

ORIOLE CORPORATION

Copyright © Oriole Corporation, 1998, 1999 <http://www.oriole.com/>

Case Study

Data Warehouse Optimization

HINTS/
OPTIMIZER/
SUGGESTIONS

Author

Larry Miller

Larry Miller is an Europe-based Oracle consultant who specializes in database administration and tuning. The current paper is based on works performed during an assignment with a major Telecoms operator.

TABLE OF CONTENTS

1. CONTEXT----- 3

2. GROUND RULES IN OPTIMIZING SQL TRANSACTIONS----- 4

 2.1. General----- 4

 2.2. Points addressed ----- 4

3. ENVIRONMENT ----- 6

 3.1. Servers and Instances----- 6

 3.2. Tools ----- 6

4. DESCRIPTION OF THE TRANSACTION----- 7

5. PROBLEMS ENCOUNTERED ----- 9

 5.1. A systematic Update against a selective update.----- 9

 5.2. Presence of « hints » ----- 9

 5.2.1. Transaction N°10----- 9

 5.2.2. Transaction N°1----- 10

 5.2.3. Removal of all the « hints » ----- 11

 5.3. Chaining ----- 11

 5.4. Fragmentation ----- 11

 5.5. Rewriting / Modification of the transactions----- 11

 5.5.1. V_client_4 view----- 11

 5.5.2. Transaction Pr_client_13 ----- 12

 5.6. Parallel degree ----- 13

 5.7. READ ONLY----- 13

 5.8. Gain ----- 13

6. SUGGESTIONS, WARNINGS-----14

1. Context

This study was conducted in order to optimize the upload of a large data warehouse in an organization where volumes are evolving very fast (mobile phones - 2,000+ new customers per day at the time of the study!). Initially, and for historical reasons, a middleware product was used to carry out the extraction from the production database and the loading into the datawarehouse. As both the source and the target ended up being Oracle databases running on powerful Unix servers, the potential benefit of being able to process heterogeneous sources of data disappeared while the overhead induced by an additional, and not quite up-to-the task, software layer remained. It was decided to drop the middleware product altogether and replace it with custom PL/SQL procedures.

As a matter of fact, the first release of these procedures proved rather disappointing, which justified the commando operation to optimize them. Quite interestingly, many of the results fly in the face of conventional wisdom and allow us to define a set of rules which can be used in similar large volume contexts.

This case study more specifically references the optimization steps undertaken upon a series of PL/SQL procedures updating client information from a remote database onto the data warehouse.

2. Ground Rules in optimizing SQL transactions

2.1. General

The main elements that have a direct impact on the performance of SQL statements are:

- the system parameters and resources that hold the database
- the instance parameters
- the physical data model
- the indexes
- the wording of the statement
- the optimization mode
- the database organization and its condition - healthy or otherwise

2.2. Points addressed

The points that we will address are the following:

- the indexes

The indexes permit Oracle to ensure, for transactional statements, an equal or constant response time whatever the data volume. We will show that the use of the indexes is to be avoided in the customer subset of our system.

- the wording of the statement

This is the most crucial element. This is also the point where we obtain most performance gains. When the statement is written incorrectly Oracle will choose the incorrect path as shown by the explain plan. This point alone can show factor of performance that varies between one and twenty, if not more.

- the optimization mode

This is another major point confirmed in the current context. Oracle analyzes a statement and determines its execution plan based upon the optimization mode. The plan shows which algorithms and methods the kernel uses to access the data.

Oracle offers two modes of optimization: optimization based upon syntactic rules [RULE] and optimization based upon the costs [COST]. These modes have been explained in detail in numerous other documents. The aim here is to eliminate certain problems encountered in the customer application subset.

- Just a reminder, the RULE based mode uses format rules (syntax, operator precedence, sub-query,...) and the indexes available. The COST mode uses data stored with the tables (volume, number of rows, average length, block usage) and the indexes (volume, selectivity, depth, tree balance, eventual distribution of keys). The second mode is the "" more "intelligent" as it takes into account the data held in the tables referenced by the statement.

In this case study we limit ourselves to checking that the objects are not too fragmented or chained so that the performance is not adversely affected. We will not be dealing with the problem of separating the I/Os onto different disks, another performance criterion.

The other points have been dealt with previously in an audit that was performed on the database. The Data Warehouse then was tuned, as well as specific system parameters. The physical data model is also not brought into question at this time.

3. Environment

3.1. *Servers and Instances*

The Data Warehouse or Info Center database used for reference was running Oracle 732 on a DIGITAL UNIX 8400 server with DEC UNIX 3.2g.

3.2. *Tools*

The tools used to generate the execution plans or to trace during the execution were (Oradebug and TKPROF) (cf. Oracle 7 Server Tuning).

It is also possible to use the AUTOTRACE mode (SET AUTOTRACE ON) under SQL*Plus, this gives indications, as well as the plan, after an execution on the volumes of data treated (blocks, physical and logical I/O).

4. Description of the transaction

The aim is to regroup from the customer database (CUST) all information relating to one client, held in several tables, into a single client table in the data warehouse.

Number of rows in the customer database - CUST:

Table Name	Rows	Transactions Numbers Used
GIFT	131114	5
CONTACT	612231	1
CUSTOMER	608908	1, 4, 5, 6
CUSTVAL	524031	9
GEOGRAPHIC	446135	2
DUNN_CUS_INF	571750	7
CUS_INF_CHK	603989	3
CUS_INF_JOIN	603989	3
CUS_INF_RES	603989	3
RATE	76	1
CREDIT	579471	10
CREDITTYPE_ALL	4	10
TERMS_ALL	6	4
TITLE	3	1
TRADE_ALL	10	4

The client subset application which consists of 13 PL/SQL stored procedures executed in a sequential manner, essentially performs updates on the local CLIENT table.

The first procedure, pr_client_1, loads the tables if it is empty, or updates the relevant data. The other procedures complete the set of information with updates. We are therefore capable of distinguishing between the two types of transactions.

The updates in this case are always run and will be looked at in detail. The general transaction template below is applied to all the procedures where possible.

```

Cursor_lines_tbd =
    SELECT .....
        FROM vw_client_x
    MINUS
    SELECT .....
        FROM CLIENT;
LOOP
    Returns one line from Cursor_lines_tbd

```

```
UPDATE : update client information
END LOOP
```

vw_client_x is a typical view returning data from CUST (via a database link). There is a different view for each procedure. The MINUS operator is used to bring only the increment from the CUST database

These views are joins between CUST tables that contain several hundred thousand rows. These views are not at all selective, as at each execution we are interested in all the customers and as a result, the joins deal with extremely large volumes of data. In these types of transactions we are interested in the overall transaction time (Batch) rather than first rows (On Line Transaction Processing - OLTP)

This will be shown in detail later in the explain plans.

5. Problems encountered

5.1. A systematic Update against a selective update.

Previous versions of the procedures performed systematic updates of all the clients.

Experience showed, that in a data warehouse model, the approach is not to use the same method practised by other tools, such as the middleware product used previously. With PL/SQL, transactions can be addressed in a more intelligent fashion given the power of SQL combined with a procedural language.

In only updating the data modified since the last transaction the following benefits are in evidence:

- few updates of the data and their indexes (7 on the CLIENT table)
- few locks placed on the rows
- considerable gain in elapsed time.

Note:

If the increment is small, the time allocated per update becomes negligible in comparison with the time required to perform the analysis between client data from CUST and that of the DW - data warehouse.

5.2. Presence of « hints »

Numerous *hints* were present in the procedures as well as in the views. The hint was declaring the use of *RULE* (RULE based optimizer, therefore no knowledge of data volumes). The effect of these hints was that Oracle was choosing the indexes in the table joins as preferred with OLTP. The result being 2, 3 or 4 times slower than the current optimized times.

Without the hints the database was in the CHOOSE mode (using statistics if present). The joins were performed without the indexes, using NESTED LOOPS and in most cases the HASH JOIN, which is well adapted to joins between large tables where full table scans are the norm.

5.2.1. Transaction N°10

View v_client_10 :

```
CREATE OR REPLACE VIEW V_CLIENT_10 AS
SELECT /*+ RULE */
    T1.CUSTOMER_ID,
    T1.BANKACCNO,
    T2.CREDITCODE
FROM
    CREDIT@CUSTPROD T1,
    CREDITTYPE_ALL@CUSTPROD T2
WHERE
    T1.CREDIT_TYPE=T2.CREDIT_ID(+)
AND T1.ACT_USED='X' ;
```

Execution Plan of the view v_client_10

with the hint <i>RULE</i>	without the hint <i>RULE</i>
<pre>SELECT STATEMENT REMOTE NESTED LOOPS OUTER TABLE ACCESS FULL CREDIT TABLE ACCESS BY ROWID CREDITTYPE_ALL INDEX UNIQUE SCAN I_P_CREDITTYPE_ALL</pre>	<pre>SELECT STATEMENT REMOTE HASH JOIN OUTER TABLE ACCESS FULL CREDIT TABLE ACCESS FULL CREDITTYPE_ALL</pre>

In the first plan Oracle performs a loop (NESTED LOOP) to retrieve each line of CREDITTYPE_ALL using the index I_P_CREDITTYPE_ALL. This loop is executed 579471 times. It is much cheaper for Oracle to use the HASH JOIN to execute the transaction. Then the execution time is halved.

A word of caution in the use of the hints. A previous analysis of the procedures had shown that a missing space ensured that the hint was not used. Thus syntax is very important.

5.2.2. Transaction N°1

The CUST portion of the SELECT (columns have been omitted for clarity) :

```

SELECT
      T1.CUSTOMER_ID NUM_CLIENT,
      T1.CUSTCODE CLIENT,          etc...
FROM
      CUSTOMER@CUSTPROD T1,
      RATE@CUSTPROD T2,
      CONTACT@CUSTPROD T3,
      TITLE@CUSTPROD T4
WHERE
      T1.CUSTOMER_ID=T3.CUSTOMER_ID AND
      T1.TMCODE=T2.TMCODE AND
      T3.CCTITLE=T4.TTL_ID(+)AND
      T1.CUSTOMER_DEALER='C' AND
      T2.VSCODE=0 AND
      T3.CCUSER='X' ;
    
```

A key typing error in the *hint* after the SELECT as follows:

Error	Correction
<pre> SELECT /* + RULE */ Execution Plan: SELECT STATEMENT REMOTE HASH JOIN TABLE ACCESS FULL RATE HASH JOIN OUTER HASH JOIN TABLE ACCESS FULL CUSTOMER TABLE ACCESS FULL CONTACT TABLE ACCESS FULL TITLE </pre>	<pre> SELECT /*+ RULE */ Execution Plan: SELECT STATEMENT REMOTE NESTED LOOPS NESTED LOOPS NESTED LOOPS OUTER TABLE ACCESS FULL CONTACT TABLE ACCESS BY ROWID TITLE INDEX UNIQUE SCAN I_P_TITLE TABLE ACCESS BY ROWID CUSTOMER INDEX UNIQUE SCAN I_P_CUSTOMER TABLE ACCESS BY ROWID RATE INDEX RANGE SCAN I_P_RATE </pre>

A space before the plus sign made Oracle consider the hint a comment

N.B.

Oracle has no error messages for the hints, they are simply ignored. The close examination of an explain plan is the only sure way to guarantee that the hints have been used.

A version of the transaction was tested with the error and then without:

- Error - the explain plan testified to the COST mode
100.000 lines in 2 minutes
- Correction - the explain plan testified to the RULE mode
100.000 lines in 5 hours

Production Panic

This then, was the panic in production, when a new delivery of the software, which appeared correct and had previously shown results of 500,000 clients treated in 40 minutes, had now only processed 100,000 clients in 5 hours.

We can see that the number of nested loops increases the transaction time exponentially

transaction 1	3 levels of loops = 25 hours	hash join = 10 minutes	factor 150
transaction 10	1 level of loop = 3min	hash join = 1min 30sec	factor 2

5.2.3. Removal of all the « hints »

ALL the HINTS were deleted from seven of the procedures and five of the views.

5.3. Chaining

Several tests were performed with PCTFREE 15 for the CLIENT table instead of 35 as in production.

The transactions performed massive updates after the first insert and the result was 189000 lines chained and the performance plunged in consequence. So yet again we see that PCTFREE and chaining have to be dealt with carefully.

5.4. Fragmentation

Attention must be taken with INITIAL and NEXT parameters. New deliveries of scripts from integration gave frequently tens of extents after the first transaction for the CLIENT table alone.

5.5. Rewriting / Modification of the transactions

5.5.1. V_client_4 view

Creation order:

```

create or replace view v_client_4 as
SELECT
    T1.CUSTOMER_ID,
    T2.TRADECODE,
    T3.TERMNET
FROM
    CUSTOMER@CUSTPROD T1,
    TRADE_ALL@CUSTPROD T2,
    TERMS_ALL@CUSTPROD T3
WHERE
/* line modified */          T1.CSTRADECODE=T2.TRADECODE| |null
AND
-- line before modification  T1.CSTRADECODE=T2.TRADECODE AND
                                T1.TERMCODE=T3.TERMCODE AND
                                T1.CUSTOMER_DEALER = 'C'

WITH READ ONLY;
```

Before Modification	After Modification
SELECT STATEMENT REMOTE NESTED LOOPS HASH JOIN TABLE ACCESS FULL TERMS_ALL TABLE ACCESS FULL CUSTOMER INDEX UNIQUE SCAN I_P_TRADE_ALL	SELECT STATEMENT REMOTE HASH JOIN TABLE ACCESS FULL TERMS_ALL HASH JOIN INDEX FULL SCAN I_P_TRADE_ALL TABLE ACCESS FULL CUSTOMER

The concatenation T2.TRADECODE||null disables the index I_P_TRADE and this neutralizes the loop. The execution time reduces from 1hour 30 min => 2 min 30 sec

5.5.2. Transaction Pr_client_13

This transaction is even more particular. It's a calculation of the average amount of the client's bill excluding the first bill. (this only being distinguished by a value of a sequence). The negation blocks Oracle from using the HASH JOIN. Here we must wake up the gray matter to ensure our desired effect.

Statement before modification :

```

SELECT
  COUNT(BILL_KEY)          NB_BILL,
  SUM(T1.AMT_BILL)/COUNT(BILL_KEY) AMT_AV_BILL,
  T1.CLIENT                CLIENT
FROM BILL T1
WHERE T1.TYP_BILL='IN' AND
      T1.BILL_KEY <> (SELECT MIN(T2.BILL_KEY) FROM BILL T2
WHERE T1.CLIENT=T2.CLIENT AND T2.TYP_BILL='IN' )
GROUP BY T1.CLIENT
MINUS
SELECT
  NB_BILL,
  AMT_AV_BILL,
  CLIENT
FROM
  CLIENT;

```

Here the select is easily understandable.

Execution Plan:

```

SELECT STATEMENT
  MINUS
    SORT UNIQUE
      SORT GROUP BY
        FILTER
          TABLE ACCESS FULL BILL
          SORT AGGREGATE
            TABLE ACCESS BY ROWID BILL
            INDEX RANGE SCAN IX_CLIENT_BILL
        SORT UNIQUE
          TABLE ACCESS FULL CLIENT

```

After being rewritten the result is:

```

SELECT
  COUNT(BILL_KEY)-1          NB_BILL,
  (SUM(T1.AMT_BILL)-(sum(t2.amt_bill)/count(bill_key)))
  /decode(COUNT(BILL_KEY)-1,0,1,COUNT(BILL_KEY)-1) AMT_AV_BILL,
  T1.CLIENT                CLIENT
FROM BILL T1,
  (SELECT client, amt_bill FROM BILL
   where (client,bill_key) in (select client,min(bill_key)
                               from bill
                               where typ_bill='IN'
                               group by client
                              )
   ) t2
WHERE T1.TYP_BILL='IN' AND
      T1.client=t2.client
GROUP BY T1.CLIENT
having count(bill_key)>1
MINUS
SELECT
  NB_BILL,
  AMT_AV_BILL,
  CLIENT
FROM

```

```
CLIENT;
```

Thus the select becomes slightly more difficult to understand.

The trick consists in transforming the negation into a join, so the hash join can be used. We therefore total all the bills and subtract the first bill.

Execution Plan:

```

SELECT STATEMENT
  MINUS
    SORT UNIQUE
      FILTER
        SORT GROUP BY
          HASH JOIN
            HASH JOIN
              VIEW
                SORT GROUP BY
                  TABLE ACCESS FULL BILL
                TABLE ACCESS FULL BILL
                TABLE ACCESS FULL BILL
            SORT UNIQUE
              TABLE ACCESS FULL CLIENT

```

Written thus, the transaction comes down from 20 minutes to less than 2 minutes.

Therefore we have a good illustration of the eternal dilemma: easily understood but not so efficient, or a little inspired and extremely fast.

5.6. *Parallel degree*

This parameter is essential if we wish to benefit from the parallel running of certain types of transactions such as the full table scans. In production the value on the client table is set to 2. It was chosen in relation to the number of CPU's and disks available.

Each new delivery of the application the scripts ignores the parameter and sets it to **default**.

This is just to ensure that we are on our toes, this is however typical of day to day problems.

5.7. *READ ONLY*

This clause has been added to all the views - WITH READ ONLY. This ensures that nothing can be inserted across views, even if this is impossible with views that join several tables. This is a measure that is recommended for reasons of security and clarity.

5.8. *Gain*

With the modifications, the overall transaction time has been reduced from 5 hours to less than one hour.

6. Suggestions, warnings

- ✓ **The use of hints is forbidden to developers** apart from exceptional cases and this defined by the DBA! Moreover, experience shows that developers tend to derive a new query from a previous one which performs well... and keep hints which no longer apply to the case.
- ✓ The statistics on the tables must be up to date for a correct parsing using the costs.
- ✓ Use of indexes is not necessarily the first act when optimizing - it depends on the type of transaction
- ✓ Detailed optimization of transactions is a project in itself. This requires understanding of the types of transactions in this domain and of the internal mechanisms used by Oracle. This is also an effort which requires time!!

N.B.

This study shows that optimization can be done but a statement by statement approach can bear fruit and it takes time and understanding.

- ✓ **Identify** the statements that consume 80-90% of the global time. Some companies, including Oracle, can provide some tools and Oriole Corporation will have one available soon. The analysis must then be performed by somebody with experience.
- ✓ The optimization of one query has nothing to do with the optimization of another query, even if the two look superficially similar. Beware of quick generalizations.