DBA

Table Partitioning

Partition, Random Number, Index

Codes for this documentation can be found in github.

What is Partition?

Partitioning is the process of taking very large table or index and physically break them into small manageable pieces.Partitioning enables tables and indexes or index-organized tables to be subdivided into smaller manageable pieces and these each small piece is called a "partition".

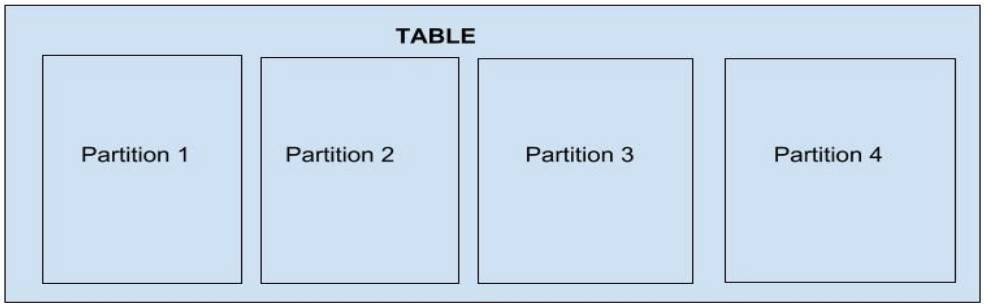

Fig : Table Partitioning

Suggestions when partitioning tables:

1. Tables greater than 2GB should always be considered as candidate for partitioning.
2. Tables containing historical data where new data is always added to new partition.
3. When the contents of tables need to be distributed across different types of storage machines.

Partition is need because:

When the size of table and indexes increases, data access performance reduce very significantly. Anyone with un-partitioned databases over 500 gigabytes is courting disaster. Databases become unmanageable, and serious problems occur:

- Files recovery takes days, not minutes
- Rebuilding indexes (important to reclaim space and improve performance) can take days
- Queries with full-table scans take hours to complete
- Index range scans become inefficient

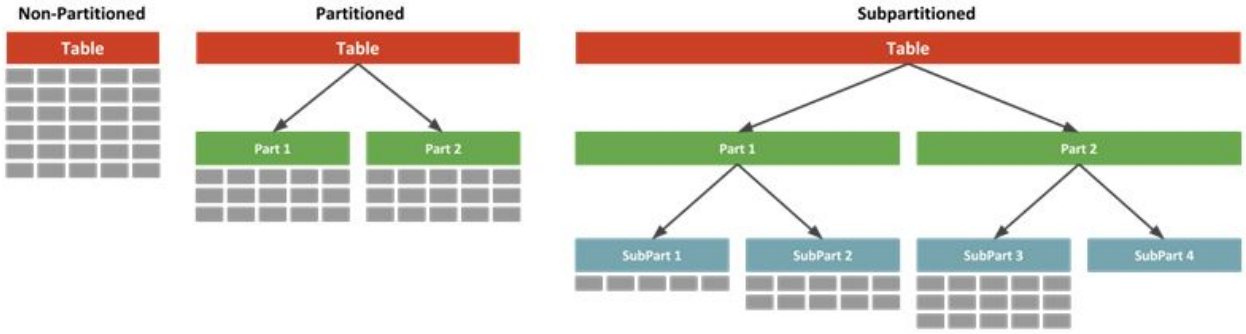

Fig : Partition and Subpartition

Partition Strategies:

A table can be partitioned by many factors. Some of the factors to partition a table are as follows:

- Partition by range
- Partition by hash
- Partition by list
- Composite partition

1. **Range Partitioning**

In range partitioning, Data are mapped based on range of values of the partitioning key that is established for each partition.

In our case, we created a range partition based on salary. Salary less than 10000 is partitioned in low_salary partition, salary less than 50000 is partitioned in medium_salary partition and salary less than 150000 is partitioned in high_salary partition.

SQL Script:

Create table tab_name (…

) partition by range(salary)

(partition low_salary values less than (10000) tablespace TEST,

partition medium_salary values less than (50000) tablespace TEST,

partition high_salary values less than (150000) tablespace TEST

);

2. **Hash Partitioning**

Hash partitioning maps data to partitions based on a hashing algorithm that Oracle applies to the partitioning key that you identify. The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size.

In our case, we have hash partitioned the table with respect to id. Hash Algorithm counts the number of id in the table, partitions the table into number of partitions--specified by the DBA-- (i.e 4) such that each partition is almost of equal size.

SQL Script:

Create table tab_name (...)

partition by hash(id)

partitions 4

store in (TEST);

Here, TEST is the tablespace_name.

3. **List Partitioning**

List partitioning maps data to partitions if the value of partition key is in the list of values specified. We can use Default partition to avoid specifying all the partition values.

In our case, we partitioned the department column based on frontend department(HTML, CSS, ANGULAR) or backend department(JAVA, DATABASE). Further, we also specified NULL department and Default unknown department.

SQL Script:

Create table tab_name (...)

partition by list(department)

(partition backend_dept values ('JAVA', 'DATABASE'),

partition frontend_dept values ('ANGULAR', 'HTML', 'CSS'),

partition dept_null values (NULL),

partition dept_unknown values (DEFAULT)

);

4. **Composite Partitioning**

Composite partitioning is the combination of two or more partitions. Each partition is divided into subpartition.

In our case, we created a partition by range and we further divided into 8 subpartitions by hash. There are 3 partitions by range and 8 partitions by hash. Total there are 8*3=24 partitions.

SQL Scripts:

Create table tab_name(…)

partition by range(join_date)

subpartition by hash(id)

subpartitions 8

(partition before_2000 values less than(TO_DATE('01/01/2000', 'DD/MM/YYYY')),

partition before_2010 values less than (TO_DATE('01/01/2010', 'DD/MM/YYYY')),

partition before_current_date values less than (MAXVALUE)

Partitioning Indexes:

There are basically two types of partitioned indexes:

- **Local:** All index entries in a single partition will correspond to a single table partition.
- **Global:** Index in a single partition may correspond to multiple table partitions.

Both type of indexes are further divided into:

- **Prefixed:** Probing this type of index is less costly. Prefixed index is created by creating an index in the column by which partition is performed. If a query specifies the partition key in the where clause partition pruning is possible, that is, not all partitions will be searched.
- **Non-prefixed:** Does not support partition pruning, but is effective in accessing data that spans multiple partitions. Often used for indexing a column that is not the tables partition key.

Additional feature used:

1. **Random Value insert:**

After we created a table with different partitions, we inserted the random values into the table. To get the random number we used DBMS_RANDOM package and called values() function.

To generate random number:

Rand_num := DBMS_RANDOM.values(low_limit, high_limit);

Here, low_limit is the lower limit of wanted the random number, high_limit is the higher limit of the wanted random number.

To generate random string:

Unlike generating random numbers, we don't have a particular function to generate random string.

In our case, we generated the random string using CASE...WHEN Clause, or IF... ELSIF....ELSE Clause.

Rand_num := DBMS_RANDOM.values(low_limit, high_limit);

Case rand_num

```
When 1 then rand_string := 'value1';
When 2 then rand_string := 'value2';
Else rand_string := 'default_value';
End case;
```

Alternatively, we can use IF..ELSIF....ELSE clause to generate the random string.

```
Rand_num := DBMS_RANDOM.values(low_limt, high_limit);
If (rand_num = 1) then
    Rand_string := 'value1'
Elsif (rand_num = 2) then
    Rand_string := 'value2'
Else
    Rand_string := 'default_value'
End if;
```

References:

[1]. https://oracle-base.com/articles/8i/partitioned-tables-and-indexes

[2]. https://docs.oracle.com/cd/B28359_01/server.111/b32024/partition.htm#i462869