**Diseño de la aplicación**

**Cambios respecto a la anterior iteración:**

No se incorporaron nuevos cambios a la aplicación, se intentó hacer la menor cantidad de cambios posibles debido a la complejidad de la iteración; esto, con el objetivo de reducir el tiempo en el desarrollo de la aplicación. Se dio un enfoque a crear datos semi-reales y con la suficiente variedad y tamaño para tener resultados variados entre consultas. También, se le dio énfasis a generar la suficiente cantidad de datos en la población de las tablas.

**Diseño físico de la aplicación:**

**Índices y requerimientos funcionales:** Debido a la gran cantidad de recursos generados, es necesario optimizar la base de datos para poder responder más rápidamente. De aquí nace el uso de los índices para optimizar las consultas establecidas. Como cada tabla requiere de diferentes tablas y campos, se explicarán los índices utilizados sobre cada consulta. El código SQL de cada sentencia se encuentra en SQL/RFC Iteración 4.sql para no saturar el documento.

* **RFC 9:**

En este requerimiento era necesario conocer los datos de los usuarios a partir de ciertos patrones en las compras, en particular el restaurante donde se compra en determinado rango de fechas. Para agrupamiento y ordenamiento, lo hacemos sobre los campos del usuario y los de producto.

Esta consulta hace uso de los campos en las tablas CHECKOUT y PRODUCTO\_CHECKOUT, sobre los campos TIEMPOR (fecha), ID\_CHECKOUT y USUARIO\_CLIENTE por lo que se hace necesario implementar índices sobre esas tablas y esos campos. Se decidió utilizar un índice compuesto sobre los campos ID y USUARIO\_CLIENTE en la tabla CHECKOUT y uno simple sobre USUARIO\_CLIENTE en la tabla PRODUCTO\_CHECKOUT puesto que son las columnas que se usan en el Join y están en el rango de fechas que se busca en el WHERE, esto también debido a que la selectividad de los campos es alta.

* **RFC 10:**

De manera similar al anterior, se requería consultar el complemento del conjunto que representa la consulta anterior, por lo que los campos involucrados son los mismos. Este requerimiento no era una modificación trivial del otro, puesto que si se usa NOT IN o similares en algún lugar de la consulta, se anulaba la utilidad de cualquier índice incrementando los tiempos de consulta. Por esta razón decidimos usar operaciones entre conjuntos, utilizando el operador de resta de conjuntos que ofrece SQL, MINUS. A pesar de esto, la consulta resultó en un mayor tiempo de ejecución que usando el IS NOT IN. Debido a esto, solamente se usan los índices de llave primaria para la localización de los registros.

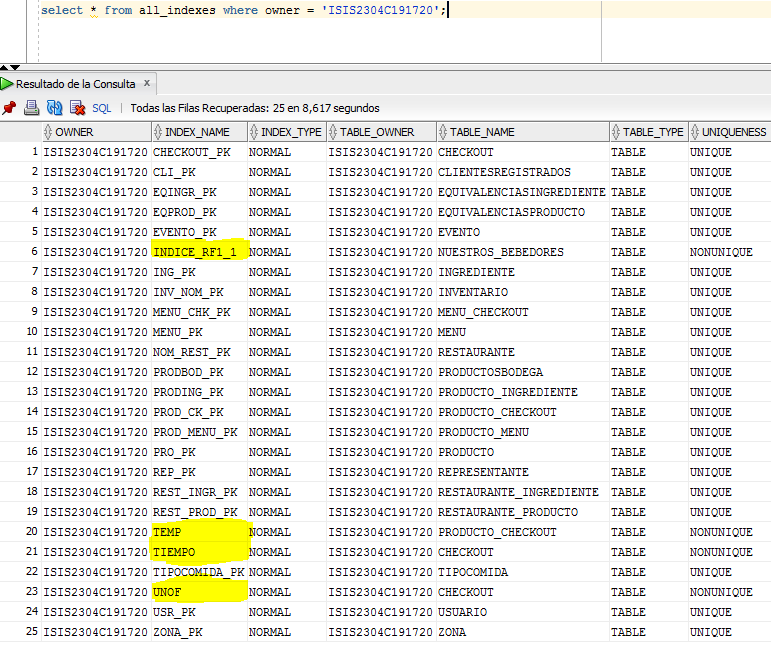
* **RFC 11:**

Para este requerimiento era necesario efectuar varias funciones agregadas sobre las tablas encargadas de modelar las ventas después de diferentes agrupamientos/modificaciones sobre CHECKOUT y PRODUCTO\_CHECKOUT. Por otra parte, para las funciones de conteo, un índice de Bitmap es mejor puesto que la cardinalidad de la categorización que usamos para las fechas es 1/7 (7 días de la semana), lo cual es bastante útil al categorizar más de 1 millón de compras.

* **RFC 12:**

El último requerimiento nos solicitaba extraer la información de aquellos clientes que cumplan varias condiciones, por lo que es necesario hacer un JOIN de las diferentes condiciones para ser ‘un buen cliente’.

**Índices creados automáticamente por Oracle:**

* 

Acá se aprecia cuantos índices crea por defecto Oracle (aquellos que no están resaltados). Es evidente que por cada tabla en la BD, se crea un índice sobre su PK ya que por defecto se busca que los registros queden en orden físico y un índice sobre la llave primaria (Clustered) hace esto posible cada vez que se crea una tabla. Debido a que de entrada ya se tiene un orden sobre los registros (sobre la PK), estos índices mejoran cualquier consulta que se haga sobre la base de datos debido a que acceder a los registros no es necesario buscarlos sino que se puede acceder a ellos a través del índice.

**Análisis de requerimientos funcionales:**

**RFC 9:**

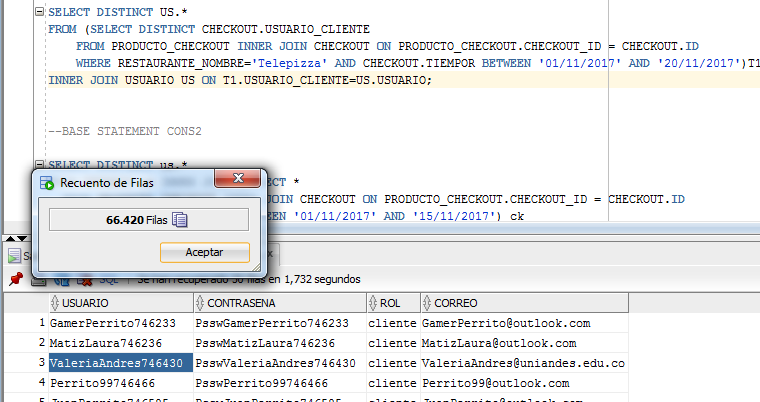
**Documentación de los datos y escenarios de pruebas:**

* **Sentencias SQL:**

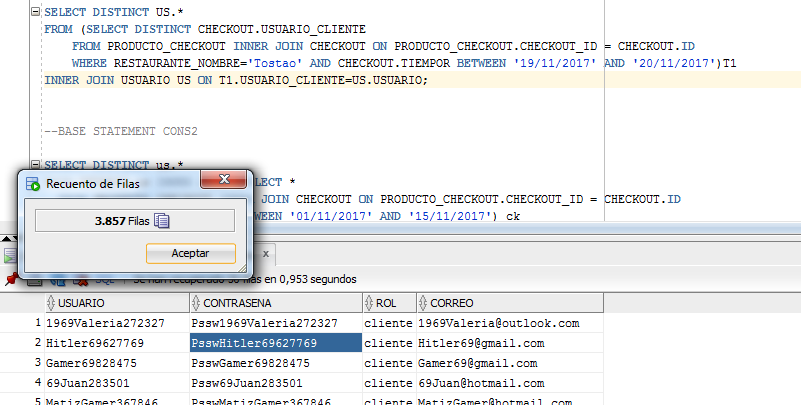
****

* **Parámetros y resultados:**

Parámetros 1: ‘Telepizza’, (1 mes)

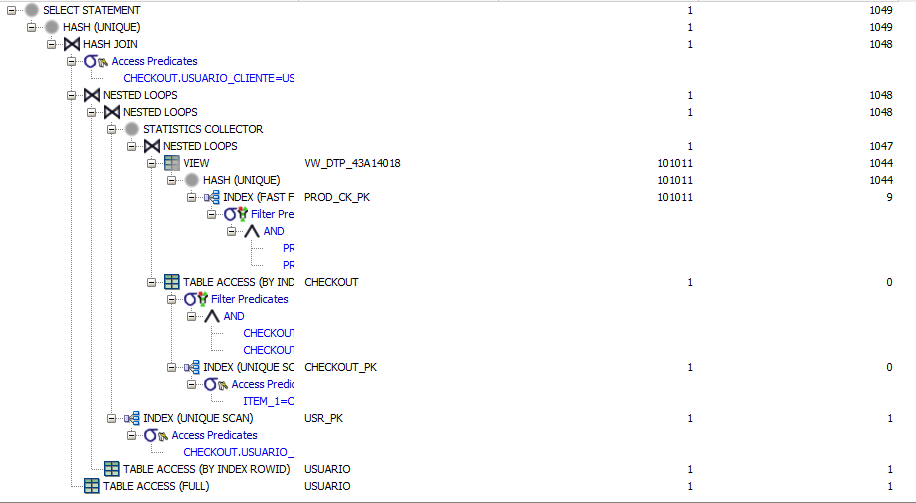


Parámetros 2: ‘Tostao’, (1 día)

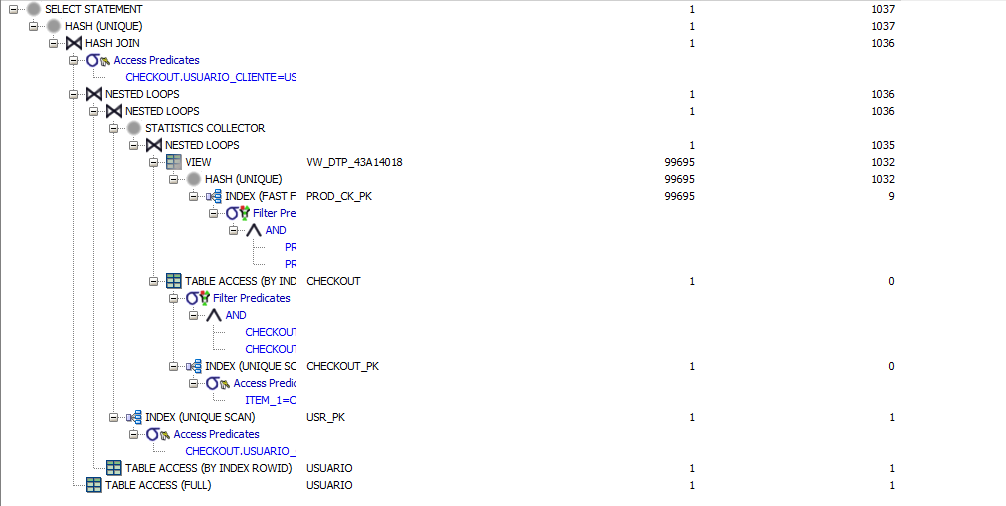


* **Parámetros y planes:**

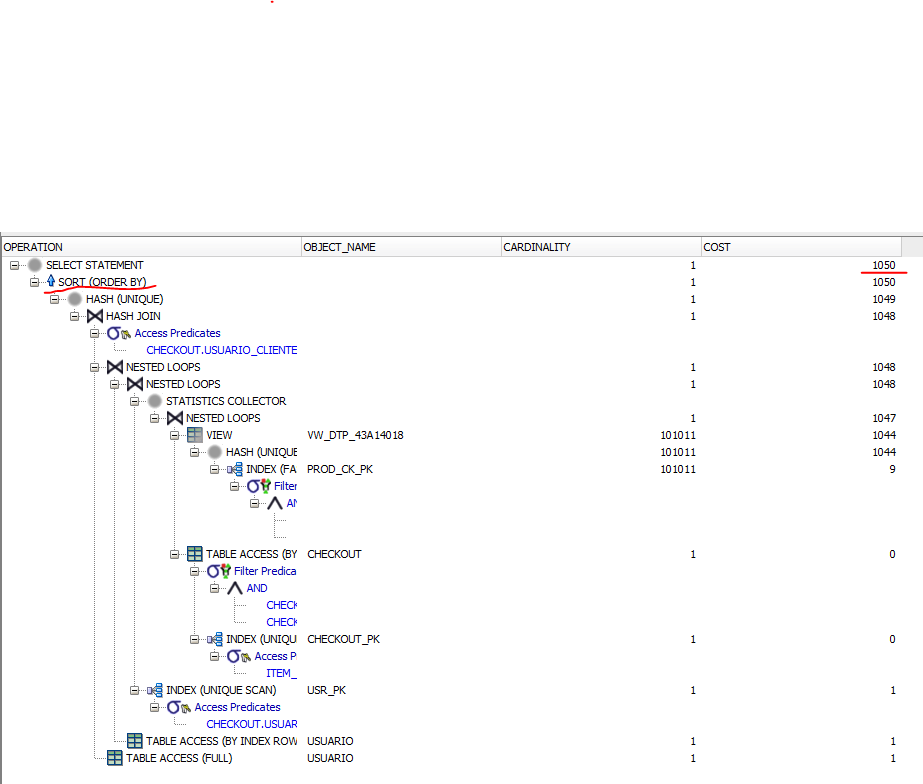
Parámetros 1: ‘Telepizza’, (1 mes)



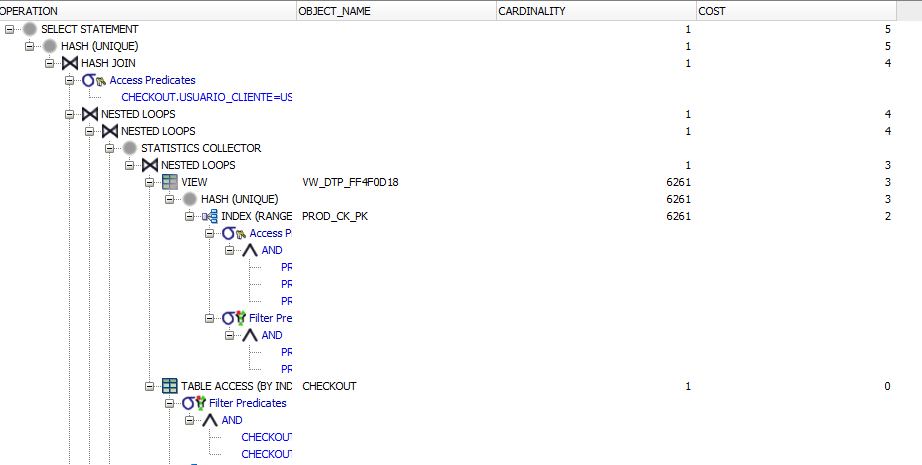
Parámetros 2: ‘Tostao’, (1 día)



* **Planes generados:**

Order by

**PLAN 2 FILTRO - AGRUPAMIENTO******



* **Tiempos de ejecución y Análisis de eficiencia:**

Se pude apreciar que los tiempos están dentro del rango permitido, teniendo en cuenta que los datos de las tablas superan los 5 millones. En términos de cada tabla, la cantidad de datos es del orden de un millón (teniendo en cuenta las tablas de las consultas) este factor altera el exponente de tiempo en algunos casos, ya que el producto cartesiano, el JOIN y otras operaciones de conjuntos, tienen una incidencia directamente proporcional al tiempo de las consultas.

**RFC 10:**

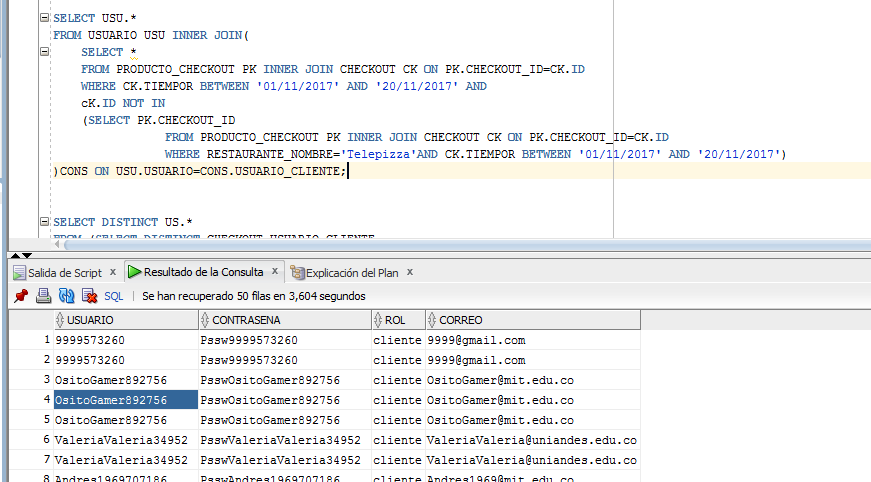
**Documentación de los datos y escenarios de pruebas:**

* **Sentencias SQL:**

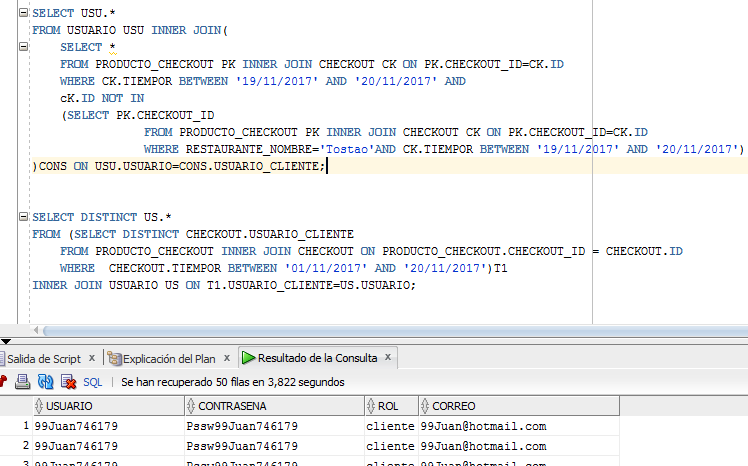
****

* **Parámetros y resultados:**

Parámetros 1: ‘Telepizza’, (1 mes)

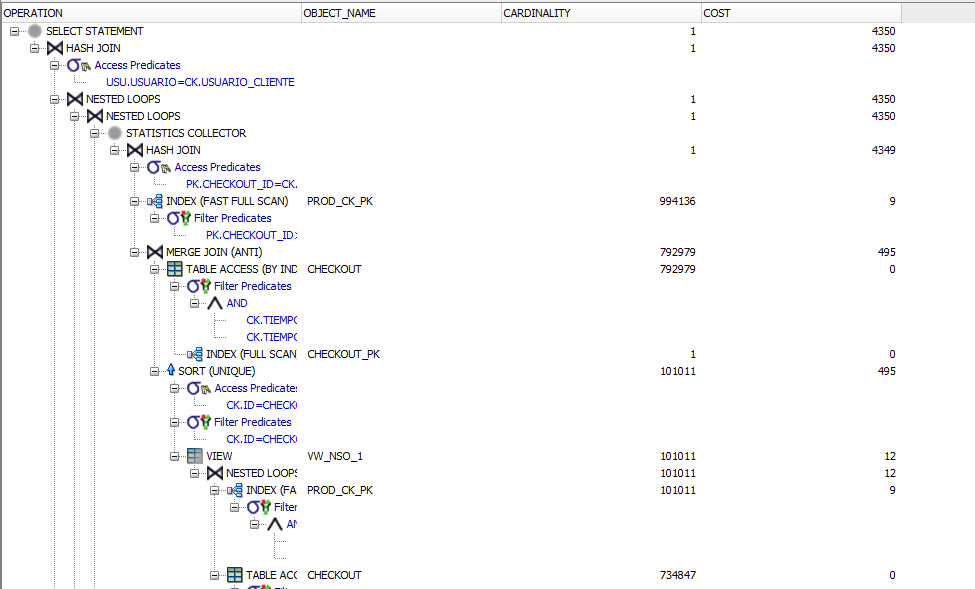


Parámetros 2: ‘Tostao’, (1 día)

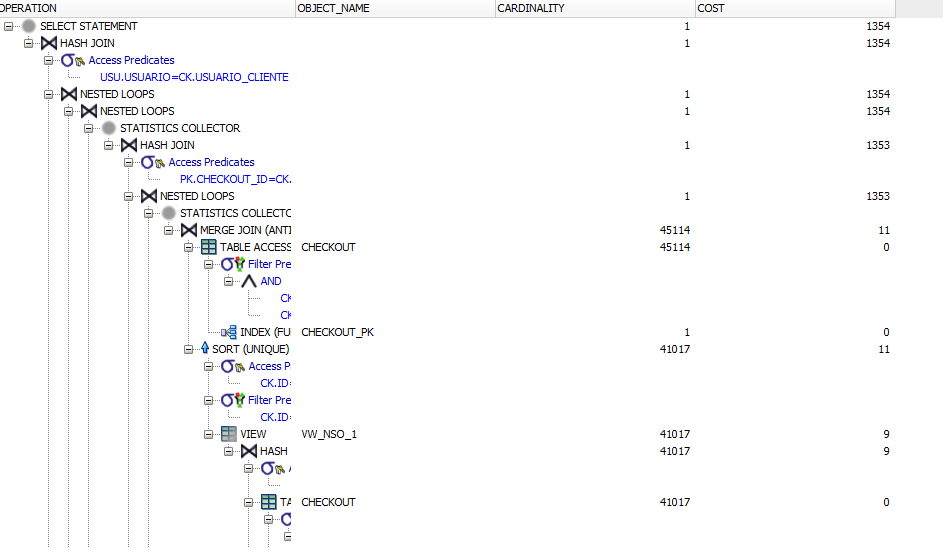


* **Parámetros y planes:**

Parámetros 1: ‘Telepizza’, (1 mes)

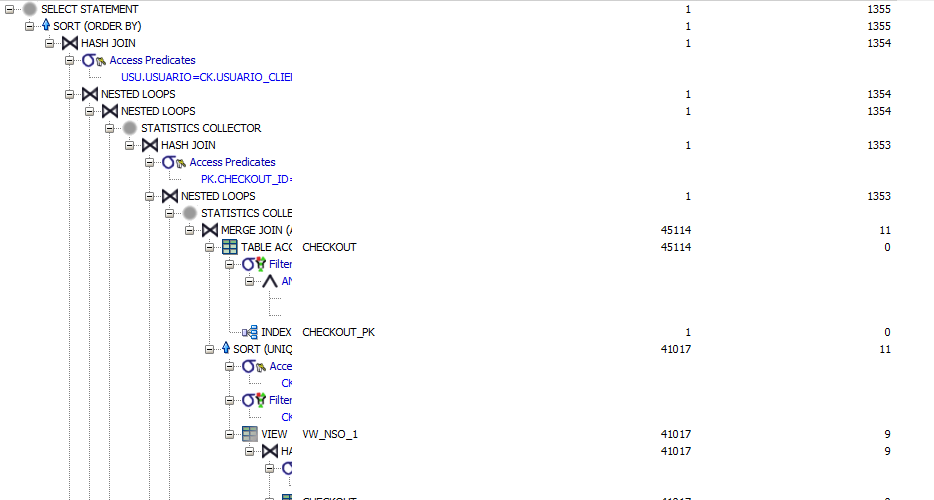


Parámetros 2: ‘Tostao’, (1 día)

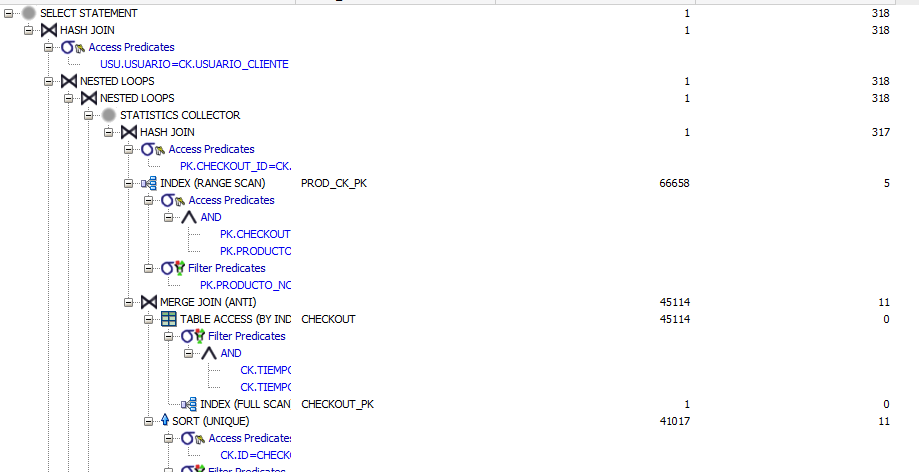


* **Planes generados:**

**Order by**



**PLAN 2 FILTRO - AGRUPAMIENTO**



* **Tiempos de ejecución y Análisis de eficiencia:**

Se pude apreciar que los tiempos no están dentro del rango permitido, teniendo en cuenta que los datos de las tablas superan los 5 millones. En términos de cada tabla, la cantidad de datos es del orden de un millón (teniendo en cuenta las tablas de las consultas) este factor altera el exponente de tiempo en algunos casos, ya que el producto cartesiano, el JOIN y otras operaciones de conjuntos, tienen una incidencia directamente proporcional al tiempo de las consultas. Ahora bien, durante esta consulta el proceso a seguir es conseguir un conjunto general al cual se le restan los elementos en otro sub conjunto contenido en el primero, para este propósito es necesario utilizar en repetidas ocasiones operadores de JOIN lo cual aumenta el exponentes de costos y tiempo sin embargo no sube considerablemente.

**RFC 11:**

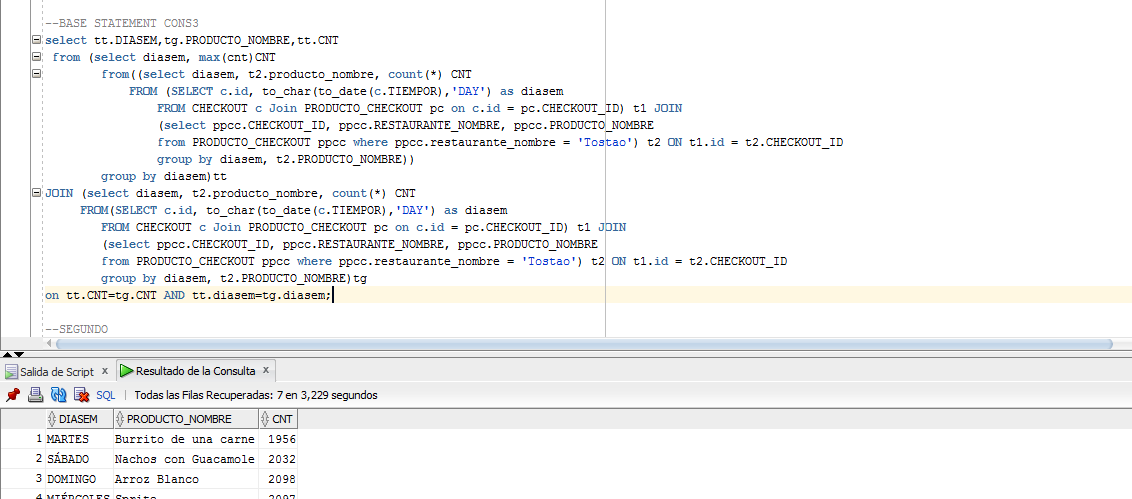
**Documentación de los datos y escenarios de pruebas:**

* **Sentencias SQL:**

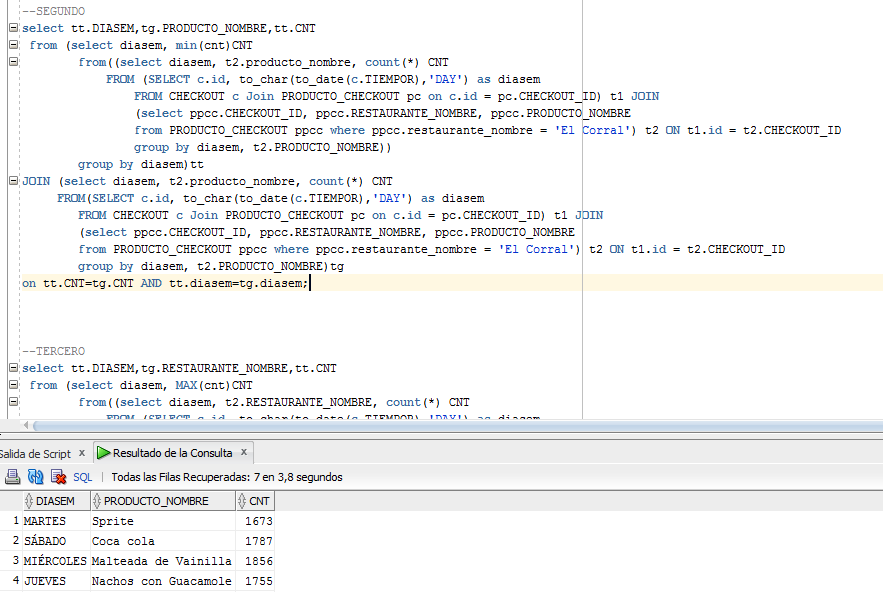
****

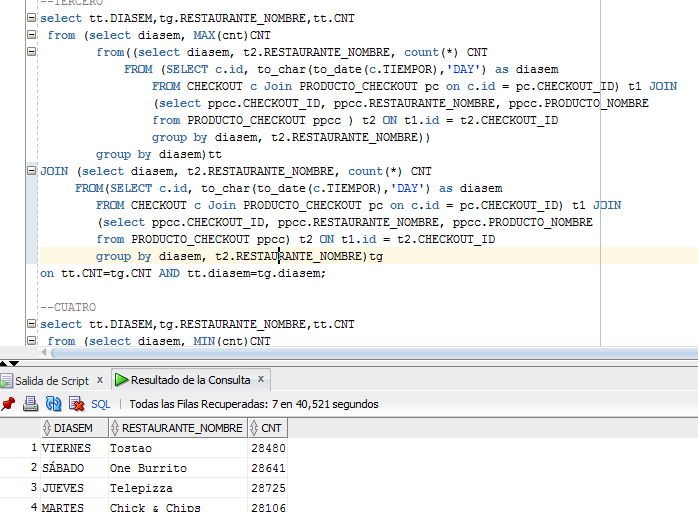
* **Parámetros y resultados:**

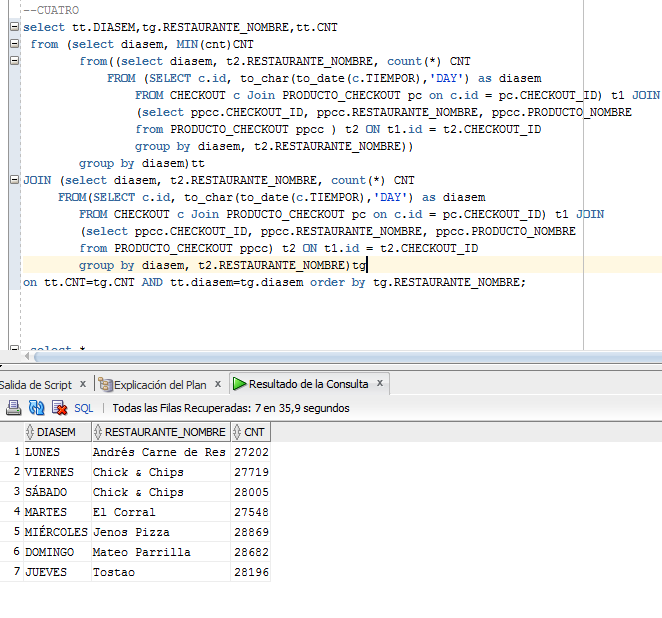
**Parámetro: ‘Tostao’**



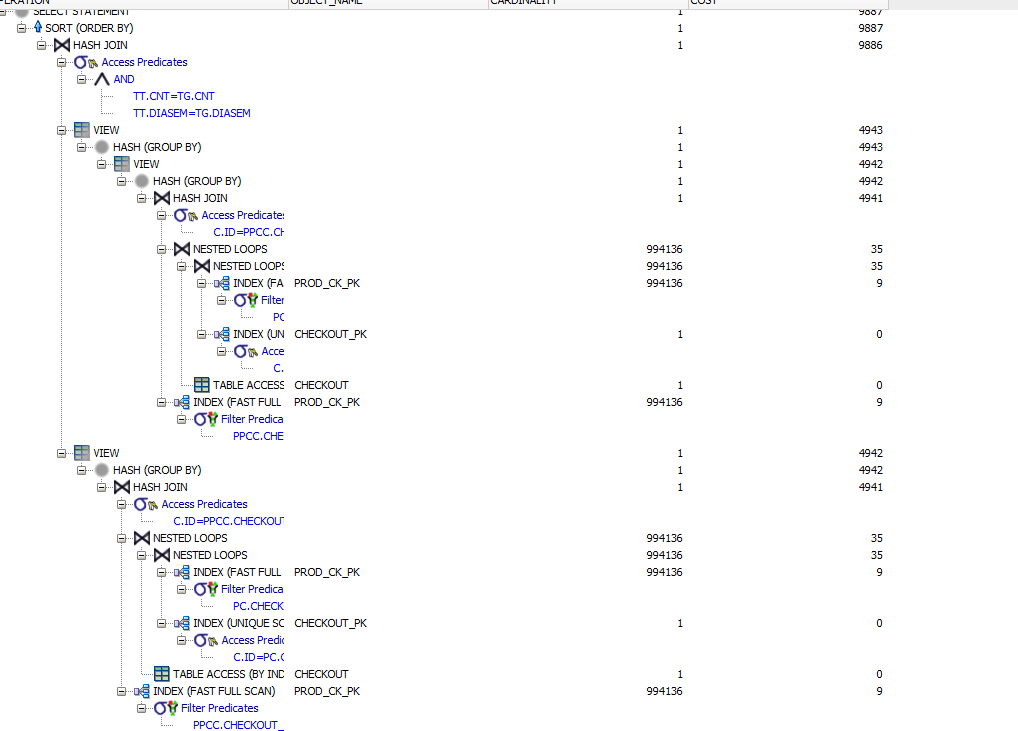
**Parámetro: ‘El Corral’**

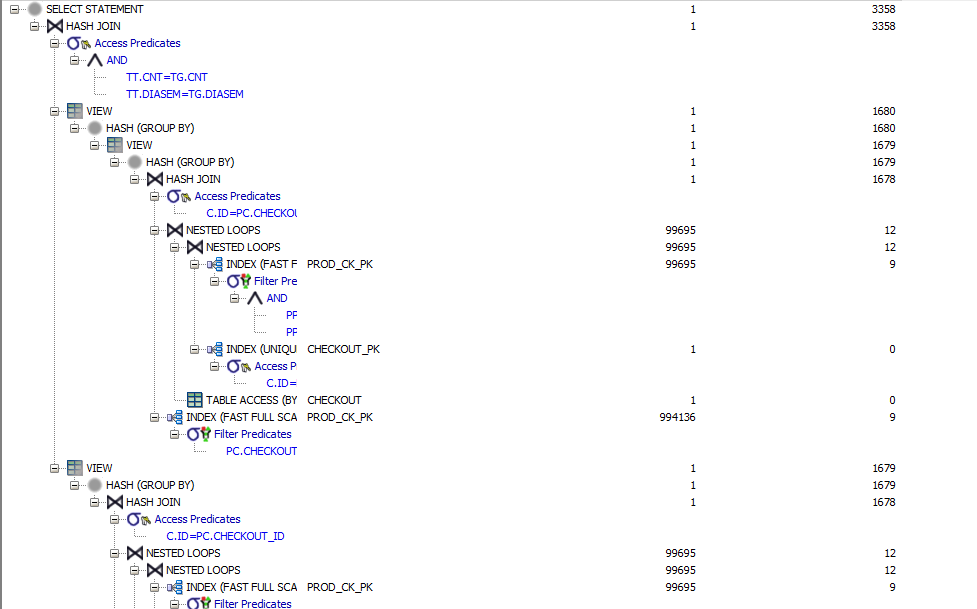






* **Parámetros y planes:**

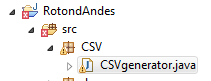




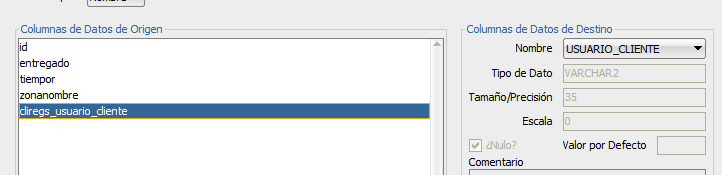
* **Tiempos de ejecución y Análisis de eficiencia:**

Se pude apreciar que los tiempos no están dentro del rango permitido, teniendo en cuenta que los datos de las tablas superan los 5 millones. En términos de cada tabla, la cantidad de datos es del orden de un millón (teniendo en cuenta las tablas de las consultas) este factor altera el exponente de tiempo en algunos casos, ya que el producto cartesiano, el JOIN y otras operaciones de conjuntos, tienen una incidencia directamente proporcional al tiempo de las consultas. Ahora bien, durante esta consulta el proceso a seguir es conseguir un conjunto general al cual se le unen elementos en otro sub conjunto contenido en el primero, para este propósito es necesario utilizar en repetidas ocasiones operadores de JOIN lo cual aumenta el exponentes de costos al limitar la búsqueda inicialmente al producto, gracias a los índices utilizados se consiguen costos menores y tiempos razonables para la cantidad de datos. Finalmente, las consultas de restaurantes por otro lado tienen un plan costoso que no fue posible reducir en costos y tiempo por el mal funcionamiento de un índice hash para los tiempos de compras.

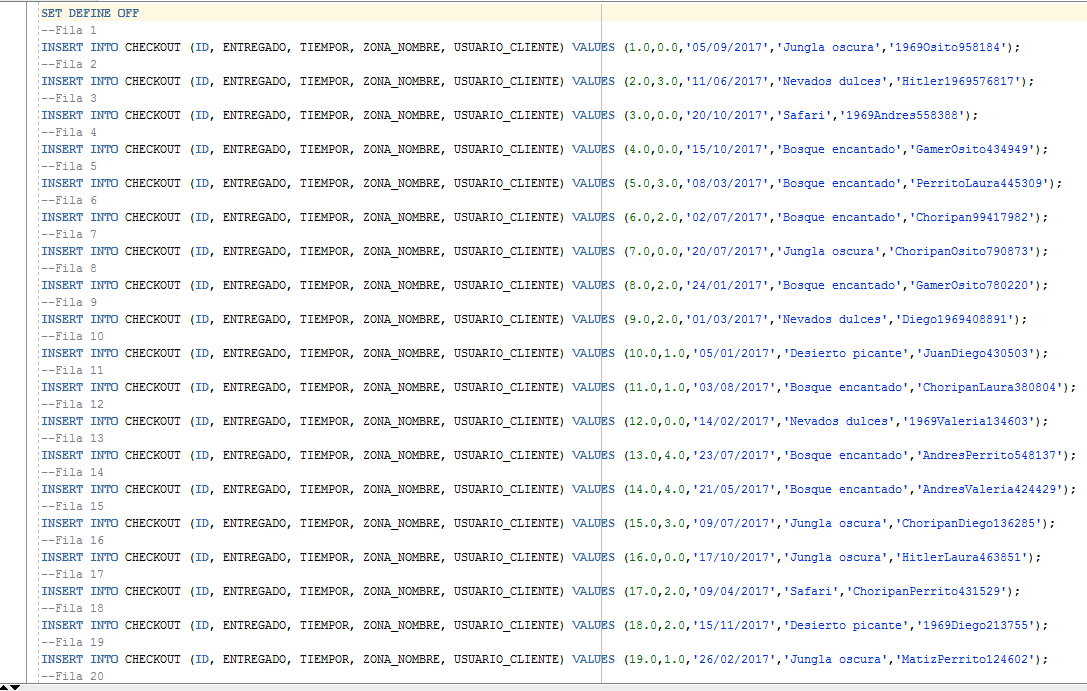
**Documentación del proceso de carga de los archivos:**

Para cargar los archivos se desarrolló un programa en java en el que se teniendo un ‘pool’ de palabras en memoria se generan datos a partir de combinaciones de ellas. Para conservar la unicidad y evitar duplicados se les agregó el id. Se optimizó el manejo de Strings en memoria usando StringBuilder. Los detalles más pequeños del código se pueden ver en el paquete CSV en la clase CSVgenerator.

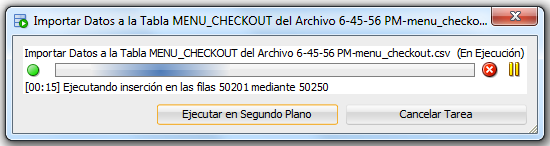
A la hora de cargar los datos en SQL se usó la funcionalidad de importar datos para cada tabla. Como se generó un CSV para cada tabla en la que se iban a ejecutar las consultas, se hizo coincidir cada espacio del CSV con cada campo de la tabla, como se muestra a continuación:



Luego, SQLDeveloper genera el DML correspondiente a la información encontrada en el CSV, como se ve:



Luego de esto, SQLDeveloper se encarga de insertar todos los datos dentro de la base de datos, lo cual puede demorar varios minutos, dependiendo el tamaño del CSV.

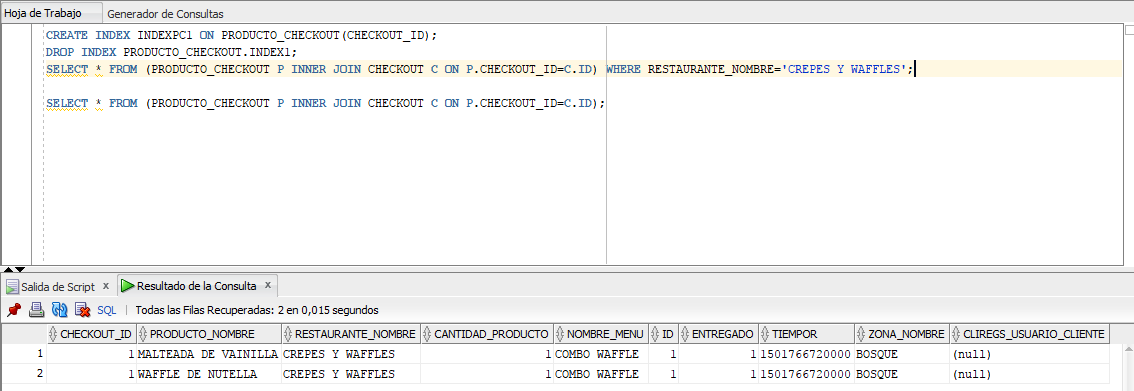


**Análisis del proceso de optimización y el modelo de ejecución de consultas:**

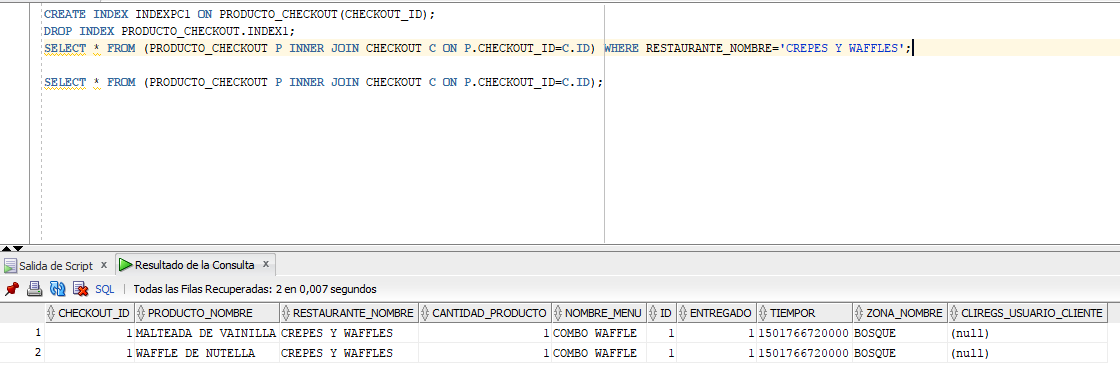
Una de las grandes diferencias entre los dos ambientes de ejecución son las estructuras de datos que usan. En SQL se usan estructuras de datos óptimas para realizar operaciones sobre álgebra relacional y conjuntos, lo que hace mucho más eficiente realizar las operaciones en SQL. También, en el ambiente de la base de datos se encuentra con un optimizador sobre las sentencias de SQL, lo cual aumenta la brecha de eficiencia entre las operaciones de control y las bases de datos. Cabe resaltar también que en el ambiente de operaciones de control hay un gran limitante y es la memoria. Debido a que se maneja una gran cantidad de datos los datos pueden no caber en memoria. Por otro lado las bases de datos contienen algoritmos que funcionan bajo esta restricción, haciendo más evidente la ventaja que tienen estos sistemas.

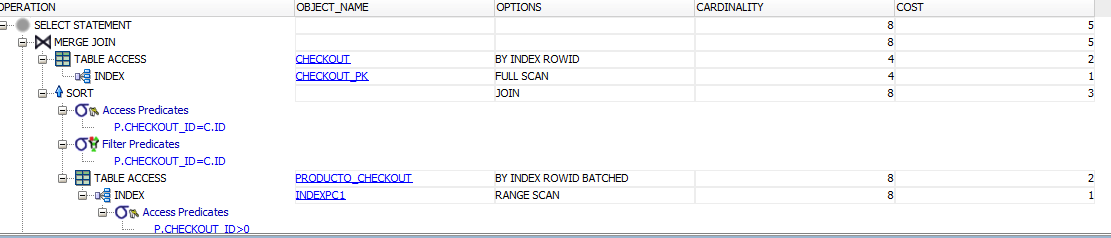
**Apéndice: consultas sin datos cargados**

JOIN 1 consulta 1



No index, 0.015 segundos

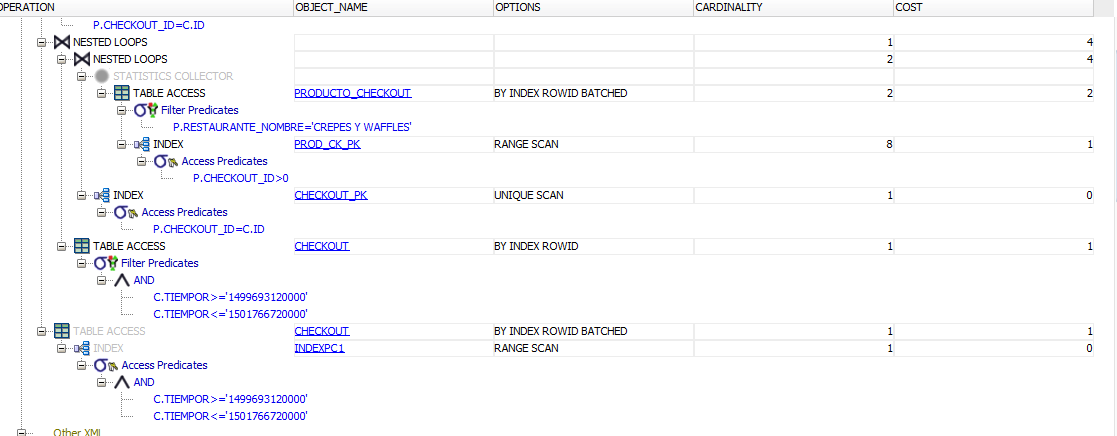




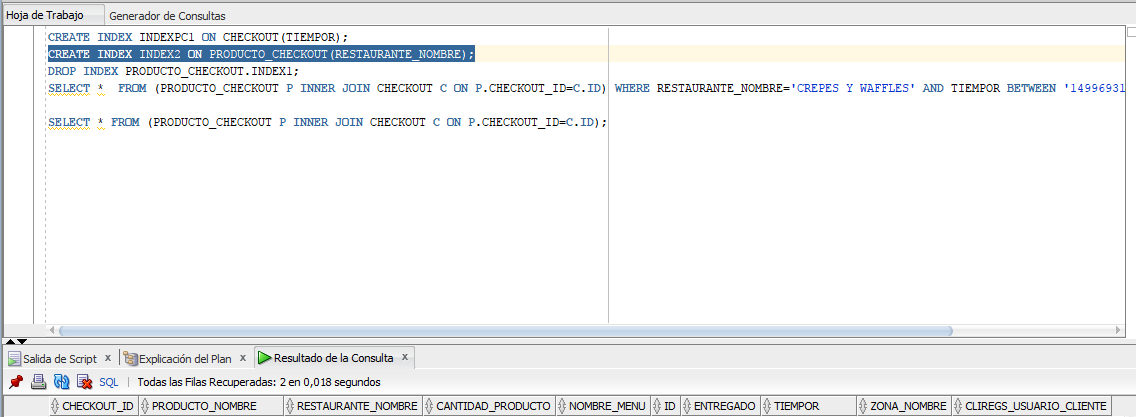
Con index 0.007 segs

Los tiempos son inconsistentes a veces se demora mas a veces menos.

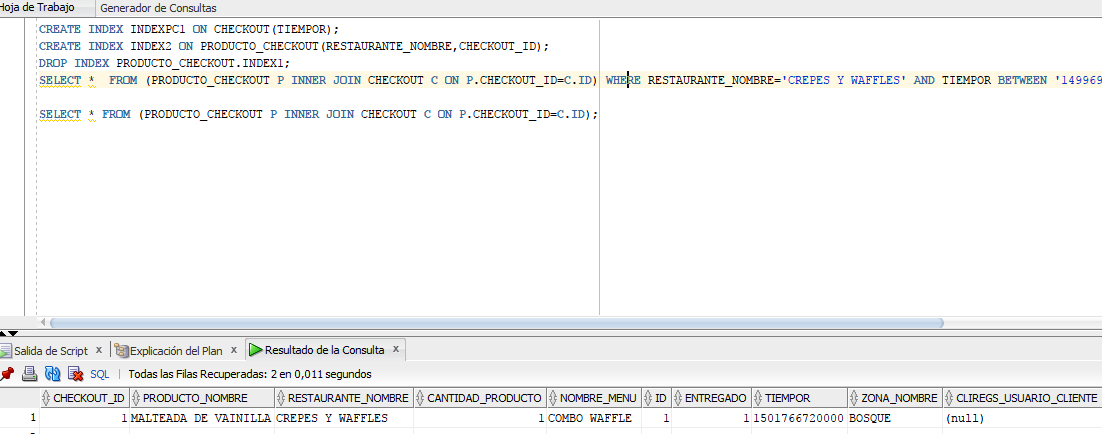
COSTOS INDEX TIEMPO



CREATE INDEX INDEX2 ON PRODUCTO\_CHECKOUT(RESTAURANTE\_NOMBRE); ES MAS SELECTIVO



El tiempo baja

 MEJORA MAS CON INDICE COMPUESTO COSTOS IGUALES.PERO APARENTEMENTE ES IGUAL DE RAPIDO SOLO CON EL ID.

Selección de índices

REQ 1 CONSULTAR CONSUMO EN ROTONDANDES:

REQ 2 CONSULTAR CONSUMO EN ROTONDANDES:

REQ 3 CONSULTAR FUNCIONAMIENTO:

REQ 4 CONSULTAR LOS BUENOS CLIENTES:

Documentación escenario de pruebas

Análisis de eficiencia