Specifying Partitioning When Creating Tables and Indexes

Creating a partitioned table or index is very similar to creating a nonpartitioned table or index.

When creating a partitioned table or index, you include a partitioning clause in the CREATE TABLE statement. The partitioning clause, and subclauses, that you include depend upon the type of partitioning you want to achieve.

Partitioning is possible on both regular (heap organized) tables and index-organized tables, except for those containing LONG or LONG RAW columns. You can create nonpartitioned global indexes, range or hash partitioned global indexes, and local indexes on partitioned tables.

When you create (or alter) a partitioned table, a row movement clause (either ENABLE ROW MOVEMENT or DISABLE ROW MOVEMENT) can be specified. This clause either enables or disables the migration of a row to a new partition if its key is updated. The default is DISABLE ROW MOVEMENT.

You can specify up to a total of 1024K-1 partitions for a single-level partitioned tables, or subpartitions for a composite partitioned table.

Creating automatic list composite partitioned tables and interval subpartitions can save space because these methods only create subpartitions in the presence of data. Deferring subpartition segment creation when creating new partitions on demand ensures that a subpartition segment is only created when the first matching row is inserted.

The following topics present details and examples of creating partitions for the various types of partitioned tables and indexes:

* [About Creating Range-Partitioned Tables and Global Indexes](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-9FCAFF8C-0AA9-49F1-A7B8-7611C8AF56DD)
* [Creating Range-Interval-Partitioned Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-F83CB777-F92C-43AB-A20D-EB2BCC82C32C)
* [About Creating Hash Partitioned Tables and Global Indexes](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-BF3F38E1-62BB-4EE3-86C1-A2EF8A258B1F)
* [About Creating List-Partitioned Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-17243D8E-03EA-44E7-AC0B-6525F14C784A)
* [Creating Reference-Partitioned Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-00923EB3-05F6-41F7-8437-E42FC9BD9571)
* [Creating Interval-Reference Partitioned Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-52397EF9-EC3B-4E80-8211-77E683263530)
* [Creating a Table Using In-Memory Column Store With Partitioning](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-01B3A14E-BC74-49D2-B719-B2D8B7692C97)
* [Creating a Table with Read-Only Partitions or Subpartitions](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-9D7149B6-A2FF-47CA-8F00-47CBFD33F82B)
* [Creating a Partitioned External Table](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-09CFA3CD-66B7-4CDF-A086-6958D113BDD1)
* [Specifying Partitioning on Key Columns](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-5C34BF3F-B728-47C7-B656-AE4FA9FFE2C3)
* [Using Virtual Column-Based Partitioning](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-AF6D08D7-5E0F-4E08-B682-74CCCAEC64D1)
* [Using Table Compression with Partitioned Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-7D211E4B-FFFD-4146-9010-20ED1232FF2C)
* [Using Key Compression with Partitioned Indexes](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-0C13E8A8-568B-4A60-8A46-F1120D193EFE)
* [Specifying Partitioning with Segments](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-548F1C63-0948-4B6E-95F9-6DCC7F5A9F95)
* [Specifying Partitioning When Creating Index-Organized Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-46136CB9-ED71-4150-96F2-D50121F0BCF0)
* [Partitioning Restrictions for Multiple Block Sizes](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-24050391-B7C5-4AE2-86D4-B5438412C3F6)
* [Partitioning of Collections in XMLType and Objects](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-7F8EBE79-9FF2-4B2D-9764-C11263FA550F)

**See Also:**

* [*Oracle Database Administrator's Guide*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=ADMIN015) for information about managing tables
* [*Oracle Database SQL Language Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=SQLRF01402) for the exact syntax of the partitioning clauses for creating and altering partitioned tables and indexes, any restrictions on their use, and specific privileges required for creating and altering tables
* [*Oracle Database SecureFiles and Large Objects Developer's Guide*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=ADLOB45309)for information specific to creating partitioned tables containing columns with LOBs or other objects stored as LOBs
* [*Oracle Database Object-Relational Developer's Guide*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=ADOBJ006)for information specific to creating tables with object types, nested tables, or VARRAYs

4.1.1 About Creating Range-Partitioned Tables and Global Indexes

The PARTITION BY RANGE clause of the CREATE TABLE statement specifies that the table or index is to be range-partitioned.

The PARTITION clauses identify the individual partition ranges, and the optional subclauses of a PARTITION clause can specify physical and other attributes specific to a partition segment. If not overridden at the partition level, partitions inherit the attributes of their underlying table.

The following topics are discussed:

* [Creating a Range-Partitioned Table](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-F0BCECA4-A485-4DA2-863A-6DB7C064059C)
* [Creating a Range-Partitioned Table With More Complexity](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-66D19C36-C512-48DE-88B4-AFC6CCACDFEB)
* [Creating a Range-Partitioned Global Index](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-4EEB5C77-4D02-430D-9AC2-10D377A06A15)

4.1.1.1 Creating a Range-Partitioned Table

Use the PARTITION BY RANGE clause of the CREATE TABLE statement to create a range-partitioned table.

[Example 4-1](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-F0BCECA4-A485-4DA2-863A-6DB7C064059C__CHDJDFAF) creates a table of four partitions, one for each quarter of sales. time\_id is the **partitioning column**, while its values constitute the **partitioning key** of a specific row. The VALUES LESS THAN clause determines the **partition bound**: rows with partitioning key values that compare less than the ordered list of values specified by the clause are stored in the partition. Each partition is given a name (sales\_q1\_2006, sales\_q2\_2006, sales\_q3\_2006, sales\_q4\_2006), and each partition is contained in a separate tablespace (tsa, tsb, tsc, tsd). A row with time\_id=17-MAR-2006 would be stored in partition sales\_q1\_2006.

Live SQL:

View and run a related example on Oracle Live SQL at [*Oracle Live SQL: Creating a Range Partitioned Table*](https://livesql.oracle.com/apex/livesql/docs/vldbg/partitioning/range-partitioning-example.html).

Example 4-1 Creating a range-partitioned table

Copy

CREATE TABLE sales

( prod\_id NUMBER(6)

, cust\_id NUMBER

, time\_id DATE

, channel\_id CHAR(1)

, promo\_id NUMBER(6)

, quantity\_sold NUMBER(3)

, amount\_sold NUMBER(10,2)

)

PARTITION BY RANGE (time\_id)

( PARTITION sales\_q1\_2006 VALUES LESS THAN (TO\_DATE('01-APR-2006','dd-MON-yyyy'))

TABLESPACE tsa

, PARTITION sales\_q2\_2006 VALUES LESS THAN (TO\_DATE('01-JUL-2006','dd-MON-yyyy'))

TABLESPACE tsb

, PARTITION sales\_q3\_2006 VALUES LESS THAN (TO\_DATE('01-OCT-2006','dd-MON-yyyy'))

TABLESPACE tsc

, PARTITION sales\_q4\_2006 VALUES LESS THAN (TO\_DATE('01-JAN-2007','dd-MON-yyyy'))

TABLESPACE tsd

);

4.1.1.2 Creating a Range-Partitioned Table With More Complexity

With attributes and storage parameters, more complexity can be added to the creation of a range-partitioned table.

In [Example 4-2](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-66D19C36-C512-48DE-88B4-AFC6CCACDFEB__CHDCEIJJ), storage parameters and a LOGGING attribute are specified at the table level. These replace the corresponding defaults inherited from the tablespace level for the table itself, and are inherited by the range partitions. However, because there was little business in the first quarter, the storage attributes for partition sales\_q1\_2006 are made smaller. The ENABLE ROW MOVEMENT clause is specified to allow the automatic migration of a row to a new partition if an update to a key value is made that would place the row in a different partition.

Example 4-2 Creating a range-partitioned table with LOGGING and ENABLE ROW MOVEMENT

Copy

CREATE TABLE sales

( prod\_id NUMBER(6)

, cust\_id NUMBER

, time\_id DATE

, channel\_id CHAR(1)

, promo\_id NUMBER(6)

, quantity\_sold NUMBER(3)

, amount\_sold NUMBER(10,2)

)

STORAGE (INITIAL 100K NEXT 50K) LOGGING

PARTITION BY RANGE (time\_id)

( PARTITION sales\_q1\_2006 VALUES LESS THAN (TO\_DATE('01-APR-2006','dd-MON-yyyy'))

TABLESPACE tsa STORAGE (INITIAL 20K NEXT 10K)

, PARTITION sales\_q2\_2006 VALUES LESS THAN (TO\_DATE('01-JUL-2006','dd-MON-yyyy'))

TABLESPACE tsb

, PARTITION sales\_q3\_2006 VALUES LESS THAN (TO\_DATE('01-OCT-2006','dd-MON-yyyy'))

TABLESPACE tsc

, PARTITION sales\_q4\_2006 VALUES LESS THAN (TO\_DATE('01-JAN-2007','dd-MON-yyyy'))

TABLESPACE tsd

)

ENABLE ROW MOVEMENT;

4.1.1.3 Creating a Range-Partitioned Global Index

The rules for creating range-partitioned global indexes are similar to those for creating range-partitioned tables.

[Example 4-3](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-4EEB5C77-4D02-430D-9AC2-10D377A06A15__CHDHGFAG) creates a range-partitioned global index on sale\_month for the tables created in the previous examples. Each index partition is named but is stored in the default tablespace for the index.

Example 4-3 Creating a range-partitioned global index table

Copy

CREATE INDEX amount\_sold\_ix ON sales(amount\_sold)

GLOBAL PARTITION BY RANGE(sale\_month)

( PARTITION p\_100 VALUES LESS THAN (100)

, PARTITION p\_1000 VALUES LESS THAN (1000)

, PARTITION p\_10000 VALUES LESS THAN (10000)

, PARTITION p\_100000 VALUES LESS THAN (100000)

, PARTITION p\_1000000 VALUES LESS THAN (1000000)

, PARTITION p\_greater\_than\_1000000 VALUES LESS THAN (maxvalue)

);

Note:

If your enterprise has databases using different character sets, use caution when partitioning on character columns, because the sort sequence of characters is not identical in all character sets. For more information, refer to [*Oracle Database Globalization Support Guide*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=NLSPG001)

4.1.2 Creating Range-Interval-Partitioned Tables

The INTERVAL clause of the CREATE TABLE statement establishes interval partitioning for the table.

You must specify at least one range partition using the PARTITION clause. The range partitioning key value determines the high value of the range partitions, which is called the transition point, and the database automatically creates interval partitions for data beyond that transition point. The lower boundary of every interval partition is the non-inclusive upper boundary of the previous range or interval partition.

For example, if you create an interval partitioned table with monthly intervals and the transition point is at January 1, 2010, then the lower boundary for the January 2010 interval is January 1, 2010. The lower boundary for the July 2010 interval is July 1, 2010, regardless of whether the June 2010 partition was previously created. Note, however, that using a date where the high or low bound of the partition would be out of the range set for storage causes an error. For example, TO\_DATE('9999-12-01', 'YYYY-MM-DD') causes the high bound to be 10000-01-01, which would not be storable if 10000 is out of the legal range.

The optional STORE IN clause lets you specify one or more tablespaces into which the database stores interval partition data using a round-robin algorithm for subsequently created interval partitions.

For interval partitioning, you can specify only one partitioning key column and the datatype is restricted.

The following example specifies four partitions with varying interval widths. It also specifies that above the transition point of January 1, 2010, partitions are created with an interval width of one month. The high bound of partition p3 represents the transition point. p3 and all partitions below it (p0, p1, and p2 in this example) are in the range section while all partitions above it fall into the interval section.

Copy

CREATE TABLE interval\_sales

( prod\_id NUMBER(6)

, cust\_id NUMBER

, time\_id DATE

, channel\_id CHAR(1)

, promo\_id NUMBER(6)

, quantity\_sold NUMBER(3)

, amount\_sold NUMBER(10,2)

)

PARTITION BY RANGE (time\_id)

INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))

( PARTITION p0 VALUES LESS THAN (TO\_DATE('1-1-2008', 'DD-MM-YYYY')),

PARTITION p1 VALUES LESS THAN (TO\_DATE('1-1-2009', 'DD-MM-YYYY')),

PARTITION p2 VALUES LESS THAN (TO\_DATE('1-7-2009', 'DD-MM-YYYY')),

PARTITION p3 VALUES LESS THAN (TO\_DATE('1-1-2010', 'DD-MM-YYYY')) );

**See Also:**

[*Oracle Database SQL Language Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=SQLRF01402) for restrictions on partitioning keys, the exact syntax of the partitioning clauses for creating and altering partitioned tables and indexes, any restrictions on their use, and specific privileges required for creating and altering tables.

4.1.3 About Creating Hash Partitioned Tables and Global Indexes

The PARTITION BY HASH clause of the CREATE TABLE statement identifies that the table is to be hash partitioned.

The PARTITIONS clause can then be used to specify the number of partitions to create, and optionally, the tablespaces to store them in. Alternatively, you can use PARTITION clauses to name the individual partitions and their tablespaces.

The only attribute you can specify for hash partitions is TABLESPACE. All of the hash partitions of a table must share the same segment attributes (except TABLESPACE), which are inherited from the table level.

The following topics are discussed:

* [Creating a Hash Partitioned Table](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-04E6C6EF-CE27-4DE9-B5B1-D23010E4A356)
* [Creating a Hash Partitioned Global Index](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-A8991A7A-CEC0-4C6D-A886-17E0554BB3C6)

4.1.3.1 Creating a Hash Partitioned Table

The example in this topic shows how to create a hash partitioned table.

The partitioning column is id, four partitions are created and assigned system generated names, and they are placed in four named tablespaces (gear1, gear2, gear3, gear4).

Copy

CREATE TABLE scubagear

(id NUMBER,

name VARCHAR2 (60))

PARTITION BY HASH (id)

PARTITIONS 4

STORE IN (gear1, gear2, gear3, gear4);

In the following example, the number of partitions is specified when creating a hash partitioned table, but system generated names are assigned to them and they are stored in the default tablespace of the table.

Copy

CREATE TABLE departments\_hash (department\_id NUMBER(4) NOT NULL,

department\_name VARCHAR2(30))

PARTITION BY HASH(department\_id) PARTITIONS 16;

In the following example, names of individual partitions, and tablespaces in which they are to reside, are specified. The initial extent size for each hash partition (segment) is also explicitly stated at the table level, and all partitions inherit this attribute.

Copy

CREATE TABLE departments\_hash (department\_id NUMBER(4) NOT NULL,

department\_name VARCHAR2(30))

STORAGE (INITIAL 10K)

PARTITION BY HASH(department\_id)

(PARTITION p1 TABLESPACE ts1, PARTITION p2 TABLESPACE ts2,

PARTITION p3 TABLESPACE ts1, PARTITION p4 TABLESPACE ts3);

If you create a local index for this table, the database constructs the index so that it is equipartitioned with the underlying table. The database also ensures that the index is maintained automatically when maintenance operations are performed on the underlying table. The following is an example of creating a local index on a table:

Copy

CREATE INDEX loc\_dept\_ix ON departments\_hash(department\_id) LOCAL;

You can optionally name the hash partitions and tablespaces into which the local index partitions are to be stored, but if you do not do so, then the database uses the name of the corresponding base partition as the index partition name, and stores the index partition in the same tablespace as the table partition.

**See Also:**

[Specifying Partitioning on Key Columns](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-5C34BF3F-B728-47C7-B656-AE4FA9FFE2C3) for more information about partitioning on key columns

4.1.3.2 Creating a Hash Partitioned Global Index

Hash partitioned global indexes can improve the performance of indexes where a small number of leaf blocks in the index have high contention in multiuser OLTP environments.

Hash partitioned global indexes can also limit the impact of index skew on monotonously increasing column values. Queries involving the equality and IN predicates on the index partitioning key can efficiently use hash partitioned global indexes.

The syntax for creating a hash partitioned global index is similar to that used for a hash partitioned table. For example, the statement in [Example 4-4](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-A8991A7A-CEC0-4C6D-A886-17E0554BB3C6__CHDBDJGC) creates a hash partitioned global index:

Example 4-4 Creating a hash partitioned global index

Copy

CREATE INDEX hgidx ON tab (c1,c2,c3) GLOBAL

PARTITION BY HASH (c1,c2)

(PARTITION p1 TABLESPACE tbs\_1,

PARTITION p2 TABLESPACE tbs\_2,

PARTITION p3 TABLESPACE tbs\_3,

PARTITION p4 TABLESPACE tbs\_4);

4.1.4 About Creating List-Partitioned Tables

The semantics for creating list partitions are very similar to those for creating range partitions.

However, to create list partitions, you specify a PARTITION BY LIST clause in the CREATE TABLE statement, and the PARTITION clauses specify lists of literal values, which are the discrete values of the partitioning columns that qualify rows to be included in the partition. For list partitioning, the partitioning key can be one or multiple column names from the table.

Available only with list partitioning, you can use the keyword DEFAULT to describe the value list for a partition. This identifies a partition that accommodates rows that do not map into any of the other partitions.

As with range partitions, optional subclauses of a PARTITION clause can specify physical and other attributes specific to a partition segment. If not overridden at the partition level, partitions inherit the attributes of their parent table.

The following topics are discussed:

* [Creating a List-Partitioned Table](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-D3E92FD8-5FE4-4CEA-9CB8-CC9277A74429)
* [Creating a List-Partitioned Table With a Default Partition](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-8928C3B0-2F83-4213-B765-EFBBF0372F64)
* [Creating an Automatic List-Partitioned Table](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-12150FFB-48E4-4169-9EBE-64974C1CEF2A)
* [Creating a Multi-column List-Partitioned Table](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-6A4131B1-BBDF-4230-B88C-80BC91A0C861)

4.1.4.1 Creating a List-Partitioned Table

The example in this topic show how to create a list-partitioned table.

[Example 4-5](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-D3E92FD8-5FE4-4CEA-9CB8-CC9277A74429__CHDDEBEB)creates table q1\_sales\_by\_region which is partitioned by regions consisting of groups of US states. A row is mapped to a partition by checking whether the value of the partitioning column for a row matches a value in the value list that describes the partition. For example, the following list describes how some sample rows are inserted into the table.

* (10, 'accounting', 100, 'WA') maps to partition q1\_northwest
* (20, 'R&D', 150, 'OR') maps to partition q1\_northwest
* (30, 'sales', 100, 'FL') maps to partition q1\_southeast
* (40, 'HR', 10, 'TX') maps to partition q1\_southwest
* (50, 'systems engineering', 10, 'CA') does not map to any partition in the table and raises an error

Live SQL:

View and run a related example on Oracle Live SQL at [*Oracle Live SQL: Creating a List Partitioned Table*](https://livesql.oracle.com/apex/livesql/docs/vldbg/partitioning/list-partitioning-example.html).

Example 4-5 Creating a list-partitioned table

Copy

CREATE TABLE q1\_sales\_by\_region

(deptno number,

deptname varchar2(20),

quarterly\_sales number(10, 2),

state varchar2(2))

PARTITION BY LIST (state)

(PARTITION q1\_northwest VALUES ('OR', 'WA'),

PARTITION q1\_southwest VALUES ('AZ', 'UT', 'NM'),

PARTITION q1\_northeast VALUES ('NY', 'VM', 'NJ'),

PARTITION q1\_southeast VALUES ('FL', 'GA'),

PARTITION q1\_northcentral VALUES ('SD', 'WI'),

PARTITION q1\_southcentral VALUES ('OK', 'TX'));

4.1.4.2 Creating a List-Partitioned Table With a Default Partition

Unlike range partitioning, with list partitioning, there is no apparent sense of order between partitions.

You can also specify a **default partition** into which rows that do not map to any other partition are mapped. If a default partition were specified in the preceding example, the state CA would map to that partition.

[Example 4-6](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-8928C3B0-2F83-4213-B765-EFBBF0372F64__CHDDDCFI) creates table sales\_by\_region and partitions it using the list method. The first two PARTITION clauses specify physical attributes, which override the table-level defaults. The remaining PARTITION clauses do not specify attributes and those partitions inherit their physical attributes from table-level defaults. A default partition is also specified.

Example 4-6 Creating a list-partitioned table with a default partition

Copy

CREATE TABLE sales\_by\_region (item# INTEGER, qty INTEGER,

store\_name VARCHAR(30), state\_code VARCHAR(2),

sale\_date DATE)

STORAGE(INITIAL 10K NEXT 20K) TABLESPACE tbs5

PARTITION BY LIST (state\_code)

(

PARTITION region\_east

VALUES ('MA','NY','CT','NH','ME','MD','VA','PA','NJ')

STORAGE (INITIAL 8M)

TABLESPACE tbs8,

PARTITION region\_west

VALUES ('CA','AZ','NM','OR','WA','UT','NV','CO')

NOLOGGING,

PARTITION region\_south

VALUES ('TX','KY','TN','LA','MS','AR','AL','GA'),

PARTITION region\_central

VALUES ('OH','ND','SD','MO','IL','MI','IA'),

PARTITION region\_null

VALUES (NULL),

PARTITION region\_unknown

VALUES (DEFAULT)

);

4.1.4.3 Creating an Automatic List-Partitioned Table

The automatic list partitioning method enables list partition creation on demand.

An auto-list partitioned table is similar to a regular list partitioned table, except that this partitioned table is easier to manage. You can create an auto-list partitioned table using only the partitioning key values that are known. As data is loaded into the table, the database automatically creates a new partition if the loaded partitioning key value does not correspond to any of the existing partitions. Because partitions are automatically created on demand, the auto-list partitioning method is conceptually similar to the existing interval partitioning method.

Automatic list partitioning on data types whose value changes very frequently are less suitable for this method unless you can adjust the data. For example, a SALES\_DATE field with a date value, when the format is not stripped, would increase every second. Each of the SALES\_DATE values, such as 05-22-2016 08:00:00, 05-22-2016 08:00:01, and so on, would generate its own partition. To avoid the creation of a very large number of partitions, you must be aware of the data that would be entered and adjust accordingly. As an example, you can truncate the SALES\_DATE date value to a day or some other time period, depending on the number of partitions required.

The CREATE and ALTER TABLE SQL statements are updated with an additional clause to specify AUTOMATIC or MANUAL list partitioning. An automatic list-partitioned table must have at least one partition when created. Because new partitions are automatically created for new, and unknown, partition key values, an automatic list partition cannot have a DEFAULT partition.

[Example 4-7](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-12150FFB-48E4-4169-9EBE-64974C1CEF2A__GUID-E760B49A-61B0-4A02-8020-6A26B8540915) is an example of the CREATE TABLE statement using the AUTOMATIC keyword for auto-list partitioning on the sales\_state field. The SQL statement creates at least one partition as required.

Example 4-7 Creating an automatic list partitioned table

Copy

CREATE TABLE sales\_auto\_list

(

salesman\_id NUMBER(5),

salesman\_name VARCHAR2(30),

sales\_state VARCHAR2(20),

sales\_amount NUMBER(10),

sales\_date DATE

)

PARTITION BY LIST (sales\_state) AUTOMATIC

(PARTITION P\_CAL VALUES ('CALIFORNIA')

);

You can check the AUTOLIST column of the \*\_PART\_TABLES view to determine whether a table is automatic list-partitioned.

**See Also:**

[*Oracle Database Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=REFRN-GUID-2E7FFA31-6241-474E-BA88-5EB5B8F69245) for information about \*\_PART\_TABLES view

4.1.4.4 Creating a Multi-column List-Partitioned Table

Multi-column list partitioning enables you to partition a table based on list values of multiple columns.

Similar to single-column list partitioning, individual partitions can contain sets containing lists of values.

Multi-column list partitioning is supported on a table using the PARTITION BY LIST clause on multiple columns of a table. For example:

Copy

PARTITION BY LIST (*column1*,*column2*)

A multi-column list-partitioned table can only have one DEFAULT partition.

The following is an example of the CREATE TABLE statement using multi-column partitioning on the state and channel columns.

Example 4-8 Creating a multicolumn list-partitioned table

Copy

CREATE TABLE sales\_by\_region\_and\_channel

(deptno NUMBER,

deptname VARCHAR2(20),

quarterly\_sales NUMBER(10,2),

state VARCHAR2(2),

channel VARCHAR2(1)

)

PARTITION BY LIST (state, channel)

(

PARTITION q1\_northwest\_direct VALUES (('OR','D'), ('WA','D')),

PARTITION q1\_northwest\_indirect VALUES (('OR','I'), ('WA','I')),

PARTITION q1\_southwest\_direct VALUES (('AZ','D'),('UT','D'),('NM','D')),

PARTITION q1\_ca\_direct VALUES ('CA','D'),

PARTITION rest VALUES (DEFAULT)

);

4.1.5 Creating Reference-Partitioned Tables

To create a reference-partitioned table, you specify a PARTITION BY REFERENCE clause in the CREATE TABLE statement.

The PARTITION BY REFERENCE clause specifies the name of a referential constraint and this constraint becomes the partitioning referential constraint that is used as the basis for reference partitioning in the table. The referential constraint must be enabled and enforced.

As with other partitioned tables, you can specify object-level default attributes, and you can optionally specify partition descriptors that override the object-level defaults on a per-partition basis.

[Example 4-9](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-00923EB3-05F6-41F7-8437-E42FC9BD9571__CHDHJCFD) creates a parent table orders which is range-partitioned on order\_date. The reference-partitioned child table order\_items is created with four partitions, Q1\_2005, Q2\_2005, Q3\_2005, and Q4\_2005, where each partition contains the order\_items rows corresponding to orders in the respective parent partition.

If partition descriptors are provided, then the number of partitions described must exactly equal the number of partitions or subpartitions in the referenced table. If the parent table is a composite partitioned table, then the table has one partition for each subpartition of its parent; otherwise the table has one partition for each partition of its parent.

Partition bounds cannot be specified for the partitions of a reference-partitioned table.

The partitions of a reference-partitioned table can be named. If a partition is not explicitly named, then it inherits its name from the corresponding partition in the parent table, unless this inherited name conflicts with an existing explicit name. In this case, the partition has a system-generated name.

Partitions of a reference-partitioned table collocate with the corresponding partition of the parent table, if no explicit tablespace is specified for the reference-partitioned table's partition.

Example 4-9 Creating reference-partitioned tables

Copy

CREATE TABLE orders

( order\_id NUMBER(12),

order\_date DATE,

order\_mode VARCHAR2(8),

customer\_id NUMBER(6),

order\_status NUMBER(2),

order\_total NUMBER(8,2),

sales\_rep\_id NUMBER(6),

promotion\_id NUMBER(6),

CONSTRAINT orders\_pk PRIMARY KEY(order\_id)

)

PARTITION BY RANGE(order\_date)

( PARTITION Q1\_2005 VALUES LESS THAN (TO\_DATE('01-APR-2005','DD-MON-YYYY')),

PARTITION Q2\_2005 VALUES LESS THAN (TO\_DATE('01-JUL-2005','DD-MON-YYYY')),

PARTITION Q3\_2005 VALUES LESS THAN (TO\_DATE('01-OCT-2005','DD-MON-YYYY')),

PARTITION Q4\_2005 VALUES LESS THAN (TO\_DATE('01-JAN-2006','DD-MON-YYYY'))

);

CREATE TABLE order\_items

( order\_id NUMBER(12) NOT NULL,

line\_item\_id NUMBER(3) NOT NULL,

product\_id NUMBER(6) NOT NULL,

unit\_price NUMBER(8,2),

quantity NUMBER(8),

CONSTRAINT order\_items\_fk

FOREIGN KEY(order\_id) REFERENCES orders(order\_id)

)

PARTITION BY REFERENCE(order\_items\_fk);

4.1.6 Creating Interval-Reference Partitioned Tables

You can use interval partitioned tables as parent tables for reference partitioning. Partitions in a reference-partitioned table corresponding to interval partitions in the parent table are created when inserting records into the reference partitioned table.

When creating an interval partition in a child table, the partition name is inherited from the associated parent table fragment. If the child table has a table-level default tablespace, then it is used as tablespace for the new interval partition; otherwise, the tablespace is inherited from the parent table fragment.

The SQL ALTER TABLE SET INTERVAL statement is not allowed for reference-partitioned tables, but can be run on tables that have reference-partitioned children. In particular, ALTER TABLE SET INTERVAL removes the interval property from the targeted table and converts any interval-reference children to ordinary reference-partitioned tables. Also, the SQL ALTER TABLE SET STORE IN statement is not allowed for reference-partitioned tables, but can be run on tables that have reference-partitioned children.

Operations that transform interval partitions to conventional partitions in the parent table, such as ALTER TABLE SPLIT PARTITION on an interval partition, construct the corresponding transformation in the child table, creating partitions in the child table as necessary.

For example, the following SQL statements provides three interval partitions in the parent table and none in the child table:

Copy

CREATE TABLE par(pk INT CONSTRAINT par\_pk PRIMARY KEY, i INT)

PARTITION BY RANGE(i) INTERVAL (10)

(PARTITION p1 VALUES LESS THAN (10));

CREATE TABLE chi(fk INT NOT NULL, i INT,

CONSTRAINT chi\_fk FOREIGN KEY(fk) REFERENCES par(pk))

PARTITION BY REFERENCE(chi\_fk);

INSERT INTO par VALUES(15, 15);

INSERT INTO par VALUES(25, 25);

INSERT INTO par VALUES(35, 35);

You can display information about partitions with the USER\_TAB\_PARTITIONS view:

Copy

SELECT table\_name, partition\_position, high\_value, interval

FROM USER\_TAB\_PARTITIONS WHERE table\_name IN ('PAR', 'CHI')

ORDER BY 1, 2;

TABLE\_NAME PARTITION\_POSITION HIGH\_VALUE INT

---------------- ------------------ ---------- ---

CHI 1 NO

PAR 1 10 NO

PAR 2 20 YES

PAR 3 30 YES

PAR 4 40 YES

If the interval partition is split in the parent table, then some interval partitions are converted to conventional partitions for all tables in the hierarchy, creating conventional partitions in the child table in the process. For example:

Copy

ALTER TABLE par SPLIT PARTITION FOR (25) AT (25)

INTO (partition x, partition y);

SELECT table\_name, partition\_position, high\_value, interval

FROM USER\_TAB\_PARTITIONS WHERE table\_name IN ('PAR', 'CHI')

ORDER BY 1, 2;

TABLE\_NAME PARTITION\_POSITION HIGH\_VALUE INT

---------------- ------------------ ---------- ---

CHI 1 NO

CHI 2 NO

CHI 3 NO

CHI 4 NO

PAR 1 10 NO

PAR 2 20 NO

PAR 3 25 NO

PAR 4 30 NO

PAR 5 40 YES

Interval-reference functionality requires that the database compatibility level (Oracle Database COMPATIBLE initialization parameter setting) be set to greater than or equal to 12.0.0.0.

4.1.7 Creating a Table Using In-Memory Column Store With Partitioning

You can create a partitioned table using the In-Memory Column Store with the INMEMORY clause.

The following example specifies that individual partitions are loaded into the In-Memory Column Store using the INMEMORY clause with the partitioning clauses of the CREATE TABLE SQL statements.

Copy

CREATE TABLE list\_customers

( customer\_id NUMBER(6)

, cust\_first\_name VARCHAR2(20)

, cust\_last\_name VARCHAR2(20)

, cust\_address CUST\_ADDRESS\_TYP

, nls\_territory VARCHAR2(30)

, cust\_email VARCHAR2(40))

PARTITION BY LIST (nls\_territory) (

PARTITION asia VALUES ('CHINA', 'THAILAND')

INMEMORY MEMCOMPRESS FOR CAPACITY HIGH,

PARTITION europe VALUES ('GERMANY', 'ITALY', 'SWITZERLAND')

INMEMORY MEMCOMPRESS FOR CAPACITY LOW,

PARTITION west VALUES ('AMERICA')

INMEMORY MEMCOMPRESS FOR CAPACITY LOW,

PARTITION east VALUES ('INDIA')

INMEMORY MEMCOMPRESS FOR CAPACITY HIGH,

PARTITION rest VALUES (DEFAULT);

**See Also:**

* [*Oracle Database In-Memory Guide*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=INMEM-GUID-BFA53515-7643-41E5-A296-654AB4A9F9E7) for overview information about In-Memory Column Store
* [*Oracle Database In-Memory Guide*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=INMEM-GUID-C5F856BF-70E3-41C6-A1BA-1E94D7D230B8) for information about enabling objects for population in the In-Memory Column Store and ADO support
* [*Oracle Database SQL Language Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=SQLRF01402) for information about SQL syntax related to In-Memory Column Store

4.1.8 Creating a Table with Read-Only Partitions or Subpartitions

You can set tables, partitions, and subpartitions to read-only status to protect data from unintentional DML operations by any user or trigger.

Any attempt to update data in a partition or subpartition that is set to read only results in an error, while updating data in partitions or subpartitions that are set to read write succeeds.

The CREATE TABLE and ALTER TABLE SQL statements provide a read-only clause for partitions and subpartitions. The values of the read-only clause can be READ ONLY or READ WRITE. READ WRITE is the default value. A higher level setting of the read-only clause is applied to partitions and subpartitions unless the read-only clause has been explicitly set for a partition or subpartition.

The following is an example of a creating a composite range-list partitioned table with both read-only and read-write status. The orders\_read\_write\_only is explicitly specified as READ WRITE, so the default attribute of the table is read write. The default attribute of partition order\_p1 is specified as read only, so the subpartitions ord\_p1\_northwest and order\_p1\_southwest inherit read only status from partition order\_p1. Subpartitions ord\_p2\_southwest and order\_p3\_northwest are explicitly specified as read only, overriding the default read write status.

Example 4-10 Creating a table with read-only and read-write partitions

Copy

CREATE TABLE orders\_read\_write\_only (

order\_id NUMBER (12),

order\_date DATE CONSTRAINT order\_date\_nn NOT NULL,

state VARCHAR2(2)

) READ WRITE

PARTITION BY RANGE (order\_date)

SUBPARTITION BY LIST (state)

( PARTITION order\_p1 VALUES LESS THAN (TO\_DATE ('01-DEC-2015','DD-MON-YYYY')) READ ONLY

( SUBPARTITION order\_p1\_northwest VALUES ('OR', 'WA'),

SUBPARTITION order\_p1\_southwest VALUES ('AZ', 'UT', 'NM')

),

PARTITION order\_p2 VALUES LESS THAN (TO\_DATE ('01-MAR-2016','DD-MON-YYYY'))

( SUBPARTITION order\_p2\_northwest VALUES ('OR', 'WA'),

SUBPARTITION order\_p2\_southwest VALUES ('AZ', 'UT', 'NM') READ ONLY

),

PARTITION order\_p3 VALUES LESS THAN (TO\_DATE ('01-JUL-2016','DD-MON-YYYY'))

(

SUBPARTITION order\_p3\_northwest VALUES ('OR', 'WA') READ ONLY,

SUBPARTITION order\_p3\_southwest VALUES ('AZ', 'UT', 'NM')

)

);

You can check the read-only status with the DEF\_READ\_ONLY column of the \*\_PART\_TABLES view, the READ\_ONLY column of the \*\_TAB\_PARTITIONS view, and the READ\_ONLY column of the \*\_TAB\_SUBPARTITIONS view. Note that only physical segments, partitions for single-level partitioning and subpartitions for composite partitioning, have a status. All other levels are logical and only have a default status.

Copy

SQL> SELECT PARTITION\_NAME, READ\_ONLY FROM USER\_TAB\_PARTITIONS WHERE TABLE\_NAME ='ORDERS\_READ\_WRITE\_ONLY';

PARTITION\_NAME READ

------------------------------- ----

ORDER\_P1 YES

ORDER\_P2 NONE

ORDER\_P3 NONE

SQL> SELECT PARTITION\_NAME, SUBPARTITION\_NAME, READ\_ONLY FROM USER\_TAB\_SUBPARTITIONS WHERE TABLE\_NAME ='ORDERS\_READ\_WRITE\_ONLY';

PARTITION\_NAME SUBPARTITION\_NAME REA

------------------------------ ----------------------------- ---

ORDER\_P1 ORDER\_P1\_NORTHWEST YES

ORDER\_P1 ORDER\_P1\_SOUTHWEST YES

ORDER\_P2 ORDER\_P2\_NORTHWEST NO

ORDER\_P2 ORDER\_P2\_SOUTHWEST YES

ORDER\_P3 ORDER\_P3\_NORTHWEST YES

ORDER\_P3 ORDER\_P3\_SOUTHWEST NO

**See Also:**

[*Oracle Database Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=REFRN-GUID-22FAAA6B-2181-47AE-8A10-6CF968B52CA4) for information about \*\_PART\_TABLES, \*\_TAB\_PARTITIONS, and \*\_TAB\_SUBPARTITIONS views

4.1.9 Creating a Partitioned External Table

You can create partitions for an external table.

The organization external clause identifies the table as external table, followed by the specification and access parameters of the external table. While parameters, such as the default directory; can be overridden on a partition or subpartition level, the external table type and its access parameters are table-level attributes and applicable to all partitions or subpartitions.

The table created in [Example 4-11](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-09CFA3CD-66B7-4CDF-A086-6958D113BDD1__GUID-385BCC5F-77CA-4579-8C7C-0E964DB65AB6) has three partitions for external data accessed from different locations. Partition p1 stores customer data for California, located in the default directory of the table. Partition p2 points to a file storing data for Washington. Partition p3 does not have a file descriptor and is empty.

Example 4-11 Creating a Partitioned External Table

Copy

CREATE TABLE sales (loc\_id number, prod\_id number, cust\_id number, amount\_sold number, quantity\_sold number)

ORGANIZATION EXTERNAL

(TYPE oracle\_loader

DEFAULT DIRECTORY load\_d1

ACCESS PARAMETERS

( RECORDS DELIMITED BY NEWLINE CHARACTERSET US7ASCII

NOBADFILE

LOGFILE log\_dir:'sales.log'

FIELDS TERMINATED BY ","

)

)

REJECT LIMIT UNLIMITED

PARTITION BY RANGE (loc\_id)

(PARTITION p1 VALUES LESS THAN (1000) LOCATION ('california.txt'),

PARTITION p2 VALUES LESS THAN (2000) DEFAULT DIRECTORY load\_d2 LOCATION ('washington.txt'),

PARTITION p3 VALUES LESS THAN (3000))

;

**See Also:**

[*Oracle Database Administrator’s Guide*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=ADMIN-GUID-2A801016-0399-4925-AD1B-A02683E81B59) for information about partitioning external tables

4.1.10 Specifying Partitioning on Key Columns

For range-partitioned and hash partitioned tables, you can specify up to 16 partitioning key columns.

Use multicolumn partitioning when the partitioning key is composed of several columns and subsequent columns define a higher granularity than the preceding ones. The most common scenario is a decomposed DATE or TIMESTAMP key, consisting of separated columns, for year, month, and day.

In evaluating multicolumn partitioning keys, the database uses the second value only if the first value cannot uniquely identify a single target partition, and uses the third value only if the first and second do not determine the correct partition, and so forth. A value cannot determine the correct partition only when a partition bound exactly matches that value and the same bound is defined for the next partition. The *n*th column is investigated only when all previous (n-1) values of the multicolumn key exactly match the (n-1) bounds of a partition. A second column, for example, is evaluated only if the first column exactly matches the partition boundary value. If all column values exactly match all of the bound values for a partition, then the database determines that the row does not fit in this partition and considers the next partition for a match.

For nondeterministic boundary definitions (successive partitions with identical values for at least one column), the partition boundary value becomes an inclusive value, representing a "less than or equal to" boundary. This is in contrast to deterministic boundaries, where the values are always regarded as "less than" boundaries.

The following topics are discussed:

* [Creating a Multicolumn Range-Partitioned Table By Date](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-814DE2FF-7147-42DE-9CAF-5E019BAB8C18)
* [Creating a Multicolumn Range-Partitioned Table to Enforce Equal-Sized Partitions](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-2E03ED66-312F-422A-9352-2128B34DEE8E)

4.1.10.1 Creating a Multicolumn Range-Partitioned Table By Date

The example in this topic shows how to create a multicolumn range-partitioned table by date.

[Example 4-12](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-814DE2FF-7147-42DE-9CAF-5E019BAB8C18__CHDEJJFJ) illustrates the column evaluation for a multicolumn range-partitioned table, storing the actual DATE information in three separate columns: year, month, and day. The partitioning granularity is a calendar quarter. The partitioned table being evaluated is created as follows:

The year value for 12-DEC-2000 satisfied the first partition, before2001, so no further evaluation is needed:

Copy

SELECT \* FROM sales\_demo PARTITION(before2001);

YEAR MONTH DAY AMOUNT\_SOLD

---------- ---------- ---------- -----------

2000 12 12 1000

The information for 17-MAR-2001 is stored in partition q1\_2001. The first partitioning key column, year, does not by itself determine the correct partition, so the second partitioning key column, month, must be evaluated.

Copy

SELECT \* FROM sales\_demo PARTITION(q1\_2001);

YEAR MONTH DAY AMOUNT\_SOLD

---------- ---------- ---------- -----------

2001 3 17 2000

Following the same determination rule as for the previous record, the second column, month, determines partition q4\_2001 as correct partition for 1-NOV-2001:

Copy

SELECT \* FROM sales\_demo PARTITION(q4\_2001);

YEAR MONTH DAY AMOUNT\_SOLD

---------- ---------- ---------- -----------

2001 11 1 5000

The partition for 01-JAN-2002 is determined by evaluating only the year column, which indicates the future partition:

Copy

SELECT \* FROM sales\_demo PARTITION(future);

YEAR MONTH DAY AMOUNT\_SOLD

---------- ---------- ---------- -----------

2002 1 1 4000

If the database encounters MAXVALUE in a partitioning key column, then all other values of subsequent columns become irrelevant. That is, a definition of partition future in the preceding example, having a bound of (MAXVALUE,0) is equivalent to a bound of (MAXVALUE,100) or a bound of (MAXVALUE,MAXVALUE).

Example 4-12 Creating a multicolumn range-partitioned table

Copy

CREATE TABLE sales\_demo (

year NUMBER,

month NUMBER,

day NUMBER,

amount\_sold NUMBER)

PARTITION BY RANGE (year,month)

(PARTITION before2001 VALUES LESS THAN (2001,1),

PARTITION q1\_2001 VALUES LESS THAN (2001,4),

PARTITION q2\_2001 VALUES LESS THAN (2001,7),

PARTITION q3\_2001 VALUES LESS THAN (2001,10),

PARTITION q4\_2001 VALUES LESS THAN (2002,1),

PARTITION future VALUES LESS THAN (MAXVALUE,0));

REM 12-DEC-2000

INSERT INTO sales\_demo VALUES(2000,12,12, 1000);

REM 17-MAR-2001

INSERT INTO sales\_demo VALUES(2001,3,17, 2000);

REM 1-NOV-2001

INSERT INTO sales\_demo VALUES(2001,11,1, 5000);

REM 1-JAN-2002

INSERT INTO sales\_demo VALUES(2002,1,1, 4000);

4.1.10.2 Creating a Multicolumn Range-Partitioned Table to Enforce Equal-Sized Partitions

The example in this topic shows how to create a multicolumn range-partitioned table to enforce equal-sized partitions.

The following example illustrates the use of a multicolumn partitioned approach for table supplier\_parts, storing the information about which suppliers deliver which parts. To distribute the data in equal-sized partitions, it is not sufficient to partition the table based on the supplier\_id, because some suppliers might provide hundreds of thousands of parts, while others provide only a few specialty parts. Instead, you partition the table on (supplier\_id, partnum) to manually enforce equal-sized partitions.

Every row with supplier\_id < 10 is stored in partition p1, regardless of the partnum value. The column partnum is evaluated only if supplier\_id =10, and the corresponding rows are inserted into partition p1, p2, or even into p3 when partnum >=200. To achieve equal-sized partitions for ranges of supplier\_parts, you could choose a composite range-hash partitioned table, range partitioned by supplier\_id, hash subpartitioned by partnum.

Defining the partition boundaries for multicolumn partitioned tables must obey some rules. For example, consider a table that is range partitioned on three columns a, b, and c. The individual partitions have range values represented as follows:

Copy

P0(a0, b0, c0)

P1(a1, b1, c1)

P2(a2, b2, c2)

...

Pn(an, bn, cn)

The range values you provide for each partition must follow these rules:

* a0 must be less than or equal to a1, and a1 must be less than or equal to a2, and so on.
* If a0=a1, then b0 must be less than or equal to b1. If a0 < a1, then b0 and b1 can have any values. If a0=a1 and b0=b1, then c0 must be less than or equal to c1. If b0<b1, then c0 and c1 can have any values, and so on.
* If a1=a2, then b1 must be less than or equal to b2. If a1<a2, then b1 and b2 can have any values. If a1=a2 and b1=b2, then c1 must be less than or equal to c2. If b1<b2, then c1 and c2 can have any values, and so on.

Copy

CREATE TABLE supplier\_parts (

supplier\_id NUMBER,

partnum NUMBER,

price NUMBER)

PARTITION BY RANGE (supplier\_id, partnum)

(PARTITION p1 VALUES LESS THAN (10,100),

PARTITION p2 VALUES LESS THAN (10,200),

PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE));

The following three records are inserted into the table:

Copy

INSERT INTO supplier\_parts VALUES (5,5, 1000);

INSERT INTO supplier\_parts VALUES (5,150, 1000);

INSERT INTO supplier\_parts VALUES (10,100, 1000);

The first two records are inserted into partition p1, uniquely identified by supplier\_id. However, the third record is inserted into partition p2; it matches all range boundary values of partition p1 exactly and the database therefore considers the following partition for a match. The value of partnum satisfies the criteria < 200, so it is inserted into partition p2.

Copy

SELECT \* FROM supplier\_parts PARTITION (p1);

SUPPLIER\_ID PARTNUM PRICE

----------- ---------- ----------

5 5 1000

5 150 1000

SELECT \* FROM supplier\_parts PARTITION (p2);

SUPPLIER\_ID PARTNUM PRICE

----------- ---------- ----------

10 100 1000

4.1.11 Using Virtual Column-Based Partitioning

With partitioning, a virtual column can be used as any regular column.

All partition methods are supported when using virtual columns, including interval partitioning and all different combinations of composite partitioning. A virtual column used as the partitioning column cannot use calls to a PL/SQL function.

The following example shows the sales table partitioned by range-range using a virtual column for the subpartitioning key. The virtual column calculates the total value of a sale by multiplying amount\_sold and quantity\_sold. As the example shows, row movement is also supported with virtual columns. If row movement is enabled, then a row migrates from one partition to another partition if the virtual column evaluates to a value that belongs to another partition.

Copy

CREATE TABLE sales

( prod\_id NUMBER(6) NOT NULL

, cust\_id NUMBER NOT NULL

, time\_id DATE NOT NULL

, channel\_id CHAR(1) NOT NULL

, promo\_id NUMBER(6) NOT NULL

, quantity\_sold NUMBER(3) NOT NULL

, amount\_sold NUMBER(10,2) NOT NULL

, total\_amount AS (quantity\_sold \* amount\_sold)

)

PARTITION BY RANGE (time\_id) INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))

SUBPARTITION BY RANGE(total\_amount)

SUBPARTITION TEMPLATE

( SUBPARTITION p\_small VALUES LESS THAN (1000)

, SUBPARTITION p\_medium VALUES LESS THAN (5000)

, SUBPARTITION p\_large VALUES LESS THAN (10000)

, SUBPARTITION p\_extreme VALUES LESS THAN (MAXVALUE)

)

(PARTITION sales\_before\_2007 VALUES LESS THAN

(TO\_DATE('01-JAN-2007','dd-MON-yyyy'))

)

ENABLE ROW MOVEMENT

PARALLEL NOLOGGING;

**See Also:**

[*Oracle Database SQL Language Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=SQLRF0025) for the syntax on how to create a virtual column

4.1.12 Using Table Compression with Partitioned Tables

For heap-organized partitioned tables, you can compress some or all partitions using table compression.

The compression attribute can be declared for a tablespace, a table, or a partition of a table. Whenever the compress attribute is not specified, it is inherited like any other storage attribute.

[Example 4-13](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-7D211E4B-FFFD-4146-9010-20ED1232FF2C__CHDCCFGE) creates a range-partitioned table with one compressed partition costs\_old. The compression attribute for the table and all other partitions is inherited from the tablespace level.

Example 4-13 Creating a range-partitioned table with a compressed partition

Copy

CREATE TABLE costs\_demo (

prod\_id NUMBER(6), time\_id DATE,

unit\_cost NUMBER(10,2), unit\_price NUMBER(10,2))

PARTITION BY RANGE (time\_id)

(PARTITION costs\_old

VALUES LESS THAN (TO\_DATE('01-JAN-2003', 'DD-MON-YYYY')) COMPRESS,

PARTITION costs\_q1\_2003

VALUES LESS THAN (TO\_DATE('01-APR-2003', 'DD-MON-YYYY')),

PARTITION costs\_q2\_2003

VALUES LESS THAN (TO\_DATE('01-JUN-2003', 'DD-MON-YYYY')),

PARTITION costs\_recent VALUES LESS THAN (MAXVALUE));

4.1.13 Using Key Compression with Partitioned Indexes

You can compress some or all partitions of a B-tree index using key compression.

Key compression is applicable only to B-tree indexes. Bitmap indexes are stored in a compressed manner by default. An index using key compression eliminates repeated occurrences of key column prefix values, thus saving space and I/O.

The following example creates a local partitioned index with all partitions except the most recent one compressed:

Copy

CREATE INDEX i\_cost1 ON costs\_demo (prod\_id) COMPRESS LOCAL

(PARTITION costs\_old, PARTITION costs\_q1\_2003,

PARTITION costs\_q2\_2003, PARTITION costs\_recent NOCOMPRESS);

You cannot specify COMPRESS (or NOCOMPRESS) explicitly for an index subpartition. All index subpartitions of a given partition inherit the key compression setting from the parent partition.

To modify the key compression attribute for all subpartitions of a given partition, you must first issue an ALTER INDEX...MODIFY PARTITION statement and then rebuild all subpartitions. The MODIFY PARTITION clause marks all index subpartitions as UNUSABLE.

4.1.14 Specifying Partitioning with Segments

Partitioning with segments is introduced in this topic.

These topics discuss the functionality when using partitioning with segments.

* [Deferred Segment Creation for Partitioning](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-DF75897A-DF8D-4AF3-AECF-748D60A94949)
* [Truncating Segments That Are Empty](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-4EBDF098-A3DB-44DF-8B0C-CE24246EBE64)
* [Maintenance Procedures for Segment Creation on Demand](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-FCAD2073-5AF3-42A6-8485-A5D4F9A814C3)

4.1.14.1 Deferred Segment Creation for Partitioning

You can defer the creation of segments when creating a partitioned table until the first row is inserted into a partition.

When the first row is inserted, segments are created for the base table partition, LOB columns, all global indexes, and local index partitions. Deferred segment creation can be controlled by the following:

* Setting the DEFERRED\_SEGMENT\_CREATION initialization parameter to TRUE or FALSE in the initialization parameter file.
* Setting the initialization parameter DEFERRED\_SEGMENT\_CREATION to TRUE or FALSE with the ALTER SESSION or ALTER SYSTEM SQL statements.
* Specifying the keywords SEGMENT CREATION IMMEDIATE or SEGMENT CREATION DEFERRED with the partition clause when issuing the CREATE TABLE SQL statement.

You can force the creation of segments for an existing created partition with the ALTER TABLE MODIFY PARTITION ALLOCATE EXTENT SQL statement. This statement allocates one extent more than the initial number of extents specified during the CREATE TABLE.

Serializable transactions are not supported with deferred segment creation. Inserting data into an empty table with no segment created, or into a partition of an interval partitioned table that does not have a segment yet, can cause an error.

**See Also:**

* [*Oracle Database Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=REFRN10307) for more information about the DEFERRED\_SEGMENT\_CREATION initialization parameter
* [*Oracle Database SQL Language Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=SQLRF00901) for more information about the ALTER SESSION and ALTER SYSTEM SQL statements
* [*Oracle Database SQL Language Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=SQLRF01402) for more information about the keywords SEGMENT CREATION IMMEDIATE and SEGMENT CREATION DEFERRED of the CREATE TABLE SQL statement

4.1.14.2 Truncating Segments That Are Empty

You can drop empty segments in tables and table fragments with the DBMS\_SPACE\_ADMIN.DROP\_EMPTY\_SEGMENTS procedure.

In addition, if a partition or subpartition has a segment, then the truncate feature drops the segment if the DROP ALL STORAGE clause is specified with the ALTER TABLE TRUNCATE PARTITION SQL statement.

**See Also:**

* [*Oracle Database PL/SQL Packages and Types Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=ARPLS057) for more information about the DBMS\_SPACE\_ADMIN package
* [*Oracle Database SQL Language Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=SQLRF01001) for more information about the DROP ALL STORAGE clause of ALTER TABLE

4.1.14.3 Maintenance Procedures for Segment Creation on Demand

You can use the MATERIALIZE\_DEFERRED\_SEGMENTS procedure in the DBMS\_SPACE\_ADMIN package to create segments for tables and dependent objects for tables with the deferred segment property.

You can also force the creation of segments for an existing created table and table fragment with the DBMS\_SPACE\_ADMIN.MATERIALIZE\_DEFERRED\_SEGMENTS procedure. The MATERIALIZE\_DEFERRED\_SEGMENTS procedure differs from the ALTER TABLE MODIFY PARTITION ALLOCATE EXTENT SQL statement because it does not allocate one additional extent for the table or table fragment.

**See Also:**

* [*Oracle Database PL/SQL Packages and Types Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=ARPLS057) for more information about the DBMS\_SPACE\_ADMIN package

4.1.15 Specifying Partitioning When Creating Index-Organized Tables

For index-organized tables, you can use the range, list, or hash partitioning method.

The semantics for creating partitioned index-organized tables are similar to that for regular tables with these differences:

* When you create the table, you specify the ORGANIZATION INDEX clause, and INCLUDING and OVERFLOW clauses as necessary.
* The PARTITION clause can have OVERFLOW subclauses that allow you to specify attributes of the overflow segments at the partition level.

Specifying an OVERFLOW clause results in the overflow data segments themselves being equipartitioned with the primary key index segments. Thus, for partitioned index-organized tables with overflow, each partition has an index segment and an overflow data segment.

For index-organized tables, the set of partitioning columns must be a subset of the primary key columns. Because rows of an index-organized table are stored in the primary key index for the table, the partitioning criterion affects the availability. By choosing the partitioning key to be a subset of the primary key, an insert operation must only verify uniqueness of the primary key in a single partition, thereby maintaining partition independence.

Support for secondary indexes on index-organized tables is similar to the support for regular tables. Because of the logical nature of the secondary indexes, global indexes on index-organized tables remain usable for certain operations where they would be marked UNUSABLE for regular tables.

The following topics are discussed:

* [Creating Range-Partitioned Index-Organized Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-A944913C-82F4-4A66-A208-E696C374AB1B)
* [Creating Hash Partitioned Index-Organized Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-ACA2C4B5-6C3D-4503-BFF1-927F89831CD3)
* [Creating List-Partitioned Index-Organized Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-77B812A7-6427-429A-8AA7-099EB631FBF8)

**See Also:**

* [Maintenance Operations for Partitioned Tables and Indexes](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/maintenance-partition-tables-indexes.html#GUID-A226B597-BCF1-49E2-8284-739A99D3F9ED) for information about maintenance operations on index-organized tables
* [*Oracle Database Administrator’s Guide*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=ADMIN01506) for more information about managing index-organized tables
* [*Oracle Database Concepts*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=CNCPT911) for more information about index-organized tables

4.1.15.1 Creating Range-Partitioned Index-Organized Tables

You can partition index-organized tables, and their secondary indexes, by the range method.

In [Example 4-14](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-A944913C-82F4-4A66-A208-E696C374AB1B__CHDJCEAJ), a range-partitioned index-organized table sales is created. The INCLUDING clause specifies that all columns after week\_no are to be stored in an overflow segment. There is one overflow segment for each partition, all stored in the same tablespace (overflow\_here). Optionally, OVERFLOW TABLESPACE could be specified at the individual partition level, in which case some or all of the overflow segments could have separate TABLESPACE attributes.

Example 4-14 Creating a range-partitioned index-organized table

Copy

CREATE TABLE sales(acct\_no NUMBER(5),

acct\_name CHAR(30),

amount\_of\_sale NUMBER(6),

week\_no INTEGER,

sale\_details VARCHAR2(1000),

PRIMARY KEY (acct\_no, acct\_name, week\_no))

ORGANIZATION INDEX

INCLUDING week\_no

OVERFLOW TABLESPACE overflow\_here

PARTITION BY RANGE (week\_no)

(PARTITION VALUES LESS THAN (5)

TABLESPACE ts1,

PARTITION VALUES LESS THAN (9)

TABLESPACE ts2 OVERFLOW TABLESPACE overflow\_ts2,

...

PARTITION VALUES LESS THAN (MAXVALUE)

TABLESPACE ts13);

4.1.15.2 Creating Hash Partitioned Index-Organized Tables

Another option for partitioning index-organized tables is to use the hash method.

In [Example 4-15](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-ACA2C4B5-6C3D-4503-BFF1-927F89831CD3__CHDDCFHH), the sales index-organized table is partitioned by the hash method.

Note:

A well-designed hash function is intended to distribute rows in a well-balanced fashion among the partitions. Therefore, updating the primary key column(s) of a row is very likely to move that row to a different partition. Oracle recommends that you explicitly specify the ENABLE ROW MOVEMENT clause when creating a hash partitioned index-organized table with a changeable partitioning key. The default is that ENABLE ROW MOVEMENT is disabled.

Example 4-15 Creating a hash partitioned index-organized table

Copy

CREATE TABLE sales(acct\_no NUMBER(5),

acct\_name CHAR(30),

amount\_of\_sale NUMBER(6),

week\_no INTEGER,

sale\_details VARCHAR2(1000),

PRIMARY KEY (acct\_no, acct\_name, week\_no))

ORGANIZATION INDEX

INCLUDING week\_no

OVERFLOW

PARTITION BY HASH (week\_no)

PARTITIONS 16

STORE IN (ts1, ts2, ts3, ts4)

OVERFLOW STORE IN (ts3, ts6, ts9);

4.1.15.3 Creating List-Partitioned Index-Organized Tables

The other option for partitioning index-organized tables is to use the list method.

In [Example 4-16](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-77B812A7-6427-429A-8AA7-099EB631FBF8__CHDEAGIJ), the sales index-organized table is partitioned by the list method.

Example 4-16 Creating a list-partitioned index-organized table

Copy

CREATE TABLE sales(acct\_no NUMBER(5),

acct\_name CHAR(30),

amount\_of\_sale NUMBER(6),

week\_no INTEGER,

sale\_details VARCHAR2(1000),

PRIMARY KEY (acct\_no, acct\_name, week\_no))

ORGANIZATION INDEX

INCLUDING week\_no

OVERFLOW TABLESPACE ts1

PARTITION BY LIST (week\_no)

(PARTITION VALUES (1, 2, 3, 4)

TABLESPACE ts2,

PARTITION VALUES (5, 6, 7, 8)

TABLESPACE ts3 OVERFLOW TABLESPACE ts4,

PARTITION VALUES (DEFAULT)

TABLESPACE ts5);

4.1.16 Partitioning Restrictions for Multiple Block Sizes

Use caution when creating partitioned objects in a database with tablespaces of different block sizes.

The storage of partitioned objects in such tablespaces is subject to some restrictions. Specifically, all partitions of the following entities must reside in tablespaces of the same block size:

* Conventional tables
* Indexes
* Primary key index segments of index-organized tables
* Overflow segments of index-organized tables
* LOB columns stored out of line

Therefore:

* For each conventional table, all partitions of that table must be stored in tablespaces with the same block size.
* For each index-organized table, all primary key index partitions must reside in tablespaces of the same block size, and all overflow partitions of that table must reside in tablespaces of the same block size. However, index partitions and overflow partitions can reside in tablespaces of different block size.
* For each index (global or local), each partition of that index must reside in tablespaces of the same block size. However, partitions of different indexes defined on the same object can reside in tablespaces of different block sizes.
* For each LOB column, each partition of that column must be stored in tablespaces of equal block sizes. However, different LOB columns can be stored in tablespaces of different block sizes.

When you create or alter a partitioned table or index, all tablespaces you *explicitly specify* for the partitions and subpartitions of each entity must be of the same block size. If you *do not explicitly specify* tablespace storage for an entity, then the tablespaces the database uses by default must be of the same block size. Therefore, you must be aware of the default tablespaces at each level of the partitioned object.

4.1.17 Partitioning of Collections in XMLType and Objects

Partitioning when using XMLType or object tables and columns follows the basic rules for partitioning.

For the purposes of this discussion, the term *Collection* *Tables* is used for the following two categories: (1) ordered collection tables inside XMLType tables or columns, and (2) nested tables inside object tables or columns.

When you partition Collection Tables, Oracle Database uses the partitioning scheme of the base table. Also, Collection Tables are automatically partitioned when the base table is partitioned. DML against a partitioned nested table behaves in a similar manner to that of a reference partitioned table.

Oracle Database provides a LOCAL keyword to equipartition a Collection Table with a partitioned base table. This is the default behavior in this release. The default in earlier releases was not to equipartition the Collection Table with the partitioned base table. Now you must specify the GLOBAL keyword to store an unpartitioned Collection Table with a partitioned base table.

Out-of-line (OOL) table partitioning is supported. However, you cannot create two tables of the same XML schema that has out-of-line tables. This restriction means that exchange partitioning cannot be performed for schemas with OOL tables because it is not possible to have two tables of the same schema.

The statement in the following example creates a nested table partition.

Copy

CREATE TABLE print\_media\_part (

product\_id NUMBER(6),

ad\_id NUMBER(6),

ad\_composite BLOB,

ad\_sourcetext CLOB,

ad\_finaltext CLOB,

ad\_fltextn NCLOB,

ad\_textdocs\_ntab TEXTDOC\_TAB,

ad\_photo BLOB,

ad\_graphic BFILE,

ad\_header ADHEADER\_TYP)

NESTED TABLE ad\_textdocs\_ntab STORE AS textdoc\_nt

PARTITION BY RANGE (product\_id)

(PARTITION p1 VALUES LESS THAN (100),

PARTITION p2 VALUES LESS THAN (200));

**See Also:**

* [Performing PMOs on Partitions that Contain Collection Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-4FE08BA2-9EE6-4987-BC31-1381B77B47E8) and [Partitioning of XMLIndex for Binary XML Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-create-tables-indexes.html#GUID-66FDEA3D-8076-44D9-8FD2-B6CA7D79F9AA) for additional related examples
* [Collection Tables](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/partition-pruning.html#GUID-88E16CEE-3CB1-4AD6-82B3-62B86B899A01) for an example of issuing a query against a partitioned nested table and using the EXPLAIN PLAN to improve performance
* [Changing a Nonpartitioned Table into a Partitioned Table](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/evolve-nopartition-table.html#GUID-6054142E-207A-4DF0-A62A-4C1A94DD36C4) for information about using online redefinition to convert your existing nonpartitioned collection tables to partitioned tables
* [*Oracle Database SQL Language Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=SQLRF01402) for details about CREATE TABLE syntax

4.1.17.1 Performing PMOs on Partitions that Contain Collection Tables

Whether a partition contains Collection Tables or not does not significantly affect your ability to perform partition maintenance operations (PMOs).

Usually, maintenance operations on Collection Tables are carried out on the base table. The following example illustrates a typical ADD PARTITION operation based on the preceding nested table partition:

Copy

ALTER TABLE print\_media\_part

ADD PARTITION p4 VALUES LESS THAN (400)

LOB(ad\_photo, ad\_composite) STORE AS (TABLESPACE omf\_ts1)

LOB(ad\_sourcetext, ad\_finaltext) STORE AS (TABLESPACE omf\_ts1)

NESTED TABLE ad\_textdocs\_ntab STORE AS nt\_p3;

The storage table for nested table storage column ad\_textdocs\_ntab is named nt\_p3 and inherits all other attributes from the table-level defaults and then from the tablespace defaults.

You must directly invoke the following partition maintenance operations on the storage table corresponding to the collection column:

* modify partition
* move partition
* rename partition
* modify the default attributes of a partition

**See Also:**

* [*Oracle Database SQL Language Reference*](https://docs.oracle.com/pls/topic/lookup?ctx=en/database/oracle/oracle-database/12.2/vldbg&id=SQLRF01001) for ADD PARTITION syntax
* [Maintenance Operations Supported on Partitions](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/maintenance-partition-can-be-performed.html#GUID-79391819-DCEF-46AC-977D-199BD2044DA2) for a list of partition maintenance operations that can be performed on partitioned tables and composite partitioned tables

4.1.17.2 Partitioning of XMLIndex for Binary XML Tables

For binary XML tables, XMLIndex is equipartitioned with the base table for range, hash, list, interval, and reference partitions.

In the following example, an XMLIndex is created on a range-partitioned table.

Copy

CREATE TABLE purchase\_order

(id NUMBER, doc XMLTYPE)

PARTITION BY RANGE (id)

(PARTITION p1 VALUES LESS THAN (10),

PARTITION p2 VALUES LESS THAN (MAXVALUE));

CREATE INDEX purchase\_order\_idx ON purchase\_order(doc)

INDEXTYPE IS XDB.XMLINDEX LOCAL;