# So you think you know everything about Partitioning?

**Hermann Bär**
**Partitioning Product Management**
**Oracle USA**
**Redwood Shores, CA USA**

**Schlüsselworte:**

Partitioning, Performance, large data volumes

## Introduction

Most of the people that are going to read this article will not even remember the days prior to the introduction of Oracle Partitioning, which dates back to the times before Oracle 8.0 got introduced (for the archeologists, this was back in 1997). Introducing Partitioning beginning with the basics is therefor not the goal of this paper; there is ample literature out there, both from Oracle and many Oracle experts. We rather want to cover some little edge details and lesser known functionality, providing a look under the hood here and there. Consequently this paper will be more a collection of self-enclosed tidbits than a paper that has a storyline and you have to read end-to-end. If there are more topics you can think of and that you want to covered, then please get in touch with the author.

## A closer look at composite partitioned tables

Composite partitioned tables were introduced in Oracle 8i with Range-Hash Partitioning and successively completed over the years. Today, with oracle Database 11g Release 2, Partitioning is offering all possible combinations of the basic partitioning strategies of Range, Hash, and List.

The main purpose of Partitioning is to model the data partitioning according to your business needs, and composite Partitioning is no exception here[1]. With the capability to model your data placement along two dimensions you'll automatically face the decision of what method to pick as the main, top-level Partitioning strategy and what will become the sub-level strategy. Is there even a difference? From a performance perspective? From a management perspective? Well, there is, and there isn't.

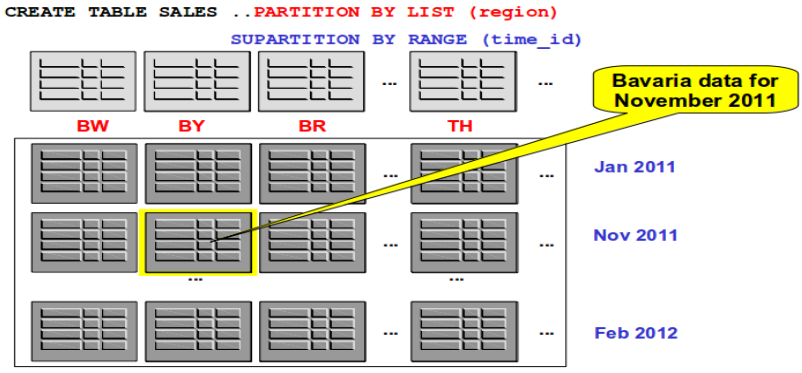


*Abb. 1: Bavaria in November 2011 in a List-Range partitioned table*

1 We are explicitly ignoring partitioning for performance here, namely for partitionwise joins

Conceptually, a composite partitioned table allows you to partition any given record based on two key values; it's like a two-dimensional space where the same pair of values will uniquely identify the data placement. Assuming you want to partition your data by moth and a distinct region identifier, it does not make any difference for the final ultimate data placement whether you choose a Range-List approach or a List-Range approach: data for Bavaria for the month of November 2011 will be stored in a subpartition that only contains data for Bavaria for November 2011.

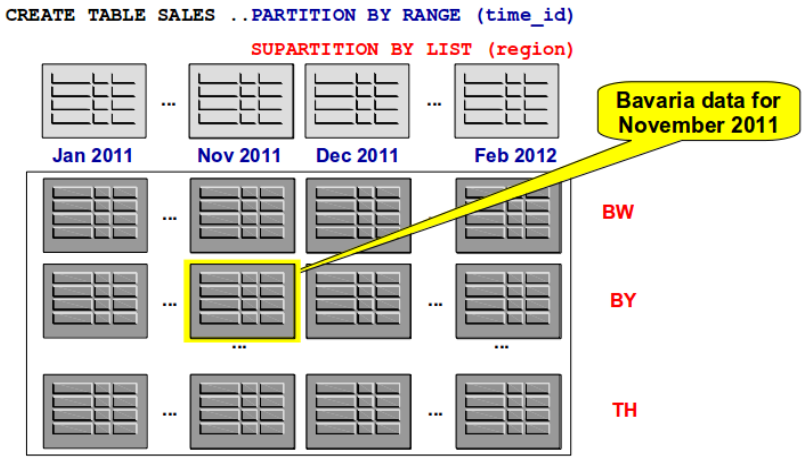


*Abb. 2: Bavaria in November 2011 in a Range-List partitioned table*

Since every data record is uniquely and deterministically placed into one – and only one – subpartition at any given point in time, there is no difference in behavior for the elimination of unnecessary partitions at data access time. Partition pruning is independent of the chosen composite order and will take place either way, pruning on one dimension - e.g. time range only – or on both, time and region.

So where's the difference? The difference is in the data maintenance and how to add new partitions or subpartitions to a table. Let's assume we have chosen a Range-List partitioned approach. As time goes by, we have to add partitions for new months. With a Range-List approach, the addition of a new partition is simple: we just do an ADD PARTITION command and get all the necessary subpartitions created in one go (you can influence the individual subpartition placement and shape), as shown in Abb. 3.

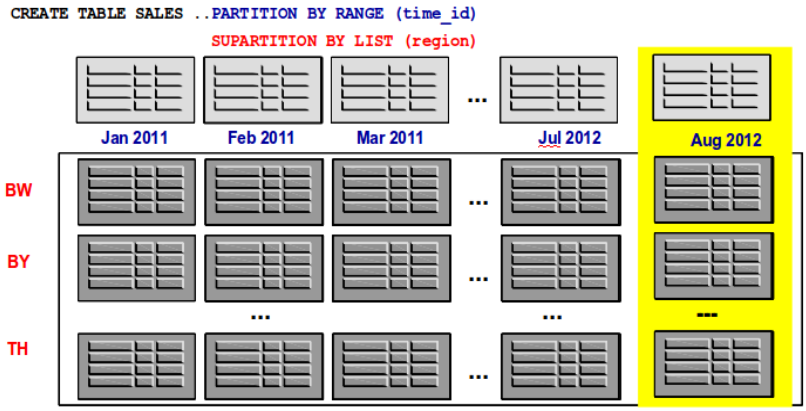


*Abb. 3: Add a month to a Range-List partitioned table*

It's one DDL command, all subpartitions are added, and the shape of all partitions stays the same; the table is fully symmetrical.

Now assume for a second that the German states are reshuffled, and a new state X get added. What does this mean for our composite partitioned table? If the new state is added for the future, life is simple: we just have to add one additional subpartition to the table for every future partition, as shown in Abb. 4.

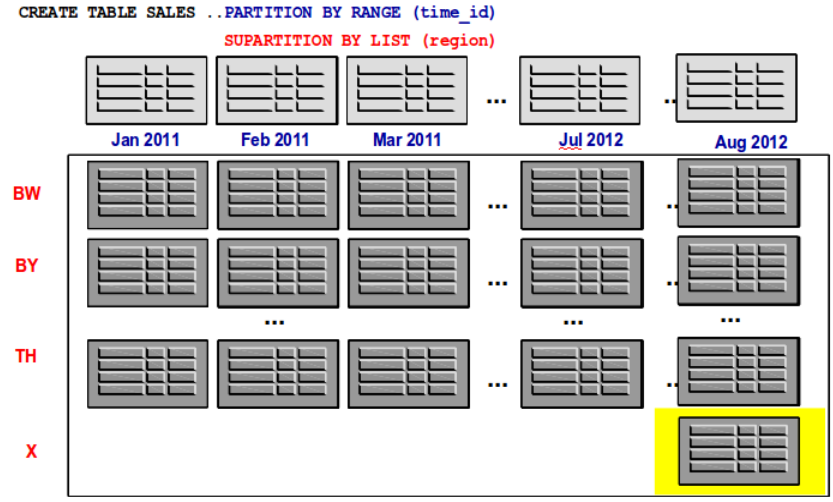


*Abb. 4: Add a new future state to a Range-List partitioned table*

That was easy. Oracle Partitioning can easily cope with this[2]. But what if the new state was formed by splitting an existing one and the business requires to have this reflected for the existing data as well. Even this is doable, but it requires more work, namely a single DDL operation for every existing monthly partition, as shown in Abb. 5. So if you have 3 years worth of data in your partitioned table and a monthly partitioning strategy, you have to add 36 subpartitions individually.

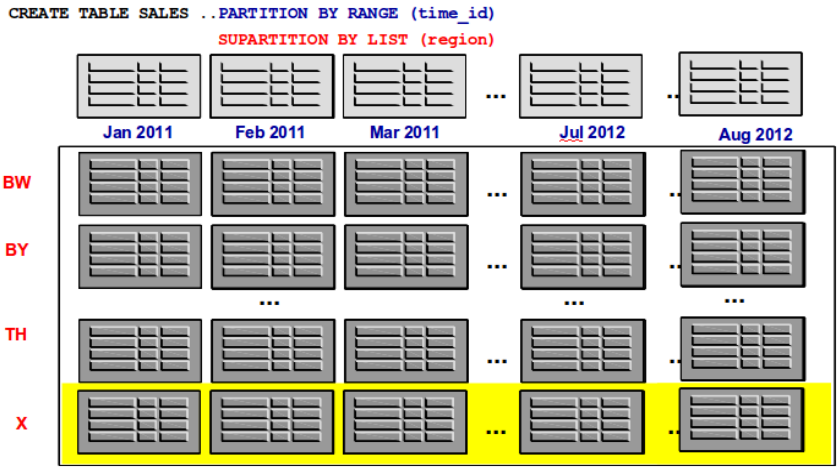


*Abb. 5: Add a new state to all partitions of a Range-List partitioned table*

2 The only "gotcha" would be if the subpartitioning strategy was chosen to enable partition-wise joins; introducing asymmetrical subpartitioning disables the capability of partition-wise joins.

The „bad news“ here is that you had to do n times more work to get it done for the existing composite partitioned table. The good news are that you can change the standard behavior for all future partition creations by changing the subpartition template for the composite partitioned table.

A subpartition template enables to specify the shape of future subpartitions, in terms of naming, data partitioning, and the storage attributes for individual subpartitions. Subpartition templates are stored as persistent metadata in the data dictionary; it can be (partially) overwritten with an explicit subpartition specification in the SQL command. Whenever possible, subpartition templates should be leveraged for composite partitioned tables.

Assuming we would have used a subpartition template for our table SALES, the SQL command to create the table would have looked like

```
CREATE TABLE sales (time_id DATE, region_id VARCHAR2(2), ...)
PARTITION BY RANGE (time_id)
SUBPARTITION BY LIST (region_id)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_bw VALUES ('BW'),
 SUBPARTITION sp_by VALUES ('BY'),
..
 SUBPARTITION sp_th VALUES ('TH'),
)
(PARTITION jan2011 VALUES LESS THAN('01-FEB-2011'), ...);
```

We would have seen subpartitions being created for every state defined in the subpartition template (to keep the spool output short I am only showing the creation of two subpartitions here)

```
SQL> SELECT partition_name, subpartition_name FROM user_tab_subpartitions;

PARTITION_NAME                 SUBPARTITION_NAME
------------------------------ ------------------------------
JAN2011                        JAN2011_SP_BY
JAN2011                        JAN2011_SP_BW
```

If we are adding a  new month to our table with an ADD PARTITION command, it looks as follows:

```
SQL> ALTER TABLE sales ADD PARTITION feb2011
     VALUES LESS THAN ('01-MAR-2011');

Table altered.

SQL> SELECT partition_name, subpartition_name FROM user_tab_subpartitions;

PARTITION_NAME                 SUBPARTITION_NAME
------------------------------ ------------------------------
FEB2011                        FEB2011_SP_BY
FEB2011                        FEB2011_SP_BW
JAN2011                        JAN2011_SP_BY
JAN2011                        JAN2011_SP_BW
```

You see that we automatically got two subpartitions created for the new partitions we added.

Flash forward in time, we now want to add our new state X into the mix, and we want this for all future partitions. By changing the subpartition template we can easily accomplish this:

```
SQL> ALTER TABLE sales
     SET SUBPARTITION TEMPLATE
     (SUBPARTITION sp_bw VALUES ('BW'),
      SUBPARTITION sp_by VALUES ('BY'),
      SUBPARTITION sp_x  VALUES ('X'));
Table altered.

SQL> ALTER TABLE sales ADD PARTITION mar2011
     VALUES LESS THAN ('01-APR-2011');

Table altered.

SQL> SELECT partition_name, subpartition_name FROM user_tab_subpartitions;

PARTITION_NAME                 SUBPARTITION_NAME
------------------------------ ------------------------------
FEB2011                        FEB2011_SP_BY
FEB2011                        FEB2011_SP_BW
JAN2011                        JAN2011_SP_BY
JAN2011                        JAN2011_SP_BW
MAR2011                        MAR2011_SP_X
MAR2011                        MAR2011_SP_BY
MAR2011                        MAR2011_SP_BW

7 rows selected.
```

Composite partitioning is a powerful two-dimensional partitioning strategy where the decision of what strategy to choose as top-level or sub-level partitioning should be predominantly a decision based on the data maintenance aspects for the partitioned object. For composite partitioned tables it is not a performance-based decision (ignoring the enablement of static partition-wise joins), since Oracle's pruning capabilities are independent of whether a key is defined as top- or sub-level partitioning key. Whenever you use composite partitioned tables you should consider using subpartition templates.

**So what's the deal with System Partitioning?**

The majority of Oracle customers should not care about System Partitioning. System Partitioning was introduced specifically for software developers dealing with non-traditional data types and indexing techniques, both internally within Oracle as well as externally; it got specifically introduced in Oracle Database 11g to enable the development of partitioned domain indexes.

The most fundamental difference between System Partitioning and all the other partitioning methods is that System Partitioning does not have any partitioning keys. Consequently the distribution/mapping of the rows to a particular partition is not defined and not known by the database. System Partitioning simply creates the number of partitions specified. There is no concept of composite partitioning, simply because there is no partitioning context to begin with.

As a logical consequence the database is not able to just insert data into a system partitioned table. Which partition shall the database choose for a record? You have to explicitly specify the partition to which a row maps by using the partition extended syntax when inserting the row.

Last but not least, System partitioned tables do not have a partitioning key, the usual performance benefits of partitioned tables will not be available for system partitioned tables, i.e. there is no support for partition pruning, partition wise joins etc.

In a nutshell: System partitioning is most likely not for you, but it was created for a reason: to provide optimal database and partitioning support for additional use cases beyond the traditional relational boundaries.

**Summary**
Partitioning is mature and proven technology for more than a decade. Tens of thousands of customers rely on Oracle partitioning every single minute to provide the performance and manageability required by the business. Nevertheless, despite its broad usage there are many little details that will help you to either get a better understanding of Oracle's powerful Partitioning functionality or even enable you to address your business requirements more optimally looking forward.

Bitte fügen Sie Ihre Kontaktadresse hinzu.

**Kontaktadresse:**

**Hermann Bär**
Oracle USA
400 Oracle Parkway
Redwood Shores, CA 94065, USA

| | |
|---|---|
| Telefon: | +1 (0) 650.506.6833 |
| Fax: | +1 (0) 650.506.6833 |
| E-Mail | hermann.baer@oracle.com |
| Internet: | www.oracle.com |