19c ORACLE Database

Oracle Partitioning

Extreme Data Management and Performance

WHITE PAPER / FEBRUARY 19, 2019

ORACLE

PURPOSE STATEMENT

This document provides an overview of features and enhancements of Oracle Partitioning. It is intended solely to help you assess the business benefits of Oracle Database and to plan your I.T. projects.

DISCLAIMER

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. 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 in this document remains at the sole discretion of Oracle.

Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

TABLE OF CONTENTS

INTRODUCTION

With more than 20 years in development, Oracle Partitioning has established itself as one of the most successful and commonly used functionalities of the Oracle database. With Oracle Partitioning, a single logical object in the database is subdivided into multiple smaller physical objects, so-called partitions. The knowledge about this physical partitioning enables the database to improve the performance, manageability, or availability for any application. Whether you have an OLTP, a data warehouse, or a mixed workload application and whether your system is hundreds of GBs or in the Petabyte range, you will benefit from Partitioning. Queries and maintenance operations are sped up by an order of magnitude, while minimizing the resources necessary for processing. Together with zone maps pruning capabilities are unlimited: tables and partitions are broken down into smaller physical zones that are used for fine-grained data pruning, in addition to the knowledge of partitions in a table.

Partitioning can greatly reduce the total cost of data ownership, using a “tiered archiving” approach of keeping older relevant information still online, in the most optimal compressed format and on low cost storage devices, while storing the hottest data in Oracle’s in-memory column store. When used together with Automatic Data Optimization and Heat Maps, Partitioning provides a simple and automated way to implement an Information Lifecycle Management (ILM) strategy. With hybrid partitioned tables, introduced in Oracle Database 19c, partitioned tables now can also spawn internal and external storage within the same logical table, bringing ILM to the next level.

Oracle Partitioning improves the performance, manageability, and availability for tens of thousands of customers and hundreds of thousands of applications. Everybody can benefit from it, and so can you.

PARTITIONING FUNDAMENTALS

Concept of Partitioning

Partitioning enables tables and indexes to be subdivided into individual smaller pieces. Each piece of the database object is called a partition. A 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 a 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. Logically, it is still only one table and any application can access this one table as they do for a non-partitioned table.

One logical object, many physical partitions

From the perspective of the application you have one table you access. From the perspective of the administrator, you have multiple partitions you manage individually.

Database objects – tables and indexes - are partitioned using a **partitioning key**, a set of columns that determine in which partition a given row will reside. The partitions of a table physically store the data, while the table itself is metadata only. For example, the Sales table shown in Figure 1 is range-partitioned on sales (order) date, using a monthly partitioning strategy; the table appears to any application as a single, 'normal' table. However, the database administrator can manage and store each monthly partition individually, optimizing the data storage according to the importance of data and the frequency of being used. Partitions storing older ranges of data can be stored in different storage tiers using table compression (or even stored in read only tablespaces or marked as read only partitions) while the newest partitions are marked for being stored in Oracle's in-memory column store. With hybrid partitioned tables, some partitions can reside on internal storage while others reside on external storage, all within the same logical table.

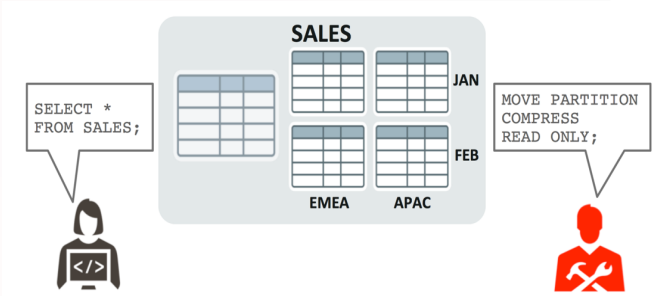

Figure 1: Application and DBA view of a partitioned table

In case of a composite partitioned table, a partition is further subdivided into subpartitions, using a second set of columns for further subdivision within a partition; the data placement of a given row is then determined by both partitioning key criteria and placed in the appropriate subpartition. With a composite partitioned table, the partition level becomes a metadata layer. Only subpartitions are physically stored on disk[1].

In the case of a partitioned external table, the concept of having different physical segments for different parts of a table is extended to physical storage outside the database. Each partition of an external table has one or multiple individual files that represent the subset of data of the partition. However, unlike regular partitioned tables, the data placement is not enforced by the database. External tables, partitioned or non-partitioned, are read only.

Hybrid partitioned tables combine the concept of both internal and external partitioned tables. As the name suggests, with such a partitioned table you can have both partitions being stored internal (in the database) and external (on physical storage outside the database). The same rules apply for external partitions of a hybrid partitioned table than for partitions of a partitioned external table: the data placement is not enforced by the database, and the content of such partitions is read only.

Application developers generally do not have to worry about whether or not a table is partitioned, but they also can leverage partitioning to their advantage: for example, a resource intensive DML operation to purge data from a table can be implemented using partition maintenance operations, improving the runtime dramatically while reducing the resource consumption significantly.

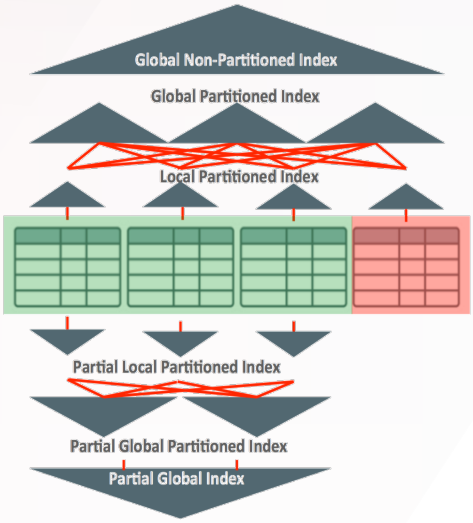

Figure 2: Indexing on partitioned tables

Irrespective of the chosen table partitioning strategy, any index of a partitioned table is either coupled or uncoupled with the underlying partitioning strategy of its table. Oracle Database 18c differentiates between three types of indexes[2].

A **local index** is an index on a partitioned table that is coupled with the underlying partitioned table; the index 'inherits' 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 since an index partition is by definition only tied to its table partition; a local index segment will never contain data of other partitions. Local indexes are most common in data warehousing environments.

Flexible indexing for partitioned tables

- Indexes can be partitioned or not.
- Indexes can be tied to table partitions or independent of those.
- Indexing can be for the whole table or partially for a subset of partitions.

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 hash-partitioned using a different partitioning key and have a different number of partitions. Decoupling an index from its table automatically means that any partition maintenance operation on the table can potentially cause index maintenance operations. Global partitioned indexes are more common for OLTP than for data warehousing environments.

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.

All of the before-mentioned index types can be either created on all partitions of a partitioned table – so-called **full indexing**, the default – or created only on a subset of the partitions of a partitioned table – so-called **partial indexing**[3].

[1] For simplicity reasons we will refer to partitions only for the rest of this document.

[2] Partitioned external tables cannot be indexed.

[3] Unique indexes cannot be partial.

Only indexes on partitioned tables can be partial indexes. Whether a particular partition will be indexed is determined by the properties of the partition and applied to all partial indexes. With partial indexing you can for example to not index the most recent partition to avoid any index maintenance work at data insertion time, therefore maximizing data load speed. Together with zone map data pruning the potential impact of not having indexes for selective data access of the most recent partition is minimized.

The appropriate indexing strategy is chosen based on the business requirements and access patterns, making partitioning well suited to support any kind of application.

Partitioning for Performance

The placement of a given row is determined by its value of the partitioning key. How the data of a table is subdivided across the partitions is stored as partitioning metadata of a table or index. This metadata is used to determine for every SQL operation – queries, DML, and partition maintenance operations - what partitions of a table are relevant for a given operation, and the database automatically only touches relevant partitions or, with zone maps, even only portions of a partition or a table. By limiting the amount of data to be examined or operated on, partitioning provides a number of performance benefits.

Partitioning pruning (a.k.a. partition elimination) is the simplest and also the most effective means to improve performance. It can often improve query performance by several orders of magnitude by leveraging the partitioning metadata to only touch the data of relevance for a SQL operation. For example, suppose an application contains an Orders table containing an historical record of orders, and that this table has been partitioned by day on order date. A query requesting orders for a single week would only access seven partitions of the Orders table. If the table had 2 years of historical data, this query would access seven partitions instead of 730 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.

Zone maps[4] expand Oracle's pruning capabilities beyond the partitioning metadata of a table. Data pruning can occur on a partition level and even on a much finer granularity, on 'zones'. A zone is a contiguous region of blocks for which a zone map tracks the minimum and maximum values for specified columns. Note that these columns are not the partition key columns; while partition key columns can be included, the most common usage of zone maps is to use other non-partition key columns. For partitioned tables the zone map also contains the aggregated minimum and maximum column values for every partition. Whenever a SQL operation is using the columns specified in a zone map to limit (filter) the data of interest, Oracle will compare the filter and the zone map information and not access zones and partitions that do not contain matching data. Zone maps are similar to Exadata Storage indexes in that sense but provide additional benefits that complement Storage Indexes. Zone maps are persistent data structures processed in the database and allow the specification of local columns – columns of the table with the zone map - and joined columns.

[4] See the Oracle Data Warehousing Guide for a detailed and comprehensive discussion of zone maps.

Having zone maps in the database allows every statement to benefit. Using the Sales table from the previous example, any query requesting information about sales orders that were shipped in a specific time period has to access all partitions of the Orders table (because the partitioning key is order date, not ship date). While there is a correlation between the order date and the ship date, it is impossible to limit the partitions being accessed using the ship date alone. With zone maps, however, the database knows about the minimum and maximum values for ship date as well; the zone map stores this information for every partition. If the order data and ship date are within a business week of one another, queries asking for products that were shipped in the last three weeks would only have to access the partitions for orders of the last four weeks, and within these partitions only zones that were shipped in that time period. You get partition and zone map pruning without having specified any filter criteria on the partition key column.

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 – the smaller one - at runtime to create equi-partitioned data sets matching the partitioning of the other table, completing the overall join in less time, using less resources. This offers significant performance benefits both for serial and parallel execution.

Partitioning for Manageability

By partitioning tables and indexes into smaller, more manageable units, database administrators can use a "divide and conquer" approach to data management. Oracle provides a comprehensive set of SQL commands for managing partitioning tables. These include commands for adding new partitions, dropping, splitting, moving, merging, truncating, and exchanging partitions.

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 2017 of a table, rather than compressing the entire table; as part of the compression operation, this partition could also be moved to a lower cost storage tier, reducing the total cost of ownership for the stored data even more. This partition maintenance operation can be done in a completely online fashion, allowing both queries and DML operations to occur while the data maintenance operation is in process.

Beginning with Oracle Database 18c you can execute partition maintenance operations on multiple partitions as single atomic operation: for example, you can merge the three partitions ‘January 2018’, ‘February 2018’, and ‘March 2018’ into a single partition ‘Q1 2018’ with a single merge partition operation.

Another 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 daily basis. That table could be range-partitioned so that each partition contains one day 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.

Removing data in a very efficient and elegant manner is another key advantage of partitioning. For example, to purge data from a partitioned table you simply drop or truncate one or multiple partitions, a very cheap and quick data dictionary operation, rather than issuing the equivalent delete command, using lots of resources and touching all the rows to being deleted. The common operation of removing data with a partition maintenance operation such as drop or truncate is optimized beginning with Oracle Database 18c: these operations do not require any immediate index maintenance to keep all indexes valid, making it fast metadata-only operations[5].

While Partition maintenance operations allow the fast removal of data, the granularity of such an operation is tied to the bounds of the partitions being dropped or truncated. But as often in life, there are rules to the exception: for example, as part of you rolling window operation you want to remove all data that is older than 3 years, but you must not remove any order that has not been officially closed. While this is a very rare situation for your business, this business requirement rules out to use a truncate or drop partition out of the box. You have to cope with this situation programmatically by preserving the outliers. Beginning with Oracle Database 18c, partition maintenance operations got enhanced to allow filtering of data as part of any partition maintenance operation. In our example, moving the partition and preserving all old records that are not officially closed achieve the removal of the data. Filtered partition maintenance operations bring data maintenance to partition maintenance operations[6].

Furthermore, existing nonpartitioned and partitioned tables can be modified to a partitioned table or to change the partitioning strategy in a fully online manner, including the modification of all existing indexes. Whether you have the need to introduce partitioning for a growing system, to adjust the partitioning strategy to address changing business requirements, or to cope with even more growth, Oracle Database got you covered.

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 (when an operation tries to access data that is not available then such an operation will obviously fail; Oracle only returns true and valid results, no matter what).

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 an individual partition or sets of partitions (by virtue of the partition-to-tablespace mapping), independent of the other partitions in the table. In the event of a disaster, the database can be recovered with just the partitions comprising of the active data, and then the inactive data in the other partitions can be recovered at a convenient time, thus decreasing the system down-time. The most relevant data becomes available again in the shortest amount of time, irrespective of the size of the overall database.

[5] Asynchronous global index maintenance is discussed in the VLDB and Partitioning Guide

[6] Filtered partition maintenance operations allow filter predicates on the partitioned table only and does not support joins or any other complex SQL constructs.

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.

Information Lifecycle Management with Partitioning

Today's challenge of storing vast quantities of data for the lowest possible cost can be optimally addressed by using Oracle Partitioning with Automatic Data Optimization and Heat Map. The independence of individual partitions, together with efficient and transparent data maintenance operations for partitions, are key enablers 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 – such as compression or whether data is read only or not - and price points. With hybrid partitioned tables some of the older partitions can even reside outside the Oracle database on external storage. Such data is by nature read only.

For internal partitions you can set individual partitions to read only, in addition to read only tablespaces - which prevent any physical changes to the underlying storage container(s) of a tablespace. Setting a partition to read only prevents any DML of the data within a partition to prevent any inadvertent changes to the data within a read only partition. Technically speaking, read only guarantees that the data in all existing columns of the table at the point when a partition was made read only must not change. For example, in a Sales orders table containing 5 years’ worth of data, you could store only the most recent quarter 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. You furthermore can store the oldest 2 years as external partitions outside the database and the next 2 years as read only partitions. Only the most recent years’ data can be changed, and all the older data is immutable and still available from within the system for regulatory purposes, even if not all data is stored within the database[7].

The addition of Automatic Data Optimization – ADO – allows you to define policies that specify when storage tiering and compression tiering should be implemented for a given partition, based on the usage statistics automatically collected by Heat Map. ADO policies are automatically evaluated and executed by the Oracle Database without any manual intervention required, making it possible to experience the cost savings and performance benefits of storage tiering and compression without creating complex scripts and jobs[8].

[7] Note that while external partitions are read only from within the database, the database does not have control over its content nor can it guarantee that the data is not changed from outside the database. Only for read only internal partitions Oracle can guarantee that the data of such partitions cannot be changed as long as a partition is set to read only.

[8] ADO supports only internal (database-managed) storage tiers as of today.

PARTITIONING STRATEGIES

Oracle provides the most comprehensive set of partitioning strategies, allowing customers 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 (two-level) partitioned tables. Furthermore, Oracle provides a variety of **partitioning extensions**, increasing the flexibility for the partitioning key selection, providing automated partition creation as-needed, sharing partitioning strategies across groups of logically connected tables through parent-child relationships, and advising on partitioning strategies for non-partitioned objects.

Data Distribution Methods for Partitioned Objects

Oracle Partitioning offers three fundamental, basic data distribution methods that control how the data is placed into 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-2018' partition contains rows with the partitioning-key values between '01-JAN-2018' and '31-JAN-2018'). Range distribution is a continuum without any holes. Ranges are always defined as an excluding upper boundary of a partition, and the lower boundary of a partition is automatically defined by the exclusive upper boundary of the preceding partition. Partition boundaries are always increasing; as a consequence, the first partition of a table – the one with the lowest range boundary - is always open-ended towards lower values. The last partition – the one with the highest partition boundary – can be optionally set to being open-ended as well. Range partitioning can have one or multiple partition key columns, up to 16 columns.
- **List:** The data distribution is defined by a discrete 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. For heap tables, list partitioning can have one or multiple partition key columns, up to 16 columns. Index-organized tables only support one partition key column.
- **Hash:** An internal hash algorithm is applied to the partitioning key to determine the partition for a given partition key. Unlike the other two data distribution methods, hash does not provide any logical mapping between the data and any partition, but it provides roughly equi-balanced sizes of the partitions. You get the best balance of partition sizes with a sufficient number of distinct values for the partitioning key and by choosing a number of partitions that is a power of two, e.g. 4, 16, 64. Hash partitioning can have one or multiple partition key columns, up to 16 columns.

Using these three fundamental data distribution methods range, list, and hash, a table can be partitioned either as single or composite partitioned table.

In addition to the fundamental methods Oracle offers **System partitioning:** the database only provides the framework to partition a table but does not store any metadata to determine the data placement. The application layer manages the data placement, both for data insertion and for data access (if the application wants to leverage partition pruning). System partitioning is designed as development framework with special needs for data placement or access, such as domain indexes, and only supports heap tables with single (one-level) partitioning. The definition and management of the equivalent of a partition key is solely in the discretion of the application.

SINGLE (ONE-LEVEL) PARTITIONING

A table is defined by specifying one of the above-mentioned 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 <= Partitioning key <1000. The partitions of a single partitioned table or index are individual physical segments in the database that store the actual data of the object.

You can specify range, list, hash, and system partitioned heap tables and index-organized tables. Hash clusters can be partitioned using range partitioning only[9].

COMPOSITE (TWO-LEVEL) PARTITIONING

Combinations 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 the second data distribution method. For example, a range-list composite partitioned table is first range-partitioned, and then each individual range-partition is further sub-partitioned using the list partitioning technique. Partitions of a composite partitioned table are metadata and do not represent the actual data storage: the subpartitions of a partition of a composite partitioned table or index are the physical segments in the database that store the data of a given partition.

Available composite partitioning techniques are range-hash, range-list, range-range, list-range, list-list, list-hash, as well as hash-hash, hash-range, and hash-list. Composite partitioning is only supported for heap tables managed by the database.

Global partitioned indexes can be partitioned using range or hash partitioning. Composite partitioning is not supported for global partitioned indexes.

Partitioning Extensions

Oracle provides partitioning extensions that enhance the usage of the basic partitioning strategies. Partitioning extensions enhance the manageability of partitioned objects and provide more flexibility in defining the partitioning key of a table or even groups of tables that are logically connected through parent-child relationships. Partitioning extensions are only supported for heap tables managed by the database.

[9] Clusters are schema objects that consist of multiple tables stored within its data structure. With Hash Clusters the database stores together rows that have the same hash value. Clusters are used predominantly in OLTP environments to minimize IO.

INTERVAL PARTITIONING

Interval partitioning extends the capabilities of the range method by defining equi-partitioned ranges for any future partitions using an interval definition as part of the table metadata. An interval partitioned table can automatically grow up to the maximum total number of 1048575 partitions without any user intervention, even when the partitioned table is initially created with one partition only. Rather than creating future individual range partitions explicitly, Oracle will create any new partition automatically as-needed whenever data for such 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 day in a calendar year; a partition is then automatically created for 'September 19th, 2031' as soon as the first record for this day is inserted into the database.

Interval partitioning is an extension to range partitioning. Any range partitioned table can be evolved into an interval partitioned table by specifying an interval definition for future partitions. The only requirement for this to happen is that the last partition of the range partitioned table has a discrete upper bound and not MAXVALUE prior to being changed. Having an open-ended infinite upper bound is contradictory to the creation of future partitions based on an interval definition.

The available techniques for an interval partitioned table are interval, interval-list, interval-hash, and interval-range. Oracle also supports the combination of the partitioning extensions interval partitioning and reference partitioning. Interval as subpartitioning strategy for any top-level partitioning method (*-Interval) is currently not supported.

AUTO LIST PARTITIONING

Similar to interval partitioning, auto list partitioning enables the automatic creation of new list partitions as soon as a new partition key value is inserted into an auto list partitioned table. Every distinct value will be stored in its individual partition if the value is not already included as partition key value of an existing partition.

Auto list partitioning is an extension to list partitioning, and any existing list partitioned table can be evolved into an auto list partitioned table. The only requirement for this to happen is that the list partitioned table must not have a DEFAULT partition defined prior to being changed. Having this catch-it-all partition is contradictory to the automatic creation of new partitions for any new partition key value.

Auto list partitioning is available as partition extension. It is currently not supported as subpartitioning strategy and not supported in combination with reference partitioning today.

REFERENCE PARTITIONING

Reference partitioning allows partitioning a table by leveraging an existing parent-child relationship. The primary key-foreign key relationship is used to inherit the partitioning strategy of the parent table to its child table without the necessity to store the parent's partitioning key columns in the child table. The partitioning strategy of a parent and child table becomes identical. For every partition in the parent table there is exactly one partition in the child table, and the child partitioning strategy is solely defined through the primary key-foreign key relationship. All child records of a given primary key value are stored in the “same” partition of the child table than the parent record. Without reference partitioning you have to duplicate all partitioning key columns from the parent table to the child table if you want to take advantage of the same partitioning strategy. Reference partitioning allows you to naturally leverage the parent-child relationship of the logical data model without duplication of the partitioning

key columns, thus reducing the manual overhead for de-normalization and saving space. Reference partitioning also transparently inherits all partition maintenance operations that change the logical shape of a table from the parent table to the child table. Partition-wise joins are automatically enabled when joining the equi-partitions of the parent and child table, improving the performance for this operation. For example, a parent table Sales 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 Sales orders table. If the orders table is partitioned by month, all order items for orders in 'March 2018' will then be stored in a single partition in the Order Items table, equi-partitioned to the parent table Orders. If a partition 'April 2018' is added to the Sales orders table – either explicitly or through Interval Partitioning - Oracle will transparently add the equivalent partition to the Order Items table.

Oracle supports the combination of reference partitioning with both virtual column-based partitioning and interval partitioning. Auto list partitioning is not supported together with reference partitioning.

VIRTUAL COLUMN-BASED PARTITIONING

Virtual columns allow 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 using virtual columns enables a more comprehensive match of the business requirements; business attributes not explicitly defined as columns in a table can be used to define the partitioning strategy of an object. It is not uncommon to see columns being overloaded with information; for example, a 10-digit account id can include 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 that becomes the partitioning key for this table.

Oracle supports virtual column-based partitioning with all other partitioning extensions.

Partition Advisor

SQL Access Advisor generates partitioning recommendations, in addition to recommendations for indexes, materialized views and materialized view logs. Recommendations generated by the SQL Access Advisor will show the anticipated performance gains that will result if the recommendations were implemented. The generated script with the recommendations can either be executed manually, as complete script or individual recommendations, or being submitted into a queue within Oracle Enterprise Manager.

The Partition Advisor is integrated into the SQL Access Advisor.

PARTITIONING FUNCTIONALITY AT A GLANCE

The following table shows all available basic partitioning methods:

Basic Partitioning methods

Partitioning Strategy	Data Distribution	Sample Business Case
Range Partitioning	Consecutive ranges of values.	Orders table range partitioned by order_date
List Partitioning	Unordered lists of values.	Orders table list partitioned by country
Hash Partitioning	Internal hash algorithm	Orders table hash partitioned by customer_id
Composite Partitioning • Range-[Range | List | Hash] • List-[Range | List | Hash] • Hash-[Range | List | Hash]	Combination of two of the above-mentioned basic techniques of Range, List, and Hash	Orders table is range partitioned by order_date and sub-partitioned by hash on customer_id Orders table is list partitioned by country and sub-partitioned by range on order_date Orders table is hash partitioned by country and sub-partitioned by hash on customer_id

The basic partitioning methods can be used in conjunction with the following partitioning extensions.

Partitioning Extensions

Partitioning Extension	Description	Sample Business Case
Interval Partitioning • Interval-[Range | List | Hash]	Extension to Range Partitioning. 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.	Orders table partitioned by order_date with a predefined daily interval, starting with '01-Jan-2013'
Auto List Partitioning	Extension to List Partitioning. Defined through keyword AUTOMATIC, partitions are created automatically when a partition key value is inserted without having a matching partition. Only one 'starter partition' has to be created initially	Orders table list partitioned by country, with only a 'GERMANY' partition being pre-created.
Reference 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.	(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
Virtual column based Partitioning	Defined by any partition techniques where the partitioning key is based on a virtual column. Virtual columns are not stored on disk and only exist as metadata.	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.

CONCLUSION

Since its first introduction in Oracle 8.0 in 1997, Oracle continually enhances the functionality of Oracle Partitioning with every release, by either adding new partitioning techniques, enhancing the scalability, or extending the manageability and maintenance capabilities. The newest release of Oracle Database is no different.

Oracle Partitioning is for everybody. Partitioning can greatly enhance the manageability, performance, and availability of almost any database application. Since partitioning is transparent to the application, it can be easily implemented for any kind of application because no costly and time-consuming application changes are required.

ORACLE CORPORATION

Worldwide Headquarters
500 Oracle Parkway, Redwood Shores, CA 94065 USA

Worldwide Inquiries
TELE + 1.650.506.7000 + 1.800.ORACLE1
FAX + 1.650.506.7200
oracle.com

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com. Outside North America, find your local office at oracle.com/contact.

 blogs.oracle.com/oracle  facebook.com/oracle  twitter.com/oracle

Integrated Cloud Applications & Platform Services

White Paper **Oracle Partitioning**
February 2019February 2019
Author: [OPTIONAL]
Contributing Authors: [OPTIONAL]

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