datos utilizados para la consulta:



La sentencia SQL que responde al requerimiento es la siguiente:



Para la distribución de los datos en este requerimiento, se pueden utilizar otros escenarios de prueba en donde se cambie la compañía, el rango de fechas y el criterio para ver cómo se cambia el resultado.

Se recomienda utilizar los siguientes parámetros:

Compañía: 3

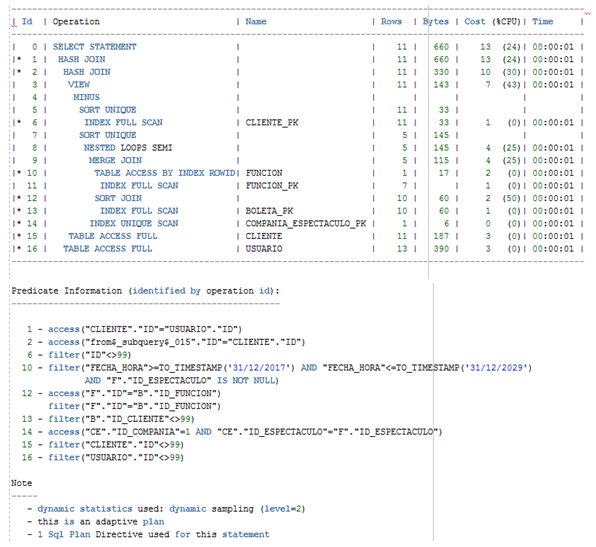
Fechas: 2010-05-01 al 2010-12-31

Criterio: Nombre

El plan de consulta obtenido en Oracle muestra la utilización de los índices creados automáticamente para el acceso a la información de manera más eficiente. Esto se evidencia al buscar la compañía por el id dado o al comparar dos llaves primarias en dos tablas que tienen relación. En este plan de ejecución se ve el uso del Merge Join y el Sort Join para realizar los respectivos joins de manera eficiente y teniendo en cuenta el orden adecuado. Además, se ve el uso del Minus que requiere del Sort Unique para organizar la información antes de realizar la resta. Finalmente se hace uso de un hash join para juntar el resultado final de los anteriores joins para culminar la consulta con el SELECT STATEMENT para mostrar los valores en el query que entregaron sus hijos.

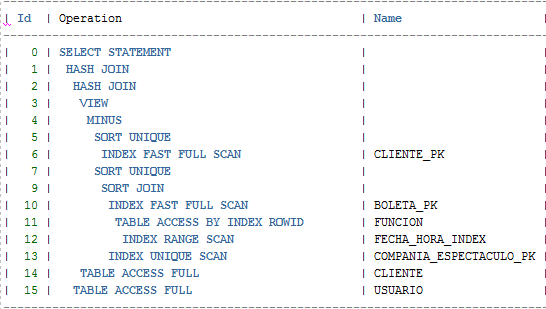
El plan de ejecución y el tiempo obtenido se muestra en la siguiente imagen:





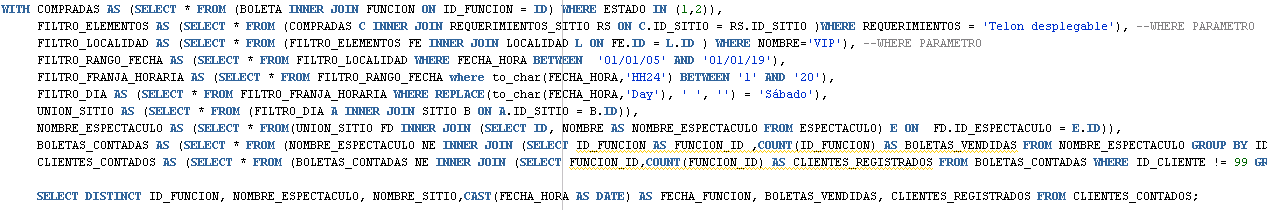
**Análisis de eficiencia**

**Plan de ejecución diseñado:**

****

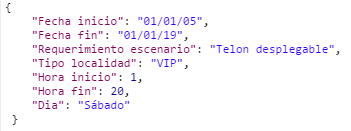
EL plan de ejecución que creamos para este requerimiento funcional utiliza los índices para el recorrido parcial de las tablas y el acceso directo la información de la misma manera que el creado por SQLDevelopper. Sin embargo, en nuestro plan de ejecución utilizamos en remplazo de los Nested loops un sort-merge join para ordenar la información de los inner joins para luego devolverla al sort unique con el motivo de realizar el Minus que requiere la información ordenada. Adicionalmente, el sort-merge join es eficiente para una gran cantidad de datos en comparación del nested loop join.

**Requerimiento funcional 11**

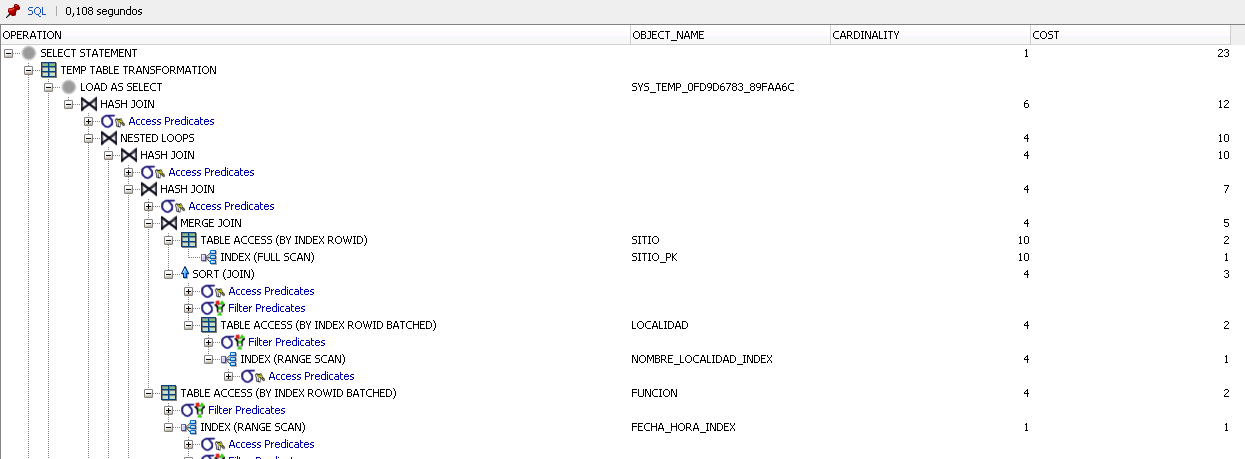
****

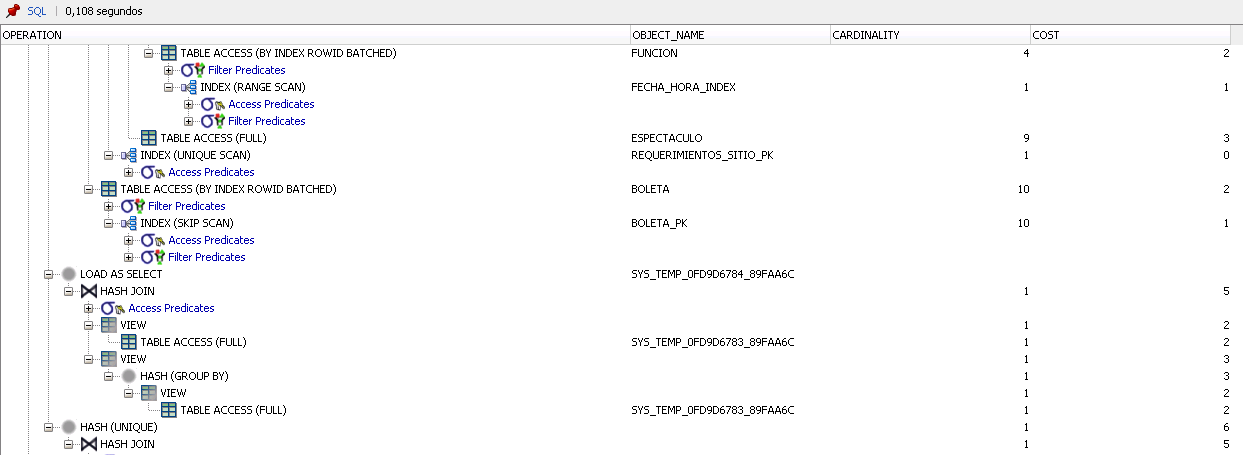
El tamaño de la respuesta puede cambiar ampliamente dependiendo del valor de los parámetros. Esto se puede justificar si se observa que existen dos rangos que se mandan como parámetros, el rango de fecha y de hora. Si se amplían ambos rangos, existirán mas datos que cumplan las condiciones. Además, esta consulta funciona a base de filtros, por lo tanto, si no se envía ningún parámetro, el tamaño de la respuesta puede aumentar mucho más dado que no se aplicará ninguno de los posibles filtros. En este caso, como se usan todos los filtros posibles, es más factible que la respuesta sea muy reducida.

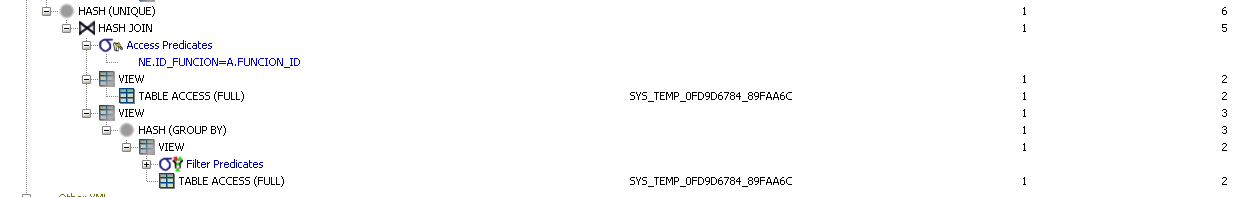
**Datos de prueba:**

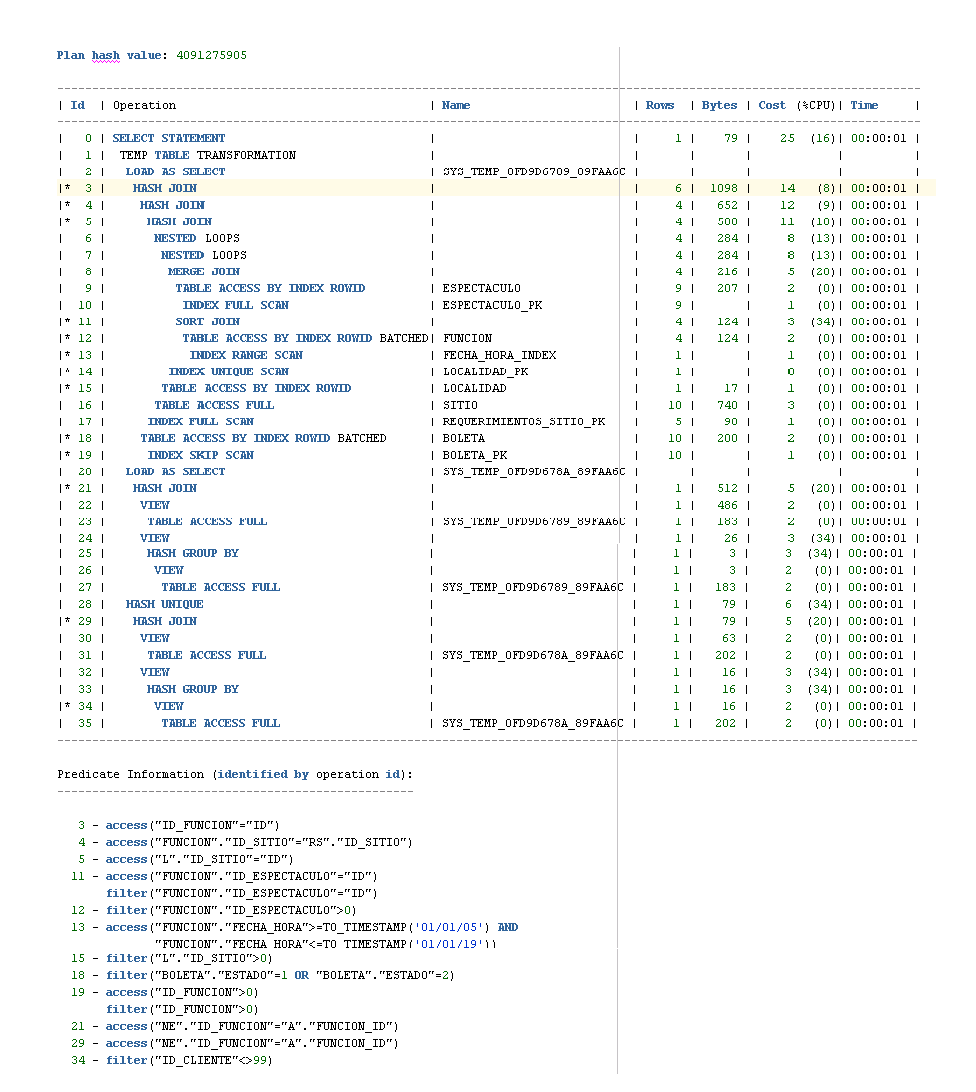
****

El uso de todos los filtros posibles en esta consulta afecta el plan de ejecución de manera que cuando se utilizan o no los filtros (si se envían o no los parámetros) las sentencias utilizadas son ligeramente distintas. De manera que los planes de ejecución cambian, ya que en los casos que se utilizan los parámetros se aplica un filtro con la instrucción WHERE.



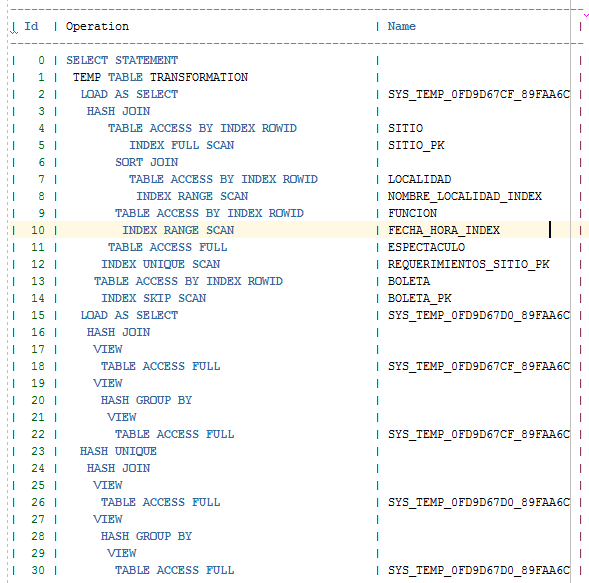






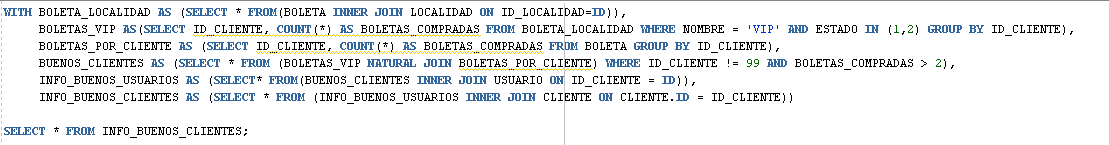
**Análisis de eficiencia**

**Plan de ejecución diseñado:**

****

EL plan de ejecución que creamos para este requerimiento funcional utiliza los índices para el recorrido parcial de las tablas y el acceso directo la información de la misma manera que el creado por SQLDevelopper. Sin embargo, en nuestro plan de ejecución utilizamos en remplazo de los Nested loops, hash joins para realizar los inner joins de la consulta debido a que para grandes cantidades de datos el costo del hash join crece de manera lineal con una buena función de hash mientras que el nested loop crece de manera exponencial. De igual forma, utilizamos una menor cantidad de hash joins en nuestro diseño debido a que no le vimos la utilidad a los propuestos en el plan de ejecución generado por Oracle.

**Requerimiento funcional 12:**

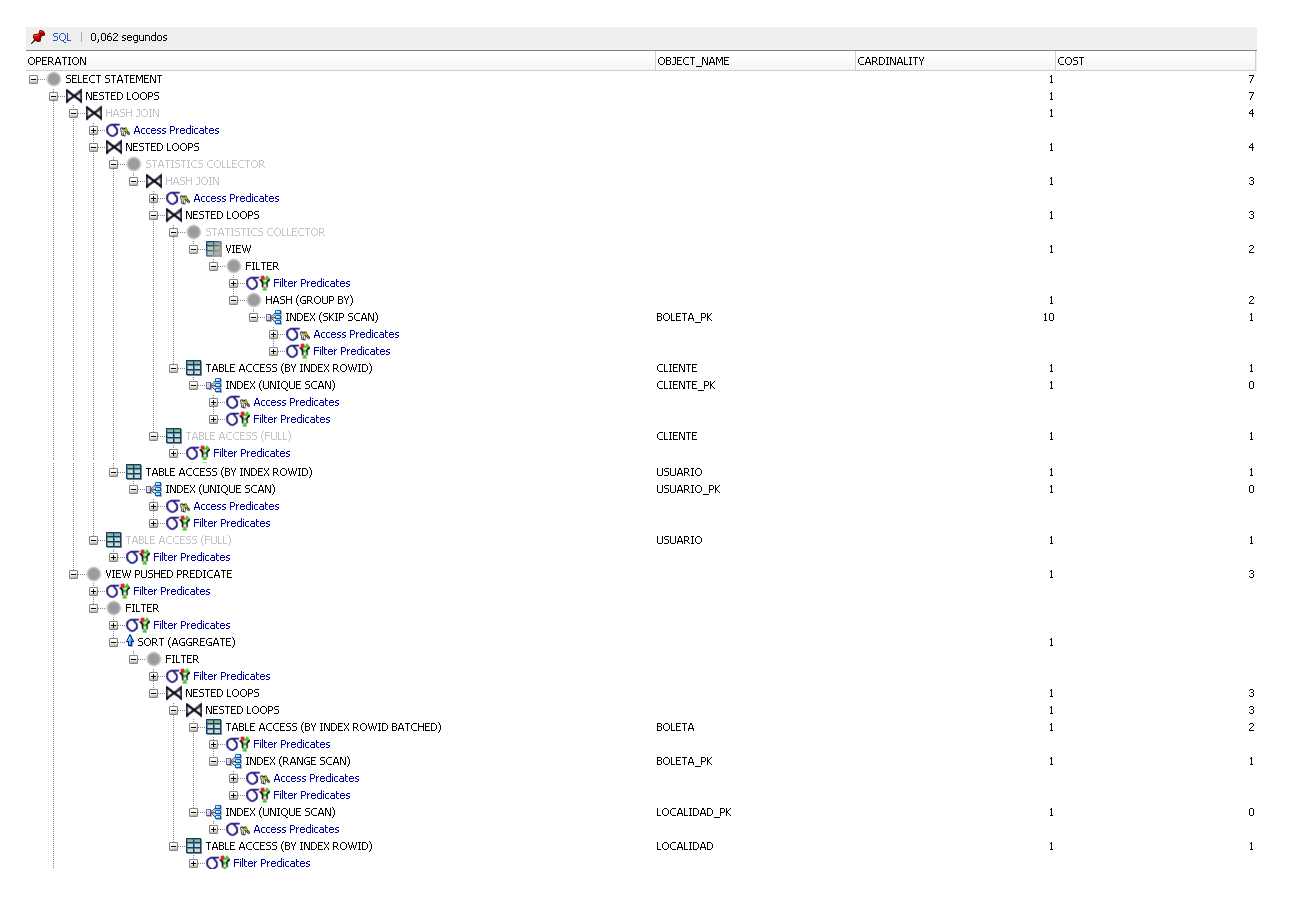
****

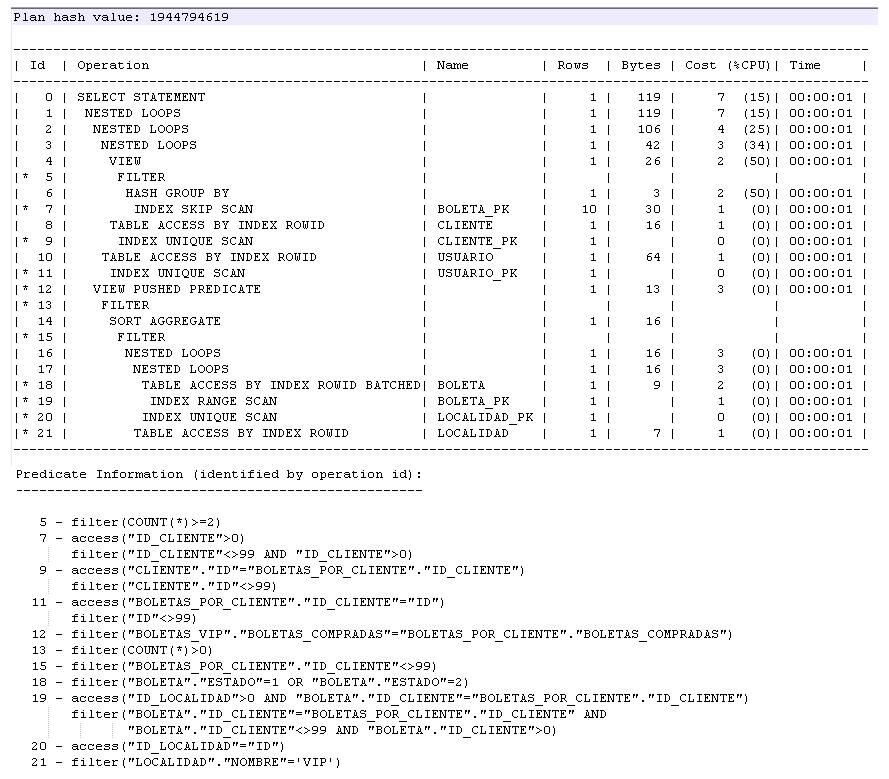
En cuanto a esta consulta, el tamaño del resultado se ve directamente afectado por la variación en el valor del único parámetro que recibe. El único parámetro que recibe esta consulta es el número de boletas VIP que al menos debe comprar un cliente para ser “un buen cliente”. Por lo tanto, entre más se reduzca este número, más amplio será el resultado obtenido, ya que la exigencia que deben cumplir los clientes para ser catalogados como buenos clientes es menos.

**Parámetro:**

N (Mínimo boletas VIP): 2

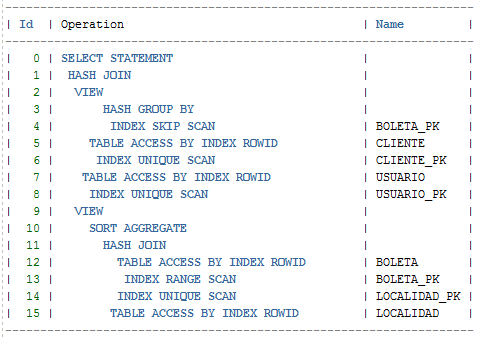
La variación del valor de este parámetro no cambia los planes de ejecución que se obtienen, ya que n siempre tiene que tener algún valor asignado para que la consulta pueda ser realizada. Lo que significa que el plan de ejecución siempre va a mostrar la etapa en donde se filtra por el valor dado por parámetro.





**Análisis de eficiencia**

**Plan de ejecución diseñado:**

****

EL plan de ejecución que creamos para este requerimiento funcional utiliza los índices para el recorrido parcial de las tablas y el acceso directo la información de la misma manera que el creado por SQLDevelopper. Sin embargo, en nuestro plan de ejecución utilizamos en remplazo de los Nested loops, hash joins para realizar los inner joins de la consulta debido a que para grandes cantidades de datos el costo del hash join crece de manera lineal con una buena función de hash mientras que el nested loop crece de manera exponencial. En adición, no vimos la utilidad de utilizar los filters que usa Oracle debido a que nosotros a que los resultados generados por las consultas los tratamos directamente en nuestros hash joins sin eliminar filas.

**Bibliografía**

[**https://docs.oracle.com/cd/E11882\_01/server.112/e40540/indexiot.htm#CNCPT811**](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CNCPT811)

[**http://www.oracle.com/technetwork/es/articles/database-performance/oracle-partitioning-en-database-12c-2244565-esa.html**](http://www.oracle.com/technetwork/es/articles/database-performance/oracle-partitioning-en-database-12c-2244565-esa.html)

[**https://docs.oracle.com/database/121/TGSQL/tgsql\_interp.htm#TGSQL277**](https://docs.oracle.com/database/121/TGSQL/tgsql_interp.htm#TGSQL277)

[**https://www.inf.utfsm.cl/~wpalma/ari/indices.pdf**](https://www.inf.utfsm.cl/~wpalma/ari/indices.pdf)