Data Partitioning

#### Implementation of different types of Partitions: Range, Hash, List and composite partitions

▷ As of Oracle8, you can divide the rows of a single table into multiple parts. Dividing a table's data in this manner is called partitioning the table; the table that is partitioned is called a partitioned table, and the parts are called partitions.

▷ Partitioning is useful for very large tables. By splitting a large table's rows across multiple smaller partitions, you accomplish several important goals:

* The performance of queries against the tables may improve, since Oracle may have to only search one partition (one part of the table) instead of the entire table to resolve a query.
* The table may be easier to manage. Since the partitioned table's data is stored in multiple parts, it may be easier to load and delete data in the partitions than in the large table.
* Backup and recovery operations may perform better. Since the partitions are smaller than the partitioned table, you may have more options for backing up and recovering the partitions than you would have for a single large table.

▷ The Oracle optimizer will know that the table has been partitioned;

▷ You can also specify the partition to use as part of the from clause of your queries.

▷ Partitioning enhances the performance, manageability, and availability of a wide variety of applications and helps reduce the total cost of ownership for storing large amounts of data.

▷ Partitioning allows tables, indexes, and index-organized tables to be subdivided into smaller pieces, enabling these database objects to be managed and accessed at a finer level of granularity.

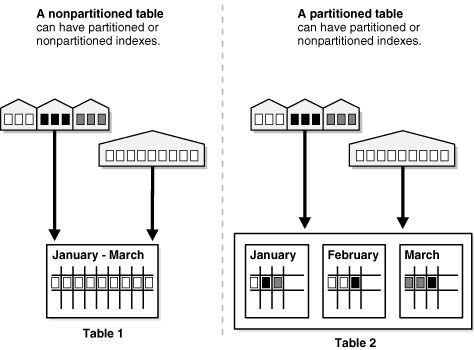
▷ Oracle provides a rich variety of partitioning strategies and extensions to address every business requirement.

▷ Because it is entirely transparent, partitioning can be applied to almost any application without the need for potentially expensive and time consuming application changes.

▷ From the perspective of a database administrator, a partitioned object has multiple pieces that can be managed either collectively or individually.

▷ This gives an administrator considerable flexibility in managing partitioned objects.

▷ However, from the perspective of the application, a partitioned table is identical to a nonpartitioned table; no modifications are necessary when accessing a partitioned table using SQL queries and DML statements.



##### ▷ Partitioning Key

* Each row in a partitioned table is unambiguously assigned to a single partition.
* The partitioning key consists of one or more columns that determine the partition where each row is stored.
* Oracle automatically directs insert, update, and delete operations to the appropriate partition with the partitioning key.

##### ▷ Partitioned Tables

* Any table can be partitioned up to a million separate partitions except those tables containing columns with LONG or LONG RAW data types.
* You can, however, use tables containing columns with CLOB or BLOB data types.

##### ▷ When to Partition a Table

▷ Here are some suggestions for situations when you should consider partitioning a table:

* Tables that are greater than 2 GB.
* These tables should always be considered as candidates for partitioning.
* Tables that contain historical data, in which new data is added into the newest partition.
* A typical example is a historical table where only the current month's data is updatable and the other 11 months are read only.
* Tables whose contents must be distributed across different types of storage devices.

##### ▷ Partitioning Strategies

* Oracle Partitioning offers three fundamental data distribution methods as basic partitioning strategies that control how data is placed into individual partitions:
  + **Range**
  + **Hash**
  + **List**
* Using these data distribution methods, a table can either be partitioned as a single-level or as a composite-partitioned table:
  + **Single-Level Partitioning**
  + **Composite Partitioning**
* Each partitioning strategy has different advantages and design considerations. Thus, each strategy is more appropriate for a particular situation.

#### ▷ Create Partitioning

* To Create a partitioned table,
  + you specify how to setup the partitions of the table’s data as part of the

**create table** command.

* + To partition the table’s records, use partition by clause

# Data Partitioning

### Range Partitioning

11

▷ This type of partitioning is useful when dealing with data that has logical ranges into which it can be distributed;

▷ for example, value of year.

▷ Performance is best when the data evenly distributes across the range.

▷ To partition the table’s records, use partition by range clause of create table command.

▷ When you partition a table or index using the range method, you specify a maximum value for the partitioning key column(s) for each partition.

▷ The ranges will determine the values stored in each partition.

**Syntax:**

CREATE TABLE *table\_name*

(

*column\_name1 data\_type*(*size*), *column\_name2 data\_type*(*size*), *column\_name3 data\_type*(*size*),

....

)

PARTITION BY RANGE (*Column\_name*)

(PARTITION partition\_name VALUES LESS THAN (value1), PARTITION partition\_name VALUES LESS THAN (value2), PARTITION partition\_name VALUES LESS THAN (MAXVALUE)

);

▷ You do not need to specify a maximum value for the last partition; the maxvalue keyword tells Oracle to use the partition to store any data that could not be stored in the earlier partitions.

▷ You can create multiple partitions, each with its own upper value defined.

▷ For each partition, you only specify the maximum value for the range.

▷ The minimum value for the range is implicitly determined by Oracle.

##### For Example:

CREATE TABLE sales (year number(4),product varchar2(10),amt number(10,2)) PARTITION BY RANGE (year)

(PARTITION p1 VALUES LESS THAN (1990), PARTITION p2 VALUES LESS THAN (1993), PARTITION p3 VALUES LESS THAN (1996), PARTITION p4 VALUES LESS THAN (1999), PARTITION p5 VALUES LESS THAN (MAXVALUE));

▷ To select data of table sales

select \* from sales;

▷ To display the partition names using User\_tab\_partitions (data dictionary table)

select PARTITION\_NAME from USER\_TAB\_PARTITIONS where TABLE\_NAME = 'SALES’;

▷ To display the partition table details using user\_part\_tables (data dictionary table)

select TABLE\_NAME, PARTITIONING\_TYPE, STATUS from USER\_PART\_TABLES where TABLE\_NAME = 'SALES';

▷ To select data from a particular partition

select \* from sales partition(p1);

# Data Partitioning

### List Partitioning

18

▷ 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.

▷ Use list partitioning when you require explicit control over how rows map to partitions.

▷ In List partitioning, you tell Oracle all of the possible values, and designate the partitions into which the corresponding rows should be inserted.

▷ List partitioning allows unordered and unrelated sets of data to be grouped and organized together very naturally.

**Syntax:** CREATE TABLE *table\_name*

(

*column\_name1 data\_type*(*size*), *column\_name2 data\_type*(*size*), *column\_name3 data\_type*(*size*),

....

)

PARTITION BY LIST (*Column\_name*)

(PARTITION partition\_name VALUES (value1,value2,…), PARTITION partition\_name VALUES (value1,value2,…), PARTITION partition\_name VALUES (value1,value2,…) PARTITION partition\_name VALUES (DEFAULT);

);

CREATE TABLE customers(custcode number(5), cust\_name varchar2(20),

Address varchar2(10), City varchar2(20),

Bal number(10)) partition by list(City)

(partition north values ('DELHI','CHANDIGARH'), partition east values ('KOLKOTA','PATNA'),

partition south values ('HYDERABAD','BANGALORE','CHENNAI'), partition west values ('BOMBAY','GOA'));

▷ If a row is inserted in the above table then oracle maps the value of city column and whichever partition list matches the city column, the row is stored in that partition.

▷ Available only with list partitioning, you can use the keyword DEFAULT to describe the value list for a partition.

▷ This identifies a partition that will accommodate rows that do not map into any of the other partitions.

▷ To select data of table customers

* select \* from customers;

▷ To select data from a particular partition

* select \* from customers partition(east);

▷ To display the partition names using User\_tab\_partitions (data dictionary table)

* select PARTITION\_NAME from USER\_TAB\_PARTITIONS where TABLE\_NAME = 'CUSTOMERS’;

▷ To display the partition table details using user\_part\_tables (data dictionary table)

* SELECT TABLE\_NAME, PARTITIONING\_TYPE, STATUS FROM USER\_PART\_TABLES WHERE TABLE\_NAME = 'CUSTOMERS';

▷ From the perspective of the application, a partitioned table is identical to a nonpartitioned table; no modifications are necessary when accessing a partitioned table using SQL queries and DML statements.

SELECT \* FROM customers where City='BOMBAY';

# Data Partitioning

**Hash Partitioning**

25

▷ Hash partitioning provides a method of evenly distributing data across a specified number of partitions.

▷ Rows are mapped into partitions based on a hash value of the partitioning key.

▷ In Hash partitioning, consecutive values of the partition key are not necessarily stored in the same partition.

▷ It distributes a set of records over a greater set of partitions, thus reducing I/O contention.

##### Syntax:

CREATE TABLE *table\_name*

(

*column\_name1 data\_type*(*size*), *column\_name2 data\_type*(*size*), *column\_name3 data\_type*(*size*),

....

)

PARTITION BY HASH (*Column\_name*) PARTITIONS partition\_number;

You can also name each partition as shown below:

**Syntax:** CREATE TABLE *table\_name*

(

*column\_name1 data\_type*(*size*), *column\_name2 data\_type*(*size*), *column\_name3 data\_type*(*size*),

....

)

PARTITION BY HASH (*Column\_name*) (PARTITION partition\_name, PARTITION partition\_name, PARTITION partition\_name

);

##### For Example:

CREATE TABLE products (partno NUMBER, description VARCHAR2 (60))

PARTITION BY HASH (partno)

(PARTITION part1, PARTITION part2, PARTITION part3);

▷ To select data of table products

* select \* from products;

▷ To select data from a particular partition

* select \* from products partition(part1);

▷ To display the partition names using User\_tab\_partitions (data dictionary table)

* select PARTITION\_NAME from USER\_TAB\_PARTITIONS where TABLE\_NAME = 'PRODUCTS';

▷ To display the partition table details using user\_part\_tables (data dictionary table)

* SELECT TABLE\_NAME, PARTITIONING\_TYPE, STATUS FROM USER\_PART\_TABLES WHERE TABLE\_NAME = 'PRODUCTS';

# Altering and Managing Partition Tables

31

▷ You can use the alter table command to add, drop, exchange, move, modify, rename, split, and truncate partitions.

▷ These alter table command options allow you to alter the existing partition structure, as may be required after a partitioned table has been used heavily.

▷ To add a partition

* You can add a new partition to the "high" end (the point after the last existing partition).
* To add a partition at the beginning or in the middle of a table, use the SPLIT PARTITION clause.
* Example: - To add a partition to sales table give the following command. ALTER TABLE sales ADD PARTITION p6 VALUES LESS THAN (2005);

(Note- this query will not work if last partition is defined as MAXVALUE)

* Example: - To add a partition to a Hash Partition table give the following command. ALTER TABLE products ADD PARTITION;
* Then Oracle adds a new partition whose name is system generated and it is created

in the default tablespace.

* To add a partition by user defined name give the following command. ALTER TABLE products ADD PARTITION part4;
* Example: - To add a partition to a List partition table give the following command. ALTER TABLE customers ADD PARTITION central\_India VALUES ('BHOPAL','NAGPUR’); (Note: Cannot add partition when DEFAULT partition exists)
* Any value in the set of literal values that describe the partition(s) being added must not exist in any of the other partitions of the table.

▷ To Merge Partitions

* Use the ALTER TABLE ... MERGE PARTITIONS statement to merge the contents of two partitions into one partition.
* The two original partitions are dropped, as are any corresponding local indexes.
* You cannot use this statement for a hash-partitioned table or for hash sub partitions of a composite-partitioned table.
* You can only merge two adjacent partitions; you cannot merge nonadjacent partitions.
* Example: - To merge the partition p2 and p3 into one partition p23 give the following statement.

ALTER TABLE sales MERGE PARTITIONS p2, p3 INTO PARTITION p23;

▷ Modifying Partitions: Adding Values

* + Use the MODIFY PARTITION ... ADD VALUES clause of the ALTER TABLE statement to extend the value list of an existing partition.
  + Literal values being added must not have been included in any other partition's value list.
  + The following statement adds a new set of cities ('KOCHI', 'MANGALORE') to an existing partition list.

ALTER TABLE customers MODIFY PARTITION south

ADD VALUES ('KOCHI', 'MANGALORE');

▷ Modifying Partitions: Dropping Values

* + Use the MODIFY PARTITION ... DROP VALUES clause of the ALTER TABLE statement to remove literal values from the value list of an existing partition.
  + The statement is always executed with validation, meaning that it checks to see if any rows exist in the partition that corresponds to the set of values being dropped.
  + If any such rows are found, then Oracle returns an error message, and the operation fails.
  + When necessary, use a DELETE statement to delete corresponding rows from the table before attempting to drop values.
  + You cannot drop all literal values from the value list describing the partition. You must use the ALTER TABLE ... DROP PARTITION statement instead.
  + The partition value list for any corresponding local index partition reflects the new value list, and any global index, or global or local index partitions, remain usable.
  + For Example, the statement below drops a set of cities (‘CHENNAI' and 'MANGALORE') from an existing partition value list.

ALTER TABLE customers MODIFY PARTITION south DROP VALUES ('CHENNAI', 'MANGALORE');

▷ Splitting Partitions

* + The SPLIT PARTITION clause of the ALTER TABLE or ALTER INDEX statement is used to redistribute the contents of a partition into two new partitions.
  + Consider doing this when a partition becomes too large and causes backup, recovery, or maintenance operations to take a long time to complete or it is felt that there is simply too much data in the partition.
  + You can also use the SPLIT PARTITION clause to redistribute the I/O load.
  + This clause cannot be used for hash partitions or subpartitions.
  + You split a range partition using the ALTER TABLE ... SPLIT PARTITION statement.
  + To split a range partition into N partitions, (N-1) values of the partitioning key column must be specified within the range of the partition at which to split the partition.
  + The first of the resulting two new partitions includes all rows in the original partition whose partitioning key column values map lower that the specified value.
  + The second partition contains all rows whose partitioning key column values map greater than or equal to the specified value.
  + The new non-inclusive upper bound values specified must be in ascending order. The high bound of Nth new partition is assigned the value of the high bound of the partition being split.
  + The names and physical attributes of the N new partitions resulting from the split can be optionally specified.
  + For example, first create table price and partition it as follows

CREATE TABLE PRICE (year number(4),product varchar2(10),amt number(10,2)) PARTITION BY RANGE (year)

(PARTITION p1 VALUES LESS THAN (1990), PARTITION p2 VALUES LESS THAN (1995), PARTITION p3 VALUES LESS THAN (2000), PARTITION p4 VALUES LESS THAN (2005));

* + Split partition p2 into two partitions p21,p22 at value 1993

ALTER TABLE PRICE SPLIT PARTITION p2 AT (1993) INTO (PARTITION p21, PARTITION p22); select PARTITION\_NAME from USER\_TAB\_PARTITIONS where TABLE\_NAME = 'PRICE';

▷ To split a list partition into N partitions, (N-1) lists of literal values must be specified, each of which defines the first (N-1) partitions into which rows with corresponding partitioning key values are inserted.

▷ The remaining rows of the original partition are inserted into the Nth new partition whose value list contains the remaining literal values from the original partition.

▷ No two value lists can contain the same partition value.

▷ The (N-1) value lists that are specified cannot contain all of the partition values of the current partition because the Nth new partition would be empty.

▷ Also, the (N-1) value lists cannot contain any partition values that do not exist for the current partition.

▷ For example, the following query will split the west partition into 2 partitions ‘northwest’ and ‘southwest’

ALTER TABLE customers SPLIT PARTITION west INTO (PARTITION northwest VALUES ('BOMBAY'), PARTITION southwest VALUES ('GOA'),

PARTITION restofthewest);

▷ Dropping Partitioned Tables

* To drop a partition from Range Partition table, List Partition or Composite Partition table give the following command.

ALTER TABLE sales DROP PARTITION p5;

* You can also first delete all the records and then drop the partition like this

DELETE FROM sales WHERE year=1994; ALTER TABLE sales DROP PARTITION p4;

* This method is most appropriate for small tables or for large tables when the partition being dropped contains a small percentage of the total data in the table.

## Composite partitioning

▷ Composite partitioning partitions data using the range method, and within each partition, subpartitions it using the hash or list method.

▷ Composite range-hash partitioning provides the improved manageability of range partitioning and the data placement, striping, and parallelism advantages of hash partitioning.

▷ Composite range-list partitioning provides the manageability of range partitioning and the explicit control of list partitioning for the subpartitions.

▷ Composite partitioning supports historical operations, such as adding new range partitions, but also provides higher degrees of parallelism for DML operations and finer granularity of data placement through subpartitioning.

▷ Composite partitioning allows range partitions to be hash subpartitioned on a different key.

▷ The greater number of partitions increases the possiblities for parallelism and reduces the chances of contention.

▷ The following example will range partition the table on invoice\_date and subpartitioned these on the invoice\_no giving a totol of 32 subpartitions.

CREATE TABLE invoices (invoice\_no NUMBER NOT NULL, invoice\_date DATE NOT NULL, comments VARCHAR2(100))

PARTITION BY RANGE (invoice\_date) SUBPARTITION BY HASH (invoice\_no)

SUBPARTITIONS 8

(PARTITION invoices\_q1 VALUES LESS THAN (TO\_DATE('01/04/2001', 'DD/MM/YYYY')), PARTITION invoices\_q2 VALUES LESS THAN (TO\_DATE('01/07/2001', 'DD/MM/YYYY')), PARTITION invoices\_q3 VALUES LESS THAN (TO\_DATE('01/09/2001', 'DD/MM/YYYY')), PARTITION invoices\_q4 VALUES LESS THAN (TO\_DATE('01/01/2002', 'DD/MM/YYYY')));

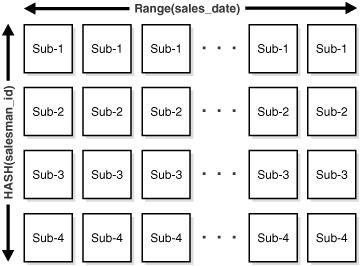
CREATE TABLE sales\_composite (salesman\_id NUMBER(5), salesman\_name VARCHAR2(30), sales\_amount NUMBER(10), sales\_date DATE)

PARTITION BY RANGE(sales\_date) SUBPARTITION BY HASH(salesman\_id) SUBPARTITION TEMPLATE( SUBPARTITION sp1,

SUBPARTITION sp2, SUBPARTITION sp3, SUBPARTITION sp4)

Command continued on next slide

(PARTITION sales\_jan2000 VALUES LESS THAN(TO\_DATE('02/01/2000','MM/DD/YYYY')), PARTITION sales\_feb2000 VALUES LESS THAN(TO\_DATE('03/01/2000','MM/DD/YYYY')), PARTITION sales\_mar2000 VALUES LESS THAN(TO\_DATE('04/01/2000','MM/DD/YYYY')), PARTITION sales\_apr2000 VALUES LESS THAN(TO\_DATE('05/01/2000','MM/DD/YYYY')), PARTITION sales\_may2000 VALUES LESS THAN(TO\_DATE('06/01/2000','MM/DD/YYYY')));



CREATE TABLE bimonthly\_regional\_sales (deptno NUMBER,

item\_no VARCHAR2(20), txn\_date DATE, txn\_amount NUMBER, state VARCHAR2(2))

PARTITION BY RANGE (txn\_date) SUBPARTITION BY LIST (state) SUBPARTITION TEMPLATE(

SUBPARTITION east VALUES('NY', 'VA', 'FL'), SUBPARTITION west VALUES('CA', 'OR', 'HI'), SUBPARTITION central VALUES('IL', 'TX', 'MO’))

When you use a template, Oracle names the subpartitions by concatenating the partition name, an underscore, and the subpartition name from the template.

Oracle places this subpartition in the tablespace specified in the template. In the previous statement, sales\_jan2000\_sp1 is created

Command continued on next slide

(

PARTITION janfeb\_2000 VALUES LESS THAN (TO\_DATE('1-MAR-2000','DD-MON-YYYY')), PARTITION marapr\_2000 VALUES LESS THAN (TO\_DATE('1-MAY-2000','DD-MON-YYYY')), PARTITION mayjun\_2000 VALUES LESS THAN (TO\_DATE('1-JUL-2000','DD-MON-YYYY'))

);

SELECT PARTITION\_NAME from USER\_TAB\_PARTITIONS where TABLE\_NAME = 'BIMONTHLY\_REGIONAL\_SALES’;

SELECT SUBPARTITION\_NAME from USER\_TAB\_SUBPARTITIONS where TABLE\_NAME = 'BIMONTHLY\_REGIONAL\_SALES';

