EDB Postgres™ Advanced Server

Release 13

Database Compatibility Table Partitioning Guide

Oct 20, 2020

Contents

CHAPTER 1

Introduction

In a partitioned table, one logically large table is broken into smaller physical pieces. Partitioning can provide several benefits:

- Query performance can be improved dramatically in certain situations, particularly when most of the heavily accessed rows of the table are in a single partition or a small number of partitions. Partitioning allows you to omit the partition column from the front of an index, reducing index size and making it more likely that the heavily used parts of the index fits in memory.
- When a query or update accesses a large percentage of a single partition, performance may improve because the server will perform a sequential scan of the partition instead of using an index and random access reads scattered across the whole table.
- A bulk load (or unload) can be implemented by adding or removing partitions, if you plan that requirement into the partitioning design. `ALTER TABLE` is far faster than a bulk operation. It also entirely avoids the `VACUUM` overhead caused by a bulk `DELETE`.
- Seldom-used data can be migrated to less-expensive (or slower) storage media.

Table partitioning is worthwhile only when a table would otherwise be very large. The exact point at which a table will benefit from partitioning depends on the application; a good rule of thumb is that the size of the table should exceed the physical memory of the database server.

This document discusses the aspects of table partitioning compatible with Oracle databases that are supported by Advanced Server.

Note: This document and particularly the partitioning presented in this chapter do not describe the *declarative partitioning* feature, which has been introduced with PostgreSQL version 10. Note that PostgreSQL declarative partitioning is supported in Advanced Server 10 in addition to the table partitioning compatible with Oracle databases as described in this chapter. For information about declarative partitioning, see the PostgreSQL core documentation available at: https://www.postgresql.org/docs/current/static/ddl-partitioning.html

The PostgreSQL 9.6 `INSERT... ON CONFLICT DO NOTHING/UPDATE` clause (commonly known as `UPSERT`) is not supported on Oracle-styled partitioned tables. If you include the `ON CONFLICT DO NOTHING/UPDATE` clause when invoking the `INSERT` command to add data to a partitioned table, the server will return an error.

CHAPTER 2

Selecting a Partition Type

When you create a partitioned table, you specify `LIST, RANGE`, or `HASH` partitioning rules. The partitioning rules provide a set of constraints that define the data that is stored in each partition. As new rows are added to the partitioned table, the server uses the partitioning rules to decide which partition should contain each row.

Advanced Server can also use the partitioning rules to enforce partition pruning, improving performance when responding to user queries. When selecting a partitioning type and partitioning keys for a table, you should take into consideration how the data that is stored within a table will be queried, and include often-queried columns in the partitioning rules.

List Partitioning

When you create a list-partitioned table, you specify a single partitioning key column. When adding a row to the table, the server compares the key values specified in the partitioning rule to the corresponding column within the row. If the column value matches a value in the partitioning rule, the row is stored in the partition named in the rule.

Note: The `List Partitioning` does not support multi-column list partitioning.

Range Partitioning

When you create a range-partitioned table, you specify one or more partitioning key columns. When you add a new row to the table, the server compares the value of the partitioning key (or keys) to the corresponding column (or columns) in a table entry. If the column values satisfy the conditions specified in the partitioning rule, the row is stored in the partition named in the rule.

Hash Partitioning

When you create a hash-partitioned table, you specify one or more partitioning key columns. Data is divided into (approx.) equal-sized partitions amongst the specified partitions. When you add a row to a hash-

partitioned table, the server computes a hash value for the data in the specified column (or columns), and stores the row in a partition according to the hash value.

Note: When upgrading Advanced Server, you must rebuild each hash-partitioned table on the upgraded version server.

Subpartitioning

Subpartitioning breaks a partitioned table into smaller subsets. All subsets must be stored in the same database server cluster. A table is typically subpartitioned by a different set of columns, and may be a different subpartitioning type than the parent partition. If one partition is subpartitioned, then each partition will have at least one subpartition.

If a table is subpartitioned, no data will be stored in any of the partition tables; the data will be stored instead in the corresponding subpartitions.

2.1 Interval Range Partitioning

Interval Range Partitioning is an extension to range partitioning that allows a database to automatically create a new partition when the newly inserted data exceeds the range of an existing partitioning. To implement interval range partitioning, include the `INTERVAL` clause and specify the range size for a new partition.

The high value of a range partition (also known as the transition point) is determined by the range partitioning key value. The database creates new partitions for inserted data with values that are beyond that high value.

If an interval is set to 1 month and if data is inserted for two months after the current transition point, only the partition for that month is created and not the intervening partition. For example, you can create an interval range partitioned table with a monthly interval and a current transition point of February 15, 2019. Now, if you try to insert data for May 10, 2019, then the required partition for April 15 to May 15, 2019 is created and data will be inserted into that partition. The partition for February 15, 2019 to March 15, 2019 and March 15, 2019 to April 15, 2019 is skipped.

For information about Interval Range Partitioning syntax, see *CREATE TABLE... PARTITION BY*.

Restrictions on Interval Range Partitioning

The following restrictions apply to the `INTERVAL` clause:

- Interval range partitioning is restricted to a single partition key; that key must be a numerical or date range.
- You must define at least one range partition.
- The `INTERVAL` clause is not supported for index-organized tables.
- A domain index cannot be created on an interval range partitioned table.
- In composite partitioning, the interval range partitioning can be useful as a primary partitioning mechanism but not supported at the subpartition level.
- `DEFAULT` and `MAXVALUE` cannot be defined for an interval range partitioned table.

- NULL, Not-a-Number, and Infinity values cannot be specified in the partitioning key column.
- Interval range partitioning expression must yield constant value and must not be a negative value.
- The partitions for an interval range partitioned table are created in increasing order only.

2.2 Automatic List Partitioning

Automatic list partitioning is an extension to LIST partitioning that allows a database to automatically create a partition for any new distinct value of the list partitioning key. A new partition is created when data is inserted into the LIST partitioned table and the inserted value does not match any of the existing table partition. Use the AUTOMATIC clause to implement automatic list partitioning.

For example, consider a table named sales with a sales_state column that contains the existing partition values CALIFORNIA and FLORIDA. Each of the sales_state values will increase with a rise in the state-wise sales. A sale in a new state (for example, INDIANA and OHIO) will require the creation of new partitions. If you have implemented automatic list partitioning, the new partitions INDIANA and OHIO are automatically created and data is entered into the sales table.

For information about automatic list partitioning syntax, see *CREATE TABLE... PARTITION BY*.

Restrictions on Automatic List Partitioning

The following restrictions apply to the AUTOMATIC clause:

- A table that enables automatic list partitioning cannot have a DEFAULT partition.
- Automatic list partitioning does not support multi-column list partitioning.
- In composite partitioning, the automatic list partitioning can be useful as a primary partitioning mechanism but is not supported at the subpartition level.

CHAPTER 3

Using Partition Pruning

Advanced Server's query planner uses *partition pruning* to compute an efficient plan to locate a row (or rows) that matches the conditions specified in the `WHERE` clause of a `SELECT` statement. To successfully prune partitions from an execution plan, the `WHERE` clause must constrain the information that is compared to the partitioning key column specified when creating the partitioned table. When querying a:

- list-partitioned table, partition pruning is effective when the `WHERE` clause compares a literal value to the partitioning key using operators like equal (=) or `AND`.
- range-partitioned table, partition pruning is effective when the `WHERE` clause compares a literal value to a partitioning key using operators such as equal (=), less than (<), or greater than (>).
- hash-partitioned table, partition pruning is effective when the `WHERE` clause compares a literal value to the partitioning key using an operator such as equal (=).

The partition pruning mechanism uses two optimization techniques:

- Fast Pruning
- Constraint exclusion

Partition pruning techniques limit the search for data to only those partitions in which the values for which you are searching might reside. Both pruning techniques remove partitions from a query's execution plan, increasing performance.

The difference between the fast pruning and constraint exclusion is that fast pruning understands the relationship between the partitions in an Oracle-partitioned table, while constraint exclusion does not. For example, when a query searches for a specific value within a list-partitioned table, fast pruning can reason that only a specific partition may hold that value, while constraint exclusion must examine the constraints defined for *each* partition. Fast pruning occurs early in the planning process to reduce the number of partitions that the planner must consider, while constraint exclusion occurs late in the planning process.

Using Constraint Exclusion

The constraint_exclusion parameter controls constraint exclusion. The constraint_exclusion parameter may have a value of on, off, or partition. To enable constraint exclusion, the parameter must be set to *either* partition or on. By default, the parameter is set to partition.

For more information about constraint exclusion, see:

https://www.postgresql.org/docs/current/static/ddl-partitioning.html

When constraint exclusion is enabled, the server examines the constraints defined for each partition to determine if that partition can satisfy a query.

When you execute a SELECT statement that *does not* contain a WHERE clause, the query planner must recommend an execution plan that searches the entire table. When you execute a SELECT statement that *does* contain a WHERE clause, the query planner determines in which partition that row would be stored, and sends query fragments to that partition, pruning the partitions that could not contain that row from the execution plan. If you are not using partitioned tables, disabling constraint exclusion may improve performance.

Fast Pruning

Like constraint exclusion, fast pruning can only optimize queries that include a WHERE (or join) clause, and only when the qualifiers in the WHERE clause match a certain form. In both cases, the query planner will avoid searching for data within partitions that cannot possibly hold the data required by the query.

Fast pruning is controlled by a boolean configuration parameter named edb_enable_pruning. If edb_enable_pruning is ON, Advanced Server will fast prune certain queries. If edb_enable_pruning is OFF, the server will disable fast pruning.

Note: Fast pruning cannot optimize queries against subpartitioned tables or optimize queries against range-partitioned tables that are partitioned on more than one column.

For LIST-partitioned tables, Advanced Server can fast prune queries that contain a WHERE clause that constrains a partitioning column to a literal value. For example, given a LIST-partitioned table such as:

```
CREATE TABLE sales_hist(..., country text, ...)
PARTITION BY LIST(country)
(
    PARTITION americas VALUES('US', 'CA', 'MX'),
    PARTITION europe VALUES('BE', 'NL', 'FR'),
    PARTITION asia VALUES('JP', 'PK', 'CN'),
    PARTITION others VALUES(DEFAULT)
)
```

Fast pruning can reason about WHERE clauses such as:

```
WHERE country = 'US'
```

```
WHERE country IS NULL;
```

Given the first WHERE clause, fast pruning would eliminate partitions europe, asia, and others because those partitions cannot hold rows that satisfy the qualifier: WHERE country = 'US'.

Given the second WHERE clause, fast pruning would eliminate partitions americas, europe, and asia because those partitions cannot hold rows where country IS NULL.

The operator specified in the WHERE clause must be an equal sign (=) or the equality operator appropriate for the data type of the partitioning column.

For range-partitioned tables, Advanced Server can fast prune queries that contain a WHERE clause that constrains a partitioning column to a literal value, but the operator may be any of the following:

>

>=

=

<=

<

Fast pruning will also reason about more complex expressions involving AND and BETWEEN operators, such as:

```
WHERE size > 100 AND size <= 200
WHERE size BETWEEN 100 AND 200
```

Fast pruning cannot prune based on expressions involving OR or IN. For example, when querying a RANGE-partitioned table, such as:

```
CREATE TABLE boxes(id int, size int, color text)
  PARTITION BY RANGE(size)
(
    PARTITION small VALUES LESS THAN(100),
    PARTITION medium VALUES LESS THAN(200),
    PARTITION large VALUES LESS THAN(300)
)
```

Fast pruning can reason about WHERE clauses such as:

```
WHERE size > 100 -- scan partitions 'medium' and 'large'

WHERE size >= 100 -- scan partitions 'medium' and 'large'

WHERE size = 100 -- scan partition 'medium'

WHERE size <= 100 -- scan partitions 'small' and 'medium'

WHERE size < 100 -- scan partition 'small'

WHERE size > 100 AND size < 199 -- scan partition 'medium'

WHERE size BETWEEN 100 AND 199 -- scan partition 'medium'

WHERE color = 'red' AND size = 100 -- scan 'medium'

WHERE color = 'red' AND (size > 100 AND size < 199) -- scan
'medium'
```

In each case, fast pruning requires that the qualifier must refer to a partitioning column and literal value (or `IS NULL/IS NOT NULL`).

Note: Fast pruning can also optimize `DELETE` and `UPDATE` statements containing `WHERE` clauses of the forms described above.

3.1 Example - Partition Pruning

The `EXPLAIN` statement displays the execution plan of a statement. You can use the `EXPLAIN` statement to confirm that Advanced Server is pruning partitions from the execution plan of a query.

To demonstrate the efficiency of partition pruning, first create a simple table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

Then, perform a constrained query that includes the `EXPLAIN` statement:

```
EXPLAIN (COSTS OFF) SELECT * FROM sales WHERE country = 'INDIA';
```

The resulting query plan shows that the server will scan only the `sales_asia` table - the table in which a row with a `country` value of `INDIA` would be stored:

```
edb=# EXPLAIN (COSTS OFF) SELECT * FROM sales WHERE country = 'INDIA';
                     QUERY PLAN
-----------------------------------------------------
 Append
   ->  Seq Scan on sales_asia
         Filter: ((country)::text = 'INDIA'::text)
(3 rows)
```

If you perform a query that searches for a row that matches a value not included in the partitioning key:

```
EXPLAIN (COSTS OFF) SELECT * FROM sales WHERE dept_no = '30';
```

The resulting query plan shows that the server must look in all of the partitions to locate the rows that satisfy the query:

```
edb=# EXPLAIN (COSTS OFF) SELECT * FROM sales WHERE dept_no = '30';
                QUERY PLAN
------------------------------------------
 Append
   ->  Seq Scan on sales_americas
         Filter: (dept_no = '30'::numeric)
   ->  Seq Scan on sales_europe
         Filter: (dept_no = '30'::numeric)
   ->  Seq Scan on sales_asia
         Filter: (dept_no = '30'::numeric)
(7 rows)
```

Constraint exclusion also applies when querying subpartitioned tables:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date) SUBPARTITION BY LIST (country)
(
  PARTITION "2011" VALUES LESS THAN('01-JAN-2012')
  (
    SUBPARTITION europe_2011 VALUES ('ITALY', 'FRANCE'),
    SUBPARTITION asia_2011 VALUES ('PAKISTAN', 'INDIA'),
    SUBPARTITION americas_2011 VALUES ('US', 'CANADA')
  ),
  PARTITION "2012" VALUES LESS THAN('01-JAN-2013')
  (
    SUBPARTITION europe_2012 VALUES ('ITALY', 'FRANCE'),
    SUBPARTITION asia_2012 VALUES ('PAKISTAN', 'INDIA'),
    SUBPARTITION americas_2012 VALUES ('US', 'CANADA')
  ),
  PARTITION "2013" VALUES LESS THAN('01-JAN-2015')
  (
    SUBPARTITION europe_2013 VALUES ('ITALY', 'FRANCE'),
    SUBPARTITION asia_2013 VALUES ('PAKISTAN', 'INDIA'),
    SUBPARTITION americas_2013 VALUES ('US', 'CANADA')
  )
);
```

When you query the table, the query planner prunes any partitions or subpartitions from the search path that cannot possibly contain the desired result set:

```
edb=# EXPLAIN (COSTS OFF) SELECT * FROM sales WHERE country = 'US' AND date
= 'Dec 12, 2012';
                                                    QUERY PLAN
------------------------------------------------------------------------------
-----------------------------------
```

(continues on next page)

(continued from previous page)

```
 Append
   ->  Seq Scan on sales_americas_2012
         Filter: (((country)::text = 'US'::text) AND (date = '12-DEC-12
         00:00:00'::timestamp without time zone))
(3 rows)
```

CHAPTER 4

Partitioning Commands Compatible with Oracle Databases

The following sections provide information about using the table partitioning syntax compatible with Oracle databases supported by Advanced Server.

4.1 CREATE TABLE. . . PARTITION BY

Use the `PARTITION BY` clause of the `CREATE TABLE` command to create a partitioned table with data distributed amongst one or more partitions (and subpartitions). The command syntax comes in the following forms:

List Partitioning Syntax

Use the first form to create a list-partitioned table:

```
CREATE TABLE [ schema. ]<table_name>
   <table_definition>
   PARTITION BY LIST(<column>) [ AUTOMATIC ]
   [SUBPARTITION BY {RANGE|LIST|HASH} (<column>[, <column> ]...)]
   (<list_partition_definition>[, <list_partition_definition>]...);

Where list_partition_definition is:

      PARTITION [<partition_name>]
        VALUES (<value>[, <value>]...)
        [TABLESPACE <tablespace_name>]
        [(<subpartition>, ...)]
```

Range Partitioning Syntax

Use the second form to create a range-partitioned table:

```
CREATE TABLE [ schema. ]<table_name>
   <table_definition>
   PARTITION BY RANGE(<column>[, <column> ]...)
   [INTERVAL (<constant> | <expression>)]
   [SUBPARTITION BY {RANGE|LIST|HASH} (<column>[, <column> ]...)]
   (<range_partition_definition>[, <range_partition_definition>]...);

Where range_partition_definition is:

      PARTITION [<partition_name>]
        VALUES LESS THAN (<value>[, <value>]...)
        [TABLESPACE <tablespace_name>]
        [(<subpartition>, ...)]
```

Hash Partitioning Syntax

Use the third form to create a hash-partitioned table:

```
CREATE TABLE [ schema. ]<table_name>
   <table_definition>
   PARTITION BY HASH(<column>[, <column> ]...)
   [SUBPARTITION BY {RANGE|LIST|HASH} (<column>[,
   <column> ]...)]
   [SUBPARTITIONS <num>] [STORE IN ( <tablespace_name> [,
   <tablespace_name>]... ) ]]
   [PARTITIONS <num>] [STORE IN ( <tablespace_name> [,
   <tablespace_name>]... ) ] |
   (<hash_partition_definition>[, <hash_partition_definition>]...);

Where hash_partition_definition is:

      [PARTITION <partition_name>]
        [TABLESPACE <tablespace_name>]
        [(<subpartition>, ...)]
```

Subpartitioning Syntax

`subpartition` may be one of the following:

```
{<list_subpartition> | <range_subpartition> | <hash_subpartition>}

where list_subpartition is:

      SUBPARTITION [<subpartition_name>]
        VALUES (<value>[, <value>]...)
        [TABLESPACE <tablespace_name>]

where range_subpartition is:

      SUBPARTITION [<subpartition_name>]
```

(continues on next page)

(continued from previous page)

```
        VALUES LESS THAN (<value>[, <value>]...)
        [TABLESPACE <tablespace_name>]

where hash_subpartition is:

      SUBPARTITION <subpartition_name>
        [TABLESPACE <tablespace_name>] |
      SUBPARTITIONS <num> [STORE IN ( <tablespace_name> [,
        <tablespace_name>]... ) ]
```

Description

The `CREATE TABLE... PARTITION BY` command creates a table with one or more partitions; each partition may have one or more subpartitions. There is no upper limit to the number of defined partitions, but if you include the `PARTITION BY` clause, you must specify at least one partitioning rule. The resulting table will be owned by the user that creates it.

Use the `PARTITION BY LIST` clause to divide a table into partitions based on the values entered in a specified column. Each partitioning rule must specify at least one literal value, but there is no upper limit placed on the number of values you may specify. Include a rule that specifies a matching value of `DEFAULT` to direct any un-qualified rows to the given partition; for more information about using the `DEFAULT` keyword, see *Handling Stray Values in a LIST or RANGE Partitioned Table*.

Use the `AUTOMATIC` clause to specify the table should use automatic list partitioning. By specifying the `AUTOMATIC` clause, the database automatically creates partitions when new data is inserted into a table that does not correspond to any value declared for an existing partition.

For more information about `AUTOMATIC LIST PARTITIONING`, see *Automatic List Partitioning*.

Use the `PARTITION BY RANGE` clause to specify boundary rules by which to create partitions. Each partitioning rule must contain at least one column of a data type that has two operators (i.e., a greater-than or equal to operator, and a less-than operator). Range boundaries are evaluated against a `LESS THAN` clause and are non-inclusive; a date boundary of January 1, 2013 will include only those date values that fall on or before December 31, 2012.

Range partition rules must be specified in ascending order. `INSERT` commands that store rows with values that exceed the top boundary of a range-partitioned table will fail unless the partitioning rules include a boundary rule that specifies a value of `MAXVALUE`. If you do not include a `MAXVALUE` partitioning rule, any row that exceeds the maximum limit specified by the boundary rules will result in an error.

For more information about using the `MAXVALUE` keyword, see *Handling Stray Values in a LIST or RANGE Partitioned Table*.

Use the `INTERVAL` clause to specify an interval range partitioned table. By specifying an `INTERVAL` clause, the range partitioning is extended by the database automatically to create partitions of a specified interval when new data is inserted into a table that exceeds an existing range partition.

For more information about `INTERVAL RANGE PARTITION`, see *Interval Range Partitioning*.

Use the `PARTITION BY HASH` clause to create a hash-partitioned table. In a `HASH` partitioned table, data is divided amongst equal-sized partitions based on the hash value of the column specified in the partitioning syntax. When specifying a `HASH` partition, choose a column (or combination of columns) that is as close

to unique as possible to help ensure that data is evenly distributed amongst the partitions. When selecting a partitioning column (or combination of columns), select a column (or columns) that you frequently search for exact matches for best performance.

Use the STORE IN clause to specify the tablespace list across which the autogenerated partitions or subpartitions will be stored.

Note: If you are upgrading Advanced Server, you must rebuild the hash-partitioned table on the upgraded version server.

Use the TABLESPACE keyword to specify the name of a tablespace on which a partition or subpartition will reside; if you do not specify a tablespace, the partition or subpartition will reside in the default tablespace.

If a table definition includes the SUBPARTITION BY clause, each partition within that table will have at least one subpartition. Each subpartition may be explicitly defined or system-defined.

If the subpartition is system-defined, the server-generated subpartition will reside in the default tablespace, and the name of the subpartition will be assigned by the server. The server will create:

- A DEFAULT subpartition if the SUBPARTITION BY clause specifies LIST.
- A MAXVALUE subpartition if the SUBPARTITION BY clause specifies RANGE.

The server will generate a subpartition name that is a combination of the partition table name and a unique identifier. You can query the ALL_TAB_SUBPARTITIONS table to review a complete list of subpartition names.

Parameters

table_name

The name (optionally schema-qualified) of the table to be created.

table_definition

The column names, data types, and constraint information as described in the PostgreSQL core documentation for the CREATE TABLE statement, available at:

https://www.postgresql.org/docs/current/static/sql-createtable.html

partition_name

The name of the partition to be created. Partition names must be unique amongst all partitions and subpartitions, and must follow the naming conventions for object identifiers.

subpartition_name

The name of the subpartition to be created. Subpartition names must be unique amongst all partitions and subpartitions, and must follow the naming conventions for object identifiers.

num

The PARTITIONS num clause can be used to specify the number of HASH partitions to be created. Alternatively, you can use the partition definition to specify individual partitions and their tablespaces.

Note: You can either use PARTITIONS or PARTITION DEFINITION when creating a table.

The SUBPARTITIONS num clause is only supported for HASH partitions, and can be used to specify the number of hash subpartitions to be created. Alternatively, you can use the subpartition definition to specify individual subpartitions and their tablespaces. If no SUBPARTITIONS or SUBPARTITION DEFINITION is specified, then the partition creates a subpartition based on the SUBPARTITION TEMPLATE.

column

The name of a column on which the partitioning rules are based. Each row will be stored in a partition that corresponds to the value of the specified column(s).

constant | expression

The constant and expression specifies a NUMERIC, DATE or TIME value.

(value[, value]...)

Use value to specify a quoted literal value (or comma-delimited list of literal values) by which table entries will be grouped into partitions. Each partitioning rule must specify at least one value, but there is no limit placed on the number of values specified within a rule. value may be NULL, DEFAULT (if specifying a LIST partition), or MAXVALUE (if specifying a RANGE partition).

When specifying rules for a list-partitioned table, include the DEFAULT keyword in the last partition rule to direct any un-matched rows to the given partition. If you do not include a rule that includes a value of DEFAULT, any INSERT statement that attempts to add a row that does not match the specified rules of at least one partition will fail, and return an error.

When specifying rules for a list-partitioned table, include the MAXVALUE keyword in the last partition rule to direct any un-categorized rows to the given partition. If you do not include a MAXVALUE partition, any INSERT statement that attempts to add a row where the partitioning key is greater than the highest value specified will fail, and return an error.

tablespace_name

The name of the tablespace in which the partition or subpartition resides.

4.1.1 Example - PARTITION BY LIST

The following example creates a partitioned table (sales) using the PARTITION BY LIST clause. The sales table stores information in three partitions (europe, asia, and americas).

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

The resulting table is partitioned by the value specified in the country column.

```
edb=# SELECT partition_name, high_value from ALL_TAB_PARTITIONS;
 partition_name |     high_value
----------------+---------------------
 EUROPE         | 'FRANCE', 'ITALY'
 ASIA           | 'INDIA', 'PAKISTAN'
 AMERICAS       | 'US', 'CANADA'
(3 rows)
```

- Rows with a value of FRANCE or ITALY in the country column are stored in the europe partition.
- Rows with a value of INDIA or PAKISTAN in the country column are stored in the asia partition.
- Rows with a value of US or CANADA in the country column are stored in the americas partition.

The server would evaluate the following statement against the partitioning rules, and store the row in the europe partition.

```
INSERT INTO sales VALUES (10, '9519a', 'FRANCE', '18-Aug-2012',
'650000');
```

4.1.2 Example - AUTOMATIC LIST PARTITION

The following example shows a (sales) table that uses an AUTOMATIC clause to create an automatic list partitioned table on the sales_state column. The database creates a new partition and adds data to a table.

```
CREATE TABLE sales
(
  dept_no      number,
  part_no      varchar2,
```

(continues on next page)

(continued from previous page)

```
  sales_state  varchar2(20),
  date         date,
  amount       number
)
PARTITION BY LIST(sales_state) AUTOMATIC
(
  PARTITION P_CAL VALUES('CALIFORNIA'),
  PARTITION P_FLO VALUES('FLORIDA')
);
```

Query the ALL_TAB_PARTITIONS view to see an existing partition that is successfully created.

```
edb=# SELECT table_name, partition_name, high_value from ALL_TAB_PARTITIONS;
 table_name | partition_name |  high_value
------------+----------------+--------------
 SALES      | P_CAL          | 'CALIFORNIA'
 SALES      | P_FLO          | 'FLORIDA'
(2 rows)
```

Now, insert data into a sales table that cannot fit into an existing partition. For the regular list partitioned table, you will encounter an error but automatic list partitioning creates and inserts the data into a new partition.

```
edb=# INSERT INTO sales VALUES (1, 'IND', 'INDIANA');
INSERT 0 1
edb=# INSERT INTO sales VALUES (2, 'OHI', 'OHIO');
INSERT 0 1
```

Then, query the ALL_TAB_PARTITIONS view again after the insert. The partition is automatically created and data is inserted to hold a new value. A system-generated name of the partition is created that varies for each session.

```
edb=# SELECT table_name, partition_name, high_value from ALL_TAB_PARTITIONS;
 table_name | partition_name |  high_value
------------+----------------+--------------
 SALES      | P_CAL          | 'CALIFORNIA'
 SALES      | P_FLO          | 'FLORIDA'
 SALES      | SYS106900103   | 'INDIANA'
 SALES      | SYS106900104   | 'OHIO'
(4 rows)
```

4.1.3 Example - PARTITION BY RANGE

The following example creates a partitioned table (sales) using the PARTITION BY RANGE clause. The sales table stores information in four partitions (q1_2012, q2_2012, q3_2012 and q4_2012).

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date)
(
  PARTITION q1_2012
    VALUES LESS THAN('2012-Apr-01'),
  PARTITION q2_2012
    VALUES LESS THAN('2012-Jul-01'),
  PARTITION q3_2012
    VALUES LESS THAN('2012-Oct-01'),
  PARTITION q4_2012
    VALUES LESS THAN('2013-Jan-01')
);
```

The resulting table is partitioned by the value specified in the date column.

```
edb=# SELECT partition_name, high_value from ALL_TAB_PARTITIONS;
 partition_name |      high_value
----------------+---------------------
 Q1_2012        | '01-APR-12 00:00:00'
 Q2_2012        | '01-JUL-12 00:00:00'
 Q3_2012        | '01-OCT-12 00:00:00'
 Q4_2012        | '01-JAN-13 00:00:00'
(4 rows)
```

- Any row with a value in the date column before April 1, 2012 is stored in a partition named q1_2012.
- Any row with a value in the date column before July 1, 2012 is stored in a partition named q2_2012.
- Any row with a value in the date column before October 1, 2012 is stored in a partition named q3_2012.
- Any row with a value in the date column before January 1, 2013 is stored in a partition named q4_2012.

The server would evaluate the following statement against the partitioning rules and store the row in the q3_2012 partition.

```
INSERT INTO sales VALUES (10, '9519a', 'FRANCE', '18-Aug-2012',
'650000');
```

4.1.4 Example - INTERVAL RANGE PARTITION

The following example shows a (sales) table that is partitioned by interval on the sold_month column. The range partition is created to establish a transition point and new partitions are created beyond that transition point. The database creates a new interval range partition and adds data into a table.

```
CREATE TABLE sales
(
  prod_id           int,
  prod_quantity     int,
  sold_month        date
)
PARTITION BY RANGE(sold_month)
INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
(
  PARTITION p1
    VALUES LESS THAN('15-JAN-2019'),
  PARTITION p2
    VALUES LESS THAN('15-FEB-2019')
);
```

First, query the ALL_TAB_PARTITIONS view before an interval range partition is created by the database.

```
edb=# SELECT partition_name, high_value from ALL_TAB_PARTITIONS;
 partition_name |      high_value
----------------+----------------------
 P1             | '15-JAN-19 00:00:00'
 P2             | '15-FEB-19 00:00:00'
(2 rows)
```

Now, insert data into a sales table that exceeds the high value of a range partition.

```
edb=# INSERT INTO sales VALUES (1,200,'10-MAY-2019');
INSERT 0 1
```

Then, query the ALL_TAB_PARTITIONS view again after the insert. The data is successfully inserted and a system generated name of the interval range partition is created that varies for each session.

```
edb=# SELECT partition_name, high_value from ALL_TAB_PARTITIONS;
 partition_name |      high_value
----------------+----------------------
 P1             | '15-JAN-19 00:00:00'
 P2             | '15-FEB-19 00:00:00'
 SYS916340103   | '15-MAY-19 00:00:00'
(3 rows)
```

4.1.5 Example - PARTITION BY HASH

The following example creates a partitioned table (sales) using the PARTITION BY HASH clause. The sales table stores information in three partitions (p1, p2, and p3).

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY HASH (part_no)
(
  PARTITION p1,
  PARTITION p2,
  PARTITION p3
);
```

The table will return an empty string for the hash partition value specified in the part_no column.

```
edb=# SELECT partition_name, high_value from ALL_TAB_PARTITIONS;
 partition_name | high_value
----------------+------------
 P1             |
 P2             |
 P3             |
(3 rows)
```

Use the following command to view the hash value of the part_no column.

```
edb=# \d+ sales
                                        Partitioned table "public.sales"
 Column  |            Type             |Collation|Nullable|Default|Storage  |
---------+-----------------------------+---------+--------+-------+---------+-
 dept_no | numeric                     |         |        |       | main    |
 part_no | character varying           |         |        |       | extended|
 country | character varying(20)       |         |        |       | extended|
 date    | timestamp without time zone|         |        |       | plain   |
 amount  | numeric                     |         |        |       | main    |
Partition key: HASH (part_no)
Partitions: sales_p1 FOR VALUES WITH (modulus 3, remainder 0),
            sales_p2 FOR VALUES WITH (modulus 3, remainder 1),
            sales_p3 FOR VALUES WITH (modulus 3, remainder 2)
```

The table is partitioned by the hash value of the values specified in the part_no column.

```
edb=# SELECT partition_name, partition_position from ALL_TAB_PARTITIONS;
 partition_name | partition_position
----------------+--------------------
 P1             |                  1
```

(continues on next page)

(continued from previous page)

```
 P2             |                  2
 P3             |                  3
(3 rows)
```

The server will evaluate the hash value of the `part_no` column and distribute the rows into approximately equal partitions.

4.1.6 Example - PARTITION BY HASH. . . PARTITIONS num. . .

The following example creates a hash-partitioned table `(sales)` using the `PARTITION BY HASH` clause. The partitioning column is `part_no`; the number of partitions to be created is specified.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY HASH (part_no) PARTITIONS 8;
```

The eight partitions are created and assigned a system-generated name. The partitions are stored in the default tablespace of the table.

```
edb=# SELECT table_name, partition_name FROM ALL_TAB_PARTITIONS WHERE
table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name
------------+----------------
 SALES      | SYS0101
 SALES      | SYS0102
 SALES      | SYS0103
 SALES      | SYS0104
 SALES      | SYS0105
 SALES      | SYS0106
 SALES      | SYS0107
 SALES      | SYS0108
(8 rows)
```

4.1.7 Example - PARTITION BY HASH. . . PARTITIONS num. . . STORE IN

The following example creates a hash-partitioned table named `(sales)`. The number of partitions to be created and the tablespaces in which the partition will reside are specified.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
```

(continues on next page)

(continued from previous page)

```
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY HASH (part_no) PARTITIONS 5 STORE IN (ts1, ts2, ts3);
```

The `STORE IN` clause evenly distributes the partitions across specified tablespaces `(ts1, ts2, ts3)`.

```
edb=# SELECT table_name, partition_name, tablespace_name FROM
ALL_TAB_PARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | tablespace_name
------------+----------------+-----------------
 SALES      | SYS0101        | TS1
 SALES      | SYS0102        | TS2
 SALES      | SYS0103        | TS3
 SALES      | SYS0104        | TS1
 SALES      | SYS0105        | TS2
(5 rows)
```

4.1.8 Example - HASH/RANGE PARTITIONS num...

The `HASH` partition clause allows you to define a partitioning strategy. You can extend the `PARTITION BY HASH` clause to include `SUBPARTITION BY` either `[ RANGE | LIST | HASH ]` to create subpartitions in an `HASH` partitioned table.

The following example creates a table `(sales)` that is hash-partitioned by `part_no` and subpartitioned using range by `dept_no`. The number of partitions is specified when creating a table `(sales)`.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY HASH (part_no) SUBPARTITION BY RANGE (dept_no) PARTITIONS 5;
```

The five partitions are created with default subpartitions and assigned system-generated names.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | SYS0101        | SYS0102
 SALES      | SYS0103        | SYS0104
 SALES      | SYS0105        | SYS0106
 SALES      | SYS0107        | SYS0108
 SALES      | SYS0109        | SYS0110
(5 rows)
```

4.1.9 Example - LIST/HASH SUBPARTITIONS num...

The following example shows a table (sales) that is list-partitioned by country and subpartitioned using hash partitioning by the dept_no column. The number of subpartitions is specified when creating the table.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST (country) SUBPARTITION BY HASH (dept_no) SUBPARTITIONS 3
(
  PARTITION p1 VALUES('FRANCE', 'ITALY'),
  PARTITION p2 VALUES('INDIA', 'PAKISTAN'),
  PARTITION p3 VALUES('US', 'CANADA')
);
```

The three partitions p1, p2 and p3 each contain three subpartitions with system-generated names.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | P1             | SYS0101
 SALES      | P1             | SYS0102
 SALES      | P1             | SYS0103
 SALES      | P2             | SYS0104
 SALES      | P2             | SYS0105
 SALES      | P2             | SYS0106
 SALES      | P3             | SYS0107
 SALES      | P3             | SYS0108
 SALES      | P3             | SYS0109
(9 rows)
```

4.1.10 Example - HASH/HASH PARTITIONS num... SUBPARTITIONS num...

The following example creates the (sales) table, hash-partitioned by part_no and hash-subpartitioned by dept_no.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
```

(continues on next page)

(continued from previous page)

```
PARTITION BY HASH (part_no) SUBPARTITION BY HASH (dept_no) SUBPARTITIONS 3
PARTITIONS 2;
```

The two partitions are created and each partition includes three subpartitions with the system-generated name assigned to them.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | SYS0101        | SYS0102
 SALES      | SYS0101        | SYS0103
 SALES      | SYS0101        | SYS0104
 SALES      | SYS0105        | SYS0106
 SALES      | SYS0105        | SYS0107
 SALES      | SYS0105        | SYS0108
(6 rows)
```

4.1.11 Example - HASH/HASH SUBPARTITIONS num... STORE IN

The following example creates a hash-partitioned table `(sales)`. The number of partitions and subpartitions to be created are specified along with the tablespaces in which the subpartitions will reside when creating a hash partitioned table.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY HASH (part_no) SUBPARTITION BY HASH (dept_no) SUBPARTITIONS 3
PARTITIONS 2 STORE IN (ts1, ts2);
```

The two partitions are created and assigned a system-generated name. The partitions are stored in the default tablespace and subpartitions are stored in tablespaces `(ts1)` and `(ts2)`.

```
edb=# SELECT table_name, partition_name, tablespace_name FROM
ALL_TAB_PARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | tablespace_name
------------+----------------+-----------------
 SALES      | SYS0101        |
 SALES      | SYS0105        |
(2 rows)
```

The `STORE IN` clause assigns the hash subpartitions to the tablespaces and stores them in two named tablespaces `(ts1, ts2)`.

```
edb=# SELECT table_name, partition_name, subpartition_name, tablespace_name
FROM ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name | tablespace_name
------------+----------------+-------------------+-----------------
 SALES      | SYS0101        | SYS0102           | TS1
 SALES      | SYS0101        | SYS0103           | TS2
 SALES      | SYS0101        | SYS0104           | TS1
 SALES      | SYS0105        | SYS0106           | TS1
 SALES      | SYS0105        | SYS0107           | TS2
 SALES      | SYS0105        | SYS0108           | TS1
(6 rows)
```

4.1.12 Example - HASH/HASH PARTITIONS num ...STORE IN SUBPARTITIONS num... STORE IN

The following example creates a hash-partitioned table `(sales)`. The number of partitions and subpartitions to be created are specified, along with the tablespaces in which the partitions and subpartitions will reside.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY HASH (part_no) SUBPARTITION BY HASH (dept_no) SUBPARTITIONS 3
STORE IN (ts3) PARTITIONS 2 STORE IN (ts1, ts2);
```

The two partitions are created with a system-generated name and stored in the default tablespace.

```
edb=# SELECT table_name, partition_name, tablespace_name FROM
ALL_TAB_PARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | tablespace_name
------------+----------------+-----------------
 SALES      | SYS0101        |
 SALES      | SYS0105        |
(2 rows)
```

Each partition includes three subpartitions; the `STORE IN` clause stores the subpartitions in tablespaces `(ts1)` and `(ts2)`.

```
edb=# SELECT table_name, partition_name, subpartition_name, tablespace_name
FROM ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name | tablespace_name
------------+----------------+-------------------+-----------------
 SALES      | SYS0101        | SYS0102           | TS1
 SALES      | SYS0101        | SYS0103           | TS2
 SALES      | SYS0101        | SYS0104           | TS1
```

(continues on next page)

(continued from previous page)

```
 SALES      | SYS0105        | SYS0106           | TS1
 SALES      | SYS0105        | SYS0107           | TS2
 SALES      | SYS0105        | SYS0108           | TS1
(6 rows)
```

Note: If the `STORE IN` clause is specified for partitions and subpartitions, then the subpartitions are stored in the tablespaces defined in the `PARTITIONS...STORE IN` clause and the `SUBPARTITIONS...STORE IN` clause is ignored.

4.1.13 Example - RANGE/HASH SUBPARTITIONS num...

The following example creates a range-partitioned table `(sales)` that is first partitioned by the transaction date; two range partitions are created and then hash-subpartitioned using the value of the `country` column.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE (date) SUBPARTITION BY HASH (country) SUBPARTITIONS 2
(
  PARTITION p1 VALUES LESS THAN ('2012-Apr-01') (SUBPARTITION q1_europe),
  PARTITION p2 VALUES LESS THAN ('2012-Jul-01')
);
```

This statement creates a table with two partitions; the subpartition explicitly named `q1_europe` is created for partition `p1`. Because subpartitions are not named for partition `p2`, the subpartitions are created based on the subpartition number, and are assigned a system-generated name.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | P1             | Q1_EUROPE
 SALES      | P2             | SYS0101
 SALES      | P2             | SYS0102
(3 rows)
```

4.1.14 Example - RANGE/HASH SUBPARTITIONS num... IN PARTITION DESCRIPTION

The following example creates a range-partitioned table (sales) that is first partitioned by the transaction date; two range partitions are created and then hash-subpartitioned using the value of the country column.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE (date) SUBPARTITION BY HASH (country) SUBPARTITIONS 2
(
  PARTITION p1 VALUES LESS THAN ('2012-Apr-01') SUBPARTITIONS 3,
  PARTITION p2 VALUES LESS THAN ('2012-Jul-01')
);
```

The partition p1 explicitly defines the subpartition count in the partition description. By default, two subpartitions will be created for partition p2; since subpartitions are not named, system-generated names are assigned.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | P1             | SYS0101
 SALES      | P1             | SYS0102
 SALES      | P1             | SYS0103
 SALES      | P2             | SYS0104
 SALES      | P2             | SYS0105
(5 rows)
```

4.1.15 Example - LIST/HASH SUBPARTITIONS num STORE IN... IN PARTITION DESCRIPTION

The following example creates a list-partitioned table (sales) with two list partitions. Partition p1 consists of three subpartitions and partition p2 consists of two subpartitions. Since the subpartitions are not named, system-generated names are assigned.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
```

(continues on next page)

(continued from previous page)

```
PARTITION BY LIST (country) SUBPARTITION BY HASH (part_no) SUBPARTITIONS 3
STORE IN (ts1)
(
  PARTITION p1 VALUES ('FRANCE', 'ITALY'),
  PARTITION p2 VALUES ('INDIA', 'PAKISTAN') SUBPARTITIONS 2 STORE IN
  (ts2)
);
```

The partition `p2` explicitly defines the subpartition count in the partition description. Based on the definition, two subpartitions are created and stored in the tablespace named `(ts2)`. The subpartitions for partition `p1` will be stored in the tablespace named `(ts1)`.

```
edb=# SELECT table_name, partition_name, subpartition_name, tablespace_name
FROM ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name | tablespace_name
------------+----------------+-------------------+-----------------
 SALES      | P1             | SYS0101           | TS1
 SALES      | P1             | SYS0102           | TS1
 SALES      | P1             | SYS0103           | TS1
 SALES      | P2             | SYS0104           | TS2
 SALES      | P2             | SYS0105           | TS2
(5 rows)
```

4.1.16 Example - LIST/HASH STORE IN... TABLESPACES

The following example creates a list-partitioned table `(sales)` with partition `p1` consisting of three subpartitions stored explicitly in the tablespace `(ts2)`.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST (country) SUBPARTITION BY HASH (part_no) SUBPARTITIONS 3
STORE IN (ts1)
(
  PARTITION p1 VALUES ('FRANCE', 'ITALY') TABLESPACE ts2
);
```

The `SELECT` statement shows partition `p1`, consisting of three subpartitions stored in the tablespace `(ts2)`.

```
edb=# SELECT table_name, partition_name, subpartition_name, tablespace_name
FROM ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name | tablespace_name
------------+----------------+-------------------+-----------------
 SALES      | P1             | SYS0101           | TS2
```

(continues on next page)

(continued from previous page)

```
 SALES      | P1             | SYS0102           | TS2
 SALES      | P1             | SYS0103           | TS2
(3 rows)
```

The following command adds a new partition `p2` to the `sales` table, five subpartitions will be created and distributed across the tablespace listed by the `STORE IN` clause.

```
ALTER TABLE sales ADD PARTITION p2 VALUES ('US', 'CANADA') SUBPARTITIONS 5
STORE IN (ts1);
```

A query of the `ALL_TAB_PARTITIONS` view shows the `sales` table with a partition named `p2`, with five subpartitions. The `STORE IN` clause distributes the subpartitions across a tablespace named `(ts1)`.

```
edb=# SELECT table_name, partition_name, subpartition_name, tablespace_name
FROM ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' and partition_name =
'P2' ORDER BY 1,2;
 table_name | partition_name | subpartition_name | tablespace_name
------------+----------------+-------------------+-----------------
 SALES      | P2             | SYS0105           | TS1
 SALES      | P2             | SYS0106           | TS1
 SALES      | P2             | SYS0107           | TS1
 SALES      | P2             | SYS0108           | TS1
 SALES      | P2             | SYS0109           | TS1
(5 rows)
```

4.1.17 Example - PARTITION BY RANGE, SUBPARTITION BY LIST

The following example creates a partitioned table `(sales)` that is first partitioned by the transaction date; the range partitions (`q1_2012, q2_2012, q3_2012` and `q4_2012`) are then list-subpartitioned using the value of the `country` column.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date)
  SUBPARTITION BY LIST(country)
  (
    PARTITION q1_2012
      VALUES LESS THAN('2012-Apr-01')
      (
        SUBPARTITION q1_europe VALUES ('FRANCE', 'ITALY'),
        SUBPARTITION q1_asia VALUES ('INDIA', 'PAKISTAN'),
        SUBPARTITION q1_americas VALUES ('US', 'CANADA')
      ),
```

(continues on next page)

(continued from previous page)

```
  PARTITION q2_2012
    VALUES LESS THAN('2012-Jul-01')
      (
        SUBPARTITION q2_europe VALUES ('FRANCE', 'ITALY'),
        SUBPARTITION q2_asia VALUES ('INDIA', 'PAKISTAN'),
        SUBPARTITION q2_americas VALUES ('US', 'CANADA')
      ),
  PARTITION q3_2012
    VALUES LESS THAN('2012-Oct-01')
      (
        SUBPARTITION q3_europe VALUES ('FRANCE', 'ITALY'),
        SUBPARTITION q3_asia VALUES ('INDIA', 'PAKISTAN'),
        SUBPARTITION q3_americas VALUES ('US', 'CANADA')
      ),
  PARTITION q4_2012
    VALUES LESS THAN('2013-Jan-01')
      (
        SUBPARTITION q4_europe VALUES ('FRANCE', 'ITALY'),
        SUBPARTITION q4_asia VALUES ('INDIA', 'PAKISTAN'),
        SUBPARTITION q4_americas VALUES ('US', 'CANADA')
      )
);
```

This statement creates a table with four partitions; each partition has three subpartitions.

```
edb=# SELECT subpartition_name, high_value, partition_name FROM
ALL_TAB_SUBPARTITIONS;
 subpartition_name |     high_value      | partition_name
-------------------+---------------------+----------------
 Q1_EUROPE         | 'FRANCE', 'ITALY'   | Q1_2012
 Q1_ASIA           | 'INDIA', 'PAKISTAN' | Q1_2012
 Q1_AMERICAS       | 'US', 'CANADA'      | Q1_2012
 Q2_EUROPE         | 'FRANCE', 'ITALY'   | Q2_2012
 Q2_ASIA           | 'INDIA', 'PAKISTAN' | Q2_2012
 Q2_AMERICAS       | 'US', 'CANADA'      | Q2_2012
 Q3_EUROPE         | 'FRANCE', 'ITALY'   | Q3_2012
 Q3_ASIA           | 'INDIA', 'PAKISTAN' | Q3_2012
 Q3_AMERICAS       | 'US', 'CANADA'      | Q3_2012
 Q4_EUROPE         | 'FRANCE', 'ITALY'   | Q4_2012
 Q4_ASIA           | 'INDIA', 'PAKISTAN' | Q4_2012
 Q4_AMERICAS       | 'US', 'CANADA'      | Q4_2012
(12 rows)
```

When a row is added to this table, the value in the `date` column is compared to the values specified in the range partitioning rules, and the server selects the partition in which the row should reside. The value in the `country` column is then compared to the values specified in the list subpartitioning rules; when the server locates a match for the value, the row is stored in the corresponding subpartition.

Any row added to the table will be stored in a subpartition, so the partitions will contain no data.

The server would evaluate the following statement against the partitioning and subpartitioning rules and store the row in the `q3_europe` partition.

```
INSERT INTO sales VALUES (10, '9519a', 'FRANCE', '18-Aug-2012',
'650000');
```

4.2 ALTER TABLE... ADD PARTITION

Use the `ALTER TABLE... ADD PARTITION` command to add a partition to an existing partitioned table. The syntax is:

```
ALTER TABLE <table_name> ADD PARTITION <partition_definition>;

Where partition_definition is:

        {<list_partition> | <range_partition> }

and list_partition is:

        PARTITION [<partition_name>]
          VALUES (<value>[, <value>]...)
          [TABLESPACE <tablespace_name>]
          [(<subpartition>, ...)]

and range_partition is:

        PARTITION [<partition_name>]
          VALUES LESS THAN (<value>[, <value>]...)
          [TABLESPACE <tablespace_name>]
          [(<subpartition>, ...)]

Where subpartition is:

        {<list_subpartition> | <range_subpartition> | <hash_subpartition>}

and list_subpartition is:

        SUBPARTITION [<subpartition_name>]
          VALUES (<value>[, <value>]...)
          [TABLESPACE <tablespace_name>]

and range_subpartition is:

        SUBPARTITION [<subpartition_name> ]
          VALUES LESS THAN (<value>[, <value>]...)
          [TABLESPACE <tablespace_name>]

and hash_subpartition is:

        SUBPARTITION <subpartition_name>
          [TABLESPACE <tablespace_name>] |
        SUBPARTITIONS <num> [STORE IN ( <tablespace_name> [,
          <tablespace_name>]... ) ]
```

Description

The ALTER TABLE... ADD PARTITION command adds a partition to an existing partitioned table. There is no upper limit to the number of defined partitions in a partitioned table.

New partitions must be of the same type (LIST, RANGE, or HASH) as existing partitions. The new partition rules must reference the same column specified in the partitioning rules that define the existing partition(s).

You can use the ALTER TABLE... ADD PARTITION statement to add a partition to a table with a DEFAULT rule as long as there are no conflicting values between existing rows in the table and the values of the partition to be added.

You cannot use the ALTER TABLE... ADD PARTITION statement to add a partition to a table with a MAXVALUE rule.

You can alternatively use the ALTER TABLE... SPLIT PARTITION statement to split an existing partition, effectively increasing the number of partitions in a table.

RANGE partitions must be specified in ascending order. You cannot add a new partition that precedes existing partitions in a RANGE partitioned table.

Include the TABLESPACE clause to specify the tablespace in which the new partition will reside. If you do not specify a tablespace, the partition will reside in the default tablespace.

Use the STORE IN clause to specify the tablespace list across which the autogenerated subpartitions will be stored.

If the table is indexed, the index will be created on the new partition.

To use the ALTER TABLE... ADD PARTITION command you must be the table owner, or have superuser (or administrative) privileges.

Parameters

table_name

The name (optionally schema-qualified) of the partitioned table.

partition_name

The name of the partition to be created. Partition names must be unique amongst all partitions and subpartitions, and must follow the naming conventions for object identifiers.

subpartition_name

The name of the subpartition to be created. Subpartition names must be unique amongst all partitions and subpartitions, and must follow the naming conventions for object identifiers.

(value[, value]...)

Use value to specify a quoted literal value (or comma-delimited list of literal values) by which rows will be distributed into partitions. Each partitioning rule must specify at least one value, but there is no limit placed on the number of values specified within a rule. value may also be NULL, DEFAULT (if specifying a LIST partition), or MAXVALUE (if specifying a RANGE partition).

For information about creating a DEFAULT or MAXVALUE partition, see *Handling Stray Values in a LIST or RANGE Partitioned Table*.

num

The SUBPARTITIONS num clause is only supported for HASH and can be used to specify the number of hash subpartitions. Alternatively, you can use the subpartition definition to specify individual subpartitions and their tablespaces. If no SUBPARTITIONS or SUBPARTITION DEFINITION is specified, then the partition creates a subpartition based on the SUBPARTITION TEMPLATE.

tablespace_name

The name of the tablespace in which a partition or subpartition resides.

4.2.1 Example - Adding a Partition to a LIST Partitioned Table

The example that follows adds a partition to the list-partitioned sales table. The table was created using the command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

The table contains three partitions (americas, asia, and europe).

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |     high_value
----------------+---------------------
 EUROPE         | 'FRANCE', 'ITALY'
 ASIA           | 'INDIA', 'PAKISTAN'
 AMERICAS       | 'US', 'CANADA'
(3 rows)
```

The following command adds a partition named east_asia to the sales table.

```
ALTER TABLE sales ADD PARTITION east_asia
  VALUES ('CHINA', 'KOREA');
```

After invoking the command, the table includes the east_asia partition.

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |     high_value
----------------+---------------------
```

(continues on next page)

(continued from previous page)

```
 EUROPE          | 'FRANCE', 'ITALY'
 ASIA            | 'INDIA', 'PAKISTAN'
 AMERICAS        | 'US', 'CANADA'
 EAST_ASIA       | 'CHINA', 'KOREA'
(4 rows)
```

4.2.2 Example - Adding a Partition to a RANGE Partitioned Table

The example that follows adds a partition to a range-partitioned table named `sales`:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date)
(
  PARTITION q1_2012
    VALUES LESS THAN('2012-Apr-01'),
  PARTITION q2_2012
    VALUES LESS THAN('2012-Jul-01'),
  PARTITION q3_2012
    VALUES LESS THAN('2012-Oct-01'),
  PARTITION q4_2012
    VALUES LESS THAN('2013-Jan-01')
);
```

The table contains four partitions (`q1_2012, q2_2012, q3_2012`, and `q4_2012`).

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |      high_value
----------------+----------------------
 Q1_2012        | '01-APR-12 00:00:00'
 Q2_2012        | '01-JUL-12 00:00:00'
 Q3_2012        | '01-OCT-12 00:00:00'
 Q4_2012        | '01-JAN-13 00:00:00'
(4 rows)
```

The following command adds a partition named `q1_2013` to the `sales` table.

```
ALTER TABLE sales ADD PARTITION q1_2013
  VALUES LESS THAN('01-APR-2013');
```

After invoking the command, the table includes the `q1_2013` partition.

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |      high_value
```

(continues on next page)

(continued from previous page)

```
----------------+----------------------
 Q1_2012        | '01-APR-12 00:00:00'
 Q2_2012        | '01-JUL-12 00:00:00'
 Q3_2012        | '01-OCT-12 00:00:00'
 Q4_2012        | '01-JAN-13 00:00:00'
 Q1_2013        | '01-APR-13 00:00:00'
(5 rows)
```

4.2.3 Example - Adding a Partition with SUBPARTITIONS num... IN PARTITION DESCRIPTION

The following example adds a partition to a list-partitioned `sales` table, you can specify a `SUBPARTITIONS` clause to add a specified number of subpartitions. The `sales` table was created with the command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST (country) SUBPARTITION BY HASH (part_no) SUBPARTITIONS 2
(
  PARTITION europe VALUES ('FRANCE', 'ITALY'),
  PARTITION asia  VALUES ('INDIA', 'PAKISTAN')
);
```

The table contains two partitions `europe` and `asia`, each containing two subpartitions. Because the subpartitions are not named, system-generated names are assigned to them.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | ASIA           | SYS0103
 SALES      | ASIA           | SYS0104
 SALES      | EUROPE         | SYS0101
 SALES      | EUROPE         | SYS0102
(4 rows)
```

The following command adds a new partition `americas` to the `sales` table and will create a number of subpartitions as specified in the partition description.

```
ALTER TABLE sales ADD PARTITION americas
  VALUES ('US', 'CANADA');
```

After invoking the command the table includes the partition `americas` and two newly added subpartitions.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | AMERICAS       | SYS0107
 SALES      | AMERICAS       | SYS0108
 SALES      | ASIA           | SYS0103
 SALES      | ASIA           | SYS0104
 SALES      | EUROPE         | SYS0101
 SALES      | EUROPE         | SYS0102
(6 rows)
```

4.2.4 Example - Adding a Partition with SUBPARTITIONS num...

The following example adds a partition a list-partitioned table `sales` consisting of three subpartitions. The `sales` table was created with the command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST (country) SUBPARTITION BY HASH (part_no) SUBPARTITIONS 3
(
  PARTITION europe VALUES ('FRANCE', 'ITALY'),
  PARTITION asia  VALUES ('INDIA', 'PAKISTAN')
);
```

The table contains partitions `europe` and `asia`, each containing three subpartitions.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | ASIA           | SYS0104
 SALES      | ASIA           | SYS0105
 SALES      | ASIA           | SYS0106
 SALES      | EUROPE         | SYS0101
 SALES      | EUROPE         | SYS0102
 SALES      | EUROPE         | SYS0103
(6 rows)
```

The following command adds a new partition `americas` and five subpartitions as specified in the `ADD PARTITION` clause.

```
ALTER TABLE sales ADD PARTITION americas
  VALUES ('US', 'CANADA') SUBPARTITIONS 5;
```

After invoking the command sales table includes the partition americas and five newly added subpartitions.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' and partition_name =
'AMERICAS' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | AMERICAS       | SYS0109
 SALES      | AMERICAS       | SYS0110
 SALES      | AMERICAS       | SYS0111
 SALES      | AMERICAS       | SYS0112
 SALES      | AMERICAS       | SYS0113
(5 rows)
```

4.2.5 Example - Adding a Partition with SUBPARTITIONS num... STORE IN

The following example adds a partition to a list-partitioned table sales consisting of three subpartitions. The table was created using the command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST (country) SUBPARTITION BY HASH (part_no) SUBPARTITIONS 3
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

The table contains three partitions (americas, asia, and europe), each containing three subpartitions with system-generated names.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | AMERICAS       | SYS0109
 SALES      | AMERICAS       | SYS0107
 SALES      | AMERICAS       | SYS0108
 SALES      | ASIA           | SYS0105
 SALES      | ASIA           | SYS0104
 SALES      | ASIA           | SYS0106
 SALES      | EUROPE         | SYS0101
 SALES      | EUROPE         | SYS0103
```

(continues on next page)

(continued from previous page)

```
 SALES      | EUROPE         | SYS0102
(9 rows)
```

The following command adds a new partition `east_asia` with five subpartitions as specified in the `ADD PARTITION` clause and stores them in the tablespace named `(ts1)`.

```
ALTER TABLE sales ADD PARTITION east_asia
  VALUES ('CHINA', 'KOREA') SUBPARTITIONS 5 STORE IN (ts1);
```

After invoking the command table includes the partition `east_asia` and five newly added subpartitions stored in tablespace `(ts1)`.

```
edb=# SELECT table_name, partition_name, subpartition_name, tablespace_name
FROM ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' and partition_name =
'EAST_ASIA' ORDER BY 1,2;
 table_name | partition_name | subpartition_name | tablespace_name
------------+----------------+-------------------+-----------------
 SALES      | EAST_ASIA      | SYS0113           | TS1
 SALES      | EAST_ASIA      | SYS0114           | TS1
 SALES      | EAST_ASIA      | SYS0115           | TS1
 SALES      | EAST_ASIA      | SYS0116           | TS1
 SALES      | EAST_ASIA      | SYS0117           | TS1
(5 rows)
```

4.3 ALTER TABLE... ADD SUBPARTITION

The ALTER TABLE... ADD SUBPARTITION command adds a subpartition to an existing subpartitioned partition. The syntax is:

```
ALTER TABLE <table_name> MODIFY PARTITION <partition_name>
      ADD SUBPARTITION <subpartition_definition>;

Where subpartition_definition is:

{<list_subpartition> | <range_subpartition>}

and list_subpartition is:

        SUBPARTITION [<subpartition_name>]
          VALUES (<value>[, <value>]...)
          [TABLESPACE <tablespace_name>]

and range_subpartition is:

        SUBPARTITION [subpartition_name]
          VALUES LESS THAN (value[, value]...)
          [TABLESPACE tablespace_name]
```

Description

The ALTER TABLE... ADD SUBPARTITION command adds a subpartition to an existing partition; the partition must already be subpartitioned. There is no upper limit to the number of defined subpartitions.

New subpartitions must be of the same type (LIST, RANGE or HASH) as existing subpartitions. The new subpartition rules must reference the same column specified in the subpartitioning rules that define the existing subpartition(s).

You can use the ALTER TABLE... ADD SUBPARTITION statement to add a subpartition to a table with a DEFAULT rule as long as there are no conflicting values between existing rows in the table and the values of the subpartition to be added.

You cannot use the ALTER TABLE... ADD SUBPARTITION statement to add a subpartition to a table with a MAXVALUE rule.

You can split an existing subpartition with the ALTER TABLE... SPLIT SUBPARTITION statement, effectively adding a subpartition to a table.

You cannot add a new subpartition that precedes existing subpartitions in a range subpartitioned table; range subpartitions must be specified in ascending order.

Include the TABLESPACE clause to specify the tablespace in which the subpartition will reside. If you do not specify a tablespace, the subpartition will be created in the default tablespace.

If the table is indexed, the index will be created on the new subpartition.

To use the ALTER TABLE... ADD SUBPARTITION command you must be the table owner, or have superuser (or administrative) privileges.

Parameters

`table_name`

The name (optionally schema-qualified) of the partitioned table in which the subpartition will reside.

`partition_name`

The name of the partition in which the new subpartition will reside.

`subpartition_name`

The name of the subpartition to be created. Subpartition names must be unique amongst all partitions and subpartitions, and must follow the naming conventions for object identifiers.

`(value[, value]...)`

Use `value` to specify a quoted literal value (or comma-delimited list of literal values) by which table entries will be grouped into partitions. Each partitioning rule must specify at least one value, but there is no limit placed on the number of values specified within a rule. `value` may also be `NULL, DEFAULT` (if specifying a `LIST` partition), or `MAXVALUE` (if specifying a `RANGE` partition).

For information about creating a `DEFAULT` or `MAXVALUE` partition, see *Handling Stray Values in a LIST or RANGE Partitioned Table*.

`tablespace_name`

The name of the tablespace in which the subpartition resides.

4.3.1 Example - Adding a Subpartition to a LIST/RANGE Partitioned Table

The following example adds a `RANGE` subpartition to the list-partitioned `sales` table. The `sales` table was created with the command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country)
  SUBPARTITION BY RANGE(date)
(
  PARTITION europe VALUES('FRANCE', 'ITALY')
    (
      SUBPARTITION europe_2011
        VALUES LESS THAN('2012-Jan-01'),
      SUBPARTITION europe_2012
        VALUES LESS THAN('2013-Jan-01')
    ),
  PARTITION asia VALUES('INDIA', 'PAKISTAN')
    (
```

(continues on next page)

(continued from previous page)

```
      SUBPARTITION asia_2011
        VALUES LESS THAN('2012-Jan-01'),
      SUBPARTITION asia_2012
        VALUES LESS THAN('2013-Jan-01')
    ),
  PARTITION americas VALUES('US', 'CANADA')
    (
      SUBPARTITION americas_2011
        VALUES LESS THAN('2012-Jan-01'),
      SUBPARTITION americas_2012
        VALUES LESS THAN('2013-Jan-01')
    )
);
```

The `sales` table has three partitions, named `europe, asia`, and `americas`. Each partition has two range-defined subpartitions.

```
edb=# SELECT partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS;
 partition_name | subpartition_name |      high_value
----------------+-------------------+----------------------
 EUROPE         | EUROPE_2011       | '01-JAN-12 00:00:00'
 EUROPE         | EUROPE_2012       | '01-JAN-13 00:00:00'
 ASIA           | ASIA_2011         | '01-JAN-12 00:00:00'
 ASIA           | ASIA_2012         | '01-JAN-13 00:00:00'
 AMERICAS       | AMERICAS_2011     | '01-JAN-12 00:00:00'
 AMERICAS       | AMERICAS_2012     | '01-JAN-13 00:00:00'
(6 rows)
```

The following command adds a subpartition named `europe_2013`.

```
ALTER TABLE sales MODIFY PARTITION europe
  ADD SUBPARTITION europe_2013
  VALUES LESS THAN('2015-Jan-01');
```

After invoking the command, the table includes a subpartition named `europe_2013`.

```
edb=# SELECT partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS;
 partition_name | subpartition_name |      high_value
----------------+-------------------+----------------------
 EUROPE         | EUROPE_2011       | '01-JAN-12 00:00:00'
 EUROPE         | EUROPE_2012       | '01-JAN-13 00:00:00'
 EUROPE         | EUROPE_2013       | '01-JAN-15 00:00:00'
 ASIA           | ASIA_2011         | '01-JAN-12 00:00:00'
 ASIA           | ASIA_2012         | '01-JAN-13 00:00:00'
 AMERICAS       | AMERICAS_2011     | '01-JAN-12 00:00:00'
 AMERICAS       | AMERICAS_2012     | '01-JAN-13 00:00:00'
(7 rows)
```

Note: When adding a new range subpartition, the subpartitioning rules must specify a range that falls *after*

any existing subpartitions.

4.3.2 Example - Adding a Subpartition to a RANGE/LIST Partitioned Table

The following example adds a `LIST` subpartition to the `RANGE` partitioned `sales` table. The `sales` table was created with the command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date)
  SUBPARTITION BY LIST (country)
  (
    PARTITION first_half_2012 VALUES LESS THAN('01-JUL-2012')
    (
      SUBPARTITION europe VALUES ('ITALY', 'FRANCE'),
      SUBPARTITION americas VALUES ('US', 'CANADA')
    ),
    PARTITION second_half_2012 VALUES LESS THAN('01-JAN-2013')
    (
      SUBPARTITION asia VALUES ('INDIA', 'PAKISTAN')
    )
);
```

After executing the above command, the `sales` table will have two partitions, named `first_half_2012` and `second_half_2012`. The `first_half_2012 partition` has two subpartitions, named `europe` and `americas`, and the `second_half_2012` partition has one partition, named `asia`.

```
edb=# SELECT partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS;
  partition_name  | subpartition_name |     high_value
------------------+-------------------+---------------------
 SECOND_HALF_2012 | ASIA              | 'INDIA', 'PAKISTAN'
 FIRST_HALF_2012  | AMERICAS          | 'US', 'CANADA'
 FIRST_HALF_2012  | EUROPE            | 'ITALY', 'FRANCE'
(3 rows)
```

The following command adds a subpartition to the `second_half_2012` partition, named `east_asia`.

```
ALTER TABLE sales MODIFY PARTITION second_half_2012
  ADD SUBPARTITION east_asia VALUES ('CHINA');
```

After invoking the command, the table includes a subpartition named `east_asia`.

```
edb=# SELECT partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS;
  partition_name  | subpartition_name |     high_value
------------------+-------------------+---------------------
 SECOND_HALF_2012 | ASIA              | 'INDIA', 'PAKISTAN'
 SECOND_HALF_2012 | EAST_ASIA         | 'CHINA'
 FIRST_HALF_2012  | AMERICAS          | 'US', 'CANADA'
 FIRST_HALF_2012  | EUROPE            | 'ITALY', 'FRANCE'
(4 rows)
```

4.4 ALTER TABLE...SPLIT PARTITION

Use the ALTER TABLE... SPLIT PARTITION command to divide a single partition into two partitions, maintaining the partitioning of the original table in the newly created partitions, and redistributing the partition's contents between the new partitions. The command syntax comes in two forms.

The first form splits a RANGE partition into two partitions:

```
ALTER TABLE <table_name> SPLIT PARTITION <partition_name>
  AT (<range_part_value>)
  INTO
  (
    PARTITION <new_part1>
     [TABLESPACE <tablespace_name>]
     [SUBPARTITIONS <num>] [STORE IN ( <tablespace_name> [,
     <tablespace_name>]... ) ],
    PARTITION <new_part2>
     [TABLESPACE <tablespace_name>]
     [SUBPARTITIONS <num>] [STORE IN ( <tablespace_name> [,
     <tablespace_name>]... ) ]
  );
```

The second form splits a LIST partition into two partitions:

```
ALTER TABLE <table_name> SPLIT PARTITION <partition_name>
  VALUES (<value>[, <value>]...)
  INTO
  (
    PARTITION <new_part1>
     [TABLESPACE <tablespace_name>]
     [SUBPARTITIONS <num>] [STORE IN ( <tablespace_name> [,
     <tablespace_name>]... ) ],
    PARTITION <new_part2>
     [TABLESPACE <tablespace_name>]
     [SUBPARTITIONS <num>] [STORE IN ( <tablespace_name> [,
     <tablespace_name>]... ) ]
  );
```

Description

The ALTER TABLE...SPLIT PARTITION command adds a partition to an existing LIST or RANGE partitioned table. Please note that the ALTER TABLE... SPLIT PARTITION command cannot add a partition to a HASH partitioned table. There is no upper limit to the number of partitions that a table may have.

When you execute an ALTER TABLE...SPLIT PARTITION command, Advanced Server creates two new partitions, maintains the partitioning of the original table in the newly created partitions, and redistributes the content of the old partition between them (as constrained by the partitioning rules).

Include the TABLESPACE clause to specify the tablespace in which a partition will reside. If you do not specify a tablespace, the partition will reside in the tablespace of the original partitioned table.

Use the STORE IN clause to specify the tablespace list across which the autogenerated subpartitions will

be stored.

If the table is indexed, the index will be created on the new partition.

To use the ALTER TABLE... SPLIT PARTITION command you must be the table owner, or have superuser (or administrative) privileges.

Parameters

table_name

The name (optionally schema-qualified) of the partitioned table.

partition_name

The name of the partition that is being split.

new_part1

The name of the first new partition to be created. Partition names must be unique amongst all partitions and subpartitions, and must follow the naming conventions for object identifiers.

new_part1 will receive the rows that meet the partitioning constraints specified in the ALTER TABLE... SPLIT PARTITION command.

new_part2

The name of the second new partition to be created. Partition names must be unique amongst all partitions and subpartitions, and must follow the naming conventions for object identifiers.

new_part2 will receive the rows are not directed to new_part1 by the partitioning constraints specified in the ALTER TABLE... SPLIT PARTITION command.

range_part_value

Use range_part_value to specify the boundary rules by which to create the new partition. The partitioning rule must contain at least one column of a data type that has two operators (i.e., a greater-than-or-equal to operator, and a less-than operator). Range boundaries are evaluated against a LESS THAN clause and are non-inclusive; a date boundary of January 1, 2010 will include only those date values that fall on or before December 31, 2009.

(value[, value]...)

Use value to specify a quoted literal value (or comma-delimited list of literal values) by which rows will be distributed into partitions. Each partitioning rule must specify at least one value, but there is no limit placed on the number of values specified within a rule.

For information about creating a DEFAULT or MAXVALUE partition, see *Handling Stray Values in a LIST or RANGE Partitioned Table*.

num

The SUBPARTITIONS num clause is only supported for HASH subpartitions; use the clause to specify the number of hash subpartitions. Alternatively, you can use the subpartition definition to specify individual subpartitions and their tablespaces. If no SUBPARTITIONS or SUBPARTITION DEFINITION is specified, then the partition creates a subpartition based on the SUBPARTITION TEMPLATE.

`tablespace_name`

The name of the tablespace in which the partition or subpartition resides.

4.4.1 Example - Splitting a LIST Partition

This example will divide one of the partitions in the list-partitioned `sales` table into two new partitions, and redistribute the contents of the partition between them. The `sales` table is created with the statement:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

The table definition creates three partitions (`europe, asia`, and `americas`). The following command adds rows to each partition.

```
INSERT INTO sales VALUES
  (10, '4519b', 'FRANCE', '17-Jan-2012', '45000'),
  (20, '3788a', 'INDIA', '01-Mar-2012', '75000'),
  (40, '9519b', 'US', '12-Apr-2012', '145000'),
  (20, '3788a', 'PAKISTAN', '04-Jun-2012', '37500'),
  (40, '4577b', 'US', '11-Nov-2012', '25000'),
  (30, '7588b', 'CANADA', '14-Dec-2012', '50000'),
  (30, '9519b', 'CANADA', '01-Feb-2012', '75000'),
  (30, '4519b', 'CANADA', '08-Apr-2012', '120000'),
  (40, '3788a', 'US', '12-May-2012', '4950'),
  (10, '9519b', 'ITALY', '07-Jul-2012', '15000'),
  (10, '9519a', 'FRANCE', '18-Aug-2012', '650000'),
  (10, '9519b', 'FRANCE', '18-Aug-2012', '650000'),
  (20, '3788b', 'INDIA', '21-Sept-2012', '5090'),
  (40, '4788a', 'US', '23-Sept-2012', '4950'),
  (40, '4788b', 'US', '09-Oct-2012', '15000'),
  (20, '4519a', 'INDIA', '18-Oct-2012', '650000'),
  (20, '4519b', 'INDIA', '2-Dec-2012', '5090');
```

The rows are distributed amongst the partitions.

```
edb=# SELECT tableoid::regclass, * FROM sales;
    tableoid    | dept_no | part_no | country  |        date         | amount
----------------+---------+---------+----------+---------------------+--------
 sales_americas |      40 | 9519b   | US       | 12-APR-12 00:00:00 | 145000
```

(continues on next page)

(continued from previous page)

```
 sales_americas |      40 | 4577b   | US       | 11-NOV-12 00:00:00 |  25000
 sales_americas |      30 | 7588b   | CANADA   | 14-DEC-12 00:00:00 |  50000
 sales_americas |      30 | 9519b   | CANADA   | 01-FEB-12 00:00:00 |  75000
 sales_americas |      30 | 4519b   | CANADA   | 08-APR-12 00:00:00 | 120000
 sales_americas |      40 | 3788a   | US       | 12-MAY-12 00:00:00 |   4950
 sales_americas |      40 | 4788a   | US       | 23-SEP-12 00:00:00 |   4950
 sales_americas |      40 | 4788b   | US       | 09-OCT-12 00:00:00 |  15000
 sales_europe   |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_europe   |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_europe   |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_europe   |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_asia     |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_asia     |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_asia     |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_asia     |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_asia     |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
(17 rows)
```

The following command splits the `americas` partition into two partitions named `us` and `canada`.

```
ALTER TABLE sales SPLIT PARTITION americas
  VALUES ('US')
  INTO (PARTITION us, PARTITION canada);
```

A `SELECT` statement confirms that the rows have been redistributed.

```
edb=# SELECT tableoid::regclass, * FROM sales;
   tableoid   | dept_no | part_no | country  |        date         | amount
--------------+---------+---------+----------+---------------------+--------
 sales_canada |      30 | 7588b   | CANADA   | 14-DEC-12 00:00:00 |  50000
 sales_canada |      30 | 9519b   | CANADA   | 01-FEB-12 00:00:00 |  75000
 sales_canada |      30 | 4519b   | CANADA   | 08-APR-12 00:00:00 | 120000
 sales_europe |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_europe |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_europe |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_europe |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_asia   |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_asia   |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_asia   |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_asia   |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_asia   |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
 sales_us     |      40 | 9519b   | US       | 12-APR-12 00:00:00 | 145000
 sales_us     |      40 | 4577b   | US       | 11-NOV-12 00:00:00 |  25000
 sales_us     |      40 | 3788a   | US       | 12-MAY-12 00:00:00 |   4950
 sales_us     |      40 | 4788a   | US       | 23-SEP-12 00:00:00 |   4950
 sales_us     |      40 | 4788b   | US       | 09-OCT-12 00:00:00 |  15000
(17 rows)
```

4.4.2 Example - Splitting a RANGE Partition

This example divides the `q4_2012` partition (of the range-partitioned `sales` table) into two partitions, and redistribute the partition's contents. Use the following command to create the `sales` table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date)
(
  PARTITION q1_2012
    VALUES LESS THAN('2012-Apr-01'),
  PARTITION q2_2012
    VALUES LESS THAN('2012-Jul-01'),
  PARTITION q3_2012
    VALUES LESS THAN('2012-Oct-01'),
  PARTITION q4_2012
    VALUES LESS THAN('2013-Jan-01')
);
```

The table definition creates four partitions (`q1_2012, q2_2012, q3_2012`, and `q4_2012`). The following command adds rows to each partition.

```
INSERT INTO sales VALUES
  (10, '4519b', 'FRANCE', '17-Jan-2012', '45000'),
  (20, '3788a', 'INDIA', '01-Mar-2012', '75000'),
  (40, '9519b', 'US', '12-Apr-2012', '145000'),
  (20, '3788a', 'PAKISTAN', '04-Jun-2012', '37500'),
  (40, '4577b', 'US', '11-Nov-2012', '25000'),
  (30, '7588b', 'CANADA', '14-Dec-2012', '50000'),
  (30, '9519b', 'CANADA', '01-Feb-2012', '75000'),
  (30, '4519b', 'CANADA', '08-Apr-2012', '120000'),
  (40, '3788a', 'US', '12-May-2012', '4950'),
  (10, '9519b', 'ITALY', '07-Jul-2012', '15000'),
  (10, '9519a', 'FRANCE', '18-Aug-2012', '650000'),
  (10, '9519b', 'FRANCE', '18-Aug-2012', '650000'),
  (20, '3788b', 'INDIA', '21-Sept-2012', '5090'),
  (40, '4788a', 'US', '23-Sept-2012', '4950'),
  (40, '4788b', 'US', '09-Oct-2012', '15000'),
  (20, '4519a', 'INDIA', '18-Oct-2012', '650000'),
  (20, '4519b', 'INDIA', '2-Dec-2012', '5090');
```

A `SELECT` statement confirms that the rows are distributed amongst the partitions as expected.

```
edb=# SELECT tableoid::regclass, * FROM sales;
   tableoid    | dept_no | part_no | country  |        date         | amount
---------------+---------+---------+----------+---------------------+--------
```

(continues on next page)

(continued from previous page)

```
 sales_q1_2012 |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_q1_2012 |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_q1_2012 |      30 | 9519b   | CANADA   | 01-FEB-12 00:00:00 |  75000
 sales_q2_2012 |      40 | 9519b   | US       | 12-APR-12 00:00:00 | 145000
 sales_q2_2012 |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_q2_2012 |      30 | 4519b   | CANADA   | 08-APR-12 00:00:00 | 120000
 sales_q2_2012 |      40 | 3788a   | US       | 12-MAY-12 00:00:00 |   4950
 sales_q3_2012 |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_q3_2012 |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_q3_2012 |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_q3_2012 |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_q3_2012 |      40 | 4788a   | US       | 23-SEP-12 00:00:00 |   4950
 sales_q4_2012 |      40 | 4577b   | US       | 11-NOV-12 00:00:00 |  25000
 sales_q4_2012 |      30 | 7588b   | CANADA   | 14-DEC-12 00:00:00 |  50000
 sales_q4_2012 |      40 | 4788b   | US       | 09-OCT-12 00:00:00 |   15000
 sales_q4_2012 |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_q4_2012 |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
(17 rows)
```

The following command splits the `q4_2012` partition into two partitions named `q4_2012_p1` and `q4_2012_p2`.

```
ALTER TABLE sales SPLIT PARTITION q4_2012
  AT ('15-Nov-2012')
  INTO
  (
    PARTITION q4_2012_p1,
    PARTITION q4_2012_p2
  );
```

A `SELECT` statement confirms that the rows have been redistributed across the new partitions.

```
edb=# SELECT tableoid::regclass, * FROM sales;
   tableoid      | dept_no | part_no | country  |        date         |amount
-----------------+---------+---------+----------+---------------------+------
 sales_q1_2012   |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 | 45000
 sales_q1_2012   |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 | 75000
 sales_q1_2012   |      30 | 9519b   | CANADA   | 01-FEB-12 00:00:00 | 75000
 sales_q2_2012   |      40 | 9519b   | US       | 12-APR-12 00:00:00 |145000
 sales_q2_2012   |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 | 37500
 sales_q2_2012   |      30 | 4519b   | CANADA   | 08-APR-12 00:00:00 |120000
 sales_q2_2012   |      40 | 3788a   | US       | 12-MAY-12 00:00:00 |  4950
 sales_q3_2012   |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 | 15000
 sales_q3_2012   |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 |650000
 sales_q3_2012   |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 |650000
 sales_q3_2012   |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |  5090
 sales_q3_2012   |      40 | 4788a   | US       | 23-SEP-12 00:00:00 |  4950
 sales_q4_2012_p1|      40 | 4577b   | US       | 11-NOV-12 00:00:00 | 25000
 sales_q4_2012_p1|      40 | 4788b   | US       | 09-OCT-12 00:00:00 | 15000
 sales_q4_2012_p1|      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 |650000
 sales_q4_2012_p2|      30 | 7588b   | CANADA   | 14-DEC-12 00:00:00 | 50000
```

(continues on next page)

(continued from previous page)

```
 sales_q4_2012_p2|      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |  5090
(17 rows)
```

4.4.3 Example - Splitting a RANGE/LIST Partition

The following example will divide one of the partitions in the range-partitioned `sales` table into new partitions, maintaining the partitioning of the original table in the newly created partitions, and redistributing the contents of the partition between them.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date)
  SUBPARTITION BY LIST (country)
  (
    PARTITION q1_2012 VALUES LESS THAN('2012-Apr-01')
    (
      SUBPARTITION europe VALUES('FRANCE', 'ITALY'),
      SUBPARTITION americas VALUES('US', 'CANADA'),
      SUBPARTITION asia VALUES('INDIA', 'PAKISTAN')
    )
  );
```

The `sales` table contains partition `q1_2012` and three subpartitions `europe, americas`, and `asia`.

```
edb=# SELECT table_name, partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name |     high_value
------------+----------------+-------------------+---------------------
 SALES      | Q1_2012        | EUROPE            | 'FRANCE', 'ITALY'
 SALES      | Q1_2012        | AMERICAS          | 'US', 'CANADA'
 SALES      | Q1_2012        | ASIA              | 'INDIA', 'PAKISTAN'
(3 rows)
```

The following command splits the `q1_2012` partition into two partitions named `q1_2012_p1` and `q1_2012_p2`.

```
ALTER TABLE sales SPLIT PARTITION q1_2012
  AT ('01-Mar-2012')
  INTO
  (
    PARTITION q1_2012_p1,
    PARTITION q1_2012_p2
  );
```

A SELECT statement confirms that the same number of subpartitions is created in the newly created partitions q1_2012_p1 and q1_2012_p2 with system-generated names.

```
edb=# SELECT table_name, partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2,3;
 table_name | partition_name | subpartition_name |     high_value
------------+----------------+-------------------+--------------------
 SALES      | Q1_2012_P1     | SYS0105           | 'US', 'CANADA'
 SALES      | Q1_2012_P1     | SYS0106           | 'FRANCE', 'ITALY'
 SALES      | Q1_2012_P1     | SYS0107           | 'INDIA', 'PAKISTAN'
 SALES      | Q1_2012_P2     | SYS0108           | 'US', 'CANADA'
 SALES      | Q1_2012_P2     | SYS0109           | 'FRANCE', 'ITALY'
 SALES      | Q1_2012_P2     | SYS0110           | 'INDIA', 'PAKISTAN'
(6 rows)
```

4.4.4 Example - Splitting a Partition with SUBPARTITIONS num...

The following example will divide one of the partitions in the list-partitioned sales table into new partitions, maintaining the partitioning of the original table in the newly created partitions, and redistributing the contents of the partition between them. The SUBPARTITIONS clause lets you add a specified number of subpartitions. If no SUBPARTITIONS clause is specified then the new partitions inherit the default number of subpartitions. The sales table is created with the statement:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country) SUBPARTITION BY HASH (part_no) SUBPARTITIONS 2
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

The table definition creates three partitions (europe, asia, and americas) each containing two subpartitions.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | AMERICAS       | SYS0105
 SALES      | AMERICAS       | SYS0106
 SALES      | ASIA           | SYS0103
 SALES      | ASIA           | SYS0104
 SALES      | EUROPE         | SYS0101
```

(continues on next page)

(continued from previous page)

```
 SALES      | EUROPE          | SYS0102
(6 rows)
```

The following command splits the `americas` partition into two partitions named `partition_us` and `partition_canada`.

```
ALTER TABLE sales SPLIT PARTITION americas VALUES ('US') INTO (PARTITION
partition_us SUBPARTITIONS 5, PARTITION partition_canada);
```

A `SELECT` statement confirms that the `americas` partition is split into `partition_us` and `partition_canada`. The `partition_us` contains five subpartitions and `partition_canada` contains default two subpartitions and assigned system-generated names.

```
edb=#  SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2,3;
 table_name |  partition_name   | subpartition_name
------------+-------------------+-------------------
 SALES      | ASIA              | SYS0103
 SALES      | ASIA              | SYS0104
 SALES      | EUROPE            | SYS0101
 SALES      | EUROPE            | SYS0102
 SALES      | PARTITION_CANADA  | SYS0115
 SALES      | PARTITION_CANADA  | SYS0116
 SALES      | PARTITION_US      | SYS0110
 SALES      | PARTITION_US      | SYS0111
 SALES      | PARTITION_US      | SYS0112
 SALES      | PARTITION_US      | SYS0113
 SALES      | PARTITION_US      | SYS0114
(11 rows)
```

4.4.5 Example - Splitting a Partition with SUBPARTITIONS num. . . STORE IN

The following example will divide the `europe` partition of the list-partitioned `sales` table into two partitions, maintaining the partitioning of the original table in the newly created partitions, and redistributing the partition's contents. The `SUBPARTITIONS` clause lets you add a specified number of subpartitions. Use the following command to create the `sales` table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country) SUBPARTITION BY HASH (part_no) SUBPARTITIONS 4
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
```

(continues on next page)

(continued from previous page)

```
  PARTITION americas VALUES('US', 'CANADA')
);
```

The sales table contains three partitions (europe, asia, and americas) each containing four subpartitions with system-generated names assigned to them.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | AMERICAS       | SYS0112
 SALES      | AMERICAS       | SYS0111
 SALES      | AMERICAS       | SYS0109
 SALES      | AMERICAS       | SYS0110
 SALES      | ASIA           | SYS0107
 SALES      | ASIA           | SYS0105
 SALES      | ASIA           | SYS0106
 SALES      | ASIA           | SYS0108
 SALES      | EUROPE         | SYS0101
 SALES      | EUROPE         | SYS0104
 SALES      | EUROPE         | SYS0103
 SALES      | EUROPE         | SYS0102
(12 rows)
```

The following command splits the europe partition into two partitions named france and italy.

```
ALTER TABLE sales SPLIT PARTITION europe VALUES ('FRANCE') INTO (PARTITION
france SUBPARTITIONS 10 STORE IN (ts1), PARTITION italy);
```

A SELECT statement confirms that the europe partition is split into two partitions. The partition france contains ten subpartitions that are stored in the tablespace ts1 and partition italy contains four subpartitions as in the original partition europe.

```
edb=# SELECT table_name, partition_name, subpartition_name, tablespace_name
FROM ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2,3;
 table_name | partition_name | subpartition_name | tablespace_name
------------+----------------+-------------------+-----------------
 SALES      | AMERICAS       | SYS0109           |
 SALES      | AMERICAS       | SYS0110           |
 SALES      | AMERICAS       | SYS0111           |
 SALES      | AMERICAS       | SYS0112           |
 SALES      | ASIA           | SYS0105           |
 SALES      | ASIA           | SYS0106           |
 SALES      | ASIA           | SYS0107           |
 SALES      | ASIA           | SYS0108           |
 SALES      | FRANCE         | SYS0116           | TS1
 SALES      | FRANCE         | SYS0117           | TS1
 SALES      | FRANCE         | SYS0118           | TS1
 SALES      | FRANCE         | SYS0119           | TS1
 SALES      | FRANCE         | SYS0120           | TS1
 SALES      | FRANCE         | SYS0121           | TS1
```

(continues on next page)

(continued from previous page)

```
 SALES      | FRANCE         | SYS0122           | TS1
 SALES      | FRANCE         | SYS0123           | TS1
 SALES      | FRANCE         | SYS0124           | TS1
 SALES      | FRANCE         | SYS0125           | TS1
 SALES      | ITALY          | SYS0126           |
 SALES      | ITALY          | SYS0127           |
 SALES      | ITALY          | SYS0128           |
 SALES      | ITALY          | SYS0129           |
(22 rows)
```

4.5 ALTER TABLE...SPLIT SUBPARTITION

Use the ALTER TABLE... SPLIT SUBPARTITION command to divide a single subpartition into two subpartitions, and redistribute the subpartition's contents. The command comes in two variations.

The first variation splits a range subpartition into two subpartitions:

```
ALTER TABLE <table_name> SPLIT SUBPARTITION <subpartition_name>
  AT (range_part_value)
  INTO
  (
    SUBPARTITION <new_subpart1>
      [TABLESPACE <tablespace_name>],
    SUBPARTITION <new_subpart2>
      [TABLESPACE <tablespace_name>]
  );
```

The second variation splits a list subpartition into two subpartitions:

```
ALTER TABLE <table_name> SPLIT SUBPARTITION <subpartition_name>
  VALUES (<value>[, <value>]...)
  INTO
  (
    SUBPARTITION <new_subpart1>
      [TABLESPACE <tablespace_name>],
    SUBPARTITION <new_subpart2>
      [TABLESPACE <tablespace_name>]
  );
```

Description

The ALTER TABLE...SPLIT SUBPARTITION command adds a subpartition to an existing subpartitioned table. There is no upper limit to the number of defined subpartitions. When you execute an ALTER TABLE...SPLIT SUBPARTITION command, Advanced Server creates two new subpartitions, moving any rows that contain values that are constrained by the specified subpartition rules into new_subpart1, and any remaining rows into new_subpart2.

The new subpartition rules must reference the column specified in the rules that define the existing subpartition(s).

Include the TABLESPACE clause to specify a tablespace in which a new subpartition will reside. If you do not specify a tablespace, the subpartition will be created in the default tablespace.

If the table is indexed, the index will be created on the new subpartition.

To use the ALTER TABLE... SPLIT SUBPARTITION command you must be the table owner, or have superuser (or administrative) privileges.

Parameters

table_name

The name (optionally schema-qualified) of the partitioned table.

subpartition_name

The name of the subpartition that is being split.

`new_subpart1`

The name of the first new subpartition to be created. Subpartition names must be unique amongst all partitions and subpartitions, and must follow the naming conventions for object identifiers.

`new_subpart1` will receive the rows that meet the subpartitioning constraints specified in the `ALTER TABLE... SPLIT SUBPARTITION` command.

`new_subpart2`

The name of the second new subpartition to be created. Subpartition names must be unique amongst all partitions and subpartitions, and must follow the naming conventions for object identifiers.

`new_subpart2` will receive the rows are not directed to `new_subpart1` by the subpartitioning constraints specified in the `ALTER TABLE... SPLIT SUBPARTITION` command.

`(value[, value]...)`

Use `value` to specify a quoted literal value (or comma-delimited list of literal values) by which table entries will be grouped into partitions. Each partitioning rule must specify at least one value, but there is no limit placed on the number of values specified within a rule. `value` may also be `NULL, DEFAULT` (if specifying a `LIST` subpartition), or `MAXVALUE` (if specifying a `RANGE` subpartition).

For information about creating a `DEFAULT` or `MAXVALUE` partition, see *Handling Stray Values in a LIST or RANGE Partitioned Table*.

`tablespace_name`

The name of the tablespace in which the partition or subpartition resides.

4.5.1 Example - Splitting a LIST Subpartition

The following example splits a list subpartition, redistributing the subpartition's contents between two new subpartitions. The sample table `(sales)` was created with the command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date)
  SUBPARTITION BY LIST (country)
  (
    PARTITION first_half_2012 VALUES LESS THAN('01-JUL-2012')
```

(continues on next page)

(continued from previous page)

```
    (
      SUBPARTITION p1_europe VALUES ('ITALY', 'FRANCE'),
      SUBPARTITION p1_americas VALUES ('US', 'CANADA')
    ),
    PARTITION second_half_2012 VALUES LESS THAN('01-JAN-2013')
    (
      SUBPARTITION p2_europe VALUES ('ITALY', 'FRANCE'),
      SUBPARTITION p2_americas VALUES ('US', 'CANADA')
    )
  );
```

The `sales` table has two partitions, named `first_half_2012`, and `second_half_2012`. Each partition has two range-defined subpartitions that distribute the partition's contents into subpartitions based on the value of the `country` column.

```
edb=# SELECT partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS;
  partition_name  | subpartition_name |    high_value
------------------+-------------------+-------------------
 SECOND_HALF_2012 | P2_AMERICAS       | 'US', 'CANADA'
 SECOND_HALF_2012 | P2_EUROPE         | 'ITALY', 'FRANCE'
 FIRST_HALF_2012  | P1_AMERICAS       | 'US', 'CANADA'
 FIRST_HALF_2012  | P1_EUROPE         | 'ITALY', 'FRANCE'
(4 rows)
```

The following command adds rows to each subpartition.

```
INSERT INTO sales VALUES
  (10, '4519b', 'FRANCE', '17-Jan-2012', '45000'),
  (40, '9519b', 'US', '12-Apr-2012', '145000'),
  (40, '4577b', 'US', '11-Nov-2012', '25000'),
  (30, '7588b', 'CANADA', '14-Dec-2012', '50000'),
  (30, '9519b', 'CANADA', '01-Feb-2012', '75000'),
  (30, '4519b', 'CANADA', '08-Apr-2012', '120000'),
  (40, '3788a', 'US', '12-May-2012', '4950'),
  (10, '9519b', 'ITALY', '07-Jul-2012', '15000'),
  (10, '9519a', 'FRANCE', '18-Aug-2012', '650000'),
  (10, '9519b', 'FRANCE', '18-Aug-2012', '650000'),
  (40, '4788a', 'US', '23-Sept-2012', '4950'),
  (40, '4788b', 'US', '09-Oct-2012', '15000');
```

A `SELECT` statement confirms that the rows are correctly distributed amongst the subpartitions.

```
edb=# SELECT tableoid::regclass, * FROM sales;
   tableoid      | dept_no | part_no | country |        date         | amount
-----------------+---------+---------+---------+---------------------+--------
sales_p1_americas|      40 | 9519b   | US      | 12-APR-12 00:00:00 | 145000
sales_p1_americas|      30 | 9519b   | CANADA  | 01-FEB-12 00:00:00 |  75000
sales_p1_americas|      30 | 4519b   | CANADA  | 08-APR-12 00:00:00 | 120000
sales_p1_americas|      40 | 3788a   | US      | 12-MAY-12 00:00:00 |   4950
sales_p1_europe  |      10 | 4519b   | FRANCE  | 17-JAN-12 00:00:00 |  45000
```

(continues on next page)

(continued from previous page)

```
sales_p2_americas|      40 | 4577b   | US      | 11-NOV-12 00:00:00 |  25000
sales_p2_americas|      30 | 7588b   | CANADA  | 14-DEC-12 00:00:00 |  50000
sales_p2_americas|      40 | 4788a   | US      | 23-SEP-12 00:00:00 |   4950
sales_p2_americas|      40 | 4788b   | US      | 09-OCT-12 00:00:00 |  15000
sales_p2_europe  |      10 | 9519b   | ITALY   | 07-JUL-12 00:00:00 |  15000
sales_p2_europe  |      10 | 9519a   | FRANCE  | 18-AUG-12 00:00:00 | 650000
sales_p2_europe  |      10 | 9519b   | FRANCE  | 18-AUG-12 00:00:00 | 650000
(12 rows)
```

The following command splits the `p2_americas` subpartition into two new subpartitions, and redistributes the contents.

```
ALTER TABLE sales SPLIT SUBPARTITION p2_americas
  VALUES ('US')
  INTO
  (
    SUBPARTITION p2_us,
    SUBPARTITION p2_canada
  );
```

After invoking the command, the `p2_americas` subpartition has been deleted; in its place, the server has created two new subpartitions (`p2_us` and `p2_canada`).

```
edb=# SELECT partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS;
  partition_name  | subpartition_name |    high_value
------------------+-------------------+-------------------
 FIRST_HALF_2012  | P1_EUROPE         | 'ITALY', 'FRANCE'
 FIRST_HALF_2012  | P1_AMERICAS       | 'US', 'CANADA'
 SECOND_HALF_2012 | P2_EUROPE         | 'ITALY', 'FRANCE'
 SECOND_HALF_2012 | P2_US             | 'US'
 SECOND_HALF_2012 | P2_CANADA         | 'CANADA'
(5 rows)
```

Querying the `sales` table demonstrates that the content of the `p2_americas` subpartition has been redistributed.

```
edb=# SELECT tableoid::regclass, * FROM sales;
   tableoid       | dept_no | part_no | country |        date         |amount
------------------+---------+---------+---------+---------------------+------
 sales_p1_americas|      40 | 9519b   | US      | 12-APR-12 00:00:00 |145000
 sales_p1_americas|      30 | 9519b   | CANADA  | 01-FEB-12 00:00:00 | 75000
 sales_p1_americas|      30 | 4519b   | CANADA  | 08-APR-12 00:00:00 |120000
 sales_p1_americas|      40 | 3788a   | US      | 12-MAY-12 00:00:00 |  4950
 sales_p1_europe  |      10 | 4519b   | FRANCE  | 17-JAN-12 00:00:00 | 45000
 sales_p2_canada  |      30 | 7588b   | CANADA  | 14-DEC-12 00:00:00 | 50000
 sales_p2_europ   |      10 | 9519b   | ITALY   | 07-JUL-12 00:00:00 | 15000
 sales_p2_europe  |      10 | 9519a   | FRANCE  | 18-AUG-12 00:00:00 |650000
 sales_p2_europe  |      10 | 9519b   | FRANCE  | 18-AUG-12 00:00:00 |650000
 sales_p2_us      |      40 | 4577b   | US      | 11-NOV-12 00:00:00 | 25000
 sales_p2_us      |      40 | 4788a   | US      | 23-SEP-12 00:00:00 |  4950
```

(continues on next page)

(continued from previous page)

```
 sales_p2_us      |      40 | 4788b   | US      | 09-OCT-12 00:00:00 | 15000
(12 rows)
```

4.5.2 Example - Splitting a RANGE Subpartition

The following example splits a range subpartition, redistributing the subpartition's contents between two new subpartitions. The sample table (sales) was created with the command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country)
  SUBPARTITION BY RANGE(date)
(
  PARTITION europe VALUES('FRANCE', 'ITALY')
    (
      SUBPARTITION europe_2011
        VALUES LESS THAN('2012-Jan-01'),
      SUBPARTITION europe_2012
        VALUES LESS THAN('2013-Jan-01')
    ),
  PARTITION asia VALUES('INDIA', 'PAKISTAN')
    (
      SUBPARTITION asia_2011
        VALUES LESS THAN('2012-Jan-01'),
      SUBPARTITION asia_2012
        VALUES LESS THAN('2013-Jan-01')
    ),
  PARTITION americas VALUES('US', 'CANADA')
    (
      SUBPARTITION americas_2011
        VALUES LESS THAN('2012-Jan-01'),
      SUBPARTITION americas_2012
        VALUES LESS THAN('2013-Jan-01')
    )
);
```

The sales table has three partitions (europe, asia, and americas). Each partition has two range-defined subpartitions that sort the partitions contents into subpartitions by the value of the date column.

```
edb=# SELECT partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS;
 partition_name | subpartition_name |      high_value
----------------+-------------------+----------------------
 EUROPE         | EUROPE_2011       | '01-JAN-12 00:00:00'
```

(continues on next page)

(continued from previous page)

```
 EUROPE          | EUROPE_2012       | '01-JAN-13 00:00:00'
 ASIA            | ASIA_2011         | '01-JAN-12 00:00:00'
 ASIA            | ASIA_2012         | '01-JAN-13 00:00:00'
 AMERICAS        | AMERICAS_2011     | '01-JAN-12 00:00:00'
 AMERICAS        | AMERICAS_2012     | '01-JAN-13 00:00:00'
(6 rows)
```

The following command adds rows to each subpartition.

```
INSERT INTO sales VALUES
  (10, '4519b', 'FRANCE', '17-Jan-2012', '45000'),
  (20, '3788a', 'INDIA', '01-Mar-2012', '75000'),
  (40, '9519b', 'US', '12-Apr-2012', '145000'),
  (20, '3788a', 'PAKISTAN', '04-Jun-2012', '37500'),
  (40, '4577b', 'US', '11-Nov-2012', '25000'),
  (30, '7588b', 'CANADA', '14-Dec-2012', '50000'),
  (30, '9519b', 'CANADA', '01-Feb-2012', '75000'),
  (30, '4519b', 'CANADA', '08-Apr-2012', '120000'),
  (40, '3788a', 'US', '12-May-2012', '4950'),
  (10, '9519b', 'ITALY', '07-Jul-2012', '15000'),
  (10, '9519a', 'FRANCE', '18-Aug-2012', '650000'),
  (10, '9519b', 'FRANCE', '18-Aug-2012', '650000'),
  (20, '3788b', 'INDIA', '21-Sept-2012', '5090'),
  (40, '4788a', 'US', '23-Sept-2012', '4950'),
  (40, '4788b', 'US', '09-Oct-2012', '15000'),
  (20, '4519a', 'INDIA', '18-Oct-2012', '650000'),
  (20, '4519b', 'INDIA', '2-Dec-2012', '5090');
```

A `SELECT` statement confirms that the rows are distributed amongst the subpartitions.

```
edb=# SELECT tableoid::regclass, * FROM sales;
 tableoid           |dept_no|part_no| country |       date          | amount
--------------------+-------+-------+---------+---------------------+---------
 sales_americas_2012|  40   | 9519b | US      | 12-APR-12 00:00:00 | 145000
 sales_americas_2012|  40   | 4577b | US      | 11-NOV-12 00:00:00 |  25000
 sales_americas_2012|  30   | 7588b | CANADA  | 14-DEC-12 00:00:00 |  50000
 sales_americas_2012|  30   | 9519b | CANADA  | 01-FEB-12 00:00:00 |  75000
 sales_americas_2012|  30   | 4519b | CANADA  | 08-APR-12 00:00:00 | 120000
 sales_americas_2012|  40   | 3788a | US      | 12-MAY-12 00:00:00 |   4950
 sales_americas_2012|  40   | 4788a | US      | 23-SEP-12 00:00:00 |   4950
 sales_americas_2012|  40   | 4788b | US      | 09-OCT-12 00:00:00 |  15000
 sales_europe_2012  |  10   | 4519b | FRANCE  | 17-JAN-12 00:00:00 |  45000
 sales_europe_2012  |  10   | 9519b | ITALY   | 07-JUL-12 00:00:00 |  15000
 sales_europe_2012  |  10   | 9519a | FRANCE  | 18-AUG-12 00:00:00 | 650000
 sales_europe_2012  |  10   | 9519b | FRANCE  | 18-AUG-12 00:00:00 | 650000
 sales_asia_2012    |  20   | 3788a | INDIA   | 01-MAR-12 00:00:00 |  75000
 sales_asia_2012    |  20   | 3788a | PAKISTAN| 04-JUN-12 00:00:00 |  37500
 sales_asia_2012    |  20   | 3788b | INDIA   | 21-SEP-12 00:00:00 |   5090
 sales_asia_2012    |  20   | 4519a | INDIA   | 18-OCT-12 00:00:00 | 650000
 sales_asia_2012    |  20   | 4519b | INDIA   | 02-DEC-12 00:00:00 |   5090
(17 rows)
```

The following command splits the americas_2012 subpartition into two new subpartitions, and redistributes the contents.

```
ALTER TABLE sales
  SPLIT SUBPARTITION americas_2012
  AT('2012-Jun-01')
  INTO
   (
     SUBPARTITION americas_p1_2012,
     SUBPARTITION americas_p2_2012
   );
```

After invoking the command, the americas_2012 subpartition has been deleted; in its place, the server has created two new subpartitions (americas_p1_2012 and americas_p2_2012).

```
edb=# SELECT partition_name, subpartition_name, high_value FROM
ALL_TAB_SUBPARTITIONS;
 partition_name | subpartition_name |      high_value
----------------+-------------------+----------------------
 EUROPE         | EUROPE_2011       | '01-JAN-12 00:00:00'
 EUROPE         | EUROPE_2012       | '01-JAN-13 00:00:00'
 ASIA           | ASIA_2011         | '01-JAN-12 00:00:00'
 ASIA           | ASIA_2012         | '01-JAN-13 00:00:00'
 AMERICAS       | AMERICAS_2011     | '01-JAN-12 00:00:00'
 AMERICAS       | AMERICAS_P1_2012  | '01-JUN-12 00:00:00'
 AMERICAS       | AMERICAS_P2_2012  | '01-JAN-13 00:00:00'
(7 rows)
```

Querying the sales table demonstrates that the subpartition's contents are redistributed.

```
edb=# SELECT tableoid::regclass, * FROM sales;
 tableoid              |dept_no|part_no|country  |      date          | amount
-----------------------+-------+-------+---------+--------------------+-------
 sales_americas_p1_2012|  40   | 9519b | US      | 12-APR-12 00:00:00|145000
 sales_americas_p1_2012|  30   | 9519b | CANADA  | 01-FEB-12 00:00:00|75000
 sales_americas_p1_2012|  30   | 4519b | CANADA  | 08-APR-12 00:00:00|120000
 sales_americas_p1_2012|  40   | 3788a | US      | 12-MAY-12 00:00:00| 4950
 sales_americas_p2_2012|  40   | 4577b | US      | 11-NOV-12 00:00:00|25000
 sales_americas_p2_2012|  30   | 7588b | CANADA  | 14-DEC-12 00:00:00|50000
 sales_americas_p2_2012|  40   | 4788a | US      | 23-SEP-12 00:00:00| 4950
 sales_americas_p2_2012|  40   | 4788b | US      | 09-OCT-12 00:00:00|15000
 sales_europe_2012     |  10   | 4519b | FRANCE  | 17-JAN-12 00:00:00|45000
 sales_europe_2012     |  10   | 9519b | ITALY   | 07-JUL-12 00:00:00|15000
 sales_europe_2012     |  10   | 9519a | FRANCE  | 18-AUG-12 00:00:00|650000
 sales_europe_2012     |  10   | 9519b | FRANCE  | 18-AUG-12 00:00:00|650000
 sales_asia_2012       |  20   | 3788a | INDIA   | 01-MAR-12 00:00:00|75000
 sales_asia_2012       |  20   | 3788a | PAKISTAN|04-JUN-12 00:00:00 |37500
 sales_asia_2012       |  20   | 3788b | INDIA   | 21-SEP-12 00:00:00|5090
 sales_asia_2012       |  20   | 4519a | INDIA   | 18-OCT-12 00:00:00|650000
 sales_asia_2012       |  20   | 4519b | INDIA   | 02-DEC-12 00:00:00| 5090
(17 rows)
```

4.6 ALTER TABLE...EXCHANGE PARTITION

The ALTER TABLE...EXCHANGE PARTITION command swaps an existing table with a partition. If you plan to add a large quantity of data to a partitioned table, you can use the ALTER TABLE... EXCHANGE PARTITION command to implement a bulk load. You can also use the ALTER TABLE... EXCHANGE PARTITION command to remove old or unneeded data for storage.

The command syntax is available in two forms.

The first form swaps a table for a partition:

```
ALTER TABLE <target_table>
  EXCHANGE PARTITION <target_partition>
  WITH TABLE <source_table>
  [(INCLUDING | EXCLUDING) INDEXES]
  [(WITH | WITHOUT) VALIDATION];
```

The second form swaps a table for a subpartition:

```
ALTER TABLE <target_table>
  EXCHANGE SUBPARTITION <target_subpartition>
  WITH TABLE <source_table>
  [(INCLUDING | EXCLUDING) INDEXES]
  [(WITH | WITHOUT) VALIDATION];
```

Description

When the ALTER TABLE... EXCHANGE PARTITION command completes, the data originally located in the target_partition will be located in the source_table, and the data originally located in the source_table will be located in the target_partition.

The ALTER TABLE... EXCHANGE PARTITION command can exchange partitions in a LIST, RANGE or HASH partitioned table. The structure of the source_table must match the structure of the target_table (both tables must have matching columns and data types), and the data contained within the table must adhere to the partitioning constraints.

If the INCLUDING INDEXES clause is specified with EXCHANGE PARTITION, then matching indexes in the target_partition and source_table are swapped. Indexes in the target_partition with no match in the source_table are rebuilt and vice versa (that is, indexes in the source_table with no match in the target_partition are also rebuilt).

If the EXCLUDING INDEXES clause is specified with EXCHANGE PARTITION, then matching indexes in the target_partition and source_table are swapped, but the target_partition indexes with no match in the source_table are marked as invalid and vice versa (that is, indexes in the source_table with no match in the target_partition are also marked as invalid).

The previously used *matching index* term refers to indexes that have the same attributes such as the collation order, ascending or descending direction, ordering of nulls first or nulls last, and so forth as determined by the CREATE INDEX command.

If both INCLUDING INDEXES and EXCLUDING INDEXES are omitted, then the default action is the EXCLUDING INDEXES behavior.

The same behavior as previously described applies for the target_subpartition used with the EXCHANGE SUBPARTITION clause.

You must own a table to invoke ALTER TABLE... EXCHANGE PARTITION or ALTER TABLE... EXCHANGE SUBPARTITION against that table.

Parameters:

target_table

The name (optionally schema-qualified) of the table in which the partition or subpartition resides.

target_partition

The name of the partition to be replaced.

target_subpartition

The name of the subpartition to be replaced.

source_table

The name of the table that will replace the target_partition or target_subpartition.

4.6.1 Example - Exchanging a Table for a Partition

The example that follows demonstrates swapping a table for a partition (americas) of the sales table. You can create the sales table with the following command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date    date,
  amount  number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

Use the following command to add sample data to the sales table.

```
INSERT INTO sales VALUES
  (40, '9519b', 'US', '12-Apr-2012', '145000'),
  (10, '4519b', 'FRANCE', '17-Jan-2012', '45000'),
  (20, '3788a', 'INDIA', '01-Mar-2012', '75000'),
  (20, '3788a', 'PAKISTAN', '04-Jun-2012', '37500'),
  (10, '9519b', 'ITALY', '07-Jul-2012', '15000'),
```

(continues on next page)

(continued from previous page)

```
  (10, '9519a', 'FRANCE', '18-Aug-2012', '650000'),
  (10, '9519b', 'FRANCE', '18-Aug-2012', '650000'),
  (20, '3788b', 'INDIA', '21-Sept-2012', '5090'),
  (20, '4519a', 'INDIA', '18-Oct-2012', '650000'),
  (20, '4519b', 'INDIA', '2-Dec-2012', '5090');
```

Querying the `sales` table shows that only one row resides in the `americas` partition.

```
edb=# SELECT tableoid::regclass, * FROM sales;
    tableoid    | dept_no | part_no | country  |        date         | amount
----------------+---------+---------+----------+---------------------+--------
 sales_americas |      40 | 9519b   | US       | 12-APR-12 00:00:00 | 145000
 sales_europe   |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_europe   |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_europe   |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_europe   |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_asia     |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_asia     |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_asia     |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_asia     |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_asia     |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
(10 rows)
```

The following command creates a table `(n_america)` that matches the definition of the `sales` table.

```
CREATE TABLE n_america
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date    date,
  amount  number
);
```

The following command adds data to the `n_america` table. The data conforms to the partitioning rules of the `americas` partition.

```
INSERT INTO n_america VALUES
  (40, '9519b', 'US', '12-Apr-2012', '145000'),
  (40, '4577b', 'US', '11-Nov-2012', '25000'),
  (30, '7588b', 'CANADA', '14-Dec-2012', '50000'),
  (30, '9519b', 'CANADA', '01-Feb-2012', '75000'),
  (30, '4519b', 'CANADA', '08-Apr-2012', '120000'),
  (40, '3788a', 'US', '12-May-2012', '4950'),
  (40, '4788a', 'US', '23-Sept-2012', '4950'),
  (40, '4788b', 'US', '09-Oct-2012', '15000');
```

The following command swaps the table into the partitioned table.

```
ALTER TABLE sales
  EXCHANGE PARTITION americas
```

(continues on next page)

(continued from previous page)

```
  WITH TABLE n_america;
```

Querying the sales table shows that the contents of the n_america table has been exchanged for the content of the americas partition.

```
edb=# SELECT tableoid::regclass, * FROM sales;
    tableoid    | dept_no | part_no | country  |        date         | amount
----------------+---------+---------+----------+---------------------+--------
 sales_americas |      40 | 9519b   | US       | 12-APR-12 00:00:00 | 145000
 sales_americas |      40 | 4577b   | US       | 11-NOV-12 00:00:00 |  25000
 sales_americas |      30 | 7588b   | CANADA   | 14-DEC-12 00:00:00 |  50000
 sales_americas |      30 | 9519b   | CANADA   | 01-FEB-12 00:00:00 |  75000
 sales_americas |      30 | 4519b   | CANADA   | 08-APR-12 00:00:00 | 120000
 sales_americas |      40 | 3788a   | US       | 12-MAY-12 00:00:00 |   4950
 sales_americas |      40 | 4788a   | US       | 23-SEP-12 00:00:00 |   4950
 sales_americas |      40 | 4788b   | US       | 09-OCT-12 00:00:00 |  15000
 sales_europe   |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_europe   |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_europe   |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_europe   |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_asia     |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_asia     |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_asia     |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_asia     |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_asia     |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
(17 rows)
```

Querying the n_america table shows that the row that was previously stored in the americas partition has been moved to the n_america table.

```
edb=# SELECT tableoid::regclass, * FROM n_america;
 tableoid  | dept_no | part_no | country |        date         | amount
-----------+---------+---------+---------+---------------------+--------
 n_america |      40 | 9519b   | US      | 12-APR-12 00:00:00 | 145000
(1 row)
```

4.7 ALTER TABLE...MOVE PARTITION

Use the ALTER TABLE... MOVE PARTITION command to move a partition to a different tablespace. The command takes two forms.

The first form moves a partition to a new tablespace:

```
ALTER TABLE <table_name>
  MOVE PARTITION <partition_name>
   TABLESPACE <tablespace_name>;
```

The second form moves a subpartition to a new tablespace:

```
ALTER TABLE <table_name>
  MOVE SUBPARTITION <subpartition_name>
   TABLESPACE <tablespace_name>;
```

Description

The ALTER TABLE...MOVE PARTITION command moves a partition from its current tablespace to a different tablespace. The ALTER TABLE... MOVE PARTITION command can move partitions of a LIST, RANGE or HASH partitioned table.

The same behavior as previously described applies for the subpartition_name used with the MOVE SUBPARTITION clause.

You must own the table to invoke ALTER TABLE... MOVE PARTITION or ALTER TABLE... MOVE SUBPARTITION.

Parameters

table_name

The name (optionally schema-qualified) of the table in which the partition or subpartition resides.

partition_name

The name of the partition to be moved.

subpartition_name

The name of the subpartition to be moved.

tablespace_name

The name of the tablespace to which the partition or subpartition will be moved.

4.7.1 Example - Moving a Partition to a Different Tablespace

The following example moves a partition of the `sales` table from one tablespace to another. First, create the `sales` table with the command:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date)
(
  PARTITION q1_2012 VALUES LESS THAN ('2012-Apr-01'),
  PARTITION q2_2012 VALUES LESS THAN ('2012-Jul-01'),
  PARTITION q3_2012 VALUES LESS THAN ('2012-Oct-01'),
  PARTITION q4_2012 VALUES LESS THAN ('2013-Jan-01') TABLESPACE ts_1,
  PARTITION q1_2013 VALUES LESS THAN ('2013-Mar-01') TABLESPACE ts_2
);
```

Querying the `ALL_TAB_PARTITIONS` view confirms that the partitions reside on the expected servers and tablespaces.

```
edb=# SELECT partition_name, tablespace_name FROM ALL_TAB_PARTITIONS;
 partition_name | tablespace_name
----------------+-----------------
 Q1_2012        |
 Q2_2012        |
 Q3_2012        |
 Q4_2012        | TS_1
 Q1_2013        | TS_2
(5 rows)
```

After preparing the target tablespace, invoke the `ALTER TABLE... MOVE PARTITION` command to move the `q1_2013` partition from a tablespace named `ts_2` to a tablespace named `ts_3`.

```
ALTER TABLE sales MOVE PARTITION q1_2013 TABLESPACE ts_3;
```

Querying the `ALL_TAB_PARTITIONS` view shows that the move was successful.

```
edb=# SELECT partition_name, tablespace_name FROM ALL_TAB_PARTITIONS;
 partition_name | tablespace_name
----------------+-----------------
 Q1_2012        |
 Q2_2012        |
 Q3_2012        |
 Q4_2012        | TS_1
 Q1_2013        | TS_3
(5 rows)
```

4.8 ALTER TABLE... RENAME PARTITION

Use the ALTER TABLE... RENAME PARTITION command to rename a table partition. The syntax takes two forms.

The first form renames a partition:

```
ALTER TABLE <table_name>
 RENAME PARTITION <partition_name>
 TO <new_name>;
```

The second form renames a subpartition:

```
ALTER TABLE <table_name>
 RENAME SUBPARTITION <subpartition_name>
 TO <new_name>;
```

Description

The ALTER TABLE... RENAME PARTITION command renames a partition.

The same behavior as previously described applies for the subpartition_name used with the RENAME SUBPARTITION clause.

You must own the specified table to invoke ALTER TABLE... RENAME PARTITION or ALTER TABLE... RENAME SUBPARTITION.

Parameters

table_name

The name (optionally schema-qualified) of the table in which the partition or subpartition resides.

partition_name

The name of the partition to be renamed.

subpartition_name

The name of the subpartition to be renamed.

new_name

The new name of the partition or subpartition.

4.8.1 Example - Renaming a Partition

The following command creates a list-partitioned table named sales:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date    date,
  amount  number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

Query the ALL_TAB_PARTITIONS view to display the partition names.

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |     high_value
----------------+---------------------
 EUROPE         | 'FRANCE', 'ITALY'
 ASIA           | 'INDIA', 'PAKISTAN'
 AMERICAS       | 'US', 'CANADA'
(3 rows)
```

The following command renames the americas partition to n_america.

```
ALTER TABLE sales
 RENAME PARTITION americas TO n_america;
```

Querying the ALL_TAB_PARTITIONS view demonstrates that the partition has been successfully renamed.

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |     high_value
----------------+---------------------
 EUROPE         | 'FRANCE', 'ITALY'
 ASIA           | 'INDIA', 'PAKISTAN'
 N_AMERICA      | 'US', 'CANADA'
(3 rows)
```

4.9 ALTER TABLE...SET INTERVAL

Use the ALTER TABLE... SET INTERVAL command to convert an existing range-partitioned table to an interval range partitioned table. The database automatically creates a new partition of a specified range or interval for the partitioned table when INTERVAL is set. The syntax is:

```
ALTER TABLE <table_name> SET INTERVAL (<constant> | <expression>);
```

To disable an interval range partitioned table and convert it to a range-partitioned table, the syntax is:

```
ALTER TABLE <table_name> SET INTERVAL ();
```

Parameters

table_name

The name (optionally schema-qualified) of the range-partitioned table.

constant | expression

Specifies a NUMERIC, DATE, or TIME value.

Description

The ALTER TABLE... SET INTERVAL command can be used to convert the range-partitioned table to use interval range partitioning. A new partition of a specified interval is created and data can be inserted into the new partition.

The SET INTERVAL () command can be used to disable interval range partitioning. The database converts an interval range partitioned table to range-partitioned and sets the boundaries of the interval range partitions to the boundaries for the range partitions.

4.9.1 Example - Setting an Interval Range Partition

The example that follows sets an interval range partition of the sales table from range partitioning to start using monthly interval range partitioning. Use the following command to create the sales table:

```
CREATE TABLE sales
(
  prod_id           int,
  prod_quantity     int,
  sold_month        date
)
PARTITION BY RANGE(sold_month)
(
  PARTITION p1
    VALUES LESS THAN('15-JAN-2019'),
  PARTITION p2
    VALUES LESS THAN('15-FEB-2019')
);
```

To set the interval range partitioning from the sales table, invoke the following command:

```
ALTER TABLE sales SET INTERVAL (NUMTOYMINTERVAL(1, 'MONTH'));
```

Query the ALL_TAB_PARTITIONS view before a database creates an interval range partition.

```
edb=# SELECT partition_name, high_value from ALL_TAB_PARTITIONS;
 partition_name |      high_value
----------------+----------------------
 P1             | '15-JAN-19 00:00:00'
 P2             | '15-FEB-19 00:00:00'
(2 rows)
```

Now, add data to the sales table that exceeds the high value of a range partition.

```
edb=# INSERT INTO sales VALUES (1,100,'05-APR-2019');
INSERT 0 1
```

Then, query the ALL_TAB_PARTITIONS view again after the INSERT statement. The interval range partition is successfully created and data is inserted. A system-generated name of the interval range partition is created that varies for each session.

```
edb=# SELECT partition_name, high_value from ALL_TAB_PARTITIONS;
 partition_name |      high_value
----------------+----------------------
 P1             | '15-JAN-19 00:00:00'
 P2             | '15-FEB-19 00:00:00'
 SYS916340103   | '15-APR-19 00:00:00'
(3 rows)
```

4.10 ALTER TABLE... SET [PARTITIONING] AUTOMATIC

Use the ALTER TABLE... SET [PARTITIONING] AUTOMATIC command to convert an existing list partitioned table to automatic list partitioning. The database automatically creates a partition based on a new value inserted into a table when AUTOMATIC is set. The syntax is:

```
ALTER TABLE <table_name> SET [ PARTITIONING ] AUTOMATIC;
```

To disable AUTOMATIC LIST PARTITIONING and convert to regular LIST partition, the syntax is:

```
ALTER TABLE <table_name> SET [ PARTITIONING ] MANUAL;
```

Parameters

table_name

The name (optionally schema-qualified) of the list-partitioned table.

Description

The ALTER TABLE... SET [PARTITIONING] AUTOMATIC command can be used to convert the regular list partitioned table to use automatic partitioning. A partition is created and data can be inserted into the new partition.

The ALTER TABLE... SET [PARTITIONING] MANUAL command can be used to disable the automatic list partitioning; the database converts an automatic list partitioned table to a regular list partitioned table.

4.10.1 Example - Setting an AUTOMATIC List Partition

The example that follows modifies a table sales to use automatic list partition instead of regular list partitioning. Use the following command to create a sales table:

```
CREATE TABLE sales
(
  dept_no      number,
  part_no      varchar2,
  sales_state  varchar2(20),
  date         date,
  amount       number
)
PARTITION BY LIST(sales_state)
(
  PARTITION P_KAN VALUES('KANSAS'),
  PARTITION P_TEX VALUES('TEXAS')
);
```

To implement automatic list partitioning on the sales table, invoke the following command:

```
ALTER TABLE sales SET AUTOMATIC;
```

Query the ALL_TAB_PARTITIONS view to see that an existing partition is successfully created.

```
edb=# select table_name, partition_name, high_value from all_tab_partitions;
 table_name | partition_name | high_value
------------+----------------+------------
 SALES      | P_KAN          | 'KANSAS'
 SALES      | P_TEX          | 'TEXAS'
(2 rows)
```

Now, insert data into the `sales` table to create a new partition and add the new value.

```
edb=# INSERT INTO sales VALUES (1, 'VIR', 'VIRGINIA');
INSERT 0 1
```

Then, query the `ALL_TAB_PARTITIONS` view again after the insert. The automatic list partition is successfully created and data is inserted. A system-generated name of the partition is created that varies for each session.

```
edb=# select table_name, partition_name, high_value from all_tab_partitions;
 table_name | partition_name | high_value
------------+----------------+------------
 SALES      | P_KAN          | 'KANSAS'
 SALES      | P_TEX          | 'TEXAS'
 SALES      | SYS106900103   | 'VIRGINIA'
(3 rows)
```

4.11 ALTER TABLE...SET SUBPARTITION TEMPLATE

Use the ALTER TABLE... SET SUBPARTITION TEMPLATE command to update the subpartition template for a table. The syntax is:

```
ALTER TABLE <table_name> SET SUBPARTITION TEMPLATE <num>;
```

To reset the subpartitions number to default using the subpartition template, the syntax is:

```
ALTER TABLE <table_name> SET SUBPARTITION TEMPLATE ();
```

Parameters

table_name

The name (optionally schema-qualified) of the partitioned table.

num

The number of subpartitions to be added for a partition.

Description

The ALTER TABLE... SET SUBPARTITION TEMPLATE command can be used to update the subpartition template in the table. If you are specifying a subpartition descriptor for a partition then a subpartition descriptor is used instead of a subpartition template. The subpartition template can be used whenever a subpartition descriptor is not specified for a partition. If either subpartition descriptor or subpartition template is not specified, then by default a single subpartition is created.

Note: The partitions added to a table after invoking ALTER TABLE... SET SUBPARTITION TEMPLATE command will use the new SUBPARTITION TEMPLATE.

The ALTER TABLE... SET SUBPARTITION TEMPLATE () command can be used to reset the subpartitions number to default 1.

4.11.1 Example - Setting a SUBPARTITION TEMPLATE

The following example creates a table sales that is range partitioned by date and hash subpartitioned by country. Use the following command to create the sales table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE (date) SUBPARTITION BY HASH (country) SUBPARTITIONS 2
(
```

(continues on next page)

(continued from previous page)

```
  PARTITION q1_2012
    VALUES LESS THAN('2012-Apr-01'),
  PARTITION q2_2012
    VALUES LESS THAN('2012-Jul-01'),
  PARTITION q3_2012
    VALUES LESS THAN('2012-Oct-01'),
  PARTITION q4_2012
    VALUES LESS THAN('2013-Jan-01')
);
```

The table definition creates four partitions (q1_2012, q2_2012, q3_2012, and q4_2012), each partition consisting of two subpartitions with system-generated names.

```
 edb=# SELECT table_name, partition_name, subpartition_name FROM
 ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
  table_name | partition_name | subpartition_name
 ------------+----------------+-------------------
  SALES      | Q1_2012        | SYS0101
  SALES      | Q1_2012        | SYS0102
  SALES      | Q2_2012        | SYS0103
  SALES      | Q2_2012        | SYS0104
  SALES      | Q3_2012        | SYS0105
  SALES      | Q3_2012        | SYS0106
  SALES      | Q4_2012        | SYS0107
  SALES      | Q4_2012        | SYS0108
(8 rows)
```

To set the subpartition template on the sales table, invoke the following command:

```
ALTER TABLE sales SET SUBPARTITION TEMPLATE 8;
```

The sales table is modified with the subpartition template set to eight. Now, if you try to add a new partition q1_2013, a new partition will be created consisting of eight subpartitions as described in the subpartition template.

```
ALTER TABLE sales ADD PARTITION q1_2013 VALUES LESS THAN ('2013-Apr-01');
```

Query the ALL_TAB_PARTITIONS view, the q1_2013 partition is successfully added comprising of eight subpartitions with system-generated names assigned to them.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' and partition_name =
'Q1_2013' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | Q1_2013        | SYS0113
 SALES      | Q1_2013        | SYS0114
 SALES      | Q1_2013        | SYS0115
 SALES      | Q1_2013        | SYS0116
 SALES      | Q1_2013        | SYS0117
 SALES      | Q1_2013        | SYS0118
```

(continues on next page)

(continued from previous page)

```
 SALES      | Q1_2013        | SYS0119
 SALES      | Q1_2013        | SYS0120
(8 rows)
```

4.11.2 Example - Resetting a SUBPARTITION TEMPLATE

The following example creates a list-partitioned table `sales` that is list partitioned by `country` and has subpartitioned by `part_no`. Use the following command to create the `sales` table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST (country) SUBPARTITION BY HASH (part_no) SUBPARTITIONS 3
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

The table contains three partitions (`AMERICAS`, `ASIA`, and `EUROPE`), each partition consists of three subpartitions with system-generated names.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | AMERICAS       | SYS0109
 SALES      | AMERICAS       | SYS0107
 SALES      | AMERICAS       | SYS0108
 SALES      | ASIA           | SYS0105
 SALES      | ASIA           | SYS0104
 SALES      | ASIA           | SYS0106
 SALES      | EUROPE         | SYS0101
 SALES      | EUROPE         | SYS0103
 SALES      | EUROPE         | SYS0102
(9 rows)
```

The following command resets the subpartition template on the `sales` table.

```
ALTER TABLE sales SET SUBPARTITION TEMPLATE ();
```

The `sales` table is modified with the subpartition template reset to default `1`. Now, try to add a new partition `east_asia` using the following command:

```
ALTER TABLE sales ADD PARTITION east_asia VALUES ('CHINA', 'KOREA');
```

Query the ALL_TAB_PARTITIONS view, a new partition east_asia will be created consisting of one subpartition with a system-generated name assigned to them.

```
edb=# SELECT table_name, partition_name, subpartition_name FROM
ALL_TAB_SUBPARTITIONS WHERE table_name = 'SALES' and partition_name =
'EAST_ASIA' ORDER BY 1,2;
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 SALES      | EAST_ASIA      | SYS0113
(1 row)
```

4.12 DROP TABLE

Use the PostgreSQL `DROP TABLE` command to remove a partitioned table definition, it's partitions and subpartitions, and delete the table contents. The syntax is:

```
DROP TABLE <table_name>
```

Parameters

`table_name`

The name (optionally schema-qualified) of the partitioned table.

Description

The `DROP TABLE` command removes an entire table, and the data that resides in that table. When you delete a table, any partitions or subpartitions (of that table) are deleted as well.

To use the `DROP TABLE` command, you must be the owner of the partitioning root, a member of a group that owns the table, the schema owner, or a database superuser.

Example

To delete a table, connect to the controller node (the host of the partitioning root), and invoke the `DROP TABLE` command. For example, to delete the `sales` table, invoke the following command:

```
DROP TABLE sales;
```

The server will confirm that the table has been dropped.

```
edb=# drop table sales;
DROP TABLE
edb=#
```

For more information about the `DROP TABLE` command, see the PostgreSQL core documentation at:

https://www.postgresql.org/docs/current/static/sql-droptable.html

4.13 ALTER TABLE... DROP PARTITION

Use the ALTER TABLE... DROP PARTITION command to delete a partition definition, and the data stored in that partition. The syntax is:

```
ALTER TABLE <table_name> DROP PARTITION <partition_name>;
```

Parameters

table_name

The name (optionally schema-qualified) of the partitioned table.

partition_name

The name of the partition to be deleted.

Description

The ALTER TABLE... DROP PARTITION command deletes a partition and any data stored on that partition. The ALTER TABLE... DROP PARTITION command can drop partitions of a LIST or RANGE partitioned table; please note that this command does not work on a HASH partitioned table. When you delete a partition, any subpartitions (of that partition) are deleted as well.

To use the DROP PARTITION clause, you must be the owner of the partitioning root, a member of a group that owns the table, or have database superuser or administrative privileges.

4.13.1 Example - Deleting a Partition

The example that follows deletes a partition of the sales table. Use the following command to create the sales table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date    date,
  amount  number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

Querying the ALL_TAB_PARTITIONS view displays the partition names.

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |     high_value
----------------+---------------------
```

(continues on next page)

(continued from previous page)

```
 EUROPE          | 'FRANCE', 'ITALY'
 ASIA            | 'INDIA', 'PAKISTAN'
 AMERICAS        | 'US', 'CANADA'
(3 rows)
```

To delete the americas partition from the `sales` table, invoke the following command:

```
ALTER TABLE sales DROP PARTITION americas;
```

Querying the `ALL_TAB_PARTITIONS` view demonstrates that the partition has been successfully deleted.

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |     high_value
----------------+---------------------
 EUROPE          | 'FRANCE', 'ITALY'
 ASIA            | 'INDIA', 'PAKISTAN'
(2 rows)
```

4.14 ALTER TABLE... DROP SUBPARTITION

Use the ALTER TABLE... DROP SUBPARTITION command to drop a subpartition definition, and the data stored in that subpartition. The syntax is:

```
ALTER TABLE <table_name> DROP SUBPARTITION <subpartition_name>;
```

Parameters

table_name

The name (optionally schema-qualified) of the partitioned table.

subpartition_name

The name of the subpartition to be deleted.

Description

The ALTER TABLE... DROP SUBPARTITION command deletes a subpartition, and the data stored in that subpartition. To use the DROP SUBPARTITION clause, you must be the owner of the partitioning root, a member of a group that owns the table, or have superuser or administrative privileges.

4.14.1 Example - Deleting a Subpartition

The example that follows deletes a subpartition of the sales table. Use the following command to create the sales table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date    date,
  amount  number
)
PARTITION BY RANGE(date)
  SUBPARTITION BY LIST (country)
  (
    PARTITION first_half_2012 VALUES LESS THAN('01-JUL-2012')
    (
      SUBPARTITION europe VALUES ('ITALY', 'FRANCE'),
      SUBPARTITION americas VALUES ('CANADA', 'US'),
      SUBPARTITION asia VALUES ('PAKISTAN', 'INDIA')
    ),
    PARTITION second_half_2012 VALUES LESS THAN('01-JAN-2013')
);
```

Querying the ALL_TAB_SUBPARTITIONS view displays the subpartition names.

```
edb=# SELECT subpartition_name, high_value FROM ALL_TAB_SUBPARTITIONS;
 subpartition_name |     high_value
```

(continues on next page)

(continued from previous page)

```
-------------------+---------------------
 EUROPE            | 'ITALY', 'FRANCE'
 AMERICAS          | 'CANADA', 'US'
 ASIA              | 'PAKISTAN', 'INDIA'
 SYS0101           | DEFAULT
(4 rows)
```

To delete the `americas` subpartition from the `sales` table, invoke the following command:

```
ALTER TABLE sales DROP SUBPARTITION americas;
```

Querying the `ALL_TAB_SUBPARTITIONS` view demonstrates that the subpartition has been successfully deleted.

```
edb=# SELECT subpartition_name, high_value FROM ALL_TAB_SUBPARTITIONS;
 subpartition_name |     high_value
-------------------+---------------------
 EUROPE            | 'ITALY', 'FRANCE'
 ASIA              | 'PAKISTAN', 'INDIA'
 SYS0101           | DEFAULT
(3 rows)
```

4.15 TRUNCATE TABLE

Use the TRUNCATE TABLE command to remove the contents of a table, while preserving the table definition. When you truncate a table, any partitions or subpartitions of that table are also truncated. The syntax is:

```
TRUNCATE TABLE <table_name>
```

Description

The TRUNCATE TABLE command removes an entire table, and the data that resides in that table. When you delete a table, any partitions or subpartitions (of that table) are deleted as well.

To use the TRUNCATE TABLE command, you must be the owner of the partitioning root, a member of a group that owns the table, the schema owner, or a database superuser.

Parameters

table_name

The name (optionally schema-qualified) of the partitioned table.

4.15.1 Example - Emptying a Table

The example that follows removes the data from the sales table. Use the following command to create the sales table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date    date,
  amount  number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA')
);
```

Populate the sales table with the command:

```
INSERT INTO sales VALUES
  (10, '4519b', 'FRANCE', '17-Jan-2012', '45000'),
  (20, '3788a', 'INDIA', '01-Mar-2012', '75000'),
  (40, '9519b', 'US', '12-Apr-2012', '145000'),
  (20, '3788a', 'PAKISTAN', '04-Jun-2012', '37500'),
  (40, '4577b', 'US', '11-Nov-2012', '25000'),
  (30, '7588b', 'CANADA', '14-Dec-2012', '50000'),
  (30, '9519b', 'CANADA', '01-Feb-2012', '75000'),
```

(continues on next page)

(continued from previous page)

```
  (30, '4519b', 'CANADA', '08-Apr-2012', '120000'),
  (40, '3788a', 'US', '12-May-2012', '4950'),
  (10, '9519b', 'ITALY', '07-Jul-2012', '15000'),
  (10, '9519a', 'FRANCE', '18-Aug-2012', '650000'),
  (10, '9519b', 'FRANCE', '18-Aug-2012', '650000'),
  (20, '3788b', 'INDIA', '21-Sept-2012', '5090'),
  (40, '4788a', 'US', '23-Sept-2012', '4950'),
  (40, '4788b', 'US', '09-Oct-2012', '15000'),
  (20, '4519a', 'INDIA', '18-Oct-2012', '650000'),
  (20, '4519b', 'INDIA', '2-Dec-2012', '5090');
```

Querying the `sales` table shows that the partitions are populated with data.

```
edb=# SELECT tableoid::regclass, * FROM sales;
    tableoid    | dept_no | part_no | country  |        date         | amount
----------------+---------+---------+----------+---------------------+--------
 sales_americas |      40 | 9519b   | US       | 12-APR-12 00:00:00 | 145000
 sales_americas |      40 | 4577b   | US       | 11-NOV-12 00:00:00 |  25000
 sales_americas |      30 | 7588b   | CANADA   | 14-DEC-12 00:00:00 |  50000
 sales_americas |      30 | 9519b   | CANADA   | 01-FEB-12 00:00:00 |  75000
 sales_americas |      30 | 4519b   | CANADA   | 08-APR-12 00:00:00 | 120000
 sales_americas |      40 | 3788a   | US       | 12-MAY-12 00:00:00 |   4950
 sales_americas |      40 | 4788a   | US       | 23-SEP-12 00:00:00 |   4950
 sales_americas |      40 | 4788b   | US       | 09-OCT-12 00:00:00 |  15000
 sales_europe   |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_europe   |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_europe   |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_europe   |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_asia     |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_asia     |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_asia     |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_asia     |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_asia     |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
(17 rows)
```

To delete the contents of the `sales` table, invoke the following command:

```
TRUNCATE TABLE sales;
```

Now, querying the `sales` table shows that the data has been removed but the structure is intact.

```
edb=# SELECT tableoid::regclass, * FROM sales;
 tableoid | dept_no | part_no | country | date | amount
----------+---------+---------+---------+------+--------
(0 rows)
```

For more information about the `TRUNCATE TABLE` command, see the PostgreSQL documentation at:

https://www.postgresql.org/docs/current/static/sql-truncate.html

4.16 ALTER TABLE... TRUNCATE PARTITION

Use the ALTER TABLE... TRUNCATE PARTITION command to remove the data from the specified partition, leaving the partition structure intact. The syntax is:

```
ALTER TABLE <table_name> TRUNCATE PARTITION <partition_name>
  [{DROP|REUSE} STORAGE]
```

Description

Use the ALTER TABLE... TRUNCATE PARTITION command to remove the data from the specified partition, leaving the partition structure intact. When you truncate a partition, any subpartitions of that partition are also truncated.

ALTER TABLE... TRUNCATE PARTITION will not cause ON DELETE triggers that might exist for the table to fire, but it will fire ON TRUNCATE triggers. If an ON TRUNCATE trigger is defined for the partition, all BEFORE TRUNCATE triggers are fired before any truncation happens, and all AFTER TRUNCATE triggers are fired after the last truncation occurs.

You must have the TRUNCATE privilege on a table to invoke ALTER TABLE... TRUNCATE PARTITION.

Parameters

table_name

The name (optionally schema-qualified) of the partitioned table.

partition_name

The name of the partition to be deleted.

DROP STORAGE and REUSE STORAGE are included for compatibility only; the clauses are parsed and ignored.

4.16.1 Example - Emptying a Partition

The example that follows removes the data from a partition of the sales table. Use the following command to create the sales table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
```

(continues on next page)

(continued from previous page)

```
  PARTITION americas VALUES('US', 'CANADA')
);
```

Populate the `sales` table with the command:

```
INSERT INTO sales VALUES
  (10, '4519b', 'FRANCE', '17-Jan-2012', '45000'),
  (20, '3788a', 'INDIA', '01-Mar-2012', '75000'),
  (40, '9519b', 'US', '12-Apr-2012', '145000'),
  (20, '3788a', 'PAKISTAN', '04-Jun-2012', '37500'),
  (40, '4577b', 'US', '11-Nov-2012', '25000'),
  (30, '7588b', 'CANADA', '14-Dec-2012', '50000'),
  (30, '9519b', 'CANADA', '01-Feb-2012', '75000'),
  (30, '4519b', 'CANADA', '08-Apr-2012', '120000'),
  (40, '3788a', 'US', '12-May-2012', '4950'),
  (10, '9519b', 'ITALY', '07-Jul-2012', '15000'),
  (10, '9519a', 'FRANCE', '18-Aug-2012', '650000'),
  (10, '9519b', 'FRANCE', '18-Aug-2012', '650000'),
  (20, '3788b', 'INDIA', '21-Sept-2012', '5090'),
  (40, '4788a', 'US', '23-Sept-2012', '4950'),
  (40, '4788b', 'US', '09-Oct-2012', '15000'),
  (20, '4519a', 'INDIA', '18-Oct-2012', '650000'),
  (20, '4519b', 'INDIA', '2-Dec-2012', '5090');
```

Querying the `sales` table shows that the partitions are populated with data.

```
edb=# SELECT tableoid::regclass, * FROM sales;
    tableoid    | dept_no | part_no | country  |         date         | amount
----------------+---------+---------+----------+----------------------+--------
 sales_americas |      40 | 9519b   | US       | 12-APR-12 00:00:00 | 145000
 sales_americas |      40 | 4577b   | US       | 11-NOV-12 00:00:00 |  25000
 sales_americas |      30 | 7588b   | CANADA   | 14-DEC-12 00:00:00 |  50000
 sales_americas |      30 | 9519b   | CANADA   | 01-FEB-12 00:00:00 |  75000
 sales_americas |      30 | 4519b   | CANADA   | 08-APR-12 00:00:00 | 120000
 sales_americas |      40 | 3788a   | US       | 12-MAY-12 00:00:00 |   4950
 sales_americas |      40 | 4788a   | US       | 23-SEP-12 00:00:00 |   4950
 sales_americas |      40 | 4788b   | US       | 09-OCT-12 00:00:00 |  15000
 sales_europe   |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_europe   |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_europe   |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_europe   |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_asia     |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_asia     |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_asia     |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_asia     |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_asia     |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
(17 rows)
```

To delete the contents of the `americas` partition, invoke the following command:

```
ALTER TABLE sales TRUNCATE PARTITION americas;
```

Now, querying the `sales` table shows that the content of the `americas` partition has been removed.

```
edb=# SELECT tableoid::regclass, * FROM sales;
   tableoid    | dept_no | part_no | country  |        date         | amount
---------------+---------+---------+----------+---------------------+--------
 sales_europe |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_europe |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_europe |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_europe |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_asia   |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_asia   |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_asia   |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_asia   |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_asia   |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
(9 rows)
```

While the rows have been removed, the structure of the `americas` partition is still intact.

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |     high_value
----------------+--------------------
 EUROPE         | 'FRANCE', 'ITALY'
 ASIA           | 'INDIA', 'PAKISTAN'
 AMERICAS       | 'US', 'CANADA'
(3 rows)
```

4.17 ALTER TABLE... TRUNCATE SUBPARTITION

Use the ALTER TABLE... TRUNCATE SUBPARTITION command to remove all of the data from the specified subpartition, leaving the subpartition structure intact. The syntax is:

```
ALTER TABLE <table_name>
  TRUNCATE SUBPARTITION <subpartition_name>
  [{DROP|REUSE} STORAGE]
```

Description

The ALTER TABLE... TRUNCATE SUBPARTITION command removes all data from a specified subpartition, leaving the subpartition structure intact.

ALTER TABLE... TRUNCATE SUBPARTITION will not cause ON DELETE triggers that might exist for the table to fire, but it will fire ON TRUNCATE triggers. If an ON TRUNCATE trigger is defined for the subpartition, all BEFORE TRUNCATE triggers are fired before any truncation happens, and all AFTER TRUNCATE triggers are fired after the last truncation occurs.

You must have the TRUNCATE privilege on a table to invoke ALTER TABLE... TRUNCATE SUBPARTITION.

Parameters

table_name

The name (optionally schema-qualified) of the partitioned table.

subpartition_name

The name of the subpartition to be truncated.

The DROP STORAGE and REUSE STORAGE clauses are included for compatibility only; the clauses are parsed and ignored.

4.17.1 Example - Emptying a Subpartition

The example that follows removes the data from a subpartition of the sales table. Use the following command to create the sales table:

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date) SUBPARTITION BY LIST (country)
(
  PARTITION "2011" VALUES LESS THAN('01-JAN-2012')
  (
```

(continues on next page)

(continued from previous page)

```
    SUBPARTITION europe_2011 VALUES ('ITALY', 'FRANCE'),
    SUBPARTITION asia_2011 VALUES ('PAKISTAN', 'INDIA'),
    SUBPARTITION americas_2011 VALUES ('US', 'CANADA')
  ),
  PARTITION "2012" VALUES LESS THAN('01-JAN-2013')
  (
    SUBPARTITION europe_2012 VALUES ('ITALY', 'FRANCE'),
    SUBPARTITION asia_2012 VALUES ('PAKISTAN', 'INDIA'),
    SUBPARTITION americas_2012 VALUES ('US', 'CANADA')
  ),
  PARTITION "2013" VALUES LESS THAN('01-JAN-2015')
  (
    SUBPARTITION europe_2013 VALUES ('ITALY', 'FRANCE'),
    SUBPARTITION asia_2013 VALUES ('PAKISTAN', 'INDIA'),
    SUBPARTITION americas_2013 VALUES ('US', 'CANADA')
  )
);
```

Populate the `sales` table with the command:

```
INSERT INTO sales VALUES
  (10, '4519b', 'FRANCE', '17-Jan-2011', '45000'),
  (20, '3788a', 'INDIA', '01-Mar-2012', '75000'),
  (40, '9519b', 'US', '12-Apr-2012', '145000'),
  (20, '3788a', 'PAKISTAN', '04-Jun-2012', '37500'),
  (40, '4577b', 'US', '11-Nov-2012', '25000'),
  (30, '7588b', 'CANADA', '14-Dec-2011', '50000'),
  (30, '4519b', 'CANADA', '08-Apr-2012', '120000'),
  (40, '3788a', 'US', '12-May-2011', '4950'),
  (20, '3788a', 'US', '04-Apr-2012', '37500'),
  (40, '4577b', 'INDIA', '11-Jun-2011', '25000'),
  (10, '9519b', 'ITALY', '07-Jul-2012', '15000'),
  (20, '4519b', 'INDIA', '2-Dec-2012', '5090');
```

Querying the `sales` table shows that the rows have been distributed amongst the subpartitions.

```
edb=# SELECT tableoid::regclass, * FROM sales;
    tableoid       | dept_no| part_no | country|        date         | amount
-------------------+--------+---------+--------+---------------------+--------
sales_americas_2011|      30| 7588b   | CANADA | 14-DEC-11 00:00:00 | 50000
sales_americas_2011|      40| 3788a   | US     | 12-MAY-11 00:00:00 | 4950
sales_europe_2011  |      10| 4519b   | FRANCE | 17-JAN-11 00:00:00 | 45000
sales_asia_2011    |      40| 4577b   | INDIA  | 11-JUN-11 00:00:00 | 25000
sales_americas_2012|      40| 9519b   | US     | 12-APR-12 00:00:00 |145000
sales_americas_2012|      40| 4577b   | US     | 11-NOV-12 00:00:00 | 25000
sales_americas_2012|      30| 4519b   | CANADA | 08-APR-12 00:00:00 |120000
sales_americas_2012|      20| 3788a   | US     | 04-APR-12 00:00:00 | 37500
sales_europe_2012  |      10| 9519b   | ITALY  | 07-JUL-12 00:00:00 | 15000
sales_asia_2012    |      20| 3788a   | INDIA  | 01-MAR-12 00:00:00 | 75000
sales_asia_2012    |      20| 3788a   |PAKISTAN| 04-JUN-12 00:00:00 | 37500
sales_asia_2012    |      20| 4519b   | INDIA  | 02-DEC-12 00:00:00 | 5090
(12 rows)
```

To delete the contents of the 2012_americas partition, invoke the following command:

```
ALTER TABLE sales TRUNCATE SUBPARTITION "americas_2012";
```

Now, querying the sales table shows that the content of the americas_2012 partition has been removed.

```
edb=# SELECT tableoid::regclass, * FROM sales;
 tableoid          | dept_no| part_no | country  |        date         |amount
-------------------+--------+---------+----------+---------------------+------
sales_americas_2011|      30| 7588b   | CANADA   | 14-DEC-11 00:00:00 |50000
sales_americas_2011|      40| 3788a   | US       | 12-MAY-11 00:00:00 | 4950
sales_europe_2011  |      10| 4519b   | FRANCE   | 17-JAN-11 00:00:00 |45000
sales_asia_2011    |      40| 4577b   | INDIA    | 11-JUN-11 00:00:00 |25000
sales_europe_2012  |      10| 9519b   | ITALY    | 07-JUL-12 00:00:00 |15000
sales_asia_2012    |      20| 3788a   | INDIA    | 01-MAR-12 00:00:00 |75000
sales_asia_2012    |      20| 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |37500
sales_asia_2012    |      20| 4519b   | INDIA    | 02-DEC-12 00:00:00 | 5090
(8 rows)
```

While the rows have been removed, the structure of the 2012_americas partition is still intact.

```
edb=# SELECT subpartition_name, high_value FROM ALL_TAB_SUBPARTITIONS;
 subpartition_name |     high_value
-------------------+---------------------
 EUROPE_2011       | 'ITALY', 'FRANCE'
 ASIA_2011         | 'PAKISTAN', 'INDIA'
 AMERICAS_2011     | 'US', 'CANADA'
 EUROPE_2012       | 'ITALY', 'FRANCE'
 ASIA_2012         | 'PAKISTAN', 'INDIA'
 AMERICAS_2012     | 'US', 'CANADA'
 EUROPE_2013       | 'ITALY', 'FRANCE'
 ASIA_2013         | 'PAKISTAN', 'INDIA'
 AMERICAS_2013     | 'US', 'CANADA'
(9 rows)
```

CHAPTER 5

Handling Stray Values in a LIST or RANGE Partitioned Table

A DEFAULT or MAXVALUE partition or subpartition will capture any rows that do not meet the other partitioning rules defined for a table.

Defining a DEFAULT Partition

A DEFAULT partition will capture any rows that do not fit into any other partition in a LIST partitioned (or subpartitioned) table. If you do not include a DEFAULT rule, any row that does not match one of the values in the partitioning constraints will result in an error. Each LIST partition or subpartition may have its own DEFAULT rule.

The syntax of a DEFAULT rule is:

```
PARTITION [<partition_name>] VALUES (DEFAULT)
```

Where partition_name specifies the name of the partition or subpartition that will store any rows that do not match the rules specified for other partitions.

The last example created a list partitioned table in which the server decided which partition to store the data based upon the value of the country column. If you attempt to add a row in which the value of the country column contains a value not listed in the rules, Advanced Server reports an error.

```
edb=# INSERT INTO sales VALUES
edb-#  (40, '3000x', 'IRELAND', '01-Mar-2012', '45000');
ERROR:  no partition of relation "sales_2012" found for row
DETAIL:  Partition key of the failing row contains (country) = (IRELAND).
```

The following example creates the same table, but adds a DEFAULT partition. The server will store any rows that do not match a value specified in the partitioning rules for europe, asia, or americas partitions in the others partition.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY LIST(country)
(
  PARTITION europe VALUES('FRANCE', 'ITALY'),
  PARTITION asia VALUES('INDIA', 'PAKISTAN'),
  PARTITION americas VALUES('US', 'CANADA'),
  PARTITION others VALUES (DEFAULT)
);
```

To test the DEFAULT partition, add row with a value in the country column that does not match one of the countries specified in the partitioning constraints.

```
INSERT INTO sales VALUES
  (40, '3000x', 'IRELAND', '01-Mar-2012', '45000');
```

Querying the contents of the sales table confirms that the previously rejected row is now stored in the sales_others partition.

```
edb=# SELECT tableoid::regclass, * FROM sales;
    tableoid    | dept_no | part_no | country  |        date         | amount
----------------+---------+---------+----------+---------------------+--------
 sales_americas |      40 | 9519b   | US       | 12-APR-12 00:00:00 | 145000
 sales_americas |      40 | 4577b   | US       | 11-NOV-12 00:00:00 |  25000
 sales_americas |      30 | 7588b   | CANADA   | 14-DEC-12 00:00:00 |  50000
 sales_americas |      30 | 9519b   | CANADA   | 01-FEB-12 00:00:00 |  75000
 sales_americas |      30 | 4519b   | CANADA   | 08-APR-12 00:00:00 | 120000
 sales_americas |      40 | 3788a   | US       | 12-MAY-12 00:00:00 |   4950
 sales_americas |      40 | 4788a   | US       | 23-SEP-12 00:00:00 |   4950
 sales_americas |      40 | 4788b   | US       | 09-OCT-12 00:00:00 |  15000
 sales_europe   |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_europe   |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_europe   |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_europe   |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_asia     |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_asia     |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_asia     |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_asia     |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_asia     |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
 sales_others   |      40 | 3000x   | IRELAND  | 01-MAR-12 00:00:00 |  45000
(18 rows)
```

Advanced Server provides the following methods to re-assign the contents of a DEFAULT partition or sub-partition:

- You can use the ALTER TABLE... ADD PARTITION command to add a partition to a table with a DEFAULT rule as long as there are no conflicting values between existing rows in the table and

the values of the partition to be added. You can alternatively use the ALTER TABLE... SPLIT PARTITION command to split an existing partition. Examples are shown following this bullet point list.

- You can use the ALTER TABLE... ADD SUBPARTITION command to add a subpartition to a table with a DEFAULT rule as long as there are no conflicting values between existing rows in the table and the values of the subpartition to be added. You can alternatively use the ALTER TABLE... SPLIT SUBPARTITION command to split an existing subpartition.

Adding a Partition to a Table with a DEFAULT Partition

Using the table that was created with the CREATE TABLE sales command shown at the beginning of this section, the following shows use of the ALTER TABLE... ADD PARTITION command assuming there is no conflict of values between the existing rows in the table and the values of the partition to be added.

```
edb=# ALTER TABLE sales ADD PARTITION africa values ('SOUTH AFRICA',
'KENYA');
ALTER TABLE
```

However, the following shows the error when there are conflicting values when the following rows have been inserted into the table.

```
edb=# INSERT INTO sales (dept_no, country) VALUES
(1,'FRANCE'),(2,'INDIA'),(3,'US'),(4,'SOUTH AFRICA'),(5,'NEPAL');
INSERT 0 5
```

Row (4,'SOUTH AFRICA') conflicts with the VALUES list in the ALTER TABLE... ADD PARTITION statement, thus resulting in an error.

```
edb=# ALTER TABLE sales ADD PARTITION africa values ('SOUTH AFRICA',
'KENYA');
ERROR:  updated partition constraint for default partition "sales_others"
would be violated by some row
```

Splitting a DEFAULT Partition

The following example splits a DEFAULT partition, redistributing the partition's content between two new partitions. The table was created with the CREATE TABLE sales command shown at the beginning of this section.

The following inserts rows into the table including rows into the DEFAULT partition.

```
INSERT INTO sales VALUES
  (10, '4519b', 'FRANCE', '17-Jan-2012', '45000'),
  (10, '9519b', 'ITALY', '07-Jul-2012', '15000'),
  (20, '3788a', 'INDIA', '01-Mar-2012', '75000'),
  (20, '3788a', 'PAKISTAN', '04-Jun-2012', '37500'),
  (30, '9519b', 'US', '12-Apr-2012', '145000'),
  (30, '7588b', 'CANADA', '14-Dec-2012', '50000'),
  (40, '4519b', 'SOUTH AFRICA', '08-Apr-2012', '120000'),
  (40, '4519b', 'KENYA', '08-Apr-2012', '120000'),
  (50, '3788a', 'CHINA', '12-May-2012', '4950');
```

The partitions include the `DEFAULT others` partition.

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |     high_value
----------------+---------------------
 EUROPE         | 'FRANCE', 'ITALY'
 ASIA           | 'INDIA', 'PAKISTAN'
 AMERICAS       | 'US', 'CANADA'
 OTHERS         | DEFAULT
(4 rows)
```

The following shows the rows distributed amongst the partitions.

```
edb=# SELECT tableoid::regclass, * FROM sales;
 tableoid     | dept_no| part_no |   country    |        date         | amount
--------------+--------+---------+--------------+---------------------+-------
sales_americas|     30 | 9519b   | US           | 12-APR-12 00:00:00 |145000
sales_americas|     30 | 7588b   | CANADA       | 14-DEC-12 00:00:00 | 50000
sales_europe  |     10 | 4519b   | FRANCE       | 17-JAN-12 00:00:00 | 45000
sales_europe  |     10 | 9519b   | ITALY        | 07-JUL-12 00:00:00 | 15000
sales_asia    |     20 | 3788a   | INDIA        | 01-MAR-12 00:00:00 | 75000
sales_asia    |     20 | 3788a   | PAKISTAN     | 04-JUN-12 00:00:00 | 37500
sales_others  |     40 | 4519b   | SOUTH AFRICA | 08-APR-12 00:00:00 |120000
sales_others  |     40 | 4519b   | KENYA        | 08-APR-12 00:00:00 |120000
sales_others  |     50 | 3788a   | CHINA        | 12-MAY-12 00:00:00 |  4950
(9 rows)
```

The following command splits the `DEFAULT others` partition into two partitions named `africa` and `others`.

```
ALTER TABLE sales SPLIT PARTITION others VALUES
  ('SOUTH AFRICA', 'KENYA')
  INTO (PARTITION africa, PARTITION others);
```

The partitions now include the `africa` partition along with the `DEFAULT others` partition.

```
edb=# SELECT partition_name, high_value FROM ALL_TAB_PARTITIONS;
 partition_name |       high_value
----------------+-------------------------
 EUROPE         | 'FRANCE', 'ITALY'
 ASIA           | 'INDIA', 'PAKISTAN'
 AMERICAS       | 'US', 'CANADA'
 AFRICA         | 'SOUTH AFRICA', 'KENYA'
 OTHERS         | DEFAULT
(5 rows)
```

The following shows that the rows have been redistributed across the new partitions.

```
edb=# SELECT tableoid::regclass, * FROM sales;
 tableoid      |dept_no | part_no |   country   |        date         | amount
---------------+--------+---------+-------------+---------------------+-------
sales_americas |     30 | 9519b   | US          | 12-APR-12 00:00:00 |145000
```

(continues on next page)

(continued from previous page)

```
sales_americas |     30 | 7588b   | CANADA       | 14-DEC-12 00:00:00 | 50000
sales_europe   |     10 | 4519b   | FRANCE       | 17-JAN-12 00:00:00 | 45000
sales_europe   |     10 | 9519b   | ITALY        | 07-JUL-12 00:00:00 | 15000
sales_asia     |     20 | 3788a   | INDIA        | 01-MAR-12 00:00:00 | 75000
sales_asia     |     20 | 3788a   | PAKISTAN     | 04-JUN-12 00:00:00 | 37500
sales_africa   |     40 | 4519b   | SOUTH AFRICA| 08-APR-12 00:00:00 |120000
sales_africa   |     40 | 4519b   | KENYA        | 08-APR-12 00:00:00 |120000
sales_others_1 |     50 | 3788a   | CHINA        | 12-MAY-12 00:00:00 | 4950
(9 rows)
```

Defining a MAXVALUE Partition

A `MAXVALUE` partition (or subpartition) will capture any rows that do not fit into any other partition in a range-partitioned (or subpartitioned) table. If you do not include a `MAXVALUE` rule, any row that exceeds the maximum limit specified by the partitioning rules will result in an error. Each partition or subpartition may have its own `MAXVALUE` partition.

The syntax of a `MAXVALUE` rule is:

```
PARTITION [<partition_name>] VALUES LESS THAN (MAXVALUE)
```

Where `partition_name` specifies the name of the partition that will store any rows that do not match the rules specified for other partitions.

The last example created a range-partitioned table in which the data was partitioned based upon the value of the `date` column. If you attempt to add a row with a `date` that exceeds a date listed in the partitioning constraints, Advanced Server reports an error.

```
edb=# INSERT INTO sales VALUES
edb-#   (40, '3000x', 'IRELAND', '01-Mar-2013', '45000');
ERROR:  no partition of relation "sales" found for row
DETAIL:  Partition key of the failing row contains (date) = (01-MAR-13
00:00:00).
```

The following `CREATE TABLE` command creates the same table, but with a `MAXVALUE` partition. Instead of throwing an error, the server will store any rows that do not match the previous partitioning constraints in the `others` partition.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  date        date,
  amount      number
)
PARTITION BY RANGE(date)
(
  PARTITION q1_2012 VALUES LESS THAN('2012-Apr-01'),
  PARTITION q2_2012 VALUES LESS THAN('2012-Jul-01'),
  PARTITION q3_2012 VALUES LESS THAN('2012-Oct-01'),
```

(continues on next page)

(continued from previous page)

```
  PARTITION q4_2012 VALUES LESS THAN('2013-Jan-01'),
  PARTITION others VALUES LESS THAN (MAXVALUE)
);
```

To test the MAXVALUE partition, add a row with a value in the date column that exceeds the last date value listed in a partitioning rule. The server will store the row in the others partition.

```
INSERT INTO sales VALUES
  (40, '3000x', 'IRELAND', '01-Mar-2013', '45000');
```

Querying the contents of the sales table confirms that the previously rejected row is now stored in the sales_others partition.

```
edb=# SELECT tableoid::regclass, * FROM sales;
   tableoid    | dept_no | part_no | country  |        date         | amount
---------------+---------+---------+----------+---------------------+--------
 sales_q1_2012 |      10 | 4519b   | FRANCE   | 17-JAN-12 00:00:00 |  45000
 sales_q1_2012 |      20 | 3788a   | INDIA    | 01-MAR-12 00:00:00 |  75000
 sales_q1_2012 |      30 | 9519b   | CANADA   | 01-FEB-12 00:00:00 |  75000
 sales_q2_2012 |      40 | 9519b   | US       | 12-APR-12 00:00:00 | 145000
 sales_q2_2012 |      20 | 3788a   | PAKISTAN | 04-JUN-12 00:00:00 |  37500
 sales_q2_2012 |      30 | 4519b   | CANADA   | 08-APR-12 00:00:00 | 120000
 sales_q2_2012 |      40 | 3788a   | US       | 12-MAY-12 00:00:00 |   4950
 sales_q3_2012 |      10 | 9519b   | ITALY    | 07-JUL-12 00:00:00 |  15000
 sales_q3_2012 |      10 | 9519a   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_q3_2012 |      10 | 9519b   | FRANCE   | 18-AUG-12 00:00:00 | 650000
 sales_q3_2012 |      20 | 3788b   | INDIA    | 21-SEP-12 00:00:00 |   5090
 sales_q3_2012 |      40 | 4788a   | US       | 23-SEP-12 00:00:00 |   4950
 sales_q4_2012 |      40 | 4577b   | US       | 11-NOV-12 00:00:00 |  25000
 sales_q4_2012 |      30 | 7588b   | CANADA   | 14-DEC-12 00:00:00 |  50000
 sales_q4_2012 |      40 | 4788b   | US       | 09-OCT-12 00:00:00 |  15000
 sales_q4_2012 |      20 | 4519a   | INDIA    | 18-OCT-12 00:00:00 | 650000
 sales_q4_2012 |      20 | 4519b   | INDIA    | 02-DEC-12 00:00:00 |   5090
 sales_others  |      40 | 3000x   | IRELAND  | 01-MAR-13 00:00:00 |  45000
(18 rows)
```

Please note that Advanced Server does not have a way to re-assign the contents of a MAXVALUE partition or subpartition.

- You cannot use the ALTER TABLE... ADD PARTITION statement to add a partition to a table with a MAXVALUE rule, but you can use the ALTER TABLE... SPLIT PARTITION statement to split an existing partition.
- You cannot use the ALTER TABLE... ADD SUBPARTITION statement to add a subpartition to a table with a MAXVALUE rule , but you can split an existing subpartition with the ALTER TABLE... SPLIT SUBPARTITION statement.

CHAPTER 6

Specifying Multiple Partitioning Keys in a RANGE Partitioned Table

You can often improve performance by specifying multiple key columns for a `RANGE` partitioned table. If you often select rows using comparison operators (based on a greater-than or less-than value) on a small set of columns, consider using those columns in `RANGE` partitioning rules.

Specifying Multiple Keys in a Range-Partitioned Table

Range-partitioned table definitions may include multiple columns in the partitioning key. To specify multiple partitioning keys for a range-partitioned table, include the column names in a comma-separated list after the `PARTITION BY RANGE` clause.

```
CREATE TABLE sales
(
  dept_no     number,
  part_no     varchar2,
  country     varchar2(20),
  sale_year    number,
  sale_month   number,
  sale_day     number,
  amount      number
)
PARTITION BY RANGE(sale_year, sale_month)
(
  PARTITION q1_2012
    VALUES LESS THAN(2012, 4),
  PARTITION q2_2012
    VALUES LESS THAN(2012, 7),
  PARTITION q3_2012
    VALUES LESS THAN(2012, 10),
  PARTITION q4_2012
    VALUES LESS THAN(2013, 1)
);
```

If a table is created with multiple partitioning keys, you must specify multiple key values when querying the table to take full advantage of partition pruning.

```
edb=# EXPLAIN SELECT * FROM sales WHERE sale_year = 2012 AND sale_month = 8;
                                QUERY PLAN
---------------------------------------------------------------------------
 Append  (cost=0.00..14.35 rows=1 width=250)
   ->  Seq Scan on sales_q3_2012  (cost=0.00..14.35 rows=1 width=250)
         Filter: ((sale_year = '2012'::numeric) AND (sale_month =
'8'::numeric))
(3 rows)
```

Since all rows with a value of `8` in the `sale_month` column and a value of `2012` in the `sale_year` column will be stored in the `q3_2012` partition, Advanced Server searches only that partition.

CHAPTER 7

Retrieving Information about a Partitioned Table

Advanced Server provides five system catalog views that you can use to view information about the structure of partitioned tables.

Querying the Partitioning Views

You can query the following views to retrieve information about partitioned and subpartitioned tables:

- ALL_PART_TABLES
- ALL_TAB_PARTITIONS
- ALL_TAB_SUBPARTITIONS
- ALL_PART_KEY_COLUMNS
- ALL_SUBPART_KEY_COLUMNS

The structure of each view is explained in *Table Partitioning Views - Reference*. If you are using the EDB-PSQL client, you can also discover the structure of a view by entering:

```
\d <view_name>
```

Where view_name specifies the name of the table partitioning view.

Querying a view can provide information about the structure of a partitioned or subpartitioned table. For example, the following code snippet displays the names of a subpartitioned table:

```
edb=# SELECT subpartition_name, partition_name FROM ALL_TAB_SUBPARTITIONS;
 subpartition_name | partition_name
-------------------+----------------
 EUROPE_2011       | EUROPE
 EUROPE_2012       | EUROPE
 ASIA_2011         | ASIA
 ASIA_2012         | ASIA
```

(continues on next page)

(continued from previous page)

```
 AMERICAS_2011      | AMERICAS
 AMERICAS_2012      | AMERICAS
(6 rows)
```

7.1 Table Partitioning Views - Reference

Query the following catalog views (compatible with Oracle databases), to review detailed information about your partitioned tables.

7.1.1 ALL_PART_TABLES

The following table lists the information available in the `ALL_PART_TABLES` view:

Column	Type	Description
owner	name	The owner of the table.
schema_name	name	The schema in which the table resides.
table_name	name	The name of the table.
partitioning_type	text	RANGE, LIST or HASH
subpartitioning_type	text	RANGE, LIST, HASH, or NONE
partition_count	bigint	The number of partitions.
def_subpartition_count	integer	The default subpartition count - this will always be 0.
partitioning_key_count	integer	The number of columns listed in the partition by clause.
subpartitioning_key_count	integer	The number of columns in the subpartition by clause.
status	character varying(8)	This column will always be VALID.
def_tablespace_name	character varying(30)	This column will always be NULL.
def_pct_free	numeric	This column will always be NULL.
def_pct_used	numeric	This column will always be NULL.
def_ini_trans	numeric	This column will always be NULL.
def_max_trans	numeric	This column will always be NULL.
def_initial_extent	character varying(40)	This column will always be NULL.
def_next_extent	character varying(40)	This column will always be NULL.
def_min_extents	character varying(40)	This column will always be NULL.
def_max_extents	character varying(40)	This column will always be NULL.
def_pct_increase	character varying(40)	This column will always be NULL.
def_freelists	numeric	This column will always be NULL.
def_freelist_groups	numeric	This column will always be NULL.
def_logging	character varying(7)	This column will always be YES
def_compression	character varying(8)	This column will always be NONE
def_buffer_pool	character varying(7)	This column will always be DEFAULT
ref_ptn_constraint_name	character varying(30)	This column will always be NULL
interval	character varying(1000)	This column will always be NULL

7.1.2 ALL_TAB_PARTITIONS

The following table lists the information available in the ALL_TAB_PARTITIONS view:

Column	Type	Description
table_owner	name	The owner of the table.
schema_name	name	The schema in which the table resides.
table_name	name	The name of the table.
composite	text	YES if the table is subpartitioned; NO if it is not subpartitioned.
partition_name	name	The name of the partition.
subpartition_count	bigint	The number of subpartitions for this partition.
high_value	text	The high partitioning value specified in the CREATE TABLE statement.
high_value_length	integer	The length of high partitioning value.
partition_position	integer	The ordinal position of this partition.
tablespace_name	name	The tablespace in which this partition resides.
pct_free	numeric	This column will always be 0.
pct_used	numeric	This column will always be 0.
ini_trans	numeric	This column will always be 0.
max_trans	numeric	This column will always be 0.
initial_extent	numeric	This column will always be NULL.
next_extent	numeric	This column will always be NULL.
min_extent	numeric	This column will always be 0.
max_extent	numeric	This column will always be 0.
pct_increase	numeric	This column will always be 0.
freelists	numeric	This column will always be NULL
freelist_groups	numeric	This column will always be NULL
logging	character varying(7)	This column will always be YES.
compression	character varying(8)	This column will always be NONE.
num_rows	numeric	The approx. number of rows in this partition.
blocks	integer	The approx. number of blocks in this partition.
empty_blocks	numeric	This column will always be NULL
avg_space	numeric	This column will always be NULL
chain_cnt	numeric	This column will always be NULL
avg_row_len	numeric	This column will always be NULL
sample_size	numeric	This column will always be NULL
last_analyzed	timestamp without time zone	This column will always be NULL

continues on next page

Table 1 – continued from previous page

Column	Type	Description
buffer_pool	character varying(7)	This column will always be NULL
global_stats	character varying(3)	This column will always be YES.
user_stats	character varying(3)	This column will always be NO.
backing_table	regclass	OID of the backing table for this partition.

7.1.3 ALL_TAB_SUBPARTITIONS

The following table lists the information available in the ALL_TAB_SUBPARTITIONS view:

Column	Type	Description
table_owner	name	The name of the owner of the table.
schema_name	name	The name of the schema in which the table resides.
table_name	name	The name of the table.
partition_name	name	The name of the partition.
subpartition_name	name	The name of the subpartition.
high_value	text	The high partitioning value specified in the CREATE TABLE statement.
high_value_length	integer	The length of high partitioning value.
subpartition_position	integer	The ordinal position of this subpartition.
tablespace_name	name	The tablespace in which this subpartition resides.
pct_free	numeric	This column will always be 0.
pct_used	numeric	This column will always be 0.
ini_trans	numeric	This column will always be 0.
max_trans	numeric	This column will always be 0.
initial_extent	numeric	This column will always be NULL.
next_extent	numeric	This column will always be NULL.
min_extent	numeric	This column will always be 0.
max_extent	numeric	This column will always be 0.
pct_increase	numeric	This column will always be 0.
freelists	numeric	This column will always be NULL.
freelist_groups	numeric	This column will always be NULL.
logging	character varying(7)	This column will always be YES.
compression	character varying(8)	This column will always be NONE.
num_rows	numeric	The approx. number of rows in this subpartition.

continues on next page

Table 2 – continued from previous page

Column	Type	Description
blocks	integer	The approx. number of blocks in this subpartition.
empty_blocks	numeric	This column will always be NULL.
avg_space	numeric	This column will always be NULL.
chain_cnt	numeric	This column will always be NULL.
avg_row_len	numeric	This column will always be NULL.
sample_size	numeric	This column will always be NULL.
last_analyzed	timestamp without time zone	This column will always be NULL.
buffer_pool	character varying(7)	This column will always be NULL.
global_stats	character varying(3)	This column will always be YES.
user_stats	character varying(3)	This column will always be NO.
backing_table	regclass	OID of the backing table for this subpartition.

7.1.4 ALL_PART_KEY_COLUMNS

The following table lists the information available in the ALL_PART_KEY_COLUMNS view:

Column	Type	Description
owner	name	The name of the table owner.
schema_name	name	The name of the schema on which the table resides.
name	name	The name of the table.
object_type	character(5)	This column will always be TABLE.
column_name	name	The name of the partitioning key column.
column_position	integer	The position of this column within the partitioning key (the first column has a column position of 1, the second column has a column position of 2...)

7.1.5 ALL_SUBPART_KEY_COLUMNS

The following table lists the information available in the ALL_SUBPART_KEY_COLUMNS view:

Column	Type	Description
owner	name	The name of the table owner.
schema_name	name	The name of the schema on which the table resides.
name	name	The name of the table.
object_type	character(5)	This column will always be TABLE.
column_name	name	The name of the partitioning key column.
column_position	integer	The position of this column within the subpartitioning key (the first column has a column position of 1, the second column has a column position of 2. . .)

CHAPTER 8

Conclusion

EDB Postgres™ Advanced Server Database Compatibility Table Partitioning Guide

EnterpriseDB® Corporation

34 Crosby Drive, Suite 201, Bedford, MA 01730, USA

T +1 781 357 3390 F +1 978 467 1307 E

info@enterprisedb.com

www.enterprisedb.com

Index

H

I

P

R

S

T

U