

# Partitioning in Oracle Database 11g

*An Oracle White Paper*  
*June 2007*

**NOTE:**

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

# Partitioning in Oracle Database 11g

Note.....	2
Partitioning – Concepts.....	5
Introduction.....	5
Benefits of Partitioning .....	5
Basics of Partitioning.....	5
Partitioning for Manageability .....	7
Partitioning for Performance .....	7
Partitioning for Availability .....	8
Partitioning – Modeling for your Business.....	9
Basic Partitioning Strategies.....	9
Partitioning Extensions.....	10
Partition Advisor.....	11
Partitioning Strategies and Extensions at a Glance.....	12
Information Lifecycle Management with Partitioning.....	12
Conclusion.....	13

# Partitioning in Oracle Database 11g

## **PARTITIONING – CONCEPTS**

### **INTRODUCTION**

Oracle Partitioning, first introduced in Oracle 8.0 in 1997, is one of the most important and successful functionalities of the Oracle database, improving the performance, manageability, and availability for tens of thousands of applications. Oracle Database 11g introduces the 8<sup>th</sup> generation of partitioning which continues to offer ground-breaking new and enhanced functionality; new partitioning techniques enable customers to model even more business scenarios while a complete new framework of partition advice and automation enables the usage of Oracle Partitioning for everybody. Oracle Database 11g is considered the biggest new release for partitioning since its first introduction, continuing to protect our customers' investment in partitioning for a decade.

### **BENEFITS OF PARTITIONING**

Partitioning can provide tremendous benefits to a wide variety of applications by improving manageability, performance, and availability. It is not unusual for partitioning to improve the performance of certain queries or maintenance operations by an order of magnitude. Moreover, partitioning can greatly reduce the total cost of data ownership, using a “tiered archiving” approach of keeping older relevant information still online on low cost storage devices. Oracle Partitioning enables an efficient and simple, yet very powerful approach when considering Information Lifecycle Management for large environments.

Partitioning also enables database designers and administrators to tackle some of the toughest problems posed by cutting-edge applications. Partitioning is a key tool for building multi-terabyte systems or systems with extremely high availability requirements.

### **Basics of Partitioning**

Partitioning allows a table, index or index-organized table to be subdivided into smaller pieces. Each piece of the database object is called a partition. Each partition has its own name, and may optionally have its own storage characteristics. From the perspective of a database administrator, a partitioned object has multiple pieces that can be managed either collectively or individually. This gives the administrator considerable flexibility in managing partitioned

object. However, from the perspective of the application, a partitioned table is identical to a non-partitioned table; no modifications are necessary when accessing a partitioned table using SQL DML commands.

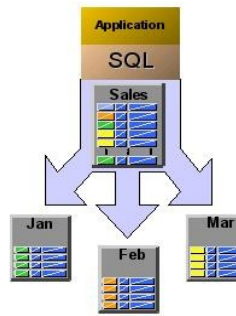


Figure 1: Application and DBA perspective of a partitioned table

Database objects - tables, indexes, and index-organized tables - are partitioned using a '**partitioning key**', a set of columns which determine in which partition a given row will reside. For example the sales table shown in figure 1 is range-partitioned on sales date, using a monthly partitioning strategy; the table appears to any application as a single, 'normal' table. However, the DBA can manage and store each monthly partition individually, potentially using different storage tiers, applying table compression to the older data, or store complete ranges of older data in read only tablespaces.

Irrespective of the chosen index partitioning strategy, an index is either coupled or uncoupled with the underlying partitioning strategy of the underlying table. The appropriate index partitioning strategy is chosen based on the business requirements, making partitioning well suited to support any kind of application. Oracle Database 11g differentiates between three types of partitioned indexes.

- **Local Indexes:** A local index is an index on a partitioned table that is coupled with the underlying partitioned table, 'inheriting' the partitioning strategy from the table. Consequently, each partition of a local index corresponds to one - and only one - partition of the underlying table. The coupling enables optimized partition maintenance; for example, when a table partition is dropped, Oracle simply has to drop the corresponding index partition as well. No costly index maintenance is required. Local indexes are most common in data warehousing environments.
- **Global Partitioned Indexes:** A global partitioned index is an index on a partitioned or non-partitioned table that is partitioned using a different partitioning-key or partitioning strategy than the table. Global-partitioned indexes can be partitioned using range or hash partitioning and are uncoupled from the underlying table. For example, a table could be range-partitioned by month and have twelve partitions, while an index on that table could be range-partitioned using a different partitioning key and have

a different number of partitions. Global partitioned indexes are more common for OLTP than for data warehousing environments.

- **Global Non-Partitioned Indexes:** A global non-partitioned index is essentially identical to an index on a non-partitioned table. The index structure is not partitioned and uncoupled from the underlying table. In data warehousing environments, the most common usage of global non-partitioned indexes is to enforce primary key constraints. OLTP environments on the other hand mostly rely on global non-partitioned indexes.

Oracle additionally provides a comprehensive set of SQL commands for managing partitioning tables. These include commands for adding new partitions, dropping, splitting, moving, merging, truncating, and optionally compressing partitions.

### **Partitioning for Manageability**

Oracle Partitioning allows tables and indexes to be partitioned into smaller, more manageable units, providing database administrators with the ability to pursue a "divide and conquer" approach to data management.

With partitioning, maintenance operations can be focused on particular portions of tables. For example, a database administrator could compress a single partition containing say the data for the year 2006 of a table, rather than compressing the entire table. For maintenance operations across an entire database object, it is possible to perform these operations on a per-partition basis, thus dividing the maintenance process into more manageable chunks.

A typical usage of partitioning for manageability is to support a 'rolling window' load process in a data warehouse. Suppose that a DBA loads new data into a table on weekly basis. That table could be range-partitioned so that each partition contains one week of data. The load process is simply the addition of a new partition. Adding a single partition is much more efficient than modifying the entire table, since the DBA does not need to modify any other partitions.

Another advantage of using partitioning is when it is time to remove data, an entire partition can be dropped which is very efficient and fast, compared to deleting each row individually.

### **Partitioning for Performance**

By limiting the amount of data to be examined or operated on, partitioning provides a number of performance benefits. These features include:

- **Partitioning Pruning:** Partitioning pruning (a.k.a. Partition elimination) is the simplest and also the most substantial means to improve performance using partitioning. Partition pruning can often improve query performance by several orders of magnitude. For example, suppose an application contains an ORDERS table containing an historical record

of orders, and that this table has been partitioned by week. A query requesting orders for a single week would only access a single partition of the ORDERS table. If the table had 2 years of historical data, this query would access one partition instead of 104 partitions. This query could potentially execute 100x faster simply because of partition pruning. Partition pruning works with all of Oracle's other performance features. Oracle will utilize partition pruning in conjunction with any indexing technique, join technique, or parallel access method.

- **Partition-wise Joins:** Partitioning can also improve the performance of multi-table joins, by using a technique known as partition-wise joins. Partition-wise joins can be applied when two tables are being joined together, and at least one of these tables is partitioned on the join key. Partition-wise joins break a large join into smaller joins of 'identical' data sets for the joined tables. 'Identical' here is defined as covering exactly the same set of partitioning key values on both sides of the join, thus ensuring that only a join of these 'identical' data sets will produce a result and that other data sets do not have to be considered. Oracle is using either the fact of already (physical) equi-partitioned tables for the join or is transparently redistributing (= "repartitioning") one table at runtime to create equi-partitioned data sets matching the partitioning of the other table, completing the overall join in less time. This offers significant performance benefits both for serial and parallel execution.

### **Partitioning for Availability**

Partitioned database objects provide partition independence. This characteristic of partition independence can be an important part of a high-availability strategy. For example, if one partition of a partitioned table is unavailable, all of the other partitions of the table remain online and available. The application can continue to execute queries and transactions against this partitioned table, and these database operations will run successfully if they do not need to access the unavailable partition.

The database administrator can specify that each partition be stored in a separate tablespace; this would allow the administrator to do backup and recovery operations on each individual partition, independent of the other partitions in the table. Therefore in the event of a disaster, the database could be recovered with just the partitions comprising of the active data, and then the inactive data in the other partitions could be recovered at a convenient time. Thus decreasing the system down-time.

Moreover, partitioning can reduce scheduled downtime. The performance gains provided by partitioning may enable database administrators to complete maintenance operations on large database objects in relatively small batch windows.

## PARTITIONING – MODELING FOR YOUR BUSINESS

Oracle Database 11g provides the most comprehensive set of partitioning strategies, allowing a customer to optimally align the data subdivision with the actual business requirements. All available partitioning strategies rely on **fundamental data distribution methods** that can be used for either single (one-level) or composite partitioned tables. Furthermore, Oracle provides a variety of **partitioning extensions**, increasing the flexibility for the partitioning key selection, providing automated partition creation as-needed, and advising on partitioning strategies for non-partitioned objects.

### Basic Partitioning Strategies

Oracle Partitioning offers three fundamental data distribution methods that control how the data is actually going to be placed into the various individual partitions, namely:

- **Range:** The data is distributed based on a range of values of the partitioning key (for a date column as the partitioning key, the 'January-2007' partition contains rows with the partitioning-key values between '01-JAN-2007' and '31-JAN-2007'). The data distribution is a continuum without any holes and the lower boundary of a range is automatically defined by the upper boundary of the preceding range.
- **List:** The data distribution is defined by a list of values of the partitioning key (for a region column as the partitioning key, the 'North America' partition may contain values 'Canada', 'USA', and 'Mexico'). A special 'DEFAULT' partition can be defined to catch all values for a partition key that are not explicitly defined by any of the lists.
- **Hash:** A hash algorithm is applied to the partitioning key to determine the partition for a given row. Unlike the other two data distribution methods, hash does not provide any logical mapping between the data and any partition.

Using the above-mentioned data distribution methods, a table can be partitioned either as single or composite partitioned table:

- **Single (one-level) Partitioning:** A table is defined by specifying one of the data distribution methodologies, using one or more columns as the partitioning key. For example consider a table with a number column as the partitioning key and two partitions 'less\_than\_five\_hundred' and 'less\_than\_thousand', the 'less\_than\_thousand' partition contains rows where the following condition is true:  $500 \leq \text{Partitioning key} < 1000$ .  
You can specify Range, List, and Hash partitioned tables.
- **Composite Partitioning:** A combination of two data distribution methods are used to define a composite partitioned table. First, the table is partitioned by data distribution method one and then each partition is



further subdivided into subpartitions using a second data distribution method. All sub-partitions for a given partition together represent a logical subset of the data. For example, a range-hash composite partitioned table is first range-partitioned, and then each individual range-partition is further sub-partitioned using the hash partitioning technique.

Available composite partitioning techniques are range-hash, range-list, range-range, list-range, list-list, and list-hash.

- Index-organized tables (IOTs) can be partitioned using range, hash, and list partitioning. Composite partitioning is not supported for IOTs.

## Partitioning Extensions

In addition to the basic partitioning strategies, Oracle provides partitioning extensions. The extensions in Oracle Database 11g mainly focus on two objectives:

- (a) Enhance the manageability of a partitioned table significantly.
- (b) Extend the flexibility in defining a partitioning key.

The extensions are namely:

**Interval Partitioning:** A new partitioning strategy in Oracle Database 11g, Interval partitioning extends the capabilities of the range method to define equi-partitioned ranges using an interval definition. Rather than specifying individual ranges explicitly, Oracle will create any partition automatically as-needed whenever data for a partition is inserted for the very first time. Interval partitioning greatly improves the manageability of a partitioned table. For example, an interval partitioned table could be defined so that Oracle creates a new partition for every month in a calendar year; a partition is then automatically created for 'September 2007' as soon as the first record for this month is inserted into the database.

The available techniques for an interval partitioned table are Interval, Interval-List, Interval-Hash, and Interval-Range.

**REF Partitioning:** Oracle Database 11g allows to partition a table by leveraging an existing parent-child relationship. The partitioning strategy of the parent table is inherited to its child table without the necessity to store the parent's partitioning key columns in the child table. Without REF Partitioning you have to duplicate all partitioning key columns from the parent table to the child table if you want to take advantage from the same partitioning strategy; REF Partitioning on the other hand allows you to naturally partition tables according to the logical data model without requiring to store the partitioning key columns, thus reducing the manual overhead for denormalization and saving space. REF Partitioning also transparently inherits all partition maintenance operations that change the logical shape of a table from the parent table to the child table. Furthermore, REF Partitioning automatically enables partition-wise joins for the equi-partitions of

the parent and child table, improving the performance for this operation. For example, a parent table ORDERS is Range partitioned on the ORDER\_DATE column; its child table ORDER ITEMS does not contain the ORDER\_DATE column but can be partitioned by reference to the ORDERS table. If the ORDERS table is partitioned by month, all order items for orders in 'Jan-2007' will then be stored in a single partition in the ORDER ITEMS table, equi-partitioned to the parent table ORDERS. If a partition 'Feb-2007' is added to the ORDERS table Oracle will transparently add the equivalent partition to the ORDER ITEMS table.

All basic partitioning strategies are available for REF Partitioning.

**Virtual column-based Partitioning:** In previous versions of Oracle, a table could only be partitioned if the partitioning key physically existed in the table. Virtual columns, a new functionality in Oracle Database 11g, removes that restriction and allows the partitioning key to be defined by an expression, using one or more existing columns of a table, and storing the expression as metadata only.

Partitioning has been enhanced to allow a partitioning strategy being defined on virtual columns, thus enabling a more comprehensive match of the business requirements. It is not uncommon to see columns being overloaded with information; for example a 10 digit account ID can include an account branch information as the leading three digits. With the extension of virtual column-based Partitioning, the ACCOUNTS table containing a column ACCOUNT\_ID can be extended with a virtual (derived) column ACCOUNT\_BRANCH that is derived from the first three digits of the ACCOUNT\_ID column which becomes the partitioning key for this table.

Virtual column-based Partitioning is supported with all basic partitioning strategies.

### **Partition Advisor**

The SQL Access Advisor in Oracle Database 11g has been enhanced to generate partitioning recommendations, in addition to the ones it already provides for indexes, materialized views and materialized view logs. Recommendations generated by the SQL Access Advisor – either for Partitioning only or holistically - will show the anticipated performance gains that will result if they are implemented. The generated script can either be implemented manually or submitted onto a queue within Oracle Enterprise Manager.

With the extension of partitioning advice, customers not only can get recommendation specifically for partitioning but also a more comprehensive holistic recommendation of SQL Access Advisor, improving the collective performance of SQL statements overall.

The Partition Advisor, integrated into the SQL Access Advisor, is part of Oracle's Tuning Pack, an extra licensable option. It can be used from within Enterprise Manager or via a command line interface.

## Partitioning Strategies and Extensions at a Glance

The following table gives a conceptual overview of all available basic partitioning strategies in Oracle Database 11g:

Partitioning Strategy	Data Distribution	Sample Business Case
Range Partitioning	Based on consecutive ranges of values.	<ul style="list-style-type: none"> <li>Orders table range partitioned by order_date</li> </ul>
List Partitioning	Based on unordered lists of values.	<ul style="list-style-type: none"> <li>Orders table list partitioned by country</li> </ul>
Hash Partitioning	Based on a hash algorithm.	<ul style="list-style-type: none"> <li>Orders table hash partitioned by customer_id</li> </ul>
Composite Partitioning <ul style="list-style-type: none"> <li>Range-Range</li> <li>Range-List</li> <li>Range-Hash</li> <li>List-List</li> <li>List-Range</li> <li>List-Hash</li> </ul>	Based on a combination of two of the above-mentioned basic techniques of Range, List, Hash, and Interval Partitioning	<ul style="list-style-type: none"> <li>Orders table is range partitioned by order_date and sub-partitioned by hash on customer_id</li> <li>Orders table is range partitioned by order_date and sub-partitioned by range on shipment_date</li> </ul>

In addition to the available partitioning strategies, Oracle Database 11g provides the following partitioning extensions:

Partitioning Extension	Partitioning Key	Sample Business Case
Interval Partitioning <ul style="list-style-type: none"> <li>Interval</li> <li>Interval-Range</li> <li>Interval-List</li> <li>Interval-Hash</li> </ul>	An extension to Range Partition. Defined by an interval, providing equi-width ranges. With the exception of the first partition all partitions are automatically created on-demand when matching data arrives.	<ul style="list-style-type: none"> <li>Orders table partitioned by order_date with a predefined daily interval, starting with '01-Jan-2007'</li> </ul>
REF Partitioning	Partitioning for a child table is inherited from the parent table through a primary key – foreign key relationship. The partitioning keys are not stored in actual columns in the child table.	<ul style="list-style-type: none"> <li>(Parent) Orders table range partitioned by order_date and inherits the partitioning technique to (child) order lines table. Column order_date is only present in the parent orders table</li> </ul>
Virtual column based Partitioning	Defined by one of the above-mentioned partition techniques and the partitioning key is based on a virtual column. Virtual columns are not stored on disk and only exist as metadata.	<ul style="list-style-type: none"> <li>Orders table has a virtual column that derives the sales region based on the first three digits of the customer account number. The orders table is then list partitioned by sales region.</li> </ul>

## **INFORMATION LIFECYCLE MANAGEMENT WITH PARTITIONING**

Today's challenge of storing vast quantities of data for the lowest possible cost can be optimally addressed using Oracle Partitioning. The independence of individual partitions is the key enabler for addressing the online portion of a “tiered archiving” strategy. Specifically in tables containing historical data, the importance - and access pattern – of the data heavily relies on the age of the data; Partitioning enables individual partitions (or groups of partitions) to be stored on different storage tiers, providing different physical attributes and price points. For example an Orders table containing 2 years worth of data could have only the most recent quarter being stored on an expensive high-end storage tier and keep the rest of the table (almost 90% of the data) on an inexpensive low cost storage tier. Through Oracle Partitioning, the storage costs are reduced by factors (cost savings of 50% or more are not uncommon), without impacting the end user access, thus optimizing the cost of ownership for the stored information.

The Oracle ILM Assistant which is a freely available tool downloadable from OTN, can illustrate those cost savings, show you how to partition the table and advise when it is time to move partitions to other storage tiers.

## **CONCLUSION**

Considering the new and improved functionality for Oracle Partitioning, Oracle Database 11g is the most significant release since the introduction of Oracle Partitioning in 1997. In every major release, Oracle has enhanced the functionality of Partitioning, by either adding new partitioning techniques, enhancing the scalability, or extending the manageability and maintenance capabilities. Oracle plans to continue to add new partitioning techniques to ensure that an optimal partitioning technique is available for every business requirement.

Partitioning is for everybody. Oracle Partitioning can greatly enhance the manageability, performance, and availability of almost any database application. Partitioning can be applied to cutting-edge applications and indeed partitioning can be a crucial technology ingredient to ensure these applications' success. Partitioning can also be applied to more commonplace database applications in order to simplify the administration and costs of managing such applications. Since partitioning is transparent to the application, it can be easily implemented because no costly and time-consuming application changes are required.



Partitioning in Oracle Database 11g

June 2007

Author: Hermann Baer

Contributing Authors:

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](http://oracle.com)

Copyright © 2007, Oracle. 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.