Module No.01: -Distributed Database

Practical No.1

Aim: - Implementation of Data partitioning through Range and List partitioning

Objective: - To learn data partitioning using Oracle

Theory: -

Distributed Database: -
A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Data partitioning: -
Maintenance of large tables and indexes can become very time and resource consuming.
At the same time, data access performance can reduce drastically for these objects.
Partitioning of tables and indexes can benefit the performance and maintenance in several ways.
Partition independence means backup and recovery operations can be performed on individual partitions, whilst leaving the other partitions available.
Query performance can be improved as access can be limited to relevant partitions only.
There is a greater ability for parallelism with more partitions.

Range Partitioning Tables: -
The RANGE partitioning type is used to assign each partition a range of values generated by the partitioning expression.
Ranges must be ordered, contiguous and non-overlapping.
The minimum value is always included in the first range. The highest value may or may not be included in the last range.
A variant of this partitioning method, RANGE COLUMNS, allows us to use multiple columns and more datatypes.

Syntax: -

The last part of a CREATE TABLE statement can be definition of the new table's partitions. In the case of RANGE partitioning, the syntax is the following:

PARTITION BY RANGE (partitioning_expression)

(

PARTITION partition_name VALUES LESS THAN (value),[PARTITION partition_name VALUES LESS THAN (value), ...]

)

Implementation: -

Step 1: - Create table with Range Partitioning

Query: -

```
CREATE TABLE sales_range5
(
salesman_id NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount NUMBER(10),
sales_date DATE)
PARTITION BY RANGE(sales_date)
(
PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('01/02/2000','DD/MM/YYYY'),
PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('01/03/2000','DD/MM/YYYY'),
PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('01/04/2000','DD/MM/YYYY'),
PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('01/05/2000','DD/MM/YYYY')
)
) ;
```

Output: -

```
SQL>  CREATE TABLE sales_range5
  2   (salesman_id NUMBER(5),
  3   salesman_name VARCHAR2(30),
  4   sales_amount NUMBER(10),
  5   sales_date DATE)
  6   PARTITION BY RANGE(sales_date)
  7   (
  8   PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('01/02/2000','DD/MM/YYYY')),  PARTITION sales_feb2000 VALUES LESS
 THAN(TO_DATE('01/03/2000','DD/MM/YYYY')),  PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('01/04/2000','DD/MM/YYYY'))
,  PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('01/05/2000','DD/MM/YYYY'))
  9   )
 10   ;

Table created.
```

Step 2: - Insert data into the above table

Query: -

```
insert into sales_range5 values(1,'John
Smith',5000,TO_DATE('23/02/2000','DD/MM/YYY
Y'));
insert into sales_range5
values(2,'David',4000,TO_DATE('25/03/2000','DD/M
M/YYYY'));
insert into sales_range5
values(3,'Tina',6000,TO_DATE('12/04/2000','DD/M
M/YYYY'));
insert into sales_range5
values(4,'Seema',8000,TO_DATE('26/02/2000','DD/
MM/YYYY'));
insert into sales_range5
values(5,'Rahul',3000,TO_DATE('01/01/2000','DD/M
M/YYYY'));
```

Output: -

```
SQL Plus

SQL> insert into sales_range5 values(1,'John
  2  Smith',5000,TO_DATE('23/02/2000','DD/MM/YYYY'));

1 row created.

SQL> insert into sales_range5 values(2,'David',4000,TO_DATE('25/03/2000','DD/MM/YYYY'));

1 row created.

SQL> insert into sales_range5 values(3,'Tina',6000,TO_DATE('12/04/2000','DD/MM/YYYY'));

1 row created.
```

```
SQL> insert into sales_range5 values(4,'Seema',8000,TO_DATE('26/02/2000','DD/MM/YYYY'));

1 row created.

SQL> insert into sales_range5 values(5,'Rahul',3000,TO_DATE('01/01/2000','DD/MM/YYYY'));

1 row created.

SQL>
```

Step 3: - Display data from sales_range5 table

Query: -

```
SELECT * FROM sales_range5
```

Output: -

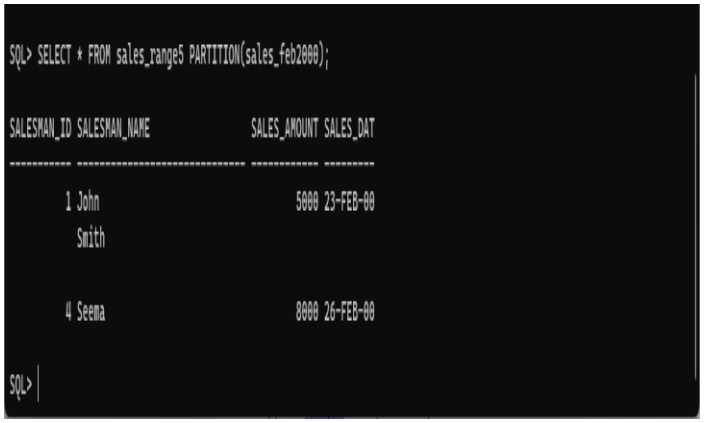
```
SQL> SELECT * FROM sales_range5;

SALESMAN_ID SALESMAN_NAME              SALES_AMOUNT SALES_DAT
----------- ------------------------------ ------------ ---------
          5 Rahul                          3000 01-JAN-00
          1 John                           5000 23-FEB-00
            Smith

          4 Seema                          8000 26-FEB-00
          2 David                          4000 25-MAR-00
          3 Tina                           6000 12-APR-00

SQL>
```

Step 4: - Display data from sales_feb2000 partition of sales_range5 table

Query: -

```
SELECT * FROM sales_range5
PARTITION(sales_feb2000);
```

Output: -



List Partitioning: -

LIST partitioning is conceptually similar to RANGE partitioning.

In both cases you decide a partitioning expression (a column, or a slightly more complex calculation) and use it to determine which partitions will contain each row.

However, with the RANGE type, partitioning is done by assigning a range of values to each partition.

With the LIST type, we assign a set of values to each partition.

This is usually preferred if the partitioning expression can return a limited set of values.

Syntax: -

The last part of a CREATE TABLE statement can be the definition of the new table's partitions. In the case of LIST partitioning, the syntax is the following:

```
PARTITION BY LIST (partitioning_expression)
(
PARTITION partition_name VALUES IN
(value_list),
[ PARTITION partition_name VALUES IN
(value_list), ... ]
[ PARTITION partition_name DEFAULT ])
```

Implementation: -

Step 1: - Create table with List Partitioning

Query: -

```
CREATE TABLE sales_list6
 (
 salesman_id NUMBER(5),
 salesman_name VARCHAR2(30),
 sales_state VARCHAR2(20),
 sales_amount NUMBER(10),
 sales_date DATE
)
PARTITION BY LIST(sales_state)
 (
 PARTITION sales_west VALUES('California',
'Hawaii'),
 PARTITION sales_east VALUES ('New York',
'Virginia', 'Florida'),
 PARTITION sales_central VALUES('Texas',
'Illinois'),
 PARTITION sales_other VALUES(DEFAULT)
 )
 enable row movement ;
```

Step 2: - Insert data into the above table

Query: -

```
insert into sales_list5 values(1,'John
Smith','NewYork',2300,TO_DATE('23/02/2000','DD/
MM/YYYY'));
 insert into sales_list5
values(2,'Alis','California',4500,TO_DATE('20/04/20
04','DD/MM/YYYY'));
 insert into sales_list5
values(3,'David','Texas',3600,TO_DATE('14/05/2004'
,'DD/MM/YYYY'));
 insert into sales_list5
values(4,'Steve','Hongkong',1200,TO_DATE('03/02/2
004','DD/MM/YYYY'));
insert into sales_list5
values(5,'Mark','Florida',6600,TO_DATE('08/04/2004
','DD/MM/YYYY'));
```

Output: -

```
SQL> insert into sales_list5 values(1,'John Smith','NewYork',2300,TO_DATE('23/02/2000','DD/MM/YYYY'));

1 row created.

SQL>  insert into sales_list5 values(2,'Alis','California',4500,TO_DATE('20/04/2004','DD/MM/YYYY'));

1 row created.

SQL>  insert into sales_list5  values(3,'David','Texas',3600,TO_DATE('14/05/2004','DD/MM/YYYY'));

1 row created.

SQL>  insert into sales_list5  values(4,'Steve','Hongkong',1200,TO_DATE('03/02/2004','DD/MM/YYYY'));

1 row created.

SQL>  insert into sales_list5  values(5,'Mark','Florida',6600,TO_DATE('08/04/2004','DD/MM/YYYY') );

1 row created.

SQL>
```

Step 3: - Display data from sales_list table

Query: -

SELECT * FROM sales_list5;

Output: -

```
SQL> SELECT * FROM sales_list5;

SALESMAN_ID SALESMAN_NAME                SALES_STATE          SALES_AMOUNT
----------- ---------------------------- -------------------- ------------
SALES_DAT
---------
          2 Alis                         California                   4500
20-APR-04

          5 Mark                         Florida                      6600
08-APR-04

          3 David                        Texas                        3600
14-MAY-04

SALESMAN_ID SALESMAN_NAME                SALES_STATE          SALES_AMOUNT
----------- ---------------------------- -------------------- ------------
SALES_DAT
---------
          1 John Smith                   NewYork                      2300
23-FEB-00

          4 Steve                        Hongkong                     1200
03-FEB-04

SQL>
```

Step 4: - Display data from sales_east and sales_west partition of sales_list5 table

Query: -

SELECT * FROM sales_list partition(sales_east);

Output: -

```
SQL> SELECT * FROM sales_list6 partition(sales_east);

SALESMAN_ID SALESMAN_NAME                SALES_STATE          SALES_AMOUNT
----------- ---------------------------- -------------------- ------------
SALES_DAT
---------
          5 Mark                         Florida                      6600
08-APR-04


SQL>
```

Query: -

SELECT * FROM sales_list partition(sales_west);

Output: -

```
SQL> SELECT * FROM sales_list6 partition(sales_west);

SALESMAN_ID SALESMAN_NAME                SALES_STATE          SALES_AMOUNT
----------- ---------------------------- -------------------- ------------
SALES_DAT
---------
          2 Alis                         California                   4500
20-APR-04


SQL>
```

Hash Partitioning: -

Partitioning by **HASH** is used primarily to ensure an even distribution of data among a predetermined number of partitions. With range or list partitioning, you must specify explicitly into which partition a given column value or set of column values is to be stored; with hash partitioning, MySQL takes care of this for you, and you need only specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided.

To partition a table using **HASH** partitioning, it is necessary to append to the **CREATE TABLE** statement a **PARTITION BY HASH (*expr*)** clause, where ***expr*** is an expression that returns an integer. This can simply be the name of a column whose type is one of MySQL's integer types. In addition, you most likely want to follow this with **PARTITIONS *num***, where ***num*** is a positive integer representing the number of partitions into which the table is to be divided.

Syntax: -
The last part of a CREATE TABLE statement can be the definition of the new table's partitions. In the case of HASH partitioning, the syntax is the following:

PARTITION BY HASH (partitioning_expression)
PARTITIONS (noofpartions);

Implementation: -

Step 1: - Create table with hash Partitioning

Query: -

```
CREATE TABLE sales
(
dept_no number(10),
part_no varchar2(30),
country varchar2(20),
day date,
amount number
)
PARTITION BY HASH (part_no)PARTITIONS 2;
```

Step 3: - Display data from hash table

Query: -

```
SELECT table_name, partition_name FROM
ALL_TAB_PARTITIONS WHERE
table_name = 'SALES' ORDER BY 1,2;
```

Output: -

OUTPUT

```
 table_name | partition_name
------------+----------------
 SALES      | SYS0101
 SALES      | SYS0102
 SALES      | SYS0103
 SALES      | SYS0104
 SALES      | SYS0105
 SALES      | SYS0106
 SALES      | SYS0107
 SALES      | SYS0108
(8 rows)
```