

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 atienio 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_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1 ISIS2304A231720	ORDEN_PK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID
2 ISIS2304A231720	UD_ORDEN_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID_PERSONA
3 ISIS2304A231720	FECHA_ORDEN_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	FECHA

INDEX PERSONA

INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1 ISIS2304A231720	PERSONA_PK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	USUARIO_ID
2 ISIS2304A231720	PERSONA_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	NOMBRE

INDEX ITEMS

INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1 ISIS2304A231720	ITEMS_PK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID
2 ISIS2304A231720	ITEMS_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	NOMBRE
3 ISIS2304A231720	IDRITEMS_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID_RESTAURANTE

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.

The screenshot shows the SQL Developer interface. The top pane displays a SQL query:


```
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
ORDER BY FECHA ASC
```

 The bottom pane shows the query results in a table with 6 columns: ID, ROL, NOMBRE, NOMBRE_PRODUC, and FECHA. There are 15 rows of data. The status bar at the bottom indicates "Se han recuperado 50 filas en 0,031 segundos".

ID	ROL	NOMBRE	NOMBRE_PRODUC	FECHA	
1	1184198	CLIENTE	fux led bas	chivito kaiseki	13/01/15
2	960410	CLIENTE	fiona conor hop	chivito frijoles	14/01/15
3	1706971	CLIENTE	dacota fan game	chivito smoke meat sandwich	03/04/15
4	1112875	CLIENTE	rigel lilit frik	chivito sardinas	31/05/15
5	1031202	CLIENTE	dacota rambo lol	chivito silpacho	22/06/15
6	1359900	CLIENTE	happyhop ramirez wow	chivito kaiseki	25/06/15
7	1708636	CLIENTE	dacota hul punba	chivito ensalda	30/07/15
8	1412220	CLIENTE	kaplan baul fur	chivito trucha	07/08/15
9	983184	CLIENTE	brosigar sniper ri	chivito helado	18/08/15
10	1147721	CLIENTE	catalina rise fez	chivito la suprema	25/08/15
11	1000205	CLIENTE	traxe dusty sr	chivito hormigas	09/09/15
12	1709072	CLIENTE	dacota bu wain	chivito empanada	10/09/15
13	1316211	CLIENTE	hanzo tontom cac	chivito pepsi	17/09/15
14	1722428	CLIENTE	moonshar yu gomez	chivito hamburguesa	26/09/15
15	1387933	CLIENTE	lichkking churchyll king	chivito sardinas	03/11/15

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:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				97
SORT		ORDER BY		97
HASH JOIN				97
Access Predicates	PERSONA.USUARIO_ID=ORDEN.ID_PERSONA			
NESTED LOOPS				97
NESTED LOOPS				97
STATISTICS COLLECTOR				
HASH JOIN				97
Access Predicates	ITEMS_ORDEN.ITEMS_ID=ITEMS.ID			
TABLE ACCESS	ITEMS	BY INDEX ROWID BATCHED		94
INDEX	IDRITEMS_INDEX1	RANGE SCAN		94
Access Predicates	ITEMS.ID_RESTAURANTE=1670			
MERGE JOIN				811
TABLE ACCESS	ORDEN	BY INDEX ROWID		1005
Filter Predicates	AND			
ORDEN.FECHA<=TO_DATE(' 2017-10-11 00:00:00', 'yyyy-mm-dd hh24:mi:ss')				
ORDEN.FECHA>=TO_DATE(' 2014-10-04 00:00:00', 'yyyy-mm-dd hh24:mi:ss')				
INDEX	ORDEN_PK	FULL SCAN		1129
SORT		JOIN		811
Access Predicates	ID=ITEMS_ORDEN.ORDEN_ID			
Filter Predicates	ID=ITEMS_ORDEN.ORDEN_ID			
TABLE ACCESS	ITEMS_ORDEN	FULL	811	
INDEX	PERSONA_PK	UNIQUE SCAN	1	
Access Predicates	PERSONA.USUARIO_ID=ORDEN.ID_PERSONA			
TABLE ACCESS	PERSONA	BY INDEX ROWID	1	
TABLE ACCESS	PERSONA	FULL	1	

como se puede ver en el plan de ejecución Oracle usa hash joins y merge 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 merge 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 hizo 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

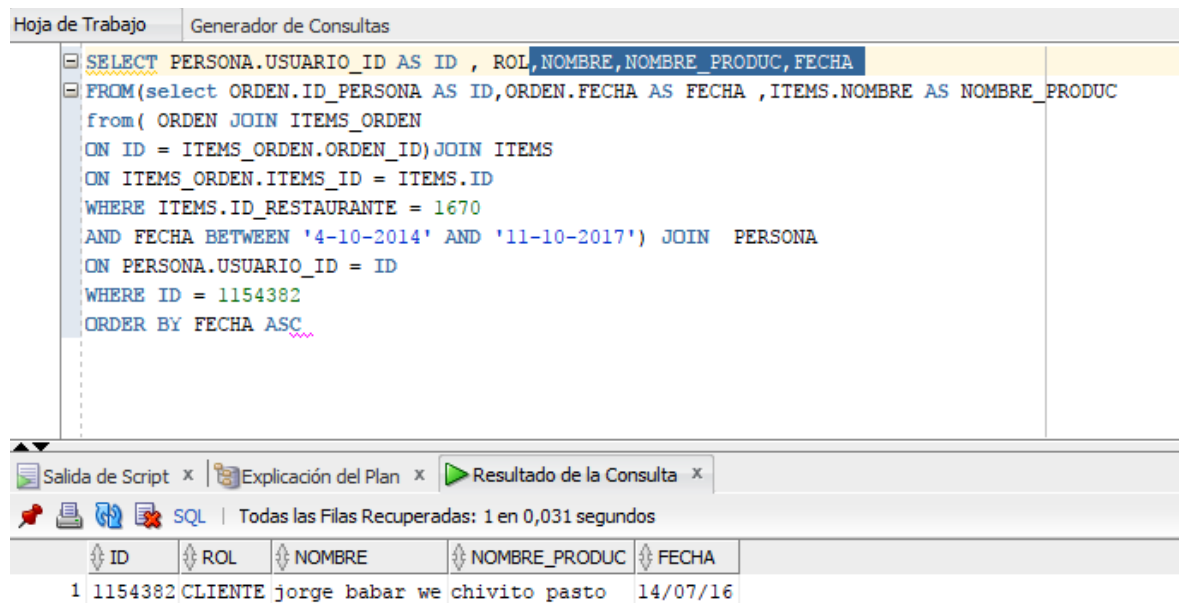
```

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.



The screenshot shows a database query tool interface. The top tab is 'Hoja de Trabajo' and the sub-tab is 'Generador de Consultas'. The SQL query is displayed in the main area:

```

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

```

Below the query, there are three tabs: 'Salida de Script', 'Explicación del Plan', and 'Resultado de la Consulta'. The 'Resultado de la Consulta' tab is active, showing the results of the query. The status bar indicates 'Todas las Filas Recuperadas: 1 en 0,031 segundos'.

ID	ROL	NOMBRE	NOMBRE_PRODUC	FECHA
1 1154382	CLIENTE	jorge babar we	chivito pasto	14/07/16

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

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			1
SORT			1
NESTED LOOPS			1
NESTED LOOPS			1
HASH JOIN			1
Access Predicates			1
AND			
PERSONA.USUARIO_ID=ORDEN.ID_PERSONA			
ID=ITEMS_ORDEN.ORDEN_ID			
TABLE ACCESS	ORDEN	BY INDEX ROWID BATCHED	1
Filter Predicates			
AND			
ORDEN.FECHA<=TO_DATE(' 2017-10-11 00:00:00', 'yyyy-mm-dd hh24:mi:ss')			
ORDEN.FECHA>=TO_DATE(' 2014-10-04 00:00:00', 'yyyy-mm-dd hh24:mi:ss')			
INDEX	UD_ORDEN_INDEX1	RANGE SCAN	1
Access Predicates			
ORDEN.ID_PERSONA=1154382			
NESTED LOOPS			811
TABLE ACCESS	PERSONA	BY INDEX ROWID	1
INDEX	PERSONA_PK	UNIQUE SCAN	1
Access Predicates			
PERSONA.USUARIO_ID=1154382			
TABLE ACCESS	ITEMS_ORDEN	FULL	811
INDEX	ITEMS_PK	UNIQUE SCAN	1
INDEX	PERSONA_PK	UNIQUE SCAN	1
Access Predicates			
PERSONA.USUARIO_ID=1154382			
TABLE ACCESS	ITEMS_ORDEN	FULL	811
INDEX	ITEMS_PK	UNIQUE SCAN	1
Access Predicates			
ITEMS_ORDEN.ITEMS_ID=ITEMS.ID			
TABLE ACCESS	ITEMS	BY INDEX ROWID	1
Filter Predicates			
ITEMS.ID RESTAURANTE=1670			

En el plan de desarrollo se muestra igual que el anterior como Oracle realiza 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_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1	ISIS2304A231720 ORDEN_PK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID
2	ISIS2304A231720 UD_ORDEN_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID_PERSONA
3	ISIS2304A231720 FECHA_ORDEN_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	FECHA

INDEX PERSONA

INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1	ISIS2304A231720 PERSONA_PK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	USUARIO_ID
2	ISIS2304A231720 PERSONA_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	NOMBRE

INDEX ITEMS

	INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1	ISIS2304A231720	ITEMS_PK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID
2	ISIS2304A231720	ITEMS_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	NOMBRE
3	ISIS2304A231720	IDRITEMS_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID_RESTAURANTE

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.

The screenshot shows the SQL Developer interface with a query window titled 'Hoja de Trabajo' and 'Generador de Consultas'. The query is as follows:

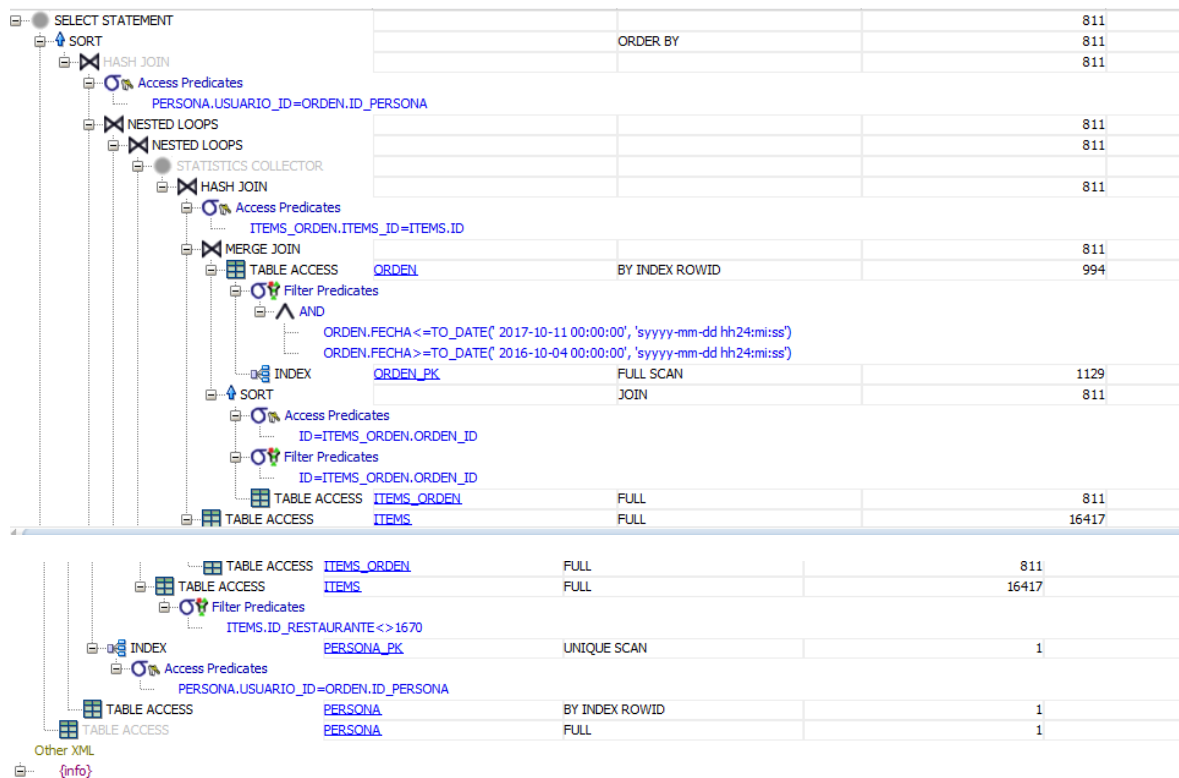
```
SELECT PERSONA.USUARIO_ID AS ID , ROL,NOMBRE,NOMBRE_PRODUC,FECHA
FROM(select ORDEN.ID_PERSONA AS ID,ORDEN.FECHA AS FECHA ,ITEMS.NOMBRE AS NOMBRE_PRODUC
from( ORDEN JOIN ITEMS_ORDEN
ON ID = ITEMS_ORDEN.ORDEN_ID)JOIN ITEMS
ON ITEMS_ORDEN.ITEMS_ID = ITEMS.ID
WHERE ITEMS.ID_RESTAURANTE != 1670
AND FECHA BETWEEN '4-10-2016' AND '11-10-2017') JOIN PERSONA
ON PERSONA.USUARIO_ID = ID
ORDER BY FECHA ASC
```

The results window shows the output of the query, displaying 50 rows of data. The columns are ID, ROL, NOMBRE, NOMBRE_PRODUC, and FECHA. The data is sorted by FECHA ASC.

ID	ROL	NOMBRE	NOMBRE_PRODUC	FECHA
1 1203072	CLIENTE	tiny brosi kik	kaiseki carne	04/10/16
2 1612662	CLIENTE	claudia fiu salas	carne hormigas	04/10/16
3 1404056	CLIENTE	voijin cuadrado te	gallo en chica a la argentina	04/10/16
4 1404056	CLIENTE	voijin cuadrado te	a la emus yuca	04/10/16
5 957462	CLIENTE	lorena tontom fg	ramen chorizo	04/10/16
6 1612662	CLIENTE	claudia fiu salas	sanduche gintonic	04/10/16
7 1201524	CLIENTE	timbersaw darksoul fg	mono a la francia	04/10/16
8 957462	CLIENTE	lorena tontom fg	carpacho a la ui	04/10/16
9 957462	CLIENTE	lorena tontom fg	mondongo rata	04/10/16
10 1091985	CLIENTE	malfurion gameese qwe	a la poonorte huevo	04/10/16
11 1012470	CLIENTE	gwen ikari fur	gallo en chica mondongo	05/10/16
12 1520164	CLIENTE	luis saz dusty	a la alemana chicharron	05/10/16
13 1012470	CLIENTE	gwen ikari fur	a la china mora	05/10/16
14 1012470	CLIENTE	gwen ikari fur	a la picante perro	05/10/16

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 es 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.

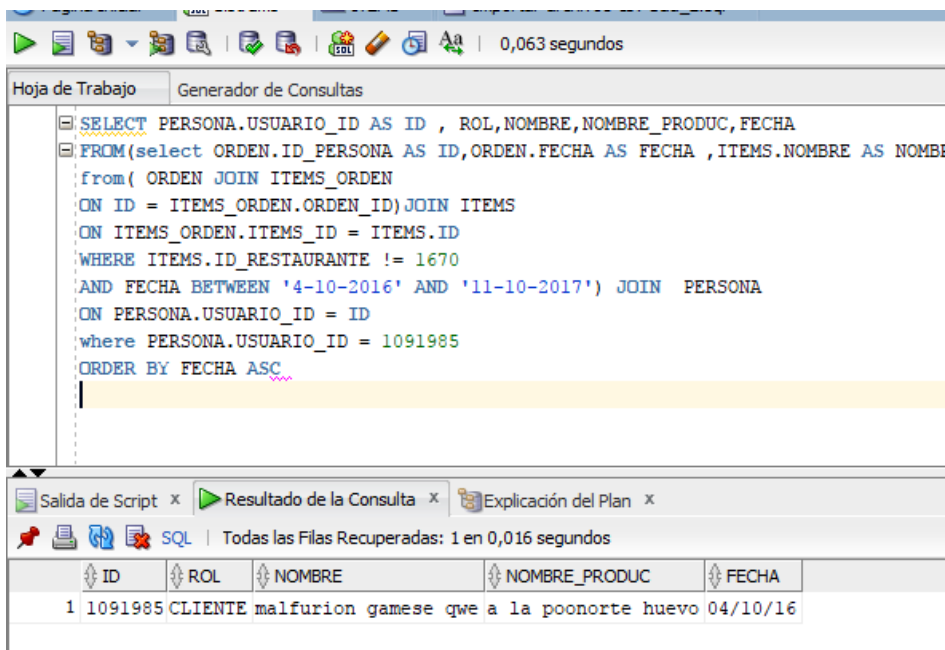
Sentencia SQL:

```
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

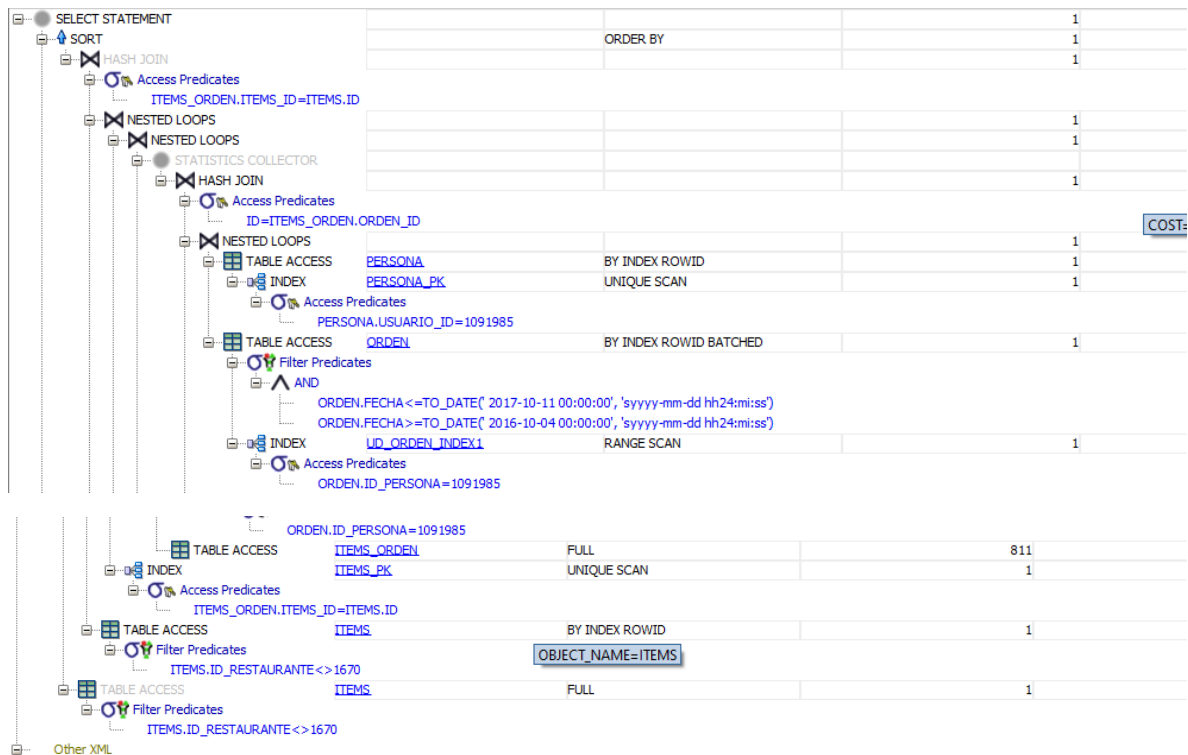
Para las pruebas se uso el restaurante con id=1670 ya que este posee una cantidad significativa de ordenes y se dio un rango de fecha entre el 4-10-2016 y el 11-10-2017 y un usuario con id=1091985 este usuario se eligió ya que es un usuario que tiene consumos en este periodo de tiempo en otros restaurantes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



The screenshot shows a SQL query execution interface. The top toolbar includes icons for running the query, saving, and other functions, along with a timer showing 0,063 segundos. The main window is titled 'Hoja de Trabajo' and 'Generador de Consultas'. The SQL query is displayed in a text area, and the results are shown in a table at the bottom. The table has columns for ID, ROL, NOMBRE, NOMBRE_PRODUC, and FECHA. The results show a single row with the following data: 1, 1091985, CLIENTE, malfurion gamede qwe a la poonorte huevo, 04/10/16.

ID	ROL	NOMBRE	NOMBRE_PRODUC	FECHA
1	1091985	CLIENTE	malfurion gamede qwe a la poonorte huevo	04/10/16

Plan de ejecución



RCF11:

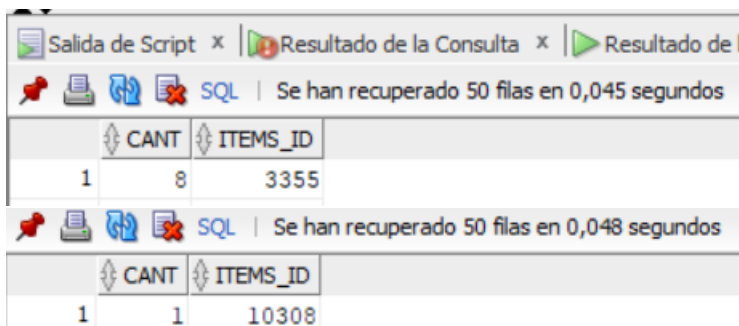
En este requerimiento utilizamos tablas orden ítems, ítems y restaurante así conseguir los ítems mas vendidos, los menos vendidos y los restaurantes más y menos frecuentados, el índice que consideramos va a ser utilizado va a ser el de ID_restaurante, luego se realizara el count a los ítems (MAX,MIN) 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:

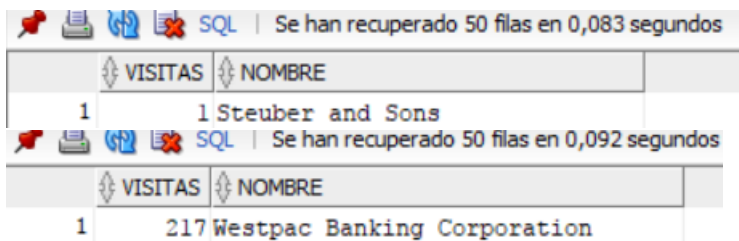


The screenshot shows a database query result window with two tabs: 'Salida de Script' and 'Resultado de la Consulta'. The 'Resultado de la Consulta' tab is active, displaying two tables. The first table has columns 'CANT' and 'ITEMS_ID' and contains one row with values 8 and 3355. The second table also has columns 'CANT' and 'ITEMS_ID' and contains one row with values 1 and 10308. Both tables are preceded by a status bar indicating 'Se han recuperado 50 filas en 0,045 segundos' and 'Se han recuperado 50 filas en 0,048 segundos' respectively.

CANT	ITEMS_ID
8	3355

CANT	ITEMS_ID
1	10308

Restaurantes:



The screenshot shows a database query result window with two tabs: 'Salida de Script' and 'Resultado de la Consulta'. The 'Resultado de la Consulta' tab is active, displaying two tables. The first table has columns 'VISITAS' and 'NOMBRE' and contains one row with values 1 and '1 Steuber and Sons'. The second table also has columns 'VISITAS' and 'NOMBRE' and contains one row with values 1 and '217 Westpac Banking Corporation'. Both tables are preceded by a status bar indicating 'Se han recuperado 50 filas en 0,083 segundos' and 'Se han recuperado 50 filas en 0,092 segundos' respectively.

VISITAS	NOMBRE
1	1 Steuber and Sons

VISITAS	NOMBRE
1	217 Westpac Banking Corporation

Índex:

INDEX ORDEN.

INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1 ISIS2304A231720	ORDEN_PK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID
2 ISIS2304A231720	UD_ORDEN_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID_PERSONA
3 ISIS2304A231720	FECHA_ORDEN_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	FECHA

INDEX ITEMS

INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1 ISIS2304A231720	ITEMS_PK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID
2 ISIS2304A231720	ITEMS_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	NOMBRE
3 ISIS2304A231720	IDRITEMS_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID_RESTAURANTE

INDEX RESTAURANTE

INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1 ISIS2304A231720	RESTAURANTE_PK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID

El plan desarrollado es el siguiente:

Ítems:

SQL 0,207 segundos			
OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			784 15
SORT (ORDER BY)			784 15
HASH (GROUP BY)			784 15
TABLE ACCESS (FULL)	ITEMS_ORDEN		811 13
Other XML			
{info}			
info type="db_version"			
12.1.0.2			
info type="parse_schema"			
"ISIS2304A231720"			
info type="plan_hash_full"			

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

Restaurantes:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		811	91
SORT (ORDER BY)		811	91
HASH (GROUP BY)		811	91
HASH JOIN		811	89
Access Predicates	REST.ID=IT.ID_RESTAURANTE		
NESTED LOOPS		811	89
NESTED LOOPS			
STATISTICS COLLECTOR			
HASH JOIN		811	80
Access Predicates	IT.ID=ITO.ITEM		
TABLE ACCESS (FULL) ITEMS_ORDEN	index\$_join\$_004	811	13
VIEW		16511	67
HASH JOIN		CARDINALITY=16511	
Access Pre	ROWID=		
INDEX (FAST) ITEMS_PK		16511	39
INDEX (FAST) ITEMS_INDEX1		16511	45
INDEX (UNIQUE SCAN) RESTAURANTE_PK			
Access Predicates	REST.ID=IT.ID_RES		
TABLE ACCESS (BY INDEX ROWID) RESTAURANTE		1	9
TABLE ACCESS (FULL) RESTAURANTE		1722	9

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.

INDEX_OWNER	INDEX_NAME	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED	FUNCIDX_STATUS	JOIN_INDEX	COLUMNS
1 ISIS2304A231720	ORDEN_FK	UNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID
2 ISIS2304A231720	UD_ORDEN_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	ID_PERSONA
3 ISIS2304A231720	FECHA_ORDEN_INDEX1	NONUNIQUE	VALID	NORMAL	N	NO	(null)	NO	FECHA

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

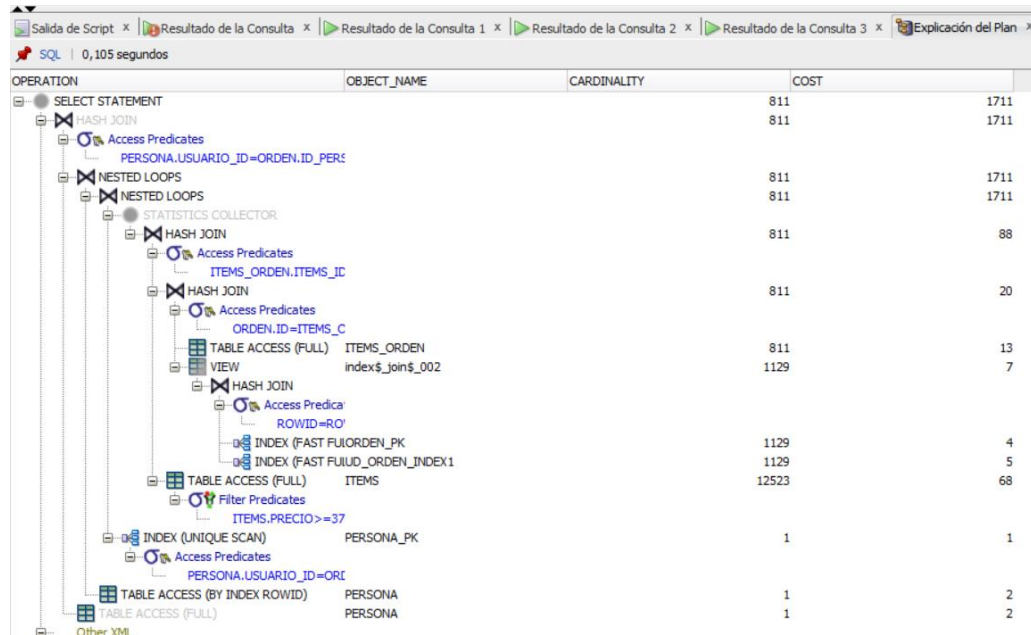
```

JOIN MENU_ORDEN ON ORDEN.ID=MENU_ORDEN.ID_ORDEN

WHERE ITEMS.PRECIO>=37 OR MENU_ORDEN.solicitado ='F';

	USUARIO_ID	ROL	CLAVE	TELEFONO	NOMBRE
1	1449376	CLIENTE	4602	3622	juan sebastian aa rise
2	1449386	CLIENTE	5350	6054	juan sebastian aa allice
3	1449442	CLIENTE	1553	5073	juan sebastian er cuadrado
4	1449497	CLIENTE	552	2042	juan sebastian er overwacth
5	1450773	CLIENTE	9941	7143	juan sebastian bu bote
6	1450942	CLIENTE	8583	1902	juan sebastian oie suamil
7	1450942	CLIENTE	8583	1902	juan sebastian oie suamil
8	1450942	CLIENTE	8583	1902	juan sebastian oie suamil
9	1450942	CLIENTE	8583	1902	juan sebastian oie suamil
10	1450942	CLIENTE	8583	1902	juan sebastian oie suamil
11	1449686	CLIENTE	8452	5054	juan sebastian tr menethi
12	1449948	CLIENTE	2418	2998	juan sebastian we jojor
13	1450003	CLIENTE	2811	4525	juan sebastian op peru
14	1450003	CLIENTE	2811	4525	juan sebastian op peru
15	1450003	CLIENTE	2811	4525	juan sebastian op peru
16	1450003	CLIENTE	2811	4525	juan sebastian op peru
17	1450003	CLIENTE	2811	4525	juan sebastian op peru
18	1450025	CLIENTE	2120	8220	juan sebastian op
19	1450025	CLIENTE	2120	8220	juan sebastian op
20	1450072	CLIENTE	3774	9197	juan sebastian juj ramirez
21	1450268	CLIENTE	6509	6623	juan sebastian hul kull

Plan:



Salida de Script x Resultado de la Consulta x Resultado de la Consulta 1 x Resultado de la Consulta 2 x Resultado de la Consulta 3 x Explicación del Plan x

SQL | 0,105 segundos

OPERATION	OBJECT_NAME	CARDINALITY	COST	
SELECT STATEMENT			811	1711
HASH JOIN			811	1711
Access Predicates				
PERSONA.USUARIO_ID=ORDEN.ID_PER				
NESTED LOOPS		811		1711
NESTED LOOPS		811		1711
STATISTICS COLLECTOR				
HASH JOIN		811		88
Access Predicates				
ITEMS_ORDEN.ITEMS_IC				
HASH JOIN		811		20
Access Predicates				
ORDEN.ID=ITEMS_C				
TABLE ACCESS (FULL) ITEMS_ORDEN		811		13
VIEW index\$_join\$_002		1129		7
HASH JOIN				
Access Predicates				
ROWID=RO				
INDEX (FAST FULL) ORDER_PK		1129		4
INDEX (FAST FULL) ORDER_INDEX1		1129		5
TABLE ACCESS (FULL) ITEMS		12523		68
Filter Predicates				
ITEMS.PRECIO>=37				
INDEX (UNIQUE SCAN) PERSONA_PK		1		1
Access Predicates				
PERSONA.USUARIO_ID=ORI				
TABLE ACCESS (BY INDEX ROWID) PERSONA		1		2
TABLE ACCESS (FULL) PERSONA		1		2
Other XML				

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 específicas, 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.