CSCI317 Database Performance Tuning

SQL Tuning (2)

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

SQL Tuning (2)

Outline

Index (nested loop) joins

Typical index based join uses an index on a primary key

Index based join

```
SELECT /*+ LEADING(ORDERS)
           USE_NL_WITH_INDEX(CUSTOMER CUSTOMER_PKEY) */ *
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows| Bytes | Cost (%CPU)| Time   |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               | 450K|   115M| 452K (1)| 00:00:18 |
|   1 |  NESTED LOOPS                 |               |     |       |         |          |
|   2 |   NESTED LOOPS                |               | 450K|   115M| 452K (1)| 00:00:18 |
|*  3 |    TABLE ACCESS FULL          | ORDERS        | 450K|    46M| 1950 (1)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN          | CUSTOMER_PKEY |   1 |       |    0 (0)| 00:00:01 |
|   5 |     TABLE ACCESS BY INDEX ROWID| CUSTOMER      |   1 |   159 |    1 (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
   3 - filter("ORDERS"."O_CUSTKEY">=0)
   4 - access("ORDERS"."O_CUSTKEY"="CUSTOMER"."C_CUSTKEY")
```

Index (nested loop) joins

If attributes from one table are only needed and index on a foreign key speeds up index based join

Creating and index on a foreign key

```
CREATE INDEX IDX ON ORDERS(O_CUSTKEY);
SELECT /*+ LEADING(ORDERS)
           USE_NL_WITH_INDEX(CUSTOMER CUSTOMER_PKEY) */ C_CUSTKEY, C_NAME
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY;
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name         | Rows| Bytes| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |              | 450K|   12M| 450K    (1)| 00:00:18 |
|   1 |  NESTED LOOPS                  |              |     |      |            |          |
|   2 |   NESTED LOOPS                 |              | 450K|   12M| 450K    (1)| 00:00:18 |
|*  3 |    INDEX FAST FULL SCAN        |          IDX | 450K| 2197K| 272     (1)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN           | CUSTOMER_PKEY|   1 |      |   0     (0)| 00:00:01 |
|   5 |     TABLE ACCESS BY INDEX ROWID| CUSTOMER     |   1 |   24 |   1     (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
   3 - filter("ORDERS"."O_CUSTKEY">=0)
   4 - access("ORDERS"."O_CUSTKEY"="CUSTOMER"."C_CUSTKEY")
```

Index (nested loop) joins

If join is performed over `PK-FK` and all attributes selected are from `FK` side then there is no need for join

SELECT statement with redundant JOIN operation

```
SELECT /*+ LEADING(ORDERS)
           USE_NL_WITH_INDEX(CUSTOMER CUSTOMER_PKEY) */
       ORDERS.O_ORDERKEY, ORDERS.O_ORDERSTATUS
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows| Bytes| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          | 450K| 3515K| 1948    (1)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| ORDERS   | 450K| 3515K| 1948    (1)| 00:00:01 |
-------------------------------------------------------------------------------
```

SQL Tuning (2)

Outline

Hash joins

Hash join is the most effective implementation of join operation when two large relational tables are joined over an equality conditions

SELECT statement with JOIN operation

```
SELECT ORDERS.O_ORDERKEY, ORDERS.O_ORDERSTATUS, CUSTOMER.C_NAME
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |  450K|    15M|       | 2843    (1)| 00:00:01 |
|*  1 |  HASH JOIN         |          |  450K|    15M|  1584K| 2843    (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| CUSTOMER| 45000|  1054K|       | 282     (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| ORDERS   |  450K|  5712K|       | 1949    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
   1 - access("ORDERS"."O_CUSTKEY"="CUSTOMER"."C_CUSTKEY")
   3 - filter("ORDERS"."O_CUSTKEY">=0)
```

Hash joins

Changing an order of arguments in hash join may have an impact on performance

SELECT statement with LEADING hint and JOIN operation

```
SELECT /*+ LEADING(ORDERS) */
       ORDERS.O_ORDERKEY, ORDERS.O_ORDERSTATUS, CUSTOMER.C_NAME
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |  450K|    15M|       | 2844    (1)| 00:00:01 |
|*  1 |  HASH JOIN         |          |  450K|    15M|    10M| 2844    (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| ORDERS   |  450K|  5712K|       | 1949    (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| CUSTOMER | 45000|  1054K|       |  282    (0)| 00:00:01 |
-------------------------------------------------------------------------------------
   1 - access("ORDERS"."O_CUSTKEY"="CUSTOMER"."C_CUSTKEY")
   2 - filter("ORDERS"."O_CUSTKEY">=0)
```

Hash joins

Indexing can be used to speed up hash join when only the attributes from `PK` side are selected

Creating an index

```
CREATE INDEX IDX ON ORDERS(O_CUSTKEY);
```

SELECT statement with JOIN operation

```
SELECT CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |  450K|    11M|       |  970    (1)| 00:00:01 |
|*  1 |  HASH JOIN            |          |  450K|    11M|  1496K|  970    (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL   | CUSTOMER |45000 |   966K|       |  282    (0)| 00:00:01 |
|*  3 |   INDEX FAST FULL SCAN|      IDX |  450K|  1757K|       |  272    (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
   1 - access("ORDERS"."O_CUSTKEY"="CUSTOMER"."C_CUSTKEY")
   3 - filter("ORDERS"."O_CUSTKEY">=0)
```

Hash joins

Hash join cannot be used when the relational tables are joined over non equality join conditions

SELECT statement with JOIN operation

```
SELECT /*+ LEADING(ORDERS) USE_HASH(CUSTOMER) */ *
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_TOTALPRICE > CUSTOMER.C_ACCTBAL;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |   20G|  5015G| 126M    (1)| 01:22:09 |
|   1 |  NESTED LOOPS      |          |   20G|  5015G| 126M    (1)| 01:22:09 |
|   2 |   TABLE ACCESS FULL| ORDERS   |  450K|    46M| 1950    (1)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| CUSTOMER| 44658|  6934K|  280    (0)| 00:00:01 |
-------------------------------------------------------------------------------------
   3 - filter("ORDERS"."O_TOTALPRICE">"CUSTOMER"."C_ACCTBAL")
```

SQL Tuning (2)

Outline

Sort-merge joins

Sort-merge join is used when the relational tables are joined over non equality join conditions

SELECT statement with JOIN operation

```
SELECT *
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_TOTALPRICE > CUSTOMER.C_ACCTBAL;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |   20G|  5015G|       | 66173  (78)| 00:00:03 |
|   1 |  MERGE JOIN         |          |   20G|  5015G|       | 66173  (78)| 00:00:03 |
|   2 |   SORT JOIN         |          | 45000|  6987K|    15M|  1858   (1)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| CUSTOMER | 45000|  6987K|       |   283   (1)| 00:00:01 |
|*  4 |   SORT JOIN         |          |  450K|    46M|   115M| 13009   (1)| 00:00:01 |
|   5 |    TABLE ACCESS FULL|   ORDERS |  450K|    46M|       |  1950   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
   4 - access("ORDERS"."O_TOTALPRICE">"CUSTOMER"."C_ACCTBAL")
       filter("ORDERS"."O_TOTALPRICE">"CUSTOMER"."C_ACCTBAL")
```

Sort-merge joins

Sort-merge join can be enforced through a hint

SELECT statement with JOIN operation and a hint

```
SELECT /*+ USE_MERGE(ORDERS CUSTOMER) */
       ORDERS.O_ORDERKEY, ORDERS.O_ORDERSTATUS, CUSTOMER.C_NAME
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes|TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |  450K| 15M  |       | 4685    (1)| 00:00:01 |
|   1 |  MERGE JOIN         |          |  450K| 15M  |       | 4685    (1)| 00:00:01 |
|   2 |   SORT JOIN         |          | 45000| 1054K|  2840K|  599    (1)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| CUSTOMER| 45000| 1054K|       |  282    (0)| 00:00:01 |
|*  4 |   SORT JOIN         |          |  450K| 5712K|    20M| 4086    (1)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL| ORDERS   |  450K| 5712K|       | 1949    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
   4 - access("ORDERS"."O_CUSTKEY"="CUSTOMER"."C_CUSTKEY")
filter("ORDERS"."O_CUSTKEY"="CUSTOMER"."C_CUSTKEY")
   5 - filter("ORDERS"."O_CUSTKEY">=0)
```

Sort-merge joins

Sort-merge join can be improved through indexing

Creating an index

```
CREATE INDEX IDX1 ON ORDERS(O_TOTALPRICE);
```

Creating an index

```
CREATE INDEX IDX2 ON CUSTOMER(C_ACCTBAL);
```

SELECT statement with JOIN operation and a hint

```
SELECT /*+ INDEX_ASC(ORDERS IDX1) INDEX_ASC(CUSTOMER IDX2) */ *
FROM ORDERS JOIN CUSTOMER
            ON ORDERS.O_TOTALPRICE > CUSTOMER.C_ACCTBAL;
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name     | Rows | Bytes|TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |          |   20G| 5015G|       | 549K   (10)| 00:00:22 |
|   1 |  MERGE JOIN                           |          |   20G| 5015G|       | 549K   (10)| 00:00:22 |
|   2 |   SORT JOIN                           |          |  450K|   46M|       | 451K    (1)| 00:00:18 |
|   3 |    TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS   |  450K|   46M|       | 451K    (1)| 00:00:18 |
|   4 |     INDEX FULL SCAN                   |   IDX1   |  450K|      |       | 1069    (1)| 00:00:01 |
|*  5 |   SORT JOIN                           |          | 45000| 6987K|    15M| 46654   (1)| 00:00:02 |
|   6 |    TABLE ACCESS BY INDEX ROWID BATCHED| CUSTOMER | 45000| 6987K|       | 45079   (1)| 00:00:02 |
|   7 |     INDEX FULL SCAN                   |    IDX2  | 45000|      |       | 102     (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------
   5 - access(INTERNAL_FUNCTION("ORDERS"."O_TOTALPRICE")>INTERNAL_FUNCTION("CUSTOMER"."C_ACCTBAL"))
       filter(INTERNAL_FUNCTION("ORDERS"."O_TOTALPRICE")>INTERNAL_FUNCTION("CUSTOMER"."C_ACCTBAL"))
```

SQL Tuning (2)

Outline

Order of join arguments in 2-way join

When using hash join a smaller argument must be first

SELECT statement with JOIN operation and a hint

```
SELECT /*+ LEADING(LINEITEM) */ *
FROM REGION JOIN LINEITEM
            ON REGION.R_COMMENT = LINEITEM.L_COMMENT;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |    9 |  1998 |       | 20462   (1)| 00:00:01 |
|*  1 |  HASH JOIN         |          |    9 |  1998 |   235M| 20462   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| LINEITEM | 1800K|   214M|       | 8788    (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| REGION   |    5 |   485 |       |    9    (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Order of join arguments in 2-way join

When using hash join a smaller argument must be first

SELECT statement with JOIN operation and a hint

```
SELECT /*+ LEADING(REGION) */ *
FROM REGION JOIN LINEITEM
            ON REGION.R_COMMENT = LINEITEM.L_COMMENT;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |    9 |  1998 | 8801    (1)| 00:00:01 |
|*  1 |  HASH JOIN         |         |    9 |  1998 | 8801    (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| REGION  |    5 |   485 |    9    (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| LINEITEM| 1800K|   214M| 8788    (1)| 00:00:01 |
-------------------------------------------------------------------------------
   1 - access("REGION"."R_COMMENT"="LINEITEM"."L_COMMENT")
```

SQL Tuning (2)

Outline

Order of join arguments in m-way join

When joining 3 or more tables join smaller tables first

SELECT statement with JOIN operation and a hint

```
SELECT /*+ LEADING(LINEITEM) */ *
FROM LINEITEM JOIN ORDERS
              ON LINEITEM.L_COMMENT = ORDERS.O_COMMENT
              JOIN REGION
              ON ORDERS.O_COMMENT = REGION.R_COMMENT;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         |    9 |  2979 |       | 24989   (1)| 00:00:01 |
|*  1 | HASH JOIN           |         |    9 |  2979 |       | 24989   (1)| 00:00:01 |
|   2 |  TABLE ACCESS FULL  |  REGION |    5 |   485 |       |     9   (0)| 00:00:01 |
|*  3 |   HASH JOIN         |         |  787K|   175M|   235M| 24978   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| LINEITEM| 1800K|   214M|       |  8788   (1)| 00:00:01 |
|   5 |    TABLE ACCESS FULL|  ORDERS |  450K|    46M|       |  1950   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
   1 - access("ORDERS"."O_COMMENT"="REGION"."R_COMMENT")
   3 - access("LINEITEM"."L_COMMENT"="ORDERS"."O_COMMENT")
```

Order of join arguments in m-way join

When joining 3 or more tables join smaller tables first

SELECT statement with JOIN operation and a hint

```
SELECT /*+ LEADING(ORDERS) */ *
FROM LINEITEM JOIN ORDERS
              ON LINEITEM.L_COMMENT = ORDERS.O_COMMENT
              JOIN REGION
              ON ORDERS.O_COMMENT = REGION.R_COMMENT;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         |    9 | 2979 |        | 13327   (1)| 00:00:01 |
|*  1 |  HASH JOIN          |         |    9 | 2979 |        | 13327   (1)| 00:00:01 |
|*  2 |   HASH JOIN         |         |    5 | 1030 |     51M| 4535    (1)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| ORDERS  |  450K|  46M |        | 1950    (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| REGION  |    5 |  485 |        |    9    (0)| 00:00:01 |
|   5 |   TABLE ACCESS FULL | LINEITE | 1800K| 214M |        | 8788    (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
   1 - access("LINEITEM"."L_COMMENT"="ORDERS"."O_COMMENT")
   2 - access("ORDERS"."O_COMMENT"="REGION"."R_COMMENT")
```

Order of join arguments in m-way join

When joining 3 or more tables join smaller tables first

SELECT statement with JOIN operation and a hint

```
SELECT /*+ LEADING(REGION) */ *
FROM LINEITEM JOIN ORDERS
              ON LINEITEM.L_COMMENT = ORDERS.O_COMMENT
              JOIN REGION
              ON ORDERS.O_COMMENT = REGION.R_COMMENT;
```

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows | Bytes| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |    9 | 2979 | 10752 (1)| 00:00:01 |
|*  1 |  HASH JOIN          |          |    9 | 2979 | 10752 (1)| 00:00:01 |
|*  2 |   HASH JOIN         |          |    5 | 1030 | 1960  (1)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| REGION   |    5 |  485 |    9  (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| ORDERS   |  450K|  46M | 1950  (1)| 00:00:01 |
|   5 |   TABLE ACCESS FULL | LINEITEM| 1800K| 214M | 8788  (1)| 00:00:01 |
-------------------------------------------------------------------------------
   1 - access("LINEITEM"."L_COMMENT"="ORDERS"."O_COMMENT")
   2 - access("ORDERS"."O_COMMENT"="REGION"."R_COMMENT")
```

SQL Tuning (2)

Outline

Comparison of join algorithms

Sort-merge/hash versus index-based join

- Index-based join provides a better response time
- Sort-merge/hash join provides better throughput
- Sort-merge/hash join is faster for large tables
- Sort-merge/hash join needs more cpu and memory
- Index-based join needs an index
- Hash join is superior to sort-merge join
- Index-based join is superior to sort-merge/hash join when a small number of rows is returned

SQL Tuning (2)

Outline

Optimization of hash join through parameters

`HASH_AREA_SIZE` system initialization parameter determines the amount of memory available to hash join forn the creation and storage of hash table

`HASH_MULTIBLOCK_IO_COUNT` system initialization parameter determines the total number of blocks that will be written to, or read from hash join partition in a single I/O operation

SQL Tuning (2)

Outline

Outer joins

Performance of outer join is comparable to performance of equijoin

Outer join imposes a particular order of the arguments: a table which returns all rows must be used in outer loop

SQL Tuning (2)

Outline

Nested queries

A subquery that returns only one value is computed first

Nested SELECT statement with a subquery that returns one value

```
SELECT COUNT(*)
FROM ORDERS
WHERE ORDERS.O_TOTALPRICE = ( SELECT MIN(C_ACCTBAL)
                              FROM CUSTOMER );
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |    1 |     6 | 2232    (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |          |    1 |     6 |            |          |
|*  2 |   TABLE ACCESS FULL| ORDERS   |    1 |     6 | 1949    (1)| 00:00:01 |
|   3 |  SORT AGGREGATE    |          |    1 |     6 |            |          |
|   4 |   TABLE ACCESS FULL| CUSTOMER | 45000|   263K| 282     (0)| 00:00:01 |
-------------------------------------------------------------------------------------

   2 - filter("ORDERS"."O_TOTALPRICE"= (SELECT MIN("C_ACCTBAL") FROM
              "CUSTOMER" "CUSTOMER"))
```

Nested queries

If a subquery returns more than one value then a semijoin is used

Nested SELECT statement with a subquery that returns more than one row

```
SELECT COUNT(*)
FROM ORDERS
WHERE ORDERS.O_TOTALPRICE IN ( SELECT C_ACCTBAL
                               FROM CUSTOMER );
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |     1 |    12 | 2233    (1)| 00:00:01 |
|   1 |  SORT AGGREGATE          |          |     1 |    12 |            |          |
|*  2 |   HASH JOIN RIGHT SEMI   |          | 40406 |   473K| 2233    (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL     | CUSTOMER | 40910 |   239K| 283     (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL     | ORDERS   |  450K |  2636K| 1949    (1)| 00:00:01 |
-------------------------------------------------------------------------------------------

   2 - access("ORDERS"."O_TOTALPRICE"="C_ACCTBAL")
   3 - filter("C_ACCTBAL">=0)
```

Nested queries

If a subquery returns more than one value then a antijoin is used

Nested SELECT statement with NOT IN relation and a subquery that returns more than one row

```
SELECT COUNT(*)
FROM ORDERS
WHERE ORDERS.O_TOTALPRICE NOT IN ( SELECT C_ACCTBAL
                                   FROM CUSTOMER );
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |     1 |    12 |  3087   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE       |          |     1 |    12 |            |          |
|*  2 |   HASH JOIN RIGHT ANTI|          |   441K|  5177K|  3087   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL  | CUSTOMER | 40910 |   239K|   390   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL  | ORDERS   |   450K|  2636K|  2696   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------

   2 - access("ORDERS"."O_TOTALPRICE"="C_ACCTBAL")
   3 - filter("C_ACCTBAL">=0)
```

Nested queries

If an outer and inner query operates on the same table then it is possible to optimize entire query with a single index

CREATE INDEX statement

```
CREATE INDEX IDX ON ORDERS(O_TOTALPRICE);
```

Nested SELECT statement with a subquery that returns one row

```
SELECT COUNT(*)
FROM ORDERS
WHERE ORDERS.O_TOTALPRICE = ( SELECT MIN(ORDERS.O_TOTALPRICE)
                              FROM ORDERS );
```

```
-----------------------------------------------------------------------------------------
| 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 RANGE SCAN           |  IDX |    1 |     6 |    3    (0)| 00:00:01 |
|   3 |  SORT AGGREGATE              |      |    1 |     6 |            |          |
|   4 |   INDEX FULL SCAN (MIN/MAX)|  IDX |    1 |     6 |    3    (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

   2 - access("ORDERS"."O_TOTALPRICE"= (SELECT MIN("ORDERS"."O_TOTALPRICE")
               FROM "ORDERS" "ORDERS")
   3 - filter("C_ACCTBAL">=0)
```

Nested queries

Nested query with `EXISTS` quantifier is implemented with a semijoin operation

SELECT statement with EXISTS quantifier

```
SELECT *
FROM CUSTOMER
WHERE EXISTS ( SELECT *
               FROM ORDERS
               WHERE CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY );
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes| TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          | 30162 | 4830K|        | 2960   (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT SEMI |          | 30162 | 4830K|   7472K| 2960   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL   | ORDERS   |   450K| 2197K|        | 1949   (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL   | CUSTOMER| 45000 | 6987K|        |  283   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
   1 - access("CUSTOMER"."C_CUSTKEY"="ORDERS"."O_CUSTKEY")
   2 - filter("ORDERS"."O_CUSTKEY">=0)
```

Nested queries

Performance of a nested query with `EXISTS` quantifier implemented with semijoin can be improved with an index

CREATE INDEX statement

```
CREATE INDEX IDX ON ORDERS(O_CUSTKEY);
```

SELECT statement with EXISTS quantifier

```
SELECT *
FROM CUSTOMER
WHERE EXISTS ( SELECT *
               FROM ORDERS
               WHERE CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY );
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          | 30162 |  4830K|       | 1283    (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT SEMI |          | 30162 |  4830K|  7472K| 1283    (1)| 00:00:01 |
|*  2 |   INDEX FAST FULL SCAN| IDX      |  450K |  2197K|       |  272    (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL   | CUSTOMER | 45000 |  6987K|       |  283    (1)| 00:00:01 |
-----------------------------------------------------------------------------------------
   1 - access("CUSTOMER"."C_CUSTKEY"="ORDERS"."O_CUSTKEY")
   2 - filter("ORDERS"."O_CUSTKEY">=0)
```

Nested queries

Nested query with `NOT EXISTS` negated quantifier is implemented with an antijoin operation

SELECT statement with NOT EXISTS negated quantifier

```
SELECT *
FROM CUSTOMER
WHERE NOT EXISTS ( SELECT *
                   FROM ORDERS
                   WHERE CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY );
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          | 14838 | 2376K|       | 2960    (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT ANTI  |          | 14838 | 2376K|  7472K| 2960    (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL    | ORDERS   |   450K| 2197K|       | 1949    (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL    | CUSTOMER | 45000 | 6987K|       | 283     (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
   1 - access("CUSTOMER"."C_CUSTKEY"="ORDERS"."O_CUSTKEY")
   2 - filter("ORDERS"."O_CUSTKEY">=0)
```

Nested queries

Performance of a nested query with `NOT EXISTS` negated quantifier implemented with a antijoin can be improved with an index

CREATE INDEX statement

```
CREATE INDEX IDX ON ORDERS(O_CUSTKEY);
```

SELECT statement with NOT EXISTS negated quantifier

```
SELECT *
FROM CUSTOMER
WHERE NOT EXISTS ( SELECT *
                   FROM ORDERS
                   WHERE CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY );
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes| TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          | 14838 | 2376K|        | 1283    (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT ANTI  |          | 14838 | 2376K|   7472K| 1283    (1)| 00:00:01 |
|*  2 |   INDEX FAST FULL SCAN |      IDX |   450K| 2197K|        |  272    (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL    | CUSTOMER | 45000 | 6987K|        |  283    (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
   1 - access("CUSTOMER"."C_CUSTKEY"="ORDERS"."O_CUSTKEY")
   2 - filter("ORDERS"."O_CUSTKEY">=0))
```

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