|  |  |
| --- | --- |
|  |  |
|  |

**Gather:**

Estimated plan rows are 5993, but actual rows returned are 5244. With difference of 769, the estimated “plan rows” is not extremely bad.

**Seq scan:**

From node type is “Sequential Scan” we can say filter key “lineitem.l\_suppkey” is not indexded. If the filter key was indexed, then it would run “Indexed Scan” instead.

Although the filter has been used, it will scan through all the rows and omit those which do not pass the filter.

Actual startup time of sequential scan is 2.639 means initializing this step took 2.639 ms.

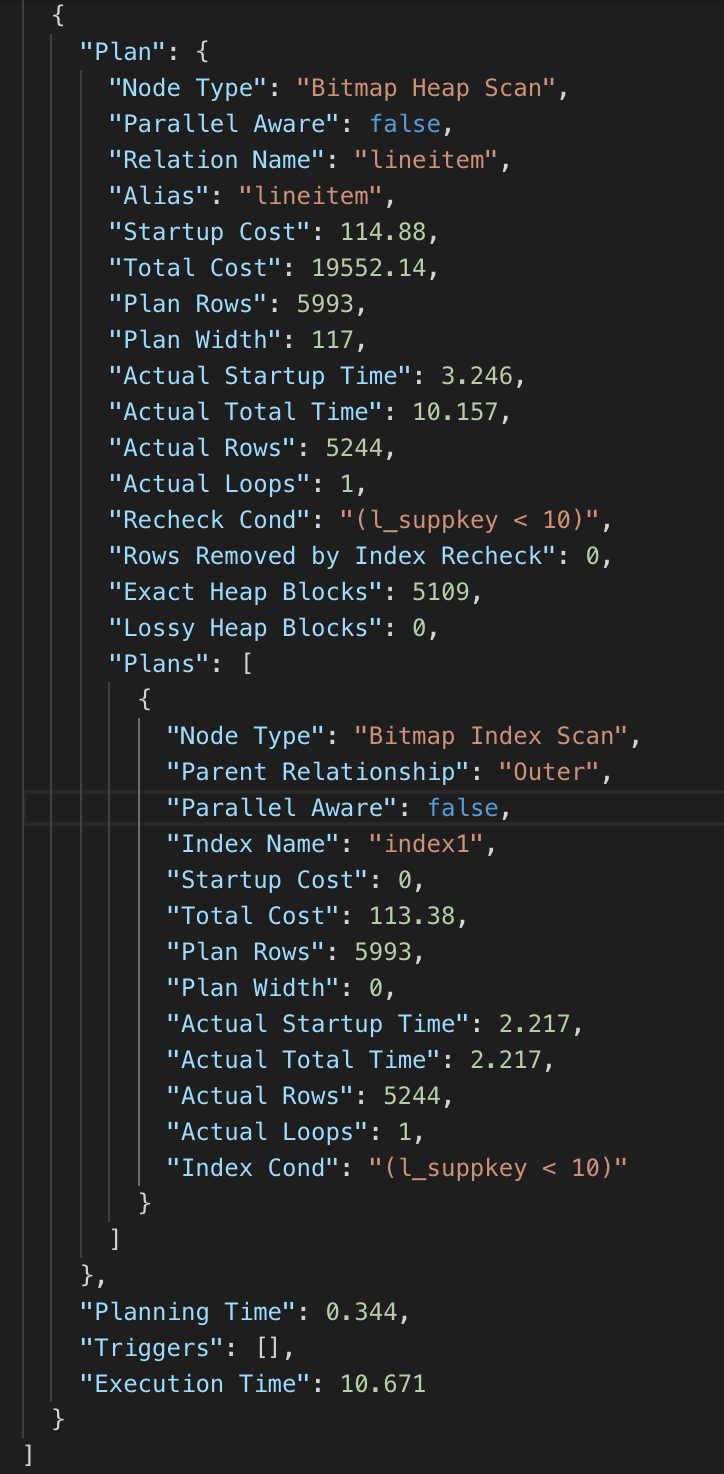
Actual total time spent is total 1341.201 means running this step took 1341.201 ms.

On the other hand, actual total time spent on whole query is 1353.668, which indicates high amount of time spent on sequential scan for the filter.

We can observe high time spent for sequential scan also from the estimated startup and total cost. Almost all the calculation cost is estimated to be done for sequential scan phase.

Estimated plan width 117 means each row returned is size of 117 bytes.

Total execution time of the query is 1354.921 ms.



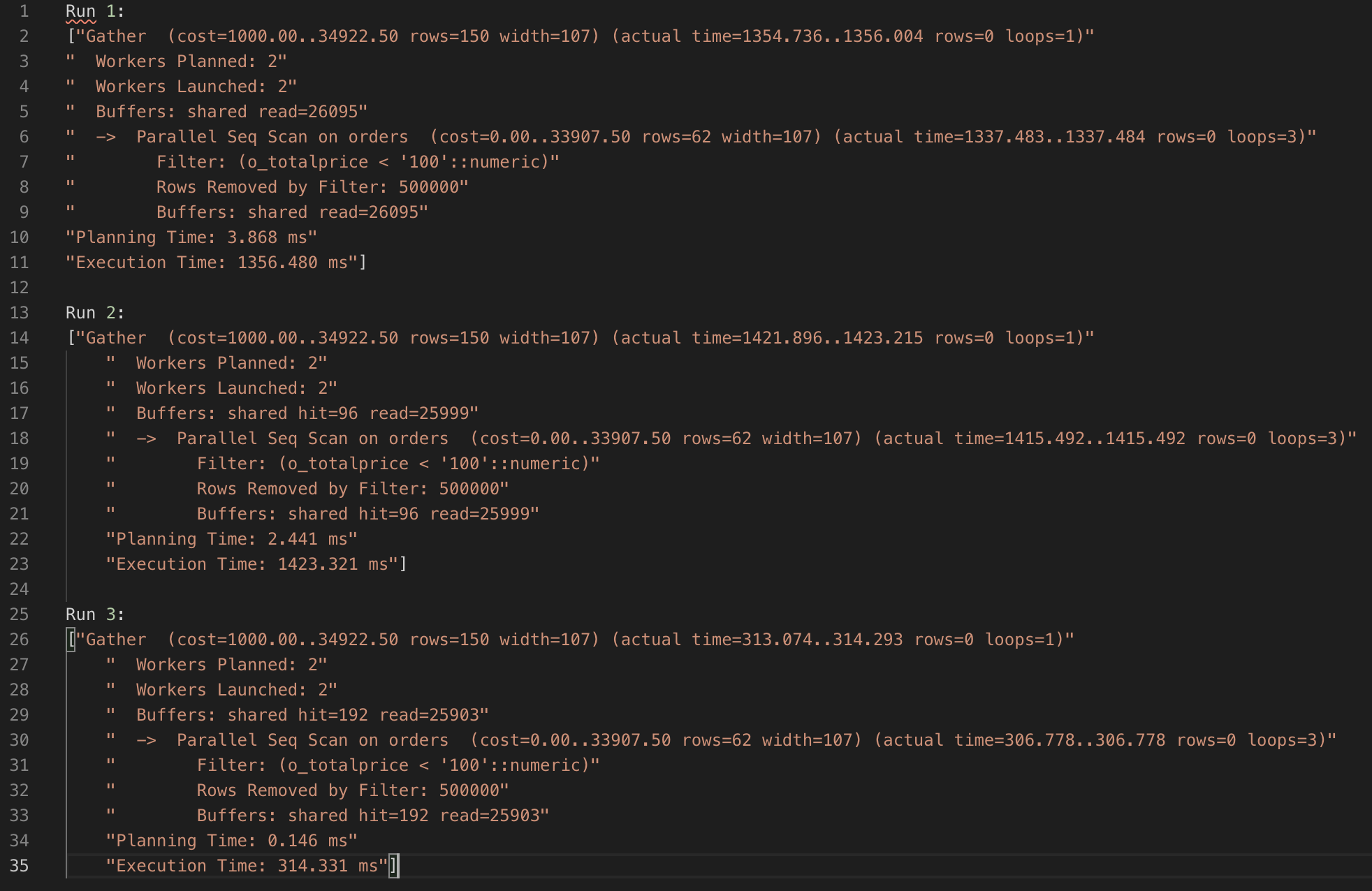
After running the query in #b, we have created an INDEX for lineitem.l\_suppkey field.

Not the query plan has changed from Sequential scan to Bitmap Index Scan, which is a drastic improvement.

From Actual Startup Time and Actual Total Time, we can observe both are same, maybe has ignorable difference. Thus, the Index Scan ran lot faster than Sequential scan.

From Startup Cost and Total Cost, we can say that the time improvement was also estimated by the query plan estimation.

Total execution time of the query is now 130% better than the previous output.

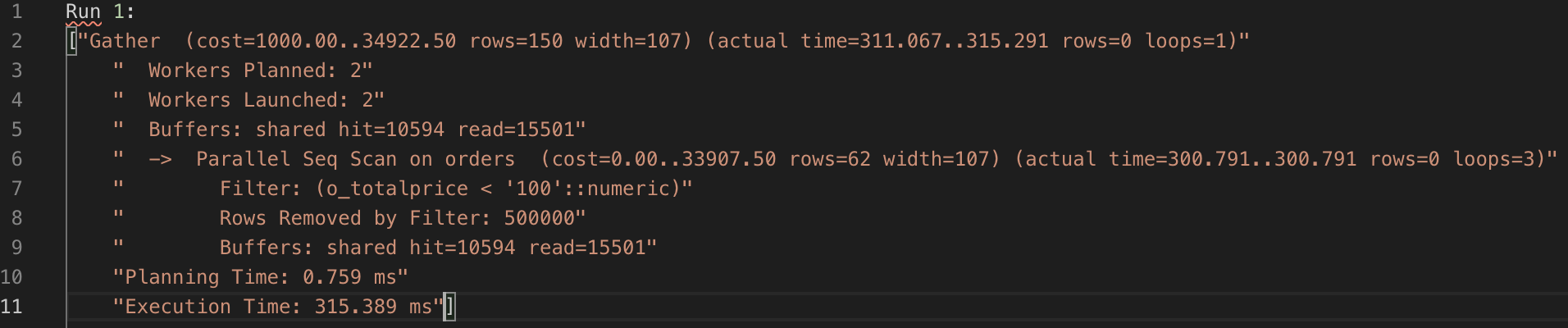


Run 1: Buffers: shared read=26095

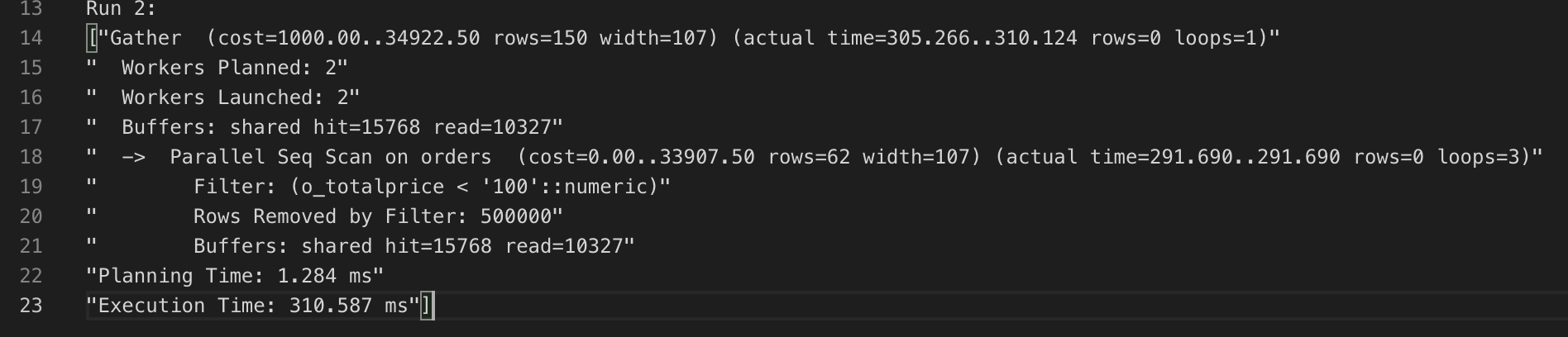
Run 2: Buffers: shared hit=96 read=25999

Run 3: Buffers: shared hit=192 read=25903

* “Buffers: shared read” is the number of blocks postgres read from the disk.
* Total 26,095 memory blocks were needed to read the whole output of the query from disk.
* “Buffers: shared hit” is the number of blocks retrieved from the postgres cache. With each query, postgres takes more and more data from and into the cache, and increase it’s own cache.
* Reading from cache is much faster, we can observe it from the better planning and execution time of the query after each run.
* After observing closely, we can see the size of “Buffers: shared hit” is increasing by the multiple of 96.
* After running 7 times, total 576 blocks were stored in postgres cache.



* With “prewarming” “orders” table, postgres instantiates it’s shared buffer or OS cache or both with relavant pages from “orders” table.
* Thus “Buffers: shared hit” is much higher. Total 10,594 blocks were read from postgres cache.
* This query was run after running previous query 7 times. Thus, actual buffered pages were 10,594-576 = 10,018 after prewarming.



* Same query was run after a fresh restart of the system, here we get a Buffers: shared hit of 15,768.