In normal scenario, when table contains data of different regions or any other scenario, region like APAC, EMEA and north America and if we try to retrieval data of EMEA region which contains around 30% of the table data.

Then indexing is ineffective, index will work or effective if we are fetching 10 to 15% of total data, if retrieval data more than 30% then optimizer will go for FULL table scan and it takes too much time.

The solution is to go for partition

**What is partitioning**

partition allows table, index or index organized tables to be subdivided in to smaller pieces, enabling these database objects to be managed and accessed at a finer level of granularity and each piece of table, index or index organized table is called as Partition.

**Following are Advantages of Partition:**

1.Increase Performance

Partition pruning

Partition wise joins

2.Increases availability

3.Enable storage cost optimization

4.Enables Simpler management

Divide and conquer approach

Faster backup and restore

Low storage usage

**When to partition the table?**

1.Table should be greater than 2 GB

2.Tables which contains historical data in which new data will be added in to newest partition. The real life example of this is historical table which contains updatable data for one year other data is read only.

3.When contents of the table needs to be distributed in different storage devices.

4.When table performance is weak and we need to improve performance of application.

**Partition key**

It helps uniquely identify how each row should be mapped to which partition.

**Types of Partition**

1. **List partition**
2. **Range partition**
3. **Hash partition**

**List Partition**

List partition enables you to explicitly control how the partition of tables needs to be done by specifying list of distinct values as partition key in description of each partition. When there is a set of distinct values in the table which is properly divided then user should go with list partition. By listing the distinct values user should do the partition.

Example

Create table transaction\_Demo(transaction\_id number,

transaction\_type varchar2(30),

amount number,

tran\_Date date,

region varchar2(10))

partition by list(region)

(partition p\_ap VALUES('AP'),

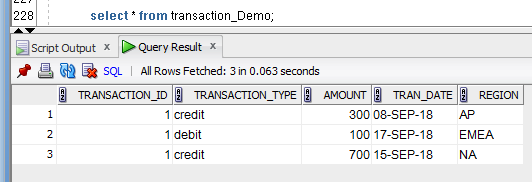
partition p\_emea values('EMEA'),

partition p\_na values('NA'));

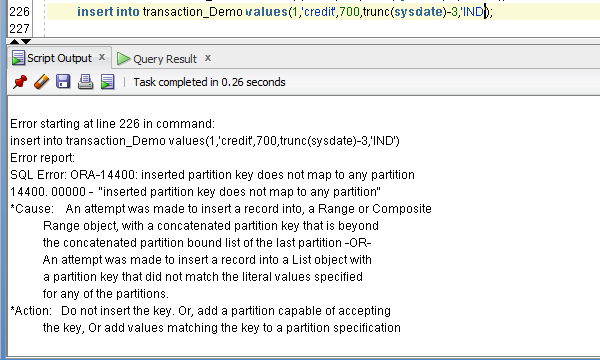
insert into transaction\_Demo values(1,'credit',300,trunc(sysdate)-10,'AP');

insert into transaction\_Demo values(1,'debit',100,trunc(sysdate)-1,'EMEA');

insert into transaction\_Demo values(1,'credit',700,trunc(sysdate)-3,'NA');



If there is no corresponding partition key value then oracle throws error while inserting data



We can overcome above problem by adding default partition value.

Create table transaction\_Demo(transaction\_id number,transaction\_type varchar2(30),amount number,tran\_Date date,region varchar2(10))

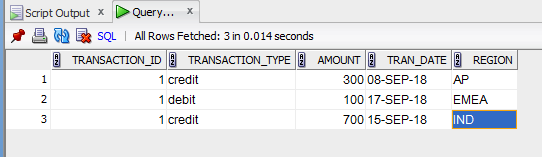
partition by list(region)

(partition p\_ap VALUES('AP'),

partition p\_emea values('EMEA'),

partition p\_na values('NA'),

**partition p\_unknown values(DEFAULT));**



**Range partition**

When in the specified table the data is based on the specific date range and it is properly divided in some range then user should go for the partitioned named as ‘Range Partition’.This partition type is most common type of Table partitioning which is been useful for Data warehouse to store the historical data in given date range.The partitioning is done in such way that the expression values lies within the specific range.This kind of Table partition is used when there is a particular date range available.

**Example**

drop table transaction\_Demo;

Create table transaction\_Demo(transaction\_id number,

transaction\_type varchar2(30),

amount number,

tran\_Date date,

region varchar2(10))

partition by range(tran\_Date)

(partition p\_ap VALUES less than (to\_date('2016-01-01','YYYY-MM-DD')),

partition p\_emea values less than (to\_date('2017-01-01','YYYY-MM-DD')),

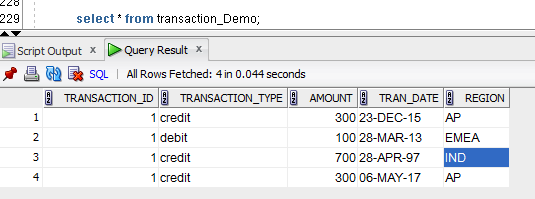
partition p\_na values less than (to\_date('2018-01-01','YYYY-MM-DD'))

);

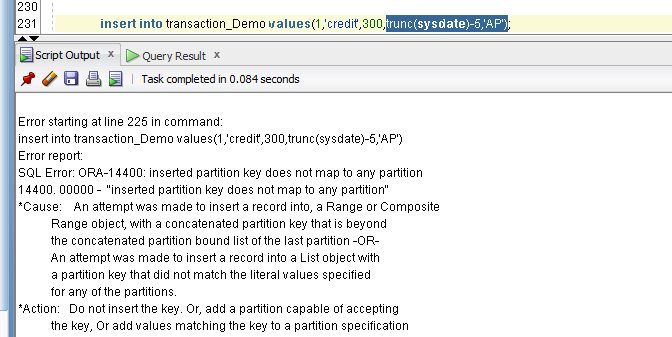
insert into transaction\_Demo values(1,'credit',300,trunc(sysdate)-500,'AP');

insert into transaction\_Demo values(1,'debit',100,trunc(sysdate)-2000,'EMEA');

insert into transaction\_Demo values(1,'credit',700,trunc(sysdate)-300000,'IND');



When we try to insert out of range



We can overcome above by using MAXVALUE

Create table transaction\_Demo(transaction\_id number,transaction\_type varchar2(30),amount number,tran\_Date date,region varchar2(10))

partition by range(tran\_Date)

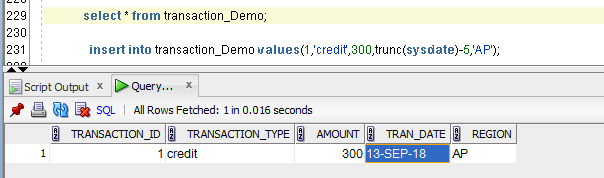
(partition p\_ap VALUES less than (to\_date('2016-01-01','YYYY-MM-DD')),

partition p\_emea values less than (to\_date('2017-01-01','YYYY-MM-DD')),

partition p\_na values less than (to\_date('2018-01-01','YYYY-MM-DD')),

**partition p\_default values less than (MAXVALUE)**

);



**Interval Partitioning**

Interval partitioning is an extension to range partitioning in which, beyond a point in time, partitions are defined by an interval. Interval partitions are automatically created by the database when data is inserted into the partition.

Interval partitioning provides an easy way for interval partitions to be automatically created as data arrives. Interval partitions can also be used for all other partition maintenance operations.

CREATE TABLE salestable

(s\_productid NUMBER,

s\_saledate DATE,

s\_custid NUMBER,

s\_totalprice NUMBER)

PARTITION BY RANGE(s\_saledate)

INTERVAL(NUMTOYMINTERVAL(1,'MONTH')) STORE IN (tbs1,tbs2,tbs3,tbs4)

(PARTITION sal05q1 VALUES LESS THAN (TO\_DATE('01-APR-2005', 'DD-MON-YYYY')) TABLESPACE tbs1,

PARTITION sal05q2 VALUES LESS THAN (TO\_DATE('01-JUL-2005', 'DD-MON-YYYY')) TABLESPACE tbs2,

PARTITION sal05q3 VALUES LESS THAN (TO\_DATE('01-OCT-2005', 'DD-MON-YYYY')) TABLESPACE tbs3,

PARTITION sal05q4 VALUES LESS THAN (TO\_DATE('01-JAN-2006', 'DD-MON-YYYY')) TABLESPACE tbs4,

PARTITION sal06q1 VALUES LESS THAN (TO\_DATE('01-APR-2006', 'DD-MON-YYYY')) TABLESPACE tbs1,

PARTITION sal06q2 VALUES LESS THAN (TO\_DATE('01-JUL-2006', 'DD-MON-YYYY')) TABLESPACE tbs2,

PARTITION sal06q3 VALUES LESS THAN (TO\_DATE('01-OCT-2006', 'DD-MON-YYYY')) TABLESPACE tbs3,

PARTITION sal06q4 VALUES LESS THAN (TO\_DATE('01-JAN-2007', 'DD-MON-YYYY')) TABLESPACE tbs4);

**Hash partition**

Hash partitioning is type of partitioning where data is partitioned by using the hashing algorithms.Different hashing algorithms are applied to partition key that you identify.Hash partition is mainly used to distribute data among the different storage devices.Hash partition is easy to use and best alternative for list partition when data is not historical.

Example

Create table Emp

(emp\_no number(2),emp\_name varchar(2))

partition by hash(emp\_no) partitions 5;

similar one is

Create table Emp

(emp\_no number(2),emp\_name varchar(2))

partition by hash(emp\_no) (partition p\_1,partition p\_2,partition p\_3,partition p\_4,partition p\_5);

**Composite Partitioning**

Below are different types of composite partitioning

Range – Hash

Range – List

Range – range

List – hash

List – range

List – List

Hash – hash

Hash – list

Hash – Range

CREATE TABLE quarterly\_regional\_sales

(deptno number, item\_no varchar2(20),

txn\_date date, txn\_amount number, state varchar2(2))

TABLESPACE ts4

PARTITION BY RANGE (txn\_date)

SUBPARTITION BY LIST (state)

(PARTITION q1\_1999 VALUES LESS THAN (TO\_DATE('1-APR-1999','DD-MON-YYYY'))

(SUBPARTITION q1\_1999\_northwest VALUES ('OR', 'WA'),

SUBPARTITION q1\_1999\_southwest VALUES ('AZ', 'UT', 'NM'),

SUBPARTITION q1\_1999\_northeast VALUES ('NY', 'VM', 'NJ'),

SUBPARTITION q1\_1999\_southeast VALUES ('FL', 'GA'),

SUBPARTITION q1\_1999\_northcentral VALUES ('SD', 'WI'),

SUBPARTITION q1\_1999\_southcentral VALUES ('OK', 'TX')

),

PARTITION q2\_1999 VALUES LESS THAN ( TO\_DATE('1-JUL-1999','DD-MON-YYYY'))

(SUBPARTITION q2\_1999\_northwest VALUES ('OR', 'WA'),

SUBPARTITION q2\_1999\_southwest VALUES ('AZ', 'UT', 'NM'),

SUBPARTITION q2\_1999\_northeast VALUES ('NY', 'VM', 'NJ'),

SUBPARTITION q2\_1999\_southeast VALUES ('FL', 'GA'),

SUBPARTITION q2\_1999\_northcentral VALUES ('SD', 'WI'),

SUBPARTITION q2\_1999\_southcentral VALUES ('OK', 'TX')

));

**Local and Global Indexes**

Local prefixed indexes are indexes that are partitioned using the same partitioned key and same range boundaries as the partitioned table.

create table tran\_part\_idx\_Demo

( id number, name varchar2(20),tran\_Date date)

partition by range(tran\_Date)

Interval(numtoyminterval(1,'MONTH'))

(partition p1 values less than (to\_date('1-1-2018','DD-MM-YYYY')),

partition p2 values less than(to\_date('1-2-2018','DD-MM-YYYY'))

);

Create index idx\_tran\_part\_idx\_tran\_Date on tran\_part\_idx\_Demo

(tran\_date) local;

Create index idx\_tran\_part\_idx\_tran\_id on tran\_part\_idx\_Demo

(id) local;

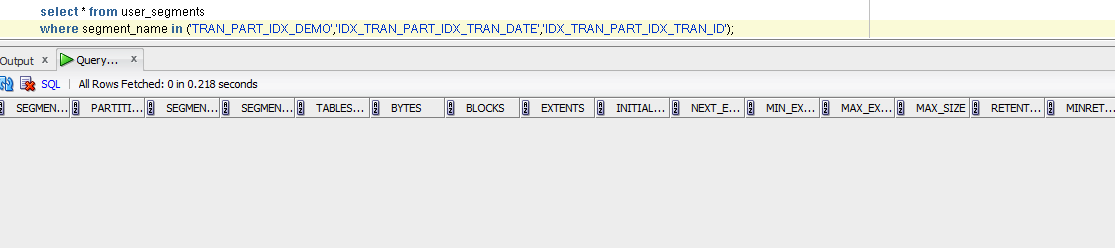
**Difference**

Local prefixed index will use partition pruning.

Local Non prefixed index will not use partition pruning.

No of segments allocated we can check thorugh

Below query

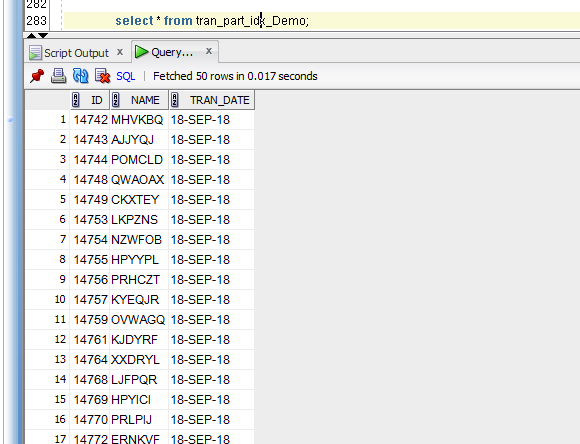


Segment will not be allocated till data is inserted.

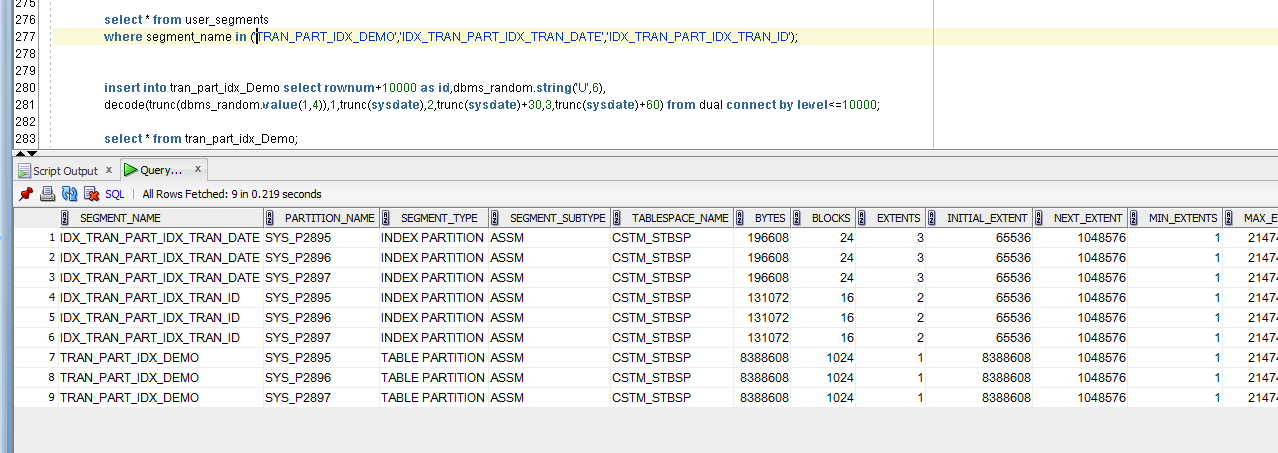
Lets insert data

insert into tran\_part\_idx\_Demo select rownum+10000 as id,dbms\_random.string('U',6),

decode(trunc(dbms\_random.value(1,4)),1,trunc(sysdate),2,trunc(sysdate)+30,3,trunc(sysdate)+60) from dual connect by level<=10000;



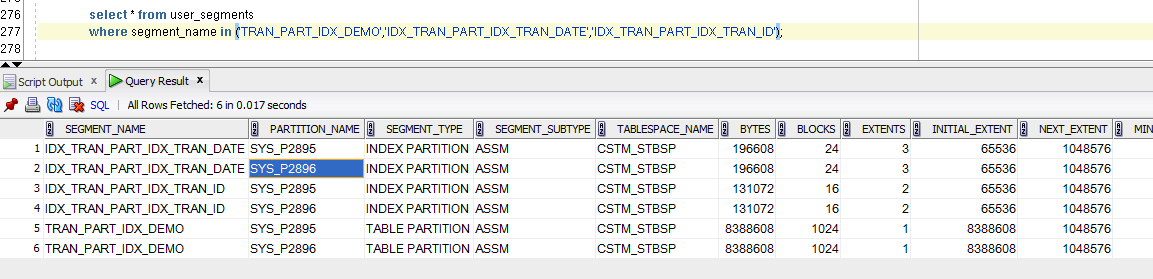
Segments are created after insert.



Now dropping partition

alter table tran\_part\_idx\_Demo drop partition SYS\_P2897;

we can see segment from both indexes are dropped



Global prefixed partitioned index

Index boundary is usually not same as the table partition boundaries.

One partition of indexes can point to multiple partitions

create index idx\_tran\_part\_idx\_id on tran\_part\_idx\_Demo(id) Global

partition by range(id)

(partition p1 values less than(100),

partition p2 values less than(200),

partition p3 values less than(MAXVALUE));

Global index

. preferred in OLTP systems

.partition pruning of tables is not utilized

. become unusable if a partition is dropped or moved

.instant index rebuild

. not managed by oracle

.alter table tran\_part\_idx\_Demo drop partition SYS\_P2897 update global indexes;

**Partition pruning**

Even when you don't name a specific partition in a SQL statement, the fact that a table is partitioned might still influence the manner in which the statement accesses the table. When a SQL statement accesses one or more partitioned tables, the Oracle optimizer attempts to use the information in the WHERE clause to eliminate some of the partitions from consideration during statement execution. This process, called partition pruning, speeds statement execution by ignoring any partitions that cannot satisfy the statement's WHERE clause. To do so, the optimizer uses information from the table definition combined with information from the statement's WHERE clause.

CREATE TABLE tab1 ) col1 NUMBER(5) NOT NULL,

col2 DATE NOT NULL,

col3 VARCHAR2(10) NOT NULL)

PARTITION BY RANGE (col2)

(PARTITION tab1\_1998

VALUES LESS THAN (TO\_DATE('01-JAN-1999','DD-MON-YYYY'))

TABLESPACE t1,

PARTITION tab1\_1999

VALUES LESS THAN (TO\_DATE('01-JAN-2000','DD-MON-YYYY'))

TABLESPACE t1,

PARTITION tab1\_2000

VALUES LESS THAN (TO\_DATE('01-JAN-2001','DD-MON-YYYY'))

TABLESPACE t3,

PARTITION tab1\_2001

VALUES LESS THAN (TO\_DATE('01-JAN-2002','DD-MON-YYYY'))

TABLESPACE t4);

the optimizer would eliminate partitions tab1\_1998 and tab1\_1999 from consideration, since neither partition could contain rows with a value for col2 greater than October 1, 2000.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | Partition pruning is sometimes referred to as partition elimination. | |

For the optimizer to make these types of decisions, the WHERE clause must reference at least one column from the set of columns that comprise the partition key. Although this might seem fairly straightforward, not all queries against a partitioned table naturally include the partition key. If a unique index exists on the col1 column of the tab1 table from the previous example, for instance, the following query would generally offer the most efficient access:

SELECT col1, col2, col3

FROM tab1

WHERE col1 = 1578;

**Selecting records from partitioned tables.**

Select \* from Employee;

Select \* from Employee partition(p1);

**Adding new Table partition:**

Alter table Employee

add partition p5 values less than(50000);

**Drop Table partition:**

Alter table Employee

drop partition p1;

**Rename Table partition:**

Alter table Employee

Rename partition p1 to p6;

**Truncate partition:**

Alter table Employee

Truncate partition p1;

**Split partition:**

Alter table Employee

Split partition p1 at (5000)

into (partition p10,partition p11);

**Moving partition:**

Alter table Employee

move partition p1 to tablespace ABCD;

DL for partiotns

User\_Tab\_partittions