CSCI317 Database Performance Tuning

SQL Tuning (1)

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

SQLTuning (1)

Outline

When to avoid full table scans ?

(1) Avoid full table scan when access through an index retrieves less than 40-50% of a relational table

(2) Avoid full table scan when it is possible get all needed data from an index, e.g. through fast full index scan, full index scan, or application of index organized tables

SQLTuning (1)

Outline

How to avoid full table scans ?

(1) Influence query optimizer with statistics on distribution of values in the columns of relational tables (`DBMS_STAT`)

(2) Influence query optimizer with system initialization parameters (`OPTIMIZER_MODE`, `OPTIMIZER_INDEX_COST_ADJ`)

(3) Influence query optimizer with the hints

(4) Make sure that conditions are specified correctly, for example `STATUS = 1` instead of `NOT( STATUS <> 1)` or `STATUS IN (2, 3)` instead `STATUS <> 1` when the only values of `STATUS` are `1`, `2`, and `3`

(5) Force full index scan when searching for `NULL`s, for example `STATUS IS NOT NULL`

(6) Do not disable index with a function or expression, for example `FLOAT(STATUS) = 4.0` instead of `STATUS = INT(4.0)`

How to avoid full table scans ?

(7) Avoid unnecessary complex Boolean expressions, for example
`NOT(( STATUS <> 5) OR (PRICE <> 200))` instead use
`(STATUS = 5) AND (PRICE = 200)`

(8) Transform condition to simple form, for example
`(LEVEL = ' A1' ) OR (LEVEL = ' A2' ) OR (LEVEL = ' A3' ) OR (LEVEL = ' A4' ) OR (LEVEL = ' A5' )` into
`LEVEL LIKE ' A_'`

SQLTuning (1)

Outline

How to speed up full table scans ?

(1) Use multiblock transfer (`DB_FILE_MULTIBLOCK_READ_COUNT`)

(2) Use data stripping to spread the data blocks across several partitions on several hard disk and to perform parallel full table scan later on

(3) Use sampling of table scans, for example
`SELECT * FROM PART sample (10)` selects 10% of the rows from `PART` table

(4) Select only required attributes instead of full rows, for example
`SELECT P_NAME FROM PART` instead of `SELECT * FROM PART`

(5) Force projections through temporary tables, for example
`CREATE TABLE PNAME AS SELECT P_NAME FROM PART`
and later on use a table `PNAME` instead of `PART`

(6) Reorganize a database, for example put frequently scanned relational tables on different persistent storage devices, use faster persistent storage devices to keep tables accessed more frequently

How to speed up full table scans ?

(7) Pin frequently accessed relational tables in data buffer cache or bypass data buffer cache when accessing a table

(8) Make I/O operations more efficient through application of specialised file organizations, for example raw partitions or Automatic Segment Management in Oracle

(9) Increase a size of data block (`DB_BLOCK_SIZE`)

(10) Use indexing to eliminate full table scans when selectivity of selections is high

(11) Reduce number of data blocks occupied by the relational tables being fully scanned

(12) Partition the relational tables

(13) Eliminate row "chaining" over multiple blocks

(14) Compress the relational tables

(15) Do not worry about the problem when a relational table is very small and it is not accessed frequently

SQLTuning (1)

Outline

How to speed up counting ?

Counting of the total number of rows in entire table must use an index on primary or an index on candidate key

Counting total number of rows

```
SELECT COUNT(*)
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation             | Name         | Rows | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |              |    1 | 1132    (1)| 00:00:01 |
|   1 |  SORT AGGREGATE       |              |    1 |            |          |
|   2 |   INDEX FAST FULL SCAN| LINEITEM_PKEY | 1800K| 1132    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Note, that `SORT AGGREGATE` does not mean sorting !

How to speed up counting ?

Counting of the total number of rows in a selected group of rows cannot be done with an index on primary key or an index on candidate key

Counting with selection

```
SELECT COUNT(*)
FROM LINEITEM
WHERE L_TAX > 0 OR L_QUANTITY < 100;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |    1 |     3 | 8783    (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |          |    1 |     3 |            |          |
|*  2 |   TABLE ACCESS FULL| LINEITEM | 1600K|  4688K| 8783    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
2 - filter("L_QUANTITY"<100 OR "L_TAX">0)
```

How to speed up counting ?

Counting of the total number of rows in a selected group of rows can be done with an index on all attributes involved in a selection condition

Creating an index

```
CREATE INDEX IDX ON LINEITEM(L_TAX, L_QUANTITY);
```

Counting with selection

```
SELECT COUNT(*)
FROM LINEITEM
WHERE L_TAX > 0 OR L_QUANTITY < 100;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation             | Name | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |      |    1 |     6 | 1159    (1)| 00:00:01 |
|   1 |  SORT AGGREGATE       |      |    1 |     6 |            |          |
|*  2 |   INDEX FAST FULL SCAN| IDX  | 1800K|    10M| 1159    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
2 - filter("L_QUANTITY"<100 OR "L_TAX">0)
```

How to speed up counting ?

Counting of the total number of values in a column returns the same number as counting of the total number of rows when the column is NOT NULL

Counting the values in a non key column

```
SELECT COUNT(L_COMMENT)
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation             | Name         | Rows | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |              |    1 | 1132    (1)| 00:00:01|
|   1 |  SORT AGGREGATE       |              |    1 |            |          |
|   2 |   INDEX FAST FULL SCAN| LINEITEM_PKEY| 1800K| 1132    (1)| 00:00:01|
-------------------------------------------------------------------------------------
```

How to speed up counting ?

Counting of the total number of values in a column may return a different value from the total number of rows when the column is NULL

Make a column NULLable

```
ALTER TABLE LINEITEM MODIFY (L_COMMENT VARCHAR2(44) NULL);
```

Counting values in a NULLable column

```
SELECT COUNT(L_COMMENT)
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |    1 |    28 | 8788    (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |          |    1 |    28 |            |          |
|   2 |   TABLE ACCESS FULL| LINEITEM | 1800K|    48M| 8788    (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

How to speed up counting ?

Counting of the total number of distinct values in a column may return a different value from the total number of rows

Counting distinct values

```
SELECT COUNT(DISTINCT L_COMMENT)
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |     1 |    24 |       | 18893   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE     |          |     1 |    24 |       |            |          |
|   2 |   VIEW              | VW_DAG_0|  1028K|    23M|       | 18893   (1)| 00:00:01 |
|   3 |   HASH GROUP BY     |          |  1028K|    27M|    62M| 18893   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| LINEITEM|  1800K|    48M|       | 8788    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

How to speed up counting ?

Counting of the total number of distinct values in a column uses an index on a selected column

Creating an index

```
CREATE INDEX IDX ON LINEITEM(L_COMMENT);
```

Counting distinct values in NULLable column

```
SELECT COUNT(DISTINCT L_COMMENT)
FROM LINEITEM;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |     1 |    24 | 9690    (1)| 00:00:01 |
|   1 |  SORT AGGREGATE          |          |     1 |    24 |            |          |
|   2 |   VIEW                   | VW_DAG_0 | 1028K|    23M| 9690    (1)| 00:00:01 |
|   3 |    SORT GROUP BY NOSORT  |          | 1028K|    27M| 9690    (1)| 00:00:01 |
|   4 |     INDEX FULL SCAN      | IDX      | 1800K|    48M| 9690    (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

SQLTuning (1)

Outline

How to speed up sorting ?

Sorting is a direct consequence of `ORDER BY` clause

Sorting a relational table

```
SELECT *
FROM LINEITEM
ORDER BY L_SHIPDATE;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          | 1800K|   214M|       | 59138   (1)| 00:00:03 |
|   1 |  SORT ORDER BY     |          | 1800K|   214M|   287M| 59138   (1)| 00:00:03 |
|   2 |   TABLE ACCESS FULL| LINEITEM | 1800K|   214M|       |  8788   (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

How to speed up sorting ?

Sorting in certain circumstances can be avoided through indexing

Creating an index

```
CREATE INDEX IDX ON LINEITEM(L_SHIPDATE);
```

Sorting a relational table

```
SELECT L_SHIPDATE
FROM LINEITEM
ORDER BY L_SHIPDATE;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      | 1800K|    13M| 4775    (1)| 00:00:01 |
|   1 |  INDEX FULL SCAN   |  IDX | 1800K|    13M| 4775    (1)| 00:00:01 |
-------------------------------------------------------------------------------
```

SQLTuning (1)

Outline

How to speed up sorting in top n queries ?

Sorting is required in top n class of queries

Retrieving the first 20 rows

```
SELECT *
FROM ( SELECT L_ORDERKEY, L_LINENUMBER, L_EXTENDEDPRICE
       FROM LINEITEM
       ORDER BY L_EXTENDEDPRICE DESC )
WHERE ROWNUM <= 20;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |   20 |   780 |       | 59138   (1)| 00:00:03 |
|*  1 |  COUNT STOPKEY           |          |      |       |       |            |          |
|   2 |   VIEW                   |          | 1800K|    66M|       | 59138   (1)| 00:00:03 |
|*  3 |    SORT ORDER BY STOPKEY |          | 1800K|   214M|   287M| 59138   (1)| 00:00:03 |
|   4 |     TABLE ACCESS FULL    | LINEITEM | 1800K|   214M|       | 8788    (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------
   1 - filter(ROWNUM<=20)
   3 - filter(ROWNUM<=20)
```

How to speed up sorting in top n queries ?

RANK function can be used improve performance of top n class of queries

Ranking and retrieving the first 20 ranks

```
SELECT *
FROM ( SELECT L_ORDERKEY, L_LINENUMBER, L_EXTENDEDPRICE,
              RANK() OVER (ORDER BY L_EXTENDEDPRICE DESC) RANKING
       FROM LINEITEM )
WHERE RANKING <= 20;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          | 1800K|    89M|       | 18000   (1)| 00:00:01 |
|*  1 |  VIEW                    |          | 1800K|    89M|       | 18000   (1)| 00:00:01 |
|*  2 |   WINDOW SORT PUSHED RANK |          | 1800K|    25M|    48M| 18000   (1)| 00:00:01 |
|   3 |    TABLE ACCESS FULL     | LINEITEM| 1800K|    25M|       |  8778   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
1 - filter("RANKING"<=20)
2 - filter(RANK() OVER ( ORDER BY INTERNAL_FUNCTION("L_EXTENDEDPRICE") DESC )<=20))
```

How to speed up sorting in `top n` queries ?

`DENSE_RANK` function can be used improve performance of `top n` class of queries

Dense ranking and retrieving the first 20 ranks

```
SELECT *
FROM (SELECT L_ORDERKEY, L_LINENUMBER, L_EXTENDEDPRICE,
      DENSE_RANK() OVER (ORDER BY L_EXTENDEDPRICE DESC) RANKING
      FROM LINEITEM )
WHERE RANKING <= 20;
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |          | 1800K|    89M|       | 18000   (1)| 00:00:01 |
|*  1 |  VIEW                      |          | 1800K|    89M|       | 18000   (1)| 00:00:01 |
|*  2 |   WINDOW SORT PUSHED RANK  |          | 1800K|    25M|    48M| 18000   (1)| 00:00:01 |
|   3 |    TABLE ACCESS FULL       | LINEITEM | 1800K|    25M|       |  8778   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------
   1 - filter("RANKING"<=20)
   2 - filter(DENSE_RANK() OVER ( ORDER BY INTERNAL_FUNCTION("L_EXTENDEDPRICE") DESC
)<=20)
```

How to speed up sorting in "top n" queries ?

`DENSE_RANK` function can be used improve performance of `top n` class of queries

Dense ranking and retrieving the first 20 ranks

```
SELECT *
FROM (SELECT L_ORDERKEY, L_LINENUMBER, L_EXTENDEDPRICE,
             DENSE_RANK() OVER (ORDER BY L_EXTENDEDPRICE DESC) RANKING
      FROM LINEITEM )
      WHERE RANKING <= 20;
```

```
L_ORDERKEY L_LINENUMBER L_EXTENDEDPRICE     RANKING
---------- ------------ --------------- ----------
    313958            2         97900.5          1
   1598819            2         97900.5          1
    925410            6         97900.5          1
    403298            3         97850.5          2
   1593924            1         97850.5          2
    762627            4         97800.5          3
   1320706            1         97800.5          3
   1134944            1         97800.5          3
   1116000            2         97800.5          3
   1082278            1         97750.5          4
    245186            1         97700.5          5
...       ...       ...       ...       ...
```

SQLTuning (1)

Outline

What are the other ways to speed up sorting ?

(1) A value of system initialization parameter `SORT_AREA_SIZE` has impact on sorting

(2) Whenever it is possible use in-memory sorts

(3) Sort fewer rows

(4) Sort fewer columns

(5) Whenever it is possible use parallel sort

(6) Use large temporary tablespace for disk sorts

(7) Do not sort when `FIRST_ROWS` query optimizer parameter is used

SQLTuning (1)

Outline

How to speed up `min/max` queries ?

`MIN/MAX` functions must use index whenever it is available

Creating index

```
CREATE INDEX IDX ON LINEITEM(L_EXTENDEDPRICE);
SELECT MAX(L_EXTENDEDPRICE)
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------
| Id  | Operation                  | Name| Rows| Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |     |    1|     6 | 3       (0)| 00:00:01|
|   1 |  SORT AGGREGATE            |     |    1|     6 |            |         |
|   2 |   INDEX FULL SCAN (MIN/MAX)| IDX |    6|     3 |(0)         | 00:00:01|
-------------------------------------------------------------------------------
```

It is possible to get `min/max` values from a data dictionary

Retrieving max/min values from data dictionary

```
SELECT UTL_RAW.CAST_TO_NUMBER(LOW_VALUE), UTL_RAW.CAST_TO_NUMBER(HIGH_VALUE)
FROM USER_TAB_COL_STATISTICS
WHERE TABLE_NAME = 'LINEITEM' AND
      COLUMN_NAME = 'L_EXTENDEDPRICE';
```

```
UTL_RAW.CAST_TO_NUMBER(LOW_VALUE) UTL_RAW.CAST_TO_NUMBER(HIGH_VALUE)
--------------------------------- ---------------------------------
                              902                           97900.5
```

SQLTuning (1)

Outline

How to speed up grouping?

Queries with `GROUP BY` clause must use index whenever it is available

Creating an index

```
CREATE INDEX IDX ON LINEITEM(L_QUANTITY);
```

SELECT statement with GROUP BY clause

```
SELECT L_QUANTITY, COUNT(*)
FROM LINEITEM
GROUP BY L_QUANTITY;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation             | Name| Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |     |    50 |   150 | 1001    (5)| 00:00:01 |
|   1 |  HASH GROUP BY        |     |    50 |   150 | 1001    (5)| 00:00:01 |
|   2 |   INDEX FAST FULL SCAN| IDX | 1800K|  5273K|  958    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

SQLTuning (1)

Outline

How to speed up set algebra queries ?

Implementation of `UNION` operation requires sorting to eliminate the duplicates

SELECT statements with UNION operation

```
SELECT L_TAX
FROM LINEITEM
 UNION
SELECT L_DISCOUNT
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes|TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          | 3600K|   10M|       | 27169  (51)| 00:00:02 |
|   1 |  SORT UNIQUE        |          | 3600K|   10M|    41M| 27169  (51)| 00:00:02 |
|   2 |  UNION-ALL          |          |      |      |       |            |          |
|   3 |   TABLE ACCESS FULL| LINEITEM | 1800K| 5273K|       | 8780    (1)| 00:00:01 |
|   4 |   TABLE ACCESS FULL| LINEITEM | 1800K| 5273K|       | 8779    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

How to speed up set algebra queries ?

Indexing can speed up access to the arguments of UNION operation

Creating an index

```
CREATE INDEX IDX1 ON LINEITEM(L_TAX);
```

Creating an index

```
CREATE INDEX IDX2 ON LINEITEM(L_DISCOUNT);
```

SELECT statements with UNION operation

```
SELECT L_TAX
FROM LINEITEM
 UNION
SELECT L_DISCOUNT
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation              | Name | Rows | Bytes|TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |      | 3600K|   10M|       | 11513 (51)| 00:00:01 |
|   1 |  SORT UNIQUE           |      | 3600K|   10M|    41M| 11513 (51)| 00:00:01 |
|   2 |   UNION-ALL            |      |      |      |       |           |          |
|   3 |    INDEX FAST FULL SCAN| IDX1 | 1800K| 5273K|       |   951  (1)| 00:00:01 |
|   4 |    INDEX FAST FULL SCAN| IDX2 | 1800K| 5273K|       |   952  (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

How to speed up set algebra queries ?

Whenever it is possible we must use `UNION ALL` operation instead of `UNION` operation

SELECT statement with UNION-ALL operation

```
SELECT L_TAX
FROM LINEITEM
 UNION ALL
SELECT L_DISCOUNT
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          | 3600K|    10M| 17559 (51)| 00:00:01 |
|   1 |  UNION-ALL          |          |      |       |           |          |
|   2 |   TABLE ACCESS FULL| LINEITEM | 1800K| 5273K| 8780    (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| LINEITEM | 1800K| 5273K| 8779    (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

How to speed up set algebra queries ?

Implementation of `INTERSECT` operation requires sorting

SELECT statements with INTERSECT operation

```
SELECT L_TAX
FROM LINEITEM
 INTERSECT
SELECT L_DISCOUNT
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes|TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          | 1800K|   10M|       | 27169  (51)| 00:00:02 |
|   1 |  INTERSECTION       |          |      |      |       |            |          |
|   2 |   SORT UNIQUE       |          | 1800K| 5273K|    20M| 13585   (1)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| LINEITEM| 1800K| 5273K|       | 8780    (1)| 00:00:01 |
|   4 |   SORT UNIQUE       |          | 1800K| 5273K|    20M| 13584   (1)| 00:00:01 |
|   5 |    TABLE ACCESS FULL| LINEITEM| 1800K| 5273K|       | 8779    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

How to speed up set algebra queries ?

`INTERSECT` operation should be implemented as semi join operation

INTERSECT implemented as semijoin

```
SELECT L_TAX
FROM LINEITEM
WHERE L_TAX IN (SELECT L_DISCOUNT
                FROM LINEITEM);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          | 1800K|    10M|       | 20510   (1)| 00:00:01 |
|*  1 |  HASH JOIN SEMI    |          | 1800K|    10M|    25M| 20510   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| LINEITEM | 1800K|  5273K|       |  8783   (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| LINEITEM | 1800K|  5273K|       |  8779   (1)| 00:00:01 |
-------------------------------------------------------------------------------------
   1 - access("L_TAX"="L_DISCOUNT")
   2 - filter("L_TAX"<=1.00)
```

How to speed up set algebra queries ?

Implementation of `MINUS` operation requires sorting

SELECT statements with MINUS operation

```
SELECT L_TAX
FROM LINEITEM
 MINUS
SELECT L_DISCOUNT
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows | Byte |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          | 1800K|   10M|       | 27169  (51)| 00:00:02 |
|   1 |  MINUS              |          |      |      |       |            |          |
|   2 |   SORT UNIQUE       |          | 1800K| 5273K|    20M| 13585   (1)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| LINEITEM | 1800K| 5273K|       | 8780    (1)| 00:00:01 |
|   4 |   SORT UNIQUE       |          | 1800K| 5273K|    20M| 13584   (1)| 00:00:01 |
|   5 |    TABLE ACCESS FULL| LINEITEM | 1800K| 5273K|       | 8779    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

How to speed up set algebra queries ?

`MINUS` operation should be implemented as antijoin

Antijoin implementation of MINUS operation

```
SELECT L_TAX
FROM LINEITEM
WHERE L_TAX NOT IN (SELECT L_DISCOUNT
                    FROM LINEITEM);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          | 99676|   584K|       | 20507   (1)| 00:00:01 |
|*  1 |  HASH JOIN ANTI    |          | 99676|   584K|    25M| 20507   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| LINEITEM| 1800K|  5273K|       | 8780    (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| LINEITEM| 1800K|  5273K|       | 8779    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

How to speed up set algebra queries ?

How to simplify complex set algebra expressions

Complex set algebra expression

```
SELECT L_TAX
FROM LINEITEM
MINUS (SELECT L_TAX
       FROM LINEITEM
       MINUS
       SELECT L_DISCOUNT
       FROM LINEITEM);
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes|TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          | 1800K|   15M|       | 40755  (67)| 00:00:02|
|   1 |  MINUS                 |          |      |      |       |            |         |
|   2 |   SORT UNIQUE          |          | 1800K| 5273K|    20M| 13585   (1)| 00:00:01|
|   3 |    TABLE ACCESS FULL   | LINEITEM | 1800K| 5273K|       |  8780   (1)| 00:00:01|
|   4 |     MINUS              |          |      |      |       |            |         |
|   5 |      SORT UNIQUE       |          | 1800K| 5273K|    20M| 13585   (1)| 00:00:01|
|   6 |       TABLE ACCESS FULL| LINEITEM | 1800K| 5273K|       | 8780    (1)| 00:00:01|
|   7 |      SORT UNIQUE       |          | 1800K| 5273K|    20M| 13584   (1)| 00:00:01|
|   8 |       TABLE ACCESS FULL| LINEITEM | 1800K| 5273K|       | 8779    (1)| 00:00:01|
-------------------------------------------------------------------------------------------
```

How to speed up set algebra queries ?

```
T MINUS ( T MINUS D ) = T INTERSECT D
```

SELECT statements with INTERSECT operation

```
SELECT L_TAX
FROM LINEITEM
 INTERSECT
SELECT L_DISCOUNT
FROM LINEITEM;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows | Bytes|TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          | 1800K|   10M|       | 27169  (51)| 00:00:02|
|   1 |  INTERSECTION       |          |      |      |       |            |         |
|   2 |   SORT UNIQUE       |          | 1800K| 5273K|    20M| 13585   (1)| 00:00:01|
|   3 |    TABLE ACCESS FULL| LINEITEM | 1800K| 5273K|       |  8780   (1)| 00:00:01|
|   4 |   SORT UNIQUE       |          | 1800K| 5273K|    20M| 13584   (1)| 00:00:01|
|   5 |    TABLE ACCESS FULL| LINEITEM | 1800K| 5273K|       |  8779   (1)| 00:00:01|
-------------------------------------------------------------------------------------
```

References

T. Hasler, Expert Oracle SQL Optimization, Deployment, and Statistics , Apress, 2014

S. R. Alapati, D. Kuhn, B. Padfield, Oracle Database 12c Performance Tuning Recipes A Problem-Solution Approach, Apress, 2013

G. Harrison, Oracle Performance Survival Guide, A Systematic Approach to Database Optimization, Prentice Hall Professional Oracle Series, 2010