Santiago Vela – 202026767

Jesús Alejandro Dávila Pinchao - 202014263

Iteración 4 Hotel Andes

Contenido

1. **Diseño físico**
   1. **selección de indices**
   2. **informacion asociada a los indices**
2. **Analisis requerimientos funcionales de consulta**
3. **Analisis de eficiencia**
4. **Construccion aplicación de ejecución**
5. **Carga de datos**

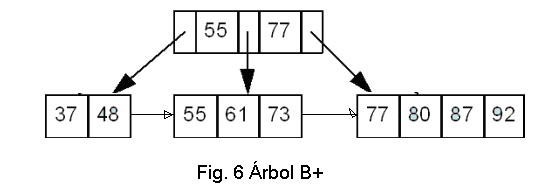
**1.DISEÑO FISICO**

**Selección de índices:**

**Requerimientos funcionales 1-11**

Para los requerimientos funcionales de 1 al 11, la mayoría si no todos los requerimientos son de actualización, esto significa ingresar nuevos elementos a mis tablas, actualizar nuevos elementos a mis tablas, como es el caso del requerimiento 2 en donde agrego un nuevo usuario a mi base de datos de usuario, para este escenario, se sabe que cada usuario tendrá un distintivo único que lo va a diferenciar de los demás y es su PK, entonces, con esto dicho un árbol B+ me ayudara bastante a hacer todo este tipo de consultas y

este nuevo tipo de inserciones, ya que todos los caminos de la raíz a las hojas tienen la misma longitud, es decir que tanto para insertar nuevos registros que como para leerlo este va a ser muy bueno, adicionalmente los nodos hoja se encuentran unidos entre sí como una lista enlazada es decir los usuario nuevos se podrán encontrar fácilmente esto para permitir principalmente recuperación en rango mediante búsqueda secuencial, como se dijo anteriormente para este tipo de requerimientos este tipo de árbol B+ le ayudaría a todos.



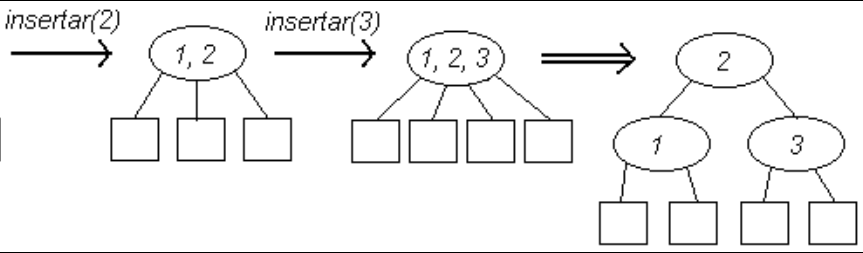
**Árbol b+**

**Requerimientos funcionales 13-14**

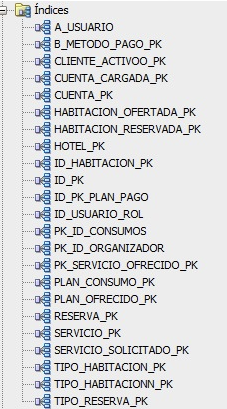
Para los requerimientos funcionales del 13 al 14 es bueno que utilicen B+ , ya que en este caso la consultas que hacen son sobre índices ya existente en nuestras tablas como lo es el caso de cancelar las reservas de una convención, cuando nos piden eso claramente nos dará un pk el cual servirá para localizar la convención de una manera mas rápida , de igual manera en el momento de localizar las reservas que se quieren cancelar se tiene que hacer un Join con las tablas correspondientes, pero para eso un b+ no serviría mucho ya que tendría que recorrer cada registro de mi tabla y preguntarme si esta tabla tiene el registro de mi pk de mi tabla anterior

**Requerimientos funcionales 12-15-16**

Para el caso de los requerimientos 12,15 y 16 en donde no estamos trabajando con PK o con índices debido a que la creación del modelo relacional es así, no serviría hacerlo por un índice B+ ya que no se podría filtrar ni saber información, sin embargo con uno árbol secundario, a pesar de que la respuesta en términos de tiempo no va a ser la misma ya que esta se demorara mas porque le toca hacer una lectura secuencial (LSR) que a diferencia de la aleatoria es un poco mas demorada. Lo malo del uso de usar arboles secundarios es que no se podrá tener la información en orden y no se podrá buscar información sobre algo si no hasta que recorra todo árbol podre saberlo.

****

**Información asociada a los índices**

****

Los índices realizados por Oracle tienen como fin obtener un acceso rápido a la información por medio de llaves primarias en su mayoría, ya que interpreta a la llave primaria como el principal distintivo entre cada tupla y por ende mas fácil de encontrar. Para el caso de los requerimientos, estos índices que fueron creados por Oracle, nos ayudaron en gran manera ya que para operaciones como los Joins o lecturas aleatoria, fue muy bueno, puesto que se evitaba recorrer toda la tabla buscando registros que cumplieran con las condiciones impuestas desde la creación de la consulta, adicionalmente Oracle nos ayuda a crear estos índices para optimización de consultas, dependiendo de lo que se pide. Finalmente a pesar de que estos índices no están estructurados en un árbol b+ primario, debido a que muchas veces puede que se repita información, estos índices nos ayudaran a mejorar el costo tanto de CPU como de Tiempo.

**A**NALISIS REQUERIMIENTO FUNCIONAL DE CONSULTA

**RFC1:**- MOSTRAR EL DINERO RECOLECTADO POR SERVICIOS EN CADA HABITACIÓN DURANTE UN PERIODO DE TIEMPO Y EN EL AÑO CORRIDO

**Sentencia SQL:**

SELECT h.NUMEROHABITACION, c.TOTAL

FROM a\_habitacion h, a\_habitacion\_reservada hr, a\_cuenta c, a\_reserva r

WHERE h.idHabitacion = hr.idHabitacion

AND hr.idCuenta = c.idCuenta

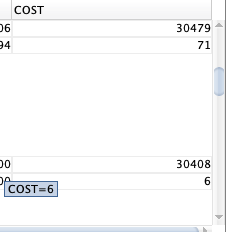
AND r.fechaEntrada BETWEEN '11/02/12' AND '16/12/13'

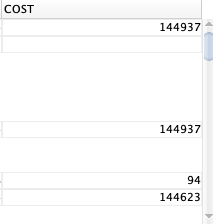
AND r.fechasalida BETWEEN '11/02/14' AND '16/12/14';

**Análisis de distribución:**

Para este análisis nos piden dar el dinero recolectado por los servicios en cada habitación, dicho esto en nuestro caso debido a la forma en que organizamos todas las tablas, tenemos que involucrar 4 tablas, ya que para cumplir con la normalización de toda la base de datos, salieron mas tablas que en comparación al primer caso.

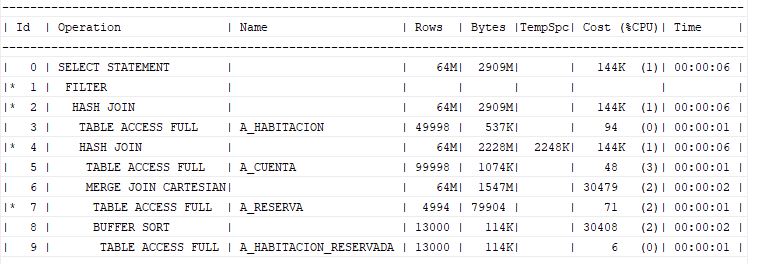
Con esto dicho, para cada tabla que vamos a usar, hay unas que tienen mas registros que otra como ejemplo habitación que tiene aproximadamente 10000 registros, por otro lado la tabla “habitación reservada”, tiene 13000 registros; la tabla “cuenta” consta de 100 000 registros y finalmente la tabla “reserva” cuenta con 50.000 registros. Según lo que nos pide el requerimiento, necesitamos saber de todas las habitaciones cuanto dinero han generado los servicios, el hecho de uno usar un índice de búsqueda o no preguntar en específicamente por una habitación, implicara que nos tocara revisar todas la tablas completas para eso los costos serán sumamente significativos, y dichos costos son debido también a los “**joins**” que nos toca usar, puesto que son 4 tablas, dependiendo la unión usaremos un “**hash join** ” o un “**merge join**”. Finalmente si la consulta hubiera sido por una habitación en especifico, el costo no seria tan grande como el costo final que se tuvo.





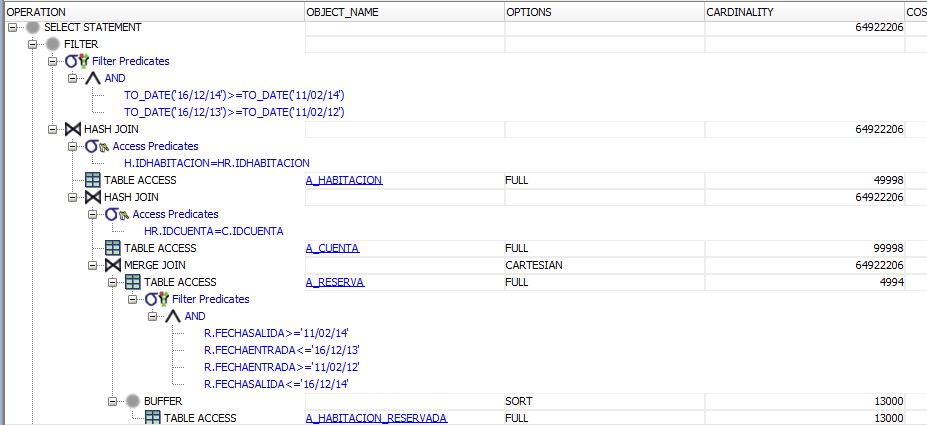
COSTO CONSULTA

**Valores Parámetros utilizados:**

****

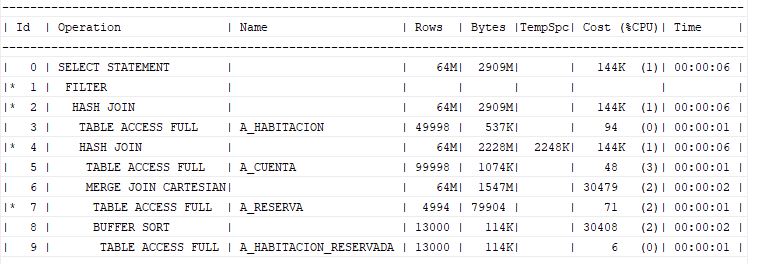
Como podemos darnos cuenta, en este caso los parámetros utilizados para este análisis fue el uso completo de todas las tablas, sin embargo con el uso de los algoritmos los valores de esos mismos para metros se ven reducidos poque Oracle busca aprovechar de mejor manera las tablas y los parámetros

**Plan de consulta Oracle:**

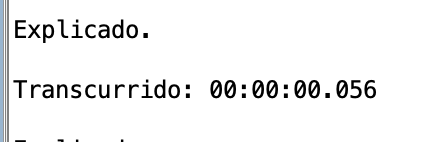
****

Como podemos darnos cuenta en el plan de ejecución lo primero y lo que hace primero Oracle es el predicado o la condición de mi consulta, ya que si se hiciera una consulta en base a un índice sería mucho más fácil hacer el plan de ejecución ya que, tendríamos como índice, el cual nos ayudaría a recorrer la tabla de una manera mas rápida y mas sencilla, desues podemos ver que dependiendo la tabla y la relación en común que se tiene, se utilizara un “Hash join” o un “mege join” los cuales con base a los aspectos en común se podrán manejar mejor las tablas en memoria.

**Tiempos obtenidos por la ejecución:**







Como podemos darnos cuenta esta consulta el hecho de que sea costosa, no siempre significa que en todos los caso vaya a ser rápida o lenta, en este caso en la primera tabla antes de hacer el Join lo tenemos que hacer es un LSR(lectura secuncial rápida) ya que tenemos que recorrer toda la tabla que coincida con el predicado que quierp, luego se usa un “ Hash join” por lo cual ninguna de las tablas caben en memoria juntas, por lo cual el uso de ese Join me permitirá agilizar los procesos ya que es dividir las tablas ya aplicarlos función hash y comparar. Gracias a eso la consulta no llega a ser tan costosa como se esperaría o se pensaría.

**RFC2:**- MOSTRAR LOS 20 SERVICIOS MÁS POPULARES. MOSTRAR LOS 20 SERVICIOS MÁS POPULARES:

**Sentencia SQL:**

SELECT idservicio, count(idservicio)

from a\_cuenta\_cargada

GROUP BY idservicio

order by count(idservicio) desc

FETCH NEXT 20 ROWS ONLY;

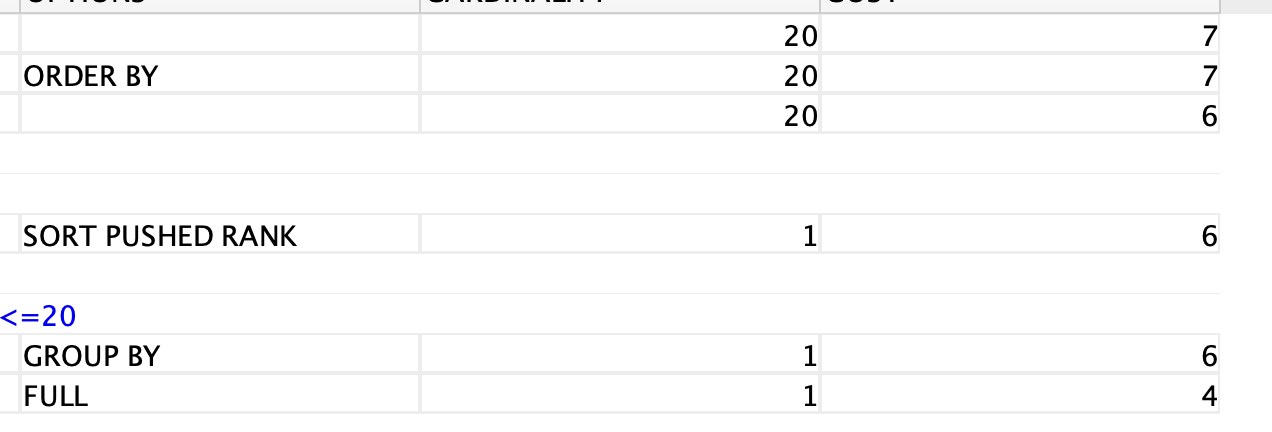
**Análisis de distribución :**

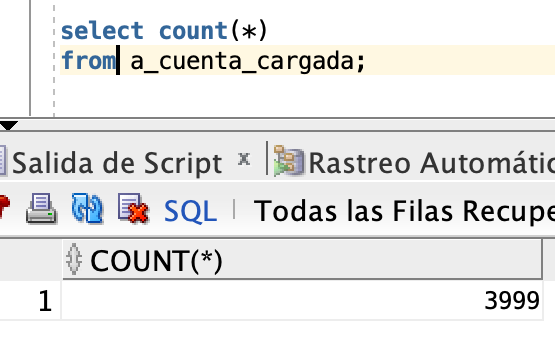
Para este análisis nos están pidiendo por los 20 servicios mas populares, si bien nos piden solo 20 , Oracle empezara realizando primero las consultas del Sort y luego hará un ordenamiento de esas consultas para que posteriormente se aplique el “Where” el cual ayudara a filtrar esos datos previamente seleccionados.

Para el caso de la tabla “**a\_cuenta\_cargada**” la cual nos dice por las todas las cuentas de clientes los servicios que ha pedido desde su llegada a hotel andes, nos podemos dar cuenta que la tabla contiene 4 mil registros lo que puede llegar a significar que un servicio a sido pedido una y mas veces. Si bien es cierto existen 1583 servicios únicos, estos son pedidos 4 mil veces.

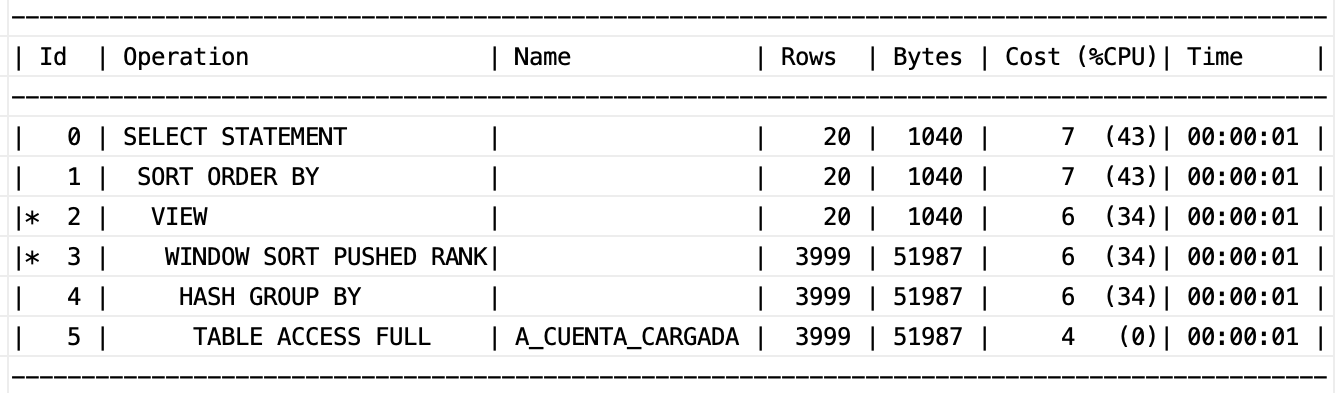
Para el primer análisis en donde nos piden los 20 primeros mas pedidos podemos darnos cuenta que tiene un tiempo de respuesta de el cual es una ventaja grande debido a que al no tener que hacer operaciones en mas tablas, si no que todo se hace en la misma tabla que es la que tiene la cantidad de servicios pedidos. De la misma forma este requerimiento tiene un costo el cual comparado con otros requerimiento no es tan alto, sin embargo el costo de consulta mas grande se puede ver presentando en los “Select” y en el “ Sort”.





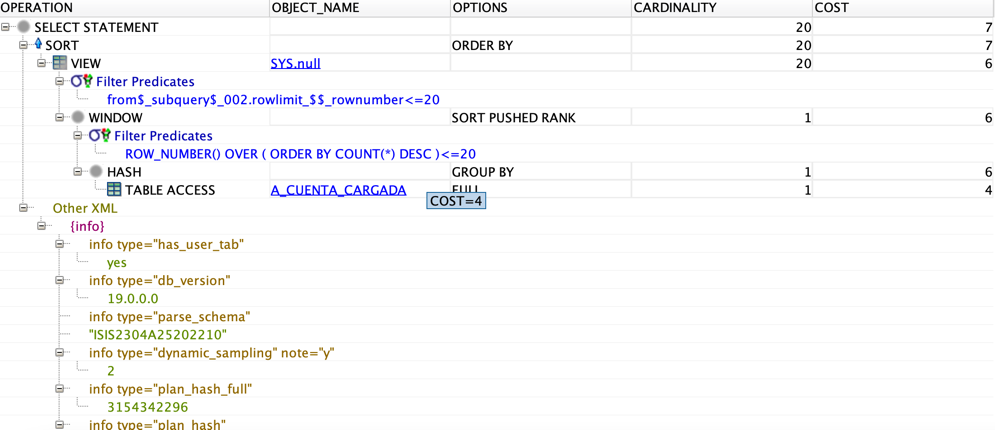


**Valores Parámetros utilizados:**

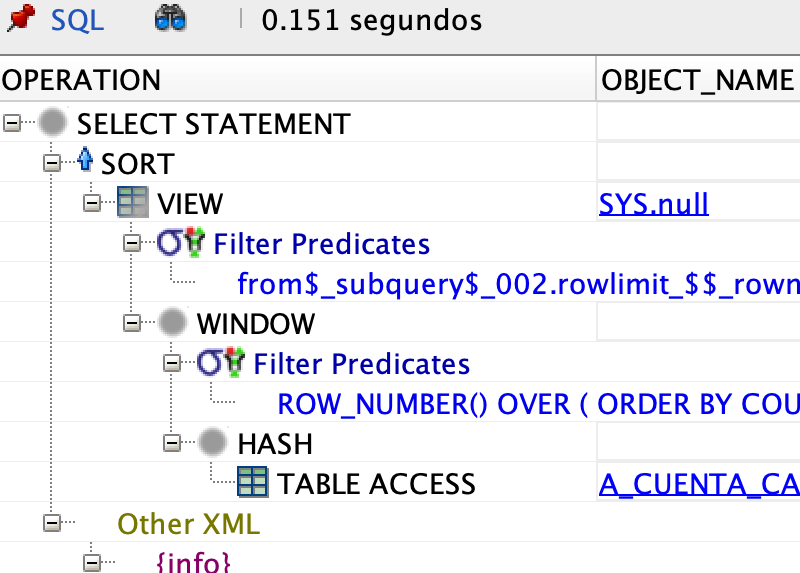


Debido a que todo esta en una tabla costo no son tan altos ni los valores utilizados, sin embargo como podemos darnos cuenta en el momento de hacer el hash group la cantidad de Bytes aumenta, por ende también la cantidad de registros igual

**Plan de consulta Oracle:**

****

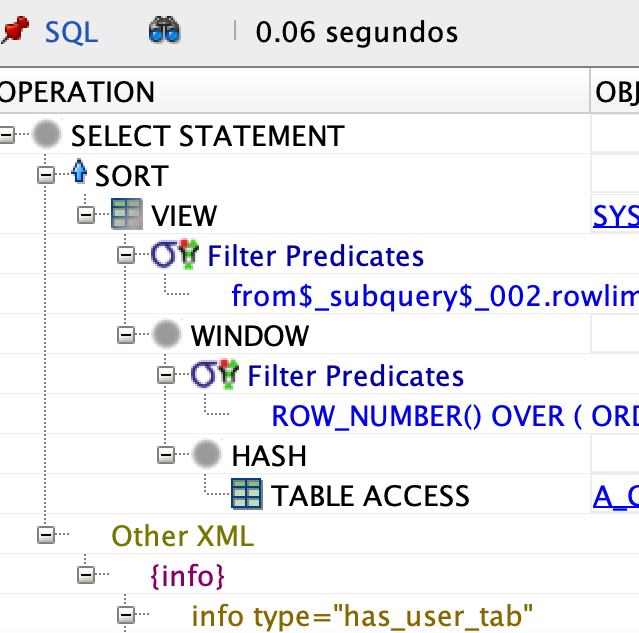
**Tiempos obtenidos por la ejecución:**



Los tiempos de ejecución de esta requerimiento no son tan grandes comparados con otros, y esto se debe a la implementación que creamos y usamos desde el principio y todo es debido a que no tenemos que hacer “Joins” y puesto que lo que tenemos que hacer es lectura secuenciales rápida (LSR) ya que tenemos que recorrer toda la tabla mirando cuantas veces un servicio se repite y posteriormente a esto, y ordenando uno a uno los servicios que mas se van repitiendo.

Si cambiamos la ejecución y si pedimos con las 30 registros mas utilizados la ejecución y el tiempo de respuesta cambiara.

Puesto a que se había hecho la ejecución antes la respuesta nuevamente no fue demorada ya que era organizar datos que habom habíamos tenido antes .



**RFC3:**- MOSTRAR EL ÍNDICE DE OCUPACIÓN DE CADA UNA DE LAS HABITACIONES DEL HOTEL

**Sentencia SQL:**

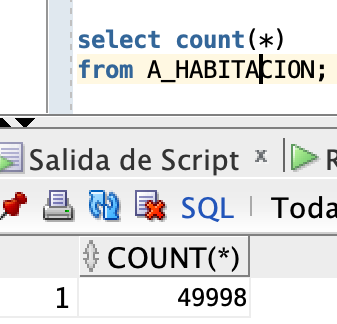
SELECT HH.IDHABITACION, ah.capacidad

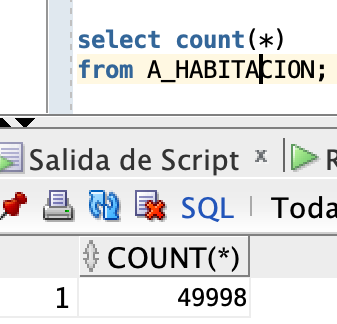
FROM A\_HABITACION HH ,A\_TIPO\_HABITACION AH

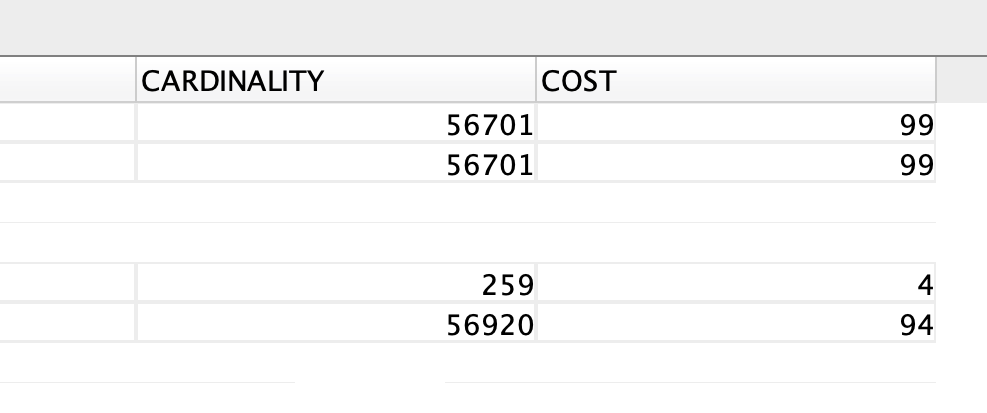
WHERE HH.TIPOHABITACION= AH.TIPOHABITACION;

**Análisis de distribución :**

Para este requerimiento nos piden dar lo índices de ocupación de cada una de las habitaciones del hotel, lo que básicamente se traduce en cuantas personas le caben a cada una de las habitaciones, es por eso que para hacer este requerimiento necesitamos tener acceso a dos tablas las cuales son de gran importancia como lo son “ a\_habitacion” y “a\_tipo\_habitacion” ,en las cuales primero vamos a saber la cantidad de habitaciones que hay, y para cada de habitación el tipo de habitación que es, puesto que tenemos 260 tipos diferentes de habitaciones, cada una con una ocupación diferente, por otro lado tenemos un total de 50.000 habitaciones distribuidas en los diferentes hoteles que tenemos, con esto dicho tenemos que primero saber el tipo de habitación que es cada una de las habitaciones de los hoteles.



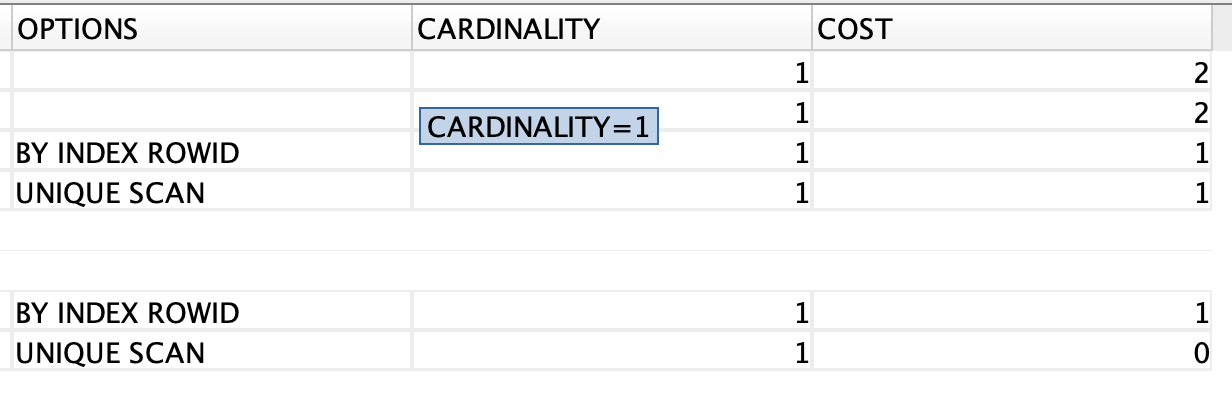




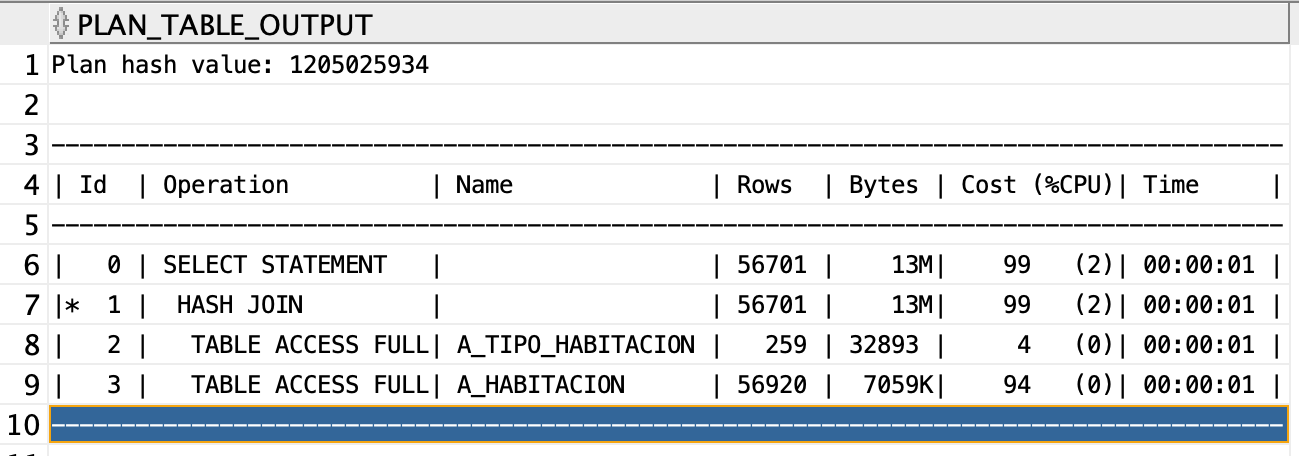
Puesto que ambas tablas tienes grandes cantidades de registros , para ser mas precisos la tabla de habitaciones es mucho mas grande, su costo de ejecución será mucho mayor debido a que el solo “select staitment” tendrá un costo superior incluso a la consulta realizada antes, debido a que las lecturas secuenciales serán mucho mayores, y no vale la pena acceder a un registro aleatorio ya que tenemos que leer todos los índices y según cada índice ir lo comparando con el tipo de habitación que se tiene.

Sin embargo esta el tamaño de respuesta cambiaria si le preguntamos por un tipo de habitación en especifico, o si simplemente le preguntamos por una habitación en especifico, es por eso que le agregamos algo adicional a la consulta la cual fue el ID= 49956, debido a que este Id representa un primary key( pk), su consto será mucho menor ya que estaríamos trabajando con índices y por lo tanto el costo seria mucho menor.



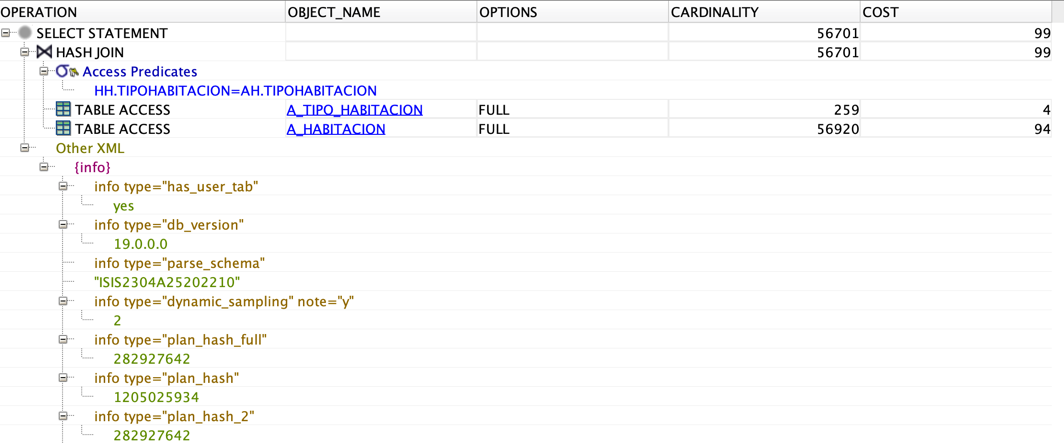


**Valores Parámetros utilizados:**



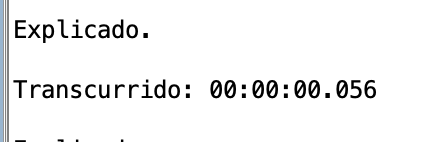
En este plan nos podemos dar cuenta de como es la distribución de los datos y de la tabla, de igual forma podemos ver porque tiene un costo significativo en el momento de que hacemos el requerimiento puesto el peso de hacer un “hash join” a pesar de ser el mejor también tiene su costo asociado, y tiene el consumo de memoria correspondiente.

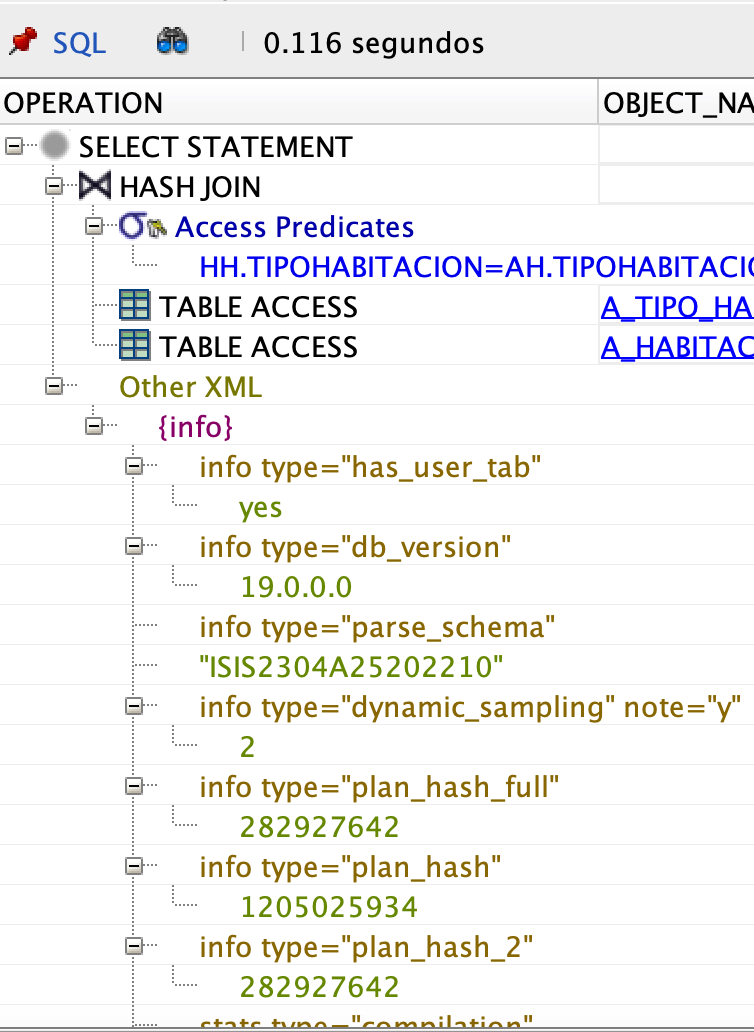
**Plan de consulta Oracle:**

****

En este plan de consulta vemos que el tipo de Join que se implemento fue un Hash join, el cual divivira cada tabla y le aplicara una función hash para después agrupar cada hash equivalente, , es importante saber que primero siempre el plan de consulta hará el select correspondiente y después cumplirá la condición estipulada en el Where.

**Tiempos obtenidos por la ejecución:**





Podemos darnos cuenta que en este requerimiento tanto el costo, como el tiempo de ejecución es mayor debido a que la cantidad de registros que usamos es mucho mayor, de la misma manera tenemos que recorrer las dos tablas y por cada registro aplicar una función Hash el cual convertirá el registro en bits y después será mucho mas fácil agrupar registros que tengan el mismo hash, sin embargo esto equivale a hacer una lectura secuencial rápida (LSR) de cada tabla ya que tenemos que usar todos los registros de ambas tablas.

**RFC4:**- MOSTRAR LOS SERVICIOS QUE CUMPLEN CON CIERTA CARACTERÍSTICA

**Sentencia SQL:**

**Característica 1**

select \*

FROM A\_SERVICIO

where COSTO BETWEEN 3000000 AND 4000000;

**Característica 2:**

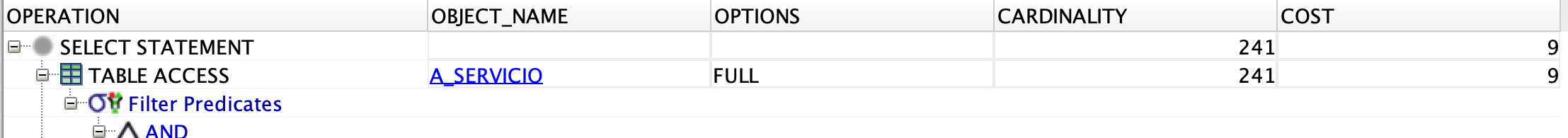
select \*

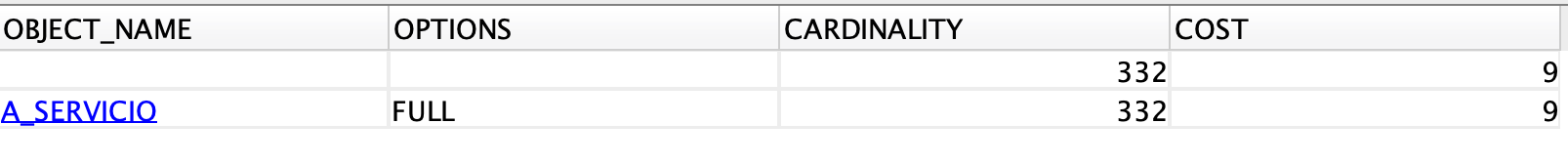
FROM A\_SERVICIO

where AREA BETWEEN 50 AND 120;

**Análisis de distribución:**

Para este análisis tenemos una tabla de servicios con 1500 registros únicos, en el momento de hacer esta consulta es menos costosa que otros requerimientos a comparación del 3 y el 2, debido a que estamos trabajando en una misma tabla, nuestro costo y nuestro tiempo no se va a ver aumentado sin embargo como esta consulta depende de la característica , se han hecho dos sentencias SQL, que representan dos tipos de características diferentes una es un servicio que cueste entre 3 millones de pesos y 4 millones de pesos. De igual manera, tenemos la otra característica que es un servicio que su área de ejecución sea entre 50 y 120 metros cuadrados. Para cada una de estas consultas sus planes de ejecución y resultados no son tan distintos, sin embargo dependiendo de la característica que se pida, puede que el tiempo se vea afectado muy mínimamente.

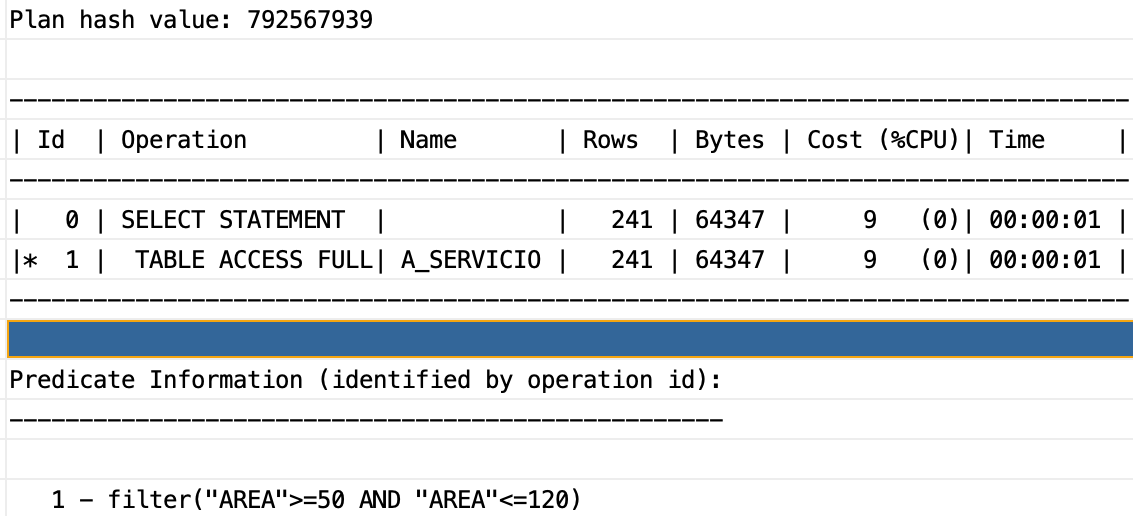
****

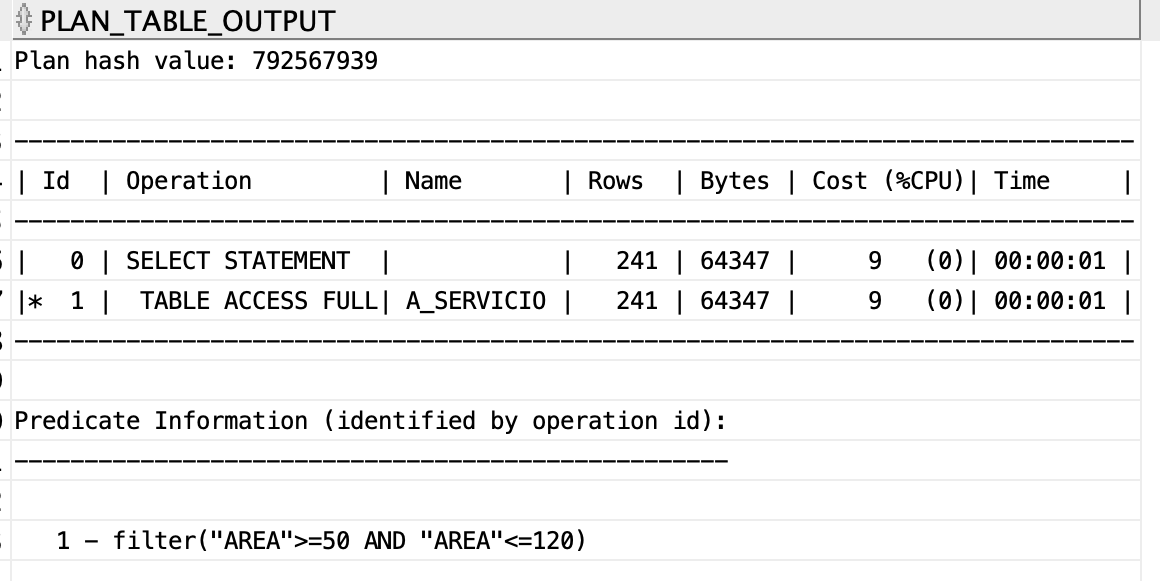


Costo 1 caracteristica Costo 2 caracteristica

Como podemos darnos cuenta, en ambas consultas se tienen que recorrer las tablas completas, sin embargo su cardinalidad cambia dependiendo lo que se le este solicitando en la consulta.

**Valores Parámetros utilizados:**

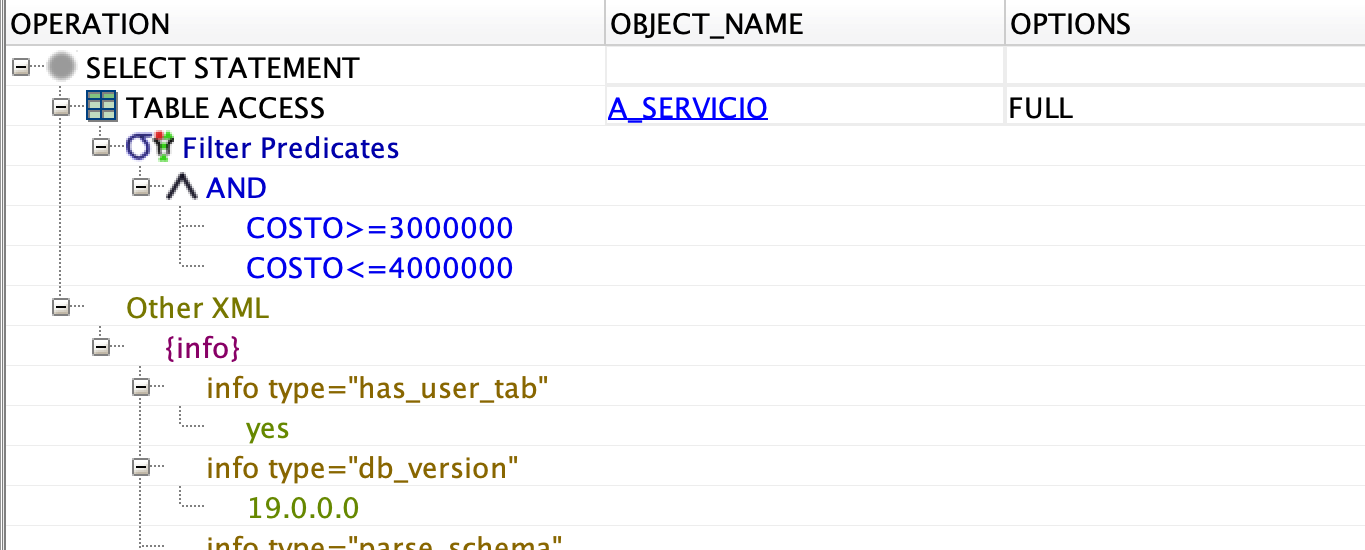


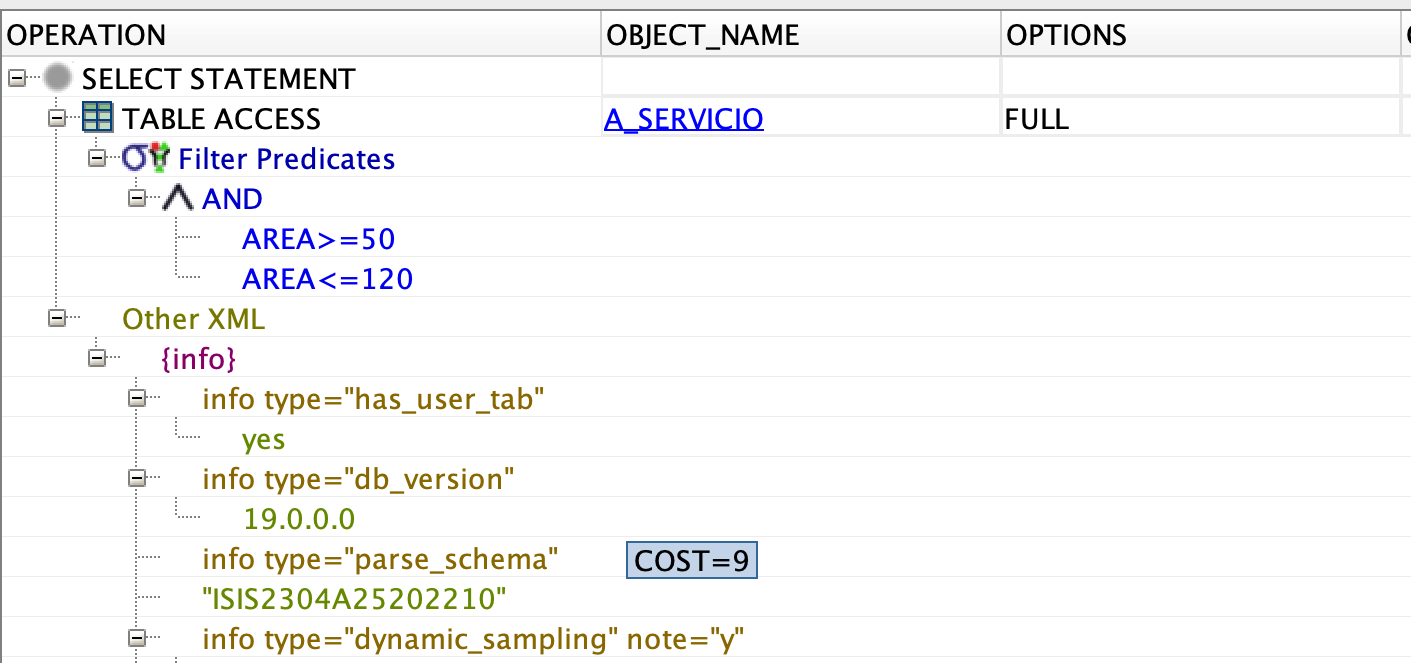
****

parametros 1 caracteristica parametros 2 caracteristica

Como podemos darnos cuenta para ambas características se tiene que recorrer la tabla completa debido a que no se están utilizando índices, es por eso que la lectura secuencial rápida, es la mas apropiada debido a que no existe algo en específico el cual nos pida buscar en esa consulta, solo nos piden por los servicios que cumplan con las características pedidas por el usuario.

**Plan de consulta Oracle:**

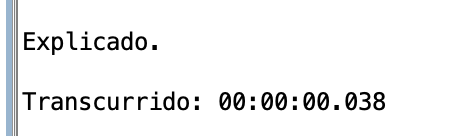


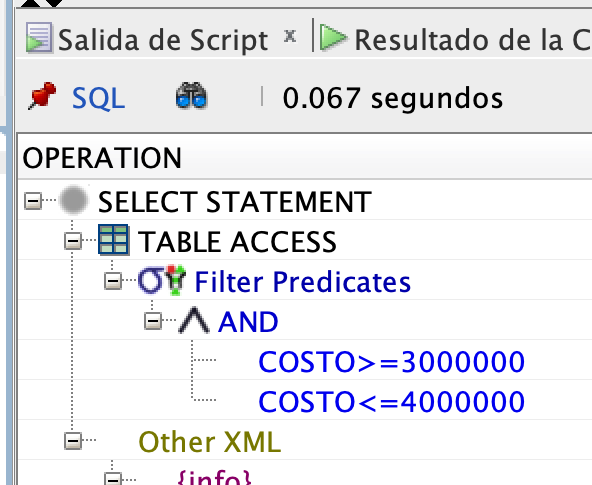


Plan ejecucion 1 caracteristica plan ejecucion 2 caracteristica

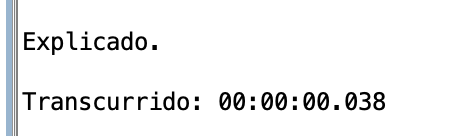
**Tiempos obtenidos por la ejecución:**

Para este requerimiento sus tiempos de respuesta son los mismo ya que siempre se va a recorrer toda la tabla, debido a que en este requerimiento no se estan usando índices ya que nos preguntan por características , lo que si se sabe es que en el plan de ejecución hace primeramente se recorre toda la tabla para que después se busca que cada registro cumpla con los aspectos y características que indica el usuario.

****

****

Consulta 1

****

Consulta 2

Como podemos cuenta los tiempos de ejecución para cada característica es la misma porque se realiza en si la misma operación que es recorrer la tabla.

**RFC5:**- RFC5 - MOSTRAR EL CONSUMO EN HOTELANDES POR UN USUARIO DADO, EN UN RANGO DE FECHAS INDICADO.

**Sentencia SQL:**

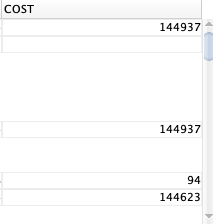
SELECT \*

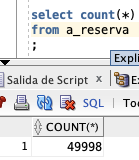
FROM a\_reserva

where idcliente=14966 and FECHASALIDA BETWEEN '11/02/12' AND '16/12/13';

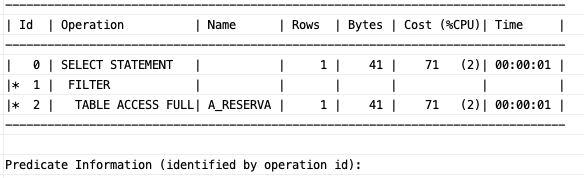
**Análisis de distribución:**

Para este análisis nos están pidiendo el consumo de un usuario en Hotel Andes según un rango de fechas, una de las ventajas que tenemos, es que cuando estuvimos creando nuestras tablas y las normalizamos, decidimos que en la tabla de reservas hubiera una columna que se llamara “id de cliente” esto debido a que las reservas son únicas y no se repiten , y al yo tener el id de una reserva puedo tener saber el id de cliente, mas bien de la persona que la reservo. Con esto mencionado, la tabla de reservas tiene una cantidad total de 50 mil registros lo que significa que tendríamos que recorrer toda la tabla. Pero, cuando en el requerimiento nos dicen “un Usuario” lo que esto significa es que vamos a tener un índice, y ese índice es el que nos permitirá recorrer de mejor manera la tabla debido a que solo estamos trabajando con una, con esto dicho hacemos una lectura aleatoria con el id del cliente, el tamaño de respuesta cambiaria si no nos piden un índice en específico o si nos piden más condiciones, ahí se vería afectado el costo final.



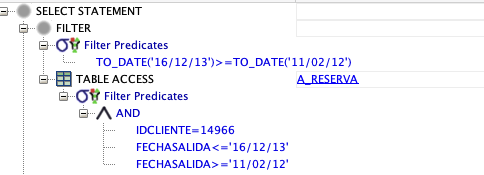


**Valores Parámetros utilizados:**

****

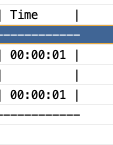
Como podemos darnos debido a que esta consulta no fue tan compleja ya que era el consumo de un usuario en un rango de fechas, lo que tuvimos que hacer simplemente era un filtro por la tabla con base al “Where” o al predicado que nos indicaba el usario que en este caso eran las fechas.

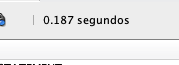
**Plan de consulta Oracle:**

****

Al tener un índice, esto nos permitirá recorrer mejor la tabla y hacer el filtro de mejor manera ya que el ID del cliente es único y no se repite por eso Oracle nos indica que tenemos que hacer un filtro con el el ID, si no estuviera el Id tendríamos que recorrer toda la tabla.

**Tiempos obtenidos por la ejecución:**

****



Como se menciono antes, el hecho de tener el pk del cliente, permite que los tiempos de ejecucion sean mucho mas rápido porque tendremos que usaremos una lectura aleatoria rápida, nuevamente porque estamos trabajando con índices, la respuesta cambiara, a la medida que el usario alargue las fechas, sin embargo no afectara en gran manera al tiempo debido a que este, solo cambiara un poco.

**RFC6:**- RFC6 – ANALIZAR LA OPERACIÓN DE HOTELANDES

**Sentencia SQL:**

**Análisis de distribución:**

**Valores Parámetros utilizados:**

**Plan de consulta Oracle:**

**Tiempos obtenidos por la ejecución:**

**RFC7:**- RFC7 - ENCONTRAR LOS BUENOS CLIENTES

**Sentencia SQL:**

SELECT us.nombre,re.idcliente,SUM (cu.total) total

FROM a\_reserva re ,a\_habitacion\_reservada hab, a\_cuenta cu,a\_usuario us

where hab.idreserva=re.idreserva and hab.idcuenta=cu.idcuenta and us.id=re.idcliente

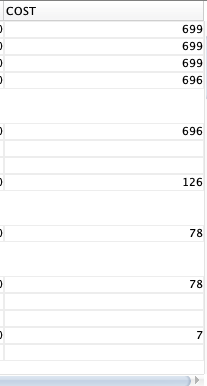
GROUP BY (us.nombre,re.idcliente)

order by total desc;

**Análisis de distribución:**

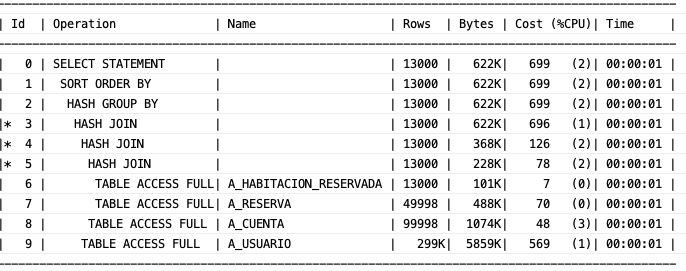
Para este requerimiento funcional no están pidiendo encontrar todos aquellos buenos clientes que han estado por vario tiempo con Hotel andes y han utilizado los servicios del mismo, con esto dicho tendemos que usar 4 diferentes tablas las cuales son: “reserva” en la cual podremos ver cuantas reservas ha tenido un cliente, “habitación resevada” en la cual sabremos con exactitud el lugar donde el usuario con su familia se hospedo, “cuenta” la cual nos servirá para saber en la habitación donde estaba hospedado cuanto dinero gasto” y por ultimo “usuario” con la que podremos saber mas información sobre el cliente, como su nombre y su apellido completo.

Para esta consulta no se tiene un índice principal con el cual se investiga, sin embargo se usan índices en el momento que trabajamos con los Joins, pues con esto mencionado, no se va a tener que recorrer toda las tablas completas si no que solo las tablas que cumplan con la condición que se dice, sin embargo este costo podría cambiar si se definirá una habitación exacta, al hacer esto los costos cambiarían y se reducirían.

****

Costo de consulta

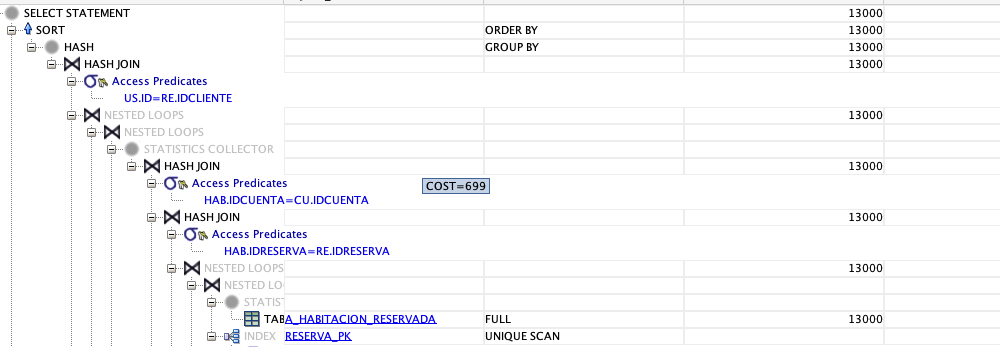
**Valores Parámetros utilizados:**

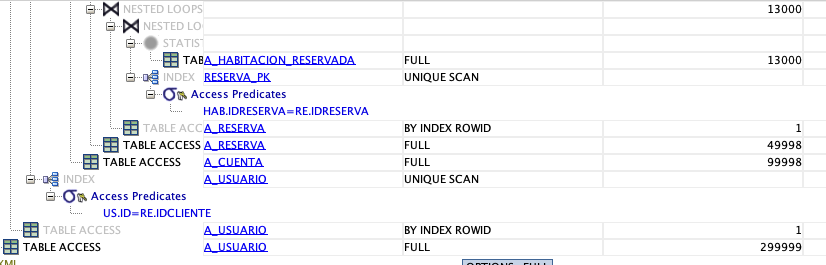
****

Una de las mayores diferencias con los demás planes de ejecución revisados anterior mente , es el tamaño y el costo que tiene que hacer cada operación y cada Join, debido a que se están manejando tablas con grandes cantidades de registros Oracle sabe utilizar la mejor manera de aprovechar toda esa cantidad de recursos, y a pesar de que se consume esa cantidad de información el plan de ejecución va encaminado en gran parte a lo que el usuario le ingresa.

**Plan de consulta Oracle:**

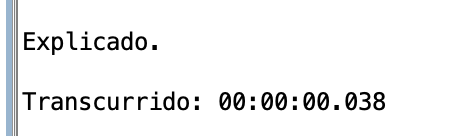
En este plan de ejecion Oracle trata diferentes maneras de resolver la consulta lo mejor posible,es por eso cuando trato de hacer los joins primero intentaba hacer un tipo de joins pero al darse cuanta que estos podrían tomar un poco mas de tiempo, este decide abortarlos y usar otros que eran de mejor forma, también podemos observar que para algunos joins, no se tienen índices por lo que toca recorrer toda la tabla buscando valores que cumplieran la condición exacta que nos decía el usuario, obviamente en términos de costo, es mucho complejo hacer este requerimiento pero al final Oracle nos muestra el plan mas preciso

****

****

**Tiempos obtenidos por la ejecución:**



****

Para analizar el tiempo de consulta nos podemos dar cuenta que Como se menciono antes, el hecho de tener el pk del cliente, permite que los tiempos de ejecución sean mucho mas rápido porque tendremos que usaremos una lectura aleatoria rápida, sin embargo al no tener una pk el costo es hacer lecturas secuenciales rápidas, en los casos cuando no tenemos índices en los joins, y lecturas alectorias en los casos donde si tenemos índices, con esto dicho esta es una de las consultas mas pesadas en cuanto a costo y en cuanto a tiempo.

**RFC8:**- ENCONTRAR LOS SERVICIOS QUE NO TIENEN MUCHA DEMANDA

**Sentencia SQL:**

SELECT cu.IDSERVICIO ,ser.fecha\_solicitud, COUNT(cu.IDSERVICIO)

FROM a\_cuenta\_cargada cu , a\_servicio\_solicitado ser

where ser.idservicio=cu.idservicio

GROUP BY cu.IDSERVICIO ,ser.fecha\_solicitud

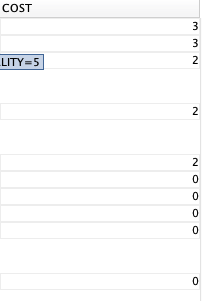
order by count(cu.IDSERVICIO) asc

FETCH NEXT 5 ROWS ONLY;

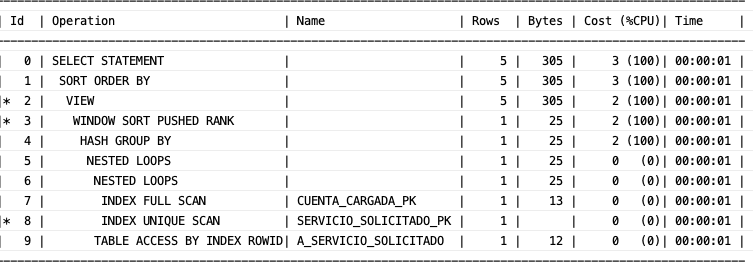
**Análisis de distribución:**

En esta consulta nos piden encontrar los servicios que no tienen mucha demanda, lo que significa que son los servicios que a la semana han sido utilizados muy poco, gracias a la estructuración y a la forma en la que armamos nuestro modelo relacional, es fácil saber las veces que un servicio ha sido solicitado, y todo es gracias a la tabla cuenta cagada podemos tener los servicios que han sido mas pedidos durante la historia y es gracias a este registro que nos podemos basar en los servicios menos usado.

En este caso al no tener un índice se tendrá que recorrer toda la tabla ya que para eso necesitamos saber cuantas veces un servicio se repite (se pide); la situación seria diferente si se estuviera preguntando por un servicio en especifico, ahí la respuesta seria mas eficiente ya que estaríamos trabajando con índices.

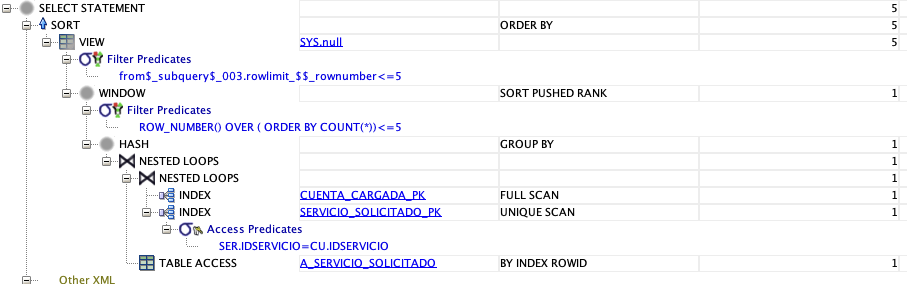
****

**Valores Parámetros utilizados:**

****

A diferencia de las demás consultas y planes, este requerimiento no ocupa tantos bytes ni trabaja con tantas filas, sin embargo al principio el costo en la CPU es muy algo pero este mismo después disminuye.

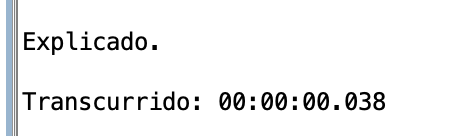
**Plan de consulta Oracle:**

****

**Tiempos obtenidos por la ejecución:**



Podemos darnos cuenta que al no haber un índice nos toca recorrer las tablas completas y adicionalmente nos toca calcular la sumatrorias de sus valores lo que significa que tenemos que hacer 4 LSR(lecturas secuencial rápida), si tuviéramos índices podríamos utilizar lectura aleatoria.

****

**RFC9:**- CONSULTARCONSUMOENHOTELANDES

**Sentencia SQL:**

-SELECT usi.id,ser.idservicio,ser.fecha\_solicitud,count(ser.idservicio)

FROM a\_servicio\_solicitado ser, a\_cuenta\_cargada cue,a\_habitacion\_reservada re, a\_reserva bas,a\_usuario usi

where ser.idservicio=cue.idservicio and cue.idcuenta=re.idcuenta and re.idreserva=bas.idreserva and bas.idcliente=usi.id

and ser.fecha\_solicitud BETWEEN '11/02/12' AND '16/12/12'

group by usi.id,ser.idservicio,ser.fecha\_solicitud

order by count(ser.idservicio)desc;

**CONSULTA POR EL CLIENTE:**

-SELECT usi.?,ser.?,ser.?count(ser.?)

FROM a\_servicio\_solicitado ser, a\_cuenta\_cargada cue,a\_habitacion\_reservada re, a\_reserva bas,a\_usuario usi

where ser.idservicio=cue.idservicio and cue.idcuenta=re.idcuenta and re.idreserva=bas.idreserva and bas.idcliente=usi.id

and ser.fecha\_solicitud BETWEEN ? AND ?

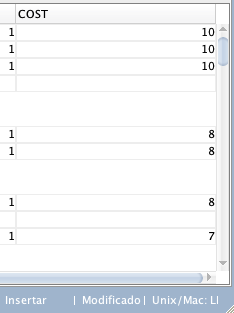
group by usi.?,ser.?,ser.f?

order by count(ser.?)desc;

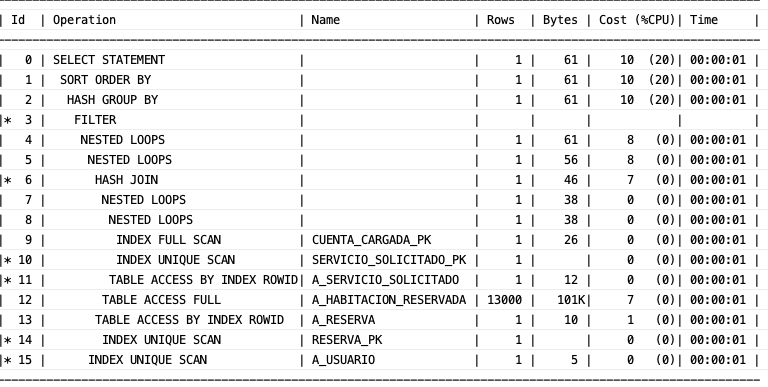
**Análisis de distribución:**

Para el caso de este análisis, es importante saber y reconocer que este es uno de los requerimientos funcionales que mas tablas involucra, lo cual puede significar un aumento de costo, depende lo que el usuario quiera ingresar, ya que en gran parte el costo de esta consulta depende de lo que el usuario inserte, si bien en el ejemplo que se tiene arriba hicimos que el usuario seleccionara el orden en como quería que estuviera la tabla y también las cosas que aparecieran, cabe decir que esa consulta no costo tanto, en relación a otras, es claro que los recorridos por la tablas fueron mas extensos, pero gracias a la ayuda de los índices, el costo no fue tan alto como se pudo llegar a pensar.

Si bien un factor determinante para esta consulta es el hecho de que tenemos un predicado en donde tenemos por medio de la fecha nos ayuda a filtrar los datos, esa fecha es la que nos ayuda a también a ahorrarnos tiempo buscando en la tabla, si bien no un índice primario ayuda mucho en términos de costos, por otra parte, la consulta hubiera sido menos costosa si solo la información por un cliente, ya que con ese índice primario mis resultados bajarían mas de lo que se muestra.

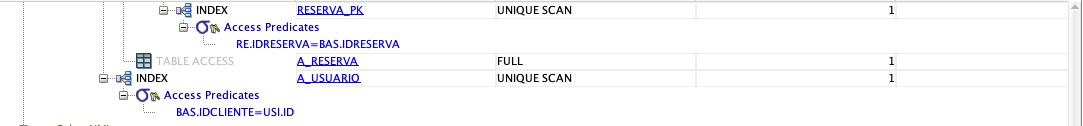
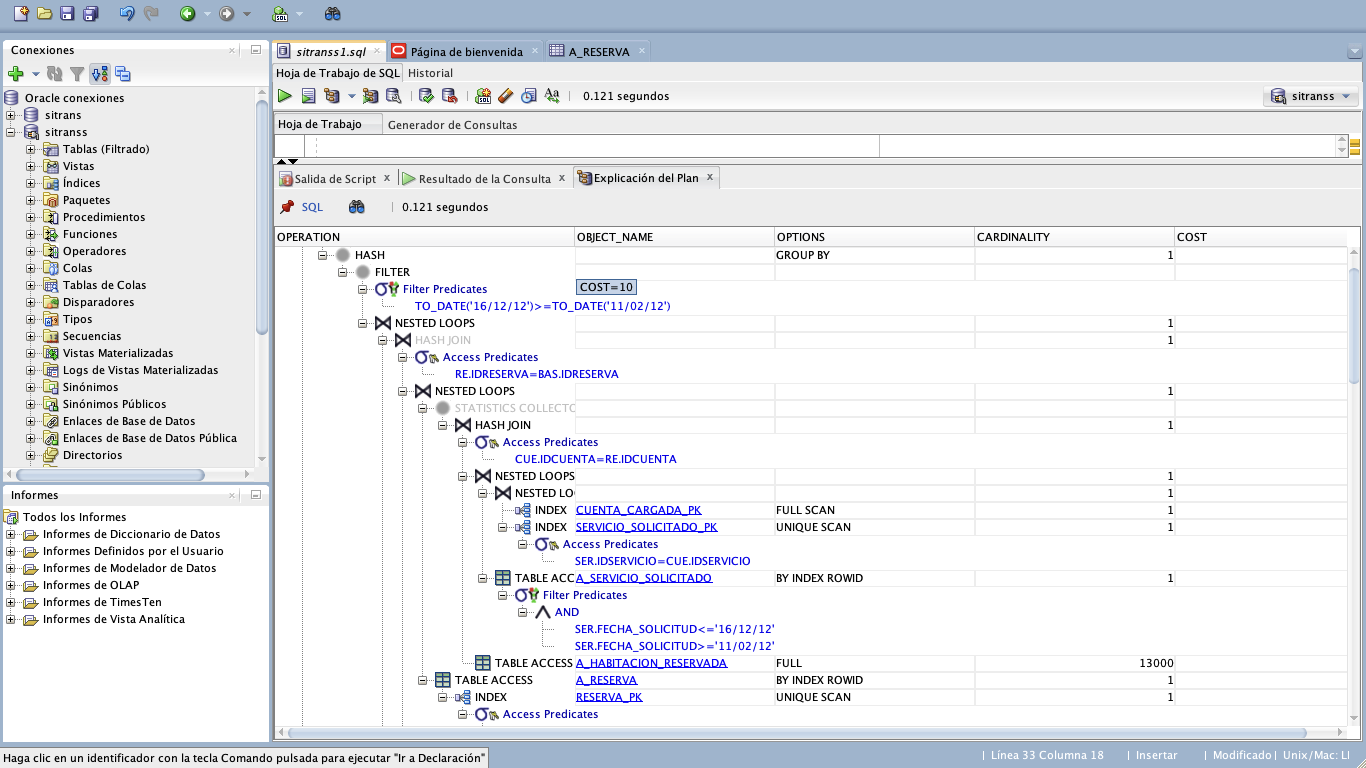
**** costo de la consulta

**Valores Parámetros utilizados:**

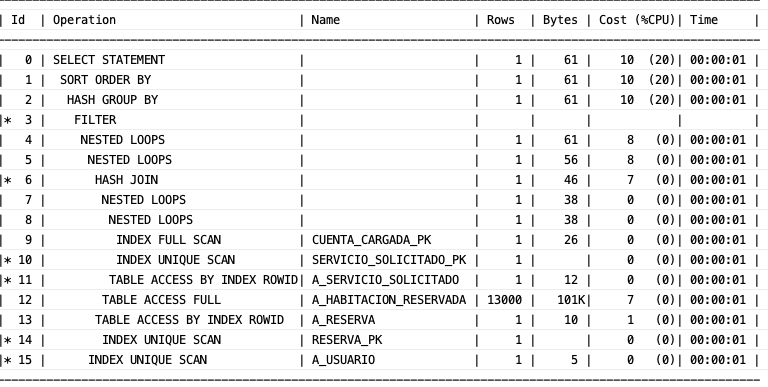
****

Como podemos darnos cuenta esta consulta no consume porcentaje como se pensaría y en gracias a ese mismo plan de Oracle que se hace de una manera rápida, de igual manera podemos ver que el hecho de trabajar con tablas muy largas, no signficaba utilizar todos los regisros de las mismas y podemos darnos cuenta de eso cuando se hacen los “nested loop” o los mismos “Hash join”.

**Plan de consulta Oracle:**

****

**Tiempos obtenidos por la ejecución:**

****



Como podemos darnos cuenta esta consulta, o requerimiento toma un buen

Considerable tiempo, se podría decir que no es una de las más largas ni

Tampoco una de las más cortas debido a que, a pesar de que en algunas

Ejecuciones se tiene que recorrer toda la tabla pero en otra usa índices, es

Una consulta que tiene buena eficiencia, al principio para recorrer la primera tabla

Se tiene que hacer un LSR(lectura secuencial rápida) ya que se tiene que recorrer la

Tabla buscando registros que cumplan con el predicado, para las demás tablas y

Para hacer los “joins” lo que se hace es partir las tablas debido a que ninguna cabe

En memoria y el mismo Oracle por tiempo indica y ejecuta las mejores operaciones que a ella le parezca

**RFC10:**- CONSULTAR CONSUMO EN HOTELANDES

**Sentencia SQL:**

SELECT usi.id,ser.idservicio,ser.fecha\_solicitud

FROM a\_servicio\_solicitado ser, a\_cuenta\_cargada cue,a\_habitacion\_reservada re, a\_reserva bas,a\_usuario usi

where ser.idservicio=cue.idservicio and cue.idcuenta=re.idcuenta and re.idreserva=bas.idreserva and bas.idcliente=usi.id

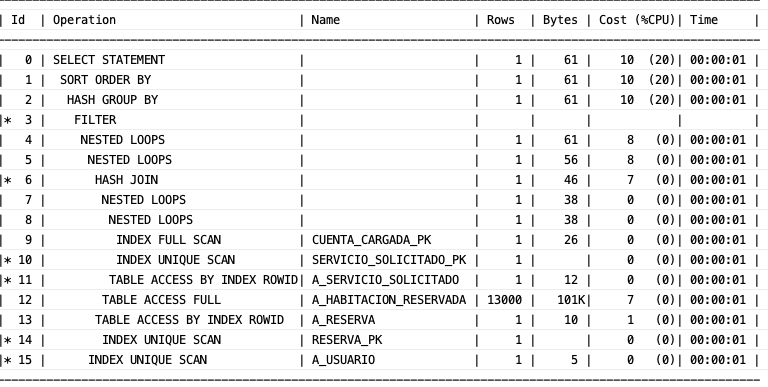
and ser.fecha\_solicitud BETWEEN '11/02/12' AND '16/12/12'

group by usi.id,ser.idservicio,ser.fecha\_solicitud;

**Análisis de distribución:**

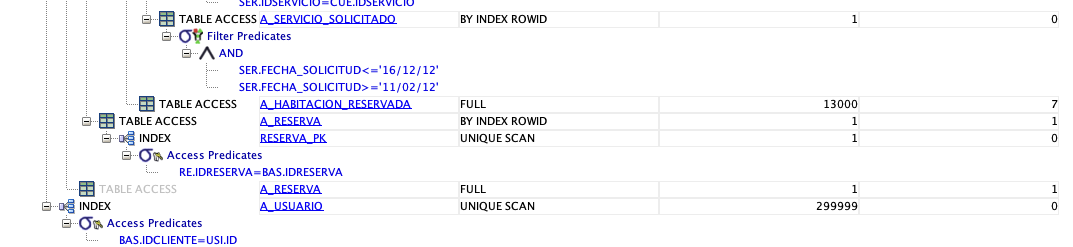
Para el caso de este análisis, es importante saber y reconocer que este es uno de los requerimientos funcionales que mas tablas involucra, lo cual puede significar un aumento de costo, depende lo que el usuario quiera ingresar, ya que en gran parte el costo de esta consulta depende de lo que el usuario inserte, si bien en el ejemplo que se tiene arriba hicimos que el usuario seleccionara el orden en como quería que estuviera la tabla y también las cosas que aparecieran, cabe decir que esa consulta no costo tanto, en relación a otras, es claro que los recorridos por la tablas fueron mas extensos, pero gracias a la ayuda de los índices, el costo no fue tan alto como se pudo llegar a pensar.

Si bien un factor determinante para esta consulta es el hecho de que tenemos un predicado en donde tenemos por medio de la fecha nos ayuda a filtrar los datos, esa fecha es la que nos ayuda a también a ahorrarnos tiempo buscando en la tabla, si bien no un índice primario ayuda mucho en términos de costos, por otra parte, la consulta hubiera sido menos costosa si solo la información por un cliente, ya que con ese índice primario mis resultados bajarían mas de lo que se muestra.

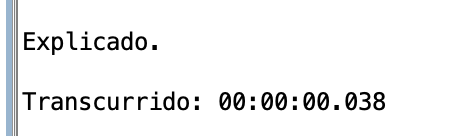
**Valores Parámetros utilizados: **

Como podemos darnos cuenta esta consulta no consume porcentaje como se pensaría y en gracias a ese mismo plan de Oracle que se hace de una manera rápida, de igual manera podemos ver que el hecho de trabajar con tablas muy largas, no signficaba utilizar todos los regisros de las mismas y podemos darnos cuenta de eso cuando se hacen los “nested loop” o los mismos “Hash join”.

**Plan de consulta Oracle:**

****

**Tiempos obtenidos por la ejecución:**

****



Como podemos darnos cuenta esta consulta, o requerimiento toma un buen

Considerable tiempo, se podría decir que no es una de las más largas ni

Tampoco una de las más cortas debido a que, a pesar de que en algunas

Ejecuciones se tiene que recorrer toda la tabla pero en otra usa índices, es

Una consulta que tiene buena eficiencia, al principio para recorrer la primera tabla

Se tiene que hacer un LSR(lectura secuencial rápida) ya que se tiene que recorrer la

Tabla buscando registros que cumplan con el predicado, para las demás tablas y

Para hacer los “joins” lo que se hace es partir las tablas debido a que ninguna cabe

En memoria y el mismo Oracle por tiempo indica y ejecuta las mejores operaciones que a ella le parezca

**RFC11:**- CONSULTAR FUNCIONAMIENTO

**Sentencia SQL:**

SELECT idServicio, MAX(cont)

FROM (SELECT idServicio, COUNT(idServicio) cont

FROM A\_SERVICIO

GROUP BY idServicio) contador

GROUP BY idServicio;

SELECT idServicio, MIN(cont)

FROM (SELECT idServicio, COUNT(idServicio) cont

FROM A\_SERVICIO

GROUP BY idServicio) contador

GROUP BY idServicio;

SELECT idHabitacion, MAX(cont)

FROM (SELECT idHabitacion, COUNT(idHabitacion) cont

FROM A\_HABITACION\_RESERVADA

GROUP BY idHabitacion) contador

GROUP BY idHabitacion;

SELECT idHabitacion, MIN(cont)

FROM (SELECT idHabitacion, COUNT(idHabitacion) cont

FROM A\_HABITACION\_RESERVADA

GROUP BY idHabitacion) contador

GROUP BY idHabitacion;

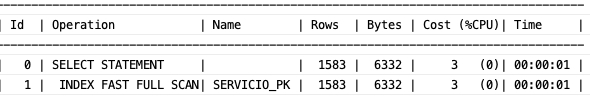
**Análisis de distribución:**

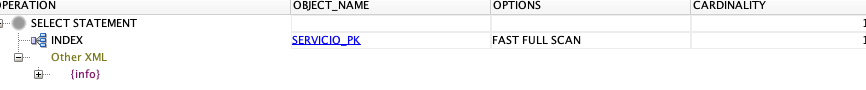
Para este análisis se nos piden hacer varias consultas para saber información sobre de las habitaciones mas reservadas y de los servicios mas usados, el hecho de que no se usaran un pk, o un índice en ninguna de estas consultas nos obligaba que se recorriera toda la tabla por lo cual implicaba buscar en toda la información la habitación que fuera mas visitada que la otra, una ventaja que teníamos es que gracias al modelo relacional que implementamos los costos que tuvimos que calcular no fueron tan elevados a comparacion de otros, ya que tuvimos unas tablas donde siempre aparecían las veces que se pedían un servicio o las veces que este era utilizado, la verdad con un indice quisas el costo seria mas pequeño , pero al ser esta unas simples lecturas secuenciales aleatorias, no fue tan grande el costo de consumo de cpu.

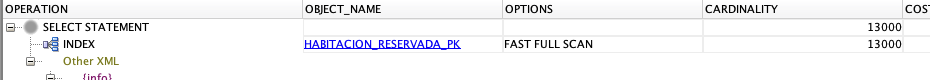
**Valores Parámetros utilizados:**

****

****

****

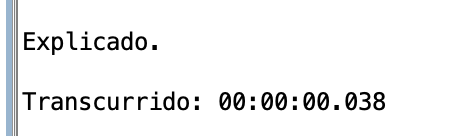
**Plan de consulta Oracle: **

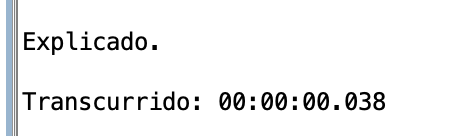
****

****

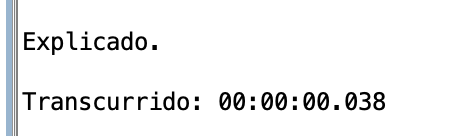
**Tiempos obtenidos por la ejecución:**

Para este requerimiento sus tiempos de respuesta son los mismo ya que siempre se va a recorrer toda la tabla, debido a que en este requerimiento no se estan usando índices ya que nos preguntan por características , lo que si se sabe es que en el plan de ejecución hace primeramente se recorre toda la tabla para que después se busca que cada registro cumpla con los aspectos y características que indica el usuario.

****

****

Consulta 1 Consulta 2

****

Como podemos cuenta los tiempos de ejecución para cada característica es la misma porque se realiza en si la misma operación que es recorrer la tabla.

**RFC12:**- CONSULTAR LOS BUENOS CLIENTES

**Sentencia SQL:**

SELECT userr.nombre,userr.apellido,userr.correo,re.fechaentrada,re.fechasalida

FROM a\_reserva re, a\_habitacion\_reservada hab, A\_CUENTA\_CARGADA cuu, a\_servicio ser,a\_usuario userr

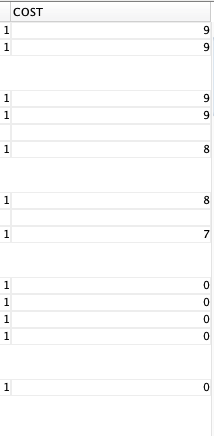
where re.idreserva=hab.idreserva and hab.idcuenta=cuu.idcuenta and cuu.idservicio=ser.idservicio and ser.costo >300000

and ser.nombre LIKE'spa%' and userr.id=re.idcliente ;

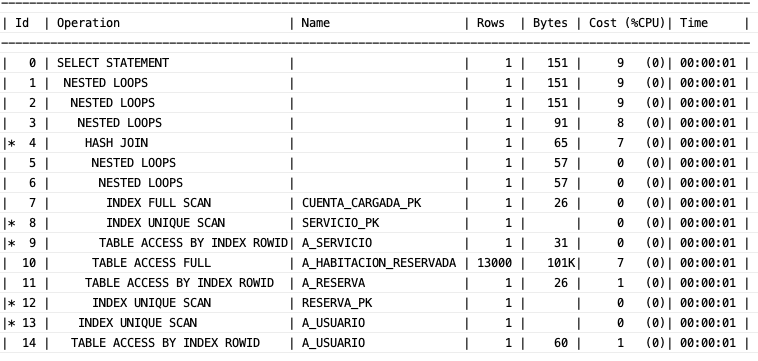
**Análisis de distribución:**

En esta consulta nos están están preguntando por aquellos clientes que, frecuentan mucho los hoteles y que usan servicios determinados como lo es el Spa o los salones de reuniones, debido a la forma que tenemos nuestro modelo relacional esta consulta implico usar muchas tablas debido a que lo que querían saber era la información exacta sobre de los clientes, es por eso que nos toco hacer muchos “Joins” algunos que toman mas tiempo que otros, pero que al final y al cabo ayudan a que la consulta sea tanto eficiente como rápida, si bien al haber recorrido tablas como “habitación reservada” nos toco hacer un full Access por ya que si bien la restricción pedía recorrer toda la tabla o en algunos casos no se contaban con índices.

El tamaño de la repuesta hubiera sido diferente si se hubiera utilizado un índice como el pk de un cliente o el pk de otra tabla, quizás ahí el costo hubiera cambiado, es preciso decir que en términos de costo, no fue tan costosa como se espero o se pensó, pero si fue una consulta compleja, pero que gracias a Oracle se realizo de la mejor manera.

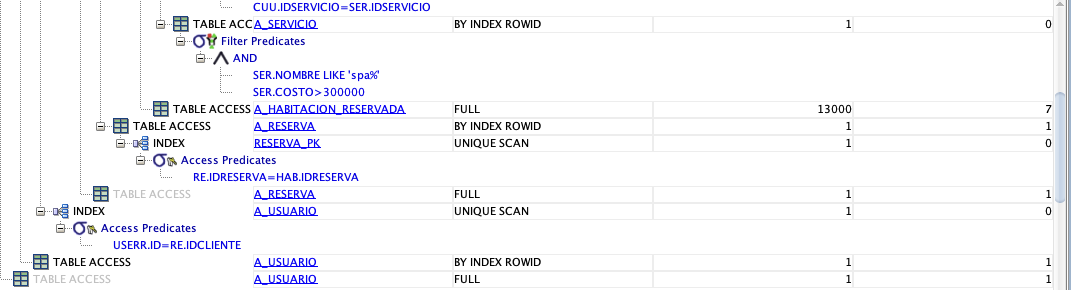
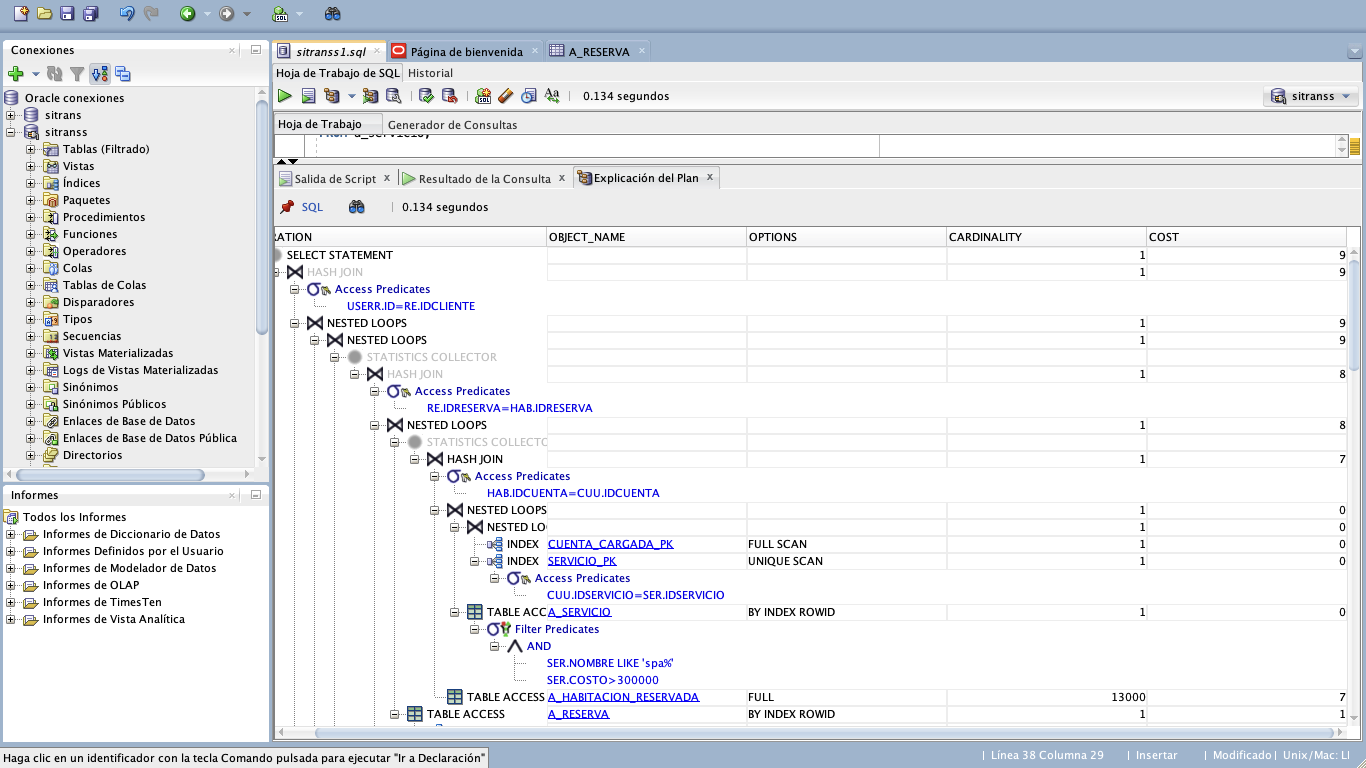
Costo de hacer la consulta

**Valores Parámetros utilizados:**

****

Nos podemos dar cuenta que este requerimiento a pesar de que manejan muchas tablas, no todas de ellas son recorridas completamente, y esto se debe a los índices que se usan para hacer los joins como “id usuario” o “id reserva” que al final nos ayudan a que no se tenga que recorrer toda la tabla, sin embargo hay tablas que se tienen que recorrer completas ya que se necesita saber información acorde con el predicado, en % de CPU no afecta, pero en cuanto a costo, puede a afectar, no en una manera significativa pero puede aumentar

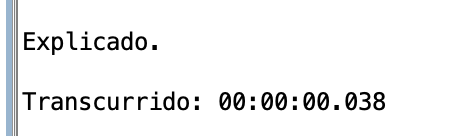
**Plan de consulta Oracle:**

****

**Tiempos obtenidos por la ejecución:**

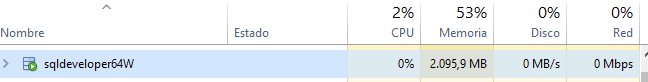




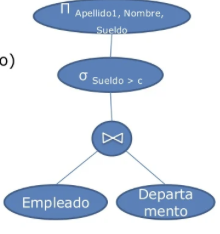
****

Como se menciono antes, el hecho de tener el pk del cliente, permite que los tiempos de ejecucion sean mucho mas rápido porque tendremos que usaremos una lectura aleatoria rápida, nuevamente porque estamos trabajando con índices, la respuesta cambiara, a la medida que el usario alargue las fechas, sin embargo no afectara en gran manera al tiempo debido a que este, solo cambiara un poco.

Análisis de eficiencia



Nos pudimos dar cuenta que para todas las consultas hechas tanto para Oracle como para uno Oracle simpre pudo hacerlas y responderlas de la mejor manera y evidencia de esto es cuando vemos cada RFC(requerimiento funcional consulta), en donde apesra de que uno tiene un plan el cual puede que sea muy bueno Oracle utiliza uno mejor ya que el piensa en relación a costo tiempo , y nosotros pensamos en relacion lógica y coherencia, debido a esto, los planes de ejecucion propuestos por nostros a pesar de ser acordes y correctos no se pueden comprar a lo que oracle realiza.



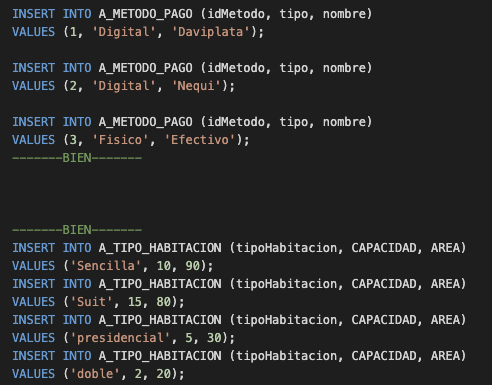
A\_RESERVA.costo >= 2000000

Usuario

A\_RESERVA

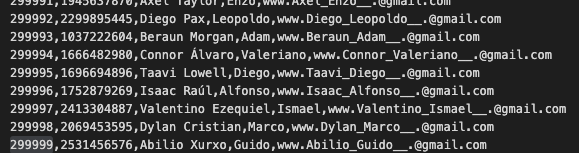
**CONSTRUCCION DE LA APLICACIÓN, EJECUCION DE PRUEBAS Y ANALISIS DE RESULTADOS**

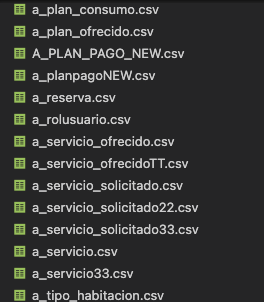
Para la construcción de la aplicación usamos las misma tablas de el caso 3 debido, debido a que en ese mismo caso ya habíamos actualizado nuestras tablas y habíamos hecho una pequeñas pruebas con datos que no pasaban de 10 registros por tabla, al hacer esas pequeñas pruebas nos dimos cuenta que para cargar un datos sobre una tablas estás tienen que tener un orden y no pueden ser cargadas como uno quiere, razón por la cual al principio se tuvo inconvenientes en la inserción de las mismas.



primeras pruebas con pocos datos

Al darnos cuenta que para hacer esa inserción de datos estas tenían que tener un orden decidimos crear un plan, un plan lógico que se acomodara con el caso de hotel andes en donde siempre deberían haber mas clientes que hoteles y donde la información fuera acorde a lo que se manejaría, decidimos crear archivos CSV , en base a listas de Python creadas por nosotros, la ventaja que tenia esto era que era fácil crear las listas y después importarlas. Para el caso de la creación del números de registros, en realidad fue muy sencillo debido a que se utilizaba combinaciones de palabras unidas a otra palabras ya existentes para tener según lo que uno quisiera la cantidad de registros. Cada CSV que creamos lo separamos por comas para que fuera mas fácil y eficiente de leer





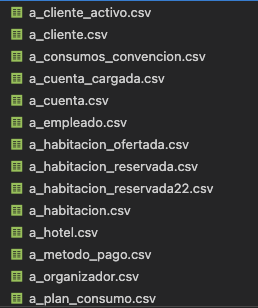


Tabla registros por tabla

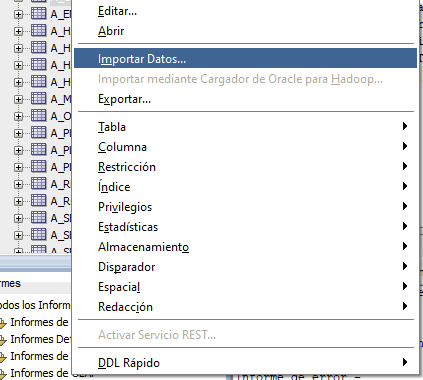
|  |  |
| --- | --- |
| **TABLA** | **# REGISTROS** |
| cliente\_activo | 49999 |
| cliente | 99999 |
| consumos convencion | 5000 |
| cuenta cargada | 4000 |
| cuenta | 99999 |
| empleado | 100000 |
| habitacion ofertada | 50000 |
| habitacion reservada | 14999 |
| habitacion | 49999 |
| hotel | 10000 |
| metodo de pago | 100000 |
| organizador | 50000 |
| plan consumo | 5000 |
| plan ofrecido | 49994 |
| reserva | 50000 |
| rol usuario | 300000 |
| servicio ofrecido | 2499800 |
| servicio solicitado | 1582 |
| servicio | 1582 |
| tipo habitacion | 260 |
| tipo reserva | 25000 |
| usuario | 299999 |
| **TOTAL** | 3.867.212 |

Evidentemente al final el total de registros fue mas que el esperado y el pedido, sin embargo, eso no impidio que los datos no se pudieran cargar, sin embargo fue lento en varias ocaciones, un de las cosas que nos dimos cuenta es que en el principio quicimos hacer muchos hoteles muchos planes de pago, pero no calculamos que por cada usario o por cada hotel las habitaciones tenian que tener un regitro mayor para que estas pudieran ser distribuidas equitativamente por todos los hoteles, un caso similar paso con usuarios, creamos 300 mil usuarios, y no todos los usuarios tienen solo una reserva. Es por eso que luego de darnos cuenta de ese error, por medio de la funcion “.random” de phyton no neceseriamente un usuario tenia una sola reserva, si no que ya con esta funcion podia que un usuario tuviera una o muchas reservas, como que tampoco tuviera ninguna.

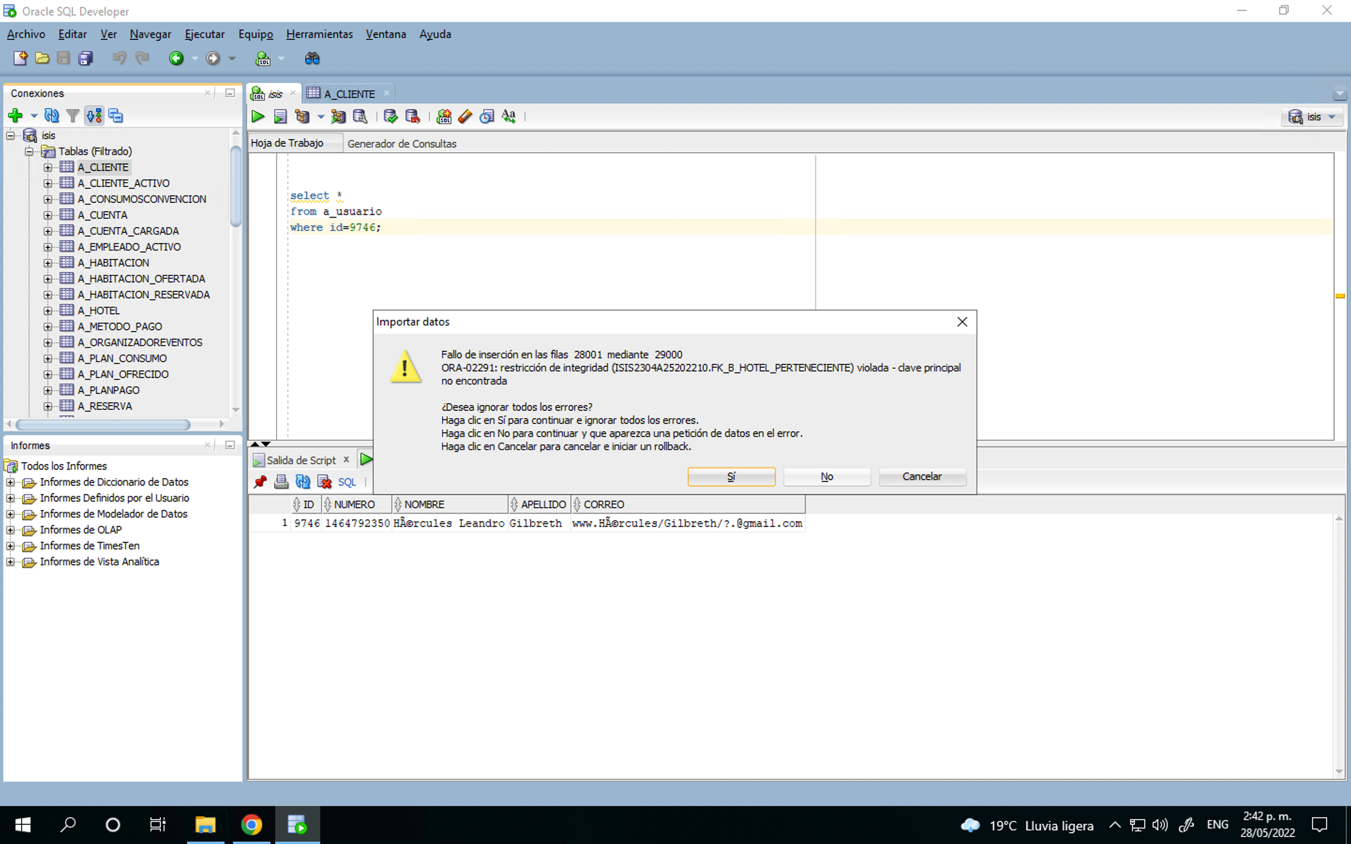
Finalmente, la creacion de tablas que tienen FK fue dificil crearlas, ya que con respecto a las tablas normales que habiamos creado, teniamos que importar la informacion. Al principio fue un poco complejo porque muchas veces copiaba toda la informacion de listas creada mas de 4 veces en diferentes archivos, hasta que nos dimos cuenta que si simplemente llamabamos la tabla que ya teniamos esta iba a funcionar, al final pudimos crear todas la tablas en el orden previamente aprendido.

**Proceso de carga de Datos**

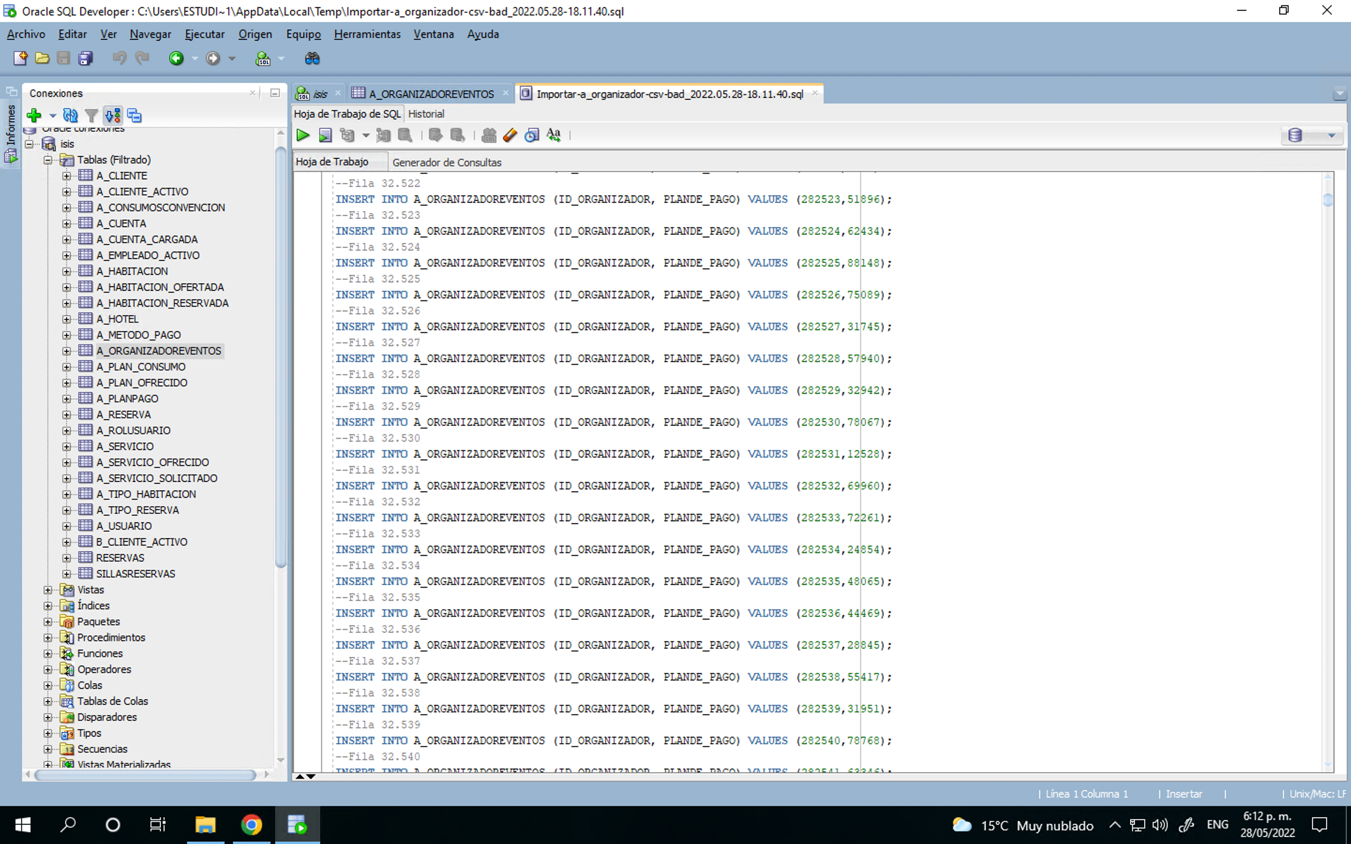
Una vez creadas y cargadas mis tablas, era el momento de cargar los datos, luego de haber aprendido que la carga de datos tiene un proceso por medio de la importacion de datos en oracle fue muy sencillo exportar la tablas.



Al principio la carga de todas las tablas que no tenían Fk fueron muy rápidas, en cuestión de segundos las tablas ya estaban subidas, sin embargo, en el momento que íbamos a subir las demás tablas para conectar con las tablas ya subidas, fue cuando empezaron a haber problemas. Por un lado salia un error que decía que “FK\_NO\_ENCONTRAD” lo cual significaba que las tablas que había subido, tenían un problema en sus PK, lo cual fue muy complejo de entender. Por más que se borraran y se insertaran las tablas nuevamente este error era persistente.



Este error genero que volviera a crear nuevamente los CSV, ya que el error había quedado ahí y, no importaba que uno ignorara los errores al final para hacer las consultas fue muy difícil porque los joins no se podían realizar de manera correcta ya que, al no encontrar una relación con otra tabla, pues simplemente no iba a poder unir dos tablas y adicionalmente a eso, mis tablas no quedaban del tamaño esperado, si no que quedaban de un tamaño mucho menor.



Luego de haber reparado el CSV la carga de la tablas fuera considerablemente mas rápidas que cuando se intentaban subir con errores, sin embargo, el hecho de que muchas personas estaban subiendo y actualizando sus tablas significo un aumento en el tiempo, que luego se vio afectado en tiempo de trabajo.

**Análisis del proceso de optimización y el modelo de ejecución de consultas.**

En este caso, en el momento que se hace una consulta delegada al manejador de bases de datos usualmente lo que estamos haciendo es queriendo tener un resultado en base a uno como conoce la tabla y los datos obviamente, esta consulta siempre va a responder de la mejor manera, ya que esta siempre quiera responder de la manera mas eficiente posible, si bien Oracle es un sistema autónomo que toma las mejores decisiones para dar respuesta a lo que uno pide, muchas veces esa toma decisiones esta ligada a como la base de datos, entonces al tener una base de datos mala, no solo por falta de normalización se vera afectado la base de datos, si no que el mismo Oracle aun con eso intentara de buscar la mejor forma de responder a esto.

En relación a los “Joins” y a las consultas, dependiendo de lo que el usuario solicite el mismo sistema es auto eficiente y sabe cuanto pesa, una tabla y sabe si vale la pena usar un “hash join” un “one pass join” o un “two and half pass join”, si bien el join mas frecuente utilizado en este caso fue el “el hash join” y el “indexjoin” nos pudimos dar cuenta que fueron los mas efectivos en relación a costo tiempo. Si bien muchas veces el usuario le indica aveces a Oracle el tipo de Join, usar con las condiciones que tiene que tener ese join o con las condiciones, muchas veces no es eficiente, a menos que uno conozca mucho la base de datos y conozca muy bien de algorítmica de base de datos, se podrá hacer unas consultas muy eficientes, pero si no es el caso es mejor que Oracle se encargue de ello porque o si no, las consultas que uno hace pueden consumir memoria y pueden demorarse mucho tiempo, lo cual para una operación en la vida real no será muy efectivo.