Partitioning for more Performance

Sebastian Winkler

CarajanDB GmbH

CarajanDB

- **Oracle professionals with more than 25 years of experience**
- **Located near Cologne**
- **Specialized in**
  - Oracle Database Administration
  - High Availability (RAC, Data Guard, Failsafe, etc.)
  - Oracle Standard Edition
  - Oracle Migrations (i.e. Unicode, Standard Edition)
  - Replication (Goldengate, SharePlex, Dbvisit)
  - Database Cloning (Actifio, Delphix, CloneDB)
  - Performance Optimization
- **Trainings and Workshops (Oracle, Toad)**

Agenda

- **Overview**
  - Partitioning - Why and how?
- **Decision guidance**
  - Advantages, objectives and analysis
- **Partitioning types**
  - Standard, composite and extensions
- **Conclusion**
  - New features 12c and resume

CARAJANDB

Overview

- Why?
- How?

Partitioning - Why?

- **(Very) Large Databases**
  - 10-20 years ago large = today normally
    - Very Large Database and Data Warehouse
  - Bigger databases = bigger challenges in business
    - Size, transaction volume, activity, parallel user, query complexity, response time, availability
  - Standard technologies not sufficient

Partitioning one of the most important technologies

Partitioning - Why?

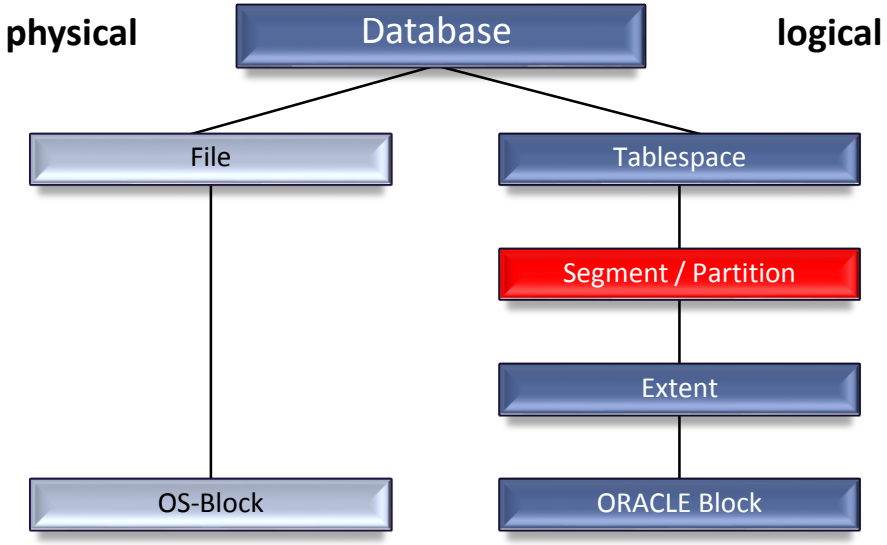
- **Partitioning as benefit for …**

- **Administration of (very) large databases**
  - Simplified and manageable by delete, add, move and archive of partitions

- **Oracle Optimizer**
  - More efficient request handling caused by partition pruning, partition wise joins and parallelization

How Partitioning works

- **Partitioning works on segment level**
  - **A single segment gets**
    - Tables, Indexes
  - **… seperated by a certain rule**
    - Partition criteria set at table creation
    - Range of values, list of values or hash values
  - **… into several segments.**
    - Partitions
    - New data will be sorted automatically



CARAJANDB

Decision Guidance

- Advantages
- Objectives
- Analysis

Advantages

- **Partitions are physically independant**
  - storing and managing in separate tablespaces with different storage parameters possible (e.g. PCTFREE, COMPRESSION)
- **Transparent**
  - from application perspective all data is read- and writeable via the name of the partitioned table
- **Optimizer**
  - knows the physical allocation of the table
  - is able to generate optimized access paths
- **Indexes**
  - associated indexes can also be – dependent or independent – partitioned
  - smaller partition segments allow better usability and more efficient constructions

Access optimization

- **Partition Pruning**
  - Partitions a smaller unit compared to the logical object
  - Depending on the WHERE clause a table-scan-operation can profit by excluding of certain partitions
- **Flatter Index-Trees**
  - Partition specific built Index-Trees can be more flat
  - Fewer layers reduce I/O overhead on access
- **Partition wise table operations**
  - Joins can be reduced to corresponding partitions
  - Partition wise joins can run parallel on physically separated data

Optimization of administrative tasks

- **Simplification and shortening**
  - Partitions can be added, splitted and deleted individually
  - Exchange of data between tables and partitions over efficient switch operations
    - EXCHANGE PARTITION (without locking)

Individual Case Analysis

- **Benefits possible in principle**
  - Concrete benefit depends on individual case
  - Requirement: careful analysis before implementation
- **Prioritize optimization objectives**
  - More efficient management or access optimization?
- **Object Size**
  - Makes partitioning of a table with a few lines sense?

Individual Case Analysis

- **Type of access**
  - OLTP applications generally benefit less by very selective Index accesses
  - Batch applications or DWH with scans on large areas benefit from denser partition areas and flatter index structures

- **Type of data management**
  - For example is it possible to synchronize the partitioning criteria with the deletion cycle of a Rolling-Window-Application

Individual Case Analysis

- **Partitioning is recommended …**
  - If all to a prioritization matching criteria are fulfilled.
  - Partitioning can generally be considered for large tables - in the 2-digit GB range
  - If large amounts of data should be provided over many years (In Database Archiving)

CARAJANDB

Partitioning Types

- Range, List, Hash
- Composite
- Extension

Overview – Partitioning Types

- **Standard technologies**
  - Value ranges - Range
  - Value lists - List
  - Hash values - Hash
- **Extensions**
  - Intervall, Referential, Virtual Columns
- **Composite**
  - Composite- or Subpartitioning
  - Two types are hierarchically combined

Overview – Partitioning Types

- **Composite- or Subpartitioning**
  - Range-Range
  - Range-List
  - Range-Hash
  - List-List
  - List-Range
  - List-Hash
  - Hash-Hash
  - Hash-List
  - Hash-Range

Overview – Partitioning Types

- **Extended Partitioning Methods**
  - Interval Partitioning
    - Interval
    - Interval-Range
    - Interval-List
    - Interval-Hash
  - Reference Partitioning
  - Virtual column based Partitioning

Scalability

- **Maximum partitions per segment**
  - until 10gR1 64K-1
    = 65.535 partitions
  - since 10gR2 1024K-1 until today 12c
    = 1.048.575 partitions as well as subpartitions

Partitions	Maximum length of linear partitioning key	4 KB - overhead
Partitions	Maximum number of columns in partition key	16 columns
Partitions	Maximum number of partitions allowed per table or index	1024K - 1
Subpartitions	Maximum number of subpartitions in a composite partitioned table	1024K - 1

- **Manageability determines the number and granularity**
  - Single partitions should not grow beyond the 2-digit GB range
  - Also determines the granularity of the partitioning criterion e.g. week, month, quarter or year

Scalability

- **Partitioning columns**
  - Should be unalterable or at least rarely changeable after the initial insertion of the record
  - Since 11g - virtual columns can be used
    - Substring of a column, functional expressions of several columns

- **Attention**
  - Subsequent modification of such columns can mean another partition - another partition can mean physically moving the data!

Range Partitioning

- **Partitioning by range**
  - Especially appropriate for time-sensitive data
  - Partition per week, month or quarter for order date column
  - Probably most common variant in practice

- **Special case Interval Partitioning**
  - Since 11g
  - Creates a new partition automatically when needed

List Partitioning

- **Partitioning by value list**
  - Useful e.g. for data allocated by area for example regions, state column
  - Not useful for frequently changing columns - after the initial insertion
    - e.g. Status column

Hash Partitioning

- **Partitioning by hash values**
  - The actual value of the partitioning column does not depend
  - The aim is to create „any“ partitions
  - Commonly used for subpartitions – Range-Hash, List-Hash

- **Example Euqi Partitioning**
  - Two tables are hash partitioned by their join column
  - Optimizer benefits from partition wise joins
  - Partition pruning can not be used here

Reference Partitioning

- **Partitioning according to a parent table**
  - It can be sorted by columns that appear only in a parent table per foreign key
  - For example an order table can be partitioned by the state of the client
  - Logical continuation and expansion of equi partitioning

- **Benefit**
  - Easier administration
  - Supports partition wise joins of according table

How to partition?

- **Which type of partitioning?**
  - What is the typicall way to access the data?

- **Partitioning column must appear in the WHERE clause!**
  - Only this provides that the Optimizer can use partition pruning and partition wise joins
    - Instead of the entire table only single partitions where read or joined – best case only one single partition

Partitioning anyway?

- **Basics**
  - If – can only be decided when creating a table
  - Inserting a partition for a non-partitioned table - means complete reconstruction
  - Modify the type of partitioning - means complete reconstruction
  - Subsequent adding, deleting, splitting or merging of a partition of an already partitioned table possible without any problems
  - Heavy revisable decisions analyze precisely in advance!

CARAJANDB

Partitioning Types

Range Partitioning

Range Partitioning

- **Partitioning column**
  - Large amounts of data often contain a date field
  - Data Warehouse often detail data (fact tables) with date
  - Appropriate if mostly unchangeable

  - Partitioning of tables and associated indexes by date

Range Partitioning

- **Example: Order Table**
- **SQL>**


```
CREATE TABLE order_part
(
orderid       NUMBER(10) PRIMARY KEY,
clientid      NUMBER(10) REFERENCES clients (clientid),
orderdate     DATE,
deliverydate  DATE,
orderstatus   CHAR(1) REFERENCES status (statusid)
)
PARTITION BY RANGE (orderdate)
(
  PARTITION q1_15 VALUES LESS THAN (to_date('01042015','DDMMYYYY'))
  TABLESPACE ts01,
  PARTITION q2_15 VALUES LESS THAN (to_date('01072015','DDMMYYYY'))
  TABLESPACE ts02,
  PARTITION q3_15 VALUES LESS THAN (to_date('01102015','DDMMYYYY'))
  TABLESPACE ts03,
  PARTITION q4_15 VALUES LESS THAN (to_date('01012016','DDMMYYYY'))
  TABLESPACE ts04,
  PARTITION q1_16 VALUES LESS THAN (MAXVALUE)
  TABLESPACE ts01
);
```

Range Partitioning

- **Example: Order Table**
  - Partitioning criteria are assigned by strictly defined range of values of one or more columns
    - `PARTITION BY RANGE (orderdate)`
  - Each partition gets a corresponding limit via VALUES clause
    - `PARTITION q1_15 VALUES LESS THAN (to_date('01042015','DDMMYYYY')) TABLESPACE ts01,`
  - Easy to understand – which data is in which partition
  - Optimizer can thus exclude certain partitions and restrict access to individual partitions

Range Partitioning

- **Example: Order Table**
  - Related equi-partitioned Index

- **SQL>**

```
CREATE INDEX order_part_orderdate
ON order_part (orderdate)
LOCAL
(
  PARTITION q1_15 TABLESPACE idxts01,
  PARTITION q2_15 TABLESPACE idxts02,
  PARTITION q3_15 TABLESPACE idxts03,
  PARTITION q4_15 TABLESPACE idxts04,
  PARTITION q1_16 TABLESPACE idxts01
);
```

Range Partitioning

- **Example: Order Table**
  - Advantage in query processing - it is accessed solely to partition 3 and 4 = Partition Pruning
  - Important: approximately equal distribution of data

- **SQL>**

```
SELECT orderid, orderstatus
FROM order_part
WHERE orderdate BETWEEN TO_DATE('01.09.2015','DD.MM.YYYY')
                    AND TO_DATE('05.11.2015','DD.MM.YYYY');
```

```
-------------------------------------------------------------------------
| Id  | Operation                          | Name             | Pstart| Pstop |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                  |       |       |
|   1 |  PARTITION RANGE ITERATOR          |                  |     3 |     4 |
|   2 |   TABLE ACCESS BY LOCAL INDEX ROWID| ORDER_PART       |     3 |     4 |
|*  3 |    INDEX RANGE SCAN                | ORDER_ORDERDATE  |     3 |     4 |
-------------------------------------------------------------------------
```

Range Partitioning

- **DROP PARTITION or EXCHANGE**
  - Advantage: old data can be deleted or archived by DDL operation
  - Rolling-Window
    - Time window for viewing shifts
    - 1st quarter 2015 swapped (e.g. to read-only media) and a new for Q1 2016 will be created

Range Partitioning

- **SPLIT PARTITION**
  - Caution: SPLIT causes all data of a partition are physically moved
  - Example: First, Q1 2016 will be created and only at the end of the quarter, Q2 2016 will be generated by a split
    - Q1 and Q2 will be completely re-created - undo and redo log information is generated
  - Better: always use a "dummy" partition - then carry on splitting the empty partition

Range Partitioning

- If it is guaranteed that no data is running into the MAXVALUE partition it can be omitted
  - `PARTITION q1_16 VALUES LESS THAN (MAXVALUE) TABLESPACE ts01);`

- **ADD PARTITION**
  - For creating a new partition
  - SQL>

```
ALTER TABLE order_part
ADD PARTITION q2_16
VALUES LESS THAN (TO_DATE('01072016','DDMMYYYY'))
TABLESPACE ts02;
```

  - Command can only be used if no MAXVALUE exists - adds a partition at upper partition boundary

CARAJANDB

Partitioning Types

Interval Partitioning

Interval Partitioning

- **Rolling-Window**
  - Creating new range partitions can be automated since 11g
  - Possible for both new and existing range-partitioned tables

  - Requirements:
    - Partitioned by a single column of type DATE or NUMBER
    - No MAXVALUE partition as an upper limit

Interval Partitioning

- **Start-partition**
  - Partition with the highest existing partition border
  - Starting point for automatically generated partitions

- **INTERVAL Clause**
  - `INTERVAL (NUMTOYMINTERVAL(3,'MONTH'))`
  - Specifies the partition width

- **STORE IN Clause**
  - `STORE IN (ts01,ts02)`
  - Optional - in which tablespaces should partitions be created
  - Tablespace must exist at least when data gets inserted

Interval Partitioning

- SQL>

```
CREATE TABLE order_part
(
orderid       NUMBER(10) PRIMARY KEY,
clientid      NUMBER(10) REFERENCES clients (clientid),
orderdate     DATE,
deliverydate  DATE,
orderstatus   CHAR(1) REFERENCES status (statusid)
)
PARTITION BY RANGE (orderdate)
INTERVAL (NUMTOYMINTERVAL(3,'MONTH'))
STORE IN (ts01,ts02)
(
  PARTITION q1_15 VALUES LESS THAN (TO_DATE('01042015','DDMMYYYY'))
  TABLESPACE ts01,
  PARTITION q2_15 VALUES LESS THAN (TO_DATE('01072015','DDMMYYYY'))
  TABLESPACE ts02
);
```

Interval Partitioning

- **Creation with an existing order table with range partitioning**

- **SQL>**

```
ALTER TABLE order_part
SET INTERVAL (NUMTOYMINTERVAL(3,'MONTH'));

ALTER TABLE order_part
SET STORE IN (ts01,ts02);
```

  - The same way partition width and used tablespaces can be adjusted

  - **Note: Affects only future partitions**

Interval Partitioning

- **Example: Insertion of new data**

- SQL>
```
SELECT partition_name AS pname, high_value,
tablespcace_name AS tsname, interval
FROM user_tab_partitions
WHERE table_name = 'ORDER_PART';
```

- Status quo:

```
PNAME    HIGH_VALUE                          TNAME INTERVAL
-------- ----------------------------------  ----- -----
Q1_15    TO_DATE(' 2015-04-01 00:00:00'      TS01  NO
Q2_15    TO_DATE(' 2015-07-01 00:00:00'      TS02  NO
```

Interval Partitioning

- **Example: Insertion of new data**

- **SQL>**

```
INSERT INTO order_part (orderid, orderdate)
VALUES (500001, TO_DATE ('01.07.2015', 'DD.MM.YYYY'));
```

  - 1st change:

```
PNAME    HIGH_VALUE                          TNAME  INTERVAL
-------- ----------------------------------  -----  -----
Q1_15    TO_DATE(' 2015-04-01 00:00:00'      TS01   NO
Q2_15    TO_DATE(' 2015-07-01 00:00:00'      TS02   NO
SYS_P501 TO_DATE(' 2015-10-01 00:00:00'      TS01   YES
```

  - INTERVAL column provides information on manually and automatically generated partition

Interval Partitioning

- **Example: Insertion of new data**

- **SQL>**

```
INSERT INTO order_part (orderid, orderdate)
VALUES (500002, TO_DATE ('01.01.2016', 'DD.MM.YYYY'));
```

  - 2nd change:

```
PNAME    HIGH_VALUE                          TNAME INTERVAL
-------- ----------------------------------- ----- -----
Q1_15    TO_DATE(' 2015-04-01 00:00:00'      TS01  NO
Q2_15    TO_DATE(' 2015-07-01 00:00:00'      TS02  NO
SYS_P501 TO_DATE(' 2015-10-01 00:00:00'      TS01  YES
SYS_P502 TO_DATE(' 2016-04-01 00:00:00'      TS01  YES
```

  - New partition is created with system-generated names
  - Behaviour can not be changed - only subsequent change possible

Interval Partitioning

- **Example: Insertion of new data**

- **SQL>**

```
INSERT INTO order_part (orderid, orderdate)
VALUES (500003, TO_DATE ('01.10.2015', 'DD.MM.YYYY'));
ROLLBACK;
```

  - 3rd Change:

```
PNAME    HIGH_VALUE                          TNAME INTERVAL
-------- ----------------------------------  ----- -----
Q1_15    TO_DATE(' 2015-04-01 00:00:00'      TS01  NO
Q2_15    TO_DATE(' 2015-07-01 00:00:00'      TS02  NO
SYS_P501 TO_DATE(' 2015-10-01 00:00:00'      TS01  YES
SYS_P503 TO_DATE(' 2016-01-01 00:00:00'      TS02  YES
SYS_P502 TO_DATE(' 2016-04-01 00:00:00'      TS01  YES
```

  - At a Rollback automatically generated partitions remains

Interval Partitioning

- **Example: Insertion of new data**
  - At a record beyond the next partition to be created, not all partitions in between are created, but only one large partition - 2nd change:


```
PNAME    HIGH_VALUE                          TNAME INTERVAL
-------- ----------------------------------  ----- -----
Q1_15    TO_DATE(' 2015-04-01 00:00:00'  TS01  NO
Q2_15    TO_DATE(' 2015-07-01 00:00:00'  TS02  NO
SYS_P501 TO_DATE(' 2015-10-01 00:00:00'  TS01  YES
SYS_P502 TO_DATE(' 2016-04-01 00:00:00'  TS01  YES
```

  - If necessary, it automatically gets splitted without physical move or invalidation of global indexes - 3rd change:

```
PNAME    HIGH_VALUE                          TNAME INTERVAL
-------- ----------------------------------  ----- -----
Q1_15    TO_DATE(' 2015-04-01 00:00:00'  TS01  NO
Q2_15    TO_DATE(' 2015-07-01 00:00:00'  TS02  NO
SYS_P501 TO_DATE(' 2015-10-01 00:00:00'  TS01  YES
SYS_P503 TO_DATE(' 2016-01-01 00:00:00'  TS02  YES
SYS_P502 TO_DATE(' 2016-04-01 00:00:00'  TS01  YES
```

Interval Partitioning

- **Automates only creation of >new< partitions**
  - Old partitions can not be deleted automatically
  - Management of >old< partitions will remain with the administrator

- **Create partitions manually now impossible**

- **Turn off automatic creation of partitions possible**
  - SQL>

```
ALTER TABLE order_part
SET INTERVAL();
```

CARAJANDB

Partitioning Types

List Partitioning

List Partitioning

- **Information is grouped by regions or other unordered lists of values**
  - relatively steady
  - Criteria is often restricted in queries
  - Well combinable with virtual columns

List Partitioning

- **Example: Products table**
  - SQL>

```
CREATE TABLE products_part
(
prodid       NUMBER(10) PRIMARY KEY,
pgrid        NUMBER(10),
productname  VARCHAR(255),
pricebuy     NUMBER (10,2),
pricesale    NUMBER (10,2),
pgrid_2      VARCHAR(2) AS (SUBSTR(TO_CHAR(pgrid),1,2))
)
PARTITION BY LIST (pgrid_2)
(
PARTITION p01 VALUES ('10','11','12','13','14','15') TABLESPACE ts01,
PARTITION p02 VALUES ('16','17','18','19') TABLESPACE ts02,
PARTITION p03 VALUES ('20','21','22','23','24','25','26') TABLESPACE ts03,
PARTITION p04 VALUES ('27','28','29') TABLESPACE ts04,
PARTITION p05 VALUES ('30','31','32','33','34','35') TABLESPACE ts01,
PARTITION p_rest VALUES (DEFAULT) TABLESPACE ts04
);
```

List Partitioning

- **Example: Products table**
  - No combinations of columns can be used - only a single column (wait for 12.2)
    - `PARTITION BY LIST (pgrid_2)`
  - Last partition forms the rest partition and accepts all data not explicitly mentioned
    - `PARTITION p_rest VALUES (DEFAULT) TABLESPACE ts04,`

List Partitioning

- **Example: Products table**
  - Each value that can occur should be assigned
  - Without allocation or DEFAULT table - Error:


```
ORA-14400: inserted partition key does not map to any partition
```

  - Add an extra value:
    - SQL>

```
ALTER TABLE products_part
MODIFY PARTITION p05 ADD VALUES ('36');
```

List Partitioning

- **Partition Pruning**
  - Requirement: Column or definition expression of the virtual column appears in the query:
  - SQL>

```
SELECT *
FROM products_part
WHERE pgrid='201342';
```

```
-----------------------------------------------------------
| Id  | Operation           | Name          | Pstart| Pstop|
-----------------------------------------------------------
|   0 | SELECT STATEMENT    |               |       |      |
|   1 |  PARTITION LIST ALL|               |     1 |    6 |
|*  2 |   TABLE ACCESS FULL| PRODUCTS_PART |     1 |    6 |
-----------------------------------------------------------
```

  - Optimizer can not consider partitioning

List Partitioning

- **Partition Pruning**
  - Requirement: Column or definition expression of the virtual column appears in the query:
  - SQL>

```
SELECT *
FROM products_part
WHERE SUBSTR(TO_CHAR(pgrid),1,2)='20';
```

```
-----------------------------------------------------------
| Id  | Operation                | Name          | Pstart| Pstop|
-----------------------------------------------------------
|   0 | SELECT STATEMENT         |               |       |      |
|   1 |  PARTITION LIST SINGLE|               |   KEY |  KEY |
|*  2 |   TABLE ACCESS FULL    | PRODUCTS_PART |     3 |    3 |
-----------------------------------------------------------
```

  - Optimizer considers filter condition
  - Typical form of query decides on partitioning

CARAJANDB

Partitioning Types

Hash Partitioning

Hash Partitioning

- **It is not possible to find any meaningful value lists or value ranges**

- **Recommendation:**
  - Always two or more tables should be equi-partitioned on their join columns
  - Hash partitioning mainly on foreign key columns and their parent primary key or unique columns

- **Benefit:**
  - Partition wise joins
  - No partition pruning
  - Nevertheless considerable advantage for Optimizer on large tables

Hash Partitioning

- **Characteristics:**
  - Internal hash function makes allocation
  - Number of partitions can be determined
  - Number of partitions should be a multiple of 2
  - Not foreseeable which column value is stored in which partition
  - Makes deletion or archiving of hash partitions worthless
  - Columns with very high selectivity - optimally unique values - ensure a homogeneous distribution

Hash Partitioning

- **Example: Clients and Order table**
  - SQL>


```
CREATE TABLE clients_part
(
clientid    NUMBER(10) PRIMARY KEY,
title       VARCHAR(5),
firstname   VARCHAR(50),
lastname    VARCHAR(50),
birthday    DATE,
image       BLOB
)
PARTITION BY HASH (clientid)
PARTITIONS 16 STORE IN (ts01,ts02,ts03,ts04);
```

```
CREATE TABLE order_part
(
orderid       NUMBER(10) PRIMARY KEY,
clientid      NUMBER(10) REFERENCES clients_part
(clientid),
orderdate     DATE,
deliverydate  DATE,
orderstatus   CHAR(1) REFERENCES status (statusid)
)
PARTITION BY HASH (clientid)
PARTITION 16 STORE IN (ts01,ts02,ts03,ts04);
```

```
CREATE INDEX order_part_clientid
ON order_part (clientid)
LOCAL;
```

Hash Partitioning

- **Example: Clients and order table**
  - Joining a single client record to the according order data now does not need to access the whole table, but only the associated order partition and the associated index – here partition 7:
  - SQL>

```
SELECT p.firstname, p.lastname, a.orderid, a.orderstatus
FROM clients_part p, order_part a
WHERE p.clientid = a.clientid
AND p.clientid = 100127;
```

```
-------------------------------------------------------------------------------
| Id  | Operation                           | Name               | Pstart| Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                    |       |       |
|   1 |  NESTED LOOPS                       |                    |       |       |
|   2 |   TABLE ACCESS BY GLOBAL INDEX ROWID| CLIENTS_PART       |     7 |     7 |
|*  3 |    INDEX UNIQUE SCAN                | SYS_C009627        |       |       |
|   4 |  PARTITION HASH SINGLE              |                    |     7 |     7 |
|   5 |   TABLE ACCESS BY GLOBAL INDEX ROWID| ORDER_PART         |     7 |     7 |
|*  6 |    INDEX RANGE SCAN                 | ORDER_PART_PERSID  |     7 |     7 |
-------------------------------------------------------------------------------
```

CARAJANDB

Partitioning Types

Reference Partitioning

Reference Partitioning

- **Equi partitioning of dependent tables**
  - available since 11g
  - based on a parent table via foreign key that is already partitioned
  - No NULL Values
    - All affected foreign key columns must be provided with NOT NULL constraints
    - Otherwise Error:

```
ORA-14652: reference partitioning foreign key is not supported
```

- **Example: Order and positions table**
  - Frequent joins between two tables make a congruent partitioning useful

Reference Partitioning

- **Example: Order and Positions table**
  - At defenition the name of the foreign key constraint must be specified


```
CREATE TABLE order_part
(
orderid       NUMBER(10) PRIMARY KEY,
clientid      NUMBER(10) REFERENCES clients (clientid),
orderdate     DATE,
deliverydate  DATE,
orderstatus   CHAR(1) REFERENCES status (statusid)
)
PARTITION BY RANGE (orderdate)
(
  PARTITION q1_15 VALUES LESS THAN (TO_DATE('01042015','DDMMYYYY'))
  TABLESPACE ts01,
  PARTITION q2_15 VALUES LESS THAN (TO_DATE('01072015','DDMMYYYY'))
  TABLESPACE ts02,
  PARTITION q3_15 VALUES LESS THAN (TO_DATE('01102015','DDMMYYYY'))
  TABLESPACE ts03,
  PARTITION q4_15 VALUES LESS THAN (TO_DATE('01012016','DDMMYYYY'))
  TABLESPACE ts04,
  PARTITION q1_16 VALUES LESS THAN (MAXVALUE)
  TABLESPACE ts01
);
```

```
CREATE TABLE positions_part
(
orderid      NUMBER(10) NOT NULL,
posid        NUMBER(10) NOT NULL,
prodid       NUMBER(10) REFERENCES products (prodid),
quantity     NUMBER(10,2),
unitprice    NUMBER(7,2),
CONSTRAINT fk_positions_part_order FOREIGN KEY (orderid)
REFERENCES order_part (orderid)
)
PARTITION BY REFERENCE (fk_positions_part_order);
```

Reference Partitioning

- **Example: Order and positions table**
  - Order and position table are now Equi-Partitions
  - Each repartitioning (adding, deleting, splitting or merging) of the orders table automatically inherited to the positions table
  - Administrator does not need to pay attention to consistent partitioning
  - Partitions of the position table are by default stored in the same tablespace as the corresponding partitions of the orders table
  - Individual repartitioning of the position table, however, is no longer possible

Reference Partitioning

- **Can branch out randomly wide and deep**
  - Another table that depends by foreign key from the Positions table
  - Other tables that depends by foreign key from the Orders table
  - Repartitioning of the top table cascade to all directly or indirectly dependent tables

- **Parent Interval partitioned table with 12c**
  - Until 11gR2 not possible
  - Minimizes the administrative effort to a minimum
  - INTERVAL-REFERENCE-Partitioning

Reference Partitioning

- **Example: Order and positions table**
  - Utilization of equi- or reference partitioning at joining
  - SQL>

```
SELECT *
FROM order_part a, positions_part p
WHERE a.orderid = p.orderid
AND a.orderdate BETWEEN TO_DATE('01.09.2015','DD.MM.YYYY')
                AND     TO_DATE('05.11.2015','DD.MM.YYYY');
```

```
-------------------------------------------------------------------
| Id  | Operation                | Name           | Pstart| Pstop |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                |       |       |
|   1 |  PARTITION RANGE ITERATOR|                |     3 |     4 |
|   2 |   NESTED LOOPS           |                |       |       |
|*  3 |    TABLE ACCESS FULL     | ORDER_PART     |     3 |     4 |
|*  4 |    TABLE ACCESS FULL     | POSITIONS_PART |     3 |     4 |
-------------------------------------------------------------------
```

  - Optimizer is limited to partition 3 and 4 - taken over by reference for the positions table

CARAJANDB

Partitioning Types

Subpartitioning

Subpartitioning

- **Linking two types of partitioning**
  - makes sense for very large tables
  - At first instance, range or list partition
  - In the second instance sub partitions by range, list or hash method
    - Within each partition
  - Each subpartition is independent regarding storage
    - Can be equipped with its own storage parameters
  - Administrative Operations
    - Adding, deleting, splitting, merging or physical movement available for each subpartition
  - Optimization options
    - Partition Pruning and partition wise joins for each subpartition available

Subpartitioning

- **Example: Clients and order table**
  - Generally incompatible – but with subpartitioning by clientid partition wise joins can be used
  - SQL>

```
CREATE TABLE clients_part
(
clientid NUMBER(10) PRIMARY KEY,
title VARCHAR(5),
firstname VARCHAR(50),
lastname VARCHAR(50),
birthday DATE,
image BLOB
)
PARTITION BY HASH (clientid)
PARTITIONS 16 STORE IN (ts01,ts02,ts03,ts04);
```

```
CREATE TABLE order_part
(
orderid        NUMBER(10) PRIMARY KEY,
clientid       NUMBER(10) REFERENCES clients (clientid),
orderdate      DATE,
deliverydate    DATE,
orderstatus    CHAR(1) REFERENCES status (statusid)
)
PARTITION BY RANGE (orderdate)
SUBPARTITION BY HASH (clientid)
SUBPARTITIONS 16 STORE IN (ts01,ts02,ts03,ts04)
(
  PARTITION q1_15 VALUES LESS THAN (TO_DATE('01042015','DDMMYYYY'))
  TABLESPACE ts01,
  PARTITION q2_15 VALUES LESS THAN (TO_DATE('01072015','DDMMYYYY'))
  TABLESPACE ts02,
  PARTITION q3_15 VALUES LESS THAN (TO_DATE('01102015','DDMMYYYY'))
  TABLESPACE ts03,
  PARTITION q4_15 VALUES LESS THAN (TO_DATE('01012016','DDMMYYYY'))
  TABLESPACE ts04,
  PARTITION q1_16 VALUES LESS THAN (MAXVALUE)
  TABLESPACE ts01
);
```

Subpartitioning

- **Example: Clients and order table**
  - Optimizer recognizes that only suitable HASH partitions of the clients table must be joined with the 5 matching subpartitions of the Orders table
  - SQL>


```
SELECT p.firstname, p.lastname, a.orderid, a.orderstatus
FROM clients_part p, order_part a
WHERE p.clientid = a.clientid
AND p.clientid = 100127;
```

```
-------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Pstart| Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |       |       |
|   1 |  NESTED LOOPS                       |               |       |       |
|   2 |   TABLE ACCESS BY GLOBAL INDEX ROWID| CLIENTS_PART  |     7 |     7 |
|*  3 |    INDEX UNIQUE SCAN                | SYS_C009661   |       |       |
|   4 |  PARTITION RANGE ALL                |               |     1 |     5 |
|   5 |   PARTITION HASH SINGLE             | ORDER_PART    |     7 |     7 |
|*  6 |    TABLE ACCESS FULL                | ORDER_PART    |       |       |
-------------------------------------------------------------------------------
```

CARAJANDB

Partitioning Types

Index Partitioning

Index Partitioning

- **Partitioning indexes**
  - Partitioned indexes can be created for partitioned and non-partitioned tables
  - Partitioned tables can also be provided with non-partitioned indexes

  - B*Tree and bitmap indexes can be used
  - Restrictions:
    - non-partitioned tables and partitioned bitmap indexes is not possible
    - Bitmap indexes must be partitioned according to the same criteria as the associated partitioned table

Index Partitioning

- **Types**
  - Attribute pairs
    - Local and Global Indexes
    - Prefixed and Non-Prefixed Indexes

  - Local prefixed Index
  - Local non-prefixed Index
  - Global prefixed Index
  - Global non-prefixed Index – not supported

Index Partitioning

- **Local indexes**
  - Index is local if it has the same partitioning criteria as the underlying table - Equi-partitioning
  - 1:1 mapping between table and index partition
    - Partition changes to the table are transferred 1:1 to the corresponding index
    - Local index is automatically included in all operations on partitioned tables
  - Particularly suitable for Rolling Window
  - creation with LOCAL

Index Partitioning

- **Global Indexes**
  - Index is global if it has an independent partitioning criteria to the table
  - or is not partitioned anyway
  - Operations on the associated partitioned table are not directly transferred
    - Accesses on a table partition can span multiple index partitions and vice versa
  - Only Range or Hash index partitioning possible
    - List or Composite index partitioning not possible
  - creation with GLOBAL

Index Partitioning

- **Prefixed Indexes**
  - Index is prefixed if the column used for partitioning themselves are also indexed
  - and are also at front of the list of index columns

- **Non-Prefixed Indizes**
  - Index is non-prefixed if the column is not right at front of the list of index columns
  - or not under the index columns

Index Partitioning

- **Local Prefixed Indexes**
  - Easy handling by Equi-partitioning
  - After SPLIT, MOVE, DROP not rebuild of the index necessary
  - Optimizer benefits of 1:1 mapping table and index partition
  - Can be defined as a unique index

- SQL>

```
CREATE [UNIQUE] INDEX
order_part_datum_status
ON order_part (orderdate, orderstatus)
LOCAL;
```

Index Partitioning

- **Local Non-Prefixed Indexes**
  - Can only be used in certain circumstances as a unique index
  - Constraint values may occur in multiple table partitions
  - Fewer opportunities for the Optimizer

- SQL>

```
CREATE [UNIQUE] INDEX
order_part_status_datum
ON order_part (orderstatus, orderdate)
LOCAL;

CREATE INDEX order_part_deliverydate
ON order_part (deliverydate)
LOCAL;
```

Index Partitioning

- **Global Prefixed Indexes**
  - Can be defined as a unique index
  - Often for archive data needed - indexing according to other criteria as date, which is usually partitioning column
  - Exclusively range or hash partitioning
  - Rebuild when table partition changes
    - ALTER TABLE … UPDATE GLOBAL INDEXES
  - Fewer opportunities for the Optimizer

- SQL>

```
CREATE [UNIQUE] INDEX pk_order_part
ON order_part (orderid)
GLOBAL;
```

Conclusion

- New features 12c
- Resume

New features 12c R1

- **Online-Move-Partition**
  - MOVE PARTITION … ONLINE – DML allowed on the table during the MOVE
- **Interval-Reference-Partitioning**
  - Combination finally available
- **Reference Partitioning**
  - Inheritance of TRUNCATE and EXCHANGE PARTITION - operations cascaded to child tables
- **SPLIT and MERGE**
  - Command Extension - process with a single command several (sub) partitions
- **Index creation**
  - More flexibility - Create local and global indexes on a subset of the table partitions
    - INDEXING ON / OFF per table and / or per partition

New features 12c R2

- Multi-column list partitioning
- Auto-list partitioning
- Interval Subpartitioning
- Read only partitions
- Alter table modify online

Resume

- **ILM and Advanced Compression**
  - Hot and Cold Data
- **Oracle Partitionierung can be used by everybody …**
  with Enterprise Edition and Partitioning Option
  - Directly usable because of transparency for each application
  - Handicap: Additional costs by Partitioning Option
- **Flashback Data Archive and Unified Auditing**
  - Use partitioning automatically also in SE (no license needed)

Thank you!

- **Time for your questions.**

- **Contact**

Sebastian Winkler

CarajanDB GmbH
Siemensstraße 25
50374 Erftstadt

Email: sebastian.winkler@carajandb.com
Fon: +49 2235 170 91 86
Mobil: +49 175 864 90 61
Twitter: sjw1011
Blog: www.carajandb.com/en/blogs/blog-swinkler