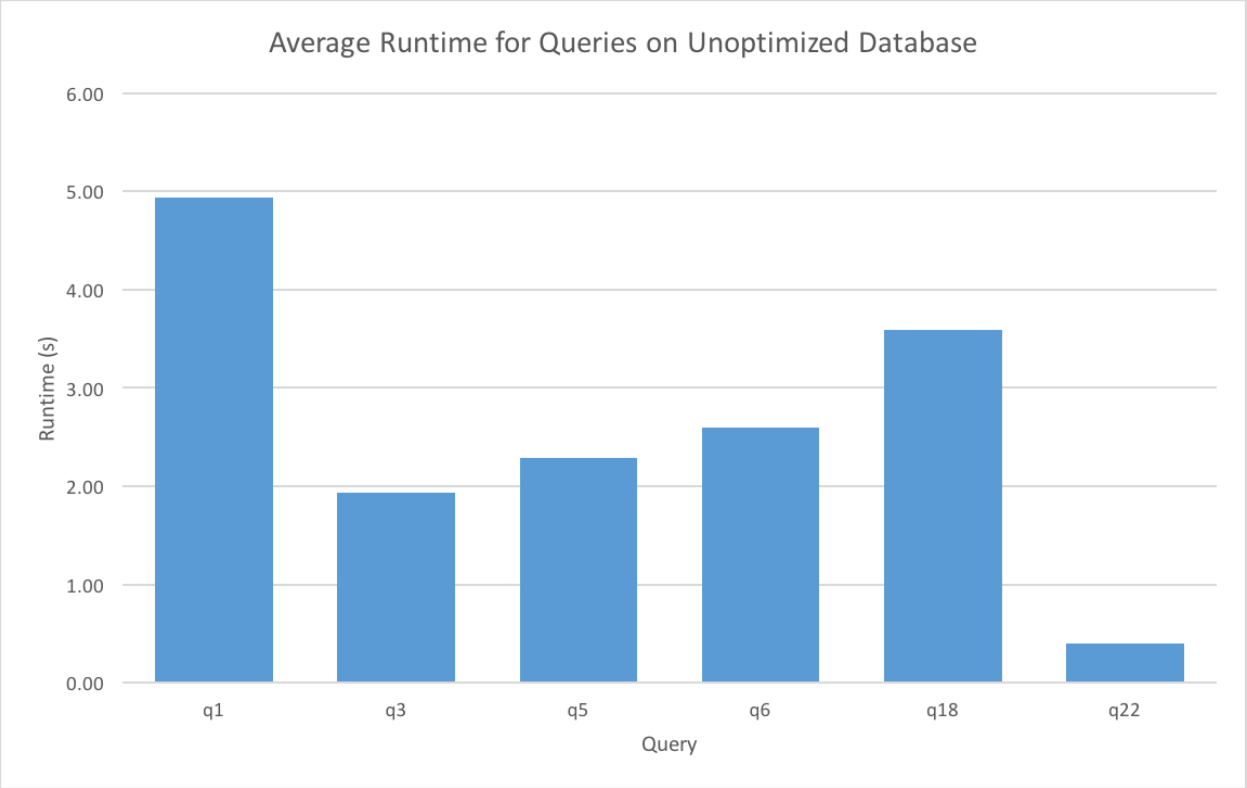
Database Systems Homework 4

April 24, 2016

Joon Hyuck Choi (jchoi100), William Sun (wsun19)

Part 1



Query 1

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |     5 |   135 | 29727       (2)| 00:00:02 |
|   1 |  HASH GROUP BY      |          |     5 |   135 | 29727       (2)| 00:00:02 |
|*  2 |   TABLE ACCESS FULL| LINEITEM |  5789K|   149M| 29514(1)| 00:00:02 |
-------------------------------------------------------------------------------
```

Comments:

- If there were an index, we would not have to scan over the entire table

Query 3

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		11620	680K		42875 (1)	00:00:02
1	HASH GROUP BY		11620	680K		42875 (1)	00:00:02
*2	HASH JOIN		495K	28M	10M	42859 (1)	00:00:02
*3	HASH JOIN		217K	7847K		7479 (1)	00:00:01
*4	TABLE ACCESS FULL	CUSTOMER	30000	468K		935 (1)	00:00:01
*5	TABLE ACCESS FULL	ORDERS	729K	14M		6542 (1)	00:00:01
*6	TABLE ACCESS FULL	LINEITEM	3225K	70M		29514 (1)	00:00:02

Comments:

- It may be that the two hash joins could have their ordering switched around. To do this we would need to compute statistics or sample the tables.
- If there were an index, we would not have to scan over the entire table

Query 5

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		25	2300		42747 (1)	00:00:02
1	HASH GROUP BY		25	2300		42747 (1)	00:00:02
* 2	HASH JOIN		7410	665K	2936K	42746 (1)	00:00:02
3	TABLE ACCESS FULL	CUSTOMER	150K	1171K		933 (1)	00:00:01
* 4	HASH JOIN		185K	14M	6912K	40828 (1)	00:00:02
* 5	TABLE ACCESS FULL	ORDERS	228K	4235K		6542 (1)	00:00:01
* 6	HASH JOIN		1200K	74M		29572 (1)	00:00:02
7	VIEW	VW_GBF_33	2000	92000		64 (2)	00:00:01
8	HASH GROUP BY		2000	132K		64 (2)	00:00:01
* 9	HASH JOIN		2000	132K		63 (0)	00:00:01
10	TABLE ACCESS FULL	NATION	25	800		2 (0)	00:00:01
11	MERGE JOIN CARTESIAN		10000	351K		61 (0)	00:00:01
* 12	TABLE ACCESS FULL	REGION	1	29		2 (0)	00:00:01
13	BUFFER SORT		10000	70000	2	59 (0)	00:00:01
14	TABLE ACCESS FULL	SUPPLIER	10000	70000		59 (0)	00:00:01
15	TABLE ACCESS FULL	LINEITEM	6001K	108M		29486 (1)	00:00:02

Comments:

- Like in query 3, the joins could be optimized if the tables were sampled or statistics were gathered. In addition, a lot of the operations (sort, join) may benefit from parallelism
- If there were an index, we would not have to scan over the entire table

Query 6

| Id | Operation | Name | Rows | Bytes
TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		1	48		2207 (1)	00:00:01
1	SORT AGGREGATE		1	48			
* 2	TABLE ACCESS BY INDEX ROWID BATCHED| LINEITEM	114K| 5355K|
2207 (1)	00:00:01
3	BITMAP CONVERSION TO ROWIDS						
4	BITMAP AND						
5	BITMAP CONVERSION FROM ROWIDS						
6	SORT ORDER BY						
* 7	INDEX RANGE SCAN	LINEITEM_DISCOUNT	39898			55 (0)
00:00:01
8	BITMAP CONVERSION FROM ROWIDS						
9	SORT ORDER BY				6968K		
* 10	INDEX RANGE SCAN	LINEITEM_QUANTITY	39898			108 (0)
00:00:01 |

Comments:

- Indexing would make the index range scan more efficient.

Query 18

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		4	256	66675 (1)	00:00:03
1	HASH GROUP BY		4	256	66675 (1)	00:00:03
* 2	HASH JOIN		4	256	66674 (1)	00:00:03
* 3	HASH JOIN		1	55	37175 (1)	00:00:02
* 4	HASH JOIN RIGHT SEMI		1	31	36241 (1)	00:00:02
5	VIEW	VW_NSO_1	1	6	29698 (2)	00:00:02
* 6	FILTER					
7	HASH GROUP BY		1	9	29698 (2)	00:00:02
8	TABLE ACCESS FULL	LINEITEM	6001K	51M	29478 (1)	00:00:02
9	TABLE ACCESS FULL	ORDERS	1500K	35M	6538 (1)	00:00:01
10	TABLE ACCESS FULL	CUSTOMER	150K	3515K	933 (1)	00:00:01
11	TABLE ACCESS FULL	LINEITEM	6001K	51M	29478 (1)	00:00:02

Comments:

- This query could be improved by statistics to improve the join ordering.
- If there were an index, we would not have to scan over the entire table

Query 22

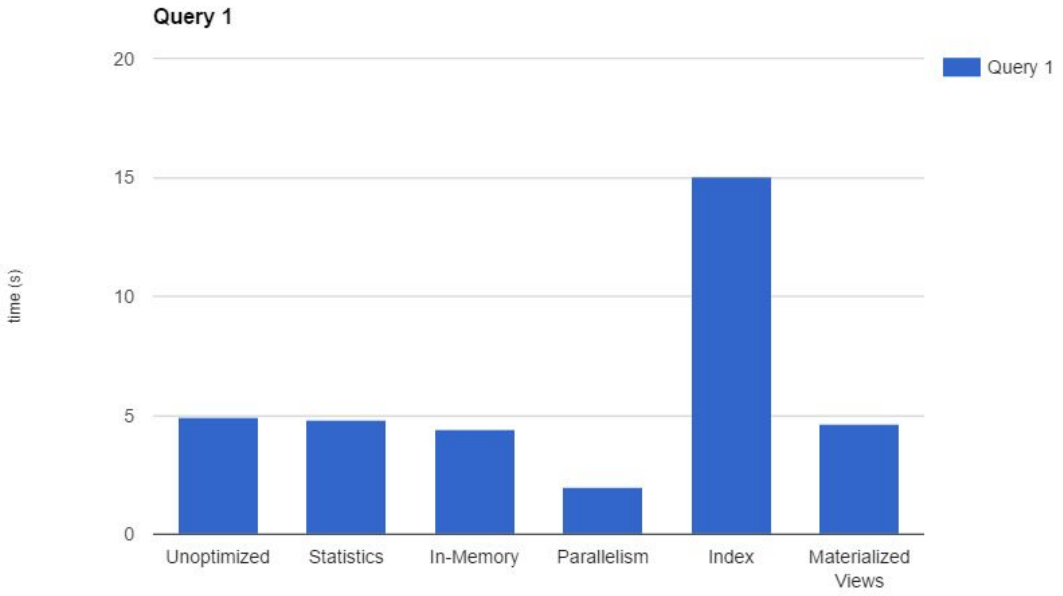
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		5	160	8413 (1)	00:00:01
1	HASH GROUP BY		5	160	8413 (1)	00:00:01
* 2	HASH JOIN ANTI		5	160	7474 (1)	00:00:01
* 3	TABLE ACCESS FULL	CUSTOMER	510	13770	934 (1)	00:00:01
4	SORT AGGREGATE		1	22		
* 5	TABLE ACCESS FULL	CUSTOMER	44719	960K	939 (1)	00:00:01
6	TABLE ACCESS FULL	ORDERS	1500K	7324K	6534 (1)	00:00:01

Comments:

- This query could improve join ordering with statistics
- If there were an index, we would not have to scan over the entire table

Part 2

Query 1



Comments:

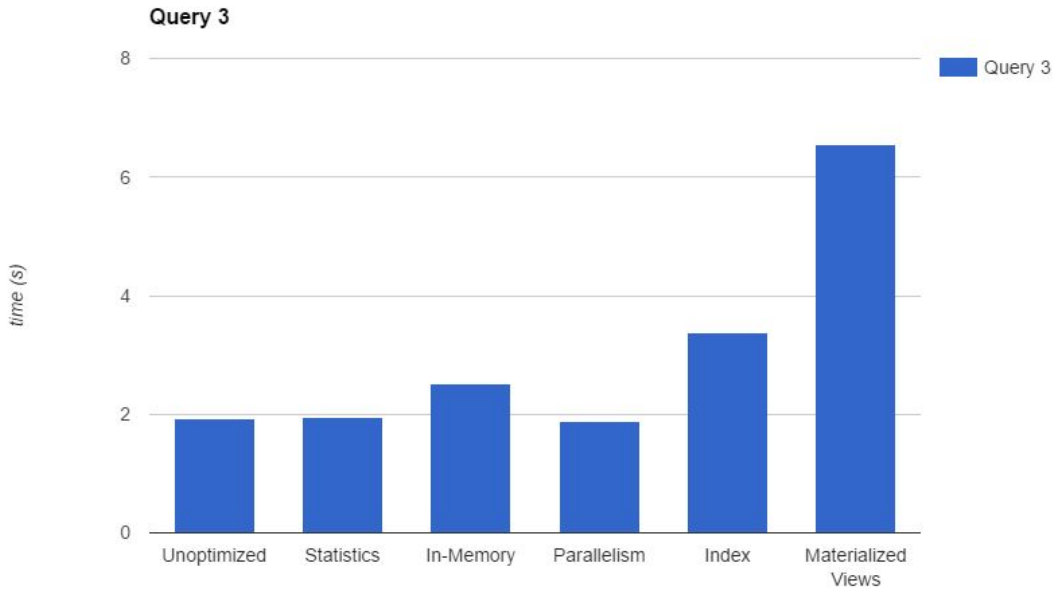
- Parallelism took shortest time:
    - Query was short and simple
    - Otherwise, mostly same performance as regular plan
    - Using indexes on attribute l_shipdate did not perform well.

Note: The first run using materialized views took disproportionately long. So we excluded it when computing the average run time for that method.

```
-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |          |  6657K|   425M|  1598  (32)| 00:00:01 |
|   1 |  HASH GROUP BY              |          |  6657K|   425M|  1598  (32)| 00:00:01 |
|*  2 |   TABLE ACCESS INMEMORY FULL| LINEITEM |  6657K|   425M|  1352  (20)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Figure. Plan using inmemory table

Query 3



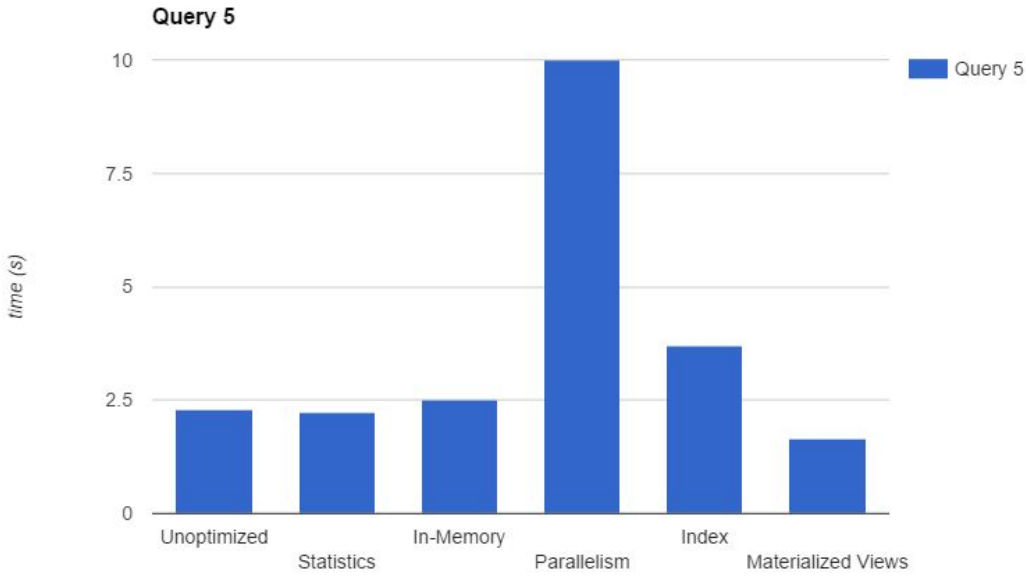
Comments:

- The optimizations that turned out better showed similar performance to regular
- Others performed worse than the unoptimized version
    - Query involved joining three tables
    - For Materialized Views, we used a view that already joined the tables

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |  6657K|   425M|  1598  (32)| 00:00:01 |
|   1 |  HASH GROUP BY               |          |  6657K|   425M|  1598  (32)| 00:00:01 |
|*  2 |   TABLE ACCESS INMEMORY FULL| LINEITEM |  6657K|   425M|  1352  (20)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Figure. Plan using materialized views

Query 5



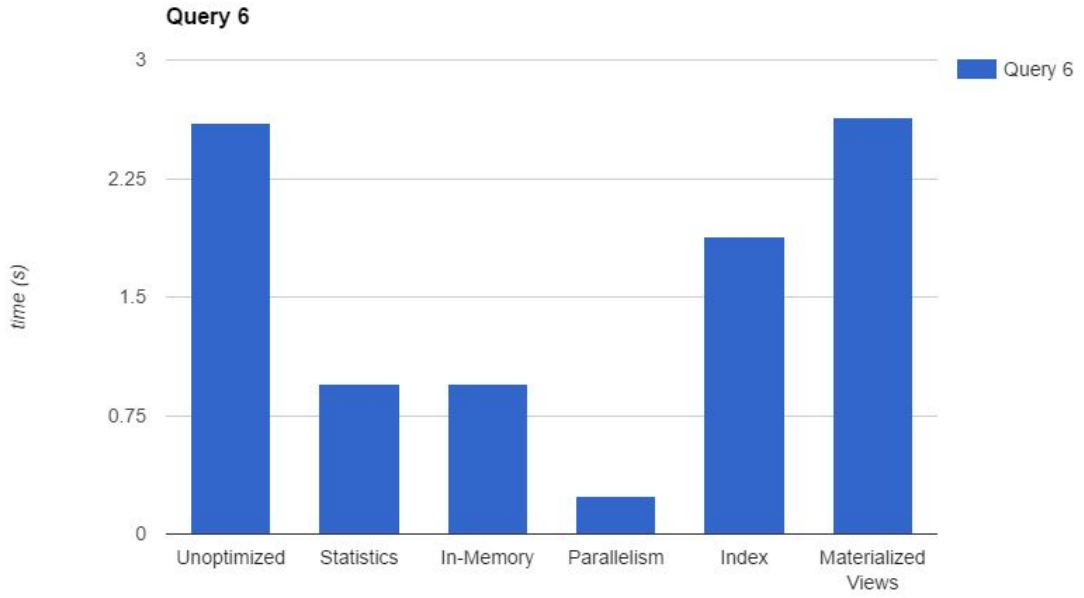
Comments:

- For parallelism, it took too long to run the query, so we had to cut the query off.
    - So its runtime is not actually 10.
- Others performed better than the unoptimized version
- Unlike for most of the other queries, materialized views performed well here.
    - Again, the first couple of runs took too long, so we excluded them in avg.

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		8296	850K		7946 (4)	00:00:01
1	HASH GROUP BY		8296	850K		7946 (4)	00:00:01
* 2	HASH JOIN		8296	850K	10M	7945 (4)	00:00:01
3	VIEW	VW_GBF_39	215K	8189K		1114 (6)	00:00:01
4	HASH GROUP BY		215K	12M		1114 (6)	00:00:01
* 5	HASH JOIN		215K	12M	6096K	1107 (6)	00:00:01
6	TABLE ACCESS INMEMORY FULL	CUSTOMER	164K	4167K		39 (11)	00:00:01
* 7	TABLE ACCESS INMEMORY FULL	ORDERS	215K	7349K		292 (18)	00:00:01
8	VIEW	VW_GBC_38	1351K	85M		1319 (17)	00:00:01
9	HASH GROUP BY		1351K	220M		1319 (17)	00:00:01
* 10	HASH JOIN		1351K	220M		1273 (14)	00:00:01
11	JOIN FILTER CREATE	:BF0000	2151	249K		4 (0)	00:00:01
* 12	HASH JOIN		2151	249K		4 (0)	00:00:01
13	TABLE ACCESS INMEMORY FULL	NATION	25	1325		1 (0)	00:00:01
14	MERGE JOIN CARTESIAN		10753	693K		3 (0)	00:00:01
* 15	TABLE ACCESS INMEMORY FULL	REGION	1	40		1 (0)	00:00:01
16	BUFFER SORT		10753	273K		3 (0)	00:00:01
17	TABLE ACCESS INMEMORY FULL	SUPPLIER	10753	273K		3 (0)	00:00:01
18	JOIN FILTER USE	:BF0000	6755K	335M		1245 (13)	00:00:01
* 19	TABLE ACCESS INMEMORY FULL	LINEITEM	6755K	335M		1245 (13)	00:00:01

Figure. Plan using inmemory table

Query 6



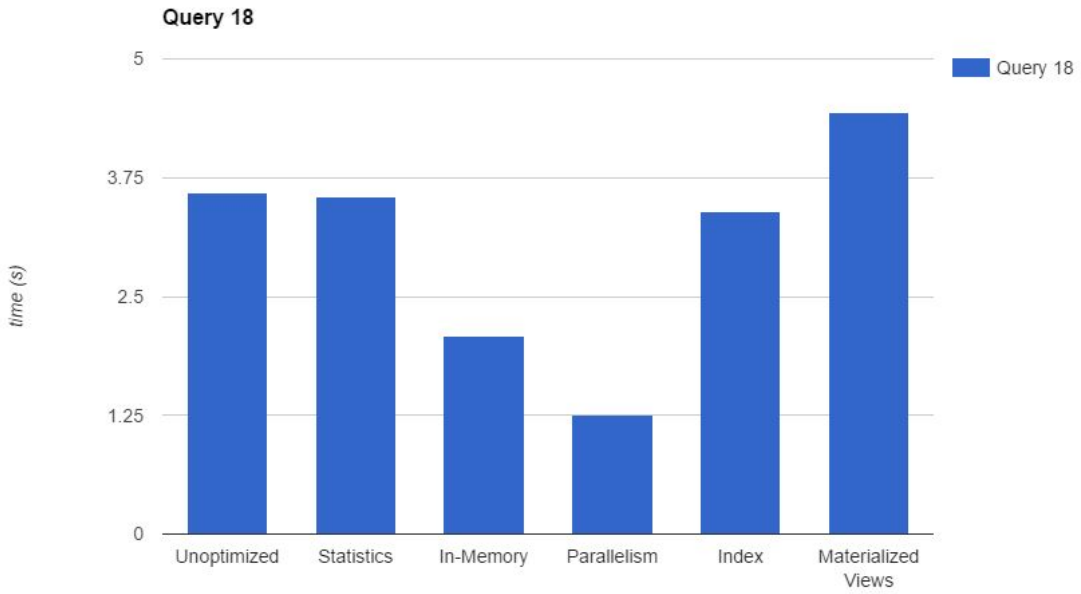
Comments:

- For all queries (except for query 5), parallelism took about half the time of unoptimized
- Using materialized views did not give much improvement.

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		1	13		10408 (1)	00:00:01
1	SORT AGGREGATE		1	13			
2	VIEW		2	26		10408 (1)	00:00:01
3	UNION-ALL						
4	SORT AGGREGATE		1	48			
* 5	MAT_VIEW REWRITE ACCESS FULL	LINEITEM_SHIPDATE2	479	22992		8202 (1)	00:00:01
6	SORT AGGREGATE		1	48			
* 7	TABLE ACCESS BY INDEX ROWID BATCHED	LINEITEM	113K	5342K		2207 (1)	00:00:01
8	BITMAP CONVERSION TO ROWIDS						
9	BITMAP AND						
10	BITMAP CONVERSION FROM ROWIDS						
11	SORT ORDER BY						
* 12	INDEX RANGE SCAN	LINEITEM_DISCOUNT	39898			55 (0)	00:00:01
13	BITMAP CONVERSION FROM ROWIDS						
14	SORT ORDER BY				6968K		
* 15	INDEX RANGE SCAN	LINEITEM_QUANTITY	39898			108 (0)	00:00:01

Figure. Plan using materialized views

Query 18



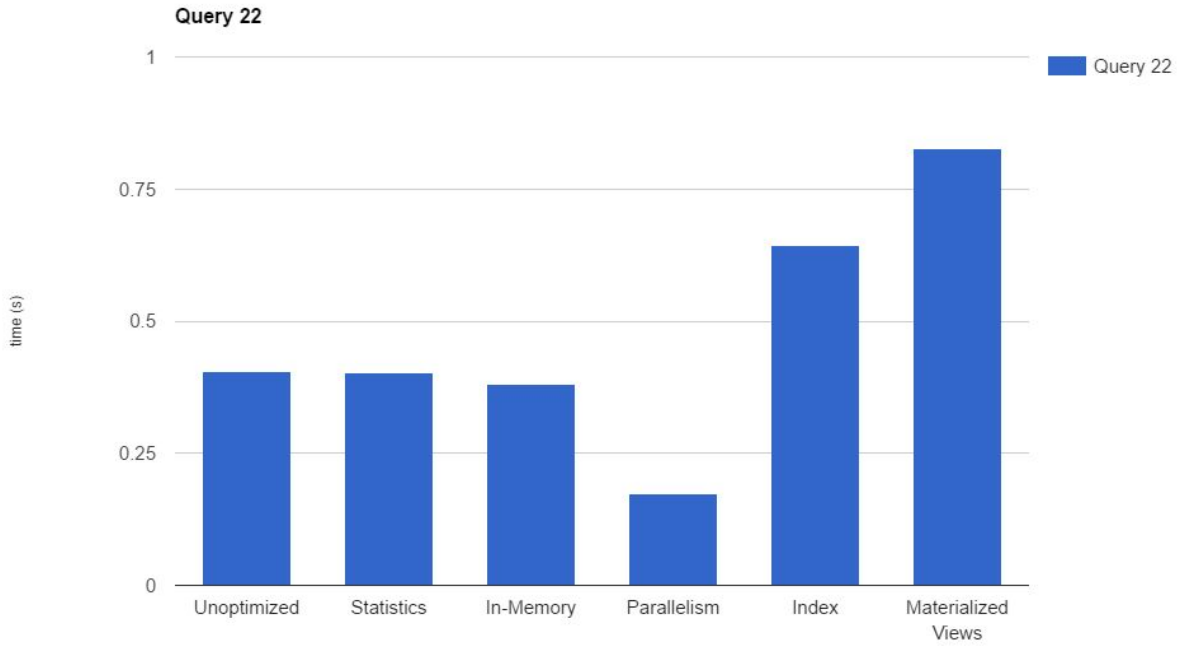
Comments:

- In-memory and parallelism did better than unoptimized (t = 3.588)
- Others performed worse than the unoptimized version

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		5	570	78382 (1)	00:00:04
1	HASH GROUP BY		5	570	78382 (1)	00:00:04
* 2	HASH JOIN		5	570	78381 (1)	00:00:04
* 3	HASH JOIN		1	88	55543 (2)	00:00:03
* 4	HASH JOIN		1	61	55505 (2)	00:00:03
5	VIEW	VW_NSO_1	6755K	83M	23064 (2)	00:00:01
* 6	FILTER					
7	HASH GROUP BY		1	167M	23064 (2)	00:00:01
8	TABLE ACCESS INMEMORY FULL	LINEITEM	6755K	167M	22814 (1)	00:00:01
9	TABLE ACCESS INMEMORY FULL	ORDERS	1400K	64M	287 (16)	00:00:01
10	TABLE ACCESS INMEMORY FULL	CUSTOMER	164K	4328K	38 (8)	00:00:01
11	TABLE ACCESS INMEMORY FULL	LINEITEM	6755K	167M	22814 (1)	00:00:01

Figure. Plan using in-memory table

Query 22



Comments:

- Parallelism performed almost twice as fast as the unoptimized version
- Others performed similarly or worse than the unoptimized version

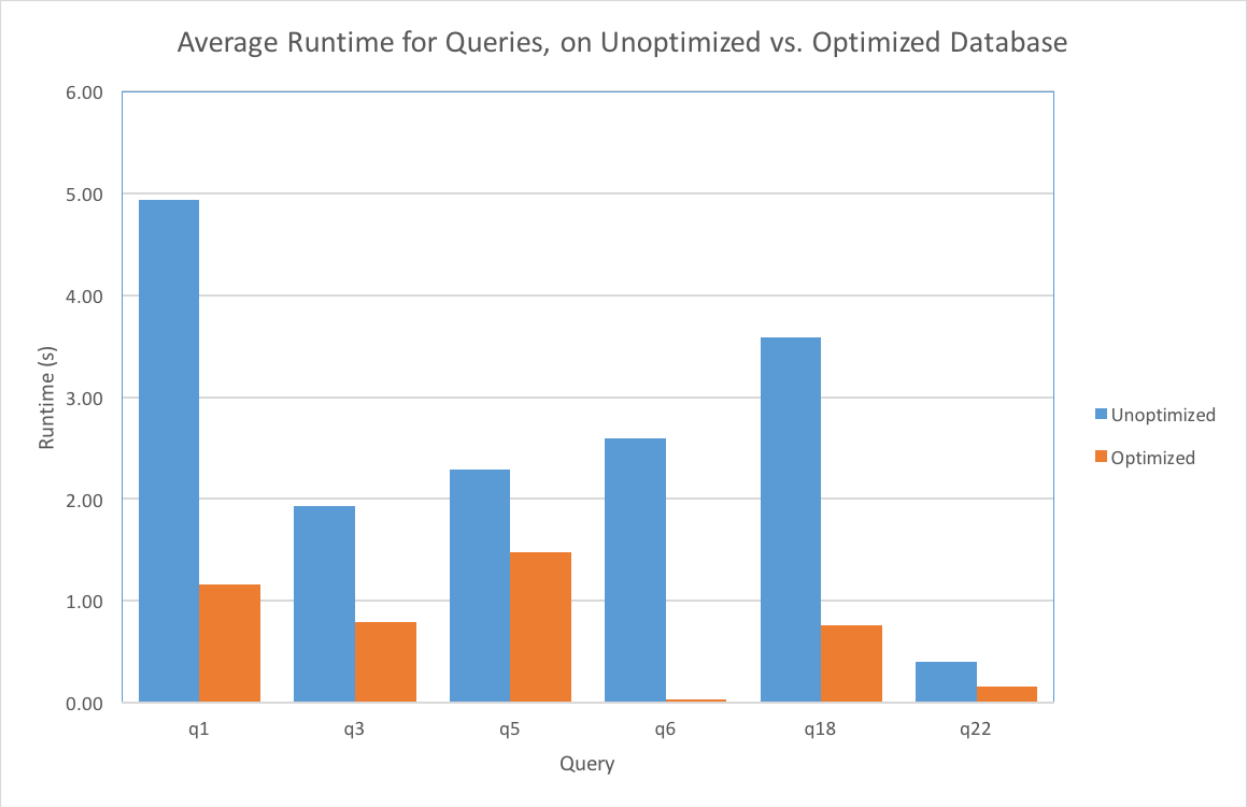
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT		10	560	2336 (1)	00:00:01			
1	PX COORDINATOR								
2	PX SEND QC (RANDOM)	:TQ20003	10	560	2336 (1)	00:00:01	Q2,03	P->S	QC (RAND)
3	HASH GROUP BY		10	560	2336 (1)	00:00:01	Q2,03	PCWP	
4	PX RECEIVE		10	560	2336 (1)	00:00:01	Q2,03	PCWP	
5	PX SEND HASH	:TQ20002	10	560	2336 (1)	00:00:01	Q2,02	P->P	HASH
6	HASH GROUP BY		10	560	2336 (1)	00:00:01	Q2,02	PCWP	
* 7	HASH JOIN ANTI		10	560	2076 (1)	00:00:01	Q2,02	PCWP	
8	PX RECEIVE		952	40936	260 (1)	00:00:01	Q2,02	PCWP	
9	PX SEND HASH	:TQ20000	952	40936	260 (1)	00:00:01	Q2,00	P->P	HASH
10	PX BLOCK ITERATOR		952	40936	260 (1)	00:00:01	Q2,00	PCWC	
* 11	TABLE ACCESS FULL	CUSTOMER	952	40936	260 (1)	00:00:01	Q2,00	PCWP	
12	SORT AGGREGATE		1	30			Q2,00	PCWP	
13	PX COORDINATOR								
14	PX SEND QC (RANDOM)	:TQ10000	1	30			Q1,00	P->S	QC (RAND)
15	SORT AGGREGATE		1	30			Q1,00	PCWP	
16	PX BLOCK ITERATOR		44719	1310K	260 (1)	00:00:01	Q1,00	PCWC	
* 17	TABLE ACCESS FULL	CUSTOMER	44719	1310K	260 (1)	00:00:01	Q1,00	PCWP	
18	PX RECEIVE		1400K	17M	1815 (1)	00:00:01	Q2,02	PCWP	
19	PX SEND HASH	:TQ20001	1400K	17M	1815 (1)	00:00:01	Q2,01	P->P	HASH
20	PX BLOCK ITERATOR		1400K	17M	1815 (1)	00:00:01	Q2,01	PCWC	
21	TABLE ACCESS FULL	ORDERS	1400K	17M	1815 (1)	00:00:01	Q2,01	PCWP	

Figure. Plan using parallelism

Note: For each of the queries, we just showed one example plan output. Just for reference, all the plan outputs for each of the runs for each of the queries is in the /plans directory.

Note: The raw run-time statistics are included in the /stats directory.

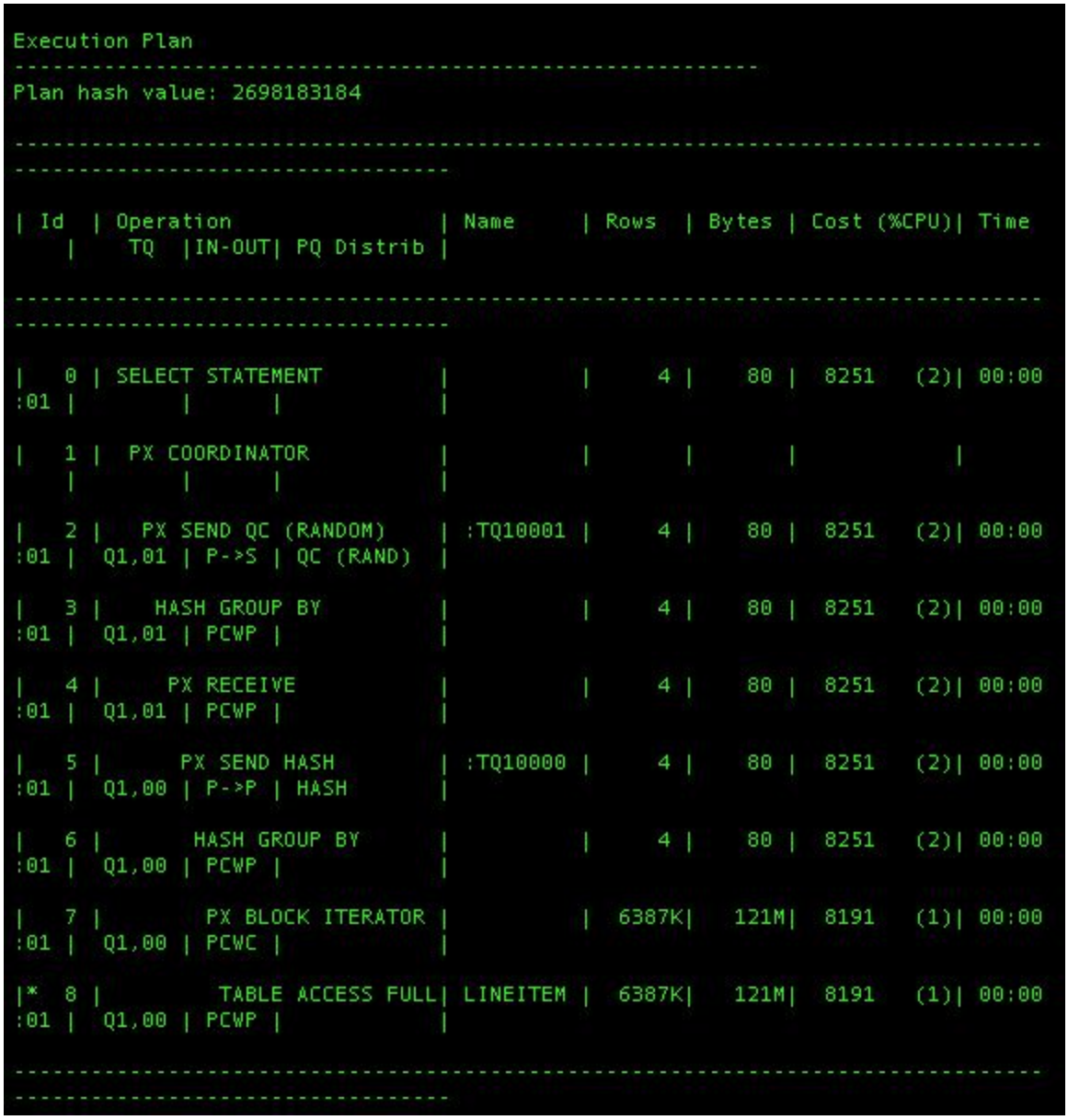
Part 3



Based off the results in part 2, we tried multiple strategies for optimization on the various queries. They included:

1. Use all of the optimizations that resulted in lower run times than the unoptimized run.
2. Use only the optimizations that significantly reduced the run time.
3. Pairing parallelism with the optimizations that worked best.

Query 1 - Statistics, In-memory, Parallelism

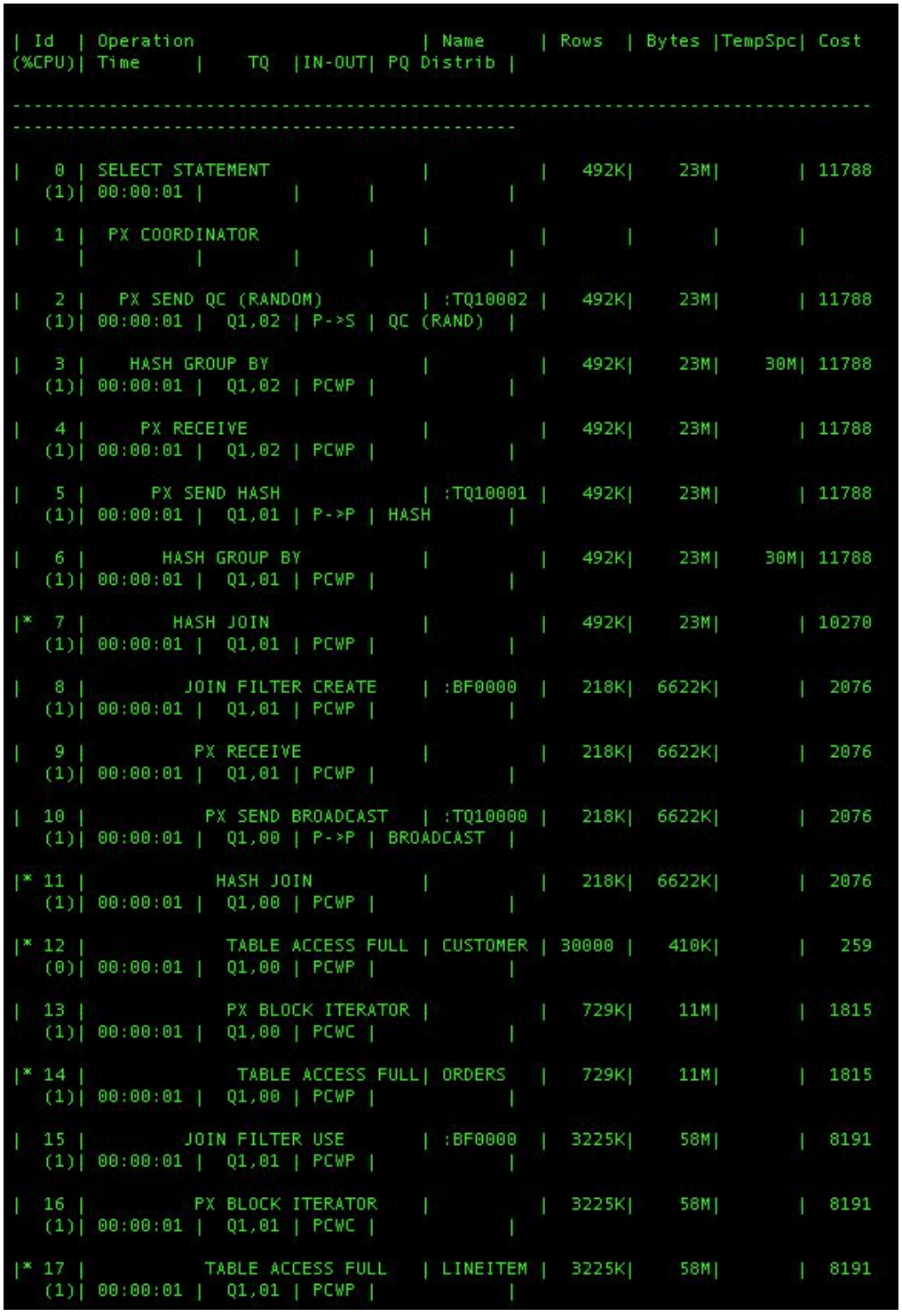


```
Execution Plan
----------------------------------------------------------
Plan hash value: 2698183184

-------------------------------------------------------------------------------
---------------------------------

| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time
    |    TQ  |IN-OUT| PQ Distrib |

-------------------------------------------------------------------------------
---------------------------------

|   0 | SELECT STATEMENT         |          |     4 |    80 |  8251   (2)| 00:00
:01 |        |      |            |

|   1 |  PX COORDINATOR          |          |       |       |            |
    |        |      |            |

|   2 |   PX SEND QC (RANDOM)    | :TQ10001 |     4 |    80 |  8251   (2)| 00:00
:01 |  Q1,01 | P->S | QC (RAND)  |

|   3 |    HASH GROUP BY         |          |     4 |    80 |  8251   (2)| 00:00
:01 |  Q1,01 | PCWP |            |

|   4 |     PX RECEIVE           |          |     4 |    80 |  8251   (2)| 00:00
:01 |  Q1,01 | PCWP |            |

|   5 |      PX SEND HASH        | :TQ10000 |     4 |    80 |  8251   (2)| 00:00
:01 |  Q1,00 | P->P | HASH       |

|   6 |       HASH GROUP BY      |          |     4 |    80 |  8251   (2)| 00:00
:01 |  Q1,00 | PCWP |            |

|   7 |        PX BLOCK ITERATOR |          |  6387K|   121M|  8191   (1)| 00:00
:01 |  Q1,00 | PCWC |            |

|*  8 |         TABLE ACCESS FULL| LINEITEM |  6387K|   121M|  8191   (1)| 00:00
:01 |  Q1,00 | PCWP |            |

-------------------------------------------------------------------------------
---------------------------------
```

Comments

- Statistics and keeping the tables in-memory seem to have improved the runtime equally.
- Oddly, only using the materialized view improved run time, but using the materialized view with these three optimizations increased run time.

Query 3 - Statistics, Parallelism

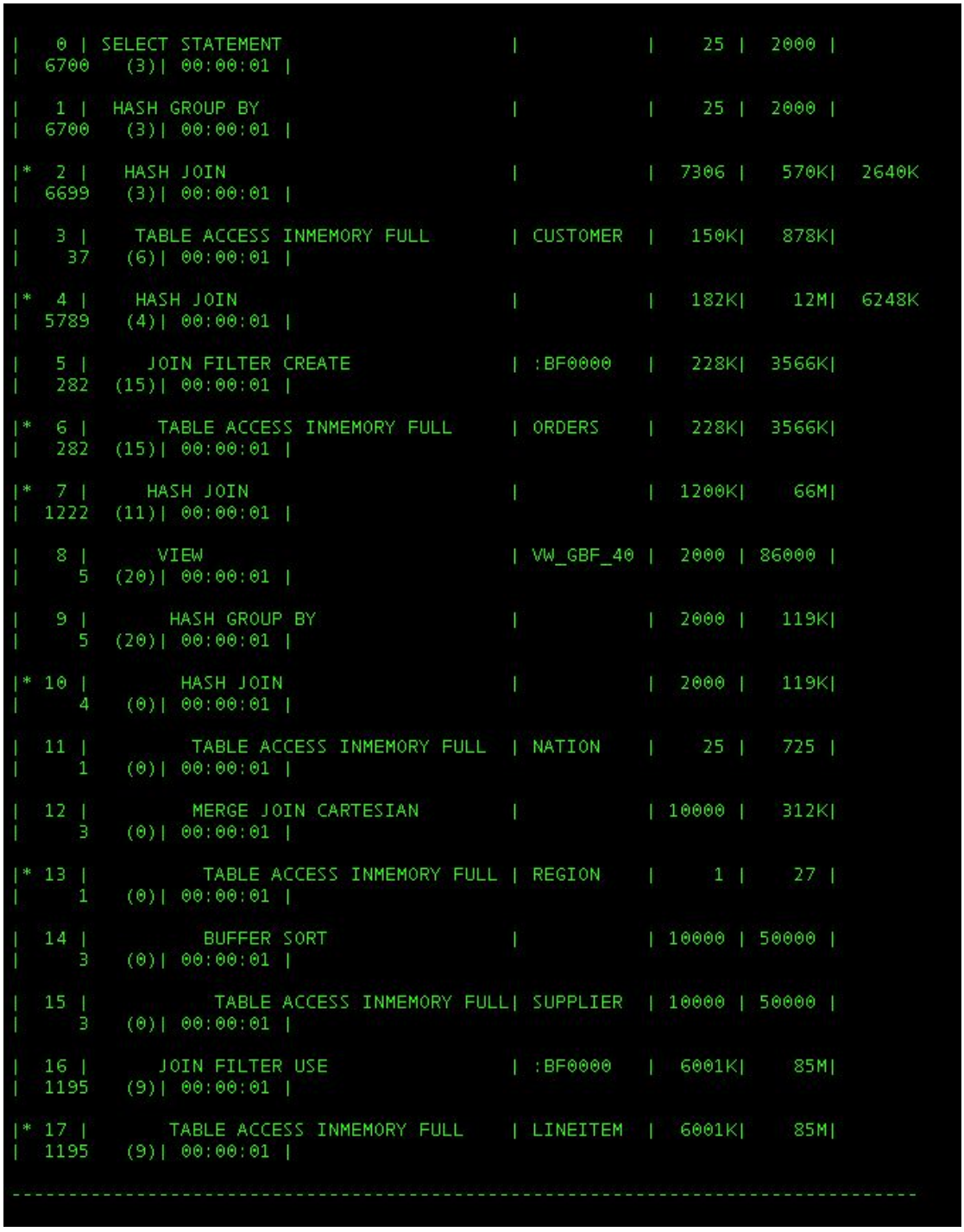


```
| Id  | Operation                    | Name     | Rows  | Bytes |TempSpc| Cost
(%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |

-------------------------------------------------------------------------------
-----------------------------------------------

|   0 | SELECT STATEMENT             |          |   492K|    23M|       | 11788
   (1)| 00:00:01 |        |      |            |

|   1 |  PX COORDINATOR              |          |       |       |       |
      |          |        |      |            |

|   2 |   PX SEND QC (RANDOM)        | :TQ10002 |   492K|    23M|       | 11788
   (1)| 00:00:01 |  Q1,02 | P->S | QC (RAND)  |

|   3 |    HASH GROUP BY             |          |   492K|    23M|    30M| 11788
   (1)| 00:00:01 |  Q1,02 | PCWP |            |

|   4 |     PX RECEIVE               |          |   492K|    23M|       | 11788
   (1)| 00:00:01 |  Q1,02 | PCWP |            |

|   5 |      PX SEND HASH            | :TQ10001 |   492K|    23M|       | 11788
   (1)| 00:00:01 |  Q1,01 | P->P | HASH       |

|   6 |       HASH GROUP BY          |          |   492K|    23M|    30M| 11788
   (1)| 00:00:01 |  Q1,01 | PCWP |            |

|*  7 |        HASH JOIN             |          |   492K|    23M|       | 10270
   (1)| 00:00:01 |  Q1,01 | PCWP |            |

|   8 |         JOIN FILTER CREATE   | :BF0000  |   218K|  6622K|       |  2076
   (1)| 00:00:01 |  Q1,01 | PCWP |            |

|   9 |          PX RECEIVE          |          |   218K|  6622K|       |  2076
   (1)| 00:00:01 |  Q1,01 | PCWP |            |

|  10 |           PX SEND BROADCAST  | :TQ10000 |   218K|  6622K|       |  2076
   (1)| 00:00:01 |  Q1,00 | P->P | BROADCAST  |

|* 11 |            HASH JOIN         |          |   218K|  6622K|       |  2076
   (1)| 00:00:01 |  Q1,00 | PCWP |            |

|* 12 |             TABLE ACCESS FULL | CUSTOMER | 30000 |   410K|       |   259
   (0)| 00:00:01 |  Q1,00 | PCWP |            |

|  13 |             PX BLOCK ITERATOR |          |   729K|    11M|       |  1815
   (1)| 00:00:01 |  Q1,00 | PCWC |            |

|* 14 |              TABLE ACCESS FULL| ORDERS   |   729K|    11M|       |  1815
   (1)| 00:00:01 |  Q1,00 | PCWP |            |

|  15 |         JOIN FILTER USE      | :BF0000  |  3225K|    58M|       |  8191
   (1)| 00:00:01 |  Q1,01 | PCWP |            |

|  16 |          PX BLOCK ITERATOR   |          |  3225K|    58M|       |  8191
   (1)| 00:00:01 |  Q1,01 | PCWC |            |

|* 17 |           TABLE ACCESS FULL  | LINEITEM |  3225K|    58M|       |  8191
   (1)| 00:00:01 |  Q1,01 | PCWP |            |
```

Comments

- Statistics (1.946) and parallelism (1.88) alone did not produce significantly better results compared to the original (1.932); however, when combined, there was a big increase in performance (0.79).

Query 5 - Statistics, In Memory

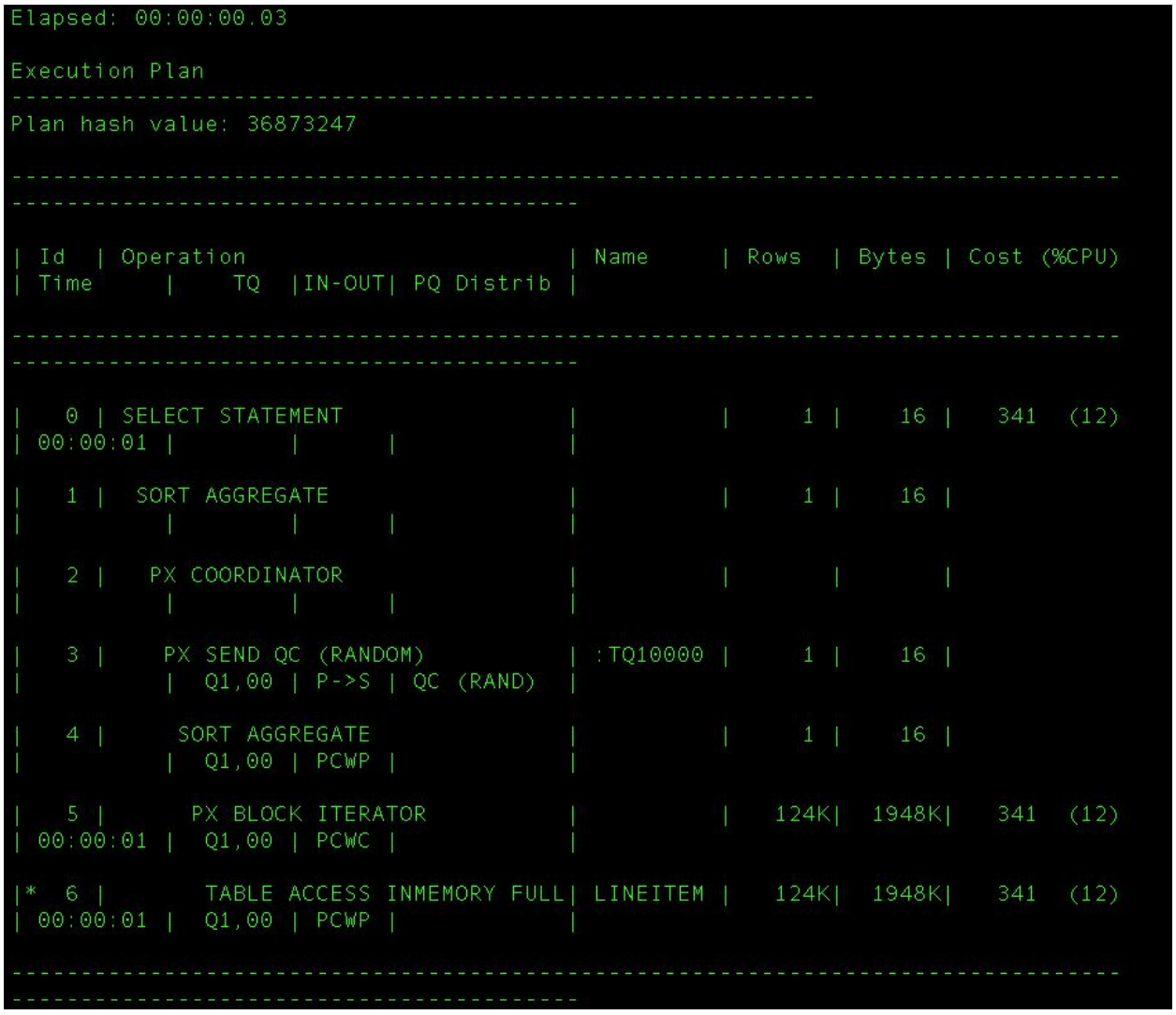


```
|   0 | SELECT STATEMENT                    |           |    25 |  2000 |
|  6700   (3)| 00:00:01 |

|   1 |  HASH GROUP BY                      |           |    25 |  2000 |
|  6700   (3)| 00:00:01 |

|*  2 |   HASH JOIN                         |           |  7306 |   570K|  2640K
|  6699   (3)| 00:00:01 |

|   3 |    TABLE ACCESS INMEMORY FULL       | CUSTOMER  |   150K|   878K|
|    37   (6)| 00:00:01 |

|*  4 |    HASH JOIN                        |           |   182K|    12M|  6248K
|  5789   (4)| 00:00:01 |

|   5 |     JOIN FILTER CREATE              | :BF0000   |   228K|  3566K|
|   282  (15)| 00:00:01 |

|*  6 |      TABLE ACCESS INMEMORY FULL     | ORDERS    |   228K|  3566K|
|   282  (15)| 00:00:01 |

|*  7 |     HASH JOIN                       |           |  1200K|    66M|
|  1222  (11)| 00:00:01 |

|   8 |      VIEW                           | VW_GBF_40 |  2000 | 86000 |
|     5  (20)| 00:00:01 |

|   9 |       HASH GROUP BY                 |           |  2000 |   119K|
|     5  (20)| 00:00:01 |

|* 10 |        HASH JOIN                    |           |  2000 |   119K|
|     4   (0)| 00:00:01 |

|  11 |         TABLE ACCESS INMEMORY FULL  | NATION    |    25 |   725 |
|     1   (0)| 00:00:01 |

|  12 |         MERGE JOIN CARTESIAN        |           | 10000 |   312K|
|     3   (0)| 00:00:01 |

|* 13 |          TABLE ACCESS INMEMORY FULL | REGION    |     1 |    27 |
|     1   (0)| 00:00:01 |

|  14 |          BUFFER SORT                |           | 10000 | 50000 |
|     3   (0)| 00:00:01 |

|  15 |           TABLE ACCESS INMEMORY FULL| SUPPLIER  | 10000 | 50000 |
|     3   (0)| 00:00:01 |

|  16 |       JOIN FILTER USE               | :BF0000   |  6001K|    85M|
|  1195   (9)| 00:00:01 |

|* 17 |        TABLE ACCESS INMEMORY FULL   | LINEITEM  |  6001K|    85M|
|  1195   (9)| 00:00:01 |

--------------------------------------------------------------------------------
```

Comments

- Like for query 1, combining the memory view with these two optimizations increased run time. Also like in query 1, statistics and in memory view seemed to improve performance equally.

Query 6 - Statistics, In Memory, Parallelism



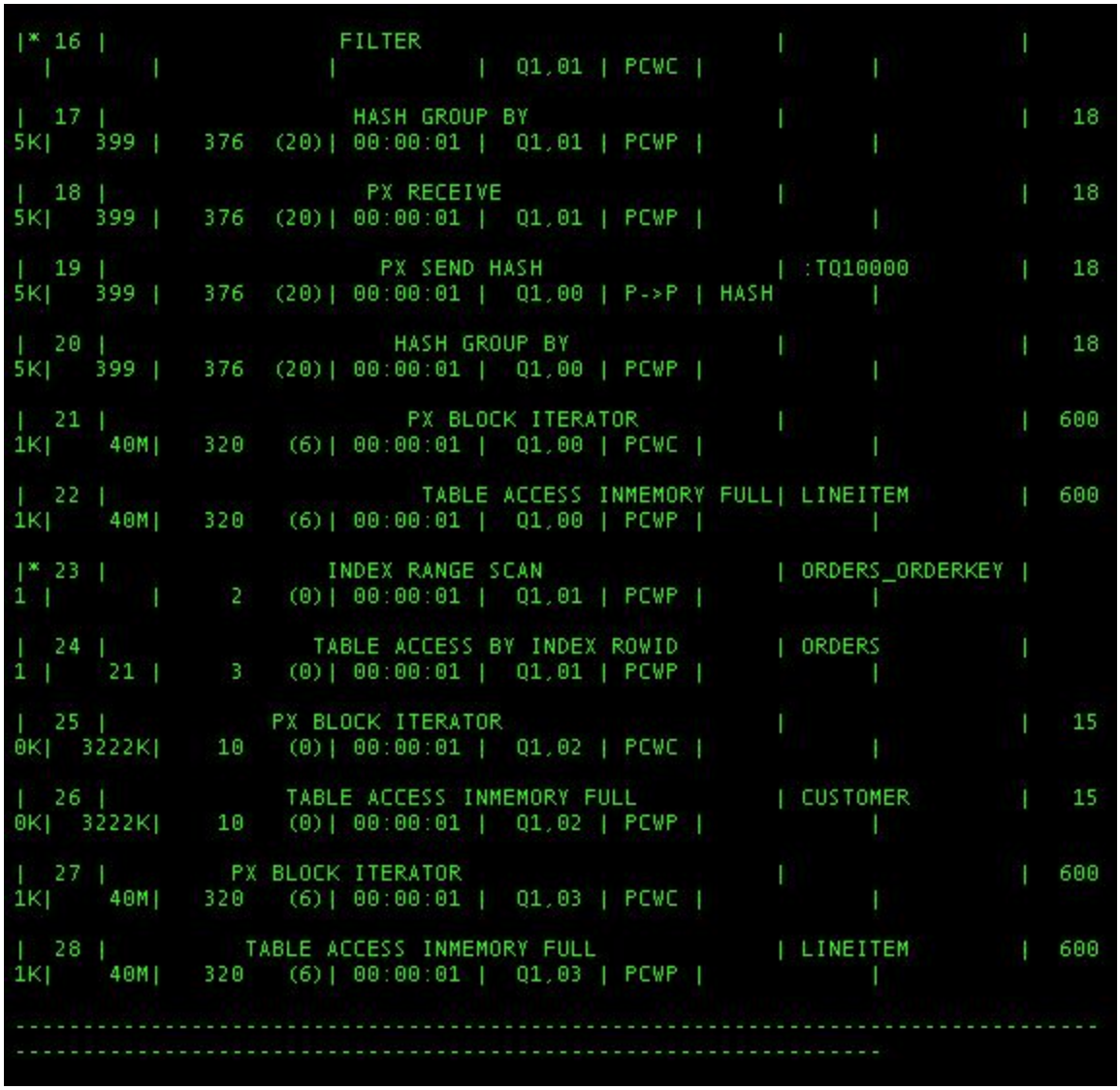
```
Elapsed: 00:00:00.03

Execution Plan
----------------------------------------------------------
Plan hash value: 36873247

-------------------------------------------------------------------------------------------
-----------------------------------------

| Id  | Operation                         | Name     | Rows  | Bytes | Cost (%CPU)
| Time     |    TQ  |IN-OUT| PQ Distrib |

-------------------------------------------------------------------------------------------
-----------------------------------------

|   0 | SELECT STATEMENT                  |          |     1 |    16 |   341  (12)
| 00:00:01 |        |      |            |

|   1 |  SORT AGGREGATE                   |          |     1 |    16 |
|          |        |      |            |

|   2 |   PX COORDINATOR                  |          |       |       |
|          |        |      |            |

|   3 |    PX SEND QC (RANDOM)            | :TQ10000 |     1 |    16 |
|          |  Q1,00 | P->S | QC (RAND)  |

|   4 |     SORT AGGREGATE                |          |     1 |    16 |
|          |  Q1,00 | PCWP |            |

|   5 |      PX BLOCK ITERATOR            |          |   124K|  1948K|   341  (12)
| 00:00:01 |  Q1,00 | PCWC |            |

|*  6 |       TABLE ACCESS INMEMORY FULL| LINEITEM |   124K|  1948K|   341  (12)
| 00:00:01 |  Q1,00 | PCWP |            |

-------------------------------------------------------------------------------------------
-----------------------------------------
```

Comments

- This was by far the best optimization we had during the bake-off. It is worth noting that this may be because caching was very effective for this query; the first run took .97 seconds, and subsequent runs took around .03 seconds.
    - However, caching was also very effective for the runner-up setup (statistics and parallelism), where the first run was 1.48 seconds and subsequent runs were about .29 seconds
- It seemed that statistics made a huge difference in performance. Only in memory tables and parallelism ran for 1.45 seconds, but this setup of memory tables, parallelism, statistics ran in .03 seconds.
- Indexing, when combined with other optimizations, harmed performance

Query 18 - Statistics, In Memory, Parallelism, Indexing

```
Elapsed: 00:00:00.75

Execution Plan
----------------------------------------------------------
Plan hash value: 2239335018

-------------------------------------------------------------------------------
-------------------------------------------------------------

| Id  | Operation                                | Name            | Rows
  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |

-------------------------------------------------------------------------------
-------------------------------------------------------------

|   0 | SELECT STATEMENT                         |                 |   12
0 |  6600 |   746  (14)| 00:00:01 |        |      |            |

|   1 |  PX COORDINATOR                          |                 |
  |       |            |          |        |      |            |

|   2 |   PX SEND QC (RANDOM)                    | :TQ10004        |   12
0 |  6600 |   746  (14)| 00:00:01 |  Q1,04 | P->S | QC (RAND)  |

|   3 |    HASH GROUP BY                         |                 |   12
0 |  6600 |   746  (14)| 00:00:01 |  Q1,04 | PCWP |            |

|   4 |     PX RECEIVE                           |                 |   12
0 |  6600 |   746  (14)| 00:00:01 |  Q1,04 | PCWP |            |

|   5 |      PX SEND HASH                        | :TQ10003        |   12
0 |  6600 |   746  (14)| 00:00:01 |  Q1,03 | P->P | HASH       |

|   6 |       HASH GROUP BY                      |                 |   12
0 |  6600 |   746  (14)| 00:00:01 |  Q1,03 | PCWP |            |

|*  7 |        HASH JOIN                         |                 |   12
0 |  6600 |   745  (14)| 00:00:01 |  Q1,03 | PCWP |            |

|   8 |         PX RECEIVE                       |                 |    3
0 |  1440 |   420  (18)| 00:00:01 |  Q1,03 | PCWP |            |

|   9 |          PX SEND BROADCAST               | :TQ10002        |    3
0 |  1440 |   420  (18)| 00:00:01 |  Q1,02 | P->P | BROADCAST  |

|* 10 |           HASH JOIN                      |                 |    3
0 |  1440 |   420  (18)| 00:00:01 |  Q1,02 | PCWP |            |

|  11 |            PX RECEIVE                    |                 |    3
0 |   780 |   410  (19)| 00:00:01 |  Q1,02 | PCWP |            |

|  12 |             PX SEND BROADCAST            | :TQ10001        |    3
0 |   780 |   410  (19)| 00:00:01 |  Q1,01 | P->P | BROADCAST  |

|  13 |              NESTED LOOPS                |                 |    3
0 |   780 |   410  (19)| 00:00:01 |  Q1,01 | PCWP |            |

|  14 |               NESTED LOOPS               |                 |    5
7 |   780 |   410  (19)| 00:00:01 |  Q1,01 | PCWP |            |

|  15 |                VIEW                      | VW_NSO_1        |    5
7 |   285 |   376  (20)| 00:00:01 |  Q1,01 | PCWP |            |
```

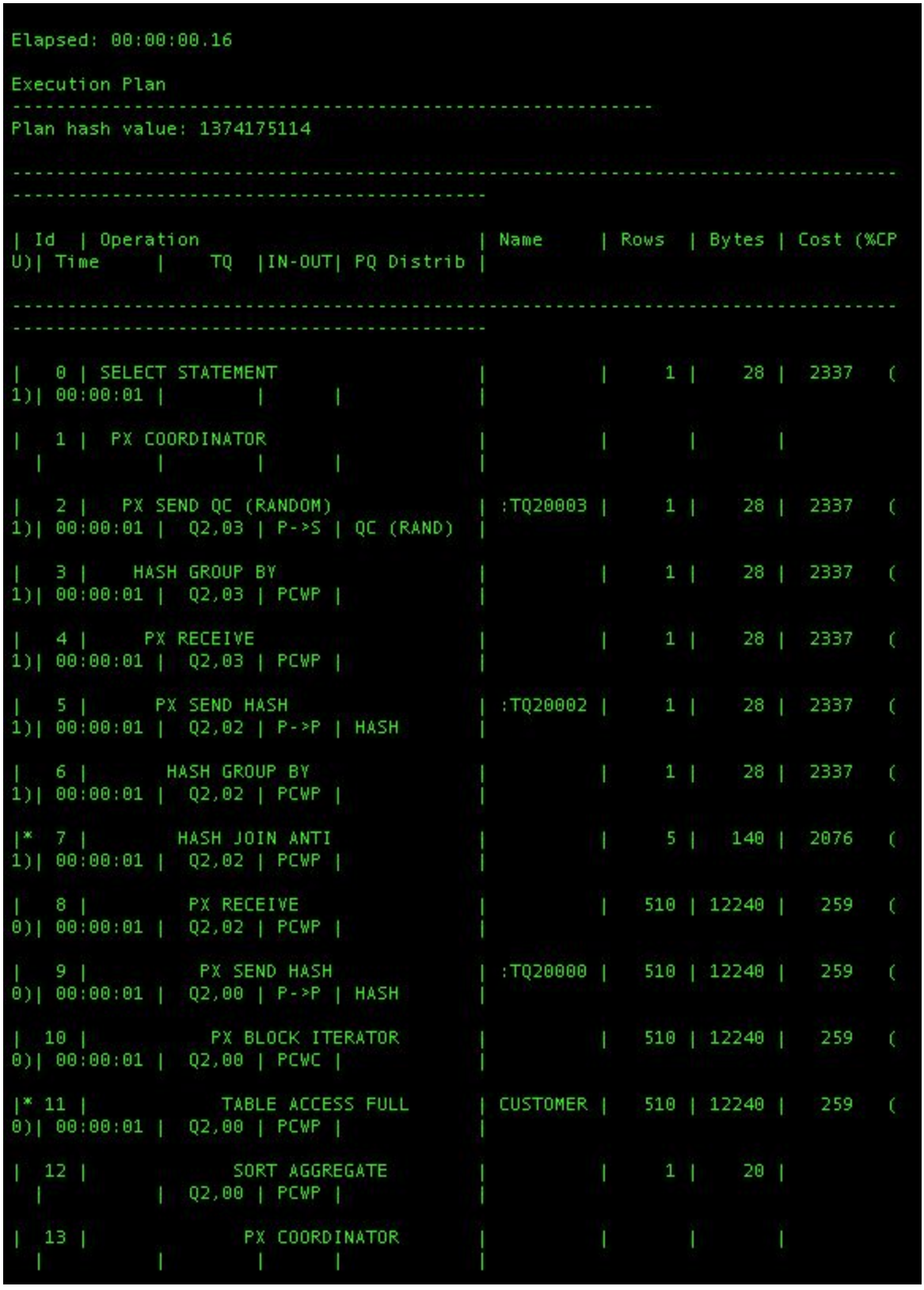


```
|* 16 |                FILTER                          |                 |
  |       |            |          |  Q1,01 | PCWC |            |
|  17 |                  HASH GROUP BY                  |                 |   18
5K|   399 |   376  (20)| 00:00:01 |  Q1,01 | PCWP |            |
|  18 |                   PX RECEIVE                    |                 |   18
5K|   399 |   376  (20)| 00:00:01 |  Q1,01 | PCWP |            |
|  19 |                    PX SEND HASH                 | :TQ10000        |   18
5K|   399 |   376  (20)| 00:00:01 |  Q1,00 | P->P | HASH       |
|  20 |                     HASH GROUP BY               |                 |   18
5K|   399 |   376  (20)| 00:00:01 |  Q1,00 | PCWP |            |
|  21 |                      PX BLOCK ITERATOR          |                 |  600
1K|    40M|   320   (6)| 00:00:01 |  Q1,00 | PCWC |            |
|  22 |                       TABLE ACCESS INMEMORY FULL| LINEITEM        |  600
1K|    40M|   320   (6)| 00:00:01 |  Q1,00 | PCWP |            |
|* 23 |                INDEX RANGE SCAN                 | ORDERS_ORDERKEY |
1 |       |     2   (0)| 00:00:01 |  Q1,01 | PCWP |            |
|  24 |                 TABLE ACCESS BY INDEX ROWID     | ORDERS          |
1 |    21 |     3   (0)| 00:00:01 |  Q1,01 | PCWP |            |
|  25 |              PX BLOCK ITERATOR                  |                 |   15
0K|  3222K|    10   (0)| 00:00:01 |  Q1,02 | PCWC |            |
|  26 |               TABLE ACCESS INMEMORY FULL        | CUSTOMER        |   15
0K|  3222K|    10   (0)| 00:00:01 |  Q1,02 | PCWP |            |
|  27 |          PX BLOCK ITERATOR                      |                 |  600
1K|    40M|   320   (6)| 00:00:01 |  Q1,03 | PCWC |            |
|  28 |            TABLE ACCESS INMEMORY FULL           | LINEITEM        |  600
1K|    40M|   320   (6)| 00:00:01 |  Q1,03 | PCWP |            |

---------------------------------------------------------------------------
---------------------------------------------------------------
```

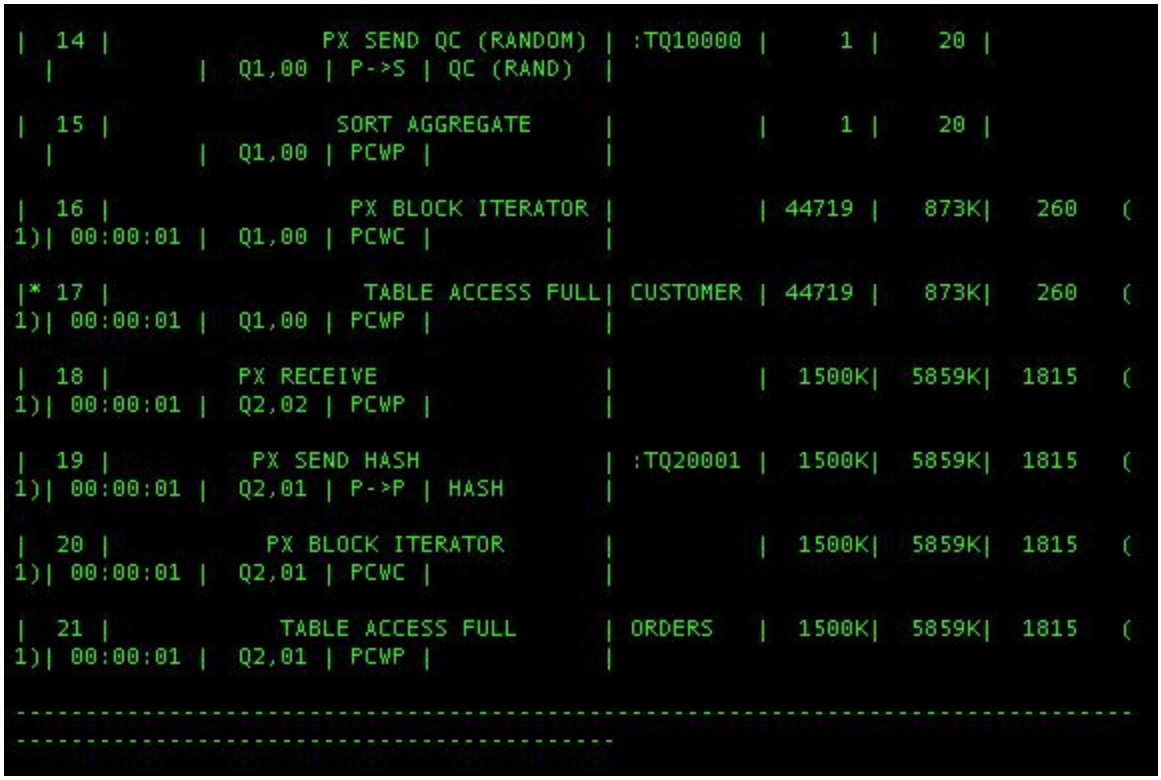
Comments

- This performance was not significantly better than just in memory tables and parallelism (.76 seconds vs .81 seconds)

Query 22 - Statistics, Parallelism



```
Elapsed: 00:00:00.16

Execution Plan
----------------------------------------------------------
Plan hash value: 1374175114

-------------------------------------------------------------------------------------
-----------------------------------------

| Id  | Operation                          | Name     | Rows  | Bytes | Cost (%CP
U)| Time     |    TQ  |IN-OUT| PQ Distrib |

-------------------------------------------------------------------------------------
-----------------------------------------

|   0 | SELECT STATEMENT                   |          |     1 |    28 |  2337   (
1)| 00:00:01 |        |      |            |

|   1 |  PX COORDINATOR                    |          |       |       |
  |          |        |      |            |

|   2 |   PX SEND QC (RANDOM)              | :TQ20003 |     1 |    28 |  2337   (
1)| 00:00:01 |  Q2,03 | P->S | QC (RAND)  |

|   3 |    HASH GROUP BY                   |          |     1 |    28 |  2337   (
1)| 00:00:01 |  Q2,03 | PCWP |            |

|   4 |     PX RECEIVE                     |          |     1 |    28 |  2337   (
1)| 00:00:01 |  Q2,03 | PCWP |            |

|   5 |      PX SEND HASH                  | :TQ20002 |     1 |    28 |  2337   (
1)| 00:00:01 |  Q2,02 | P->P | HASH       |

|   6 |       HASH GROUP BY                |          |     1 |    28 |  2337   (
1)| 00:00:01 |  Q2,02 | PCWP |            |

|*  7 |        HASH JOIN ANTI              |          |     5 |   140 |  2076   (
1)| 00:00:01 |  Q2,02 | PCWP |            |

|   8 |         PX RECEIVE                 |          |   510 | 12240 |   259   (
0)| 00:00:01 |  Q2,02 | PCWP |            |

|   9 |          PX SEND HASH              | :TQ20000 |   510 | 12240 |   259   (
0)| 00:00:01 |  Q2,00 | P->P | HASH       |

|  10 |           PX BLOCK ITERATOR        |          |   510 | 12240 |   259   (
0)| 00:00:01 |  Q2,00 | PCWC |            |

|* 11 |            TABLE ACCESS FULL       | CUSTOMER |   510 | 12240 |   259   (
0)| 00:00:01 |  Q2,00 | PCWP |            |

|  12 |             SORT AGGREGATE         |          |     1 |    20 |
  |          |  Q2,00 | PCWP |            |

|  13 |              PX COORDINATOR        |          |       |       |
  |          |        |      |            |
```



Comments

- In-memory tables, when combined with other optimizations, harmed the performance of this query.

Concluding comments

- Since the statistics are precomputed, the greater amount of information while running a query will almost always improve performance.
- Putting the tables in memory should have also led to a general performance increase, since the I/O of memory is significantly faster than that of hard drives.
- Furthermore, parallelism should generally increase performance, though not always linearly, because of the greater number of cores processing the query at the same time.
- Indexing and creating memory views did not always improve performance. It may have been because we did not create the most optimal indices and memory views.

Thank you!