Andrzej Barczak[1]
Dariusz Zacharczuk[1]
Damian Pluta[1]

[1] University of Natural Sciences and Humanities, Institute of Computer Science, 3 Maja 54, 08-110 Siedlce, Poland

Tools and methods for optimization of databases in Oracle 10g. Part 2 – Tuning of hardware, applications and SQL queries

Abstract. Article provides information on effective optimizing of Oracle. Discussed aspects are: hardware and statistics tools on which one can build the optimal SQL code.

Keywords. database optimization, Oracle 10g, tools and methods

1. Introduction

Storing large amounts of information at a relatively low price become possible. This has enabled users to increase the amount of data and processing them in an increasingly complex ways. For optimum performance, you can not concentrate on only one part of the system. It is necessary to examine the application, database instance, operating system and hardware configuration. In this article we will be discussed aspects of the hardware, database instance and tools that have DBMS to achieve good optimization. Covered subjects have a direct impact on database performance. This document may be seen as a continuation of the article "Tools and methods of databases optimization in Oracle Database 10g. Tuning instance".

2. Tuning hardware – IO operations

IO subsystem is a vital component of the Oracle database. Performance of many applications is limited by the input-output operations. The IO subsystem is often the cause of performance issues in Oracle. The problems associated with the IO translate directly into system performance.

2.1. Hard drive (HDD)

The disk is a key component of the IO subsystem. It is one of the few mechanical components in your computer, so the laws of physics determine the boundaries of its

performance. Disk is made up of plates, on which are the paths. Each path is divided into sectors where data is stored. Over the rotating plates there are moving heads, reading the data from the sector. Head movement only takes place inside and outside the axis of the plate. The amount of plate rotation has the impact on the speed of data reading. Move head over the tracks is called scanning. However, the movement over the sectors under the head or movement of plates is called rotation.

To read data from a particular sector, the head must first set the appropriate path (seek time) and then wait until the plate is rotated so that the head was in the right sector (rotational latency).

Time Search - searches can be divided into three types:

- full disk seek - the disk head has to move from the outermost track to the innermost track or vice versa. For high-performance SCSI drives this type of search time for reading is 7.5 ms, and 8ms for the record. Search of this type are very rare - these parameters are often used to calculate the worst-case time.
- Track-to-track - the head moves from one track to an adjacent. For SCSI disks, the search time is 0.5ms for read and 0.7ms for the record - it is the fastest type of search.
- Free search - they can include a type intermediate between the two previously described. Time to search for the same SCSI disk is 3.9ms for read and 4.5ms for the record. This type of search has the greatest impact on performance.

Rotational delay - waiting time depends mainly on the speed of plates rotation. In the worst case, the waiting time will be equal to the amount of time required for one complete rotation. However, the average waiting time is half of the maximum waiting time. Time of one complete plate rotation of the disk on which they spin at a speed 15000rpm is 1.11ms, therefore the average time of delay is equal to 0.55ms. The time required to read data from the disk includes the following factors:

- search time;
- rotational latency;
- transfer time.

From the HDD data will be sent to the controller, hence to the delay generated by the hard drive, you still need to add delay introduced by the controller. Other discs delays will be discussed later in this article.

At the moment the drive can support only one input and output operations. When the controller realized a request to perform input-output operations, and drive processes the previous operation, the request is queued. As you approach the theoretical performance limit, the delay caused by HDD is increasing.

2.2. Redundant Array of Independent (or Inexpensive) Disks − RAID

RAID are designed to achieve two objectives:

- increase fault tolerance - failure of one of the hard drives will not lose data.
- configuration of multiple small disks into one large virtual disk, which is easier to manage and can work faster.

Performance benefits from use of RAID is to increase data transfer rates

and increasing the number of input-output operations per second. There are different ways of striping data and different methods to provide fault tolerance. These different configurations are called RAID levels. Each RAID level has a different level of protection against failures, performance characteristics and price. RAID can be implemented in two ways: hardware or software.

Hardware RAID arrays are more efficient, because other processes require CPU time do not affect the RAID. An additional advantage is that the hardware implementation allows on hot_swap, as well as the ability to generate warnings about errors that lead to failure. On the other hand, hardware RAID involve certain limitations. Striping can apply only to those directly connected drives, and in addition the number of these drives is limited.

Software RAID is cheaper, since you do not have to bear the costs associated with the purchase of additional hardware. They can strip through all the disks contained within the system, and thus, you can create arrays of disks attached to different input-output controllers.

2.3. Raid levels

The most common levels are 0, 1, 10 and 5. Besides them, there are 2, 3, 4 and 6 but they are extremely rare and are not included in the article.

RAID0 is characterized by simple striping without fault-tolerance, and therefore it is not a redundant array. In RAID0 striping data is available on a single small disk, forming one large virtual disk. Failure of any drive results the loss of data from the virtual drive.

RAID1 (mirroring). All data stored on the hard disk is duplicated on another disk. Mirrored disks work in pairs, so even if half of them are broke it does not disturb the system to work. The saving time is equal to the time on the slower of a pair of disks. However, when reading the data, we have seen a substantial profit, which is caused by the fact, that different data can be simultaneously read from mirrored disks. The data is read from that disks, in which the heads are closer to the path where the data is physically stored. RAID1 is the safest but also the most expensive option.

RAID10 is a combination of RAID0 and RAID1, thus combining the advantages of striping and mirroring. RAID10 achieves high performance and high security. In the case of databases, RAID10 configuration is the most recommended

RAID 5 uses parity striping. Parity is based on the use of appropriate algorithms to obtain a value that will allow you to recover data from lost hard disk. Parity information is categorizes for all the disks included in the system. Writing need 4 IO operations, which is related to the maintenance of parity. However in RAID5 at the same time more than one write operation can be done. RAID5 is an economic system but the downside is the poor record performance. RAID5 should be used where the save to read ratio is 1:9 or more.

2.4. Summary of RAID performance

In RAID0 performance calculation is the easiest. Assume that is it fo free access, the optimal number of operations in this matrix is:

*(# Readings) + (# of records) = m * n,*

where:
m − the optimal number of free IO operations of the disk;
n − the number of drives in the array.
In RAID1 and RAID10 arrays record will need to make two IO operations. This is due to the fact that the stored data must be placed on two physical disks. The optimal number of reads or writes to the array is:

*(# Readings) + (# of records * 2) = m * n,*

where m, n – as above, for RAID1 n = 2
In RAID5 write involves up to four input-output operations. First you have to read the old data and old information associated with parity, then you need to perform two operations XOR, store the new data and new parity bits on disk. The optimal number of reads or writes to the array can be calculated with the following formula:

*(# Readings) + (# of records * 4) = m * n,*

where m, n – as above
On the basis of the information it can be concluded that:
RAID0 is the most efficient level, due to the lack of additional load caused by the mechanisms that protect against failures.
RAID1 and RAID10 despite the duplication of input and output for recording, also have a good performance. The added advantage is the best protection against failures. Unfortunately, they are the most expensive option.
RAID5 when writing takes up four times the IO operations, it gives him the worst performance record. This RAID level provides protection against failures, but can tolerate the loss of only one drive in the array.
The good news is that none of these levels does not cause additional overhead IO operations during reading.
If the goal is to ensure optimal performance and resilience to failure, the best solution may be RAID10.

3. Oracle and IO operations

3.1. Oracle relationship between input and output devices

Performance of Oracle server depends on the performance of IO subsystem. The time takes to execute query is the time you will have to spend waiting for the results you want. 6.3ms (if there is no queuing effect) appears to be negligible latency. The problem is that one read operation is not sufficient to complete the query. It is often necessary millions of such operations.

Oracle performance relationship does not result directly from the write speed. In the case of data recording, mechanism of action is somewhat different than in the case of reading data. Users can modify only the data in the buffer. However, writing modified data to disk can be don only by DBWR process. This process work in the background, so the efficiency of the operation record is not as critical as in the case of readings. That mechanism records only data files. The situation is different for the redo log area.

Transactions can not be considered completed until the relevant information is not recorded in the redo log files. This information is first written to the redo log buffer, where LGWR process rewrites it to the files. When the log buffer is full, the operations will have to wait as long as the buffer becomes available. Therefore, it is fair to say that writing to redo log has great importance for the performance of the Oracle server.

If the database does a lot of data modification operations, rollback space is very busy. The excessive number of savings create delays, that directly translate into performance of database.

The situation is quite different for the records to the data file. Here, write delay are usually not a problem. These records are postponed and carried out in the background by the previously mentioned DBWR process. If this process carried out emptying the buffer smoothly and the process of handling checkpoints does not last too long, writing delayed should not be a problem.

3.2. Tuning IO operations

In the previous sections was talking about internal disk limitations that can not be missed. The only thing we can do is take these constraints already during the design phase of the system. After installation, first you need to deal with is the optimization of the memory, the next step will be to optimize the disk. In the reverse case, optimization time would be unnecessary wasted on deal with miss of the buffer, which always generates additional IO operations. When tuning disks, make sure that the disk load was maintained within the limits set by their performance. In order to detect the drives that have exceeded the limit of performance (hot spot), it is necessary to monitor the system. Disk performance limits are exceeded when there is strong competition for access to it.

Information on whether the capacity limit is exceeded we get by studying IO statistics generated by Oracle and operating system. Oracle provides accurate statistics for the IO data files, but often working of the entire drive is also affected by factors not related to Oracle. Information on the number of physical reads and writes can be obtained using the performance view V$FILESTAT, supplementing them with information from predictive V$DATAFILE, which will give us the name, type, current status and file size. Padding is necessary, because the view V$FILESTAT uses internal file identifiers.

To determine the competition for HDD is enough to run the monitoring process from the OS or other third-party tools. Check your IOPS. For single drives it should not exceed 125 with free calls, and 300 for the sequence. Delays in the range of 10-20ms is also acceptable, but all over 20ms may already be a serious problem. For RAID it is necessary to divide the load obtained by the number of disks. This method is correct for the most current RAID. Delays obtained for the entire matrix stays as it is, do not divide it.

After examining the level of IOPS and latency, it was just estimate types of IO operations. Free IO operations differ from sequential movement performed by the head. First one increased and second minimize head movement. Access to the data file is almost always free and for redo log file is always sequential.

There are several ways to reduce disk competition:

- Sequential isolation of input and output - it is simply a way to ensure than by placing redo log files on separate disks, especially in the case of mirrored files by Oracle.
- Balancing free IO operation - this kind of load you can easily control by adding additional drives and strip tables between them. Striping may be carried out by the Oracle server, operating system or hardware.
- Separation indices from the data - the benefit of this is you can parallel access to data and indexes. In order to find tables where access is often implemented, you can use the Oracle performance views.
- Elimination of references from sources other than Oracle - additional disk operations, of non-Oracle may cause a negative impact on the performance of Oracle.

4. Tuning applications and SQL queries

At first couple of words about tools, that will be used. Oracle provides several tracing tools that can help you monitor and analyze applications running against an Oracle database. End to End Application Tracing can identify the source of an excessive workload, such as a high load SQL statement, by client identifier, service, module, action, session, instance, or an entire database. This isolates the problem to a specific user, service, session, or application component. There is also command-line utility – trcsess, that consolidates tracing information based on specific criteria. Finally the SQL Trace facility and TKPROF are two basic performance diagnostic tools that can help you monitor applications running against the Oracle Server.

We will use SQL Trace facility and TKPROF, because they let accurately assess the efficiency of the SQL statements an application runs.

4.1. Using SQL Trace and execution plan

EXPLAIN PLAN shows the execution plan for the SELECT, INSERT, UPDATE, and DELETE. With such a plan, and based on knowledge of the application and the content of the tables, you can determine whether the Oracle optimizer choose the best variant of the query. We get a plan without having to actually execute the query. Analyzing the plan result, you can enter information about how to perform queries that will be considered by the optimizer.

EXPLAIN PLAN command creates a clear description of the steps to be taken by Oracle to perform SQL queries. These descriptions include information about how the request will be made. This information includes:

- tables used in the query and the order in which it will be implemented access to them;
- data operations such as sorting, filtering and aggregation;
- access method for each table mentioned in the SQL query;
- concatenation methods for tables involved in join operations defined in the query;
- the cost of each operation.

The output of the EXPLAIN PLAN is placed to special table with a default name plan_table. To create an execution plan for the query, use the EXPLAIN PLAN FOR clause immediately before asking, for example:
EXPLAIN PLAN FOR SELECT last_name FROM employees;
This statement will save the query execution plan in plan_table. After creating the execution plan, the results can be downloaded using the procedure DBMS_XPLAN.DISPALY. This procedure takes optional parameters such as:

- name of the table with a plan, if the name is different than the default;
- ID of the execution plan, if it was specified when creating the plan for a query;
- format option, which depends on the amount of detail: BASIC, SERIAL, TYPICAL and ALL.

E.g.

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());

The example creates a plan for the query, choosing a Employee_ID, job_tittle, and department_name salary for employees whose id is less than 103

```
EXPLAIN PLAN FOR
SELECT e.employee_id, j.job_title, e.salary, d.department_name
FROM employees e, jobs j, departments d
WHERE e.employee_id < 103
AND e.job_id = j.job_id
AND e.department_id = d.department_id;
```

The execution plan looks like shown in Figure 4.1.

```
-------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     3 |   189 |    10  (10)|
|   1 |  NESTED LOOPS                 |              |     3 |   189 |    10  (10)|
|   2 |   NESTED LOOPS                |              |     3 |   141 |     7  (15)|
|*  3 |    TABLE ACCESS FULL          | EMPLOYEES    |     3 |    60 |     4  (25)|
|   4 |    TABLE ACCESS BY INDEX ROWID| JOBS         |    19 |   513 |     2  (50)|
|*  5 |     INDEX UNIQUE SCAN         | JOB_ID_PK    |     1 |       |            |
|   6 |   TABLE ACCESS BY INDEX ROWID | DEPARTMENTS  |    27 |   432 |     2  (50)|
|*  7 |    INDEX UNIQUE SCAN          | DEPT_ID_PK   |     1 |       |            |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("E"."EMPLOYEE_ID"<103)
   5 - access("E"."JOB_ID"="J"."JOB_ID")
   7 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

Figure 4.1. Execution plan

Analysis of a sample implementation plan:

- step 3, read all the rows in the table employees;
- step 5 checked each job_id in the index JOB_ID_PK and locates all identifiers of rows related with jobs table;
- step 4 collected rows of identifiers that are returned in step 5;
- step 7 checked each department_id in the index DEPT_ID_PK and locates all identifiers associated with the department table;
- step 6 collected rows of identifiers that are returned in step 7

The following steps in the example presented above, operate on rows returned in the previous steps:

- in step 2 nested loops are executed. The input data are rows returned in steps 3 and 4. Each row in step 3 is connected to the associated row from step 4, and the result is returned to step 2;
- in step 1, the nested loops are executed. The input data are rows returned in steps 2 and 6. Each row of step 2 is combined with a related row from step 6, and the result is returned to step 1.

4.2. SQL Trace i TKPROF tools

SQL Trace and TKPROF are complementary Oracle tools. SQL Trace helps you track the execution of SQL queries. The TKPROF helps to format the trace files generated by SQL Trace tool into a readable form. Trace generates the following statistics for each query:

- CPU time and query time;
- number of parses, performances and downloads;
- physical and logical reads;
- the number of affected rows;
- execution plan;
- number of miss in the buffer library.

SQL Trace lets you see if your question does not consume excessive amounts of resources, amount of time spent by the query processor and the amount of input and output. The execution plan, which is an optional part of SQL Trace, returns the number of rows for each step of the plan, as well as the total number of appeals to the buffer (cr), physical reads (pr) and records (pw) and execution time (time) of the current step and the previous steps. Information will help to identify a step in which it is consumed the most resources. Comparing the consumption of resources to the number of rows returned by the query, you can easily determine how productive is a given query.

4.3. Configuration of initialization tracking parameters

Before enabling the SQL Trace, you must make sure that the initialization parameters are set correctly:

- TIMED_STATISTICS - Enables the collection of statistics relating to the measuring of time. Enabling this option will significantly slow down your system.

- MAX_DUMP_FILE_SIZE - specifies the maximum length of a track - the unit is a block of the operating system.
- USER_DUMP_DEST - the location of your trace.

Collecting tracking statistics is completely optional process and can be enabled for a single session or the entire instance. When tracking is enabled for the session, the resulting trace files are very detailed and complex. TKPROF convert these files to a human-perceivable form. By specifying the appropriate parameters, we have a small influence on the appearance of the final report.

Each TKPROF report presents the following data on executed SQL queries, when the user's session was been tracked:

- SQL query;
- number of parses, performances and downloads;
- the number of affected rows;
- time CPU usage;
- the number of IO operations;
- number of miss in the buffer library;
- optional execution plan;
- summary, stating the number of similar and different queries in the trace file.

TKPROF report helps identify inefficient SQL queries. The tool can organize your query, for example according to the execution time, which determines query on which to focus during the optimization process.

SQL Trace is a powerful tool for tuning SQL queries, because it provides much more information than can be obtained using the EXPLAIN PLAN. Provides information about the number of different types of requests directed to Oracle during the execution of a query, and also about how resources are used at various stages of implementation.

5. Oracle query optimizer

After considering a number of factors related to access to the objects that are referenced in the query, Oracle query optimizer determines the most efficient way to execution your request. The output of the optimizer receives a query execution plan that describes the optimal way of its implementation. Query optimization is provided by the Oracle server. For each SQL query optimizer does the following:

- Analysis of expressions and conditions – first optimizer analyzes the term and conditions containing constants.
- Transformation of the query - for complex queries optimizer can make some transformations in order to obtain equivalent query, but performed more efficiently.
- Selecting points of optimizer - determines the purposes of optimization.
- Choice of paths - for each table mentioned in the query optimizer chooses one or more of the available access paths to obtain data from the table.

- Choose the order connectors - for queries where are join more than two tables, the optimizer chooses which pair of tables is joined to the first, then, that the tables will be attached to the previous junction, etc.

We can affect on optimizer choices by setting the destination of optimizer and by collecting statistics about the distribution of the data and the conditions of their storage in a database. Using the instructions in the SQL query, you can specify how the query optimizer should be done.

5.1. Selecting optimizer points

By default, the query optimizer goal is the best throughput. This means that it chooses the least amount of resources needed to process all the records, to which access is being implemented in the query. Oracle can also optimize the query for the shortest response time. This means that it chooses the least amount of resources necessary to process the first record that is referenced by the query.

For applications that perform series of scheduled tasks, the best choice is to choose, as optimizer target, the best throughput. In the case of interactive applications, the goal of the optimizer should be set to the fastest response time.

When selecting an approach to optimizing SQL queries, the behavior of the optimizer are affected by the following factors:

- OPTIMIZER_MODE initialization parameter;
- optimizer hints changing its purpose;
- query optimizer statistics, contained in the data dictionary.

OPTIMIZER_MODE initialization parameter establishes the default behavior for the choice of approach to optimizing SQL queries:

- ALL_ROWS - cost mode for all queries in the session, regardless of the presence of statistics. Goal is the best possible throughput.
- FIRST_ROWS_n - cost mode, regardless of the presence of statistics. Goal is as shortest response time as possible to collecting the first n records, n can be 1, 10, 100 or 1000.
- FIRST_ROWS - cost and heuristic approach to find the best plan for fast delivery of the first few records. Using a heuristic approach sometimes leads to designate the plan, that cost of which is larger than when using only the cost approach. FIRST_ROWS is available due to backward compatibility.

If the SQL query optimizer uses a cost-based approach, in the absence of statistics for one of the tables listed in the query, the optimizer uses internal information such as the number of data blocks allocated to the table, in order to estimate the missing statistics for this table.

To specify the target for query optimizer for individual SQL statements, use one of the tips that will override the value of OPTIMIZER_MODE:

- FIRST_ROWS(n) - SQL data will be optimized for the shortest possible time to return the first n records.
- ALL_ROWS - for the best throughput.

5.2. Query optimizer statistics

The statistics are used by the query optimizer to choose the best execution plan for each SQL statement. Optimizer statistics include:

- table statistics
  - the number of records
  - the number of blocks
  - average record length
- Columns statistics
  - the number of different values in a column
  - the number of null values in a column
  - data distribution (histogram)
- Index statistics
  - number of leaves
  - level
- System statistics
  - efficiency and the use of input-output
  - performance and CPU utilization

Because the objects in the database can be constantly changing, statistics must be updated regularly. The statistics are collected automatically by Oracle or can be maintained manually using the DBMS_STATS package. The recommended approach for collecting optimizer statistics is allow the Oracle system to do so. Oracle collects statistics on all database objects automatically and update it regularly.

The query optimizer determines which execution plan is most effective on the basis of possible paths and the information contained in the optimizer statistics for objects specified in the SQL query. The query optimizer takes into account the instructions that are included in the query. The query optimizer performs the following steps:

1. It generates a set of potential execution plans for SQL statements based on possible access paths and directions.
2. Estimated cost of each plan based on statistics for the optimizer, stored in the data dictionary, for objects used by the query. The optimizer calculates the cost of access paths and join the user, based on the estimated hardware resources, including the number of input-output processor time and memory consumption.
3. Compared to the cost of all execution plans and chooses the plan with the lowest cost.

6. Tuning SQL queries

6.1. The implementation of SQL commands

Knowing how SQL commands are executed, allow to better understood methods of optimization. Here are the steps that Oracle must perform for each DML command:

1. Create a cursor - The structure of the cursor is created by Oracle for each SQL command.
2. Parsing the command - In this step, Oracle checks, if in the shared area is already parsed form of the command. If so, it shall be used. You can go for further steps. Otherwise, Oracle must perform the parsing of the command. Parsing is a complex process that:
    1. Validate the command syntax.
    2. Validate object references.
    3. Put required locks on the objects.
    4. Check the access permissions to the data to which access will be required.
    5. Appoint an optimal execution plan for the command.
    6. Placing a command in the shared area.
    7. Sending all or part of the distributed command to the respective nodes of distributed database.
3. Description of results of a query (SELECT only) - It is only necessary if the characteristics of the query result is not known, for example, the query is entered interactively by the user. At this step, the properties of the query results are defined (data type, length and names).
4. Output query definition (SELECT only) – Defined are the location, size and types of data variables, defined to receive each valuews retrieves by query.
5. Bind variables - If in the command are bind variables, then in the calling program corresponding variables should be set up and addresses of those variables should be specify to Oracle. Because the transfer of these parameters is carried out by reference, so in the case of re-run the command, there is no need to re-bind. Just change the values of these parameters in the program variables. In addition to the address of each variable, you must still specify the type and length, unless it can be inferred or are default.
6. Parallelization of command (only if this is possible) - Oracle can parallel commands such as SELECT, INSERT, UPDATE, MERGE, DELETE, and some DDL: create index, create a table with a subquery, and operations on the partitions. Parallelism causes executes SQL commands by many server processes, so work is done in less time.
7. Execution of the command - For UPDATE and DELETE commands, all rows, which is affected by these commands will be disabled for all other users. The lock will remain in effect until the next COMMIT, ROLLBACK, or SAVEPOINT for the transaction.
8. Gets the records returned by the query (SELECT only).
9. Close the cursor.

6.2. Types of joints

Oracle optimizer tries to choose the best type of join, but sometimes for various reasons, his decision is not optimal. It is therefore important to monitor the results of the EXPLAIN PLAN command and, if necessary, change the join algorithm.

There are three types of joints:

- nested loops;
- sort-merge;
- hash-join.

Each of the algorithms has different characteristics, which makes it in some cases good and not in others.

Cartesian joins are usually intended result, but the logical consequence of errors in the query. They occur in cases where the join does not contain WHERE clauses. Even the merging of small tables should be avoided of cartesian join because it is inefficient.

In the nested loops algorithm, outside table (called a driving table) is read line by line. The inner loop runs for each row of the outer table, looking for matching rows in the inner table. This algorithm do not stress CPU, but it generates a lot of input and output. Nested loops algorithm is suitable for the merging of small subsets of data, in which the output will be returned less than 10 000 rows.

In the case of sort-merge algorithm, the two tables are sequentially read in its entirety and ordered by key join. Then the tables are both viewed from the beginning. The outer table is read one record and compared with records which are in the initial part of the inner table. Because the data is already sorted, you can easily tell if the corresponding record is in the inner table. This process is repeated for each row of the outer table. Sort-merge algorithm does not result in high CPU utilization, but requires a large number of input and output. In the case of sort-merge algorithm, the number of input and output is smaller than the algorithm using nested loops, and the higher the CPU utilization. This type of junction works best in cases where the connector tables are provided bumps or when the tables are already sorted by the join key and re-sorting is not required.

Hash join algorithm is used for merging large tables (data sets). This algorithm selects the smaller of the table, then for selected table builds the hash table over the joins key and stores in memory. The next step is to scan a larger table, linked with checking the hash table in order to find corresponding rows. Hash join algorithm can be used only in case that the join is based on the condition of equality. This method works best when the smaller tables fit in memory. In this case, the cost is limited to a single read of each of the tables. Joining using a hash join algorithm generates a small charge for the disk and CPU but memory are used extensively.

6.3. Writing efficient SQL code

Effective SQL stands for high performance and using a minimum amount of hardware resources. Even the best configured server may lose smoothness in the exercise of ineffective commands.

Selectivity criteria in the WHERE clause have a big impact on the amount of data that Oracle must process. Inter alia on the basis of that criteria, the optimizer decides whether uses existing indexes or not. The key issue is the number of rows returned by the query, as a percentage of the total number of rows in the table. The low percentage of the data is high selectivity and vice versa. Because the WHERE clause with greater selectivity generates a smaller amount of input and output, so such a clause will be chosen by the Oracle cost-based optimizer, if you decide to use indexes. Using a WHERE clause

such as: *WHERE last_name LIKE '%MA%'*, optimizer may simply decide to bypass the index and do a full scan on table. For example, if the table has 1000 rows, placed in 200 blocks and the full scan is done, assuming that the value of the parameter DB_FILE_MULTIBLOCK_READ_COUNT is 8, it will be cost of 25 input-output operation. Conversely, when the index which is poorly selective − required to download it in 90% and consists of 40 blocks leaf, obtain cost is 32 input-output operation only to read indices.

When SQL functions are used in the WHERE clause (such as SUBSTR, TO_DATE, TO_NUMBER, etc.) Oracle optimizer ignores the index on that column. In this case, index based on the function should be used.

Most SQL queries involve a multi-tabular join. Often that join are performed in the wrong way, making query ineffective. Here are some tips on merging tables in the right way:

- Where possible, use a equi-join type, which rely on the junction of two or more tables based on the condition of equality between the two columns.
- Performing filtering at the beginning reduces the number of rows to be joined in the later steps.
- Joins should be carried out in the order that gives the smallest number of rows at the output.
- Wherever possible, use a WHERE clause instead of a HAVING clause. The WHERE clause immediately limits the number of rows affected. However, the HAVING clause forces the download many more rows than necessary.

SQL commands should be constructed so as to achieve data access, as rarely as possible. This means that you should get rid of the SQL statements that execute repeatedly access the table, in order to obtain the different columns. It should be replaced with instructions realize a multiple access.

6.4. Optimizer hints

Algorithms used by the optimizer does not replace the expertise of a programmer's about stored data. This is why, hints mechanism was created. Through this tips you can specify:

- approach and goals of optimization;
- how to access the data;
- order of merging tables;
- algorithm for merging tables;
- the degree of parallel execution of a query;
- other additional parameters.

Directions are passed to the optimizer in SQL comments after the "+" mark. SQL block can only have one comment with the instructions and it must follow the keyword SELECT, UPDATE, or DELETE. Hints applies only to this block SQL queries, in which you've placed your comment. If the SQL query is complex and consists of several blocks, for example, SELECT, UNION connected operator, each of the blocks must have their own guidelines. In Oracle's, comments come in two forms:

- single line, starting with two dashes: --+ direction

- multi line: /*+ direction */

SQL query example:

SELECT /+ FULL(bank_account) */ account_no, name, balance*
FROM bank_account WHERE account_no = '12631';

Directions will be ignored if they no following the keywords, including syntax errors or if they are contradictory. A common combination of instructions are FULL and PARALLEL:

SELECT / FULL(dogs) PARALLEL(dogs, 5) */ dogname, age, owner*
FROM dogs WHERE age < 5;

Optimizer Directions are grouped into the following categories:

- approach and goals of optimization: ALL_ROWS; FIRST_ROWS(n).
- access paths: FULL; CLUSTER; HASH; INDEX; NO_INDEX; INDEX_ASC; INDEX_COMBINE; INDEX_JOIN; INDEX_DESC; INDEX_FFS; NO_INDEX_FFS; INDEX_SS; INDEX_SS_ASC; INDEX_SS_DESC; NO_INDEX_SS. If the indicated path is not available, it will be ignored by the optimizer. Please note that provided method must include the name of the table, to which access will be realized. If query uses alias, instead of a table name, use the alias. If you set the guidelines for access paths or how to join tables together with the ALL_ROWS or FIRST_ROWS(n), the optimizer gives priority to guidance defining the path and ways to join.
- tips for transforming query: NO_QUERY_TRANSFORMATION; USE_CONCAT; NO_EXPAND; REWRITE; NO_REWRITE; MERGE; NO_MERGE; STAR_TRANSFORMATION; NO_STAR_TRANSFORMATION; FACT; NO_FACT; UNNEST; NO_UNNEST. With these tips you can affect the internal transformations of queries.
- directions concerns join order: LEADING; ORDERED.
- directions concerns type of join: USE_NL; NO_USE_NL; USE_NL_WITH_INDEX; USE_MERGE; NO_USE_MERGE; USE_HASH; NO_USE_HASH. Pointing join method in the SQL query, you must enter the table name or it alias. Using the USE_NL and USE_MERGE must be combined with directions about join order.
- hints for parallel execution: PARALLEL; PQ_DISTRIBUTE; PARALLEL_INDEX; NO_PARALLEL_INDEX. Parallel query execution can result in good performance effects.
- Other hints: APPEND; NOAPPEND; CACHE; NOCACHE; PUSH_PRED; NO_PUSH_PRED; PUSH_SUBQ; NO_PUSH_SUBQ; QB_NAME;

CURSOR_SHARING_EXACT; DRIVING_SITE; DYNAMIC_SAMPLING; MODEL_MIN_ANALYSIS.

7. Summary

Combining the knowledge presented above chapters and information from the "Tools and methods of optimization of databases in Oracle Database 10g. Tuning instance "we can say that the theory has been exhausted. There are objects designed to improve the performance of Oracle, ways to configure hardware as well as tools to assess the propriety of performing a SQL query. We also know how to react in cases, where Oracle automatic decisions are inefficient. The next step would be just to implement all this information in a life, that is a full optimization of the physical database system.

References

1. Alapati S.R.: Expert Oracle Database 10g Administration, Apress, 2005.
2. Barczak A., Florek J., Sydoruk T.: Bazy danych, Wydawnictwo Akademii Podlaskiej, Siedlce 2007.
3. Barczak A., D. Zacharczuk, D. Pluta: Tools and methods of databases optimization in Oracle Database 10g. Part 1 – tuning instance, Publishing House of University of Natural Sciences and Humanities, 2012.
4. Greenwald R., Stackowiak R., Stern J.: Oracle Essentials: Oracle Database 10g, 3rd Edition, O'Reilly, 2004.
5. Lonley K., Bryla B.: Oracle Database 10g Podręcznik administrator baz danych, Helion, Gliwice 2008.
6. Taniar D., Rahayu J. W.: A Taxonomy of Indexing Schemes for Parallel Database Systems. Distributed and Parallel Databases, Volume 12, Number 1, Kluwer Academic Publishers, pp. 73-106, 2002.
7. Liebeherr J., Omiecinski E., Akyildiz I. F.: The Effect of Index Partitioning Schemes on the Performance of Distributed Query Processing. IEEE Transactions on Knowledge and Data Engineering archive, Volume 5, Issue 3, 1993, pp. 510-522.
8. Helmer S., Moerkotte G.: A performance study of four index structures for set-valued attributes of low cardinality. VLDB Journal, 12(3): pp. 244-261, October 2003.
9. Bertino E. et al.: Indexing Techniques for Advanced Database Systems. Kluwer Academic Publishers, Boston Dordrecht London, 1997.
10. Oracle: Oracle Database Administrator's Guide, 10g Release 2 (10.2), Dokumentacja techniczna, 2006.
11. Oracle: Oracle Database 10g Administration Workshop I, Dokumentacja techniczna, 2004.
12. Oracle: Oracle Database 10g Administration Workshop II, Dokumentacja techniczna, 2004.
13. Oracle: Oracle Database Concepts, 10g Release 2 (10.2), Dokumentacja techniczna, 2005.

14. Oracle: Oracle Database Data Warehousing Guide, 10g Release 2 (10.2), Dokumentacja techniczna, 2005.
15. Oracle: Oracle Database Performance Tuning Guide, 10g Release 2 (10.2), Dokumentacja techniczna, 2008.
16. Oracle: Oracle Database SQL Reference 10g Release 2 (10.2), Dokumentacja techniczna, 2005.
17. Oracle PL/SQL Database Code Library and Resources, [online] http://psoug.org/reference/library.html.
18. Tow D.: SQL. Optymalizacja, Helion, Gliwice 2004.
19. Urman S., Hardman R., McLaughlin M.: Oracle Database 10g. Programowanie w języku PL/SQL, Helion, Gliwice 2007.
20. Whalen E., Schroeter M.: Oracle Optymalizacja wydajności, Helion, Gliwice 2003.