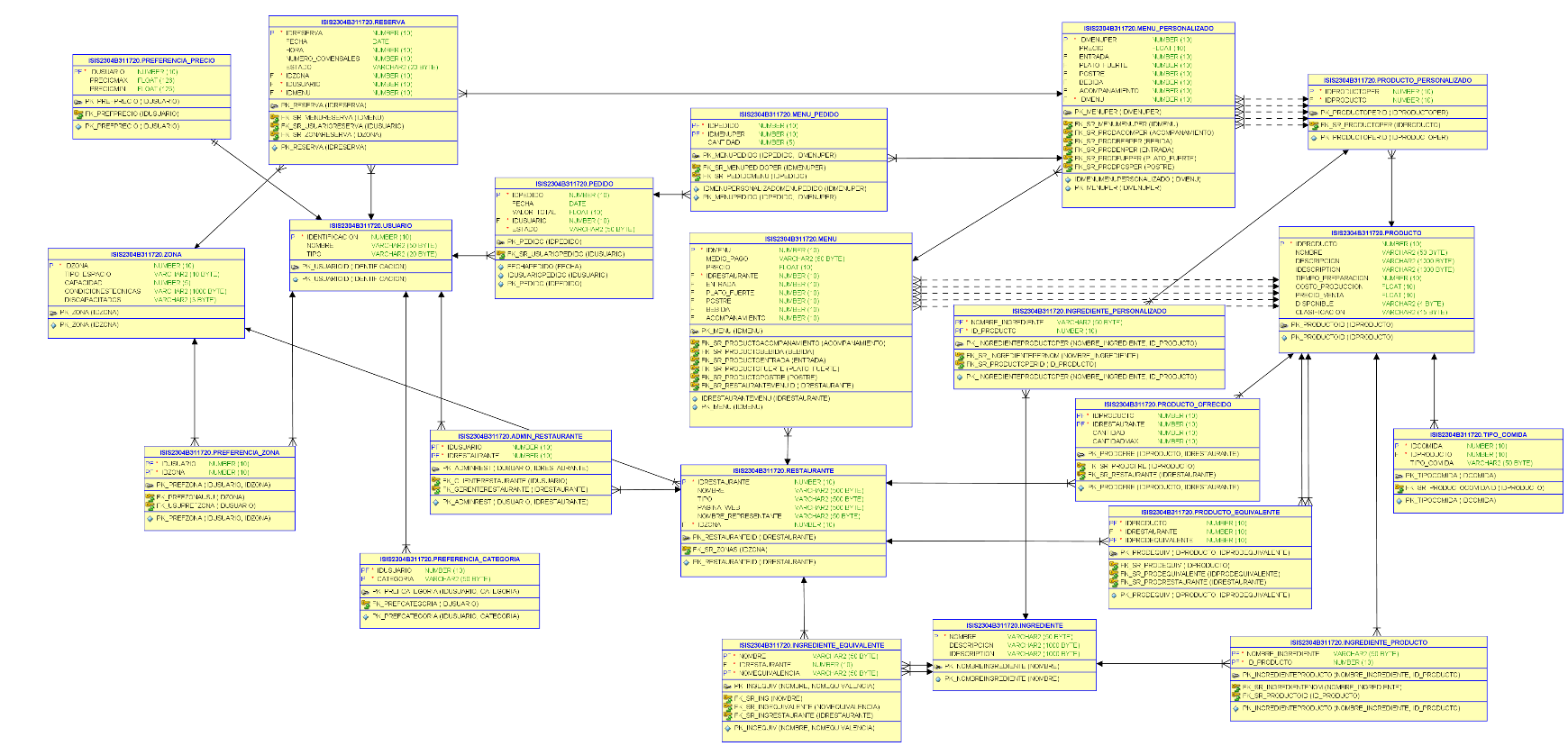
# Sistemas transaccionales – Caso de estudio Rotond Andes

Iteración 4

1-Algunos de los cambios realizados con respecto a las iteraciones pasadas para cumplir con los objetivos de la nueva fueron:

* Se agregó una nueva tabla que relaciona un administrador con un restaurante para cumplir con una de las condiciones del requerimiento nueve. Es decir, para asociar un administrador con los restaurantes que administra el mismo y que solamente pueda consultar el consumo del restaurante del cual es administrador.
* Se extendió la capacidad de las descripciones de productos e ingredientes.
* Se agregó un nuevo rol de usuario llamado AdminRestaurante, el cual representa un administrador de uno o más restaurantes.
* En las demás tablas no hicimos ningún cambio adicional.

A continuación se tiene el diagrama relacional actual.



2-**Diseño Físico**

-> Justificar la selección de índices para cada uno de los requerimientos funcionales, indicar cuál es el tipo de índice utilizado (B+, Hash, Primario o secundario) Agregar costos.

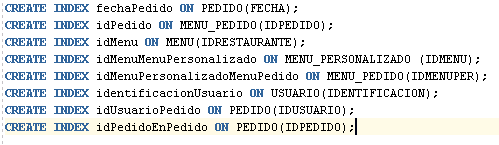
# Selección de índices

Requerimiento 9: Para el requerimiento nueve se utilizaron los siguientes índices:

* Índice en el id de un Pedido de la tabla menú pedido.\*
* Índice en el id de restaurante de la tabla menú.
* Índice en el id de menú de la tabla menú personalizado.
* Índice en el id de menú personalizado en la tabla menú pedido.
* Índice la identificación del usuario.\*
* Índice en el id de usuario de la tabla pedido.
* Índice en el id de pedido de la tabla pedido.\*

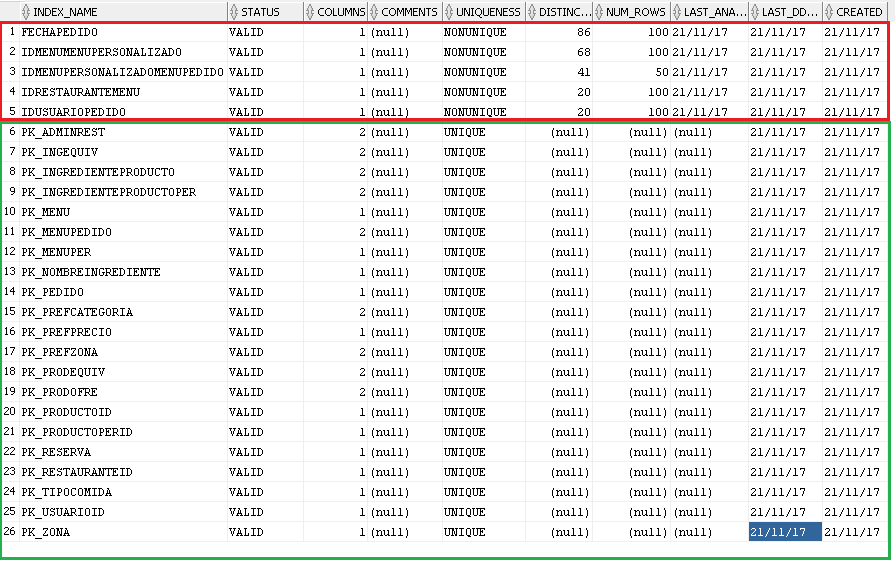
**Nota:** Índices marcados con \* son creados automáticamente por SQLdeveloper por ser o formar parte de la llave primaria.

Los índices fueron creados de la siguiente manera:



Después se llegó a esta configuración para mejorar la eficiencia.

A continuación se muestran los índices que se planearon para el desarrollo del requerimiento número nueve (en rojo). Los que inician con PK\_NAME fueron creados automáticamente por SQL developer (en verde).



Los índices fueron creados por SQL developer sobre las llaves primarias de la tabla, porque la selectividad de estos es igual a la mejor selectividad posible. Para probar esto, analicemos lo siguiente:

**Selectividad = 1 / Cantidad\_de\_filas\_con\_valores\_distintos =** 1/100000 (Las llaves primarias son únicas)

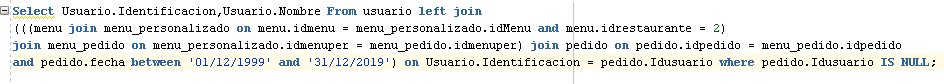
**Mejor\_Selectividad = 1 / 100000 (El número de llaves es el mismo que el de filas de la tabla)**

La selectividad calculada para este ejemplo es de 0.00001 y la mejor selectividad es de 0.00001. Entre más se acerque la selectividad calculada a la mejor selectividad es más eficiente escoger dicho índice, y mucho mejor si es la misma como en el caso anterior. Los índices mejoraron el rendimiento porque en la consulta del requerimiento nueve, se realizan varios joins sobre las llaves primarias, entonces la creación de estos índices permite una consulta más óptima pues en este caso los índices disminuyen el costo de búsqueda, entonces se puede decir que la decisión que tomó SQL developer para asignar índices en las llaves primarias mejoró el rendimiento de la búsqueda.

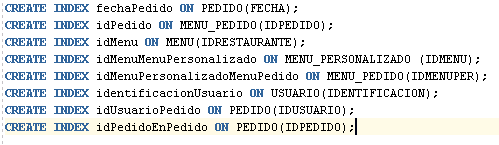
Requerimiento 10: Para el requerimiento diez se utilizaron los siguientes índices:

* Índice en el id de un Pedido de la tabla menú pedido.\*
* Índice en el id de restaurante de la tabla menú.
* Índice en el id de menú de la tabla menú personalizado.
* Índice en el id de menú personalizado en la tabla menú pedido.
* Índice la identificación del usuario.\*
* Índice en el id de usuario de la tabla pedido.
* Índice en el id de pedido de la tabla pedido.\*

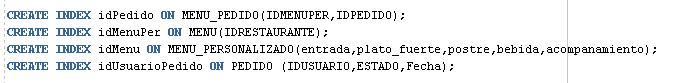
**Nota:** Índices marcados con \* son creados automáticamente por SQLdeveloper por ser o formar parte de la llave primaria.



Los índices fueron creados de la siguiente manera:



Después se llegó a esta configuración para lograr una mejor eficiencia.



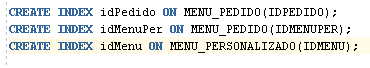
Dado que el requerimiento nueve y diez son similares, se puede observar que los índices escogidos para el requerimiento diez son los mismos del requerimiento nueve. Se escogieron estos índices para lograr una mejor eficiencia porque son los principales atributos que se involucran en las dos consultas.

Requerimiento 11: Para el requerimiento diez se utilizaron los siguientes índices:

* Índice en el id de Índice en el id de pedido en la tabla menú pedido.\*
* Índice en el id de menú personalizado menú pedido.
* Índice en el id de menú de la menú personalizado.

**Nota:** Índices marcados con \* son creados automáticamente por SQLdeveloper por ser o formar parte de la llave primaria.

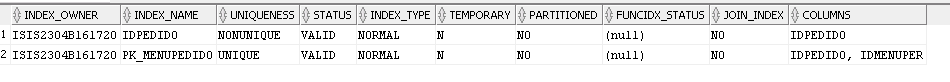
Los índices se crearon de la siguiente manera:



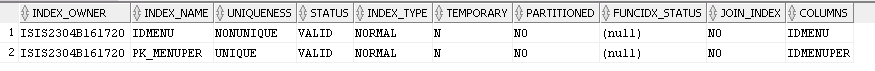
Dado que el requerimiento once implica la utilización de los ids de los pedidos, de los menús personalizados y de los menús en general, se decidió crear los anteriores índices porque en la sentencia SQL que responde al requerimiento se hacen varios joins sobre estos atributos, entonces, de esta forma se disminuiría el costo de la consulta.

La información de los índices creados para este requerimiento es la siguiente:

En la tabla menú pedido:



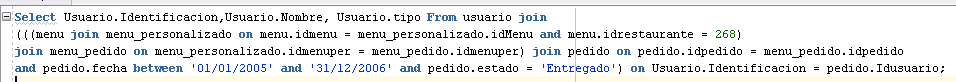
En la tabla menú personalizado:



# Requerimiento 9:

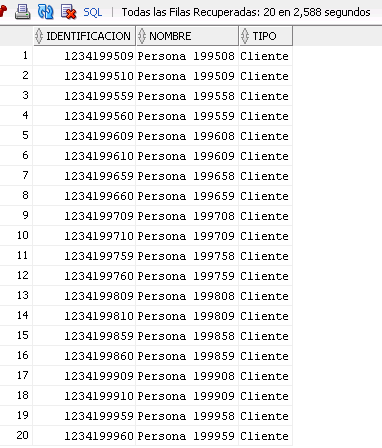
## Escenario de prueba:

Se desea consultar la identificación y el nombre de los usuarios que al menos hicieron un pedido (que fue entregado) en el restaurante con id 268 en el rango de fechas '01/01/2005' y '31/12/2006’. Dicha consulta se realiza con la siguiente sentencia SQL1:



Donde se tienen tres parámetros que son: el id del restaurante al cual se desea consultar el consumo, y las dos fechas que representan el rango por el que se desea hacer la consulta.

La anterior sentencia SQL retorna como respuesta los siguientes 20 usuarios.

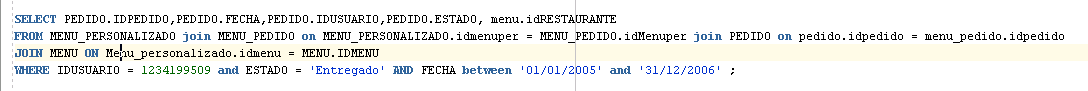


## Para demostrar el funcionamiento de la anterior consulta veamos el siguiente ejemplo con algunos datos que se tenían en la tabla:

1. Se tienen 200.000 Usuarios registrados en la base de datos. Teniendo en cuenta la tabla anterior, vamos a observar uno de los 20 usuarios mostrados. El usuario con id 1234199509 con nombre “Persona 199508”.



1. Vamos a consultar los pedidos que se fueron realizados por dicho usuario en dicho rango de fechas en.





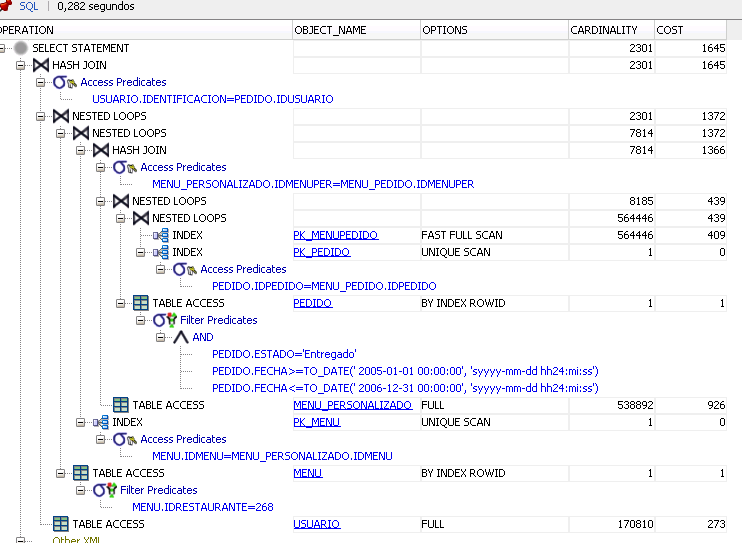
Se puede observar que se realizaron 1 pedidos (en estado entregado) dentro del rango de fechas '01/05/2017' y '31/12/2017’ por dicho usuario en el restaurante con id 268. Se pueden concluir que dicho usuario hizo al menos un pedido en el restaurante con id 268 en el rango de fechas establecido, resultado que se evidencia en la respuesta de la sentencia que se planteó para la solución, es decir debería retornar el usuario con id 1234199509 y efectivamente se encuentra en la respuesta.

## Distribución de los datos

Se tiene: 200.000 usuarios, 200.000 productos, aproximadamente 600.000 pedidos, 800 restaurantes, y 600.000 menús.

## Plan de consulta generado por SQL Developer

Se puede observar que en el plan que ejecutó SQL developer en primer lugar se utiliza un hash join, varios nested loops, y finalmente el acceso a las tablas por medio de los índices que SQL developer genera de forma automática. Se puede observar que el costo de la operación es de 1645.



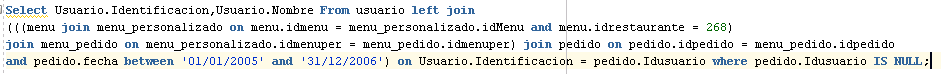
## Tiempo obtenido con la ejecución del plan

El tiempo que se tardó en ejecutar la sentencia y consultar las dos tuplas fue de 0.282 Segundos.  Y el tiempo de la consulta es alrededr de 4 segundos.

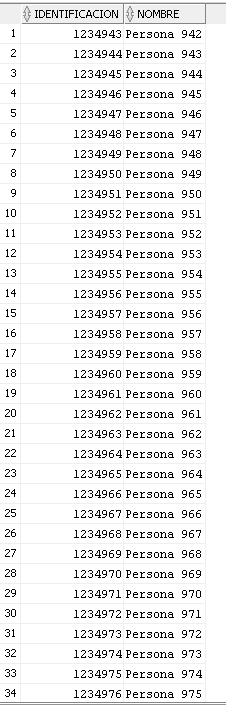
# Requerimiento 10:

## Escenario de prueba:

Se desea consultar la identificación, el nombre y el tipo de los usuarios que no hicieron pedidos o que hicieron pedidos pero ninguno fue entregado, en el restaurante con id 268 en el rango de fechas '01/01/2005' y '31/12/2006’. Dicha consulta se realiza con la siguiente sentencia SQL2:



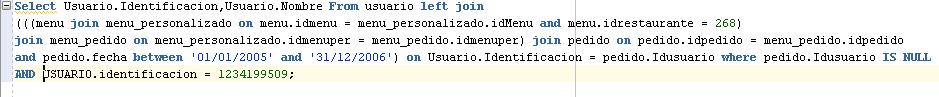
Que retorna como respuesta:



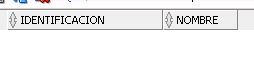
En realidad son demasiados usuarios para mostrarlos en pantalla, sin embargo el resultado de esta consulta debería ser la opuesta a la anterior, pues en la anterior se muestran los usuarios que al menos hicieron un pedido, entonces en esta se deberían mostrar los que no hicieron ningún pedido.

## Para demostrar el funcionamiento de la consulta:

1. Se debe tener en cuenta que la primera consulta nos dio como resultado un solo usuario con id 1234199509, el cual solamente realizó un pedido el cual le fue entregado. Entonces, el resultado de esta consulta debería ser todos los usuarios restantes. Para probar esto, vamos a consultar desde la tabla que nos dio como respuesta la sentencia **SQL2** si existe un registro del usuario con id1234199509, usando la siguiente sentencia SQL:



Que da como resultado:



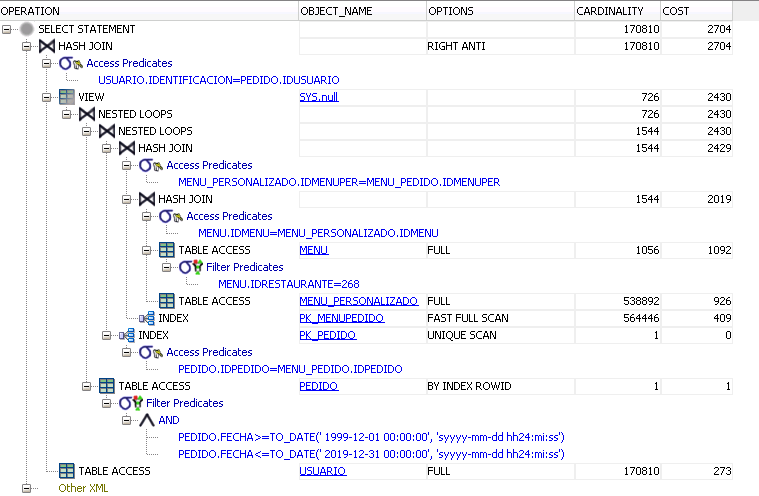
Es decir, que el resultado es coherente, pues no debería existir un registro del usuario con id 1234199509, ya que el hizo al menos un pedido en ese rango de fechas entonces no debería aparecer en la anterior consulta.

## Distribución de los datos

Se tiene: 200.000 usuarios, 200.000 productos, aproximadamente 600.000 pedidos, 800 restaurantes, y 600.000 menús.

## Plan de consulta generado por SQL Developer

Se puede observar que en el plan que ejecutó SQL developer en primer lugar se utiliza un hash join, varios nested loops, y finalmente el acceso a las tablas por medio de los índices que SQL developer genera de forma automática. Se puede observar que el costo de la operación es de 2704.

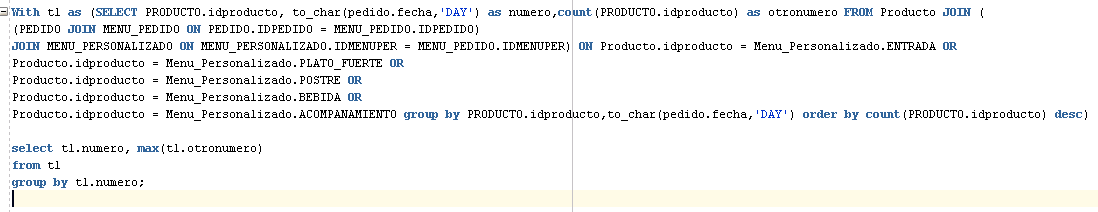


## Tiempo obtenido con la ejecución del plan

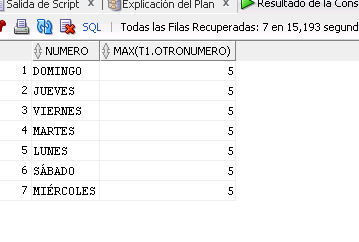
El tiempo que se tardó en ejecutar la sentencia y consultar las dos tuplas fue de 0.282 Segundos. . El tiempo de la consulta fue de masomenos 5 segundos.

## Requerimiento 11: En el requerimiento 11 se desea consultar el restaurante y el producto más frecuentado y vendido respectivamente.

**Para desarrollar este requerimiento utilizamos la siguiente sentencia SQL:**



**Que da como respuesta lo siguiente:**



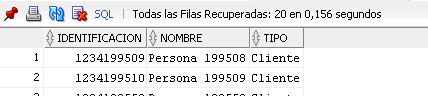
**Requerimiento 12:**

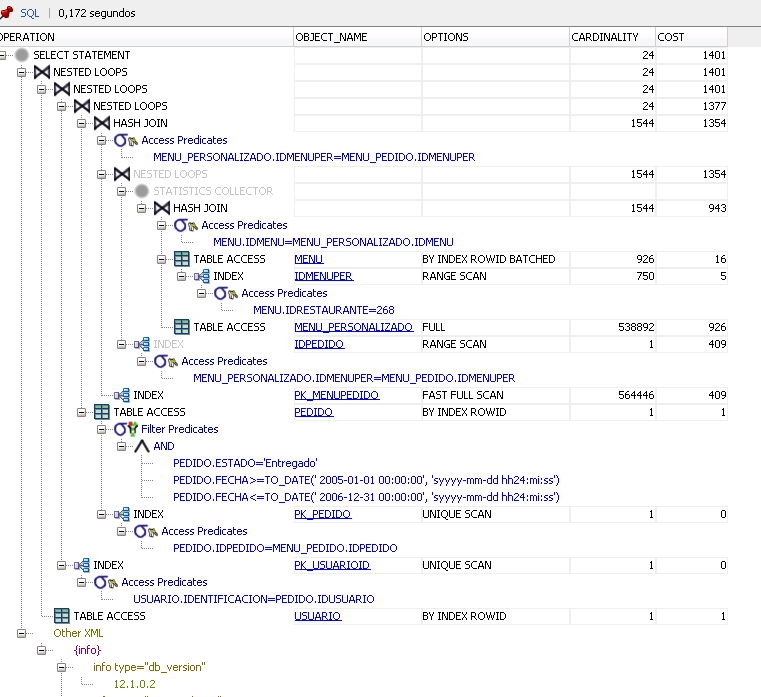
**Sentencia utilizada:**

select usuario.\* from usuario join pedido on pedido.idusuario = usuario.identificacion join MENU\_PEDIDO on MENU\_PEDIDO.IDPEDIDO = pedido.IDPEDIDO left join (select IDMENUPER from MENU\_PERSONALIZADO where ACOMPANAMIENTO is null)t1 on t1.IDMENUPER = MENU\_PEDIDO.IDMENUPER;

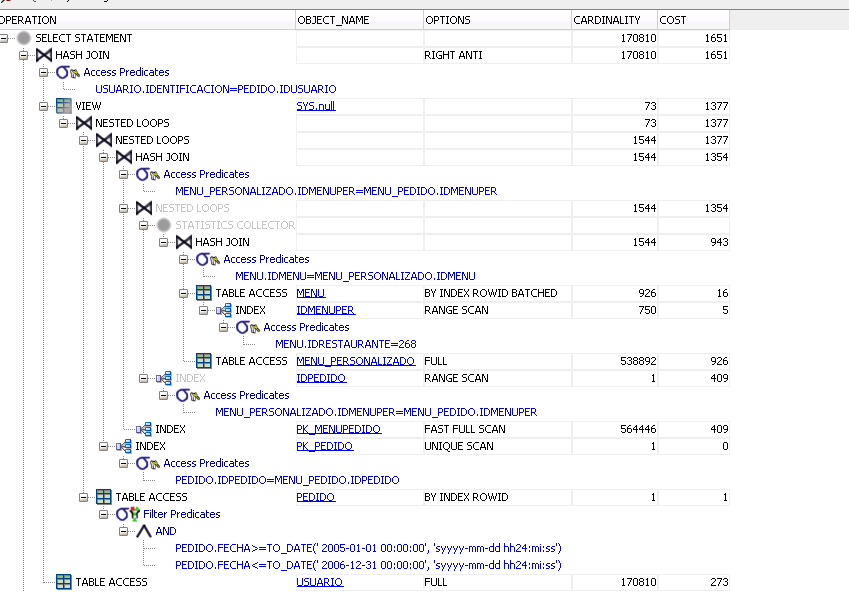
# Análisis de eficiencia

## Requerimiento 9: Para analizar la eficiencia después de haber establecido los índices vamos a ejecutar la sentencia SQL del primer requerimiento para compararla con la consulta sin índices. Al ejecutar la sentencia que planteamos para el requerimiento 9 después de haber creado los índices, se tiene un costo de 1401 que es menor al de la consulta ejecutada sin índices. Se puede observar, que los índices creados afectaron un poco de manera positiva el costo de la primera consulta. Además, hubo una disminución en el tiempo de la consulta pues antes era de más o menos 4 segundos y con índices este tiempo se redujo a 0.156 segundos.

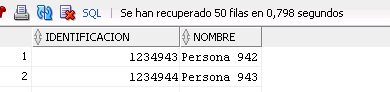




## Requerimiento 10: Para analizar la eficiencia después de haber establecido los índices vamos a ejecutar la sentencia SQL del segundo requerimiento para compararla con la consulta sin índices. Al ejecutar la sentencia que planteamos para el requerimiento 10 después de haber creado los índices, se tiene un costo de 1651 que es menor al de la consulta ejecutada sin índices. Se puede observar, que los índices creados afectaron un poco de manera positiva el costo de la primera consulta. Además, hubo una disminución en el tiempo de la consulta pues antes era de más o menos 5 segundos y con índices este tiempo se redujo a 0.798 segundos.



El tiempo en el que se desplego la respuesta del requerimiento número diez fue de 0.798 segundos, que considerablemente disminuyó en comparación con la consulta sin índices.



Documentar como fue el proceso de carga de datos, como se realizó como se logró el volumen de datos esperado.

# El proceso para cargar datos:

Se escribió un programa en eclipse que genera una cantidad bastante considerable de datos para realizar la prueba de eficiencia, a continuación se muestra una captura de pantalla del código:



