Tools and methods of databases optimization in Oracle Database 10g. Part 1 – tuning instance

Andrzej Barczak[1],
Dariusz Zacharczuk[2],
Damian Pluta[3]

[1] University of Natural Sciences and Humanities, Institute of Computer Science, 3 Maja Str. 54, 08-110 Siedlce, Poland, andrzej.barczak@neostrada.pl
[2] University of Natural Sciences and Humanities, Institute of Computer Science, 3 Maja Str. 54, 08-110 Siedlce, Poland, dzariusz@dzariusz.pl
[3] University of Natural Sciences and Humanities, Institute of Computer Science, 3 Maja Str. 54, 08-110 Siedlce, Polska

Abstract: The purpose of this article is to show the various questions and issues of databases optimization, and learn methods to improve the performance of the database based on the Oracle 10g DBS. The following parts will present working of these mechanisms, obtained effects and estimation of the results.

Key words: database, optimization, Oracle 10g, tools, methods

1 Introduction

Modern memory and hard drives are becoming increasingly cheaper and more capacious. Storage of large amounts of information for a relatively low price become possible. This allowed users to increase the amount of data and processing them in increasingly complex ways. For optimal processing performance, you can not concentrate only on one part of the system. It is necessary to analyze the application, the database instance, the operating system and hardware configuration. In this article we will raise the first aspect: the issue of optimizing databases. Discussed issues have a direct impact on database performance.

Optimization of the database involves hardware and software configuration in such a way that it execute as fast as possible all application's tasks. Each database can have different performance criteria. In some cases it requires optimization in terms of amount of data volumes, and other optimization in terms of response time. It is important to define the purpose of optimization for a given database. Ambitious objective defines the type of activities that should be carried out during the process of tuning. Databases can be optimized in terms of:

- Response time - the time from when you approve the request specific data until the data displayed on the screen. In systems, which has imposed strict time criteria, may be required to carry out certain tasks laterm to the time at which the server is less busy.
- Capacity - is the ratio of work and time required to perform the job. The work that has been done is often defined as the number of transactions.

Bandwidth = number of transactions / time

A side effect of the system configuration in terms of processing capacity, may be a worsening of other parameters such as: the number of supported users, waiting time, etc.

- Loading time - there is a group of systems that have a lower load time to load a certain amount of data. Time is limited and must be sufficient to load all the required data. Optimization of the system in terms of time will have a positive charge, also on the overall system performance.
- Fault tolerance - in some systems fault tolerance is very important in every aspect of the system. Any downtime in such a system, may have some disastrous consequences. Systems with high resistance to damage may require frequent checkpoints and regular backups. If fault tolerance is the highest priority, subsystem disks should use the appropriate level of RAID to protect against the failure of a single disk, and memory should support ECC technology, which allows for error correction.
- The number of supported users - such systems need to be tuned for effective handle large numbers of users simultaneously. When tuning the system in this regard sufficient memory plays a particular role.

2 Tuning methodology

Tuning the database can be divided into five stages.

1. System analysis

At this point, the system should be carefully investigated in the state, in which it is and pay attention to the problems occurring: bottlenecks or areas, where increased activity can be observed. During the analysis we should pay attention to the following system components:

- Application code - the problem may be not optimal application code.
- Oracle Server - the problem can be improperly defined SGA memory size or poorly chosen initialization parameters.
- Configuring the operating system - an operating system can not provide adequate amounts of resources to Oracle server.
- Hardware configuration - the problem may be bad disk configuration, which creates a bottleneck when accessing files.
- Network - the problem may also be overloaded network, causing delays.

2. Identification of the problem

If after analyzing the system turns out, that the problem occurs, you should carefully examine the cause of the problem. Valuable information can also be users feedback, which can help locate the source of the problem.

3. Designation of solutions and target

It should set a goal to achieved, eg: query optimization and increase the cache hit ratio.

4. Testing solutions

At this stage, you should test whether the implemented solution is consistent with expectations. The safest way to check the validity of the changes is to conduct tests in a special test environment, that emulates the load generated by users.

5. Analysis of results

Make sure that the targets have been achieved. If targets are not met, consider the point at which mistake has been made. Depending on the response, it is necessary to go back to the corresponding tuning phase.

Tuning Oracle

Scaling SGA (System Global Area)

In the Oracle instance, the data is stored in two types of memory: RAM and disk storage. Oracle tries to keep in SGA memory as much data as possible, to which access has been implemented recently. Except the data, Oracle caches in RAM the shared SQL and necessary data dictionary information. You can easily adjust the size of memory allocated to the Oracle instance, by changing the SGA_TARGET initialization parameter. Valid memory size allocated for the SGA depends on: the characteristics of the application, the number of users and the size of the transaction. If the amount of memory is insufficient, the application will have to perform time-consuming input-output operations. A good solution is, if the size of the allocated memory will be slightly larger than the minimum size required. It is important that the available memory is used as efficiently as possible, so you should allocate it in the right proportions to Oracle buffer and user's processes.

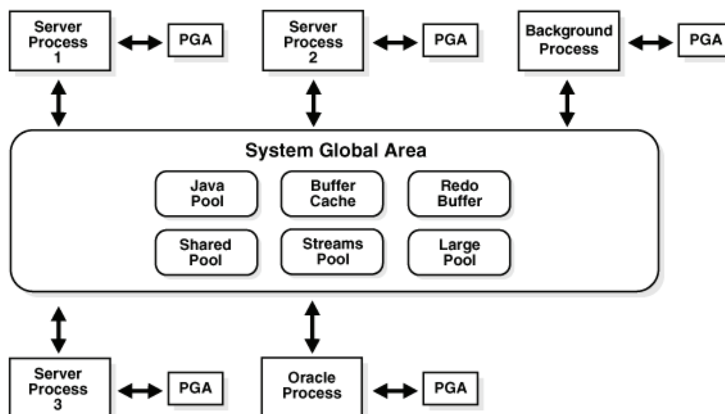

Figure 1. Oracle memory structure. Source: Oracle [8]

Tuning Shared Pool

During the process of optimizing the database, special attention should be paid to the proper configuration of shared area, because it directly affects the performance of the application. Shared area is part of the SGA that holds most of the necessary components to perform SQL queries and PL / SQL programs. Proper configuration of the shared area leads to a radical improvement in performance. However, the configuration performed improperly, can cause the following problems:

- fragmentation of the shared area;
- increasing the number of input-output operations, due to the lack of parsed version of SQL executed queries in the shared area;
- higher CPU usage caused unnecessary parsing SQL code.

A big impact on the waitng of the shared area is also not optimal SQL code, and in particular, no use of variables bound.

Shared area consists of the library buffer and dictionary buffer. Can not allocate or decrease memory of only one of these two areas. By increasing the total size of the shared area, the buffer size will automatically increasing cache the library and dictionary, in proportions determined by Oracle.

Library Cache

Library buffer stores the parsed code and an executable version of SQL and PL/SQL. Parsing is a very resource-intensive operation, so if the application needs to do the same SQL query several times, the presence of the query in the parsed form stored in memory, greatly reduce CPU usage, number of input and output, as well as memory usage. Execution plan and the path, will be stored in the library cache, before the query is executed for the first time. Subsequent calls to the same query will only have to go through a execution stage, which will bypass the step of parsing the SQL query. If the SQL statement is a SELECT, the last step is to download the data.

Libraries with limited buffer size, removes old SQL query, when there is no space to store new SQL queries. To reuse parsed SQL queries stored in the buffer, library must meet the following conditions:

- SQL code must be identical to the code stored in the buffer inquiries, including white space, the size of the letters and comments.
- All variables associated gotta be identical in name and type.
- Objects, that are referenced by the SQL query, must be the same objects that are referenced by the query stored in the buffer.
- Both queries must have the same plans for execution, set in the same mode of the optimizer.

This example shows three queries, that look exactly the same in terms of syntax, with the exception of the variable person_id.

```
SELECT * FROM persons WHERE person_id = 10;
SELECT * FROM persons WHERE person_id = 999;
SELECT * FROM persons WHERE person_id = 6666;
```

Each query will parsed and executed separately. Because of the similarity queries, separate parsing is a waste of resources. The solution to this problem is bound variables, which will mean that all three will have the same query execution plan,

and thus will only parsed the first request. The following example shows a query using a tird variable:

```
SELECT * FROM persons WHERE person_id =: var;
```

Using variables bound query performance dramatically raise.

Data Dictionary Cache

Data Dictionary Cache is a collection of tables and perspectives with information, that Oracle uses to describe all of the objects in the database. It kept information about both the physical and logical structures. Parsing queries are very frequent references to the dictionary cache, so access to it is often a sore point of applications, which can be the bottleneck. To change the size of this cache you should similarly change the size of the shared area. If the buffer library in the Oracle instance is well configured, it is highly likely that the dictionary cache is also properly tuned. Data dictionary cache capacity can check by using the statistics from the dynamic perspective V$ROWCACHE. This query describes how to obtain the number of hits to dictionary cache:

```
SELECT SUM(getmisses) as "Cache misses", SUM(gets) as "Requests",

 (100*(SUM(gets - getmisses) / SUM(gets)))as "Cache hit ratio"
FROM V$ROWCACHE;
```

The result of this query is as follows:

```
Cache misses              Requests               Cache hit ratio
337                       7051                   95.220536
```

Tabela 1. The most important column of V$ROWCACHE perspective

Kolumn	Description
GETS	The number of appeals to the buffer in general.
GETMISSES	Number of misses references to buffer.

Source: Whalen E. [15]

A good result is, if the buffer hit ratio is not less than 95%. However, the same Oracle recommends that the hit rate already at a level above 85% is adequate. To increase the hit rate, just increase the size of the shared area

Tuning Buffer Cache

The most important buffer in Oracle is data buffer. It occupies the most space in the SGA memory and is used during the execution of each query. When reading the data from the disk, Oracle copied the relevant data blocks to the data buffer. In the next step the data is supported. Modify the requested data can be carried out only if they are in the data buffer and may be made only by server process. The modified data blocks are written back to disk by the DBWR process.

Due to the very large number of references to the data buffer, it must have a sufficiently large size to ensure a high rate of hits. This factor, can be monitored by a dynamic perspective V%SYSSTAT.

Tabela 2. The most important column of V%SYSSTAT

Kolumn	Description
PHYSICAL READS	Number of misses references to buffer, which finished reading data from the disk.
DB BLOCK GETS	Number of ordinary buffer references - references CURRENT mode.
CONSISTENT GETS	The number of appeals to the buffer CONSISTENT mode.

Source: Whalen E. [15]

To get the total number of appeals, the appeal DB BLOCK GETS and CONSISTENT GETS shall be summed. Data from a dynamic perspective V%SYSSTAT:

```
SELECT name, value FROM   V$SYSSTAT
WHERE name IN ('db block gests', 'consistent gets', 'physical reads');
```

The result of this query is as follows:

```
NAME                                    VALUE
db block gests                          155
consistent gets                         5293
physical reads                          334
```

Hit Ratio should be calculated from the following formula:

```
Cache Hit Ratio = 1 - (physical reads / (db block gests + consistent gets))
Cache Hit Ratio = 1 - (334 / (155 + 5293)) = 0,938
```

In this example, the hit ratio of the data buffer is 93.8%. If the hit ratio is lower than 70% - 80%, it is recommended to increase the size of the data buffer, we can do it with the parameter DB_CACHE_SIZE. This ratio below 70%, can significantly contribute to the decline in performance of the entire Oracle instance. Despite the crucial role that buffer cache has, do not increase it at the expense of the size of shared area.

Data buffer starting from Oracle8, is divided into three parts, each of which has the same structure, but has a different function:

- Keep Pool Buffer – is intended for long-term storage blocks that have a high chance of re-use. Objects in it can be removed only by other objects to be stored in it. The size of this area can be determined by the parameter DB_KEEP_CACHE_SIZE.
- Recycle Pool Buffer – is intended for blocks that do not require long-term storage. The size of this buffer is defined bz parameter DB_RECYCLE_CACHE_SIZE.
- Default Pool Buffer – stores objects that do not hit the other two buffers. Its size can be determined by the following formula:

```
Default Pool Cache Size = DB_CACHE_SIZE - DB_KEEP_CACHE_SIZE - 
DB_RECYCLE_CACHE_SIZE
```

Automatic SGA memory management

Automatic memory management simplifies the configuration of the SGA and is the solution recommended by Oracle (the default is enabled).

To enable automatic SGA management, set the initialized parameter SGA_TARGET to a value corresponding to the total amount of memory to be allocated to the area of the SGA, and set the value of the parameter STATIS-

TICS_LEVEL TYPICAL or ALL. The amount of memory allocated to the SGA area should be equal to the size of the total available memory, minus the amount of memory required by the operating system and applications other than Oracle, running on the same platform and minus the memory area allocated to the PGA.

Oracle is actively monitoring the memory requirements for each of the areas of the SGA and dynamically allocates memory according to demand. Using the automatic management of the size of the SGA areas, you can still determine the minimum size for each of these areas. A few areas, such as: Keep Cache, Recycle cache, Streams cache and buffers with custom data block size, are still scaled manually.

You can disable automatic management by setting SGA_TARGET to zero. When off, the values of all the parameters that were set automatically, are assigned to the current values of these parameters.

Tuning PGA (Program Global Area)

Program Global Area is an area of memory, that contains data and control information about server process. It is not shared memory created by Oracle when starting the server process. This memory is read and written only by Oracle code. The total size of the PGA memory allocated to each server process is also referred to as the total PGA memory allocated by the instance. PGA elements:

- Private SQL area - Each session, which contains the SQL statement has a private SQL area. Anyone who uses the same SQL has its own private SQL area.
- Workspace - in the case of complex SQL queries, a large part of the PGA memory is designed to work for memory-absorb operators such as ORDER BY, GROUP BY, ROLLUP, join, create a bitmap.

Size of the work area can be controlled and tuned. Larger workspace can greatly improve the performance of individual operators, at the expense of greater memory utilization.

- Session memory - memory allocated to store session variables and other information related to the session. For a shared server, session memory is shared and not private.

Automatic PGA memory management

The optimal size of the work area is the size large enough to accommodate the input data and auxiliary memory structures, allocated by its associated SQL operator. The increase in the response time is caused by the lack of workspace to load the whole of input data, which is necessary to further progress.

The default and recommended an initial amount of memory allocated for the area is the size of PGA equal to 20% of the SGA, and the minimum size is 10MB.

At any time, the total amount of PGA memory available to work in all areas of the PGA memory area occupied by the sessions, and other elements of the system, is equal to the size of the parameter assigned to the PGA_AGGREGATE_TARGET initialization. To disable automatic PGA management area, set the PGA_AGGREGATE_TARGET initialization parameter to zero.

3 Ways to improve Oracle performance

Indexes

Indexes in Oracle provide quick access to the rows in the table, by storing the sorted values of specific columns and use those sorted for easy browsing of related rows.

Indexes in Oracle are optional data structure, after it is created, Oracle automatically maintains and uses them. All data modifications, such as adding new rows, updating rows, or deleting rows of the table are reflected on all indexes associated with the table, without any user intervention. Using the index requires a compromise between the fast search records, and free of updating and inserting. In databases, where the majority of the operation is to capture the data, a large number of indexes can dramatically reduce the access time to data. However, in databases, in which the operations are the most common inserts large amounts of data, and then updating and deleting indexes should be used with caution.

The main types of indexes in Oracle

- Unique and non-unique indexes. Unique index is an index based on a unique column of the table - no value in it can be repeated.

Non-unique index does not impose constraints on the uniqueness of the data. It applies in the case of the column in which the data is duplicated.

- Indexes simple and complex. Complex indexes are indexes containing two or more columns of the same table. The columns can be specified in any order and need not be consecutive columns of the table. Complex indexes can speed up search for relevant rows in SELECT query when WHERE clause column references are included in the complex index. In this case, it is also important the order of the columns. The column with the highest frequency of use, should be defined in the complex index first.
- Indexes based on functions. These are indexes that instead of columns taken from the table, keep the value calculated on the basis of the given function when creating the index. This feature can be a function of the arithmetic or PL/SQL. Index based on function may be B*tree or bitmap index. Indexes of this type are useful when the query in the WHERE clause is identical to the function used when creating the index. example:

```
SELECT c_id FROM customers
WHERE this_year_sales - last_year_sales > 0;

CREATE INDEX idx
ON customers (this_year_sales – last_year_sales);
```

Below is a collection of the most important rules, that should apply to the process of defining indexes:

- Indexes should only be used if you need access to no more than 15% of the records in the table.
- Relatively small tables shouldn't be indexed - in this case, the best solution is a full scan.
- Columns that are involved in join operations, should be indexed.

- For low selectivity columns of data (eg, data type "Yes / No") the most efficient are bitmap indexes, and for columns with high selectivity data, indexes, B*-tree.
- columns involved in the operations ORDER BY and GROUP BY or others, such as UNION or DISTINCT that require sorting should be indexed.
- Indexing columns that contain long strings is inefficient.
- The columns, which are often updated should not be indexed.
- Number of indexes should be as small as possible.

Indexed organized tables

Indexed organized tables are implemented internally as well as the basic variant of the B-tree. Unlike ordinary tables (organized as a heap), which store data in the form of disordered collection, data in indexed organized table are stored in the index structure, B *-tree and sorted by the primary key. Each leaf in the structure of the index holds the key and rest of non-key columns. Indexed organized tables have full functionality of the regular tables.

This way of writing tables provides the following benefits:

- Fast random access to data based on primary key, because you only need scan the index.
- Fast-band access to data based on the primary key because the rows are arranged according to the key.
- Less consumption of disk space, because the column which is a primary key is not duplicated, as it is a split record with index and table.

To create Indexed organized table, use the normal CREATE TABLE enriched by ORGANIZATION INDEX qualifier. It is also necessary to define the primary key for the table being created.

Clusters

A cluster is an optional data structure that improves read performance and, as well as the index, is transparent to the user and applications - cluster affects only way to store data. A cluster is a logical way to store related values together on the disk. Oracle reads the data by blocks, so keeping related data together - in one block, reduces the number of input-output operations need to download the related values. A single block of data contains only related rows.

A cluster consists of one or more related tables, that are stored in data blocks together, because they have common columns and are often used together in queries making their joints. Common columns of the tables are called the cluster key, this key is indexed and the index is called a cluster index. Each cluster index value indicates a block of data that contains only the rows with the same cluster key value.

The use of the cluster will be beneficial if the system contains two tables with related data and the queries that operate on them, often make their join. The use of cluster also allows you to search one of the tables, which is part of a cluster, but entails additional load on the system, associated with the presence of a single block of

data, records from both tables of the value of the cluster key, which has not place in case of standard tables.

In summary, in the form of a cluster should be stored tables, from which the data are collected in most cases, in a join. This will reduce the number of input-output operations. Do not use clusters for tables from which data are taken individually or carried out on them a lot of INSERT operations. Another contraindication of cluster tables are tables frequently scanned by full scan.

Hash clusters

Hash cluster works in a similar way as a normal cluster, but instead of pointing to the index key of the cluster, hash function is using for this purpose. Rows are stored based on the result of the hash function. To find the desired value line, must find the value of the hash function for the cluster key. The result is the number clearly designating the block that contains the requested data. In the case of simple cluster, to find the data block, it was necessary to perform several operations on the input-output on cluster index, and in the case of hash cluster, based on the key values, you should be able to locate the search block using only one input-output operation.

In a hash cluster, data is stored in a different order than in the ordinary cluster. In normal cluster data with the same key value are stored close to each other, and hash cluster are stored close to each other the same hash value.

Contraindications to the use of hash clusters are the same as for ordinary clusters. Not recommended for tables with frequent modified cluster key and for tables whose size increases rapidly.

In hash cluster you can put only one table. This advantage is the fact that access to the requested data is obtained by performing only one IO operation, while with B*-tree, the operation will be a few more. The best suited tables for using hash cluster, is one which key columns containing unique values and for which addressed queries, have a condition of equality for key columns.

Partitions

Partitioning is a mechanism that allows you to store tables or indexes of large size in the form of smaller parts - the partition. Smaller parts easier to manage, and access to them are faster and more efficient. From the point of view of the user and applications sharing a partitioned table is completely negligible. The only noticeable difference will increase query performance directed against partitioned tables, contains in the WHERE clause a condition corresponding to the applied partitioning scheme.

Each partition is stored in a separate segment, giving it independence. Partitions can be addressed individually, collectively or to all, treating them as one. Partitions are created on the basis of keys - key column sets partitions that contain data records.

An additional advantage of partitioning is the fact that in the event of failure of one of the drives do not lose the entire table, only a part of it, the rest of the data is

still available to users. You can also use partitioning for backup - you can only create a backup for a single partition.

Partitions can be divided into the following types:

- ranged partitions
- list partitions;
- hashed partitions;
- complex partition.

Each row of a partitioned table can only exist in one partition. Partitioning mechanism has been developed for large tables where the indexes can be inefficient. Oracle recommends the use of partitioning on all tables and indexes that are larger than 2GB. On lack of effectiveness of use of indexes on the table, affect the use of the aggregate queries that operate on a large part of the data in the table. This is where a good solution is to split the table into partitions.

Ranged partitions

Ranged partitioning involves broken down into defined ranges of values - data from individual ranges are stored in a separate partition. Fields for which the partitions are created is supplied together with the definition of the table. It is the most common type of partitioning and is often used with dates. The effectiveness of this method of partitioning is best when data is evenly distributed in all areas. In the case of uneven distribution of the data, this method of partitioning can not improve performance.

Example: The partitions are created according to the date of sale.

```
CREATE TABLE sales_data (
region VARCHAR2(10), sales_person INT, sales_amount NUMBER(10),
sales_date DATE)
PARTITION BY RANGE (sales_date) (
PARTITION jan2011 VALUES LESS THAN (TO_DATE('01/02/2011',
'DD/MM/YYYY')),
PARTITION feb2011 VALUES LESS THAN (TO_DATE('01/03/2011',
'DD/MM/YYYY')),
PARTITION mar2011 VALUES LESS THAN (TO_DATE('01/04/2011',
'DD/MM/YYYY')),
PARTITION apr2011 VALUES LESS THAN (TO_DATE('01/05/2011',
'DD/MM/YYYY')),
PARTITION may2011 VALUES LESS THAN (TO_DATE('01/06/2011',
'DD/MM/YYYY')),
PARTITION jun2011 VALUES LESS THAN (TO_DATE('01/07/2011',
'DD/MM/YYYY')),
PARTITION jan2011 VALUES LESS THAN (TO_DATE('01/08/2011',
'DD/MM/YYYY')),
PARTITION aug2011 VALUES LESS THAN (TO_DATE('01/09/2011',
'DD/MM/YYYY')),
PARTITION sep2011 VALUES LESS THAN (TO_DATE('01/10/2011',
'DD/MM/YYYY')),
PARTITION oct2011 VALUES LESS THAN (TO_DATE('01/11/2011',
'DD/MM/YYYY')),
PARTITION nov2011 VALUES LESS THAN (TO_DATE('01/12/2011',
'DD/MM/YYYY')),
PARTITION dec2011 VALUES LESS THAN (TO_DATE('01/01/2012',
'DD/MM/YYYY')),
PARTITION maxval VALUES LESS THAN (MAXVALUE));
```

Please note that in the absence of the last clause MAXVALUE partition, enter or modify data in such a way that they move out of range of the defined partitions will cause an error.

List partition

List partitioning uniquely specify the manner in which the rows are allocated to partitions. List partition are similar to the ranged partition, except that the key belongs to a set of discrete values. These values must be different to make it possible to assign a record to the appropriate partition:

```
CREATE TABLE sales_data (
region VARCHAR2(10), sales_person INT, sales_amount NUMBER(10),
sales_date DATE)
PARTITION BY LIST (region) (
PARTITION office1 VALUES (1, 2, 3, 4),
PARTITION office1 VALUES (5, 6),
PARTITION office1 VALUES (7, 8, 9, 10),
PARTITION other VALUES(DEFAULT));
```

Hash partition

In the hash partitions rows are assigned to a specific partition on the basis of an hash function, which operates on the column or several columns of the table. Together with the definition of the table you give number of available partition. Oracle ensures that the rows were assigned to the partition evenly:

```
CREATE TABLE sales_data (
region VARCHAR2(10), sales_person INT, sales_amount NUMBER(10),
sales_date DATE)
PARTITION BY HASH (sales_person)
PARTITIONS 4
STORE IN (sp1, sp2, sp3, sp4);
```

Complex partition

Complex partitions additionally improve the process of partitioning, allowing you to define two spit criteria. Complex partitioning is divided records into partitions, first by ranges, and then partitions are divided into hashed or list subpartitions. This method of partitioning works on very large tables, where just the ranged partition are not enough. Example:

```
CREATE TABLE sales_data (
region VARCHAR2(10), sales_person INT, sales_amount NUMBER(10),
sales_date DATE)
PARTITION BY RANGE (sales_date)
SUBPARTITION BY HASH (sales_person)
SUBPARTITIONS 4 (
PARTITION jan2011 VALUES LESS THAN (TO_DATE('01/02/2011',
'DD/MM/YYYY'))(
           SUBPARTITION jan2011_a TABLESPACE ts1,
           SUBPARTITION jan2011_b TABLESPACE ts2,
           SUBPARTITION jan2011_c TABLESPACE ts3,
           SUBPARTITION jan2011_d TABLESPACE ts4),
   PARTITION feb2011 VALUES LESS THAN (TO_DATE('01/03/2011',
'DD/MM/YYYY'))(
           SUBPARTITION feb2011_a TABLESPACE ts1,
           SUBPARTITION feb2011_b TABLESPACE ts2,
           SUBPARTITION feb2011_c TABLESPACE ts3,
```

```
            SUBPARTITION feb2011_d TABLESPACE ts4),
. . . . . .
PARTITION maxval VALUES LESS THAN(MAXVALUE));
```

Another example of use of the complex partition, this time range-list type, on table sales_data is a split first range by date, and then list based on regions:

```
CREATE TABLE sales_data (
region VARCHAR2(10), sales_person INT, sales_amount NUMBER(10),
sales_date DATE)
PARTITION BY RANGE (sales_date)
SUBPARTITION BY LIST (region)(
PARTITION jan2011 VALUES LESS THAN (TO_DATE('01/02/2011',
'DD/MM/YYYY'))(
            SUBPARTITION office1 VALUES (1, 2, 3, 4),
            SUBPARTITION office2 VALUES (5, 6),
            SUBPARTITION office3 VALUES (7, 8, 9, 10)),
PARTITION feb2011 VALUES LESS THAN (TO_DATE('01/03/2011',
'DD/MM/YYYY'))(
            SUBPARTITION office1 VALUES (1, 2, 3, 4),
            SUBPARTITION office2 VALUES (5, 6),
            SUBPARTITION office3 VALUES (7, 8, 9, 10)),
   . . . . . .
   PARTITION maxval VALUES LESS THAN(MAXVALUE));
```

Index partitioning

Just as they are partitioned tables, you can also to partition indexes. Indexes can be partitioned according to scheme of partitioned table (local indexes) or completely independent of partitioning table scheme (global indexes). Local partitioned indexes are easier to manage than other types of indexes. Each local index partition is associated with exactly one partition table. Oracle automatically stores the index partitions in sync with the partition table, so that each pair of table-index is independent.

References

1. Alapati S.R., 2005, Expert Oracle Database 10g Administration, Apress.
2. Barczak A., Florek J., Sydoruk T., 2007, Bazy danych, Wydawnictwo Akademii Podlaskiej, Siedlce.
3. Greenwald R., Stackowiak R., Stern J., 2004, Oracle Essentials: Oracle Database 10g, 3rd Edition, O'Reilly.
4. Lonley K., Bryla B., 2008,Oracle Database 10g Podręcznik administrator baz danych, Helion, Gliwice.
5. Taniar D., Rahayu J.W., 2002,A Taxonomy of Indexing Schemes for Parallel Database Systems. Distributed and Parallel Databases, Volume 12, Number 1, Kluwer Academic Publishers, pp. 73-106.
6. Liebeherr J., Omiecinski E., Akyildiz I.F., 1993, 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, pp. 510-522.
7. Helmer S., Moerkotte G., 2003, A performance study of four index structures for set-valued attributes of low cardinality. VLDB Journal, 12(3): pp. 244-261.
8. Bertino E. et al., 1997, Indexing Techniques for Advanced Database Systems. Kluwer Academic Publishers, Boston Dordrecht London.

9. Oracle: Oracle Database Administrator's Guide, 10g Release 2 (10.2), Dokumentacja techniczna, 2006.
10. Oracle: Oracle Database 10g Administration Workshop I, Dokumentacja techniczna, 2004.
11. Oracle: Oracle Database 10g Administration Workshop II, Dokumentacja techniczna, 2004.
12. Oracle: Oracle Database Concepts, 10g Release 2 (10.2), Dokumentacja techniczna, 2005.
13. Oracle: Oracle Database Data Warehousing Guide, 10g Release 2 (10.2), Dokumentacja techniczna, 2005.
14. Oracle: Oracle Database Performance Tuning Guide, 10g Release 2 (10.2), Dokumentacja techniczna, 2008.
15. Oracle: Oracle Database SQL Reference 10g Release 2 (10.2), Dokumentacja techniczna, 2005.
16. Oracle PL/SQL Database Code Library and Resources, [online] http://psoug.org/reference/library.html
17. Tow D., SQL. Optymalizacja, Helion, Gliwice 2004.
18. Urman S., Hardman R., McLaughlin M.: 2007, Oracle Database 10g. Programowanie w języku PL/SQL, Helion, Gliwice.
19. Whalen E., Schroeter M., 2003, Oracle Optymalizacja wydajności, Helion, Gliwice.