Oracle Diagnostics

Hemant K Chitale

Hemant K Chitale

- whoami ?
- Oracle 5 to Oracle 10gR2 ： DOS, Xenix,8 flavours of Unix, Linux, Windows
- Financial Services, Govt/Not-for-Profit, ERP, Custom
- Production Support, Consulting, Development
- A DBA, not a Developer
- *My Oracle Blog* *http://hemantoracledba.blogspot.com*

Oracle Diagnostics – Joins (1)

- For “Explain Plans” see my earlier presentation OracleDiagnostics_Explain_Plans__simple (or at WizIQ)

- This presentation is based on the case study Nested Loops and Consistent Gets

A Nested Loop join

- In a Nested Loop join, Oracle uses the rowset retrieved from the “outer” (driving) table to query the “inner” (driven) table
- The Optimizer would choose a Nested Loop join if it expects to be able to retrieve the rowset of the outer table very quickly *and* expects that the size would be small enough to not require querying the inner table too frequently

Given this query :

```
SQL> explain plan for
  2  select  product_desc, txn_id, transaction_amt
  3  from transactions t, product_table p
  4  where txn_date between to_date('01-FEB-2011','DD-MON-YYYY')
and to_date('28-FEB-2011','DD-MON-YYYY')
  5  and txn_id between 155000 and 156000
  6  and country_cd = 'IN'
  7  and t.product_cd=p.product_cd
  8  /
```

We expect a Join between TRANSACTIONS and PRODUCT_TABLE

```
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------
----------------------
Plan hash value: 1299935411

--------------------------------------------------------------------------------------------------------------
-----------
| Id  | Operation                           | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
Pstart| Pstop |
--------------------------------------------------------------------------------------------------------------
-----------
|   0 | SELECT STATEMENT                    |                  |     1 |    63 |    25   (0)| 00:00:01 |
|       |
|   1 |  NESTED LOOPS                       |                  |       |       |            |          |
|       |
|   2 |   NESTED LOOPS                      |                  |     1 |    63 |    25   (0)| 00:00:01 |
|       |
|   3 |    PARTITION RANGE SINGLE           |                  |     1 |    51 |    24   (0)| 00:00:01 |
2 |     2 |
|*  4 |     TABLE ACCESS BY LOCAL INDEX ROWID| TRANSACTIONS     |     1 |    51 |    24   (0)| 00:00:01 |
2 |     2 |
|*  5 |      INDEX RANGE SCAN               | TRANSACTIONS_NDX |   252 |       |     4   (0)| 00:00:01 |
2 |     2 |
|*  6 |    INDEX UNIQUE SCAN                | SYS_C0016611     |     1 |       |     0   (0)| 00:00:01 |
|       |
|   7 |   TABLE ACCESS BY INDEX ROWID       | PRODUCT_TABLE    |     1 |    12 |     1   (0)| 00:00:01 |
|       |
--------------------------------------------------------------------------------------------------------------

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

   4 - filter("TXN_DATE"<=TO_DATE(' 2011-02-28 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   5 - access("COUNTRY_CD"='IN' AND "TXN_ID">=155000 AND "TXN_ID"<=156000)
   6 - access("T"."PRODUCT_CD"="P"."PRODUCT_CD")

SQL>
```

.

Here :

```
|   3 |    PARTITION RANGE SINGLE            |                  |     1 |    51 |    24   (0)| 00:00:01
|     2 |     2 |
|*  4 |     TABLE ACCESS BY LOCAL INDEX ROWID| TRANSACTIONS     |     1 |    51 |    24   (0)| 00:00:01
|     2 |     2 |
|*  5 |      INDEX RANGE SCAN                | TRANSACTIONS_NDX |   252 |       |     4   (0)| 00:00:01
|     2 |     2 |

Predicate Information :
 4 - filter("TXN_DATE"<=TO_DATE(' 2011-02-28 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   5 - access("COUNTRY_CD"='IN' AND "TXN_ID">=155000 AND "TXN_ID"<=156000)
```

The execution is done in this sequence :

Step 1 : Operation 5 : Index Range Scan (expected to retrieve 252 rowids), accessing the `TRANSACTIONS_NDX` index by `COUNTRY_CD = 'IN'` and `TXN_ID between 155000 and 156000`

Step 2 : Operation 4 : Table Access of `TRANSACTIONS`, using the RowIDs fetched from Operation 5. For every row fetched by RowID, filter for `TXN_DATE less than 28-Feb-11`. Expect to return only 1 single row after the filter for `TXN_DATE` (note : This is key to the optimizer's choice of a Nested Loop join)

Parent Step 3 : Operation 3 : Determine that this is a Range Scan within Partition 2 (Start and Stop Partition IDs being "2" and "2"). This range scan will return the single row from Operation 4.

Continuing with the other operations :

```
|   2 |   NESTED LOOPS                      |                 |     1 |    63 |    25   (0)| 00:00:01
|       |       |
|   3 |    PARTITION RANGE SINGLE           |                 |     1 |    51 |    24   (0)| 00:00:01
|     2 |     2 |

|*  6 |    INDEX UNIQUE SCAN                | SYS_C0016611    |     1 |       |     0   (0)| 00:00:01
|       |       |


Predicate Information :

 6 - access("T"."PRODUCT_CD"="P"."PRODUCT_CD")
```

Step 4 Operation 2 : For the expected single row returned from the Partition Range (in Operation 3), call Operation 6 to scan the Unique Index SYS_C0016611. The Index Unique Scan is expected to be executed once only (as Operation 3 returns 1 row) and, in-turn, expected to return a single RowID (being a Unique Index scan)
This Index is the Primary Key index for PRODUCT_TABLE. The Index Scan is for the PRODUCT_CD retrieved the previous step.

```
|   1 |  NESTED LOOPS                       |                 |       |       |            |
|       |       |
|   7 |   TABLE ACCESS BY INDEX ROWID       | PRODUCT_TABLE   |     1 |    12 |     1   (0)| 00:00:01
|       |       |
```

Step 6 Operation 1 : For the single RowID expected from Operation 6, access the PRODUCT_TABLE. (Since the previous step expects to return a single RowID, this step, too, expects to read a single row from the table).

Hash Join

- This is a demonstration of a Hash Join from the same case study
- In a Hash Join, Oracle attempts to retrieve all the join column values from the first table, compute “hash” values and place them in memory
- It then expects to probe the second table and compare the values with the in-memory hash table. A match is a “join”
- Performance of a Hash Join depends on the size of the join key and the available memory for the in-memory Hash Table

For this query :

```
SQL> explain plan for
  2  select  /*+ USE_HASH (t p) */ product_desc, txn_id,
transaction_amt
  3  from transactions t, product_table p
  4  where txn_date between to_date('01-FEB-2011','DD-MON-YYYY')
and to_date('28-FEB-2011','DD-MON-YYYY')
  5  and txn_id between 155000 and 156000
  6  and country_cd = 'IN'
  7  and t.product_cd=p.product_cd
  8  /
```

We have explicitly provided a Hint (directive) to Oracle to execute a Hash Join between the two tables

It is

```
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
Plan hash value: 971735053

------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
Pstart| Pstop |
------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                  |     1 |    63 |    28   (4)| 00:00:01 |
|       |
|*  1 |  HASH JOIN                          |                  |     1 |    63 |    28   (4)| 00:00:01 |
|       |
|   2 |   PARTITION RANGE SINGLE            |                  |     1 |    51 |    24   (0)| 00:00:01 |
2 |     2 |
|*  3 |    TABLE ACCESS BY LOCAL INDEX ROWID| TRANSACTIONS     |     1 |    51 |    24   (0)| 00:00:01 |
2 |     2 |
|*  4 |     INDEX RANGE SCAN                | TRANSACTIONS_NDX |   252 |       |     4   (0)| 00:00:01 |
2 |     2 |
|   5 |   TABLE ACCESS FULL                 | PRODUCT_TABLE    |    14 |   168 |     3   (0)| 00:00:01 |
|       |
------------------------------------------------------------------------------------------------------------------------

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

   1 - access("T"."PRODUCT_CD"="P"."PRODUCT_CD")
   3 - filter("TXN_DATE"<=TO_DATE(' 2011-02-28 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   4 - access("COUNTRY_CD"='IN' AND "TXN_ID">=155000 AND "TXN_ID"<=156000)

SQL>
```

Here :

```
|   2 |   PARTITION RANGE SINGLE            |                  |     1 |    51 |    24   (0)| 00:00:01 |
2 |     2 |
|*  3 |    TABLE ACCESS BY LOCAL INDEX ROWID| TRANSACTIONS     |     1 |    51 |    24   (0)| 00:00:01 |
2 |     2 |
|*  4 |     INDEX RANGE SCAN                | TRANSACTIONS_NDX |   252 |       |     4   (0)| 00:00:01 |
2 |     2 |
Predicate Information :
 3 - filter("TXN_DATE"<=TO_DATE(' 2011-02-28 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   4 - access("COUNTRY_CD"='IN' AND "TXN_ID">=155000 AND "TXN_ID"<=156000)
```

Step 1 : Operation 4 : Retrieve 252 RowIDs from the `TRANSACTIONS_NDX`

Step 2 : Operation 3 : Use the 252 RowIDs to lookup the `TRANSACTIONS` table and filter to `TXN_DATE`. Expect to return a single row

Parent Step 3 : Confirm that these steps are executed against a single Partition

Here :

```
|*  1 |  HASH JOIN                     |                |     1 |    63 |    28   (4)| 00:00:01 |
|       |
|   2 |   PARTITION RANGE SINGLE       |                |     1 |    51 |    24   (0)| 00:00:01 |
2 |     2 |

|   5 |   TABLE ACCESS FULL            | PRODUCT_TABLE  |    14 |   168 |     3   (0)| 00:00:01 |
|       |
Predicate Information :
 1 - access("T"."PRODUCT_CD"="P"."PRODUCT_CD")
```

Step 4 Operation 1: For the single row returned from the Partition Range Scan,execute a Hash Join. The hashed value(s) for this row are stored in an in-memory hash area

Step 5 : Operation 5 : Probe the `PRODUCT_TABLE` (via a FullTableScan) to read 14 rows. Place the values in memory and attempt a join on `PRODUCT_CD`.