

An Oracle White Paper
January 2010

Partitioning for SAP ERP

A Guide for planning and setting up Oracle partitioning in SAP ERP systems

ORACLE

Scope of this document	1
Introduction.....	1
Prerequisites	6
Partitioning methods.....	8
Range partitioning	8
List partitioning	10
Hash partitioning.....	13
Indexing partitioned tables.....	14
Partitioning integration in SAP	16
General aspects of the SAP architecture	16
Suitable partitioning keys.....	18
Structure of table NRIV.....	20
Contents of table NRIV	22
Space calculation	23
Logical sub-partitioning.....	23
Partitioning administration: adding partitions.....	25
Scenario 1: No partitions are added.....	27
Scenario 2: New number ranges are added.....	27
Partitioning administration: Removing partitions	28
Online reorganization	31
Online segment shrink.....	31
Oracle solution: Oracle partition engine for SAP.....	31
Customer implementations	32

Example 1: Range partitioning-based NRIV approach.....	32
Customer request.....	32
Solution	32
Advantages.....	32
Example 2: List partitioning based on the SAP client.....	33
Customer request.....	33
Solution	33
Advantages.....	33
Example 3: List partitioning based on the WERKS column	34
Customer request.....	34
Solution	34
Advantages.....	35

Scope of this document

This document is intended to describe and explain the partitioning methods used for SAP ERP systems. The fundamental partitioning techniques are presented as is the concept of partitioning used within the SAP ERP system. It is a conceptual guideline for customers interested in using Oracle partitioning in SAP ERP systems.

Introduction

Partitioning is a well-known feature in Oracle databases and was introduced with database release 8.0. Since this time partitioning has been continuously improved.

In SAP systems, partitioning has been used in BW (Business Warehouse) systems, running on Oracle, since the very first BW systems were released. It is therefore a wellproven database technology, which was originally developed to handle very large databases, typically found in BW implementations.

Partitioning itself is not used by default in the SAP ERP system, for mainly historical reasons. The SAP ERP system is based on the architecture of the R/2 system, developed for host computers. At the time the first R/3 releases were shipped, partitioning technology was not available on UNIX based databases and was rarely used on even on host-based systems. The original SAP system was therefore designed on the basis of the database technology available 30 years ago.

Let us compare the common database sizes of today with those used 30 years ago. Initially SAP ERP systems contained only basic functionality and it was common for databases to be less than 100GB in size. Since then more and more functionality has been introduced to the R/3 system and database sizes have increased.

Today “normal” SAP installations commonly have database sizes of around 1TB whereas SAP customers, using special functionalities such as the SAP retail module, are running databases of 10TB and more.

Such large databases are linked with several challenges:

- Response times to user queries and batch processing must be guaranteed. This becomes more and more difficult because the tables are constantly growing in size.
- Old data must be archived, but archiving can have a negative impact on database performance. Deleting data from the database generates a huge transaction load, affecting normal data processing.
- Archiving very large tables can be a never-ending job. Once archiving is complete, follow-up tasks are needed either to rebuild fragmented indexes and/or to free up space in the tablespaces. These additional tasks occupy maintenance windows for the system, reducing overall availability.
- Processing larger tables uses up much more time and resources on the database as well as on the application servers. Because of the longer runtime, work processes take longer which means that more powerful application servers are needed.

Although hardware components have become more and more powerful, the classic solution of adding more CPUs and memory to the machine is no longer enough. The reason for this is very simple: modern systems don't suffer from slow CPUs or a lack of memory, instead they are limited by the performance of the storage system. Access to the data in storage is still limited by the comparably slow performance of the physical disks. For a better understanding, simply compare the progress made in CPUs and memory to that made in physical disks: CPUs have become 25 times faster in terms of

clock frequency and even more than 100 times faster in terms of processing speed. Comparable progress has been made in memory chips too. The available amount of memory has increased from 512 MB to more than 100GB and the access time has improved comparably.

Looking at the hard disk we can see only a significant improvement in disk capacity, whereas the average response time of the disks is still limited by physical rules, namely the rotation speed of the disk and the average position time for the disk head. Overall even with a disk rotating at 15,000 rpm, it can take up to 4ms for a specific disk sector to pass the disk head and with a 10,000 rpm disk, it can take 6ms. We can therefore see that the average response time for reading a single block from this disk will not fall below 5ms on average. This is also the average response time you can observe within the database for a single block access, e.g. an index lookup.

Compared with memory access this is incredibly slow. Memory access to a database block in the database cache is around 0.1ms to 0.2ms which is 10-50 times faster than access to the file system.

Processing very large tables is a task requiring access to a huge number of different database blocks, which cannot be completely cached in memory, neither in the storage cache, nor in the database cache. As a result, your system becomes IO-bound, but not CPU- or memory-bound (under normal circumstances). A simple thought experiment shows the effect of increased IO on a database query: We can assume two different extreme scenarios: In the first case, all blocks needed for a query are stored in the buffer cache and therefore a physical IO is not needed. This is indicated by the far left position of the graph. This results in the minimum response time for this query.

In the second case, we can assume that none of the blocks needed are in the memory, so we need a physical IO request for each database block involved, which results in the worst response time as can be seen on the right side of the graph.

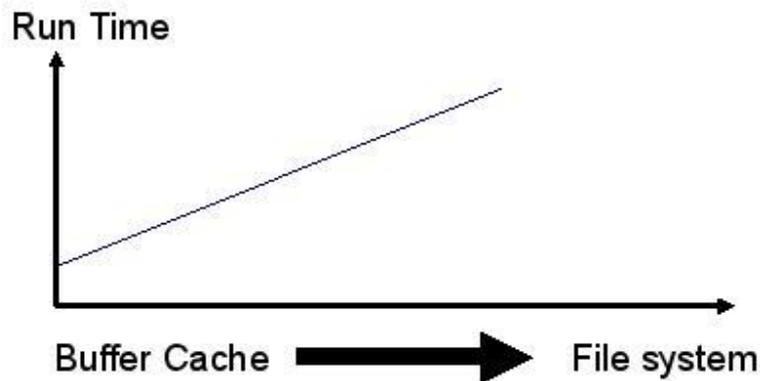


Diagram 1: IO dependency of query performance

Normally you will be somewhere between these two extreme scenarios. Moving your system to the right side (IO-dependent operation) will result in a worse run time and using the buffer cache more efficiently will result in better query and system performance.

Performing intensive delete operations on tables (archiving) will move each system to the right-hand side because new rows will be reinserted into old blocks, increasing the clustering factor of the table with regard to the business primary key.

In order to overcome the limitations of very large databases we have to move our systems to the left-hand side. Only in this situation will faster CPU and memory work best.

Therefore in order to handle these very large systems we need a solution which overcomes the current limitations and which allows us to use more "switches and knobs" in the database.

The Oracle partitioning technology for SAP ERP presented in this document overcomes the above limitations and if implemented correctly gives us many more advantages:

- Partitioned tables are almost reorganization-free and allow for efficient reuse of previously allocated space either at tablespace level or even disk level.
- Partitioning can significantly improve query performance. In real life implementations it can improve performance by between 2 and 30 times.
- Increased transparency with respect to generated and archived data. Errors in data archiving are immediately detected by non-empty partitions, holding that data which wasn't deleted. If more data than expected is created, you either notice a growth in partition size or in the number of generated partitions.
- Better system scalability by using enhanced Oracle database features, e.g. Parallel Query (PQ)
- Available hardware resources can be used by the database.

This document will show you the advantages of using Oracle partitioning in conjunction with the Oracle standard implementation. We will demonstrate how to set it up and administer it manually for some special cases. The SAP Partitioning Engine relies on the here demonstrated concept.

Prerequisites

To use the SAP partitioning engine, Oracle Database 10g or higher is required. For SAP you need depending on your SAP Basis Release at least SP26 (6.20), SP24 (6.40), SP21 (7.00), SP6 (7.01), SP11 (7.10), SP4 (7.11) in order to use this functionality.

A purely Oracle-based solution can also be used on older SAP releases/kernels. To achieve the best functionality, Oracle Database 10g is recommended for this combination, but Oracle Database 9i will also work with some restrictions: Merging old partitions will result in temporarily inaccessible tables due to unusable index partitions. These index partitions will be reorganized afterwards. If using Oracle Database 9i, I would recommend running the merge part in a maintenance window.

During the standard SAP/Oracle installation process, the partitioning option is always installed (see also OSS note [#859841 - Deinstallation of Oracle Partitioning software option](#)).

With SAP ERP, partitioning can be used with release 4.6C – service pack 48 and higher. From this version on, the SAP data dictionary is able to recognize partitioned tables and to handle them appropriately even during SAP transport.

Please note that there are still some limitations in SAP with regard to partition handling. Most of the functionality was developed for BW systems, but not for R/3. Partitioning is therefore also recognized. SAP standard transactions such as SE11, SE12, and SE14 are not suitable for creating appropriate partitioning within the system. They are mainly designed to accept customer-introduced partitioning.

Please refer to the following notes for further information on SAP and Oracle partitioning:

[#742243 - General table partitioning as of SAP BASIS 46C](#)

[#105047 - Support for Oracle functions in the SAP environment](#) (#34)

[#722188 - FAQ: Oracle partitioning](#)

[#1333328 Partitioning Engine for Oracle](#)

Principles of partitioning

The principle of partitioning is very simple: It means splitting a large table in smaller parts, called partitions, grouping and separating the rows of the table based on the content of one or more columns. Rows fulfilling the same criteria are placed in the same partition. This also allows you to find data in a table faster because the data will be grouped now instead of being distributed across the whole table. The Oracle documentation recommends considering partitioning tables if they are larger than 1GB. From experience, this threshold is too low for SAP systems. We recommend a table size of at least 10GB.

Using partitions changes the way that tables and segments are organized internally. For a simple table its name is also the name of the associated data object (segment name) Please keep in mind that a table is a kind of a logical definition for a data structure. The segment name is used in Oracle for the name of the storage object, which is displayed in the Oracle view `DBA_SEGMENTS`.

Using partitioning, the partitions are the storage objects (segments), which store only parts of the tables. Differentiating between logical table definition and physical storage definition offers a variety of options for partitioned tables:

- Because all partitions belong to the same table, which is still present in Oracle, the application itself doesn't need any modification (table transparency). For SAP systems there is therefore no need to change the ABAP/4 programming.
- All partitions are independent from each other. This allows partitions to be handled separately. Each partition can be created, dropped, merged or reorganized independently from all the others.
- This applies to partitioned indexes too. Single index partitions can also be reorganized independently, which allows you to reduce the time needed for index reorganizations dramatically because single partitions will be reorganized rather than the whole index.
- With partition pruning, the number of partitions which must be accessed during the execution can be dramatically reduced. Based on the values used within a where condition of an SQL statement, the database excludes from the execution path those partitions which do not contain requested rows.

Partitioning methods

There are three different partitioning methods available in Oracle:

- Range partitioning
- List partitioning
- Hash partitioning

Regardless of the partition type, partitions normally contain row subsets of the table and store these subsets in separate “segments” along with all the attributes owned by database segments:

- Partition-specific tablespace location and therefore dedicated storage
- Partition-specific storage definitions, e.g. extent size
- Online reorganization of single partitions as well as their local indexes

Partitions don't have table attributes, so you cannot create a constraint on a single partition, for instance. Constraint and indexes must be created for the whole table.

This document only discusses range and list partitioning in any depth. Hash partitioning is normally not suitable for SAP ERP systems. There are only a few exceptions to this rule and we do not list them here.

Range partitioning

Range partitioning is the preferred method for partitioning tables in SAP systems. With this partitioning type, a range of values is defined for each partition. A maximum value is specified (“Values less than”) for each partition, which forces Oracle to store all values which are smaller than the value specified in this partition. You can use numbers as well as character strings to specify ranges. The diagram below shows two examples of range partitioning on a table. The left side shows usage for a date based on a year and month definition whereas the example on the right shows range partitioning for number ranges. This is also a typical example for number ranges found in SAP systems.

One special partition is specified with the key word “MAXVALUE” for the values. This is a special partition which is used to store all values which do not fit in one of the predefined partitions. This special partition is needed to ensure that all insert operations can be successfully completed. Otherwise the insert will fail and an ORA-14400 will be reported.

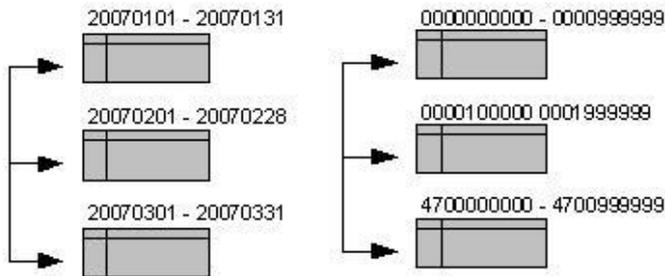


Diagram 2: Range partitioning for time ranges and numbers

The Oracle command for creating a table for the example on the left would be:

```
CREATE TABLE PART_RANGE
(MANDT VARCHAR2(3), DATE VARCHAR2(8))
PARTITION BY RANGE (DATE)
(PARTITION PART_RANGE_200701 VALUES LESS THAN ('20070201')
TABLESPACE PSAPPART1,
PARTITION PART_RANGE_200702 VALUES LESS THAN ('20070301')
TABLESPACE PSAPPART2,
PARTITION PART_RANGE_200703 VALUES LESS THAN ('20070401')
TABLESPACE PSAPPART3,
PARTITION PART_RANGE_MAXVALUE VALUES LESS THAN (MAXVALUE))
TABLESPACE PSAPUSER1D;
```

Please note that the high value for each partition specifies the next possible value for this column. So for January we specify February 1st as the high value because this is the next value allowed for this field. The rows stored in a single partition are therefore determined firstly by the high value specified for the partition itself and secondly by the high value of the partition defined below. If the partition has no parent, it will store all rows with a value less than the high value of the partition definition.

As a result the partition "PART_RANGE_200701" contains all rows which were created before February 1, 2007 (high value "20070201"). The next partition "PART_RANGE_200702" contains the rows created in February 2007 (high value "20070301", all rows generated between February 1 and February 29). Each partition is placed in its own tablespace. This is another option which can be used to place partitions in their own tablespace.

The last partition is a special one using the keyword "MAXVALUE". This partition is used if there is no matching partition for an inserted value. Specifying this partition ensures that an insert into the table will not fail if there is no matching partition. (In BW systems this special "MAXVALUE" partition is omitted, which can result in ORA-14400 errors.)

The next example shows the command for using number ranges instead of date values:

```
CREATE TABLE PART_RANGE
(MANDT VARCHAR2(3), VBELN VARCHAR2(10))
PARTITION BY RANGE (VBELN)
(PARTITION PART_RANGE_01 VALUES LESS THAN ('0001000000')
TABLESPACE PSAPPART1,
PARTITION PART_RANGE_02 VALUES LESS THAN ('0002000000')
TABLESPACE PSAPPART2,
PARTITION PART_RANGE_03 VALUES LESS THAN ('4701000000')
TABLESPACE PSAPPART3,
PARTITION PART_RANGE_MAXVALUE VALUES LESS THAN (MAXVALUE))
TABLESPACE PSAPUSER1D;
```

This is very similar to the first example but this time number ranges are used instead of date values. Please note that numbers are stored in the SAP system as character fields in the database. We therefore have to include the leading "0" (zero) characters at the start because Oracle will not compare the numerical values but the ASCII characters. This makes the leading zero characters important for correct partition mapping.

When using range partitioning it is important that the ranges increase continuously – otherwise you cannot create suitable partitioning ranges.

Range partitioning is the preferred method for partitioning tables in SAP.

List partitioning

List partitioning is very useful for tables which do not offer a suitable range of increasing numbers or characters. Its use is also recommended for specific customer applications. Normally list partitioning is suitable for tables with less discrete values for the designated column. Depending on the customer, there may be 28 (different plants) or up to 2300 (selling points) values suitable for the number of discrete values.

Especially if a customer is using random logic to assign identifiers, e.g. to subsidiary or plants, list partitioning is the only suitable way of benefitting from the advantages of partitioning. For example: There is an SAP module (SBO) which uses a hash algorithm to create unique values for each selling point. These values are completely unpredictable but always unique and consistent for a given shop. The values look like:

- "lhcBI2rD81JX00002WCHG0"
- "IQFjOrVmlZ3X00002WCHG0"
- "HpouaRPy2BRX00002WC1dG"

As of Oracle Database 10g, you can assign a list of values to a single partition. You can therefore balance partitions when there is a skew distribution of values. Please note that the values are compared in a case sensitive manner, so the values "SOUTH" or "south" would be treated as two different values. In an SAP system, uppercase notation is always used. There is no need to order the values alphabetically or by size. They can be listed in any order. New values can be added to existing partitions very easily.

List partitioning is very easy to administer because it is possible to assign new values to existing partitions. Unknown values (e.g. a new plant is established or bought) are covered by the "DEFAULT" partition where all values which cannot be assigned to the specified partitions are stored. It is possible to separate these new values later on in a separate partition.

The next diagram shows the principle behind the list partitioning method. The values of the unequally distributed lists of plants (column "WERKS" in the SAP table) are assigned to different table partitions in the example on the left, whereas different sales organizations for the column "VKORG" are used in the example on the right. Please note the upper case spelling for the values. Customers using a multi-client system can also use list partitioning to separate different clients in their system, using different tablespaces for each client.

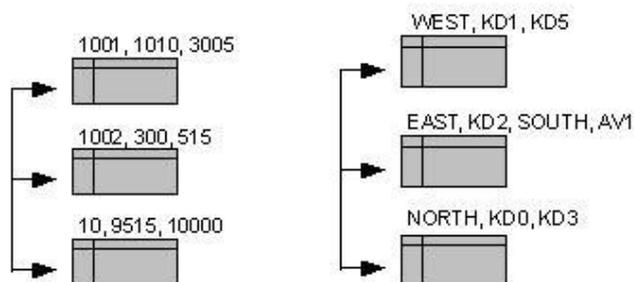


Diagram 3: List partitioning. Please note that the values are case-sensitive

The oracle command for creating a table for the left-hand example could be:

```
CREATE TABLE PART_LIST
(MANDT VARCHAR2(3), VKORG VARCHAR2(10))
PARTITION BY LIST (WERKS)
(PARTITION PART_LIST_01 VALUES ('WEST', 'KD1', 'KD5') TABLESPACE
PSAPLIST1,
PARTITION PART_LIST_02 VALUES ('EAST', 'KD2', 'South', 'AV1')
TABLESPACE PSAPLIST2,
PARTITION PART_LIST_03 VALUES ('NORTH', 'KD0', 'KD3') TABLESPACE
PSAPLIST1,
PARTITION PART_RANGE_MAXVALUE VALUES LESS THAN (DEFAULT))
TABLESPACE PSAPUSER1D;
```

The next example shows usage for a different column, “VKORG” storing sales organization identifiers. Once again any value can be stored here.

For the other example, using the column “VKORG” the SQL command is:

```
CREATE TABLE PART_LIST
(MANDT VARCHAR2(3), VKORG VARCHAR2(10))
PARTITION BY LIST (VKORG)
(PARTITION PART_LIST_01 VALUES ('1001', '1010') TABLESPACE
PSAPLIST1,
PARTITION PART_LIST_02 VALUES ('1002', '300', '515') TABLESPACE
PSAPLIST2,
PARTITION PART_LIST_03 VALUES ('10', '9515', '10000', '455')
TABLESPACE PSAPLIST1,
PARTITION PART_RANGE_MAXVALUE VALUES LESS THAN (DEFAULT))
TABLESPACE PSAPUSER1D;
```

Hash partitioning

If there is neither a suitable range of values for range partitioning nor a list of distinct values which can be used to build list partitioning, the hash partitioning method might be an option. Hash partitioning is therefore a method which uses some of the advantages of partitioning without you having to set up list or range partitioning.

Hash partitioning could be an option if a table needs to be split into many segments to reduce segment header or index block contention. It can also help you to reduce the segment sizes for reorganization. So in a few very special circumstances the hash partitioning option could be helpful. There are a few key reasons for using hash partitioning:

1. The table doesn't have either a suitable value range or a determined list of values.
2. You want to make use of Oracle features only available for partitioned objects.

To use hash partitioning you have to define a fixed number of partitions and at least one table column to which the hash algorithm is applied to distribute the rows over the predefined number of partitions. A hash value is created for each partition. It is calculated from the column values used for partitioning of the table. An appropriate hash value is calculated for each row, resulting in the target hash partition for this row. From outside, you cannot therefore predict the partition in which a particular row will be stored.

To set up smooth hash partitioning, take the following key points into consideration:

- For a good data distribution, define a sufficient number of partitions. As a starting point, partition sizes should be between 1GB and 10 GB for a single partition. Please remember that if the table is still growing, partition sizes will increase. For a good setup you should calculate the table size for its maximum expected size.
- Use a column with a significant number of distinct values. The number of distinct values in the column must be greater than the number of defined hash partitions. If one single column does not contain enough distinct values, use an additional second column to calculate the hash values in a combination of these columns.
- Make sure that the partitioning criteria are part of the most important SQL queries if you want to use the partition pruning feature of Oracle.

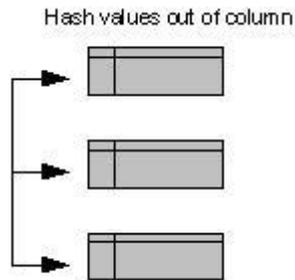


Diagram 4: Hash partitioning

To create a table with hash partitioning, the following command could be used:

```
CREATE TABLE PART_HASH  
(MANDT VARCHAR2(3), VARCHAR2(10)), VTWEG VARCHAR2(4))  
PARTITION BY HASH (VKORG, VTWEG)  
PARTITIONS 255  
STORE IN (PSAPHASH1, PSAPHASH2)
```

This will create a hash-partitioned table with 255 partitions. The rows of the tables are distributed equally among these partitions based on the values found in the columns VKORG and VTWEG.

Indexing partitioned tables

Oracle provides three different index types for creation on partitioned tables:

1. Local indexes
2. Global indexes
3. Global partitioned indexes

We will not discuss global partitioned indexes here because these indexes are not officially supported by SAP and are normally only used in special circumstances, which do not depend on the table partitioning itself. In fact global partitioned indexes can also be created on non-partitioned tables.

The most important decision therefore relates to when to use local indexes and when to use global indexes. First of all I want to discuss the main difference between these index types. An index is "Local" if the index is partitioned in the same way as the table itself. Exactly one index partition is therefore created for each table partition. This index partition only references to the

table data stored in this particular table partition. Therefore a local index consists of many independent index partitions and together they form the complete index. Local indexes reflect the table partition structure.

On the other hand, global indexes behave like indexes on a non-partitioned table. All rows of all partitions are indexed in a single index tree.

The biggest advantage of local indexes is their equally partitioned structure. This allows Oracle to use “partition pruning” (exclude all partitions from access which cannot contain the rows) during access.

The diagram below shows the structure of local and global indexes. The local index has exactly one partition for each table partition, forming multiple independent index trees, whereas the global index forms one single index tree for all rows.

There are some prerequisites for local indexes when these indexes are defined as “UNIQUE”. For a unique local index, the partition key must be part of the index definition.

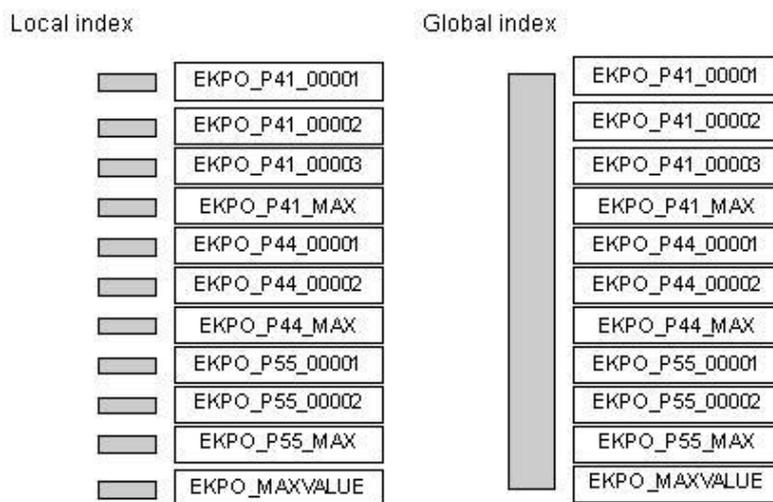


Diagram 5: Local and global indexes in Oracle

Deciding whether to create indexes as the local or global variant depends on the table structure, index definition and also the executed SQL statements. There are some generic rules which can be applied:

1. Always create a UNIQUE index as local if possible. This also implies that a table should always be partitioned by a field which is part of the unique index blocks.
2. All indexes which contain the partitioning key should be local.
3. Check the SQL statements for this table in the Shared Cursor Cache of the database. To use partition pruning, the where clause of the SQL statement need simply contain information about the partitioning key (contains columns of...). The LOCAL index itself does not need to include the partitioning key (only if the index is defined as UNIQUE must the partitioning key be part of the index definition).
4. If you cannot guarantee that partitioning will work for a specific index by having the information as part of the where clause, create a global non-partitioned index instead. This is a safe approach because global non-partitioned indexes work on the partitioned tables in the same way as on a non-partitioned table.

Partitioning integration in SAP

The real challenge for integrating Oracle partitioning into an SAP system is to find suitable partitioning keys. In the past most of the approaches to establishing partitioning in SAP ERP failed because of the wrong integration concept. With this new concept, Oracle partitioning can be smoothly integrated into the SAP application. Although a technical feature, partitioning technology works very closely with the application and its design.

To understand the basic idea behind this concept, note the time-dependent architecture of an SAP ERP system, and this time dependency is reflected in the partitioning concept.

General aspects of the SAP architecture

First of all we have to change our understanding of database organization. Instead of looking at the database as an unstructured data container, simply storing rows in tables without any relationship, change your perspective to an application point of view. SAP is a flow of data, gradually passing the different system components in well-defined time frames as they are being processed. For example, orders are received and processed, triggering material flow and bills are created and paid. This takes a certain amount of time which is typical for each customer. Normally such a workflow takes between 6 and 12 weeks. During processing we will therefore find a trace of each document through the SAP modules (MM, FI, SD, PP) as well as a distinct working status for each item.

Once the rows are completely processed they are only needed for month and year closings. They remain in the database for an additional time until they are archived.

To fully understand the partitioning concept you first have to change the way you view the database structure: Data is not stored without order as in an anonymous storage container, there is a hidden order structure in the data, defined by the SAP application design and by the customer's business process. So instead of looking at one big system, we can classify an SAP database as a group of time slides, each representing a set of processing steps (in terms of time) within the customer's business process.

Changing our understanding of the database organization to a time-structured database also brings us to one important conclusion with regard to query performance: Think about the time processing of the data: Data either entered or processed on a certain day will be stored very closely within the tables because during the insert operation, rows will be inserted consecutively into free blocks. (This is very similar to ASSM and non-ASSM tablespaces, using a freelist). In a best-case scenario, the number of table blocks used is minimized. In reality for good SAP system performance it is very important for the number of table blocks used to store the data to be as low as possible. This increases the probability of a cache hit and will move the database performance from IO-dependent performance to cache-dependent performance (please refer back to the explanation for this on page 4).

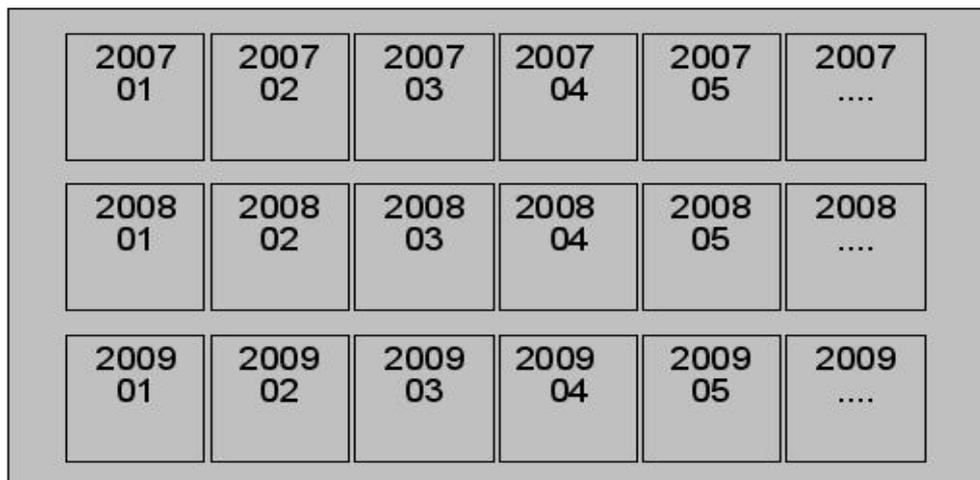


Diagram 6: Time slide organization of the database

Suitable partitioning keys

Based on the above considerations, we can now formulate the prerequisites needed to set up smooth partitioning:

- Adapted to SAP system architecture and time organization
- Minimizing the number of blocks used for the data to maximize query performance
- Support of archiving architecture and procedures
- Must be administrable within the system
- Avoiding or minimizing index reorganizations on these tables, if possible
- Avoiding even table reorganizations, normally used to free up space
- Increasing query and system performance by supporting queries on the table
- Stabilizing system and query performance, even with ever-increasing data
- Increasing system availability by reducing planned downtimes

The only partitioning method which fulfills all the above requirements is the range partitioning method. But if using range partitioning we first have to find adequate criteria for the tables which come into consideration. The preferred criteria for range partitioning are time-dependent criteria. Our first approach could be to look for a column in the SAP tables which represents a date value. Unfortunately only very few tables contain such a column (e.g. BUDAT, CREDAT). Even if such a column is present, we can't necessarily use it because these columns are mostly not indexed and because these columns are not specified when accessing data. Using these columns will not therefore fulfill the requirements formulated in points 3, 5, 7, and 8. So we have to look for another criteria. We will not find it in the table itself. We need an approach to overcome this situation. If we can't make use of direct time information, is there an indirect time key we can use?

We can make use of a feature of systems used to handle financial transactions. To comply with legal requirements, a couple of document types must be numbered in ascending order. To ensure this SAP uses a single table containing the list of available numbers. This table "NRIV" is the key to time dependency. The table NRIV is the number generator used in an SAP system, comparable to sequences available in an Oracle database. All tables using a specific document number, e.g. material number, purchase order, selling document number, will receive these values from this table. These numbers are always part of the primary table key. Examples of this are (table name – column name is always listed):

- Table MSEG - Column MBLNR
- Table VBRP - Column VBELN
- Table EKPO - Column EBELN

The design of this number generator is very simple: There is at least one entry (row) in the table NRIV for each document type. If a process needs a number, it runs a “Select for update” command on one row, picks up the current value, increases it by the numbers used and writes the new increased value back to NRIV. So the next process will find a new number, which is higher than the ones used before.

If we estimate how many numbers were generated within a defined time frame in the past, we can predict the numbers which will be generated e.g. in the next four weeks. In other words, table NRIV represents the link to time that we need for all tables which obtain their key from this table. Almost all tables within SAP using ascending number ranges are “fed” by this table. The table NRIV is a very substantial structure used throughout all modules in SAP. The diagrams below show the described time dependency. If we know that the number 2,000,000 was used from February 2009, we can predict that:

1. This number will not be reused anymore (there are actually some exceptions to this, but in general we can make this assumption)
2. The next number will be higher than the existing ones
3. Assuming that 500,000 numbers are used per month, we will assign number 3,000,000 in April, namely two months later.
4. Even if there is no linear behavior for the number assignments, a customer knows how many numbers are used for each month

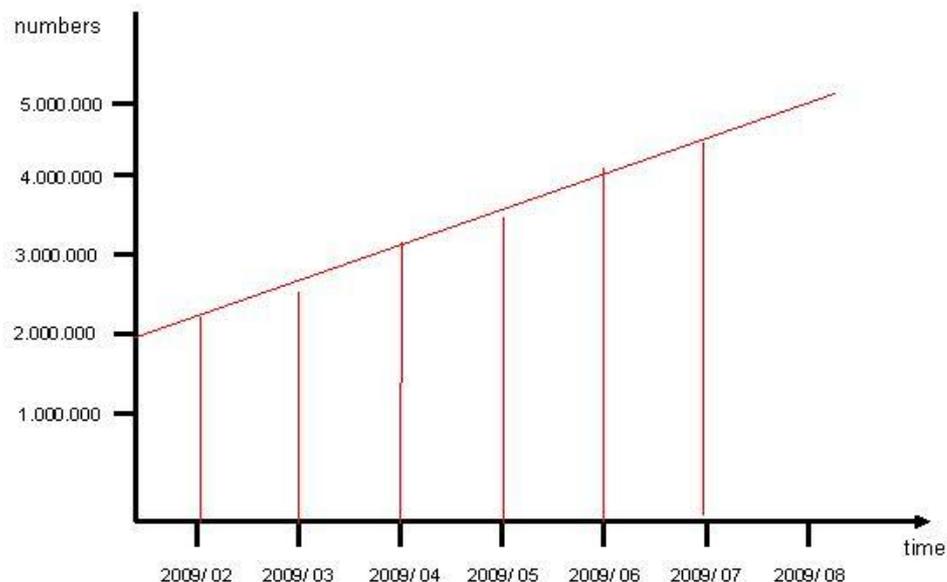


Diagram 7: Time dependency of table NRIV

This is exactly the information we need to define time-dependent range partitioning for a table. What is more, using table NRIV we can establish a link to the SAP application to implement highly customized partitioning based on a set of rules found in the table NRIV.

Structure of table NRIV

Once we have identified NRIV as a possible source for determining the number ranges for partitioning, we have to analyze the structure of the number ranges in more detail. In SAP you can use the transaction “SNRO” to do this. At database level we could directly analyze the table contents. Please note, that the table NRIV is client-dependent so you must be connected to the productive client to view the correct entries.

The table NRIV has the following structure:

NAME	NULL?	TYPE
CLIENT	NOT NULL	VARCHAR2(3)
OBJECT	NOT NULL	VARCHAR2(10)
SUBOBJECT	NOT NULL	VARCHAR2(6)
NRRANGENR	NOT NULL	VARCHAR2(2)
TOYEAR	NOT NULL	VARCHAR2(4)
FROMNUMBER	NOT NULL	VARCHAR2(20)
TONUMBER	NOT NULL	VARCHAR2(20)
NRLEVEL	NOT NULL	VARCHAR2(20)
EXTERNIND	NOT NULL	VARCHAR2(1)

The table itself is not very large. The number of entries depends on the number of modules used by the customer and customization. Normally it contains not more than a few thousand entries. What now is the meaning of these columns and how can we make use of the information stored in NRIV?

- **CLIENT:** Is the SAP client the entry is used for. So if a customer uses a multi-client system, this would result into composite partitioning because the client field must be included in partitioning
- **OBJECT:** Is the name of the number range object used within SAP. This name is used in all modules to access this number range. There are a lot of different objects, commons one are e.g. MATBELEG, EINKBELEG, RV_BELEG, RE_BELEG. For partitioning we have now to link the object name with the table name we want to partition. Unfortunately the names used in NRIV for these objects are not identical to the column or table names which make use

of them. There is also no mapping table in the SAP system we can use to bring them together, The only way to get the object name is to ask a responsible application.

- **SUBOBJECT:** For a few objects in NRIV there are sub-objects which are used to divide the object into smaller parts. Normally we can ignore entries here.
- **NRRANGENR:** This is a unique identifier used to create multiple number ranges for a specific object. You may be familiar with situations when during heavy processing a lot of enqueue waits can be observed on the table NRIV, slowing down processing. Multiple number ranges can be defined to reduce these enqueue waits, to reduce the contention and to speed up processing.
- **TOYEAR:** Some of the number ranges are year-dependent. The uniqueness of a key need only be valid within one financial year. The same numbers can therefore be reused in different financial years. If a number range is not year-dependent, values must be unique within a number range. There are three possible values you can find here:
 - “0000” The number range is not year-dependent.
 - “2009” A distinct year means that the values are valid for this year only.
 - “9999” The number range is year-dependent, but numbers are created until all available numbers have been used. The system then goes back to the beginning of the number range. The customer must ensure that there are enough numbers for one financial year.
- **FROMNUMBER:** This defines the starting point (minimum number) for the particular number range.
- **TONUMBER:** This defines the end point (maximum number) for the particular number range.
- **NRLEVEL:** This is the current level of the number usage for the particular number range. This is therefore also one good possible way of obtaining information about the current level for a number range.
- **EXTERNIND:** If checked (“X”), this indicates a number range as an external one. These external ranges are used by foreign applications and will not be used by SAP. Contrary to the SAP internal number ranges, external ones can also contain character signs.

Contents of table NRIV

Now we have discussed the table structure of NRIV, let's look in more detail at an example entry for the NRIV object "EINKBELEG", which is used e.g. in the tables "EKPO", "EKBE", "EKKO".

CLI	OBJECT	SUBOBJ	NR	TOYE	FROMNUMBER	TONUMBER	NRLEVEL
001	EINKBELEG		41	0000	4100000000	4199999999	4100512869
001	EINKBELEG		44	0000	4400000000	4499999999	4415697351
001	EINKBELEG		55	0000	5900000000	5999999999	5900000245

Client "001" has three different number ranges; "41", "44" and "55". Each of these number ranges defines an overall number range of about 100 million possible records. The number range starts at 4100000000, ends at 4199999999 and its current maximum used number is 4100512869. The other two number ranges can be interpreted accordingly. Please note there is no dependency between the naming of the number ranges (41, 44, 59) and the defined number range. The customer is completely free to use any naming but I strongly recommend only using numbers and characters.

What is interesting is the different usage level of the three number ranges. Number range "44" is used the most, whereas number range "41" is used significantly less and only 245 numbers were used in number range 55. The usage of the number ranges is different for each system and depends on system customization.

Now we have identified the valid number ranges used in the table NRIV for the object type "EINKBELEG" used e.g. in the tables EKPO, EKKO, EKBE, EKBN, EKBA. But where can we find the numbers within the table definition? All these tables have a column named "EBELN", which is defined as VARCHAR2(10). This corresponds to the length of the entries found in the field FROMNUMBER and TONUMBER. So we can expect to find numbers in the ranges between 4100000000 and 4100512869, 4400000000 and 4415697351, 5900000000 and 5900000245.

Space calculation

An analysis of the table NRIV shows that unfortunately there is not only one number range for the NRIV object EINKBELEG, but there are three different ranges which are handled independently of each other. It's obvious that we must have partition definitions for all these defined ranges. The easiest way to achieve this would be to create partitions for all the possible number ranges in advance. Although Oracle Database 10g can now handle 1024k-1 partitions for a single table, it doesn't simplify system administration. To get an overview of the number of partitions needed and the space requirements when creating all partitions, let's look at the following table.

TABLE 1: NUMBER OF PARTITIONS AND SPACE REQUIREMENTS

COLUMN LENGTH	POSSIBLE # OF ENTRIES	# OF PARTITIONS FOR A PARTITION RANGE OF 1.000.000	# OF PARTITIONS FOR A PARTITION RANGE OF 100.000	RESULTING TABLE SIZE IN GB
10	10,000,000,000	10,000	100,000	6.1GB
12	1000,000,000,000	1000,000	10,000,000	610GB
16	10,000,000,000,000,000	10,000,000,000	100,000,000,000	6103,515 GB

Because tables also exist in SAP which use larger data fields (12 or 16 digits, up to 22 digits are possible), we can see that this approach will create a huge number of partitions. Using twelve digits, the number of partitions needed will also exceed the capabilities of the Oracle database with regard to the number of possible partitions for a single table. In the last column you can see the calculated space needed for the empty partitions (64k each). This approach is not therefore reasonable simply because of the tremendous number of partitions needed and the resulting space allocated for the empty partitions.

So we need a different approach to handle the table partitions needed.

Logical sub-partitioning

The best way of handling the partitions needed would be to create the partitions "On demand", basically by having a number of spare partitions in place to store the data that will be created e.g. in the next four weeks. These partitions must be created as required e.g. by a regular job. We can define now the prerequisites for creating partitions on demand:

- We must know the current level of numbers used and partitions defined
- We need information about the partition ranges which may possibly be used within the table
- We must be capable of strictly separating the partitions ranges from one another to avoid rows being placed in the wrong partitions
- We must be capable of creating partitions independently for each number range to guarantee maintenance (adding and removing partitions) for each partition range.

Based on the structure and contents of the table NRIV it will be possible for the above criteria to be fulfilled using so-called “Logical sub-partitioning” for the table and the number ranges.

“Logical sub-partitioning” is completely different from the Oracle sub-partitioning feature, which can also be used as partitioning method. It is not a physical partitioning method but more a means of logical table organization for handling different, independent number ranges within one table. The basics of this idea are very simple because we will find all information needed in the table NRIV.

- FROMNUMBER gives us the smallest number possible for a particular number range.
- TONUMBER gives us the maximum possible number for a particular number range.
- NRLEVEL provides information about the current usage for this particular number range.

Based on this information we can now start to organize the table into

“Logical sub-partitions” where each “Logical sub-partition” contains exactly the range of one existing number range defined by FROMNUMBER and TONUMBER.

To distinguish the resulting partitions and to allow assignment of the partitions to the SAP defined number ranges by their name we can use the following naming convention for partition names:

`<Table name>_P<NR>_nnnnn`

where:

- `Table name` => name of the partitioned table
- `_P` => fixed prefix for partitions
- `NR` => name of the SAP number range
- `nnnnn` => five digits to uniquely identify partitions within one number range

So for our entries for the NRIV object “EINKBELEG” we can now create three logical sub-partitions for those tables using the number ranges for the NRIV object: Following the above naming convention, we obtain partition names such as:

- EKPO_P41_00001, EKPO_P41_00002, EKPO_P41_00003, EKPO_P41_00....
- EKPO_P44_00001, EKPO_P44_00002, EKPO_P44_00003, EKPO_P44_00....
- EKPO_P55_00001, EKPO_P55_00002, EKPO_P55_00003, EKPO_P55_00....

Once the basic principle is clear, we now have to verify whether “Logical sub-partitioning” fulfills the other criteria for a good partitioning as well.

Partitioning administration: adding partitions

Because we are planning to create partitions on demand we have to establish a method for creating new partitions for all number ranges in the table, if needed. In Oracle, there are two commands for adding partitions to a table:

- ALTER TABLE ADD PARTITION
- ALTER TABLE SPLIT PARTITION

The first command can only be used if you are adding partitions to a table at the end. In other words, the newly created partition must be the very last one in the table. This would work for just one number range in our example: number range 55, which is the very last one. But it will not be suitable for number ranges 41 and 44. So we have to use the “ALTER TABLE SPLIT PARTITION” command. This command has one characteristic: If you split a partition and at least one of the resulting partitions is empty, Oracle can make use of the “Fast partition split” feature. Using “Fast partition split” results in a real online operation, so none of the existing indexes or partitions will be invalidated. To make use of this feature we have to ensure that there is always one empty partition we can use for the split command. To have such a partition available at all times, we add one extension to our partitioning scheme: Each number range gets partitions, which finalize the upper limit of a number range. This ensures that this partition remains empty until the whole number range is in use. At this point, a further partition split is no longer necessary because all possible partitions have been created.

To ensure that all rows can be stored in the partitioned table, we also have to set up a special partition with the keyword “MAXVALUE”. This partition will be used to store those rows which do not fit into any of the other partitions and to avoid the Oracle error message “ORA-14400: inserted partition key does not map to any partition”. This is important if we want to ensure that SAP will never fail on an insert operation.

- Create a logical sub-partitioning range for each number range found for this table in NRIV.
- Create the partitions needed for each number range to store the desired number of rows.
- Add spare partitions to each number range to store newly created rows.
- Finalize each number range with a special partition, which uses the maximum number range definition (TONUMBER from NRIV) as the partition definition.
- Define one special partition to store any unexpected value.

The resulting create table command, for a partition range with one million rows for each partition, could be as follows:

```
CREATE TABLE EKPO (MANDT VARCHAR2(3), EBELN VARCHAR2(10), . . . .)
PARTITION BY RANGE (EBELN) (
PARTITION EKPO_P41_00001      VALUES LESS THAN ('4101000000'),
PARTITION EKPO_P41_00002      VALUES LESS THAN ('4102000000'),
PARTITION EKPO_P41_00003      VALUES LESS THAN ('4103000000'),
PARTITION EKPO_P41_MAX        VALUES LESS THAN ('4200000000'),
PARTITION EKPO_P44_00001      VALUES LESS THAN ('4401000000'),
PARTITION EKPO_P44_00002      VALUES LESS THAN ('4402000000'),
PARTITION EKPO_P44_00003      VALUES LESS THAN ('4403000000'),
PARTITION EKPO_P44_00004      VALUES LESS THAN ('4404000000'),
PARTITION EKPO_P44_MAX        VALUES LESS THAN ('4500000000'),
PARTITION EKPO_P55_00001      VALUES LESS THAN ('5901000000'),
PARTITION EKPO_P55_MAX        VALUES LESS THAN ('6000000000'),
PARTITION EKPO_MAX            VALUES LESS THAN (MAXVALUE))
TABLESPACE ...
```

This design for the partitioning layout ensures that no error (ORA-14400) can occur under any circumstances. A closer examination shows the possible results if this part of the partitioning administration is not working.

The resulting partitioning layout with the logical sub-partitioning is shown in the diagram below:

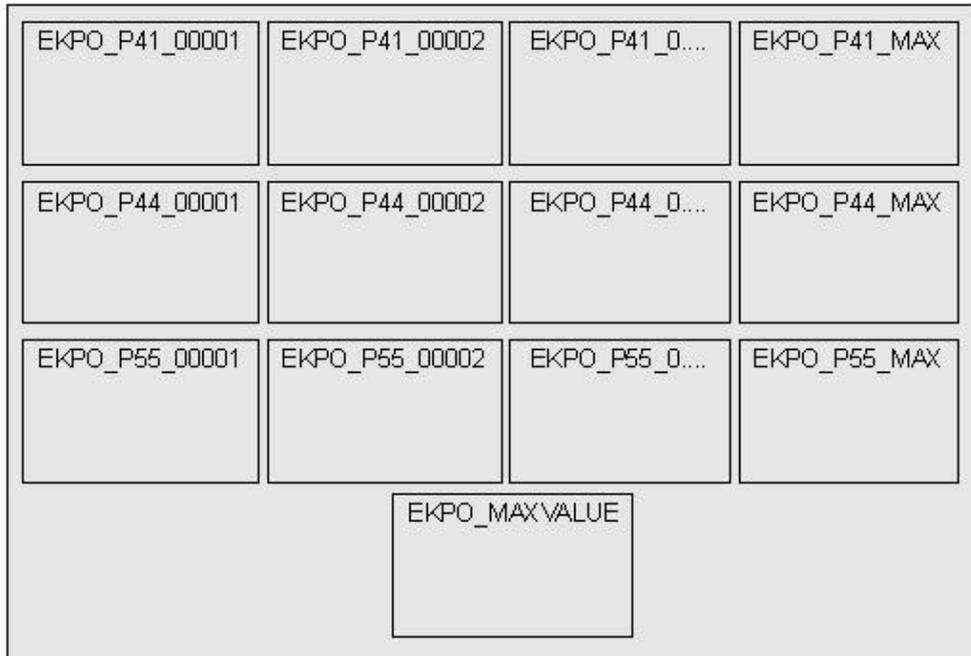


Diagram 8: Partitioning layout for SAP with logical sub-partitioning

Scenario 1: No partitions are added

What can happen in your system when no more partitions are added to the table? The outcome is very simple: All rows will be created in the very last partition of the logical sub-partition that in our example would be either EKPO_P41_MAX or EKPO_P44_MAX or EKPO_P55_MAX. Once all rows are present in only one partition, the table will behave as if it is not partitioned. It will be always possible to split this partition once again to establish a proper partitioning layout once again. This would also be possible online because we can chose a split value which ensures that one of the partitions created during the split operation is empty. This allows Oracle to perform a "Fast partitioning split". Once this split is executed, normal table maintenance can proceed.

Scenario 2: New number ranges are added

If new number ranges are introduced by the application or if the existing number ranges are split, this modification will result in unexpected rows in some partitions or in some partitions containing rows which they should not. The storage location of these rows depends on the new number range definition:

- If the new number range is introduced at the beginning of the table, all new rows will be inserted in the table partition with the lowest defined maximum value. In our example this would be partition EKPO_P41_00001.
- If the new number range is introduced by splitting an existing number range, the exact definition determines where the rows will be stored: If the new number range starts beyond the actual maximum value of the existing partitions of the current number range, we will see the rows in the MAX partition of this number range, e.g. EKPO_P41_MAX. If it was defined below the actual maximum value of the existing partitions, the rows will first fill up these existing partitions before using the MAX partition.
- If the new number range is introduced after the last defined number range, the rows are stored in the MAXVALUE partition because they do not fit into one of the existing partitions.

All of the above situations can be solved by a simple partition split command which separates the rows and corrects the partitioning layout of the table. By way of conclusion:

If there no new partitions are added to a table, in the worst-case scenario the table will behave as if it is not partitioned. Partitioning can be re-established on this table without the table having to be reorganized.

Partitioning administration: Removing partitions

Although adding new partitions is very important for maintaining the advantages of a partitioned table, we also have to think about how to reuse the space freed up by archiving (deleting) operations on the table. While space from a delete operation is normally reused within the table itself (with all discussed disadvantages), partitioning will avoid the recirculation of previously used blocks. Therefore if old, empty partitions are not removed, the table will continue to grow in size.

When archiving data within SAP, the archiving process will first select the data based on the criteria the customer has defined for the archiving objects (tables). Once the rows to archive are identified, SAP deletes the rows based on the primary table key. Because our partition design is based on the NRIV numbering and this numbering is always part of the primary key, we can expect the oldest data to be deleted from the table. Remember now the time dependency of the table NRIV: The oldest table data will always be those rows with the lowest NRIV number and will therefore be stored in the lowest numbered partitions. This behavior is shown in the diagram below:

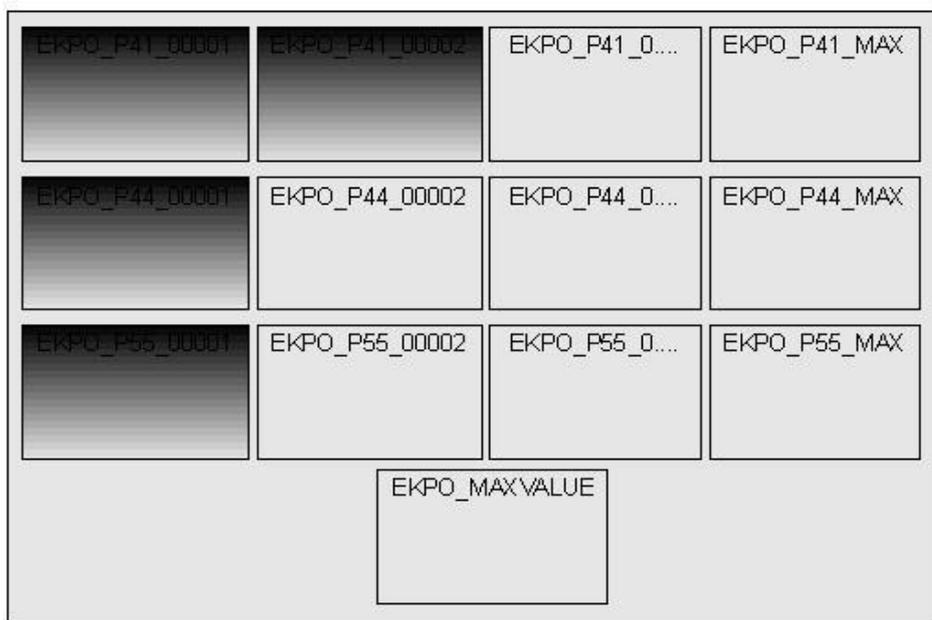


Diagram 9: SAP archiving: rows in the oldest partitions are deleted

Once the archiving process is complete for a specific time period, we can now start to free up the space allocated. To free up the space, which is hidden in the almost empty partitions, we have to either drop the partition or to merge two partitions. The partitions can only be dropped if it does not contain any rows. But in SAP we can expect not to be able to archive between 0.1% and 10% of rows due to failed processing, archiving definitions or other reasons. So under normal circumstances even after a completed archiving cycle, the partitions will not be completely empty.

Instead of using a drop command we therefore have to use the merge operation. A merge operation is always performed on two adjacent partitions. The rows of these two partitions are copied to a new partition, allocating only the space really needed by the copied rows. Once the copying process is complete, the space allocated from the original partitions is released to the tablespace and can be reused for new created partitions. The merge command will always result in a partition and is limited by the maximum value definition of the partition with the larger maximum value definition. The user is free to name the resulting partition.

Please note that all the space allocated by new created partitions will be initialized first, so these blocks can be treated as new blocks.

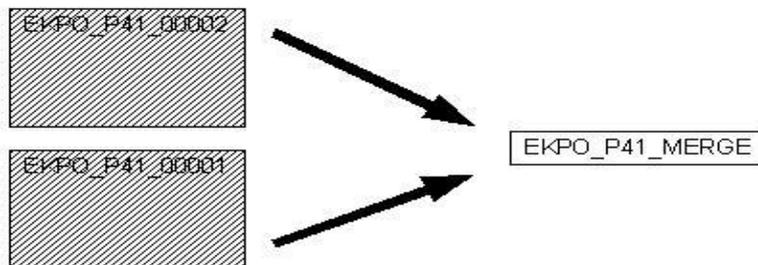


Diagram 10: Merge operations on partitions

This is a very efficient way of freeing up space without having to reorganize the table. This merge operation can be executed online (even while SAP is running). All indexes on the table are automatically maintained by Oracle during the merge operation if the clause “UPDATE INDEXES” is included.

The command to merge the partitions in the example is:

```
ALTER TABLE EKPO MERGE PARTITIONS  
(EKPO_P41_00001, EKPO_P41_00002)  
INTO PARTITION  
(EKPO_P41_MERGE) UPDATE INDEXES;
```

The duration of merge operations depends on two factors:

1. Partition size
2. Number of rows to copy

During the merge process Oracle performs a full table scan within the partitions to be merged. All rows found during this scan are then copied to the new target partition. So if the partitions are very large the scan takes a long time which means that if using very large partitions the scan part can become the time-consuming part of the operation.

If one or both of the initial partitions contain a large number of rows, the copy process is the time-consuming factor. During the current implementations partition sizes of between 1 and 5GB and row numbers of no more than 100,000 within a single partition are showing acceptable results. For this configuration, a typical merge process takes less than 60 seconds to complete.

Although all merge operations can be executed online (neither DML nor select operations will be blocked or interrupted), we would recommend running these operations outside of the peak load in a quieter period. During the merge operation local indexes will be invalidated. This temporary invalidation will not force any errors on the database, but processes accessing the table by one of these invalidated indexes will switch to a full partition scan. This can result in significant performance loss for some queries.

There are other methods for reorganizing a single partition:

- Using Oracle Online redefinition (dbms_redefinition package)
- Oracle online segment shrink

Online reorganization

You can also reorganize a single partition with the Oracle dbms_redefinition package, performing an online reorganization. Instead of reorganizing a whole table, only the rows in one single partition are reorganized. Although this sounds very attractive, there are some serious disadvantages to this method.

Reorganization of a single partition uses the "Exchange partition" command which can be applied to single partitions. To reorganize one single partition, you therefore have to first create a table with the same structure as your partitioned table. In the next step, all rows from the partition are copied online into this intermediate table with the dbms_redefinition package. Once this is complete, the intermediate table and partition are swapped: The table becomes the partition and the partition becomes the table. As a result, all global indexes on the partitioned tables are invalidated, because all row IDs indexing the rows in the exchanged partition are no longer valid (they still indicate the rows which are now part of the intermediate table and are therefore no longer valid). As a result all global indexes must be rebuilt.

Local indexes are not invalidated because you can create the appropriate indexes on the intermediate tables as part of the reorganization. These indexes will also be swapped, becoming a local index for the reorganized partition.

Therefore only if the table has only local indexes is this method an option.

Online segment shrink

Online segment shrink is available as of Oracle Database 10g or higher. As a prerequisite, the partition must be located in a tablespace using ASSM (Automatic Segment Storage Management).

You can execute the online shrink command on a single partition, e.g.:

```
ALTER TABLE VBRP MODIFY PARTITION VBRP_P0001 SHRINK SPACE;
```

This will shrink the space on the table partition and on the dependent objects (local indexes) as well.

Oracle solution: Oracle partitioning engine for SAP

To allow customers to benefit from the advantages of the Oracle partitioning option in SAP ERP we have developed a solution based on Oracle functionality. This solution consists of three parts:

1. An analysis package to create an initial partitioning layout for a table
2. An administration package to add new partitions to a partitioned table on-demand
3. An administration package to remove partitions no longer needed from the tables

This package is based on PL/SQL procedures and can be used on request by interested customers. Please contact Michael Weick (michael.weick@oracle.com)

Customer implementations

The Oracle partitioning implementation based on the above concept is being used successfully by different customers. Alongside the realization based on range partitioning, we also have a few installations which are tailored to the special needs of particular customers. These examples may give you an idea of how flexible and powerful Oracle partitioning can be.

Example 1: Range partitioning-based NRIV approach

Customer request

The customer is using the SAP retail modules and has a couple of tables which are growing rapidly. Furthermore a huge flow of data is passing through the system daily. Although archiving is in place, the overall size of the database has become more than 10TB. To reduce the amount of work involved in reorganizing tables and indexes and to reduce the downtime needed for these tasks, the customer requested a new approach.

Solution

We decided to implement range partitioning based on the NRIV approach for this system. So we partitioned the tables EKPO (and all the other tables related to this one such as EKBE, EKPA), EDIDS, EDIDC, BDCPS, BDCP, ACCTIT, ACCTCR, ACCTHD, VBRP and some others. The NRIV approach can be implemented for all these tables. Please note that for the IDOC tables, the table EDI40 is a cluster table, which normally has a LONG RAW field. This field type is not supported for partitioning. You have to convert it to a LOB type which can be done as of SAP kernel release 7.00. After the conversion it can be partitioned like EDIDS and EDICD using the same partitioning approach as these tables.

Advantages

The performance should be significantly improved for a couple of queries. The amount of administrative work for the large tables could be significantly decreased because partitions and indexes are maintained automatically using the partition administration package.

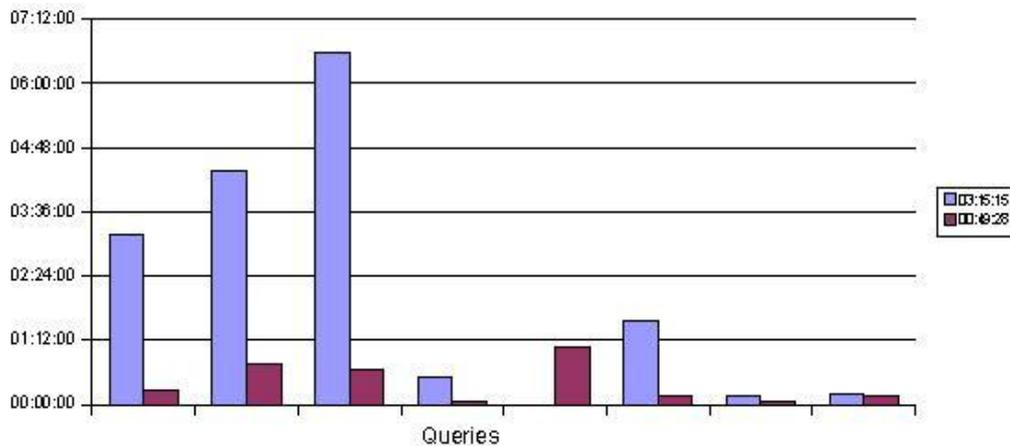


Diagram 11: Result query performance using the range partitioning NRIV approach.

Example 2: List partitioning based on the SAP client

Customer request

The customer runs a multi-client system, serving three different companies in one SAP system using different clients (100, 200, 600) for each customer. Customers are charged for the amount of space allocated on the storage. We therefore only had estimates about the space allocated to each customer, but no exact measurements for the overall billing process.

Solution

To increase transparency for the billing process as well as to increase and obtain a detailed overview of the space allocated to each customer, we decided to implement list partitioning on the SAP client on the 20 largest tables. These 20 tables include 80% of all SAP data. A separate target tablespace was chosen for each client to increase transparency for the space usage between the different clients.

Because the client field is part of almost all the indexes, we were able to create all indexes on these tables with a "LOCAL" definition, which gives maximum flexibility and the best performance results.

Advantages

Beside the increased transparency for the allocated space used by each customer, we envisage additional advantages for this customer:

- Data growth for one customer (client) does not affect the performance of the other customers. Because the client/mandt field is always part of all SAP SQL statements, Oracle can exclude all partitions from access if these belong to a different client (partition pruning). So we can reduce the amount of data which potentially has to be processed by an average of 60%.
- Partitions and indexes can be reorganized for each customer separately, which simplifies scheduling for a planned system downtime. In fact, adjustment need only be done with one customer instead of three.

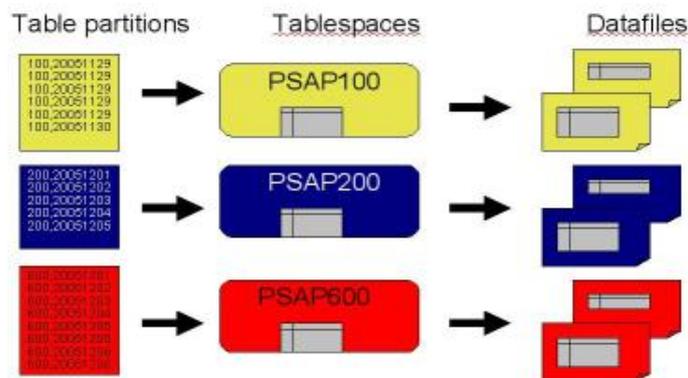


Diagram 12: Using different tablespace for client based partitioning

Example 3: List partitioning based on the WERKS column

Customer request

The customer has approx. 30 different manufacturing plants. He wanted to improve system performance and to avoid contention on the large tables during mass processing in month-end closings.

Solution

After intensive analysis of the system and the Shared Cursor Cache, we decided to go for list partitioning because all relevant SQL statements always included the column "WERKS". This criterion is used by the customer to separate data access to exactly one plant.

Based on these findings, we felt that a customized solution for this customer made most sense: Separating all tables to partition based on the column "WERKS" or a comparable column. The SQL analysis also allowed us to decide to create all indexes as LOCAL indexes. Although the partitioning key was not part of the index definition, the information to perform effective partition pruning could be extracted from the SQL statement.

Advantages

Comparing the SQL-query performance before (blue) and after partitioning (red) showed a performance enhancement by a factor of 7. Both tests were executed on reorganized tables and indexes.

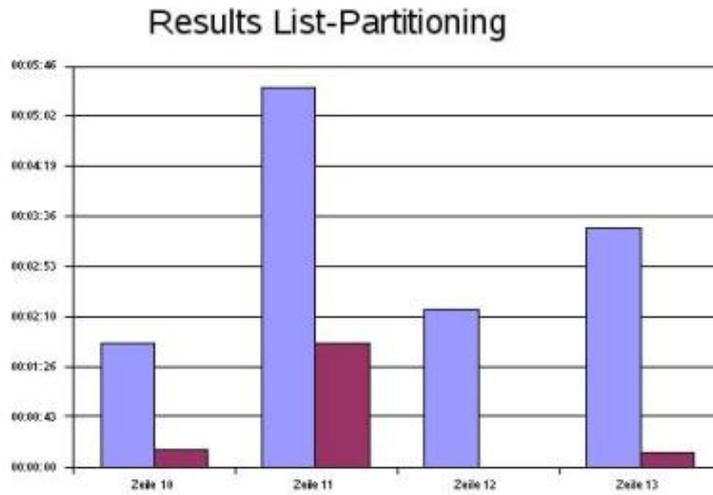


Diagram 13: Performance comparison of partitioned and unpartitioned tables



Partitioning for SAP ERP
A Guide for planning and setting up Oracle
partitioning in SAP ERP systems

January 2010

Author: Dr. Stephan Bühne

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2009, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

0110