

Oracle 8 partitions

Larry Miller, Consultant

Introduction

Partitions have been introduced in this version mainly to ease the management of large objects, in terms of number of rows. This is one of the major new features of Oracle 8 and can have a major impact on operations on very large tables.

Partitioning is always based on different ranges of values of a column (or several concatenated columns) in the table. Each partition is associated with a range. As a corollary, columns on which partitioning is based must not be updated.

Partitions bring the following benefits :

- They ease the maintenance of very large tables (partition-based maintenance)
- They reduce the time that data is unavailable (selective unavailability) during maintenance operations, or during data recovery.
- They increase performance for decisional queries which involve the partitioning criterion (full scan of the partition only).
- They allow better control of the physical location of data.
- They allow DML (data manipulation language) parallelism.

Maintenance operations

The following operations are possible on partitions :

- They can be moved to another tablespace
- Partitions can be added
- Partitions can be deleted
- Partitions can be split
- Two partitions can be combined
- A partition can be moved to a non-partitioned table and vice-versa (useful for migrations)

Indexing

Partitioned tables can have (local or global) partitioned or non-partitioned indexes. An unpartitioned table can have a global partitioned index.

Local index (created with the LOCAL attribute)

A local index is partitioned on the same lines as the corresponding table; therefore, all the keys in a partition of the index, reference rows in a single partition of the table (it is implicitly equi-partitioned with the table). This index can be prefixed or not. It is prefixed if it is partitioned on the leftmost column(s) of the index. It can be unique or non-unique. Oracle does not support non-prefixed unique local indexes (unless the index key is a subset of the partitioning key).

Global index (created with the GLOBAL attribute)

A global index is not partitioned on the same lines as the corresponding table. Keys in a partition of the index can reference rows in a number of partitions of the table (it is usually not equi-partitioned with the table). This index must be prefixed. Oracle does not support non-prefixed global indexes.

A global index prevents maintenance operations from being independent at the partition level.

A quick summary of possible combinations

Index type	Can be unique	Table partitioned on	Index columns	Index partitioned on
Local, prefixed	Y	A	A,B	A
Local, non prefixed	Y	A	B	A
Global, prefixed	Y	A	B	B
Global, non-prefixed	Not supported			

Notes :

1. To be unique, the key of a local, non-prefixed index must be a subset of the partitioning key.
2. The same rules apply to PRIMARY KEY or UNIQUE constraints on partitioned tables.
3. Implicitly, to be able to create a unique index on the main identifier of a table (the primary key) while retaining the advantage of equi-partitioning, and therefore partition independence, one MUST partition the table ON THE PRIMARY KEY. Otherwise, one must create a unique global index (and partition independence is lost).

Tests

An extensive number of tests have been carried out on a two million row table, created with a parallelism degree of 8 and divided into 8 partitions. This is a brief summary of results. The partition key was not, in this case, the primary key ID but another column we shall call P. A number of various indexes have been created for each test.

Queries

In order to have a reference, most operations were also executed on a non-partitioned copy of the same table, with the primary key index.

In order to have results meaningful to anybody, and independent from the hardware configuration, they are expressed in number of blocks accessed by Oracle (logical reads, LR in the following table) .

Operation	Partitioned table index	LR, partitioned table	LR, non-partitioned table	Comment
SELECT with a condition on ID and P	(P, ID)	4	7	Quite similar - having P in the index goes right to the good partition.
SELECT with a condition on ID alone	ID (local)	49	4	Costs are no longer similar! The Index is not prefixed, and P is not used as a search key. Oracle must access each index partition before bringing the result back.
SELECT with a condition on ID and P	ID (local)	5	4	Similar. The index is not prefixed, but as there is a condition on P, Oracle directly accesses the good index partition.
SELECT with a condition on	none	78448	79137	Mandatory full table scan in both cases!

column C (unindexed)				
SELECT with a condition on C (unindexed) and P	none	10871	79131	The difference is obvious, in the first case the full scan is limited to a single partition.

Notes :

1. One can access a table in a SELECT and specify the partition to use. But then, first one leaves the well-trodden path of SQL standards, and then queries begin to be dangerously dependent on the physical implementation of the database - a heresy in the (even unrelational) database world.
2. About partition independence, as soon as a tablespace which contains a table partition is no longer available (offline, crashed, etc.) any operation which requires a full scan of all partitions will cause an error. To work around the unavailability of the partition, the query must be rewritten in a way which no longer requires an access to the partition, or the partition must be destroyed so that the whole table looks intact.

Maintenance

All maintenance operations have been successfully tested. Here are a few comments on specific points :

Moving a partition

When a partition is moved, any related index must be rebuilt (which may be an opportunity to move it too).

Adding a partition/dropping a partition

It is also possible to truncate a partition using :

```
alter table table_name truncate partition partition_name
```

The matching local index partition is also automatically truncated. But there is, as with 'move', another side-effect : global indexes are invalidated.

BEWARE : As always, operations such as DROP or CREATE require the deactivation or deletion of referential constraints on the table for the duration of the operation.

Partition split

One must be careful, when rebuilding the index partitions, to specify the tablespaces, because the default behavior of Oracle is to store them in the same tablespaces as the table.

For a local index, the partition is automatically split too. One may specifically request the splitting of a partition of a global index, but this is of course impossible for a local index (for which the partition split is mandatorily a sub-product of the same operation applied to the table).

If there is a global index on the table, it is also automatically split, but the names of resulting partitions are automatically generated by Oracle (one must then search the data dictionary to find them); they must also be rebuilt after the split.

Partition combination

There is no SQL statement as such to combine two partitions. To combine two partitions A and B one must choose between :

- 1.

- export A
- destroy A
- import A into B

2.

- Exchange the data in A with a non-partitioned table NP created for this purpose
- destroy A
- insert into B the data from NP

All these operations are standard operations.

Concurrency during maintenance operations

Maintenance operations do not prevent other operations from being performed at the same time. During operations such as MOVE, SPLIT, EXCHANGE or direct load insert, one can at the same time :

- Perform a similar operation on a different partition of the same table.
- Query the data in the table.
- Update the table if the partition being maintained contains no row to update.
- Rebuild a partition of a local index of another partition of the table.

During operations such as CREATE INDEX, ALTER INDEX REBUILD PARTITION or ALTER INDEX DROP/SPLIT PARTITION on a global index, one can at the same time :

- Query the data in the table
- Create other indexes on the table, rebuild the partitions of other indexes, or drop or split them.

During operations such as ALTER INDEX REBUILD PARTITION on a partition of a local index, one can at the same time :

- Execute a MOVE or SPLIT or direct insert on any other partition of the table (other than the one which corresponds to the index partition been maintained).
- Query the data in the table.
- Update the table if the partition being maintained contains no row to update.
- Rebuild other partitions of the same or other indexes; create other indexes.

Conclusion

Benefits and weak points

Tests have underlined the following points :

- Better management of the balance and physical location of data
- Increased performance, especially for Management Reporting (when accesses are limited, thanks to the WHERE clause, to some partitions)
- Increased performance with parallel DML for massive updates
- Possibility to delete quickly a huge amount of data (purges)
- Good maintenance functions (split, move)

Weak points :

- The best way to partition the data is hard to determine
- How to index becomes nightmarish (many possible combinations, depends on how the table has been partitioned)
- A row cannot be migrated from a partition to another partition other than by deleting it and reinserting it (with the correct value for the partition key)

Advice

In a general way, the use of partitions can only be strongly advised when huge volumes of data are involved, whatever the type of application (OLTP or Management Reporting). Maintenance and reorganization operations will be greatly eased.

You just have to consider how difficult it is to reorganize the data of a traditional table (purging it, physically moving it or redistributing it in other tablespaces, defragmenting it, etc.) to understand all the benefits of commands such as MOVE or SPLIT.

However, the decision on how to partition and how to index implies a lot of forethought.

If we have a look at performance, we understand that partitioning is **extremely dependent** on the queries which are going to be executed, and is therefore closely related to the functional side. But one must also consider maintenance, and security if the database partially crashes (e.g. a single partition becomes corrupted). Compromises are therefore unavoidable.

Choices will be quite different, depending on the domain. There is no hard and fast rule, as there are so many possible combinations. One must test thoroughly to find the best compromises.

Management Reporting : Partitioning must be done with parallel scans (on all partitions), or full scans of a single partition in mind (beware of WHERE clauses).

Local indexes will be favored, in order to ease maintenance operations (the duration of which will be proportional to the partition size). This is especially useful for tables storing historical data (partitioned on a date).

OLTP : Partitioning can be based instead, on the primary key (if one wants to create unique, therefore prefixed, local indexes). Otherwise we shall have to create global indexes for primary keys (but maintenance operations will then take longer).

Non-prefixed indexes must be avoided, as they require one access per index partition which impairs performance