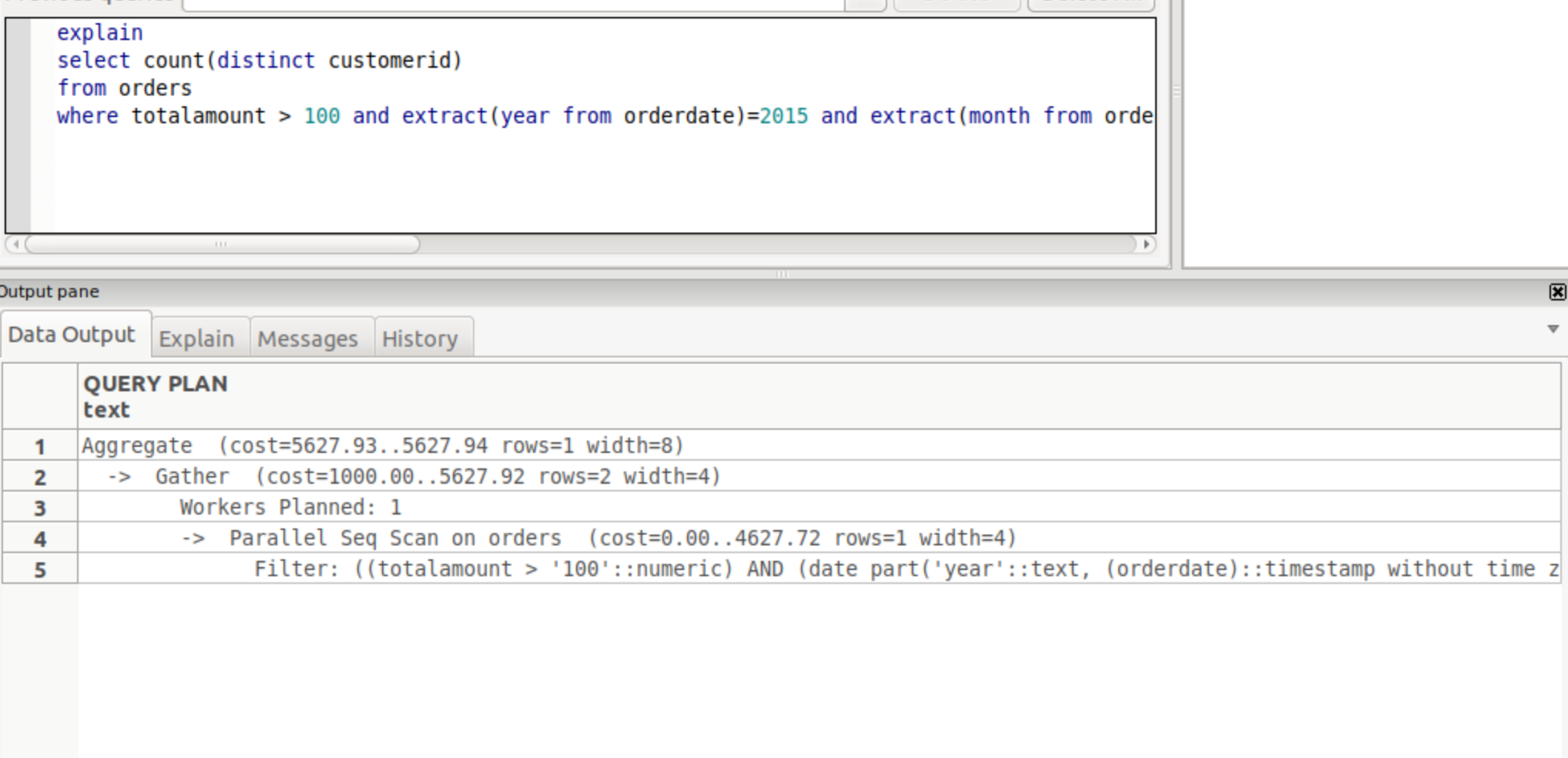
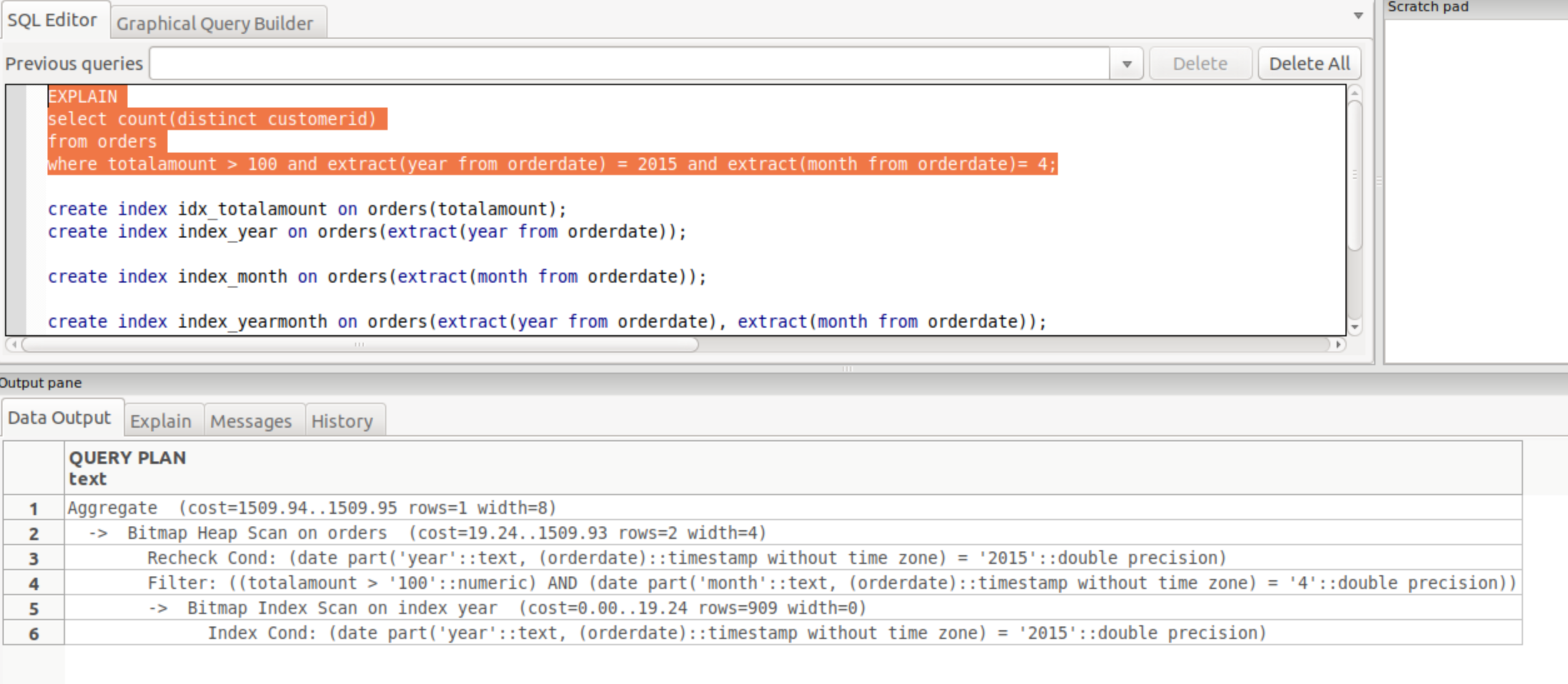
OPTIMIZACION

Se observa que la acción más costosa es el ‘seq scan’ pues este realiza un escaneo secuencial de la tabla de datos almacenada desde la primera fila hasta que la consulta se cumple.

Para mejorar el rendimiento creamos índices sobre esta tabla. Nuestra primera aproximación fue pensar en crear un índice para la selección de pedidos por el coste total, pero dado que más del 50% de los pedidos de la base de datos tenían un coste mayor de 100, el rendimiento no mejoraba mucho pues no descartábamos un gran número de filas, y con la creación de los siguientes índices (mucho más útiles) su mejora de rendimiento era imperceptible. Es por ello que lo descartamos.

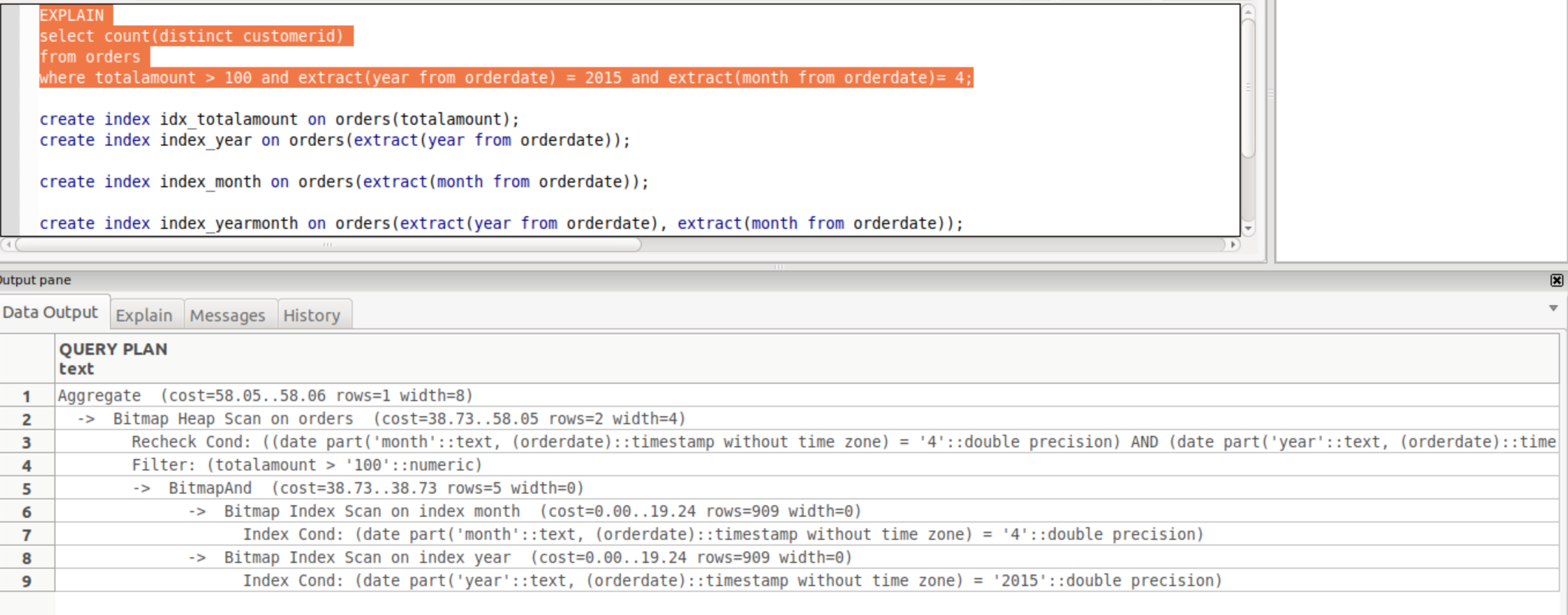
Lo siguiente que pensamos fue crear un índice para la operación de extracción del año en cuestión, esto es:

*create index index\_year on orders(extract(year from orderdate));*

**

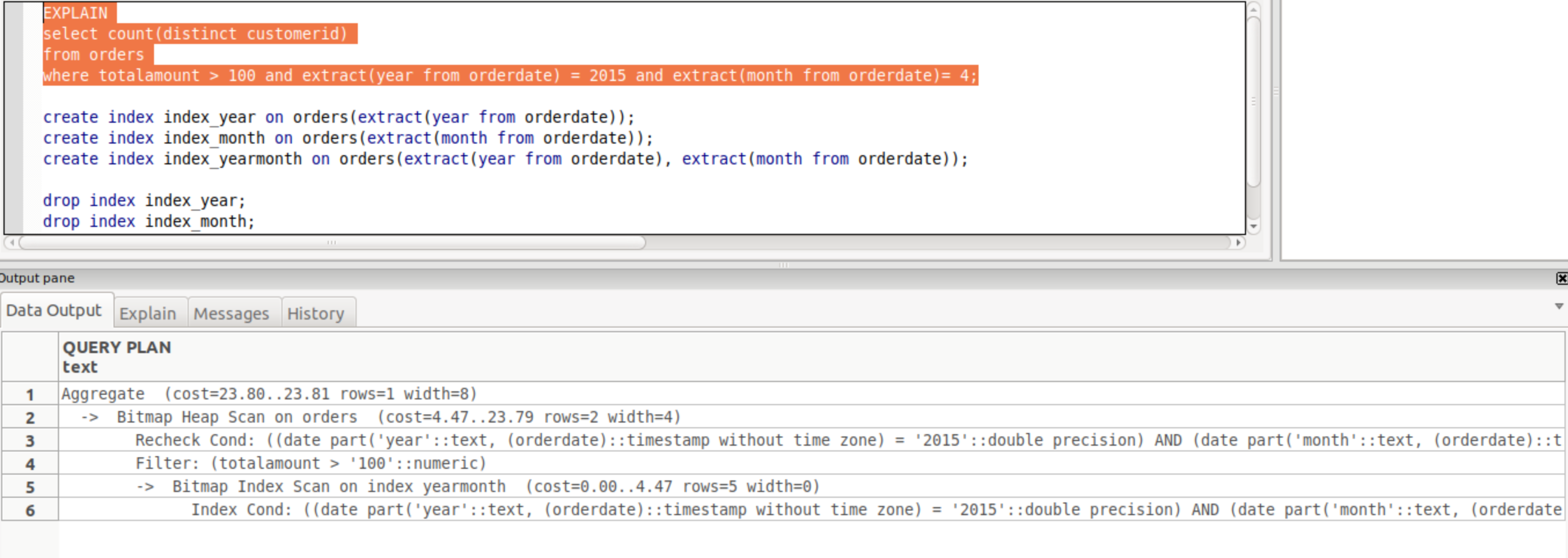
Al ejecutar de nuevo la query vimos una notable mejora en el rendimiento, pues el coste se reducía de 5627 a 1509. El cambio se observa en el cambio de ‘seq scan’ que se realiza en orders por un ‘bitmap heap scan’. Este se divide en dos pasos: la búsqueda de las filas que cumplan la condición en el índice creado y luego ‘traerlas’ del disco para mostrarlo. Traer las filas de una en una es más costoso que leerlas de manera secuencial, el motivo de que esto sea menos costoso es porque gracias al índice no tendremos que leer toda la tabla, pues el subconjunto de filas que tenemos que buscar es notablemente menor que el número total (aproximadamente un 12%).

Viendo esto decidimos probar a añadir otro índice en este caso para la extracción del mes, es decir:

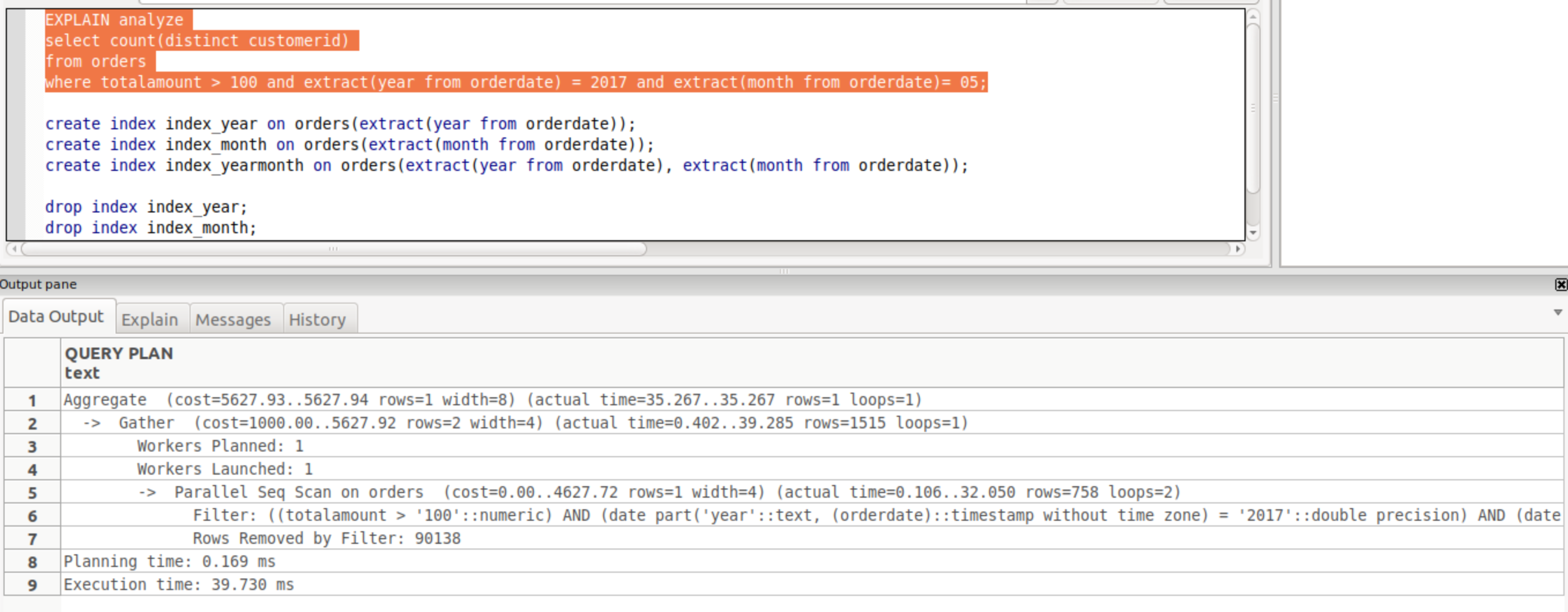
*create index index\_month on orders(extract(year from orderdate));*

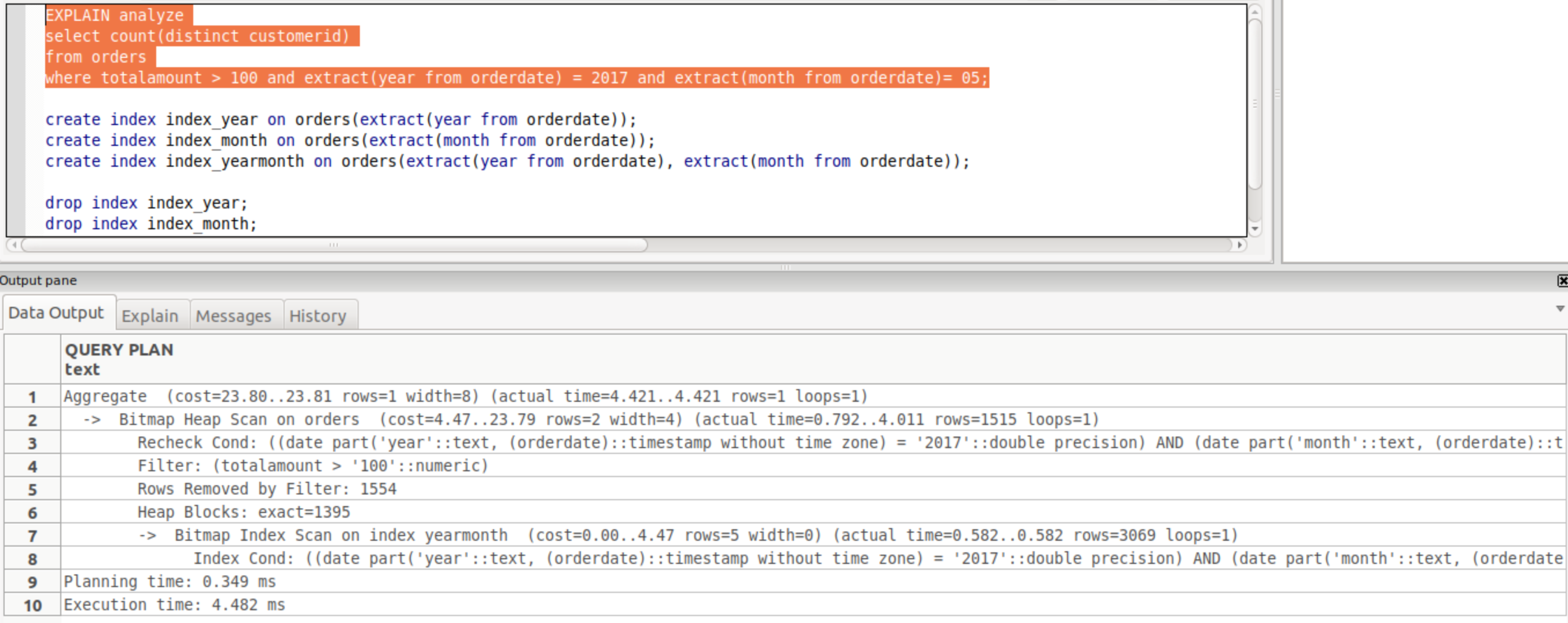
Observamos que el rendimiento de la consulta mejora de nuevo, pasando de un coste de 1509 a un coste total de 58. El razonamiento es el mismo que el anterior.

Viendo esto se nos ocurrió crear un índice más específico uniendo las dos situaciones anteriores en lugar de tenerlos por separados, es decir, un índice para la extracción de la fecha completa. De este modo nos lo imaginamos como un índice de un libro donde cada capítulo es una fecha, y en ellos están los pedidos que se realizaran ese día. Es claro que el número de filas que tendremos que ‘traer’ será menor, por lo que debería mejorar el rendimiento.

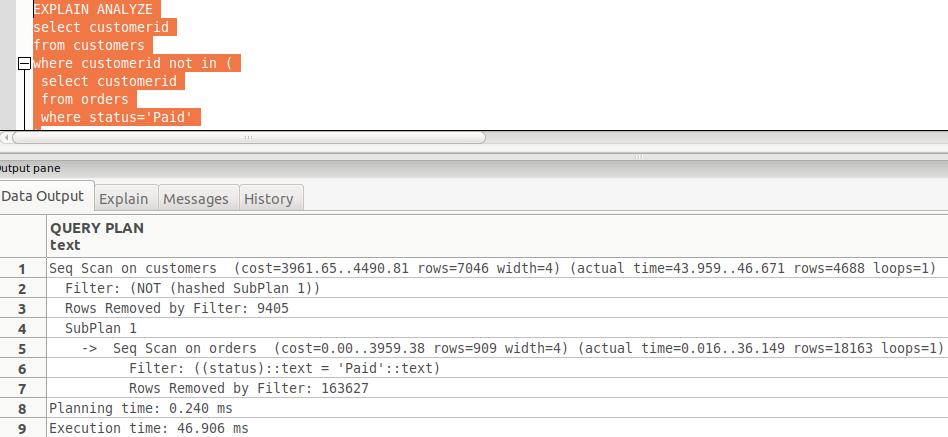
Así obtuvimos nuestro mejor resultado de rendimiento, pasando de 5627 a 23.80:

Haciendo uso de **EXPLAIN ANALYZE** podemos comprobar también una mejora en el tiempo de ejecución (a costa de tardar un poco más en tiempo de planificación pues ‘bitmap heap scan’ consta de dos pasos en su planificación) entre ejecutar la consulta sin índices, y con el último índice discutido (el de fecha completa):

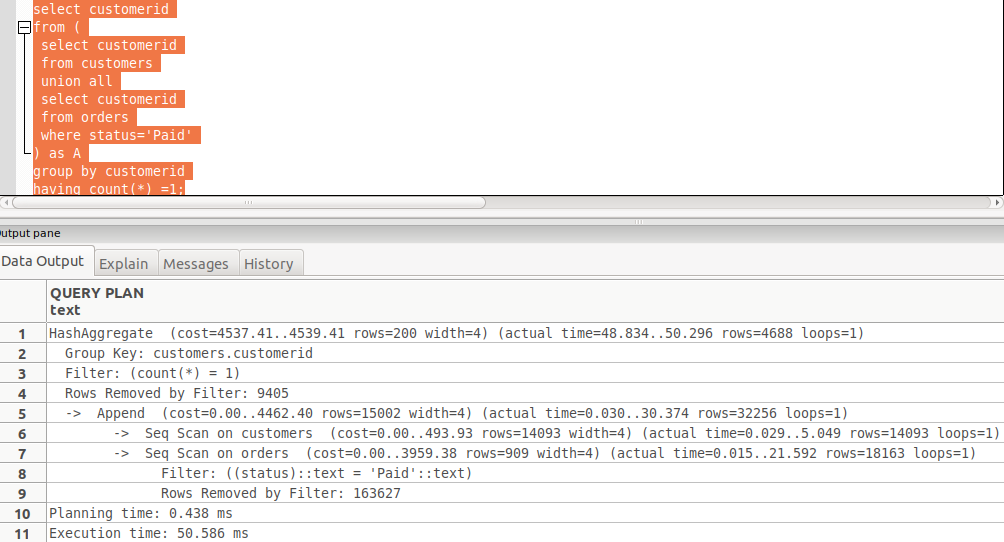
**Sin índices:**

**Con índice:**

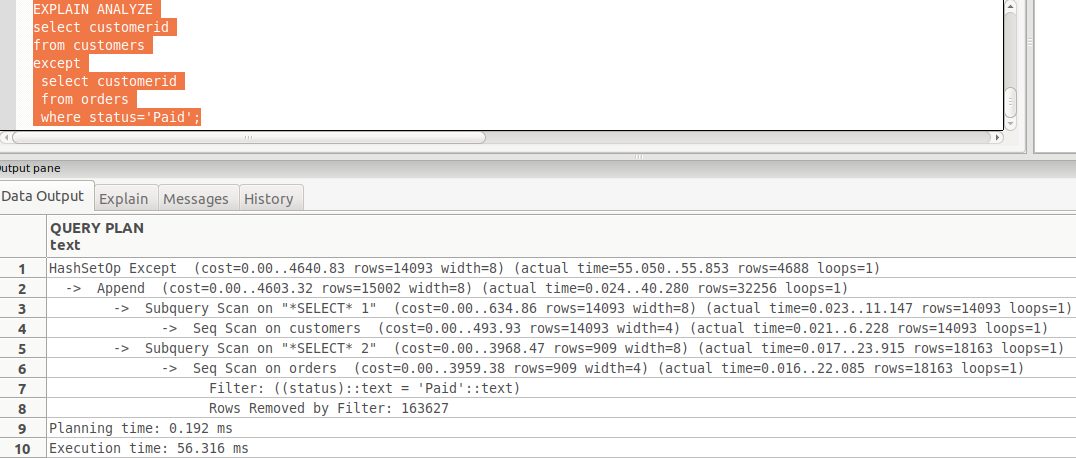
F) Vamos a estudiar la forma de las consultas del Anexo 1.



En la primera consulta obeservamos que se realizan dos *seq scans*, en primer lugar uno de la tabla *orders* y posteriormente uno de la tabla *costumers* con la restricción de *status=“paid“*.



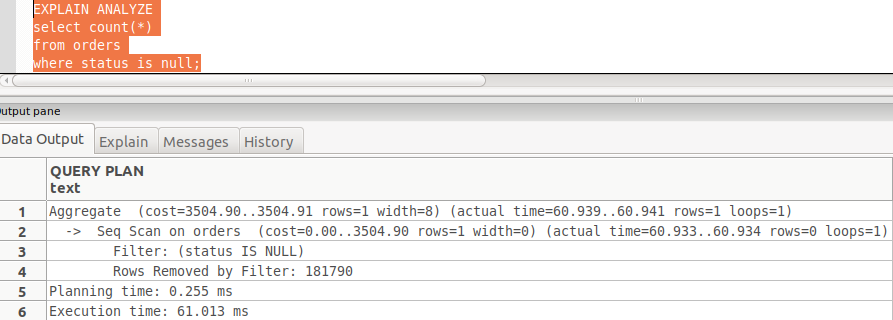
En la segunda consulta se realiza una operación de *HashAggregate* y posteriormente se realizan en paralelo dos *seq* *scan*, donde se recorren secuencialmente las tablas de *orders* y *costumers*. Por este motivo esta query se beneficia de la ejecución en paralelo.



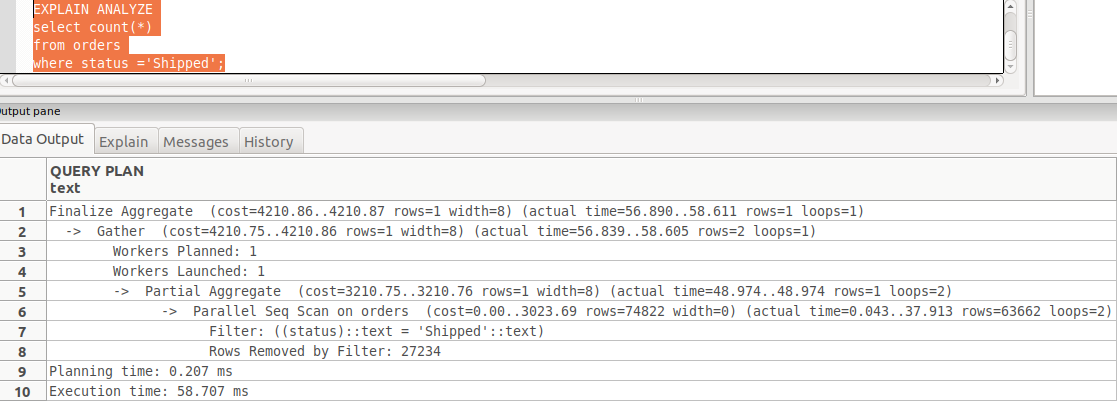
En la tercera consulta se realiza una operacion de *HashSetOp* cuyo coste de ejecución a diferencia de las dos anteriores es (cost=0.00..4640.83 rows=14093 width=8) por lo que mostrará resultados nada más comenzar la ejecución. Además, esta consulta realiza un *append*, que tiene dos *subquerys* que, al igual que la anterior consulta, se benefician de la ejecución en paralelo.

G) Estudiamos el coste de ejecución de las dos consultas del anexo 2 con la sentencia EXPLAIN. Para este primer analisis no usamos ningún índice y obtenemos los siguientes resultados:

* **Consulta 1**

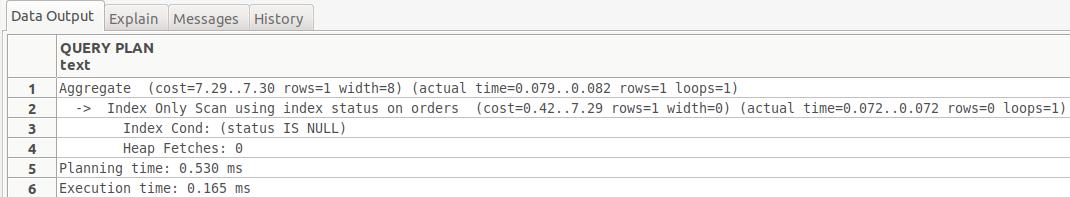
****

* **Consulta 2**

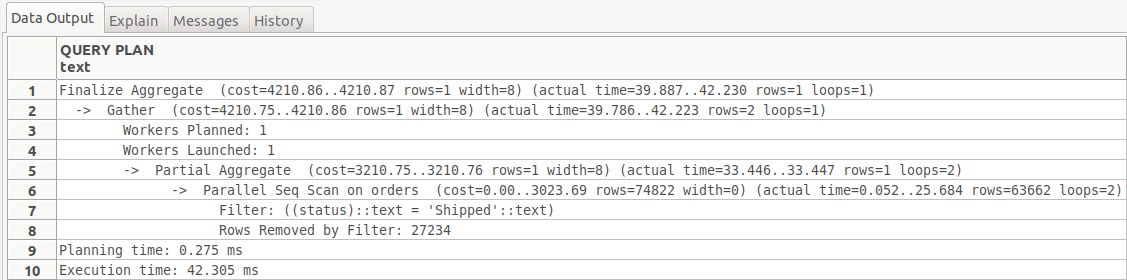
****

A continuación creamos un índice para la tabla orders y la columna status y volvemos a analizar el coste de ejecución de ambas consultas. Observamos que el coste de ejecución mejora considerablemente. Tambien observamos que el tiempo de planificación se incrementa levemente. En cambio, el tiempo de ejecución se reduce enormemente gracias a la introducción del índice. La razón es exactamente la misma que hemos explicado anteriormente en el apartado E.

* **Consulta 1**

****

* **Consulta 2**

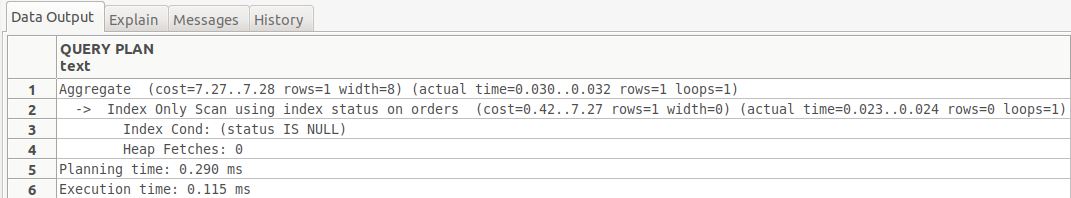
****

A continuación ejecutamos la sentencia ANALYZE sobre la tabla orders y volvemos a analizar los costes, el tiempo y el plan de ejecución de ambas consultas.

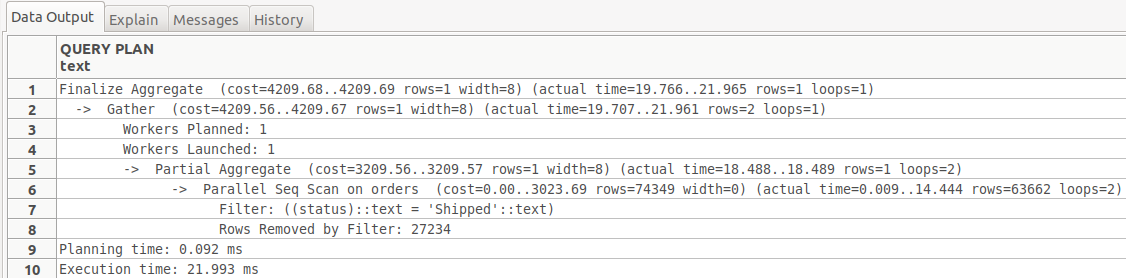
En el caso de la consulta 1, observamos que la planificación de la consulta cambia. Se sustituye el *seq scan* de la tabla orders por una búsqueda por índice. Por este motivo el tiempo de ejecución, el tiempo y el coste mejora considerablemente.

Sin embargo, en la consulta 2 la mejora de tiempo no es tan notable, aunque sigue siendo considerable. Observamos que la planificación no cambia, y la diferencia del coste es insignificante.

* **Consulta 1**

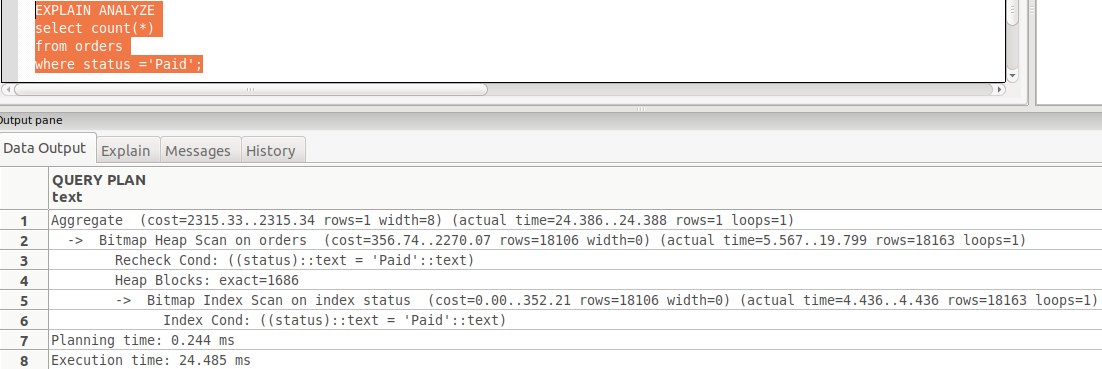
****

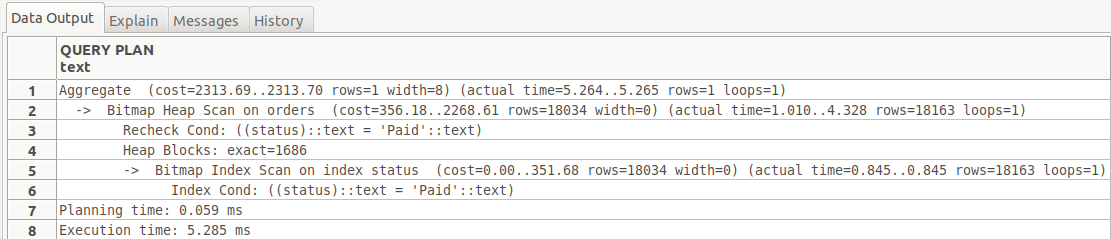
* **Consulta 2**

****

A continuación analizamos las otras dos consultas del anexo 2.

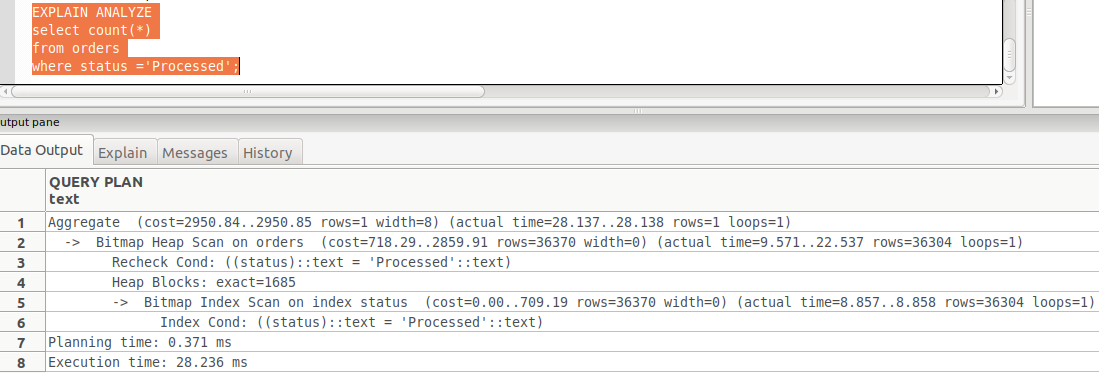
* **Consulta 3**

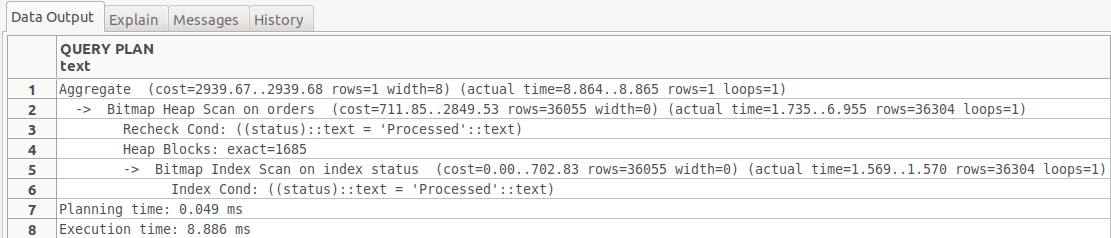
****

****

Ejecutamos la consulta 3 con y sin la ejecución de la sentencia ANALYZE. Observamos que con la sentencia, la planificación permanece idéntica, el coste de ejecución se mantiene y disminuye considerablemente el tiempo de ejecución al igual que en las otras sentencias.

* **Consulta 4**

****

****

Ejecutamos la consulta 4 igual que con la 3. La planificación se mantiene idéntica una vez más, y el coste prácticamente invariable. El tiempo de ejecución en cambio vuelve a bajar considerablemente gracias a la sentencia ANALYZE.