CSCI317 – Report2 - Samuel Ian Black – SIB979 - 6025821

Table of Contents

CSCI317 – Report2 - Samuel Ian Black – SIB979 - 6025821

Original cost of query

```
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |  1027 | 71890 |       | 17975   (1)| 00:00:01 |
|   1 |  SORT GROUP BY      |          |  1027 | 71890 |       | 17975   (1)| 00:00:01 |
|*  2 |   HASH JOIN SEMI    |          |  1027 | 71890 |    13M| 17974   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| ORDERS   |   272K|    10M|       |  2697   (1)| 00:00:01 |
|*  4 |    TABLE ACCESS FULL| LINEITEM |  1205K|    35M|       | 12159   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Total Cost = 17,975 + 17,975 + 17,974 + 2,697 + 12,159 = **68,780**

CSCI317 – Report2 - Samuel Ian Black – SIB979 - 6025821

Optimization #1 – Creation of Bitmap Index

Note that I decided against using Materialized Views as we're unsure of whether or not the client's DB has designated downtime to mitigate the more expensive insertion, and if the client doesn't have a designated downtime for the DB to perform inserts and commits to update the materialized views and instead wants the views to remain up to date and therefore, updated on insert, if the table LINEITEM is frequently inserted to which is what I assume since It's the largest table, the performance of the entire DB will greatly suffer which is why I've decided not to use Materialized Views.

(1) Description of Improvement:
I created a bitmap index over the `O_ORDERDATE` column;
The total number of rows in a table `ORDERS` is 450,000
the total number of distinct rows of a column `O_ORDERDATE` is 2,406
meaning that the cardinality of `O_ORDERDATE` = (2,406/450,000) * 100 = 0.53% making the `O_ORDERDATE` a good column for a bitmap to be created.

```
CREATE BITMAP INDEX TASK2IDX1 ON ORDERS(O_ORDERDATE);
```

(2) Benefits of Improvement:
The bitmap index significantly improves the performance of the query since the ORDERS table is relatively large.
Rather than having to traverse the entire ORDERS table, the query optimizer can instead traverse the bitmap index.

```
SQL> SET FEEDBACK ON
SQL> SET LINESIZE 300
SQL> SET PAGESIZE 300
SQL>
SQL> CREATE BITMAP INDEX TASK2IDX1 ON ORDERS(O_ORDERDATE);

INDEX TASK2IDX1 created.

SQL>
SQL> EXPLAIN PLAN FOR
  2  SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
  3  FROM ORDERS
  4  WHERE O_ORDERDATE >= '13-OCT-1994'
  5  AND O_ORDERDATE <= '19-FEB-2004'
  6  AND EXISTS (
  7      SELECT *
  8      FROM LINEITEM
  9      WHERE L_ORDERKEY = O_ORDERKEY
```

```
 10          AND L_COMMITDATE < L_RECEIPTDATE
 11       )
 12  GROUP BY
 13       O_ORDERPRIORITY
 14  ORDER BY
 15       O_ORDERPRIORITY;

Explained.

SQL>
SQL> @showplan
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------
Plan hash value: 1140149199

--------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |           |  1010 | 70700 |       | 15651   (1)| 00:00:01 |
|   1 |  SORT GROUP BY                         |           |  1010 | 70700 |       | 15651   (1)| 00:00:01 |
|*  2 |   HASH JOIN SEMI                       |           |  1010 | 70700 |    13M| 15650   (1)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS    |   269K|    10M|       |   352   (0)| 00:00:01 |
|   4 |     BITMAP CONVERSION TO ROWIDS        |           |       |       |       |            |          |
|*  5 |      BITMAP INDEX RANGE SCAN           | TASK2IDX1 |       |       |       |            |          |
|*  6 |    TABLE ACCESS FULL                   | LINEITEM  |  1221K|    36M|       | 12159   (1)| 00:00:01 |
--------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
```

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

   2 - access("L_ORDERKEY"="O_ORDERKEY")
   5 - access("O_ORDERDATE">=TO_DATE(' 1994-10-13 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "O_ORDERDATE"<=TO_DATE(' 2004-02-19 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   6 - filter("L_COMMITDATE"<"L_RECEIPTDATE")
```

Total Cost before improvement = **68,780**

Total Cost after improvement = 15,651 + 15,651 + 15,650 + 352 + 12,159 = **59,463**

Total Cost improvement = 68,780 **-** 59,463 = **9,317**

(3) Costs of Improvement:
The cost of creating the index is 1.75MB in persistent storage.

```
SQL> select sum(bytes)/1024/1024 as "Index Size (MB)" from dba_segments where segment_name='&INDEX_NAME';
old:select sum(bytes)/1024/1024 as "Index Size (MB)" from dba_segments where segment_name='&INDEX_NAME'
new:select sum(bytes)/1024/1024 as "Index Size (MB)" from dba_segments where segment_name='TASK2IDX1'

Index Size (MB)
---------------
           1.75
```

(4) Report from improvement:

```
SQL> SET ECHO ON
SQL> SET FEEDBACK ON
SQL> SET LINESIZE 300
SQL> SET PAGESIZE 300
SQL> CREATE BITMAP INDEX TASK2IDX1 ON ORDERS(O_ORDERDATE);

INDEX TASK2IDX1 created.

SQL>
SQL> SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
  2  FROM ORDERS
  3  WHERE O_ORDERDATE >= '13-OCT-1994'
  4  AND O_ORDERDATE <= '19-FEB-2004'
  5  AND EXISTS (
  6      SELECT *
  7      FROM LINEITEM
  8      WHERE L_ORDERKEY = O_ORDERKEY
  9          AND L_COMMITDATE < L_RECEIPTDATE
 10      )
 11  GROUP BY
 12       O_ORDERPRIORITY
 13  ORDER BY
 14       O_ORDERPRIORITY;

O_ORDERPRIORITY ORDER_COUNT
--------------- -----------
1-URGENT              47975
2-HIGH                48083
3-MEDIUM              47622
```

```
4-NOT SPECIFIED       47820
5-LOW                 47778

5 rows selected.
```

CSCI317 – Report2 - Samuel Ian Black – SIB979 - 6025821

Optimization #2 – Creation of B*Tree Index

(1) Description of Improvement:

I created a compressed composite B*Tree index over the `L_ORDERKEY, L_COMMITDATE, L_RECEIPTDATE` columns; compression of the first column saves 5MB of persistent storage and reduces the number of logical reads compared to not compressing the first column. This is because the column `L_ORDERKEY` is a foreign key and is therefore, often repeated.

```
CREATE INDEX TASK2IDX2 ON LINEITEM(L_ORDERKEY, L_COMMITDATE, L_RECEIPTDATE) COMPRESS 1;
```

(2) Benefits of Improvement:

The B*Tree index significantly improves the performance of the query since the ORDERS table is relatively large.
Rather than having to traverse the entire ORDERS table, the query optimizer can instead traverse the bitmap index.

```
SQL> SET ECHO ON
SQL> SET FEEDBACK ON
SQL> SET LINESIZE 300
SQL> SET PAGESIZE 300
SQL> CREATE INDEX TASK2IDX2 ON LINEITEM(L_ORDERKEY, L_COMMITDATE, L_RECEIPTDATE) COMPRESS 1;


Index TASK2IDX2 created.
```

```
SQL>
SQL> EXPLAIN PLAN FOR
  2  SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
  3  FROM ORDERS
  4  WHERE O_ORDERDATE >= '13-OCT-1994'
  5  AND O_ORDERDATE <= '19-FEB-2004'
  6  AND EXISTS (
  7      SELECT *
  8      FROM LINEITEM
  9      WHERE L_ORDERKEY = O_ORDERKEY
 10          AND L_COMMITDATE < L_RECEIPTDATE
 11       )
 12  GROUP BY
 13       O_ORDERPRIORITY
```

```
 14  ORDER BY
 15        O_ORDERPRIORITY;

Explained.

SQL>
SQL> @showplan
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------------------

Plan hash value: 26775457

-----------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |           |  1010 | 70700 |       |  6375   (1)| 00:00:01 |
|   1 |  SORT GROUP BY                         |           |  1010 | 70700 |       |  6375   (1)| 00:00:01 |
|*  2 |   HASH JOIN SEMI                       |           |  1010 | 70700 |    13M|  6374   (1)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS    |   269K|    10M|       |   352   (0)| 00:00:01 |
|   4 |     BITMAP CONVERSION TO ROWIDS        |           |       |       |       |            |          |
|*  5 |      BITMAP INDEX RANGE SCAN           | TASK2IDX1 |       |       |       |            |          |
|*  6 |    INDEX FAST FULL SCAN                | TASK2IDX2 |  1221K|    36M|       |  2883   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("L_ORDERKEY"="O_ORDERKEY")
   5 - access("O_ORDERDATE">=TO_DATE(' 1994-10-13 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
```

```
              "O_ORDERDATE"<=TO_DATE(' 2004-02-19 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   6 - filter("L_COMMITDATE"<"L_RECEIPTDATE")
```

Total Cost before improvement = **59,463**

Total Cost after improvement = 6,375 + 6,375 + 6,374 + 352 + 2,883 = **22,359**

Total Cost improvement = 59,463 - 22,359 = **37,104**

(3) Costs of Improvement:
The cost of creating the index is 60MB in persistent storage.

```
SQL> select sum(bytes)/1024/1024 as "Index Size (MB)" from dba_segments where segment_name='&INDEX_NAME';
old:select sum(bytes)/1024/1024 as "Index Size (MB)" from dba_segments where segment_name='&INDEX_NAME'
new:select sum(bytes)/1024/1024 as "Index Size (MB)" from dba_segments where segment_name='TASK2IDX2'

Index Size (MB)
---------------
             60
```

(4) Report from improvement:

```
SQL> SET ECHO ON
SQL> SET FEEDBACK ON
SQL> SET LINESIZE 300
SQL> SET PAGESIZE 300
SQL> CREATE INDEX TASK2IDX2 ON LINEITEM(L_ORDERKEY, L_COMMITDATE, L_RECEIPTDATE) COMPRESS 1;

Index TASK2IDX2 created.

SQL>
SQL> SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
  2  FROM ORDERS
  3  WHERE O_ORDERDATE >= '13-OCT-1994'
  4  AND O_ORDERDATE <= '19-FEB-2004'
  5  AND EXISTS (
  6      SELECT *
  7      FROM LINEITEM
  8      WHERE L_ORDERKEY = O_ORDERKEY
  9          AND L_COMMITDATE < L_RECEIPTDATE
 10      )
 11  GROUP BY
 12       O_ORDERPRIORITY
 13  ORDER BY
 14       O_ORDERPRIORITY;

O_ORDERPRIORITY ORDER_COUNT
--------------- -----------
1-URGENT              47975
2-HIGH                48083
3-MEDIUM              47622
```

```
4-NOT SPECIFIED       47820
5-LOW                 47778

5 rows selected.
```

Optimization #3 – Creation of 32K Tablespace, Allocation of db_32K_cache_size, and Rebuild of TASK2IDX2 In New Tablespace

(1) Description of Improvement:

Because TASK2IDX2 is large at 60MB and is a B*Tree index, I can take full advantage of storing it in the 32k data buffer cache which will better balance the tree and reduce physical and logical reads of the query due to “flattening” the tree over larger data blocks.

Source#1: [Performance Tuning of Relational Database Server (1) - Slide 20 of 36]
Source#2: [https://www.techrepublic.com/article/creating-tablespaces-with-multiple-block-sizes/]
Source #3: [https://support.esri.com/en/technical-article/000011463]
Source#4: [https://oracle-base.com/articles/9i/multiple-block-sizes]
Source #5: [https://searchoracle.techtarget.com/tip/Creating-an-Oracle-index-cache]
Source #6: [http://www.dba-oracle.com/oracle_tips_multiple_blocksizes.htm – Reducing logical I/O]

```
SQL> ALTER SYSTEM SET db_32K_cache_size = 64M SCOPE=SPFILE;


System SET altered.


SQL> CREATE TABLESPACE INDEX_TS_32K
  2  BLOCKSIZE 32K
  3  DATAFILE '/opt/oracle/oradata/DB/32k_tbs.dbf'
  4  SIZE 64M AUTOEXTEND ON
  5  extent management local;

TABLESPACE INDEX_TS_32K created.


SQL> ALTER USER TPCHR QUOTA 64M ON INDEX_TS_32K;


User TPCHR altered.
```

Note that I have allocated 64MB to the 32k data buffer cache and 32K block size Tablespace instead of 60MB because I am aware of the improvements on another index I have created in task 5.

(2) Benefits of Improvement:

The B*Tree is now more balanced which results in significantly less data blocks needed to be read resulting in less physical reads and reduced Cost of performing the query.

Explain Plan

```
SQL> SET ECHO ON
SQL> SET FEEDBACK ON
SQL> SET LINESIZE 300
SQL> SET PAGESIZE 300
SQL>
SQL> ALTER INDEX TASK2IDX2 REBUILD TABLESPACE INDEX_TS_32K;

Index TASK2IDX2 altered.

SQL>
SQL> EXPLAIN PLAN FOR
  2  SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
  3  FROM ORDERS
  4  WHERE O_ORDERDATE >= '13-OCT-1994'
  5  AND O_ORDERDATE <= '19-FEB-2004'
  6  AND EXISTS (
  7      SELECT *
  8      FROM LINEITEM
  9      WHERE L_ORDERKEY = O_ORDERKEY
 10          AND L_COMMITDATE < L_RECEIPTDATE
 11       )
 12  GROUP BY
 13       O_ORDERPRIORITY
 14  ORDER BY
 15       O_ORDERPRIORITY;
```

```
Explained.

SQL>
SQL> @showplan
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------
Plan hash value: 26775457

--------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |           |  1010 | 70700 |       |  5373   (1)| 00:00:01 |
|   1 |  SORT GROUP BY                         |           |  1010 | 70700 |       |  5373   (1)| 00:00:01 |
|*  2 |   HASH JOIN SEMI                       |           |  1010 | 70700 |    13M|  5372   (1)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS    |   269K|    10M|       |   352   (0)| 00:00:01 |
|   4 |     BITMAP CONVERSION TO ROWIDS        |           |       |       |       |            |          |
|*  5 |      BITMAP INDEX RANGE SCAN           | TASK2IDX1 |       |       |       |            |          |
|*  6 |    INDEX FAST FULL SCAN                | TASK2IDX2 |  1221K|    36M|       |  1881   (1)| 00:00:01 |
--------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("L_ORDERKEY"="O_ORDERKEY")
   5 - access("O_ORDERDATE">=TO_DATE(' 1994-10-13 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "O_ORDERDATE"<=TO_DATE(' 2004-02-19 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   6 - filter("L_COMMITDATE"<"L_RECEIPTDATE")
```

Total Cost before improvement = **22,359**

Total Cost after improvement = 5,373 + 5,373 + 5,372 + 352 + 1,881 = **18,351**

Total Cost improvement = 22,359 - 18,351 = **4,008**

utlbstat/utlestat (Screenshots used because of spacing issues)

Before improvement:

TABLE_SPACE	FILE_NAME	READS	BLKS_READ	READ_TIME	WRITES	BLKS_WRT	WRITE_TIME	MEGABYTES	AVG_RT	blocks/rd
INDEX_TS_32K	/opt/oracle/oradata/DB/32k_tbs.dbf	0	0	0	0	0	0	67	0	0
SYSAUX	/opt/oracle/oradata/DB/sysaux01.dbf	0	0	0	0	0	0	682	0	0
SYSTEM	/opt/oracle/oradata/DB/system01.dbf	10	10	0	0	0	0	954	0	1
TPCHR	/opt/oracle/oradata/DB/tpchr.dbf	2672	15070	30	0	0	0	3146	0	5.64
UNDOTBS1	/opt/oracle/oradata/DB/undotbs01.dbf	0	0	0	0	0	0	357	0	0
USERS	/opt/oracle/oradata/DB/users01.dbf	0	0	0	0	0	0	31	0	0

TPCHR + INDEX_TS_32K READS = 2,672

TPCHR + INDEX_TS_32K BLKS_READ = 15,070

After improvement:

TABLE_SPACE	FILE_NAME	READS	BLKS_READ	READ_TIME	WRITES	BLKS_WRT	WRITE_TIME	MEGABYTES	AVG_RT	blocks/rd
INDEX_TS_32K	/opt/oracle/oradata/DB/32k_tbs.dbf	1839	1839	1	0	0	0	67	0	1
SYSAUX	/opt/oracle/oradata/DB/sysaux01.dbf	13	16	0	0	0	0	682	0	1.23
SYSTEM	/opt/oracle/oradata/DB/system01.dbf	12	12	0	0	0	0	954	0	1
TPCHR	/opt/oracle/oradata/DB/tpchr.dbf	240	7486	1	0	0	0	3146	0	31.19
UNDOTBS1	/opt/oracle/oradata/DB/undotbs01.dbf	2	2	0	0	0	0	357	0	1
USERS	/opt/oracle/oradata/DB/users01.dbf	0	0	0	0	0	0	31	0	0

TPCHR + INDEX_TS_32K READS = 1,839 + 240 = 2,079

TPCHR + INDEX_TS_32K BLKS_READ = 1,839 + 7,486 = 9,325

Total READS improvement = 2,672 – 2,079 = **593**

Total BLKS_READ improvement = 2,672 – 2,079 = **5,745**

(3) Costs of Improvement:

The cost of allocating memory to the 32K data buffer cache is 64MB of transient memory.

```
SQL> SHOW PARAMETER db_32K_cache_size;
NAME                 TYPE        VALUE
-------------------- ----------- -----
db_32k_cache_size big integer 64M
```

The cost of creation of a new Tablespace is 64MB of persistent storage. Rebuilding of the index TASK2IDX2 does not add additional persistent storage, the index is simply rebuilt on the 32K Tablespace.

(4) Report from improvement:

```
SQL> SET ECHO ON
SQL> SET FEEDBACK ON
SQL> SET LINESIZE 300
SQL> SET PAGESIZE 300
SQL>
SQL> ALTER INDEX TASK2IDX2 REBUILD TABLESPACE INDEX_TS_32K;

Index TASK2IDX2 altered.

SQL>
SQL> SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
  2  FROM ORDERS
  3  WHERE O_ORDERDATE >= '13-OCT-1994'
  4  AND O_ORDERDATE <= '19-FEB-2004'
  5  AND EXISTS (
  6      SELECT *
  7      FROM LINEITEM
  8      WHERE L_ORDERKEY = O_ORDERKEY
  9          AND L_COMMITDATE < L_RECEIPTDATE
 10    )
 11  GROUP BY
 12    O_ORDERPRIORITY
 13  ORDER BY
 14    O_ORDERPRIORITY;

O_ORDERPRIORITY ORDER_COUNT
--------------- -----------
1-URGENT              47975
2-HIGH                48083
```

```
3-MEDIUM                47622
4-NOT SPECIFIED         47820
5-LOW                   47778

5 rows selected.
```

CSCI317 – Report2 - Samuel Ian Black – SIB979 - 6025821

Total Costs

Persistent Storage: 72.75MB of 300MB

1. TASK1IDX1 = 6.5MB
2. TASK1IDX2 = 0.5MB
3. TASK2IDX1 = 1.75MB
4. INDEX_TS_32K = 64MB (Size of Tablespace used to calculate Persistent Storage)
    - TASK2IDX2 = 60MB

Transient Memory: 88MB of 100MB

1. db_32K_cache_size = 64M
2. db_cache_size = 208
    - Originally 184 + 24 from allocated 100MB Transient Storage expansion