# Documentación Diseño de la aplicación:

## diseño físico y escenario de pruebas:

**índices secundarios:**

estos son los índices que consideramos necesarios para realizar todos los requerimientos solicitados a lo largo del proyecto, y otros que podrían considerarse a futuro.

|  |  |
| --- | --- |
| tabla | Índices secundarios |
| mesa | RESERVAS |
| ORDEN | FECHA,ID\_USUARIO |
| CUENTA\_RESTAURANTE | RESTAURANTE\_ID |
| ITEMS | ID\_RESTAURANTE |
| MENU | RESTAURANTE\_ID |
| PERSONA | NOMBRE |
| RESERVA | FECHA,PERSONA\_ID,ZONAS |
| ZONAS | AMBIENTE |

**Construcción de Datos**

Para la iteración se pulularon las tablas con datos creados por medio de una aplicación desarrollada por el grupo donde por medio de uno 100 nombres y 100 apellidos este generaba los datos mezclando los diferentes valores para general aproximadamente mas de 1 millón de usuario, los ítems se crearon de igual manera que los usuarios y se ingresaron en promedio uno 100 mil ítems, la distribución de los usuarios es muy parecida ya que hay una gran cantidad que comienzan con los mismos nombres pero cambian de apellido, facilitando la búsqueda por index, en el caso de los ítems esto no es tan cierto ya que al distribución de los nombres no es parecido a este, frente a las órdenes y los ítems de esta se generaron de forma que se agregara creara ordenes aleatoriamente a los restaurantes existente y ítems a estas órdenes, lo que dio como resultado que algunas ordenes tuvieran muchos ítems y otros no, en este caso no se bio afectado las búsquedas de restaurantes porque en nuestro proyecto se maneja una ordena un restaurante por su ítems y si una orden tiene un ítem de un restaurante se asume que la orden es de ese restaurante y a su vez una orden puede tener muchos restaurantes.

**Ajustes de la Aplicación**

Con lo que respecta al modelo de la aplicación no se cambio ninguna tabla ya que se considero que muchos de los campos y estructuras que se venia trabajando era suficientes para cumplir con los requerimientos presentado en esta iteración, frente al rest se agregaron las funcionalidades a administradores y a personas por igual, frente a la transitividad de cada requerimiento atenrio desde la iteración pasada se habían ajustado aspectos de aislamiento de las operaciones y en esta se termino algunas clases que les faltaban su aislamiento y su respectivas recuperaciones frente a fallas.

**Analiza de optimización**

La diferencia entre delegar la ejecución consultas al manejador de base de datos, frente a realizarlo desde la lógica es el tiempo de ejecución del manejador es mucho mas optimo que si uno lo realizara. El primer punto que se puede ver es el volumen de datos que cada uno puede manejar, ya que mientras el manejador de datos es capas de manejar volúmenes muy grandes fácilmente sin necesidad de muchas lecturas al disco, al manejar una carga masiva de datos la cual no cabe en memoria principal la lógica de una aplicación tendría que hacer muchos ciclos en un for o while y aparte de esto debe realizar muchas lecturas y escrituras a disco lo cual hace el proceso demasiado lento aun para encontrar un solo dato o un rango de datos.

Un segundo punto es la ventaja de los index dentro de la misma base de datos, gracias a esto el manejador puede realizar búsquedas binarias en poco tiempo y devolver el resultado de unas cuantas duplas, mientras que si se quiere realizar una de estas búsquedas en lógica dentro de una aplicación este tendría que crear un árbol primaria cada vez que se consulte y solo si este árbol cabe en memoria lo cual seria una búsqueda totalmente lenta.

Y por ultimo los mas importante al delegar estas consultas al manejador lo que obtienen la aplicación son pocas duplas o una cantidad manejable de duplas para que sean enviadas al usuario, si estas consultas y búsquedas no fueran filtradas antes de llegar a la aplicaciones esta tardaría un tiempo excesivo en procesarlas y presentarlas al cliente.

**RCF9:**

En este requerimiento nos solicitan la información de los usuarios que consumieron al menos un producto de un restaurante determinado en un RANGO de fechas.

Como se puede ver en la tabla los índices secundarios creados por nosotros para este requerimiento fueron FEHCA y ID\_USUARIO, además de los índices primarios que se usan como PRIMARY KEY. Tomamos esta decisión de diseño ya que al solicitar las consultas en intervalo de tiempo (fechas), Oracle creara un árbol B+ con el índice de fecha así minimizando el tiempo de recorrido y de I/O. Además, se utilizaron 4 tablas diferentes ítems, Personas, orden\_items y orden.

**Primera parte**

El primer requerimiento de este punto requería realizar una consulta en un rango de fecha determinado y encontrar los pedidos de un restaurante requerido. La sentencia Sql que se uso para la consulta es la siguiente.

SQL:

*SELECT PERSONA.USUARIO\_ID AS ID , ROL,NOMBRE,NOMBRE\_PRODUC,FECHA*

*FROM(select ORDEN.ID\_PERSONA AS ID,ORDEN.FECHA AS FECHA ,ITEMS.NOMBRE AS NOMBRE\_PRODUC*

*from( ORDEN JOIN ITEMS\_ORDEN*

*ON ID = ITEMS\_ORDEN.ORDEN\_ID)JOIN ITEMS*

*ON ITEMS\_ORDEN.ITEMS\_ID = ITEMS.ID*

*WHERE ITEMS.ID\_RESTAURANTE = 916*

*AND FECHA BETWEEN '4-10-2012' AND '11-10-2017') JOIN PERSONA*

*ON PERSONA.USUARIO\_ID = ID*

*ORDER BY FECHA ASC*

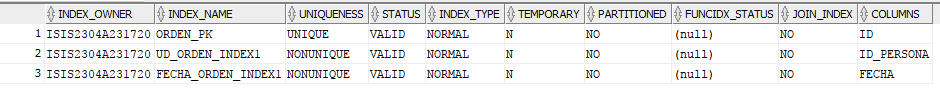
El requerimiento adicional de este punto requiere que el usuario decidiera un filtro de información, este filtro se aplica en la lógica de la aplicación, para el caso de las pruebas y documentación de este documento se usara el ordenamiento de fechas ascendentemente, en esta sentencia el usuario consultante es un administrador y se espera que este puede ver todos los pedidos de todos los usuarios de la aplicación.

**Index:**

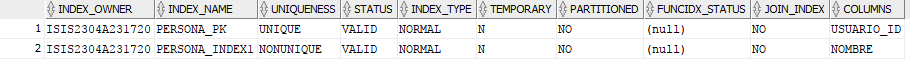
Las tablas involucradas en esta consulta son PERSONA, ITEMS, ITEMS\_ORDEN, ORDEN. En la tabla persona se implementó el index que crea Oracle el cual es USUARIO\_ID, la justificación es que esto permite encontrar rápidamente en un árbol, un usuario por su id, el otro index ingresado por nosotros es un index no único ordenado ascendentemente, esto no solo disminuye el tiempo de búsqueda por nombre, sino facilita el ordenamiento y la búsqueda de rangos por nombre.

La siguientes tablas es la tabla ITEMS donde nuevamente se utiliza el index de Oracle en el pk de id para agilizar búsquedas por Pk, también se ingreso un index el ID\_RESTAURANTE esto con el fin de hacer una búsqueda por restaurante y facilitar los joins en el requerimiento de este punto y en nombre para facilitar el join y el ordenamiento por nombre de este requerimiento, por ultimo en las tablas que se crearon index fueron ordenes, la primera es el pk echo por Oracles, el segundo index creado es usuario orden, este index se aplica para facilitar no solo la búsqueda por usuario sino el join entre esto y por último se usó un index en fecha en un árbol, esto para facilitar y reducir costos de la búsqueda de rangos de fechas de este requerimiento.

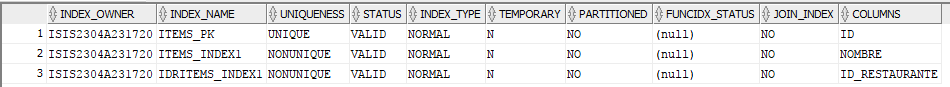
**INDEX ORDEN.**



**INDEX PERSONA**

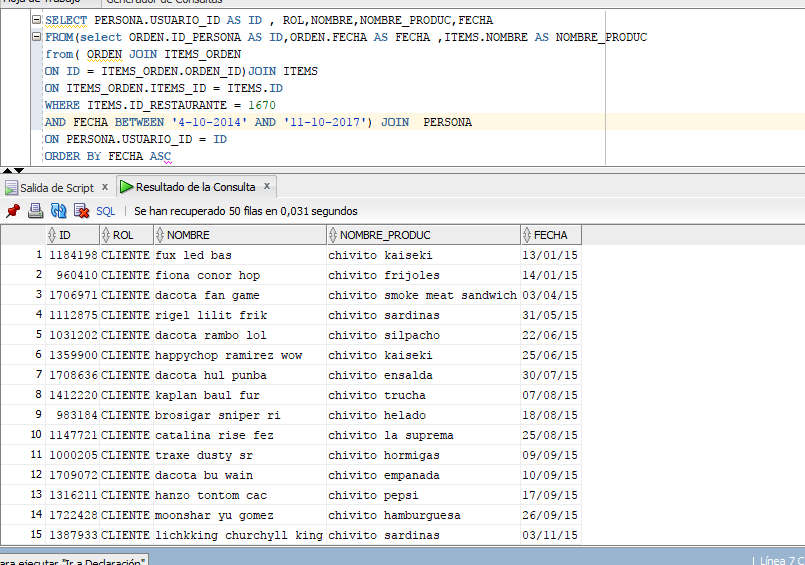


**INDEX ITEMS**



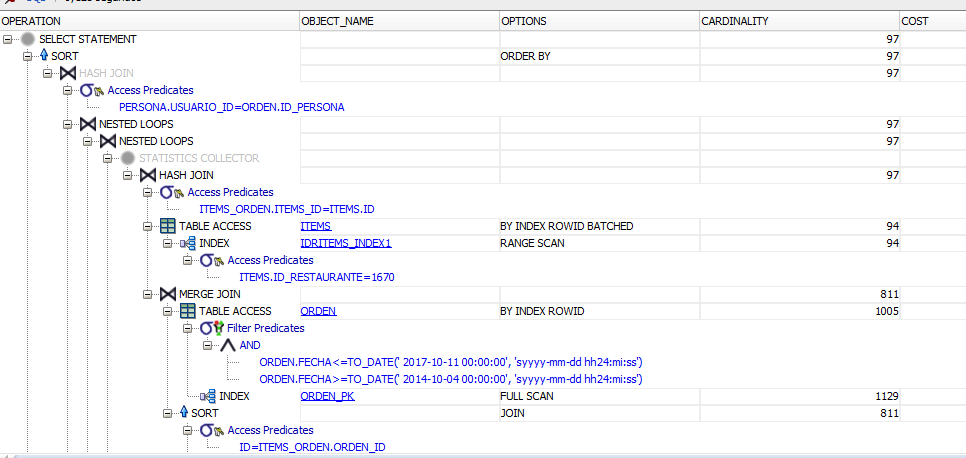
**Pruebas**

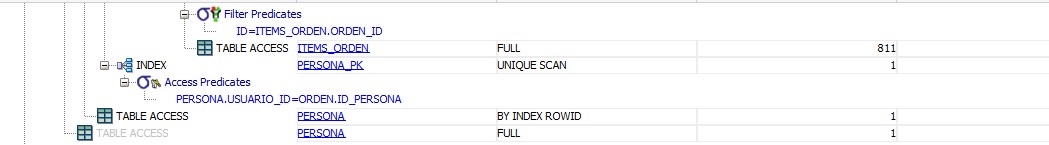
Para las pruebas de ejecución se utilizo el rango de fecha de 4-10-2014 y 11-10-2017, este es un rango superior al requerido y las fechas de los pedidos van desde el 2014 y 2017, también como restaurante se uso el restaurante con un id =1670 ya que es un restaurante con bastantes pedidos.



Como se muestra en los resultados se tubo un tiempo de 0.031 segundos de búsqueda se obtuvieron los resultados por fecha.

El plan desarrollado por SQL developer de este requerimiento es el siguiente:





como se puede ver en el plan de ejecución Oracle usa hash joins y mergue join para lograr los join rápidamente, alguno de los index que usa es el idex de items en id ítems para encontrar el restaurante y en el mergue join este usa un index rowind para obtener el rango de fechas que se le esta pidiendo.

**Plan de ejecución Planteado**

Para el plan de ejecución planteado por el grupo se considera ya que se conoce la estructura total de la aplicación una búsqueda de rango por fecha de las ordenes que se tiene y luego de obtener este rango de fechas realizar un hash join con la tabla ORDEN\_ITEMS y otro hash join restaurantes y USUARIOS, con eso obtendría un resultado esperado sin tener que hacer un join de todas las tablas a diferencia de la ejecución de Oracle donde realizó una serie de joins y luego hiso la búsqueda por rango.

**Segunda Parte de Requerimiento**

La segunda parte del requerimiento limitaba que si el consultante era un usuario cliente este solo podría ver su propia información.

La sentencia SQL usada fue la siguiente:

*SELECT PERSONA.USUARIO\_ID AS ID , ROL,NOMBRE,NOMBRE\_PRODUC,FECHA*

*FROM(select ORDEN.ID\_PERSONA AS ID,ORDEN.FECHA AS FECHA ,ITEMS.NOMBRE AS NOMBRE\_PRODUC*

*from( ORDEN JOIN ITEMS\_ORDEN*

*ON ID = ITEMS\_ORDEN.ORDEN\_ID)JOIN ITEMS*

*ON ITEMS\_ORDEN.ITEMS\_ID = ITEMS.ID*

*WHERE ITEMS.ID\_RESTAURANTE = 1670*

*AND FECHA BETWEEN '4-10-2014' AND '11-10-2017') JOIN PERSONA*

*ON PERSONA.USUARIO\_ID = ID*

*WHERE ID = 1154382*

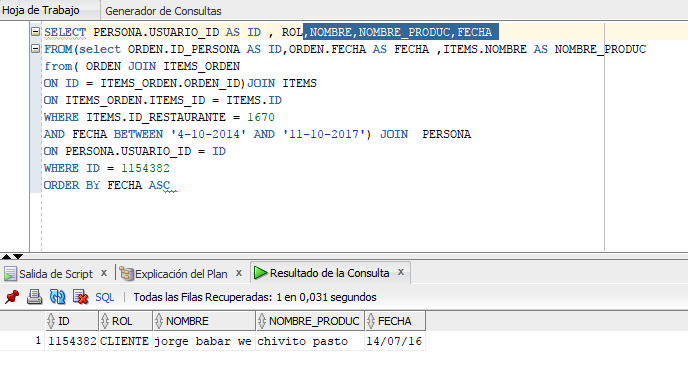
*ORDER BY FECHA ASC*

**Index**

Para este requerimiento se usaron los mismos index que en anterior y se agrego uso el index del PK de la tabla PERSONA para hallar el usuario requerido solamente.

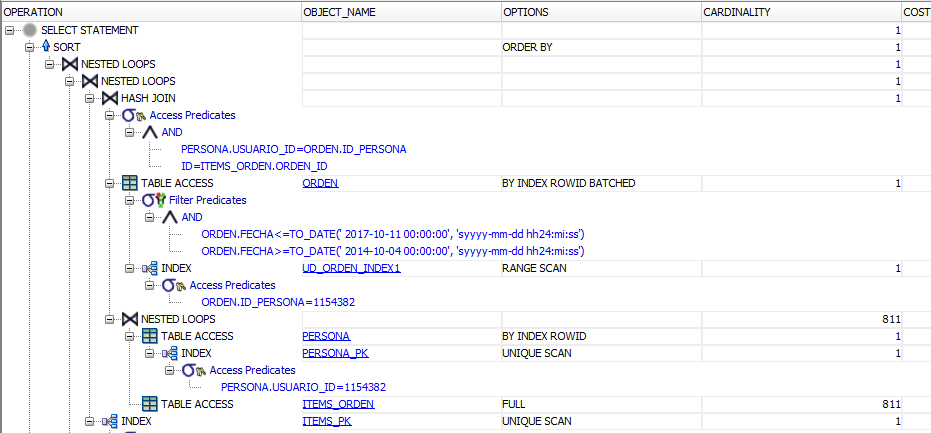
**Pruebas**

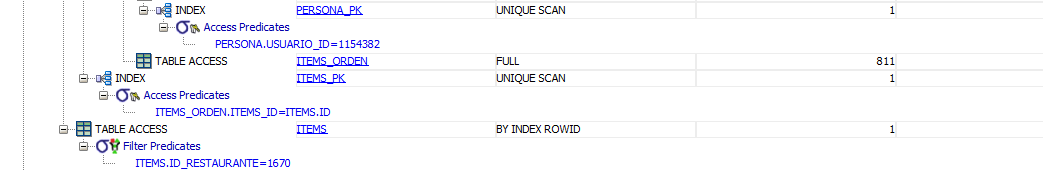
Para las pruebas de esta segunda se sigue utilizando el restaurante con id =1670 por su cantidad de ordenes en estas fechas y el usuario usado es el usuario con id = 1154382.



Como se puede ver en los resultados solo se muestra la dupla con el usuario que comió en un rango de fecha en el restaurante.

**Plan de ejecución de este parte**





En el plan de desarrollo se miestra igual que el anterior como Oracle realisa joins y un escaneo de rango pero al final realiza un Nested loop con el index de id de Usuario para hallar solo las duplas de este usuario.

Plan de ejecución Planteado

Para esta segunda parte el plan de ejecución es parecido al que se planteo en el primer punto pero se plantea que la búsqueda de usuario se haga al principio de la búsqueda antes de buscar el rango de fechas de las ordenes por la FK que tiene Usuario en Ordenes.

**RCF10:**

Este requerimiento es muy semejante al anterior, por lo tanto los mismos índices son usados en este caso FECHA, las mismas tablas se utilizan (orden\_items,orden,persona e ítems), la diferencia es que solicitan los usuarios que No consumieron ningún producto de un restaurante en un RANGO de fechas. Por lo tanto el mismo análisis puede ser implementado, se utilizara un árbol B+ para manejar los rangos, reducir el tiempo de búsqueda y de I/O.

En una búsqueda este resulta un poco mas demorado que el anterior ya que solo localizaba las duplas que tenían el id del restaurante en este caso se requiere buscar todas las duplas que no tiene ese id.

**Primera Parte**

Nuevamente para este ejercicio primero se ejecutará el requerimiento se utilizara el restaurante con el id = 1670 ya que es uno de los restaurantes con mas ordenes registradas en el programa, como la primera es una ejecución de administrador se muestra todos los usuarios que no consumieron en este restaurante en el periodo de fecha 4-10-2016 y 11-10-2017 , se uso un rango mucho mas pequeño que en la sentencia anterior debido a que una búsqueda por todo los años es mucho mas pesada ya que estaría solo eliminando las ordenes de ese restaurante, nuevamente las pruebas se realizan con el ordenamiento para fechas.

**Sentencia SQL:**

*SELECT PERSONA.USUARIO\_ID AS ID , ROL,NOMBRE,NOMBRE\_PRODUC,FECHA*

*FROM(select ORDEN.ID\_PERSONA AS ID,ORDEN.FECHA AS FECHA ,ITEMS.NOMBRE AS NOMBRE\_PRODUC*

*from( ORDEN JOIN ITEMS\_ORDEN*

*ON ID = ITEMS\_ORDEN.ORDEN\_ID)JOIN ITEMS*

*ON ITEMS\_ORDEN.ITEMS\_ID = ITEMS.ID*

*WHERE ITEMS.ID\_RESTAURANTE != 1670*

*AND FECHA BETWEEN '4-10-2016' AND '11-10-2017') JOIN PERSONA*

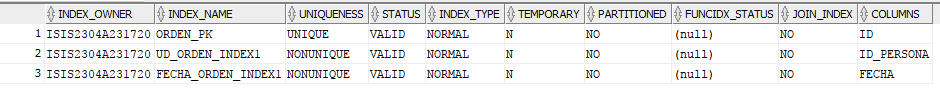
*ON PERSONA.USUARIO\_ID = ID*

*ORDER BY FECHA ASC*

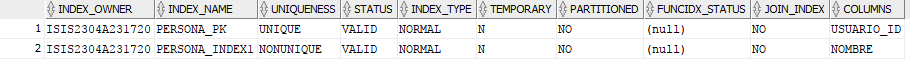
**INDEX**

Para este sentencia se usaron las mismas tablas que el punto anterior PERSONA,ORDEN,ITEMS y ITEMS\_ORDEN, por consiguiente los indexes que se usaron fueron los mismos que se usaron en el punto anterior, con la diferencias que el index de ID\_RESTAURANTE el cual no es usado ya que la alocacion de los demás index son demasiado altos y el plan de ejecución de Oracle prefiere no usarlo y recorrer los join y descartar los id de restaurante.

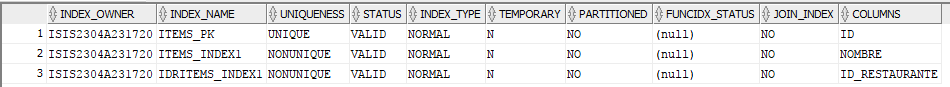
**INDEX ORDEN.**



**INDEX PERSONA**

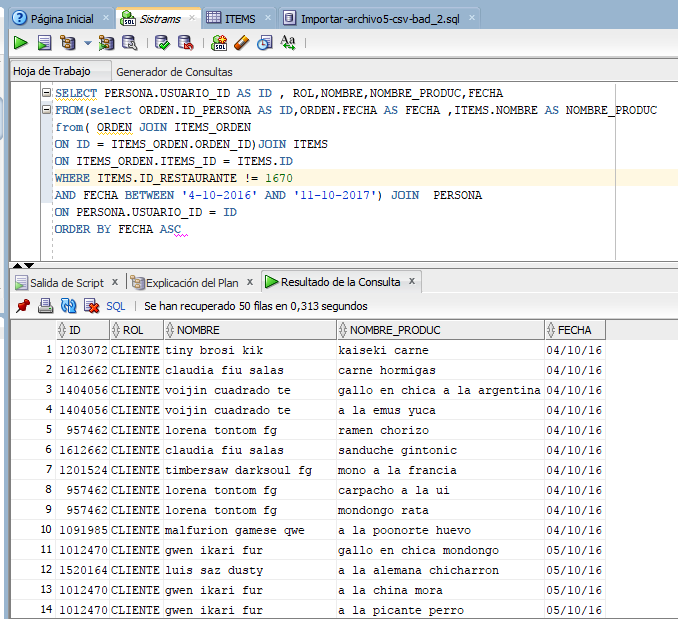


**INDEX ITEMS**



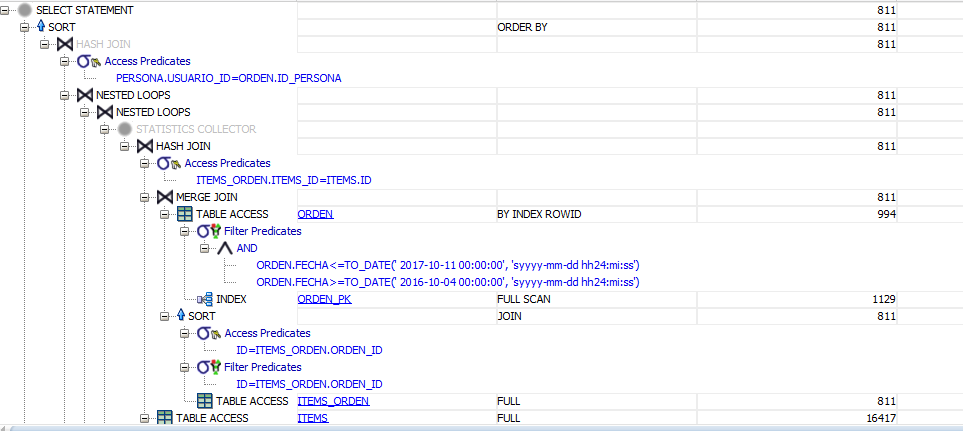
Pruebas

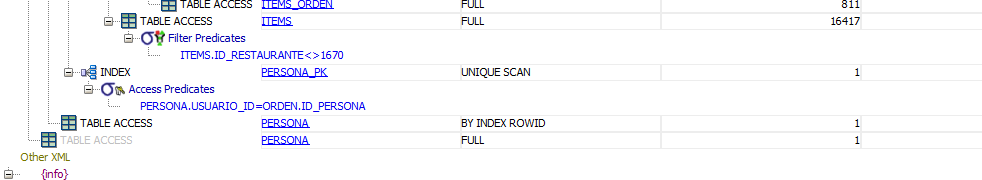
Como se mencionó para la prueba se usaron los restaurantes con ID=1670 y un rango de fechas entre fecha 4-10-2016 y 11-10-2017.



Como se mostro el tiempo de ejecución fue de 0.3 segundos para mas de 50 filas, como se mencionó la razón que se encuentra para que esta sentencia muy parecida a la anterior es que la cantidad de datos que se muestra es mayor y el index de id restaurante no es usado por Oracle debido a que no e sutil al momento de eliminar las duplas del restaurante 1670.

**Plan de ejecución:**

****

****

Como se muestra en el plan de ejecución es muy parecido al anterior donde se usan los index para hacer los join y Hash join pero a diferencia del requerimiento anterior el index de ID\_RESTAURANTE no es usado.

**Plan de ejecución planteado.**

El plan de ejecución planteado por el grupo es primero hacer una eliminación de ordenes por fechas lo que reduciría la cantidad de ordenes para el hash y luego de el hast join eliminar los pedidos del restaurante seleccionado en el rango de fechas.

**Segunda Parte**

En la segunda parte se pedía que se diera la misma consulta de pedidos en un rango de fecha pero como se consulta desde un usuario cliente este solo puede ver la información propia, si el usuario consumió algún alimento en ese rango de fecha en otro restaurante debe aparecer la dupla, si este no a consumido en ningún restaurante o en el restaurante seleccionado este no debe devolver ninguna dupla, en el programa se maneja esto por medio de una advertencia que le informa al usuario que no tiene consumos en esta fechas.

**Sentencia SQL:**

*SELECT PERSONA.USUARIO\_ID AS ID , ROL,NOMBRE,NOMBRE\_PRODUC,FECHA*

*FROM(select ORDEN.ID\_PERSONA AS ID,ORDEN.FECHA AS FECHA ,ITEMS.NOMBRE AS NOMBRE\_PRODUC*

*from( ORDEN JOIN ITEMS\_ORDEN*

*ON ID = ITEMS\_ORDEN.ORDEN\_ID)JOIN ITEMS*

*ON ITEMS\_ORDEN.ITEMS\_ID = ITEMS.ID*

*WHERE ITEMS.ID\_RESTAURANTE != 1670*

*AND FECHA BETWEEN '4-10-2016' AND '11-10-2017') JOIN PERSONA*

*ON PERSONA.USUARIO\_ID = ID*

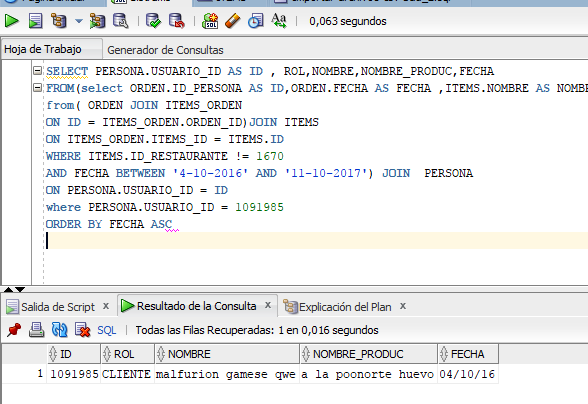
*WHERE PERSONA.USUARIO\_ID = 1091985*

*ORDER BY FECHA ASC*

La sentencia es muy parecida a la sentencia de la primera parte su única diferencia es la comprobación que las duplas devueltas solo correspondan al usuario que realizo la consulta.

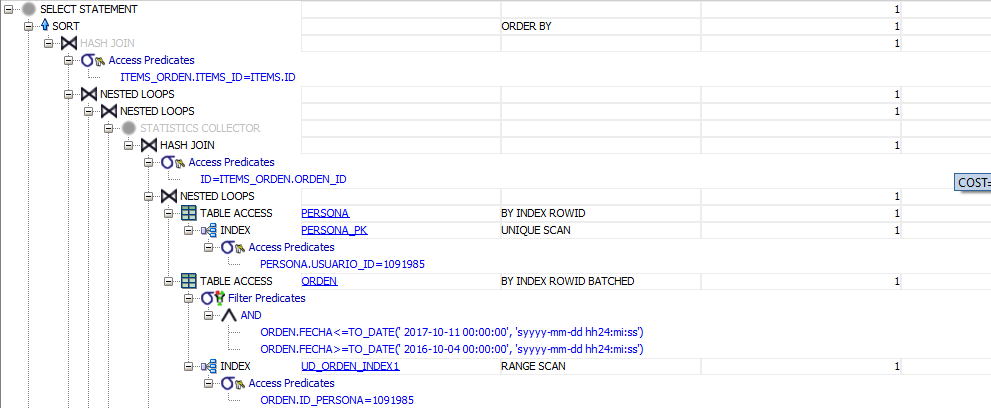
**Pruebas**

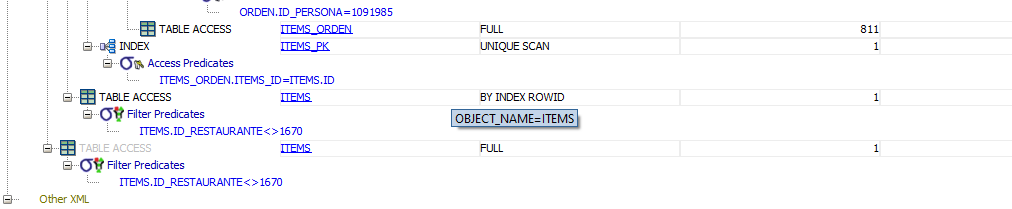
Para las pruebas se uso el restaurante con id=1670 ya que este posee una cantidad significativa de ordenes y se dio un rango de fecha entre el 4-10-2016 y el 11-10-2017 y un usuario con id=*1091985* este usuario se eligió ya que es un usuario que tiene consumos en este periodo de tiempo en otros restaurantes aparte del seleccionado.



El resultado se da en 0.016 segundos y se retorna la dupla del ítem consumido por este usuario que no pertenece al restaurante seleccionado.

**Plan de ejecución**





En este plan de ejecución es muy parecido al anterior, ya que realiza las mismas operaciones de join entre las tablas pero lo que lo diferencia y lo hace mucho más rápido, es que este usa el index de usuario para encontrar las duplas, a diferencia del anterior donde no usa index para recorrer las duplas y eliminarlas.

RCF11:

En este requerimiento utilizamos tablas orden ítems, ítems y restaurante así conseguir los ítems mas vendidos, los menos vendidos y los restaurantes más y menos frecuentados, el índice que consideramos va a ser utilizado va a ser el de ID\_restaurante, luego se realizara el count a los ítems ( MAX,MIN) y a los restaurantes de la tupla. Ya que en este caso no se utilizan rangos si no solo múltiples condiciones, posiblemente la situación será manejada por por una tabla hash, usando hashjoins. Sin embargo debido al diseño de nuestras tablas realizar la dicha consulta con solo una sentencia de SQL resulto ser bastante complicado y fue necesario hacerlo en partes. Una de de las razones, por ejemplo al solicitar el valor máximo y mínimo no puedo realizar una partición y así pedir la primera y la última tupla para obtener el mayor y el menor en una sola consulta.

SQL:

SELECT COUNT(ITO.ITEMS\_ID)as cant,ITO.ITEMS\_ID

FROM ITEMS\_ORDEN ITO JOIN ORDEN ORD ON ITO.ORDEN\_ID= ORD.ID JOIN ITEMS IT ON IT.ID=ITO.ITEMS\_ID

GROUP BY ITO.ITEMS\_ID

ORDER BY cant asc;

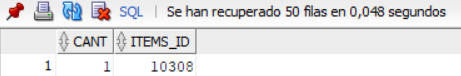
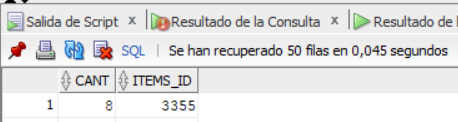
SELECT COUNT(IT.ID\_RESTAURANTE)as visitas, REST.NOMBRE

FROM ITEMS\_ORDEN ITO JOIN ORDEN ORD ON ITO.ORDEN\_ID= ORD.ID JOIN ITEMS IT ON IT.ID=ITO.ITEMS\_ID JOIN RESTAURANTE REST ON REST.ID=IT.ID\_RESTAURANTE

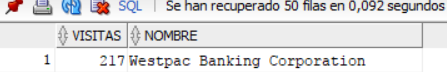
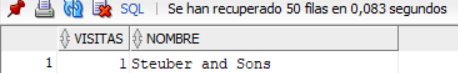
GROUP BY REST.ID, REST.NOMBRE

ORDER BY visitas asc ;

**Ítems:**

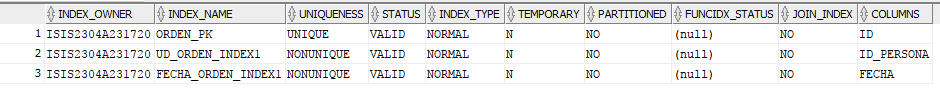


Restaurantes:

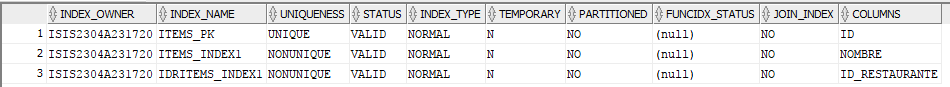


**Índex:**

**INDEX ORDEN.**



**INDEX ITEMS**

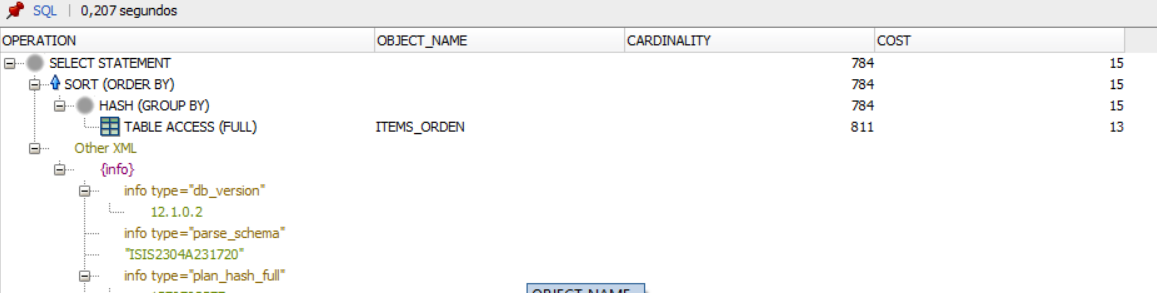


**INDEX RESTAURANTE**

****

**El plan desarrollado es el siguiente:**

**Ítems:**



Para esta parte el optimizador de Oracle no tubo necesidad de utilizar los índices todo lo manejo por medio de un HASH.

Restaurantes:



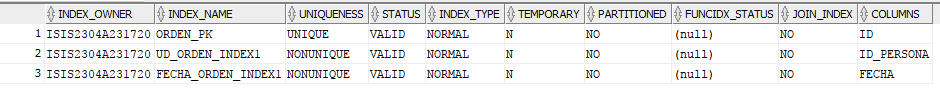
Para la segunda parte como se puede ver y nos imaginamos MySql ha utilizado Hash joins varias veces y así optimizar los tiempos de las 3 tablas que componen la consulta , además ha utilizado los índices de las Primary Keys,

**RCF12:**

En este requerimiento utilizamos las tablas persona, Orden, ítems\_Orden, ítems y finalmente menú\_orden. Para la comparación debería ser 37500 COP pero nosotros al manejar los precios en dólares, pusimos en la igualdad 37, finalmente la otra tabla verifica si en esa orden se solicitó un menú el cual le pertenece a una persona, si se encuentra que nunca ha solicitado menús entra en la tupla resultado.

**Índex:**

**INDEX ORDEN.**



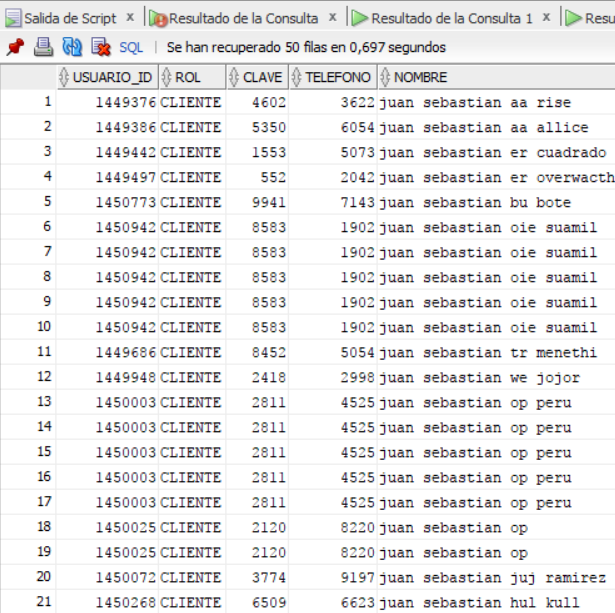
**SQL:**

SELECT PERSONA.USUARIO\_ID,PERSONA.ROL,PERSONA.CLAVE,PERSONA.Telefono,PERSONA.NOMBRE

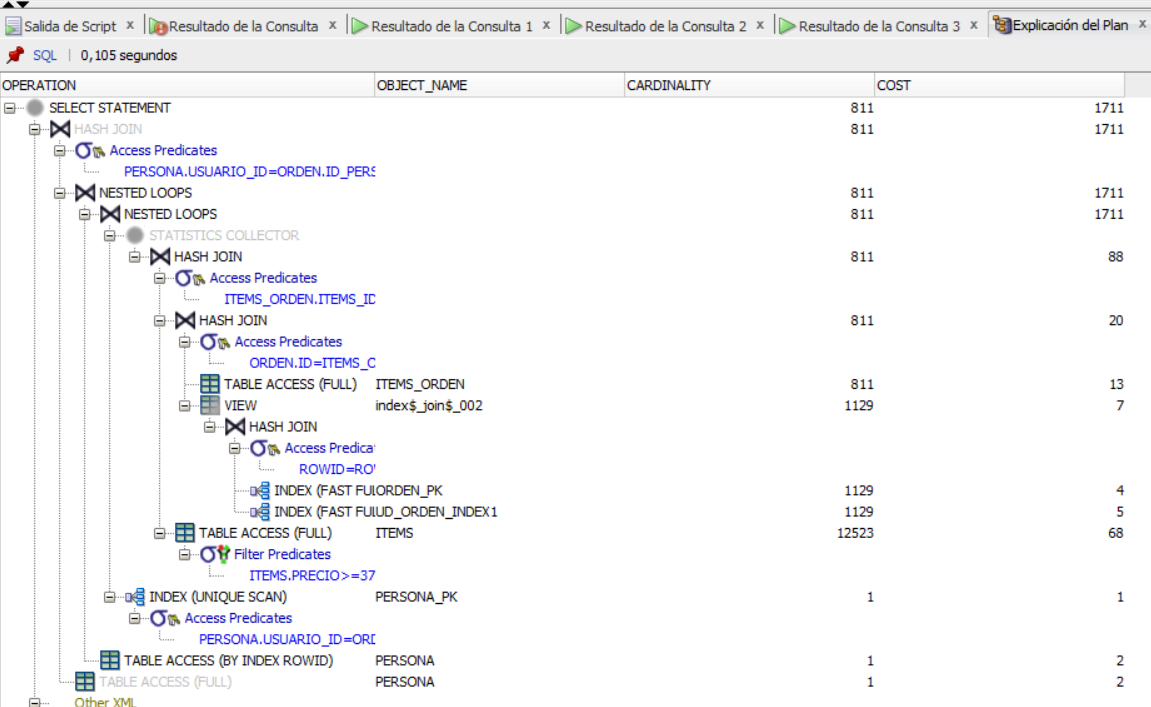
FROM PERSONA JOIN ORDEN ON PERSONA.USUARIO\_ID = ORDEN.ID\_PERSONA JOIN ITEMS\_ORDEN ON ORDEN.ID=ITEMS\_ORDEN.ORDEN\_ID JOIN ITEMS ON ITEMS\_ORDEN.ITEMS\_ID=ITEMS.ID

JOIN MENU\_ORDEN ON ORDEN.ID=MENU\_ORDEN.ID\_ORDEN

WHERE ITEMS.PRECIO>=37 OR MENU\_ORDEN.solicitado ='F';



**Plan:**



Como podemos observar Oracle ha utilizado un HASH JOIN esto se debe a que no hemos utilizado ninguna rango en las condiciones para encontrar unas tuplas especificas, en caso contrario agregando lo que falta de la sentencia utilizaría un merge join, al utilizar las PK sabemos que maneja una selectividad baja, y al ser una sentencia sencilla, no es necesario utilizar los otros índices creados en este caso.