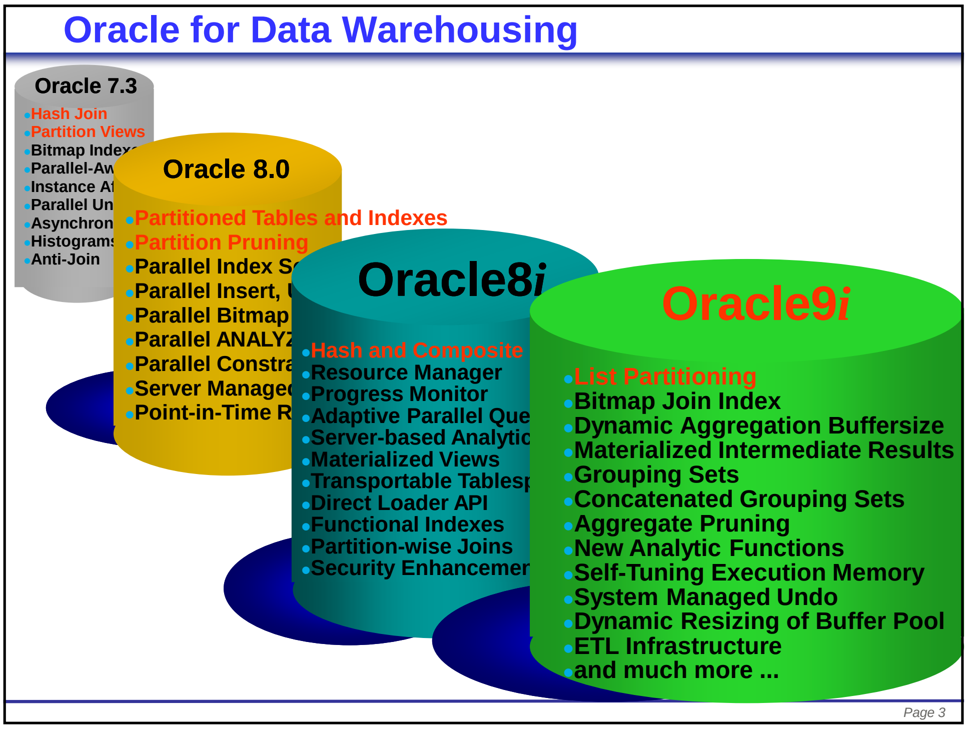
Oracle Partitioning – part 1

- **Why partitioning**
- **When to partition tables**
- **Evolution of partitioning in Oracle**
- **5 Table partitioning methods**
  - Range Partitioning
  - Hash Partitioning
  - List Partitioning
  - Composite Range-Hash Partitioning
  - Composite Range-List Partitioning
- **Creating partition tables**
- **Managing partition tables**
- **Lab: Partitioning**

Partition Overview

- **Oracle Partitioning, an option of OracleDB Enterprise Edition, can enhance the manageability, performance, and availability of a wide variety of applications.**
- **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 schemes to address every business requirement.**
- **It is entirely transparent to SQL statements, partitioning can be applied to almost any application.**
- **Partitioning is useful for many different types of applications, particularly applications that manage large volumes of data.**
  - OLTP systems often benefit from improvements in manageability and availability.
  - Data warehousing systems benefit from performance and manageability.

Oracle for Data Warehousing



Oracle 7.3

- Hash Join
- Partition Views
- Bitmap Index
- Parallel-Aw
- Instance Af
- Parallel Un
- Asynchron
- Histograms
- Anti-Join

Oracle 8.0

- Partitioned Tables and Indexes
- Partition Pruning
- Parallel Index Sc
- Parallel Insert, U
- Parallel Bitmap
- Parallel ANALYZ
- Parallel Constra
- Server Managed
- Point-in-Time R

Oracle8*i*

- Hash and Composite
- Resource Manager
- Progress Monitor
- Adaptive Parallel Que
- Server-based Analytic
- Materialized Views
- Transportable Tablesp
- Direct Loader API
- Functional Indexes
- Partition-wise Joins
- Security Enhancemen

Oracle9*i*

- List Partitioning
- Bitmap Join Index
- Dynamic Aggregation Buffersize
- Materialized Intermediate Results
- Grouping Sets
- Concatenated Grouping Sets
- Aggregate Pruning
- New Analytic Functions
- Self-Tuning Execution Memory
- System Managed Undo
- Dynamic Resizing of Buffer Pool
- ETL Infrastructure
- and much more ...

Why Partitioning

The concept of Divide and Conquer.

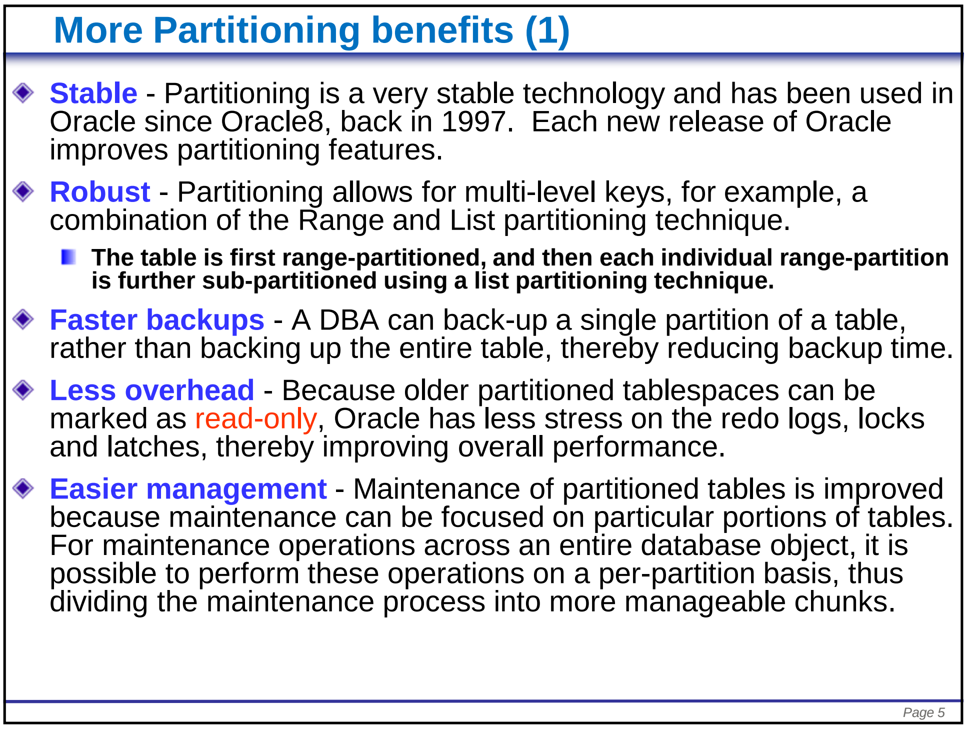
- "Dividing Tables and Indexes into manageable pieces."
- The Best Thing Since ***Sliced Bread***

Benefits

- Each partition is stored in its own segment and can be managed individually.
- It can function independently of the other partitions, thus providing a structure that can be better tuned for availability and performance.
- Operations on partitioned tables and indexes are performed in **parallel** by assigning different parallel execution servers.
- Partitioning is **transparent** to existing applications and standard DML statements run against partitioned tables. An application can be programmed to take advantage of partitioning by using partitioned table or index names in DML.
- You can use the **SQL*Loader, Import, and Export utilities** to load or unload data stored in partitioned tables. These utilities are all partition and subpartition aware.

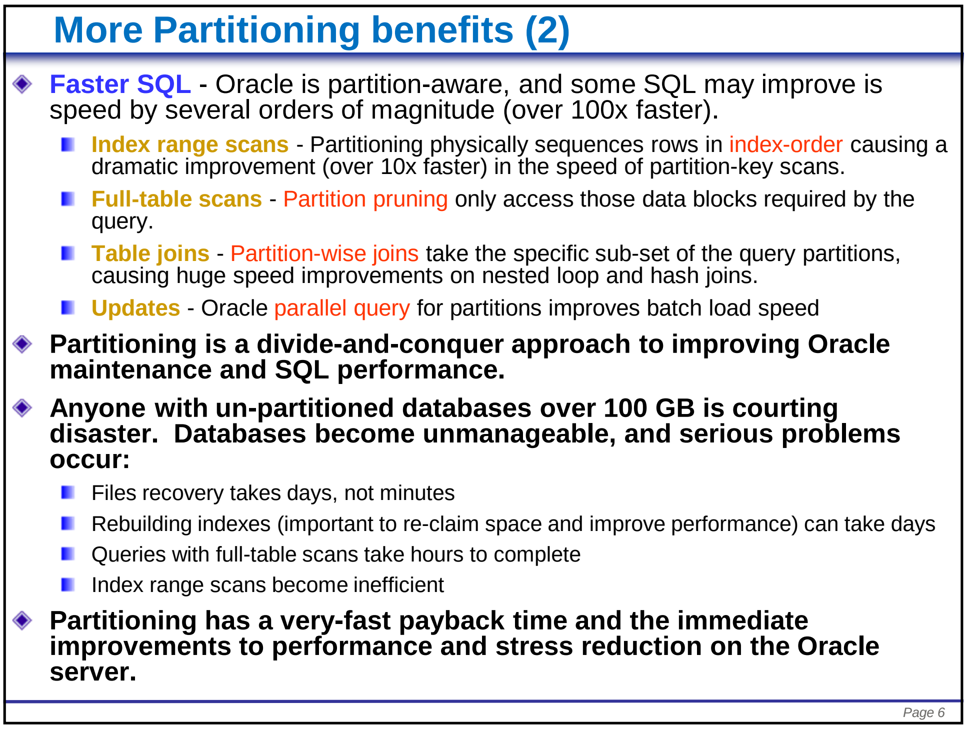
Storing partitions in separate tablespaces enables you to:

- Reduce the possibility of data corruption in multiple partitions
- Back up and recover each partition independently
- Control the mapping of partitions to disk drives (important for balancing I/O load)
- Improve manageability, availability, and performance



More Partitioning benefits (1)

- **Stable** - Partitioning is a very stable technology and has been used in Oracle since Oracle8, back in 1997. Each new release of Oracle improves partitioning features.
- **Robust** - Partitioning allows for multi-level keys, for example, a combination of the Range and List partitioning technique.
  - **The table is first range-partitioned, and then each individual range-partition is further sub-partitioned using a list partitioning technique.**
- **Faster backups** - A DBA can back-up a single partition of a table, rather than backing up the entire table, thereby reducing backup time.
- **Less overhead** - Because older partitioned tablespaces can be marked as read-only, Oracle has less stress on the redo logs, locks and latches, thereby improving overall performance.
- **Easier management** - Maintenance of partitioned tables is improved because maintenance can be focused on particular portions of tables. For maintenance operations across an entire database object, it is possible to perform these operations on a per-partition basis, thus dividing the maintenance process into more manageable chunks.



More Partitioning benefits (2)

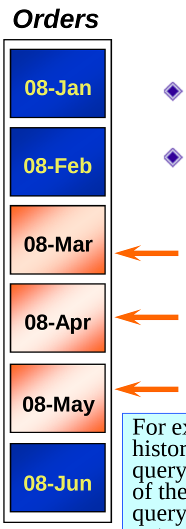
- **Faster SQL** - Oracle is partition-aware, and some SQL may improve is speed by several orders of magnitude (over 100x faster).
  - **Index range scans** - Partitioning physically sequences rows in index-order causing a dramatic improvement (over 10x faster) in the speed of partition-key scans.
  - **Full-table scans** - Partition pruning only access those data blocks required by the query.
  - **Table joins** - Partition-wise joins take the specific sub-set of the query partitions, causing huge speed improvements on nested loop and hash joins.
  - **Updates** - Oracle parallel query for partitions improves batch load speed
- **Partitioning is a divide-and-conquer approach to improving Oracle maintenance and SQL performance.**
- **Anyone with un-partitioned databases over 100 GB is courting disaster. Databases become unmanageable, and serious problems occur:**
  - Files recovery takes days, not minutes
  - Rebuilding indexes (important to re-claim space and improve performance) can take days
  - Queries with full-table scans take hours to complete
  - Index range scans become inefficient
- **Partitioning has a very-fast payback time and the immediate improvements to performance and stress reduction on the Oracle server.**

When To Partition A Table

- There are two main reasons to use partitioning in a VLDB environment. These reasons are related to management and performance improvement.
- **Partitioning offers:**
  - **Management** at the individual partition level for data loads, index creation and rebuilding, and backup/recovery. This can result in less down time because only individual partitions being actively managed are unavailable.
  - Increased query **performance** by selecting only from the relevant partitions. This weeding out process eliminates the partitions that do not contain the data needed by the query through a technique called **partition pruning**.
- **The decision about exactly when to use partitioning is rather subjective. Here are some suggestions and general guidelines for when to partition a table:**
  - When a table reaches a **"large"** size. Large being defined relative to your environment. Tables greater than 100 GB should always be considered for partitioning.
  - Tables containing **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.
  - When the contents of a table need to be distributed across different types of **storage** devices.
  - When **performance** benefits outweigh the additional management issues related to partitioning.

Partition Pruning

- **Internal mechanism to reduce the data to be accessed to a minimum**
  - Only relevant partitions will be accessed
  - Order-of-magnitude performance gains on many queries
  - Especially useful for Range, Composite, and List partitioning
- **Transparent to any application**
  - Oracle analyzes the SQL statement
- **Two flavors of Pruning**
  - Static Pruning: use static predicates
  - Dynamic Pruning: if pruning is possible and static pruning is <u>not</u> possible



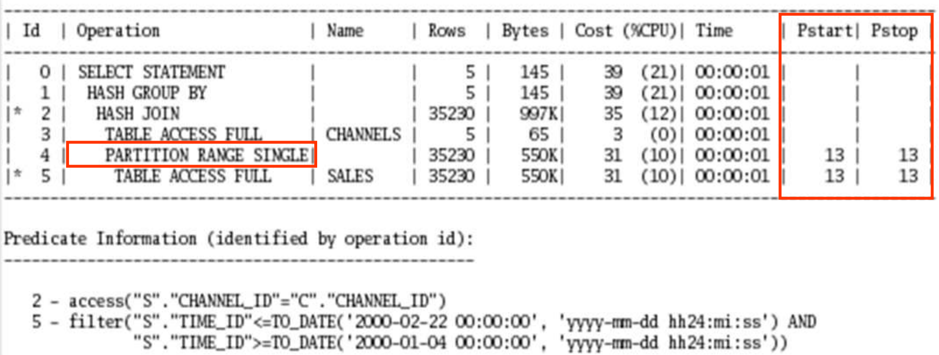
```
SELECT sum(sales_amount)
  FROM sales
  WHERE sales_date BETWEEN ‘01-MAR-2008’
        AND ‘31-MAY-2008’;
```

For example, suppose an application contains an Orders table containing a historical record of orders, and that this table has been partitioned by month. A query requesting orders for a single month would only access a single partition of the Orders table. If the Orders table had 2 years of historical data, then this query would access one partition instead of 24 partitions. This query could potentially execute **20 times faster** simply because of partition pruning.

Partition pruning

- Dynamic partition pruning can be as simple as a bind variable replacement at runtime or as complex as spawning additional recursive SQL to identify the appropriate partitions. Dynamic partition pruning takes place at query run time.
- Static partition pruning takes place when the optimizer can eliminate specific partitions at parse time. For example a query predicate on the partitioning key column.

```
SELECT c.channel_desc, sum(amount_sold)
FROM sales s, channels c
WHERE s.channel_id = c.channel_id
AND s.time_id >= to_date('04-JAN-2000','DD-MON-YYYY')
AND s.time_id <= to_date('22-FEB-2000','DD-MON-YYYY')
GROUP BY channel_desc;
```



```
---------------------------------------------------------------------------------------------------
| Id  | Operation                 | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
---------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |          |     5 |   145 |    39  (21)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY            |          |     5 |   145 |    39  (21)| 00:00:01 |       |       |
|*  2 |   HASH JOIN               |          | 35230 |   997K|    35  (12)| 00:00:01 |       |       |
|   3 |    TABLE ACCESS FULL      | CHANNELS |     5 |    65 |     3   (0)| 00:00:01 |       |       |
|   4 |    PARTITION RANGE SINGLE |          | 35230 |   550K|    31  (10)| 00:00:01 |    13 |    13 |
|*  5 |     TABLE ACCESS FULL     | SALES    | 35230 |   550K|    31  (10)| 00:00:01 |    13 |    13 |
---------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("S"."CHANNEL_ID"="C"."CHANNEL_ID")
   5 - filter("S"."TIME_ID"<=TO_DATE('2000-02-22 00:00:00', 'yyyy-mm-dd hh24:mi:ss') AND
              "S"."TIME_ID">=TO_DATE('2000-01-04 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))
```

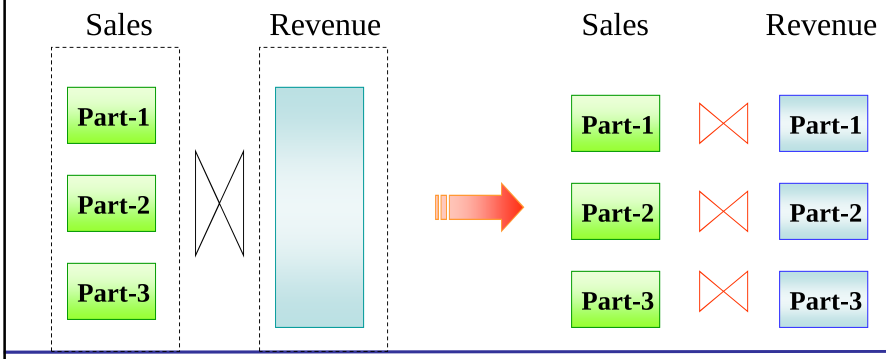
Partition-wise Join

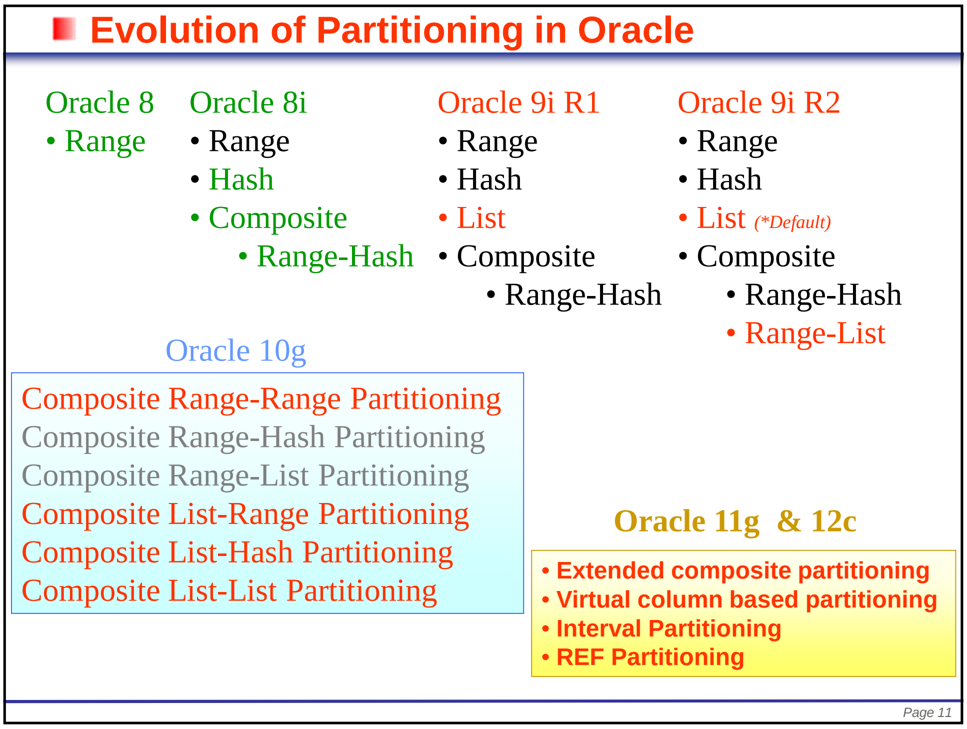
- Partition-wise joins can be applied when two tables are being joined together and both tables are partitioned on the join key, or when a reference partitioned table is joined with its parent table.
- Partition-wise joins break a large join into smaller joins that occur between each of the partitions, completing the overall join in less time.
- Significant performance benefits both for serial and parallel execution.
- If Sales table is partitioned by the join key, then Revenue table can be re-distributed to enable partition-wise join.

SALES partitioned on SALES_DATE

REVENUE partitioned on BOOKED_DATE

SELECT … FROM SALES S, REVENUE R WHERE S.SALES_DATE = R.BOOKED_DATE





Evolution of Partitioning in Oracle

Oracle 8
- Range

Oracle 8i
- Range
- Hash
- Composite
  - Range-Hash

Oracle 9i R1
- Range
- Hash
- List
- Composite
  - Range-Hash

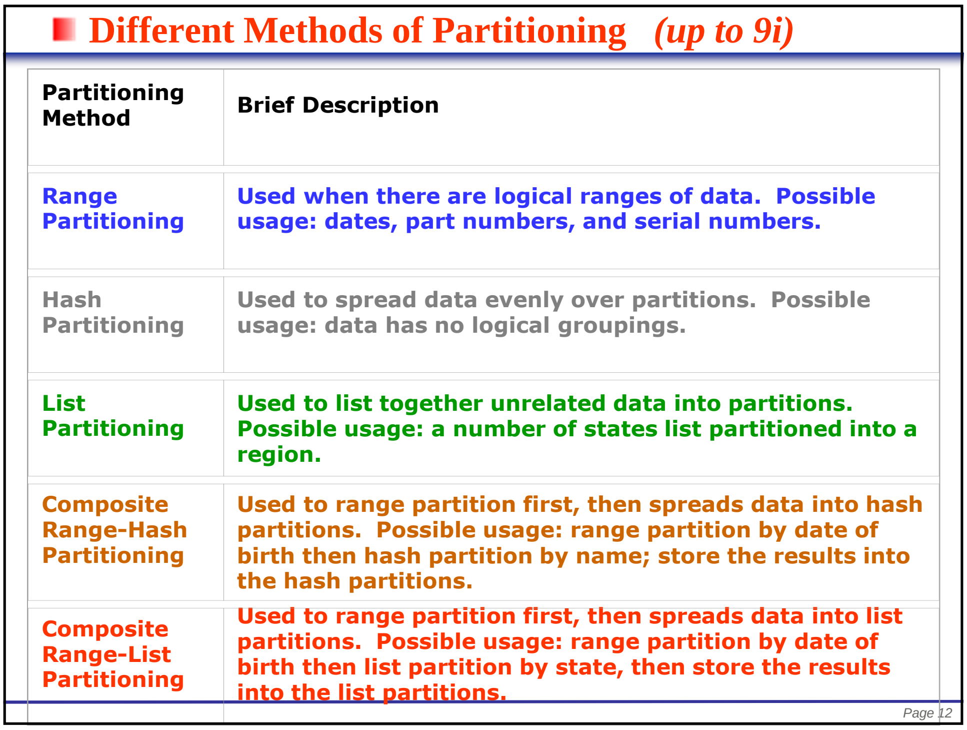
Oracle 9i R2
- Range
- Hash
- List *(*Default)*
- Composite
  - Range-Hash
  - Range-List

Oracle 10g

Composite Range-Range Partitioning
Composite Range-Hash Partitioning
Composite Range-List Partitioning
Composite List-Range Partitioning
Composite List-Hash Partitioning
Composite List-List Partitioning

Oracle 11g & 12c

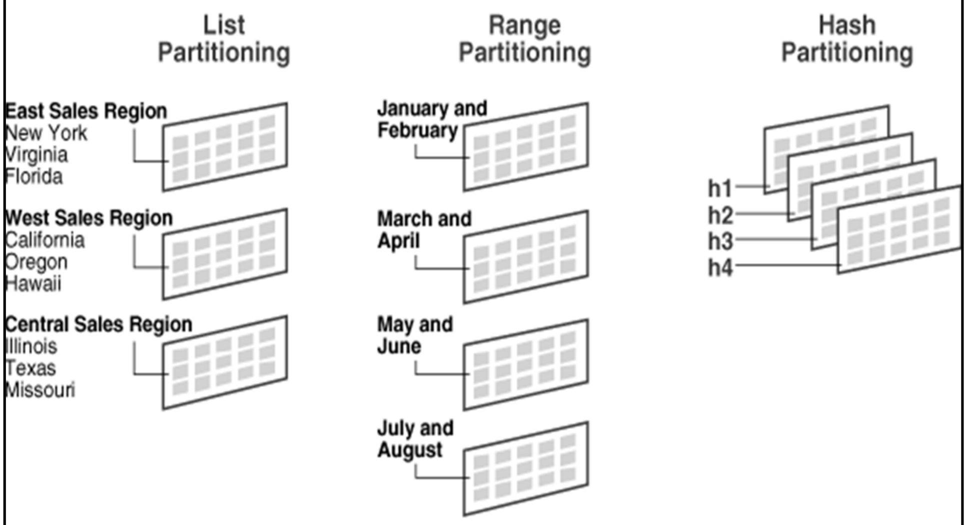
- **Extended composite partitioning**
- **Virtual column based partitioning**
- **Interval Partitioning**
- **REF Partitioning**



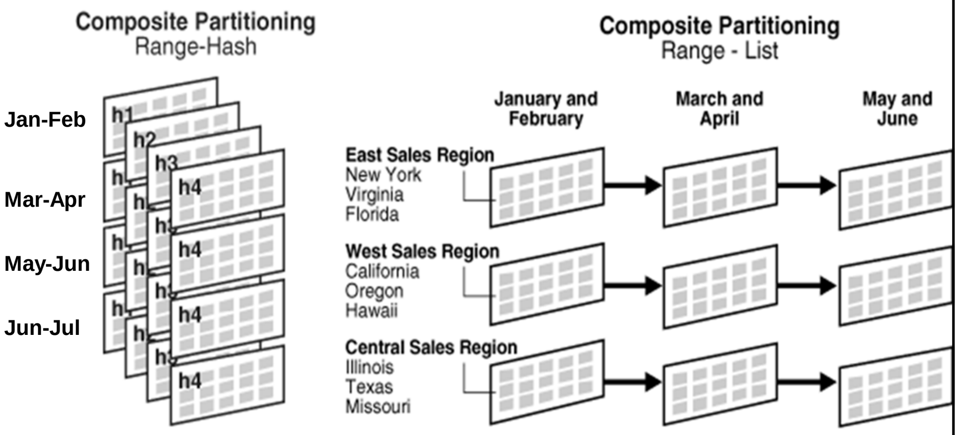
Different Methods of Partitioning *(up to 9i)*

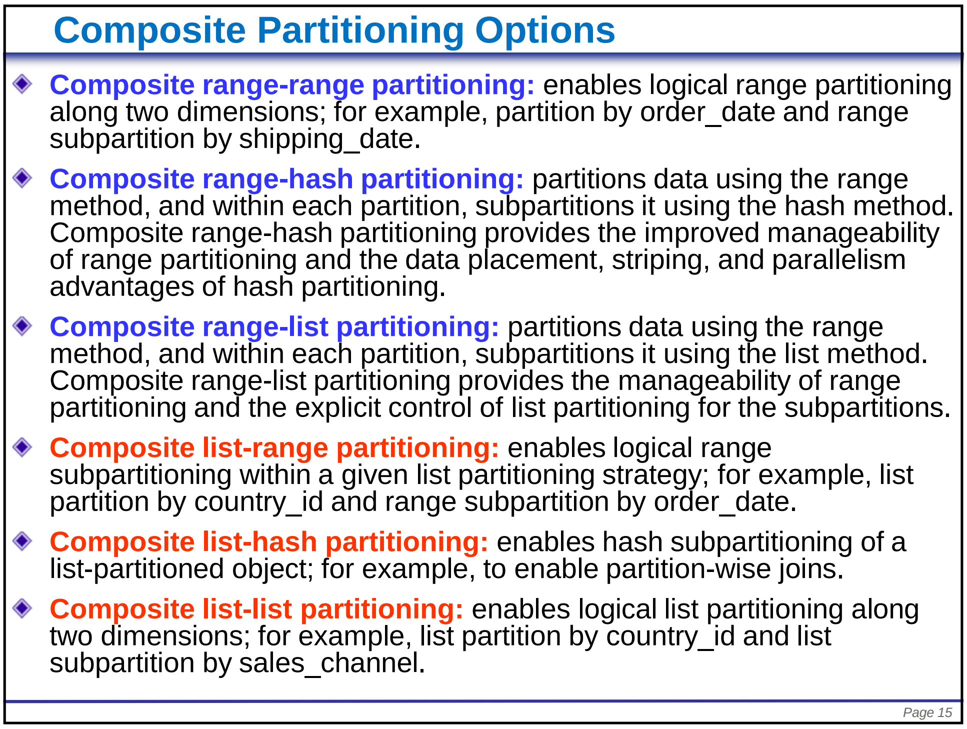
Partitioning Method	**Brief Description**
Range Partitioning	**Used when there are logical ranges of data. Possible usage: dates, part numbers, and serial numbers.**
Hash Partitioning	**Used to spread data evenly over partitions. Possible usage: data has no logical groupings.**
List Partitioning	**Used to list together unrelated data into partitions. Possible usage: a number of states list partitioned into a region.**
Composite Range-Hash Partitioning	**Used to range partition first, then spreads data into hash partitions. Possible usage: range partition by date of birth then hash partition by name; store the results into the hash partitions.**
Composite Range-List Partitioning	**Used to range partition first, then spreads data into list partitions. Possible usage: range partition by date of birth then list partition by state, then store the results into the list partitions.**

Single-level Partitioning



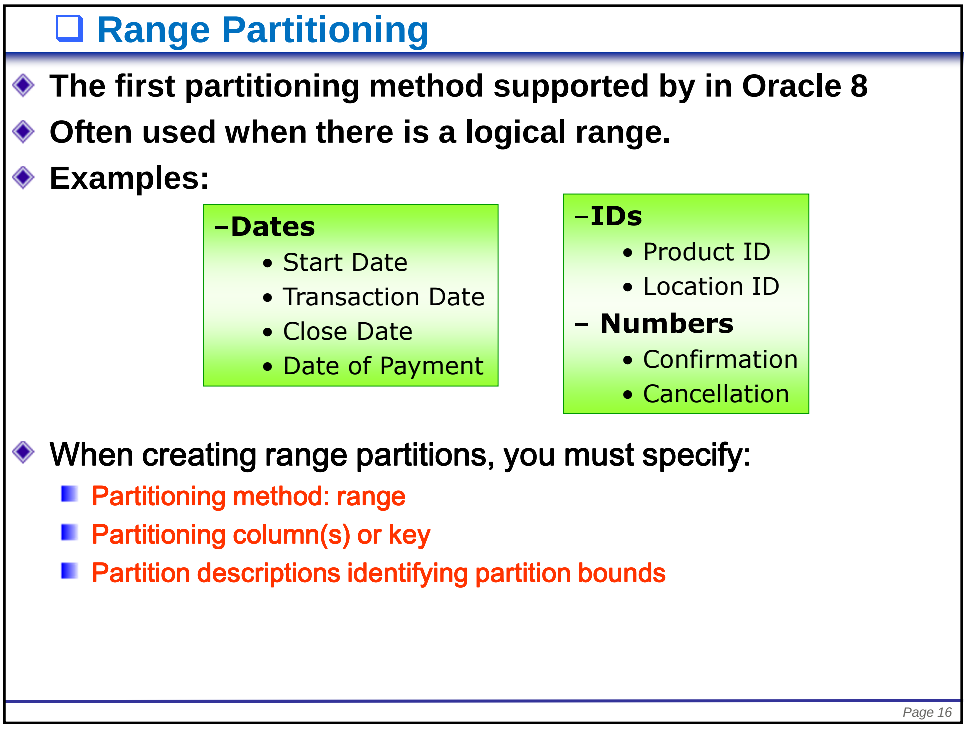
Composite Partitioning





Composite Partitioning Options

- **Composite range-range partitioning:** enables logical range partitioning along two dimensions; for example, partition by order_date and range subpartition by shipping_date.
- **Composite range-hash partitioning:** partitions data using the range method, and within each partition, subpartitions it using the hash 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:** partitions data using the range method, and within each partition, subpartitions it using the list method. Composite range-list partitioning provides the manageability of range partitioning and the explicit control of list partitioning for the subpartitions.
- **Composite list-range partitioning:** enables logical range subpartitioning within a given list partitioning strategy; for example, list partition by country_id and range subpartition by order_date.
- **Composite list-hash partitioning:** enables hash subpartitioning of a list-partitioned object; for example, to enable partition-wise joins.
- **Composite list-list partitioning:** enables logical list partitioning along two dimensions; for example, list partition by country_id and list subpartition by sales_channel.



❑ Range Partitioning

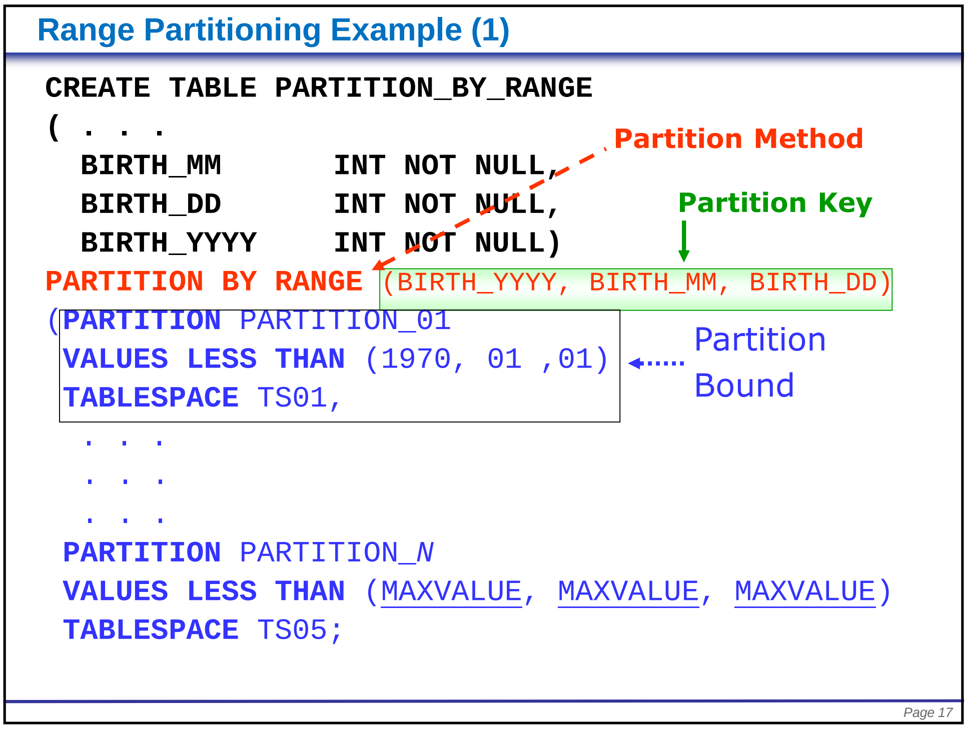
- **The first partitioning method supported by in Oracle 8**
- **Often used when there is a logical range.**
- **Examples:**

–Dates
- Start Date
- Transaction Date
- Close Date
- Date of Payment

–IDs
- Product ID
- Location ID

– Numbers
- Confirmation
- Cancellation

- When creating range partitions, you must specify:
  - Partitioning method: range
  - Partitioning column(s) or key
  - Partition descriptions identifying partition bounds



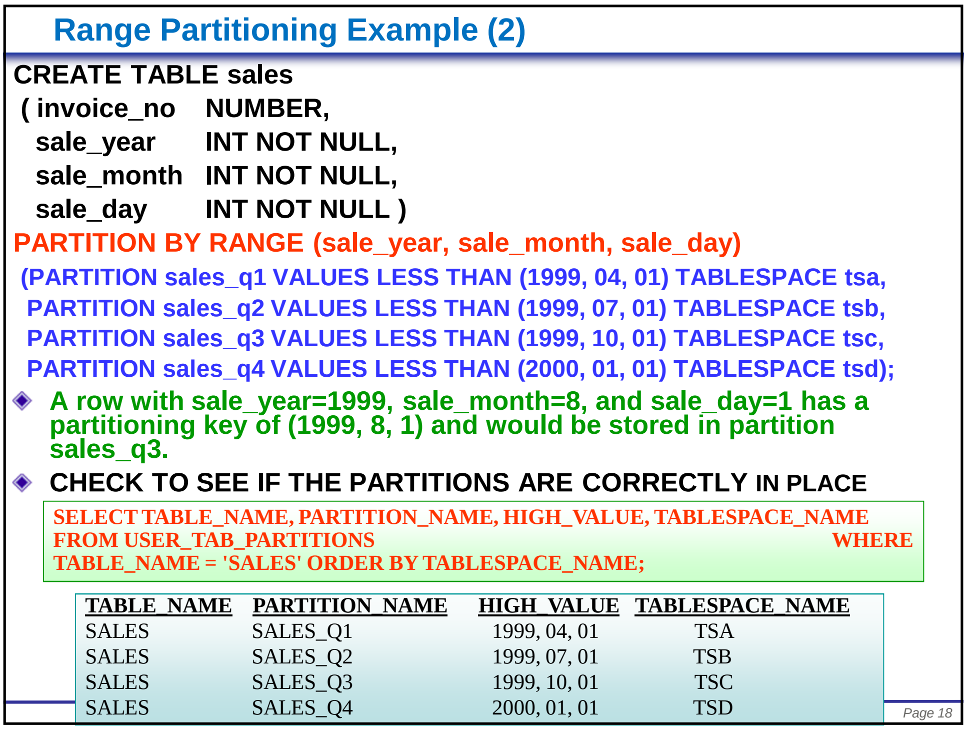
Range Partitioning Example (1)

```
CREATE TABLE PARTITION_BY_RANGE
( . . .
  BIRTH_MM       INT NOT NULL,
  BIRTH_DD       INT NOT NULL,
  BIRTH_YYYY     INT NOT NULL)
PARTITION BY RANGE (BIRTH_YYYY, BIRTH_MM, BIRTH_DD)
(PARTITION PARTITION_01
 VALUES LESS THAN (1970, 01 ,01)
 TABLESPACE TS01,
  . . .
  . . .
  . . .
 PARTITION PARTITION_N
 VALUES LESS THAN (MAXVALUE, MAXVALUE, MAXVALUE)
 TABLESPACE TS05;
```

Partition Method

Partition Key

Partition Bound



Range Partitioning Example (2)

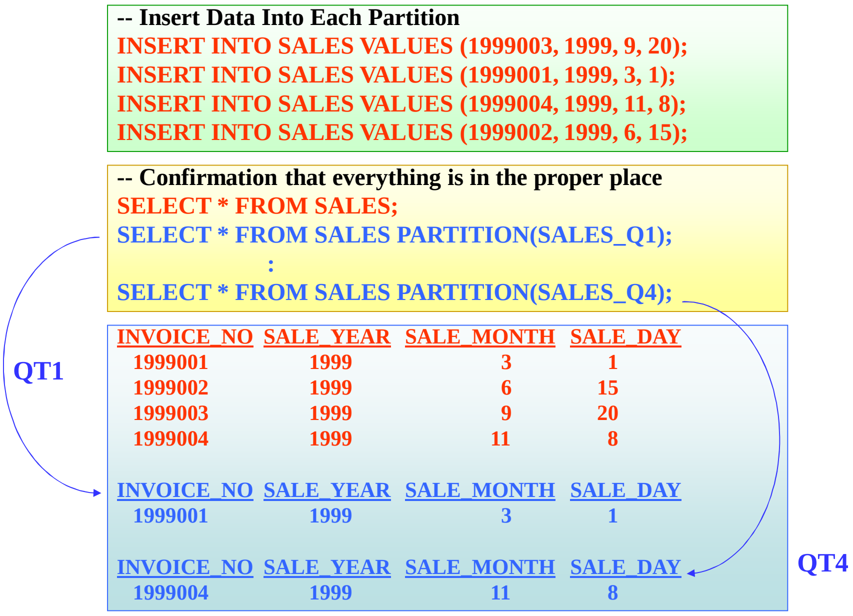
```
CREATE TABLE sales
 ( invoice_no    NUMBER,
   sale_year     INT NOT NULL,
   sale_month    INT NOT NULL,
   sale_day      INT NOT NULL )
PARTITION BY RANGE (sale_year, sale_month, sale_day)
 (PARTITION sales_q1 VALUES LESS THAN (1999, 04, 01) TABLESPACE tsa,
  PARTITION sales_q2 VALUES LESS THAN (1999, 07, 01) TABLESPACE tsb,
  PARTITION sales_q3 VALUES LESS THAN (1999, 10, 01) TABLESPACE tsc,
  PARTITION sales_q4 VALUES LESS THAN (2000, 01, 01) TABLESPACE tsd);
```

- A row with sale_year=1999, sale_month=8, and sale_day=1 has a partitioning key of (1999, 8, 1) and would be stored in partition sales_q3.
- CHECK TO SEE IF THE PARTITIONS ARE CORRECTLY IN PLACE

```
SELECT TABLE_NAME, PARTITION_NAME, HIGH_VALUE, TABLESPACE_NAME
FROM USER_TAB_PARTITIONS                                    WHERE
TABLE_NAME = 'SALES' ORDER BY TABLESPACE_NAME;
```

```
TABLE_NAME   PARTITION_NAME   HIGH_VALUE    TABLESPACE_NAME
SALES        SALES_Q1         1999, 04, 01       TSA
SALES        SALES_Q2         1999, 07, 01       TSB
SALES        SALES_Q3         1999, 10, 01       TSC
SALES        SALES_Q4         2000, 01, 01       TSD
```

Insert Data and Validate Contents



```
-- Insert Data Into Each Partition
INSERT INTO SALES VALUES (1999003, 1999, 9, 20);
INSERT INTO SALES VALUES (1999001, 1999, 3, 1);
INSERT INTO SALES VALUES (1999004, 1999, 11, 8);
INSERT INTO SALES VALUES (1999002, 1999, 6, 15);
```

```
-- Confirmation that everything is in the proper place
SELECT * FROM SALES;
SELECT * FROM SALES PARTITION(SALES_Q1);
                :
SELECT * FROM SALES PARTITION(SALES_Q4);
```

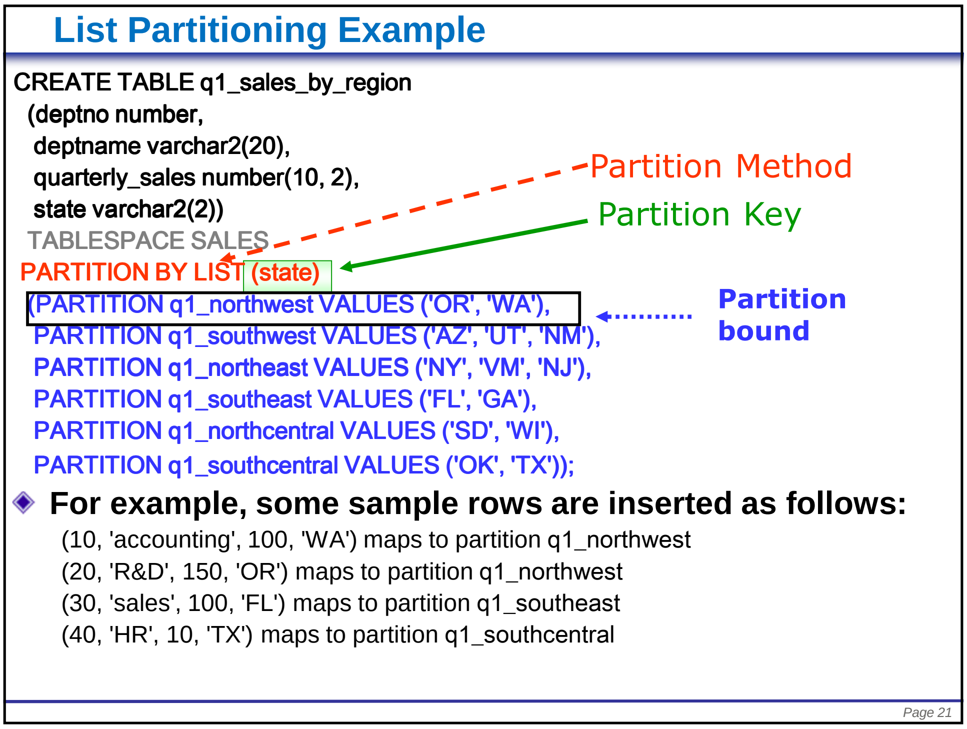
```
INVOICE_NO  SALE_YEAR  SALE_MONTH  SALE_DAY
  1999001      1999          3         1
  1999002      1999          6        15
  1999003      1999          9        20
  1999004      1999         11         8

INVOICE_NO  SALE_YEAR  SALE_MONTH  SALE_DAY
  1999001      1999          3         1

INVOICE_NO  SALE_YEAR  SALE_MONTH  SALE_DAY
  1999004      1999         11         8
```

❑ List Partitioning

- Added in Oracle 9.1
- In 9.2 the “DEFAULT” partition method was added.
- Allows DBAs to explicitly define what is in a partition.
- When creating list partitions, you must specify:
  - Partitioning method: list
  - Partitioning column(s) or key
  - Partition descriptions, each specifying a list of literal values (a value list), which are the discrete values of the partitioning column that qualify a row to be included in the partition
- List partitioning allows the DBA to enumerate the partition-key values for each partition
  - Useful for partitioning over discrete domains (e.g. geography, product category)

Example:

```
CREATE TABLE sales_by_product_line ( ...)
PARTITION BY LIST (state)  (
        PARTITION northwest VALUES IN(’OR’, ’WA’),
        PARTITION southwest VALUES IN (’AZ’, ’UT’, ’NM’),
        PARTITION northeast VALUES IN (’NY’, ’VT’, ’NJ’),
        PARTITION southeast VALUES IN (’FL’, ’GA’)));
```



List Partitioning Example

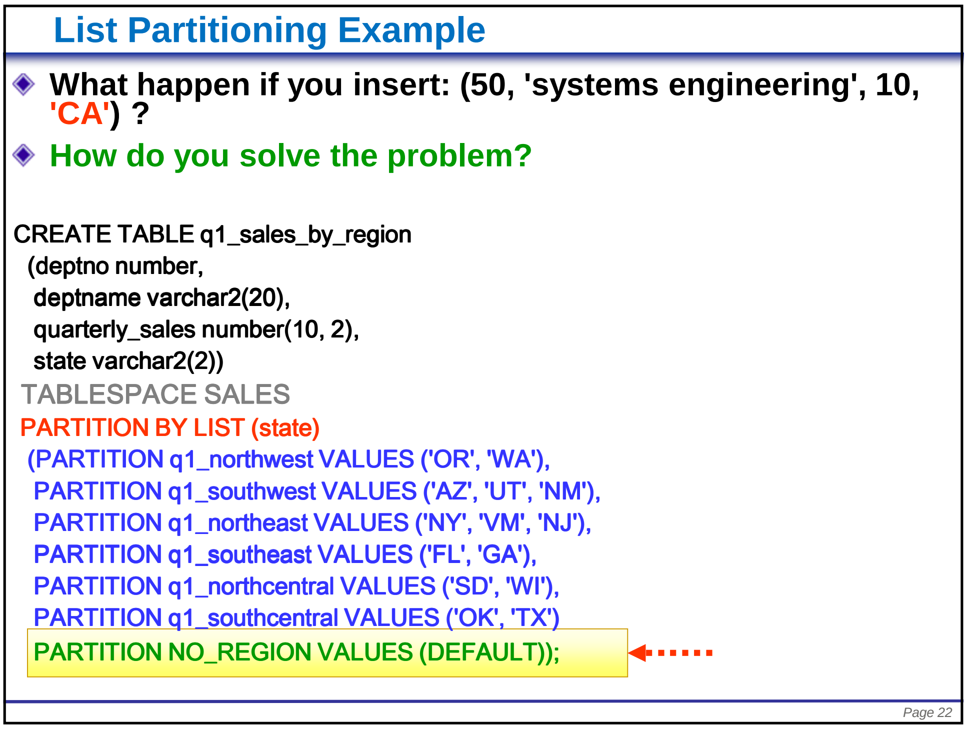
```
CREATE TABLE q1_sales_by_region
  (deptno number,
   deptname varchar2(20),
   quarterly_sales number(10, 2),
   state varchar2(2))
  TABLESPACE SALES
 PARTITION BY LIST (state)
  (PARTITION q1_northwest VALUES ('OR', 'WA'),
   PARTITION q1_southwest VALUES ('AZ', 'UT', 'NM'),
   PARTITION q1_northeast VALUES ('NY', 'VM', 'NJ'),
   PARTITION q1_southeast VALUES ('FL', 'GA'),
   PARTITION q1_northcentral VALUES ('SD', 'WI'),
   PARTITION q1_southcentral VALUES ('OK', 'TX'));
```

Partition Method

Partition Key

Partition bound

- **For example, some sample rows are inserted as follows:**
  - (10, 'accounting', 100, 'WA') maps to partition q1_northwest
  - (20, 'R&D', 150, 'OR') maps to partition q1_northwest
  - (30, 'sales', 100, 'FL') maps to partition q1_southeast
  - (40, 'HR', 10, 'TX') maps to partition q1_southcentral



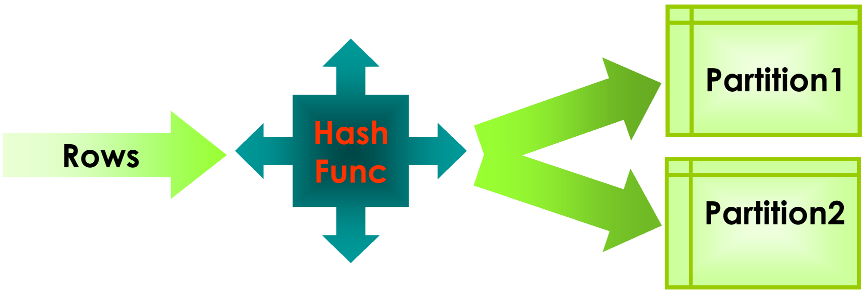
List Partitioning Example

- **What happen if you insert: (50, 'systems engineering', 10, 'CA') ?**
- **How do you solve the problem?**

```
CREATE TABLE q1_sales_by_region
  (deptno number,
   deptname varchar2(20),
   quarterly_sales number(10, 2),
   state varchar2(2))
 TABLESPACE SALES
 PARTITION BY LIST (state)
  (PARTITION q1_northwest VALUES ('OR', 'WA'),
   PARTITION q1_southwest VALUES ('AZ', 'UT', 'NM'),
   PARTITION q1_northeast VALUES ('NY', 'VM', 'NJ'),
   PARTITION q1_southeast VALUES ('FL', 'GA'),
   PARTITION q1_northcentral VALUES ('SD', 'WI'),
   PARTITION q1_southcentral VALUES ('OK', 'TX')
   PARTITION NO_REGION VALUES (DEFAULT));
```

❑ Hash Partitioning

- Hashing allows for distributed data by a hash key.
- DBAs do not have to know the data.
- Distribution is handled by Oracle.
- Each partition can have its own tablespace.
- To create hash partitions you specify the following:
  - Partitioning method: hash
  - Partitioning columns(s) or key
  - Number of partitions or individual partition descriptions



Hash Partitioning Example

```
CREATE TABLE PARTITION_BY_HASH
(FIRST_NAME VARCHAR2(10),
 MIDDLE_INIT       VARCHAR2(1),
 LAST_NAME         VARCHAR2(10),
 AGE               INT NOT NULL)
PARTITION BY HASH (AGE)
(PARTITION P1_AGE TABLESPACE TS01,
 PARTITION P2_AGE TABLESPACE TS02,
 PARTITION P3_AGE TABLESPACE TS03,
 PARTITION P4_AGE TABLESPACE TS04);
```

Partition Method

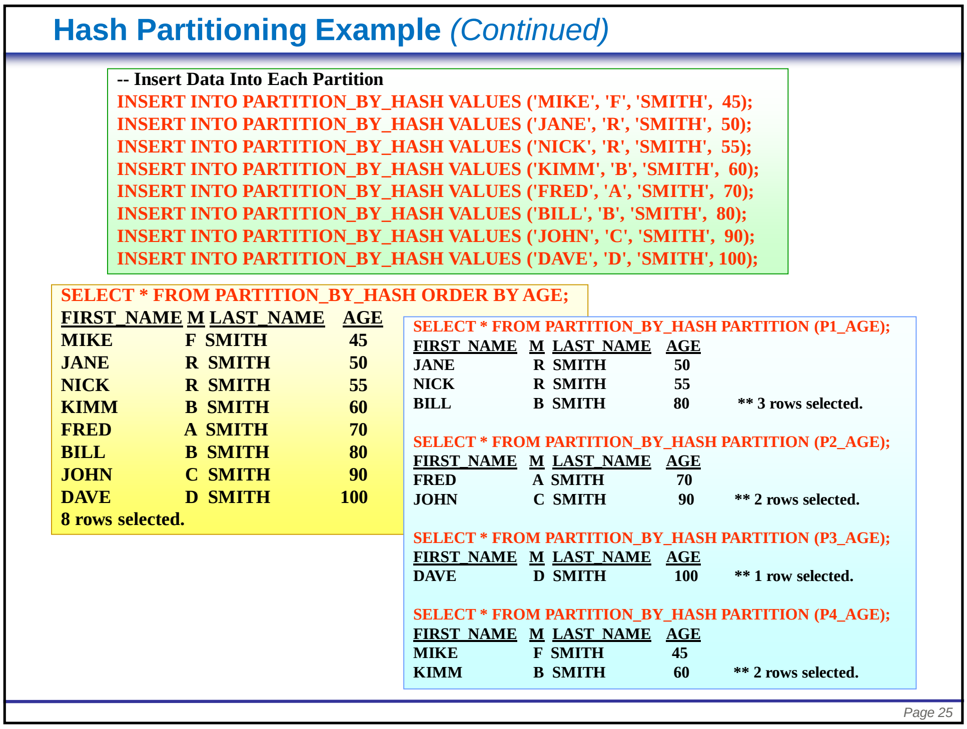
Partition Key

Partition Bound

- CHECK TO SEE IF THE PARTITIONS ARE CORRECTLY IN PLACE

```
SELECT TABLE_NAME, PARTITION_NAME, HIGH_VALUE, TABLESPACE_NAME
FROM USER_TAB_PARTITIONS                                WHERE
TABLE_NAME = 'PARTITION_BY_HASH ' ORDER BY TABLESPACE_NAME;
```

TABLE_NAME	PARTITION_NAME	HIGH_VALUE	TABLESPACE_NAME
PARTITION_BY_HASH	P1_AGE		TS01
PARTITION_BY_HASH	P2_AGE		TS02
PARTITION_BY_HASH	P3_AGE		TS03
PARTITION_BY_HASH	P4_AGE		TS04



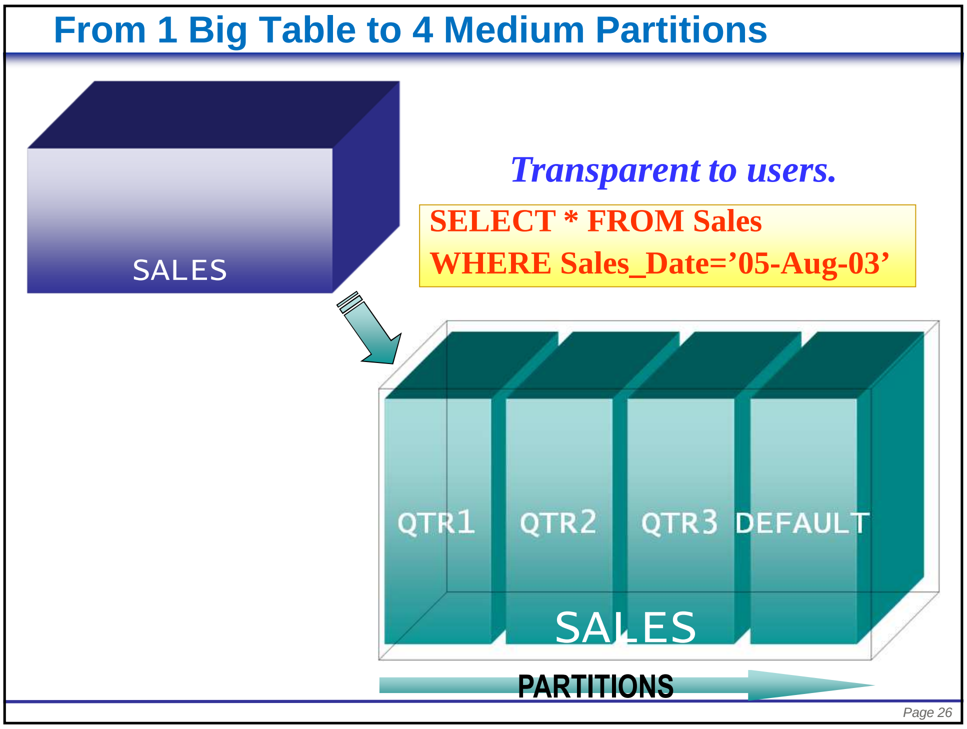
Hash Partitioning Example *(Continued)*

```
-- Insert Data Into Each Partition
INSERT INTO PARTITION_BY_HASH VALUES ('MIKE', 'F', 'SMITH',  45);
INSERT INTO PARTITION_BY_HASH VALUES ('JANE', 'R', 'SMITH',  50);
INSERT INTO PARTITION_BY_HASH VALUES ('NICK', 'R', 'SMITH',  55);
INSERT INTO PARTITION_BY_HASH VALUES ('KIMM', 'B', 'SMITH',  60);
INSERT INTO PARTITION_BY_HASH VALUES ('FRED', 'A', 'SMITH',  70);
INSERT INTO PARTITION_BY_HASH VALUES ('BILL', 'B', 'SMITH',  80);
INSERT INTO PARTITION_BY_HASH VALUES ('JOHN', 'C', 'SMITH',  90);
INSERT INTO PARTITION_BY_HASH VALUES ('DAVE', 'D', 'SMITH', 100);
```

```
SELECT * FROM PARTITION_BY_HASH ORDER BY AGE;
FIRST_NAME M LAST_NAME    AGE
MIKE       F  SMITH        45
JANE       R  SMITH        50
NICK       R  SMITH        55
KIMM       B  SMITH        60
FRED       A  SMITH        70
BILL       B  SMITH        80
JOHN       C  SMITH        90
DAVE       D  SMITH       100
8 rows selected.
```

```
SELECT * FROM PARTITION_BY_HASH PARTITION (P1_AGE);
FIRST_NAME  M  LAST_NAME  AGE
JANE        R  SMITH       50
NICK        R  SMITH       55
BILL        B  SMITH       80      ** 3 rows selected.

SELECT * FROM PARTITION_BY_HASH PARTITION (P2_AGE);
FIRST_NAME  M  LAST_NAME  AGE
FRED        A  SMITH       70
JOHN        C  SMITH       90      ** 2 rows selected.

SELECT * FROM PARTITION_BY_HASH PARTITION (P3_AGE);
FIRST_NAME  M  LAST_NAME  AGE
DAVE        D  SMITH      100      ** 1 row selected.

SELECT * FROM PARTITION_BY_HASH PARTITION (P4_AGE);
FIRST_NAME  M  LAST_NAME  AGE
MIKE        F  SMITH       45
KIMM        B  SMITH       60      ** 2 rows selected.
```

Page 25



From 1 Big Table to 4 Medium Partitions

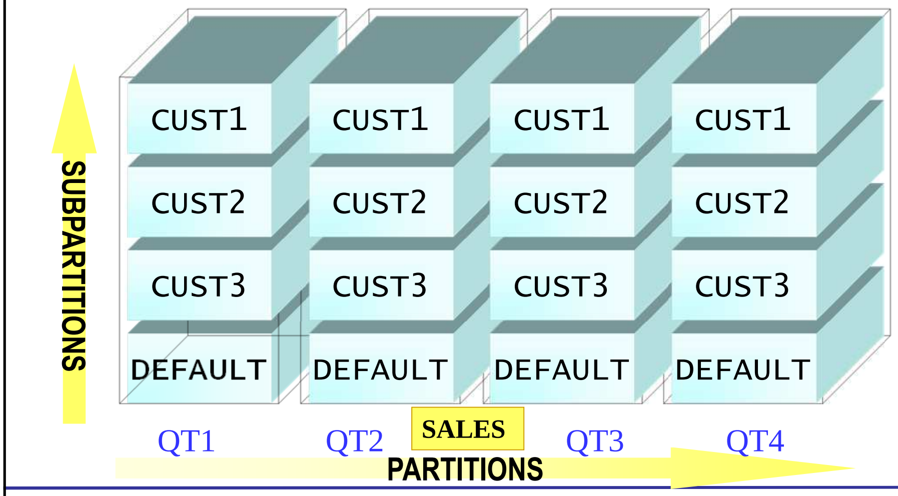
Transparent to users.

SELECT * FROM Sales
WHERE Sales_Date=’05-Aug-03’

Page 26

Composite Partitioning (Subpartitions)

- Partitions are subdivided
  - Range-Hash
  - Range-List

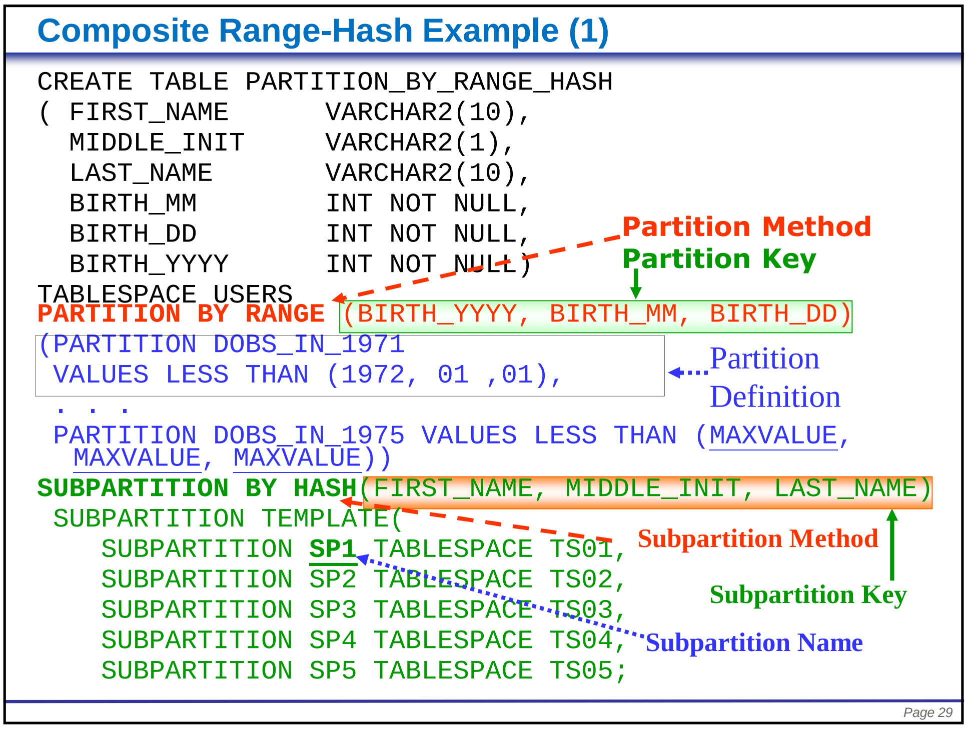


❑ Composite Range-Hash

- Partition by Range
- Stored by a Hash algorithm
- DBAs can focus on both the ease of Range Partitioning and get the benefits of Hash Partitioning
- Logically divide the data and <u>let Oracle determine where to store</u>.
- When creating range-hash partitions, you specify the following:
  - Partitioning method: range
  - Partitioning column(s) or key
  - Partition descriptions identifying partition bounds
  - Subpartitioning method: hash
  - Subpartitioning column(s) or key
  - <u>Number of subpartitions</u> for each partition or descriptions of subpartitions
- Recommended step-by-step process to build a range-hash table:

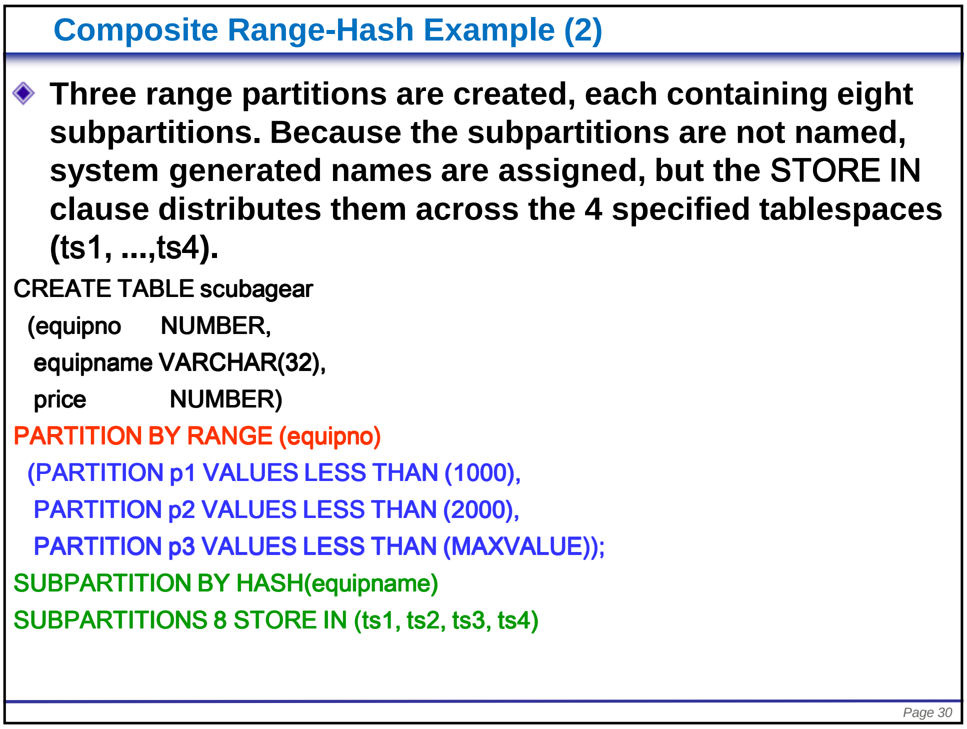
1. Determine the partition key for the range.
2. Design a range partition table.
3. Determine the partition key for the hash.
4. Create the `SUBPARTITION BY HASH` clause.
5. Create the `SUBPARTITION TEMPLATE`.

* Do Steps 1 and 2 first. Then you can insert the code created in Steps 3 –5 in the range partition table syntax.



Composite Range-Hash Example (1)

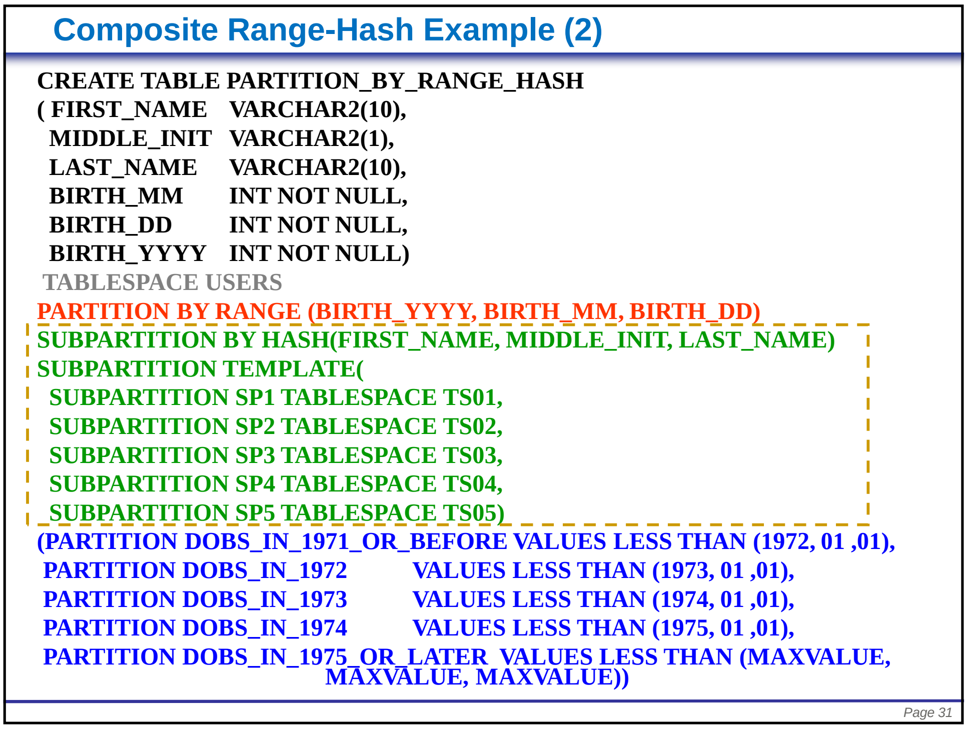
```
CREATE TABLE PARTITION_BY_RANGE_HASH
( FIRST_NAME      VARCHAR2(10),
  MIDDLE_INIT     VARCHAR2(1),
  LAST_NAME       VARCHAR2(10),
  BIRTH_MM        INT NOT NULL,
  BIRTH_DD        INT NOT NULL,
  BIRTH_YYYY      INT NOT NULL)
TABLESPACE USERS
PARTITION BY RANGE (BIRTH_YYYY, BIRTH_MM, BIRTH_DD)
(PARTITION DOBS_IN_1971
 VALUES LESS THAN (1972, 01 ,01),
 . . .
 PARTITION DOBS_IN_1975 VALUES LESS THAN (MAXVALUE,
  MAXVALUE, MAXVALUE))
SUBPARTITION BY HASH(FIRST_NAME, MIDDLE_INIT, LAST_NAME)
 SUBPARTITION TEMPLATE(
    SUBPARTITION SP1 TABLESPACE TS01,
    SUBPARTITION SP2 TABLESPACE TS02,
    SUBPARTITION SP3 TABLESPACE TS03,
    SUBPARTITION SP4 TABLESPACE TS04,
    SUBPARTITION SP5 TABLESPACE TS05;
```



Composite Range-Hash Example (2)

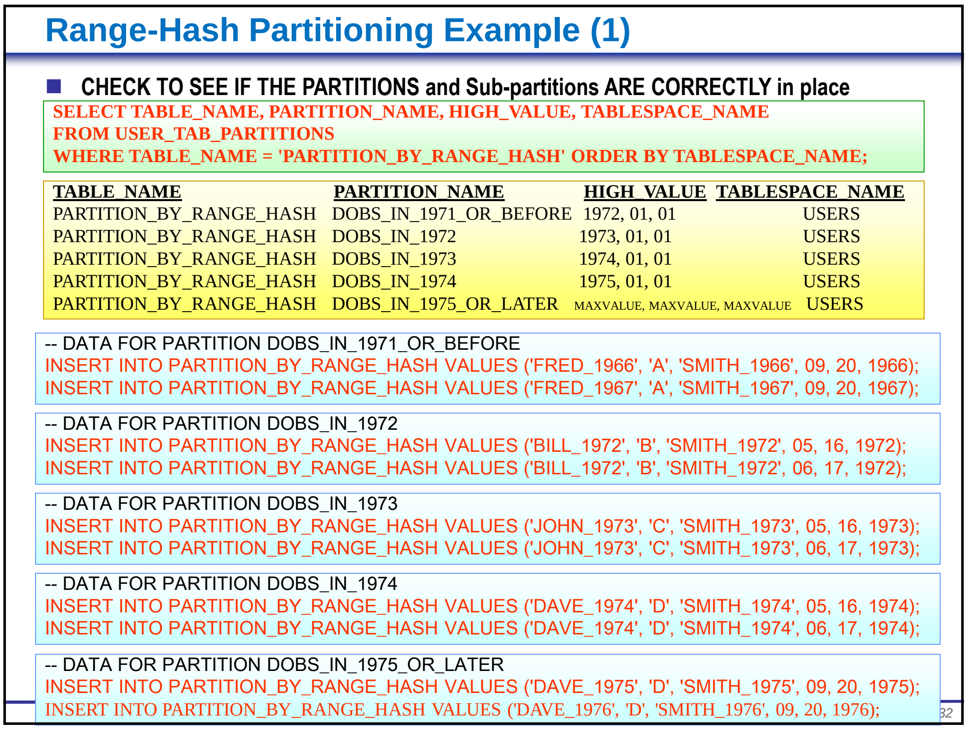
- Three range partitions are created, each containing eight subpartitions. Because the subpartitions are not named, system generated names are assigned, but the STORE IN clause distributes them across the 4 specified tablespaces (ts1, ...,ts4).

```
CREATE TABLE scubagear
 (equipno      NUMBER,
  equipname VARCHAR(32),
  price          NUMBER)
PARTITION BY RANGE (equipno)
 (PARTITION p1 VALUES LESS THAN (1000),
  PARTITION p2 VALUES LESS THAN (2000),
  PARTITION p3 VALUES LESS THAN (MAXVALUE));
SUBPARTITION BY HASH(equipname)
SUBPARTITIONS 8 STORE IN (ts1, ts2, ts3, ts4)
```



Composite Range-Hash Example (2)

```
CREATE TABLE PARTITION_BY_RANGE_HASH
( FIRST_NAME   VARCHAR2(10),
  MIDDLE_INIT  VARCHAR2(1),
  LAST_NAME    VARCHAR2(10),
  BIRTH_MM     INT NOT NULL,
  BIRTH_DD     INT NOT NULL,
  BIRTH_YYYY   INT NOT NULL)
 TABLESPACE USERS
PARTITION BY RANGE (BIRTH_YYYY, BIRTH_MM, BIRTH_DD)
SUBPARTITION BY HASH(FIRST_NAME, MIDDLE_INIT, LAST_NAME)
SUBPARTITION TEMPLATE(
  SUBPARTITION SP1 TABLESPACE TS01,
  SUBPARTITION SP2 TABLESPACE TS02,
  SUBPARTITION SP3 TABLESPACE TS03,
  SUBPARTITION SP4 TABLESPACE TS04,
  SUBPARTITION SP5 TABLESPACE TS05)
(PARTITION DOBS_IN_1971_OR_BEFORE VALUES LESS THAN (1972, 01 ,01),
 PARTITION DOBS_IN_1972         VALUES LESS THAN (1973, 01 ,01),
 PARTITION DOBS_IN_1973         VALUES LESS THAN (1974, 01 ,01),
 PARTITION DOBS_IN_1974         VALUES LESS THAN (1975, 01 ,01),
 PARTITION DOBS_IN_1975_OR_LATER  VALUES LESS THAN (MAXVALUE,
                                   MAXVALUE, MAXVALUE))
```



Range-Hash Partitioning Example (1)

- CHECK TO SEE IF THE PARTITIONS and Sub-partitions ARE CORRECTLY in place

```
SELECT TABLE_NAME, PARTITION_NAME, HIGH_VALUE, TABLESPACE_NAME
FROM USER_TAB_PARTITIONS
WHERE TABLE_NAME = 'PARTITION_BY_RANGE_HASH' ORDER BY TABLESPACE_NAME;
```

```
TABLE_NAME                PARTITION_NAME          HIGH_VALUE  TABLESPACE_NAME
PARTITION_BY_RANGE_HASH   DOBS_IN_1971_OR_BEFORE  1972, 01, 01              USERS
PARTITION_BY_RANGE_HASH   DOBS_IN_1972            1973, 01, 01              USERS
PARTITION_BY_RANGE_HASH   DOBS_IN_1973            1974, 01, 01              USERS
PARTITION_BY_RANGE_HASH   DOBS_IN_1974            1975, 01, 01              USERS
PARTITION_BY_RANGE_HASH   DOBS_IN_1975_OR_LATER   MAXVALUE, MAXVALUE, MAXVALUE  USERS
```

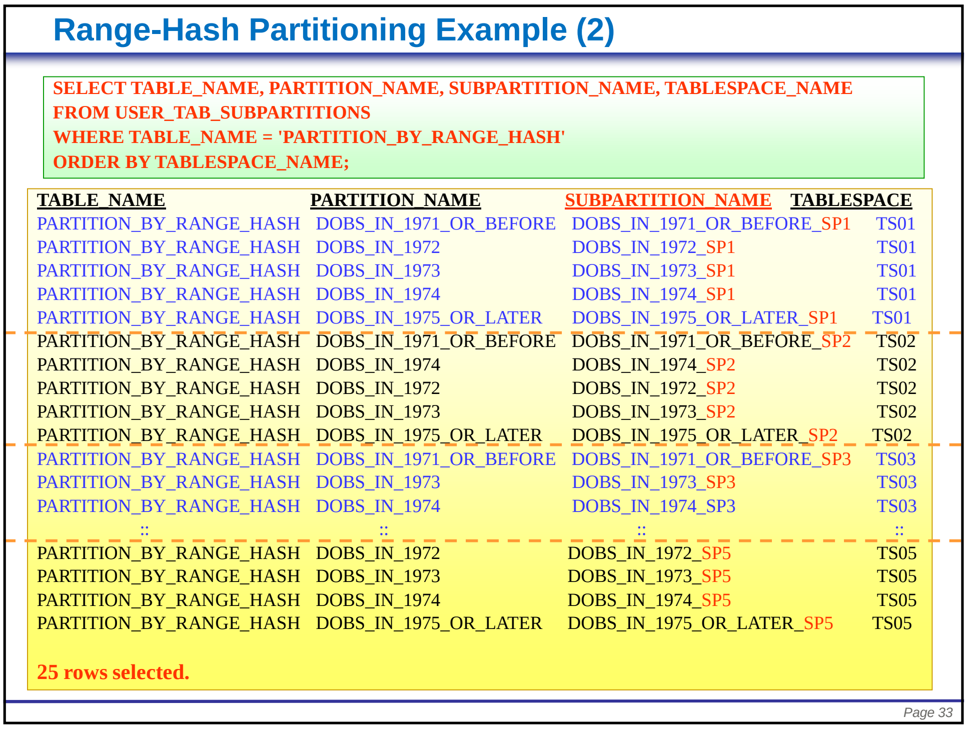
```
-- DATA FOR PARTITION DOBS_IN_1971_OR_BEFORE
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('FRED_1966', 'A', 'SMITH_1966', 09, 20, 1966);
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('FRED_1967', 'A', 'SMITH_1967', 09, 20, 1967);
```

```
-- DATA FOR PARTITION DOBS_IN_1972
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('BILL_1972', 'B', 'SMITH_1972', 05, 16, 1972);
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('BILL_1972', 'B', 'SMITH_1972', 06, 17, 1972);
```

```
-- DATA FOR PARTITION DOBS_IN_1973
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('JOHN_1973', 'C', 'SMITH_1973', 05, 16, 1973);
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('JOHN_1973', 'C', 'SMITH_1973', 06, 17, 1973);
```

```
-- DATA FOR PARTITION DOBS_IN_1974
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('DAVE_1974', 'D', 'SMITH_1974', 05, 16, 1974);
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('DAVE_1974', 'D', 'SMITH_1974', 06, 17, 1974);
```

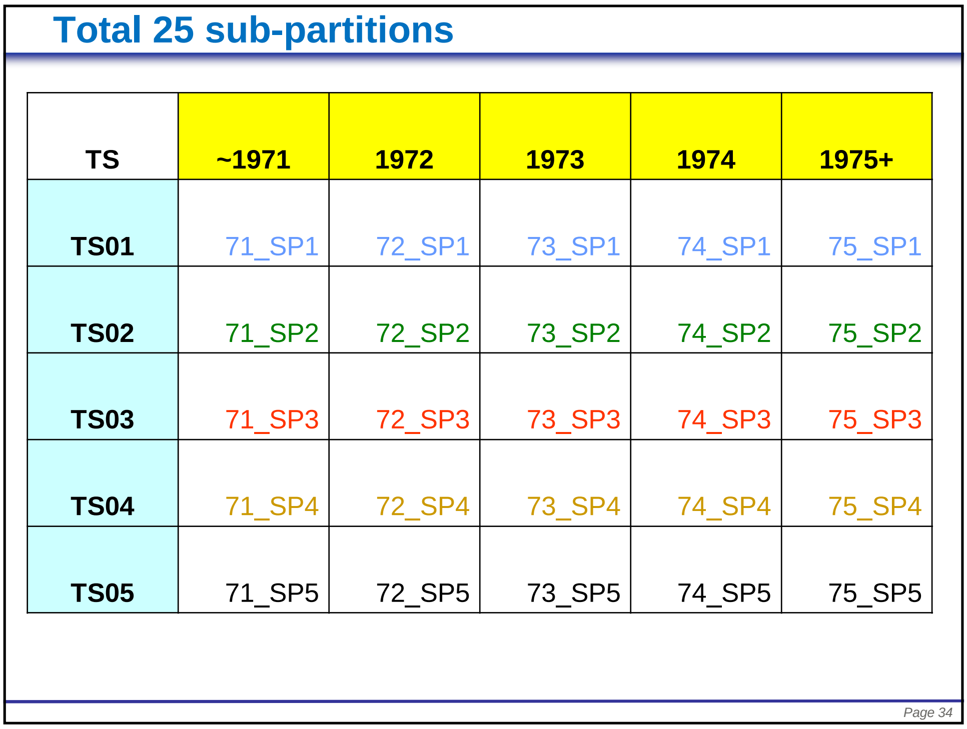
```
-- DATA FOR PARTITION DOBS_IN_1975_OR_LATER
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('DAVE_1975', 'D', 'SMITH_1975', 09, 20, 1975);
INSERT INTO PARTITION_BY_RANGE_HASH VALUES ('DAVE_1976', 'D', 'SMITH_1976', 09, 20, 1976);
```



Range-Hash Partitioning Example (2)

```
SELECT TABLE_NAME, PARTITION_NAME, SUBPARTITION_NAME, TABLESPACE_NAME
FROM USER_TAB_SUBPARTITIONS
WHERE TABLE_NAME = 'PARTITION_BY_RANGE_HASH'
ORDER BY TABLESPACE_NAME;
```

```
TABLE_NAME                PARTITION_NAME          SUBPARTITION_NAME            TABLESPACE
PARTITION_BY_RANGE_HASH  DOBS_IN_1971_OR_BEFORE  DOBS_IN_1971_OR_BEFORE_SP1    TS01
PARTITION_BY_RANGE_HASH  DOBS_IN_1972            DOBS_IN_1972_SP1              TS01
PARTITION_BY_RANGE_HASH  DOBS_IN_1973            DOBS_IN_1973_SP1              TS01
PARTITION_BY_RANGE_HASH  DOBS_IN_1974            DOBS_IN_1974_SP1              TS01
PARTITION_BY_RANGE_HASH  DOBS_IN_1975_OR_LATER   DOBS_IN_1975_OR_LATER_SP1     TS01
PARTITION_BY_RANGE_HASH  DOBS_IN_1971_OR_BEFORE  DOBS_IN_1971_OR_BEFORE_SP2    TS02
PARTITION_BY_RANGE_HASH  DOBS_IN_1974            DOBS_IN_1974_SP2              TS02
PARTITION_BY_RANGE_HASH  DOBS_IN_1972            DOBS_IN_1972_SP2              TS02
PARTITION_BY_RANGE_HASH  DOBS_IN_1973            DOBS_IN_1973_SP2              TS02
PARTITION_BY_RANGE_HASH  DOBS_IN_1975_OR_LATER   DOBS_IN_1975_OR_LATER_SP2     TS02
PARTITION_BY_RANGE_HASH  DOBS_IN_1971_OR_BEFORE  DOBS_IN_1971_OR_BEFORE_SP3    TS03
PARTITION_BY_RANGE_HASH  DOBS_IN_1973            DOBS_IN_1973_SP3              TS03
PARTITION_BY_RANGE_HASH  DOBS_IN_1974            DOBS_IN_1974_SP3              TS03
          ::                       ::                       ::                   ::
PARTITION_BY_RANGE_HASH  DOBS_IN_1972            DOBS_IN_1972_SP5              TS05
PARTITION_BY_RANGE_HASH  DOBS_IN_1973            DOBS_IN_1973_SP5              TS05
PARTITION_BY_RANGE_HASH  DOBS_IN_1974            DOBS_IN_1974_SP5              TS05
PARTITION_BY_RANGE_HASH  DOBS_IN_1975_OR_LATER   DOBS_IN_1975_OR_LATER_SP5     TS05

25 rows selected.
```



Total 25 sub-partitions

TS	~1971	1972	1973	1974	1975+
TS01	71_SP1	72_SP1	73_SP1	74_SP1	75_SP1
TS02	71_SP2	72_SP2	73_SP2	74_SP2	75_SP2
TS03	71_SP3	72_SP3	73_SP3	74_SP3	75_SP3
TS04	71_SP4	72_SP4	73_SP4	74_SP4	75_SP4
TS05	71_SP5	72_SP5	73_SP5	74_SP5	75_SP5

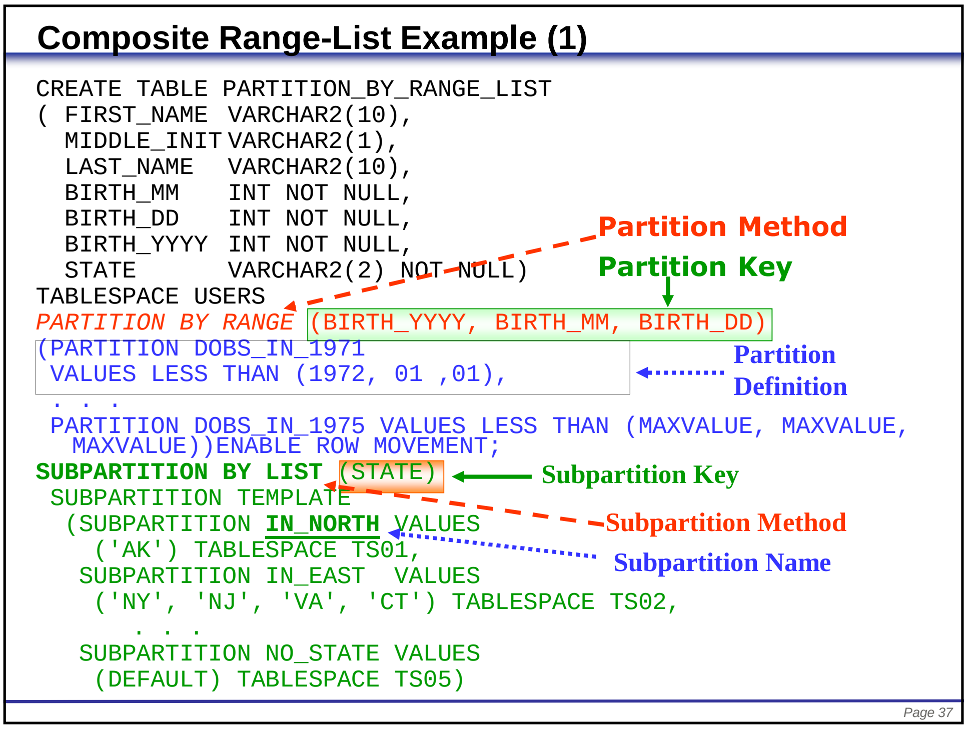
Range-Hash Partitioning Example (3)

```
SELECT * FROM PARTITION_BY_RANGE_HASH;
FIRST_NAME M LAST_NAME  BIRTH_MM  BIRTH_DD BIRTH_YYYY
FRED_1967    A  SMITH_1967        9            20          1967
FRED_1966    A  SMITH_1966        9            20          1966
BILL_1972    B  SMITH_1972        5            16          1972
BILL_1972    B  SMITH_1972        6            17          1972
JOHN_1973   C  SMITH_1973        5            16          1973
JOHN_1973   C  SMITH_1973        6            17          1973
DAVE_1974   D  SMITH_1974        5            16          1974
DAVE_1974   D  SMITH_1974        6            17          1974
DAVE_1975   D  SMITH_1975        9            20          1975
DAVE_1976   D  SMITH_1976        9            20          1976

10 rows selected.
```

```
SELECT * FROM PARTITION_BY_RANGE_HASH PARTITION (DOBS_IN_1971_OR_BEFORE) ORDER BY
BIRTH_YYYY;
FIRST_NAME  M  LAST_NAME  BIRTH_MM  BIRTH_DD  BIRTH_YYYY
FRED_1966      A  SMITH_1966          9              20              1966
FRED_1967      A  SMITH_1967          9              20              1967          ** 2 rows selected.

      SELECT * FROM PARTITION_BY_RANGE_HASH PARTITION (DOBS_IN_1972);
Or:  SELECT * FROM PARTITION_BY_RANGE_HASH SUBPARTITION(DOBS_in_1972_sp2);
 FIRST_NAME  M  LAST_NAME  BIRTH_MM  BIRTH_DD  BIRTH_YYYY
 BILL_1972       B  SMITH_1972          5              16              1972
 BILL_1972        B  SMITH_1972         6              17              1972          ** 2 rows selected.
```

❑ Composite Range-List

- Similar to Range-Hash partitioning.
- Subpartition is by List method.
- Allows for greater control by the DBAs.
- Proper use of Range-List must be carefully thought out.
- When creating range-list partitions, you specify the following:
  - Partitioning method: range
  - Partitioning column(s) or key
  - Partition descriptions identifying partition bounds
  - Subpartitioning method: list
  - Subpartitioning column or key
  - Subpartition descriptions, each specifying a list of literal values (a value list), which are the discrete values of the subpartitioning column that qualify a row to be included in the subpartition



Composite Range-List Example (1)

```
CREATE TABLE PARTITION_BY_RANGE_LIST
( FIRST_NAME  VARCHAR2(10),
  MIDDLE_INIT VARCHAR2(1),
  LAST_NAME   VARCHAR2(10),
  BIRTH_MM    INT NOT NULL,
  BIRTH_DD    INT NOT NULL,
  BIRTH_YYYY  INT NOT NULL,
  STATE       VARCHAR2(2) NOT NULL)
TABLESPACE USERS
PARTITION BY RANGE (BIRTH_YYYY, BIRTH_MM, BIRTH_DD)
(PARTITION DOBS_IN_1971
 VALUES LESS THAN (1972, 01 ,01),
 . . .
 PARTITION DOBS_IN_1975 VALUES LESS THAN (MAXVALUE, MAXVALUE,
   MAXVALUE))ENABLE ROW MOVEMENT;
SUBPARTITION BY LIST (STATE)
 SUBPARTITION TEMPLATE
  (SUBPARTITION IN_NORTH VALUES
    ('AK') TABLESPACE TS01,
   SUBPARTITION IN_EAST  VALUES
    ('NY', 'NJ', 'VA', 'CT') TABLESPACE TS02,
       . . .
   SUBPARTITION NO_STATE VALUES
    (DEFAULT) TABLESPACE TS05)
```

Partition Method

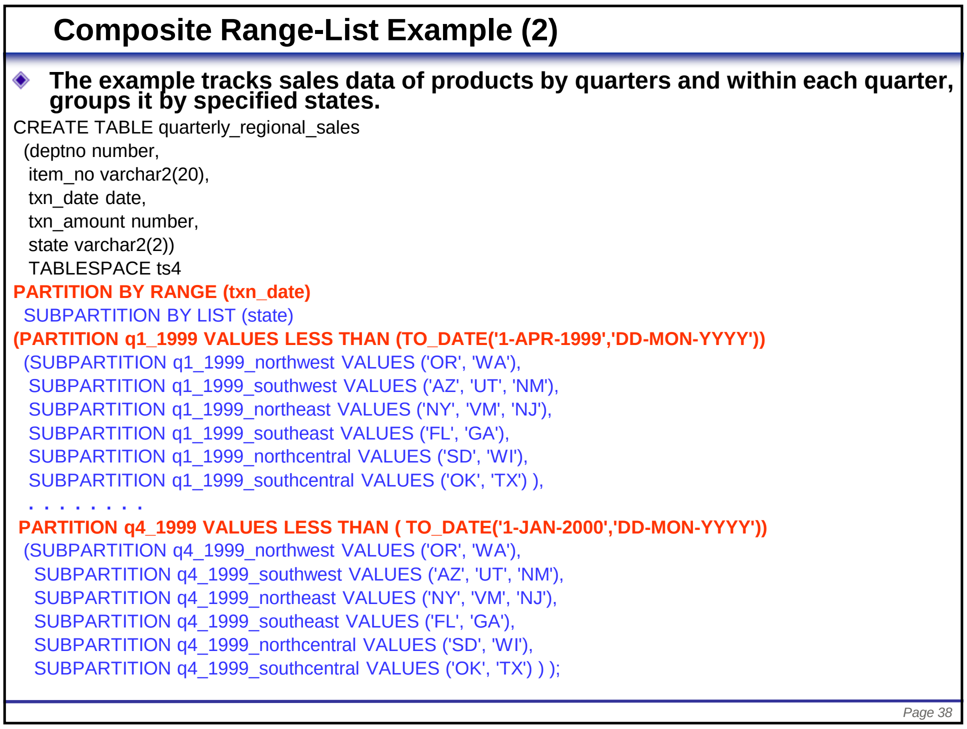
Partition Key

Partition Definition

Subpartition Key

Subpartition Method

Subpartition Name



Composite Range-List Example (2)

- **The example tracks sales data of products by quarters and within each quarter, groups it by specified states.**

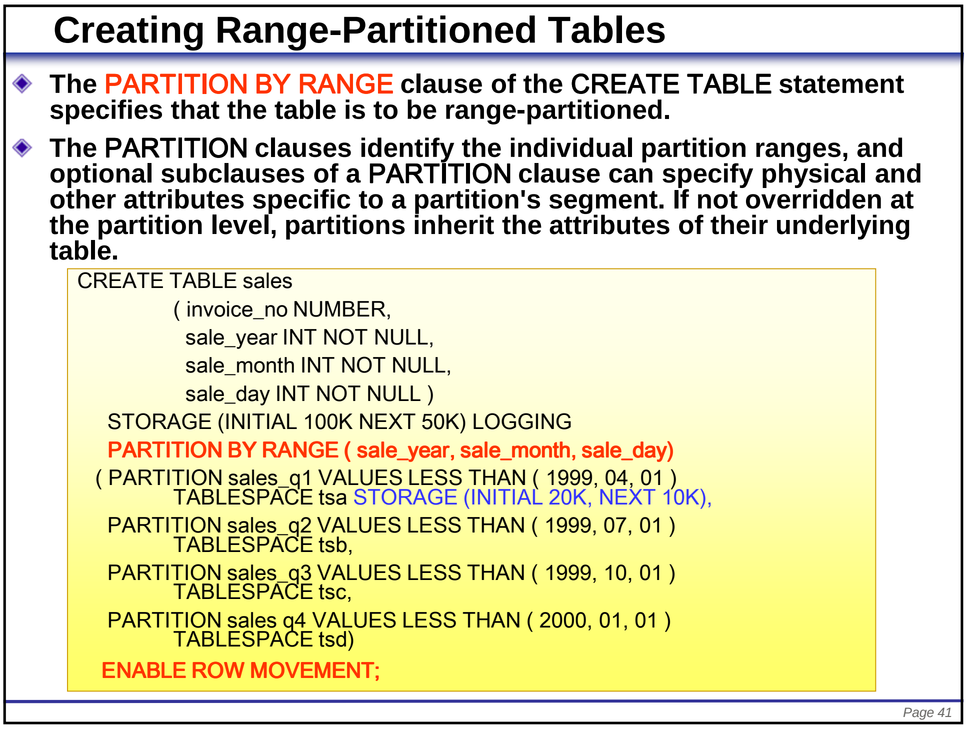
```
CREATE TABLE quarterly_regional_sales
 (deptno number,
  item_no varchar2(20),
  txn_date date,
  txn_amount number,
  state varchar2(2))
  TABLESPACE ts4
PARTITION BY RANGE (txn_date)
 SUBPARTITION BY LIST (state)
(PARTITION q1_1999 VALUES LESS THAN (TO_DATE('1-APR-1999','DD-MON-YYYY'))
 (SUBPARTITION 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 q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000','DD-MON-YYYY'))
 (SUBPARTITION q4_1999_northwest VALUES ('OR', 'WA'),
   SUBPARTITION q4_1999_southwest VALUES ('AZ', 'UT', 'NM'),
   SUBPARTITION q4_1999_northeast VALUES ('NY', 'VM', 'NJ'),
   SUBPARTITION q4_1999_southeast VALUES ('FL', 'GA'),
   SUBPARTITION q4_1999_northcentral VALUES ('SD', 'WI'),
   SUBPARTITION q4_1999_southcentral VALUES ('OK', 'TX') ) );
```

Composite Range-List Example (3)

- **A row is mapped to a partition by checking whether the value of the partitioning column for a row falls within a specific partition range. The row is then mapped to a subpartition within that partition by identifying the subpartition whose descriptor value list contains a value matching the subpartition column value.**
- **For example, some sample rows are inserted as follows:**
  - (10, 4532130, '23-Jan-1999', 8934.10, 'WA') maps to subpartition q1_1999_northwest
  - (20, 5671621, '15-May-1999', 49021.21, 'OR') maps to subpartition q2_1999_northeast
  - (30, 9977612, ,'07-Sep-1999', 30987.90, 'FL') maps to subpartition q3_1999_southeast
  - (40, 9977612, '29-Nov-1999', 67891.45, 'TX') maps to subpartition q4_1999_southwest
  - (40, 4532130, '5-Jan-2000', 897231.55, 'TX') does not map to any partition in the table and raises an error
  - (50, 5671621, '17-Dec-1999', 76123.35, 'CA') does not map to any subpartition in the table and raises an error
- **The partitions of a range-list partitioned table are logical structures only, as their data is stored in the segments of their subpartitions.**
- **The list subpartitions have the same characteristics as list partitions.**
- **You can specify a default subpartition, just as you specify a default partition for list partitioning.**

Creating Partitioned Tables

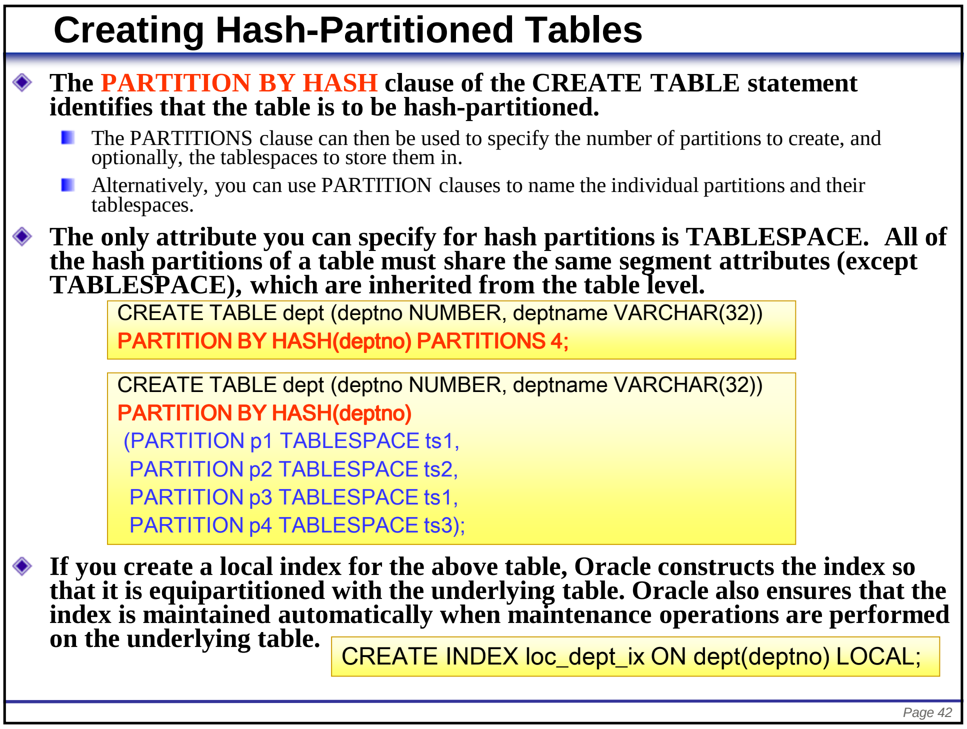
- **Creating a partitioned table or index is very similar to creating a non-partitioned table or index, but you include a partitioning clause. The partitioning clause, and subclauses, that you include depend upon the type of partitioning you want to achieve.**
  - Partition key defined at creation
  - Separates data in different pieces
  - Physical attributes (PCTFREE, PCTUSED, INITRANS,MAXTRANS) may vary for different partitions of the same table or index
- **You can partition both regular (heap organized) tables and index-organized tables. You can create nonpartitioned global indexes, range-partitioned global indexes, and local indexes on partitioned tables.**
- **When you create (or alter) a partitioned table, a row movement clause, either ENABLE ROW MOVEMENT or DISABLE ROW MOVEMENT can be specified. This clause either enables or disables the migration of a row to a new partition if its key is updated. The default is DISABLE ROW MOVEMENT.**
- **Various types of partitioned tables and indexes:**
  - Creating Range-Partitioned Tables
  - Creating Hash-Partitioned Tables
  - Creating List-Partitioned Tables
  - Creating Composite Range-Hash Partitioned Tables
  - Creating Composite Range-List Partitioned Tables
  - Using Subpartition Templates to Describe Composite Partitioned Tables
  - Creating Partitioned Index-Organized Tables



Creating Range-Partitioned Tables

- The PARTITION BY RANGE clause of the CREATE TABLE statement specifies that the table is to be range-partitioned.
- The PARTITION clauses identify the individual partition ranges, and optional subclauses of a PARTITION clause can specify physical and other attributes specific to a partition's segment. If not overridden at the partition level, partitions inherit the attributes of their underlying table.

```
CREATE TABLE sales
        ( invoice_no NUMBER,
          sale_year INT NOT NULL,
          sale_month INT NOT NULL,
          sale_day INT NOT NULL )
   STORAGE (INITIAL 100K NEXT 50K) LOGGING
   PARTITION BY RANGE ( sale_year, sale_month, sale_day)
  ( PARTITION sales_q1 VALUES LESS THAN ( 1999, 04, 01 )
        TABLESPACE tsa STORAGE (INITIAL 20K, NEXT 10K),
   PARTITION sales_q2 VALUES LESS THAN ( 1999, 07, 01 )
        TABLESPACE tsb,
   PARTITION sales_q3 VALUES LESS THAN ( 1999, 10, 01 )
        TABLESPACE tsc,
   PARTITION sales q4 VALUES LESS THAN ( 2000, 01, 01 )
        TABLESPACE tsd)
  ENABLE ROW MOVEMENT;
```



Creating Hash-Partitioned Tables

- **The PARTITION BY HASH clause of the CREATE TABLE statement identifies that the table is to be hash-partitioned.**
  - The PARTITIONS clause can then be used to specify the number of partitions to create, and optionally, the tablespaces to store them in.
  - Alternatively, you can use PARTITION clauses to name the individual partitions and their tablespaces.
- **The only attribute you can specify for hash partitions is TABLESPACE. All of the hash partitions of a table must share the same segment attributes (except TABLESPACE), which are inherited from the table level.**

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
PARTITION BY HASH(deptno) PARTITIONS 4;
```

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
PARTITION BY HASH(deptno)
 (PARTITION p1 TABLESPACE ts1,
  PARTITION p2 TABLESPACE ts2,
  PARTITION p3 TABLESPACE ts1,
  PARTITION p4 TABLESPACE ts3);
```

- **If you create a local index for the above table, Oracle constructs the index so that it is equipartitioned with the underlying table. Oracle also ensures that the index is maintained automatically when maintenance operations are performed on the underlying table.**

```
CREATE INDEX loc_dept_ix ON dept(deptno) LOCAL;
```

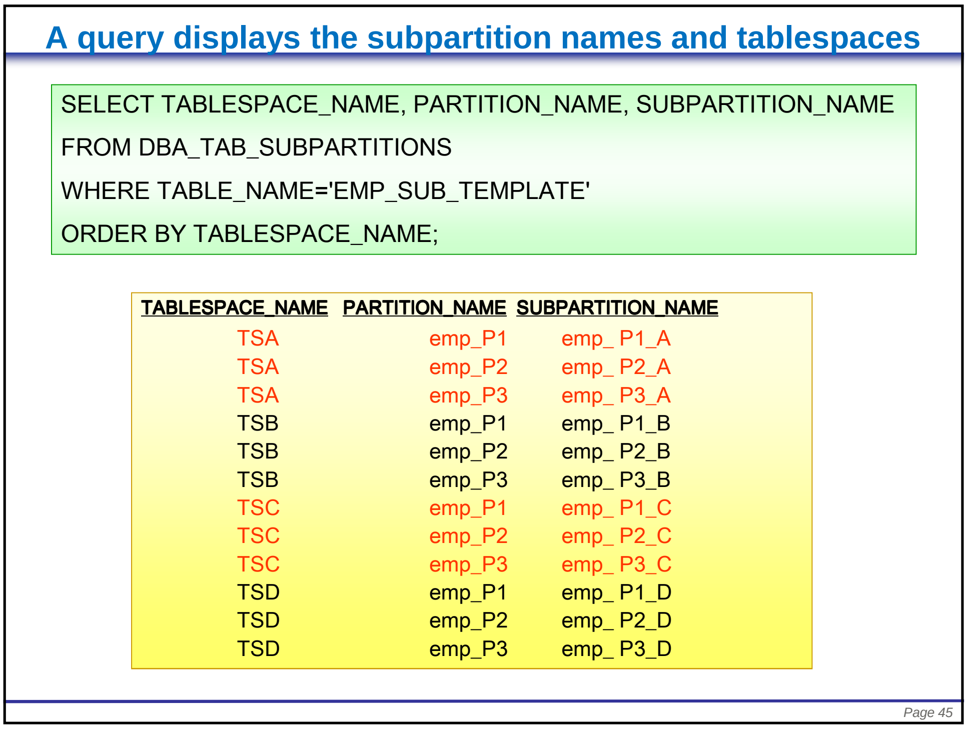
```
SELECT TABLE_NAME, PARTITION_NAME, SUBPARTITION_NAME,
TABLESPACE_NAME
FROM USER_TAB_SUBPARTITIONS
WHERE TABLE_NAME = 'EMP' ORDER BY TABLESPACE_NAME;
```

```
TABLE_NAME  PARTITION_NAME  SUBPARTITION_NAME  TABLESPACE_NAME
EMP                P1              SYS_SUBP41              TS1
EMP                P1              SYS_SUBP45              TS1
EMP                P2              SYS_SUBP49              TS2
EMP                P2              SYS_SUBP53              TS2
EMP                P1              SYS_SUBP42              TS3
EMP                P1              SYS_SUBP46              TS3
EMP                P2              SYS_SUBP50              TS4
EMP                P3              P3_S1                   TS4
EMP                P2              SYS_SUBP54              TS4
EMP                P1              SYS_SUBP43              TS5
EMP                P3              P3_S2                   TS5
EMP                P1              SYS_SUBP47              TS5
EMP                P2              SYS_SUBP51              TS6
EMP                P2              SYS_SUBP55              TS6
EMP                P1              SYS_SUBP44              TS7
EMP                P1              SYS_SUBP48              TS7
EMP                P2              SYS_SUBP52              TS8
EMP                P2              SYS_SUBP56              TS8
18 rows selected.
```

Specifying a Subpartition Template for a Range-Hash Partitioned Table

- **In the case of range-hash partitioned tables, the subpartition template can describe the subpartitions in detail, or it can specify just the number of hash subpartitions.**
  - Every partition has four subpartitions as described in the subpartition template.
  - Each subpartition has a tablespace specified. It is required that if a tablespace is specified for one subpartition in a subpartition template, then one must be specified for all.
  - The names of the subpartitions are generated by concatenating the partition name with the subpartition name in the form: ***partition name_subpartition name***

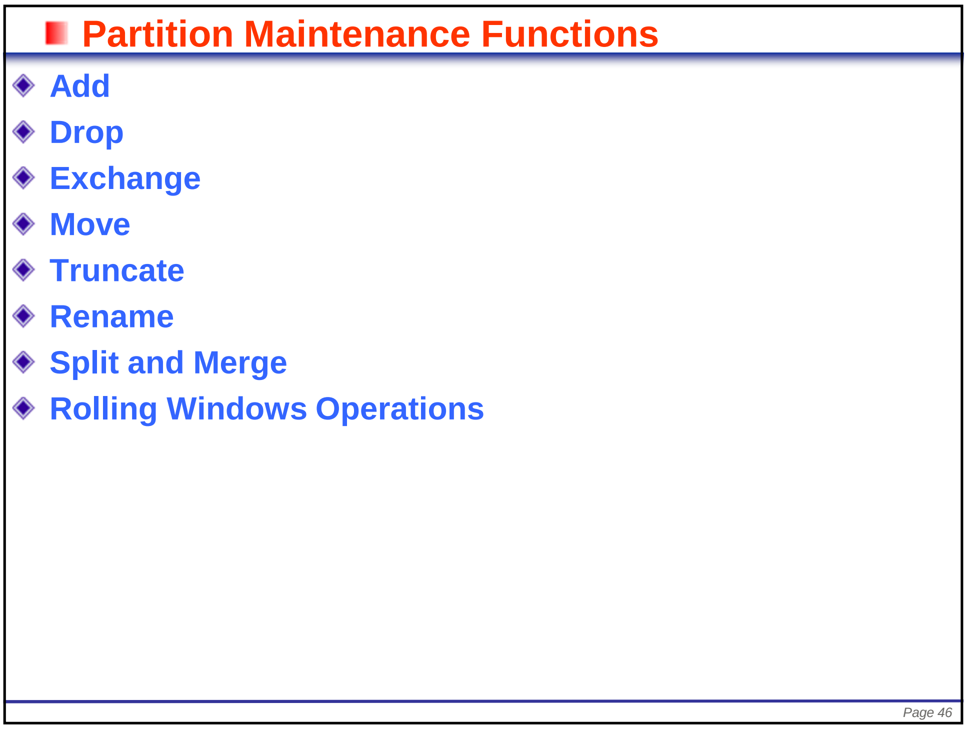
```
CREATE TABLE emp_sub_template
  (deptno NUMBER, empname VARCHAR2(32), grade NUMBER)
PARTITION BY RANGE(deptno)
SUBPARTITION BY HASH(empname)
SUBPARTITION TEMPLATE
        (SUBPARTITION a TABLESPACE tsa,
         SUBPARTITION b TABLESPACE tsb,
         SUBPARTITION c TABLESPACE tsc,
         SUBPARTITION d TABLESPACE tsd )
(PARTITION emp_p1 VALUES LESS THAN (1000),
PARTITION emp_p2 VALUES LESS THAN (2000),
PARTITION emp_p3 VALUES LESS THAN (MAXVALUE) );
```



A query displays the subpartition names and tablespaces

```
SELECT TABLESPACE_NAME, PARTITION_NAME, SUBPARTITION_NAME
FROM DBA_TAB_SUBPARTITIONS
WHERE TABLE_NAME='EMP_SUB_TEMPLATE'
ORDER BY TABLESPACE_NAME;
```

```
TABLESPACE_NAME  PARTITION_NAME  SUBPARTITION_NAME
      TSA              emp_P1        emp_ P1_A
      TSA              emp_P2        emp_ P2_A
      TSA              emp_P3        emp_ P3_A
      TSB              emp_P1        emp_ P1_B
      TSB              emp_P2        emp_ P2_B
      TSB              emp_P3        emp_ P3_B
      TSC              emp_P1        emp_ P1_C
      TSC              emp_P2        emp_ P2_C
      TSC              emp_P3        emp_ P3_C
      TSD              emp_P1        emp_ P1_D
      TSD              emp_P2        emp_ P2_D
      TSD              emp_P3        emp_ P3_D
```



Partition Maintenance Functions

- Add
- Drop
- Exchange
- Move
- Truncate
- Rename
- Split and Merge
- Rolling Windows Operations

❑ Add Partitions

- The following statement adds a partition p3 to the print_media_part table and specifies storage characteristics for the table's BLOB and CLOB columns:

```
ALTER TABLE print_media_part
ADD PARTITION p3 VALUES LESS THAN (MAXVALUE)
LOB (ad_photo, ad_composite) STORE AS (TABLESPACE omf_ts2)
LOB (ad_sourcetext, ad_finaltext) STORE AS (TABLESPACE omf_ts1);
```

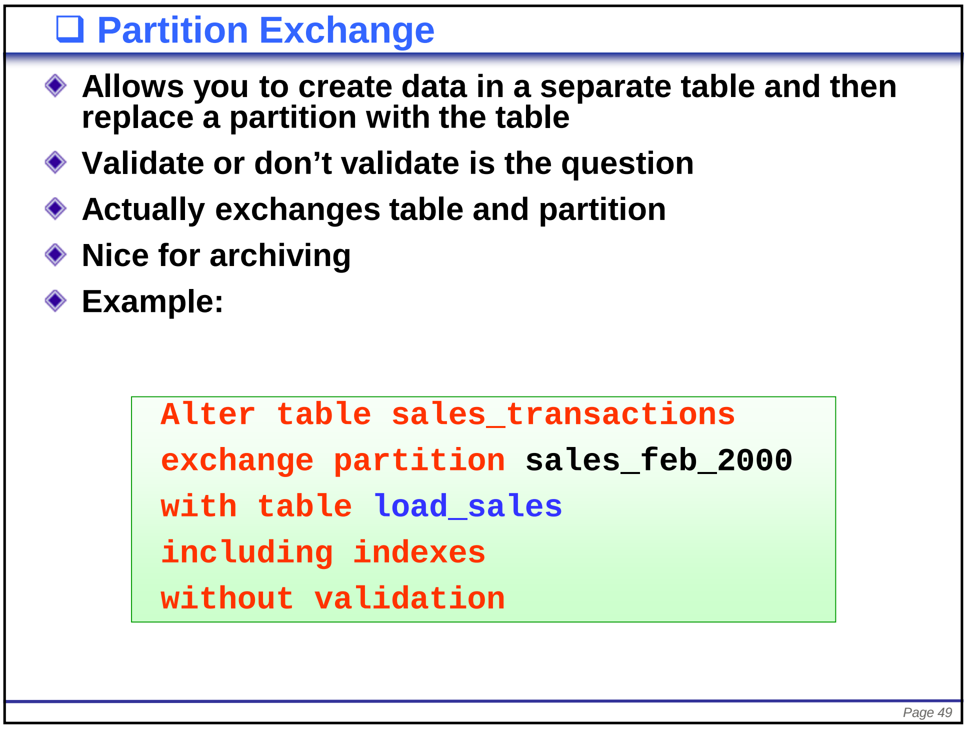
- The LOB data and LOB index segments for columns ad_photo and ad_composite in partition p3 will reside in tablespace omf_ts2. The remaining attributes for these LOB columns will be inherited first from the table-level defaults, and then from the tablespace defaults.
- The LOB data segments for columns ad_source_text and ad_finaltext will reside in the omf_ts1 tablespace, and will inherit all other attributes from the table-level defaults and then from the tablespace defaults.

❑ Drop Partitions

- The *drop_table_partition* clause removes *partition*, and the data in that partition, from a partitioned table. If you want to drop a partition but keep its data in the table, then you must merge the partition into one of the adjacent partitions.

  ```
  ALTER TABLE print_media_part DROP PARTITION p3;
  ```

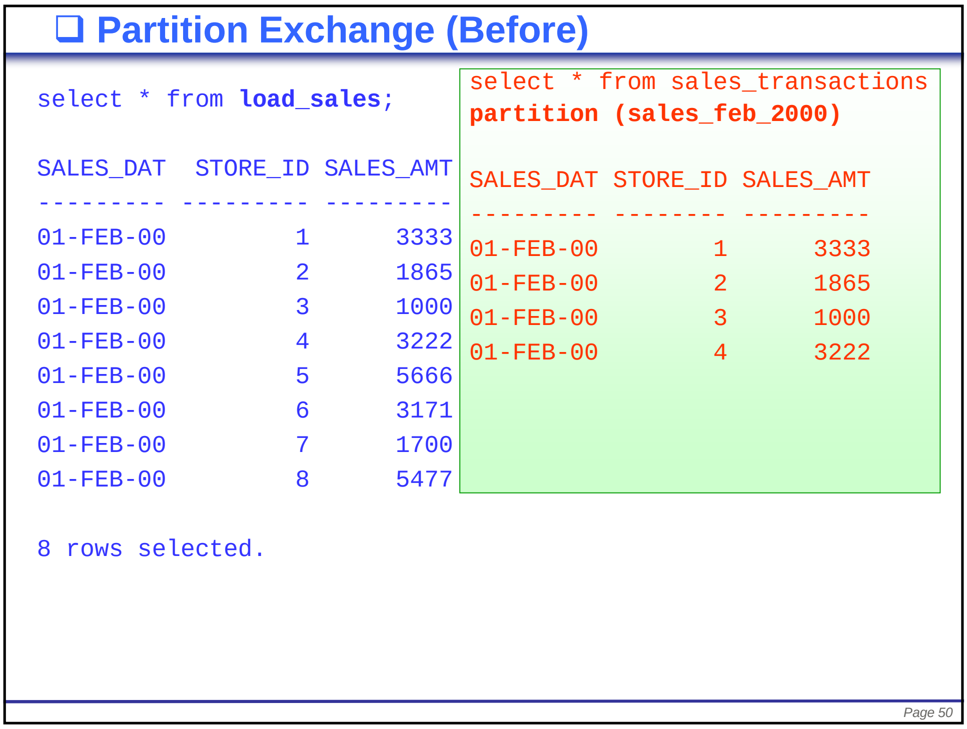
- If the table has LOB columns, then Oracle also drops the LOB data and LOB index partitions (and their subpartitions, if any) corresponding to *partition*.
- If *table* is index organized and has a mapping table defined on it, then Oracle drops the corresponding mapping table partition as well.
- Oracle drops local index partitions and subpartitions corresponding to *partition*, even if they are marked UNUSABLE.
- If you drop a range partition and later insert a row that would have belonged to the dropped partition, then Oracle stores the row in the next higher partition. However, if that partition is the highest partition, then the insert will fail because the range of values represented by the dropped partition is no longer valid for the table.
- Restrictions on Dropping Table Partitions
  - You cannot drop a partition of a hash-partitioned table.
  - If *table* contains only one partition, then you cannot drop the partition. You must drop the table.



❑ Partition Exchange

- Allows you to create data in a separate table and then replace a partition with the table
- Validate or don’t validate is the question
- Actually exchanges table and partition
- Nice for archiving
- Example:

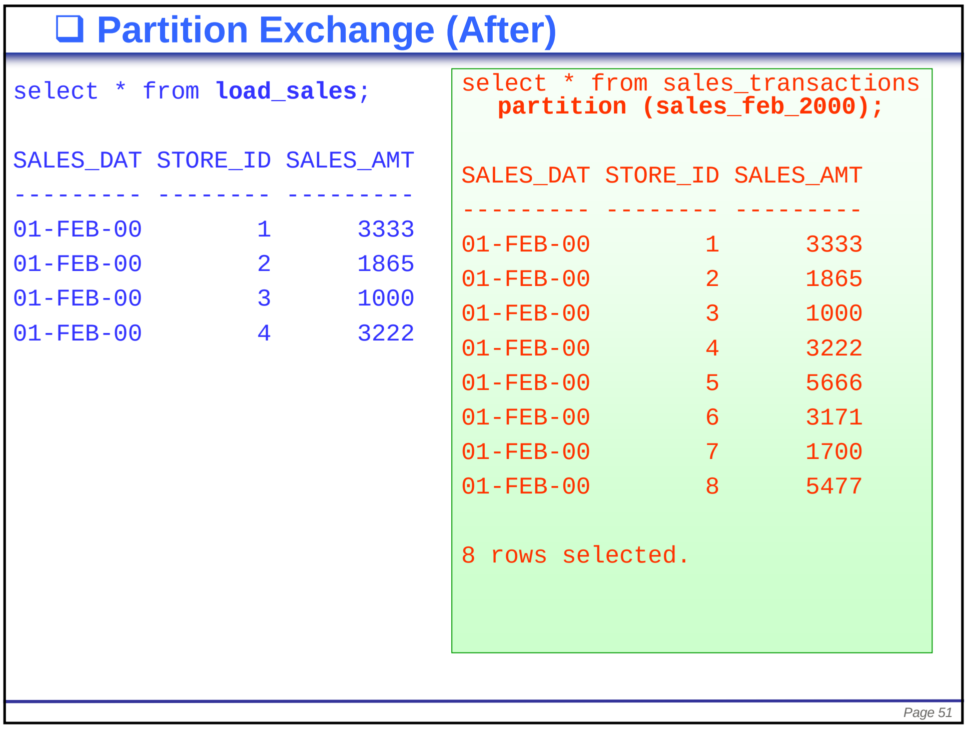
```
Alter table sales_transactions
exchange partition sales_feb_2000
with table load_sales
including indexes
without validation
```



❑ Partition Exchange (Before)

```
select * from load_sales;

SALES_DAT  STORE_ID SALES_AMT
--------- --------- ---------
01-FEB-00         1      3333
01-FEB-00         2      1865
01-FEB-00         3      1000
01-FEB-00         4      3222
01-FEB-00         5      5666
01-FEB-00         6      3171
01-FEB-00         7      1700
01-FEB-00         8      5477

8 rows selected.
```

```
select * from sales_transactions
partition (sales_feb_2000)

SALES_DAT STORE_ID SALES_AMT
--------- -------- ---------
01-FEB-00        1      3333
01-FEB-00        2      1865
01-FEB-00        3      1000
01-FEB-00        4      3222
```

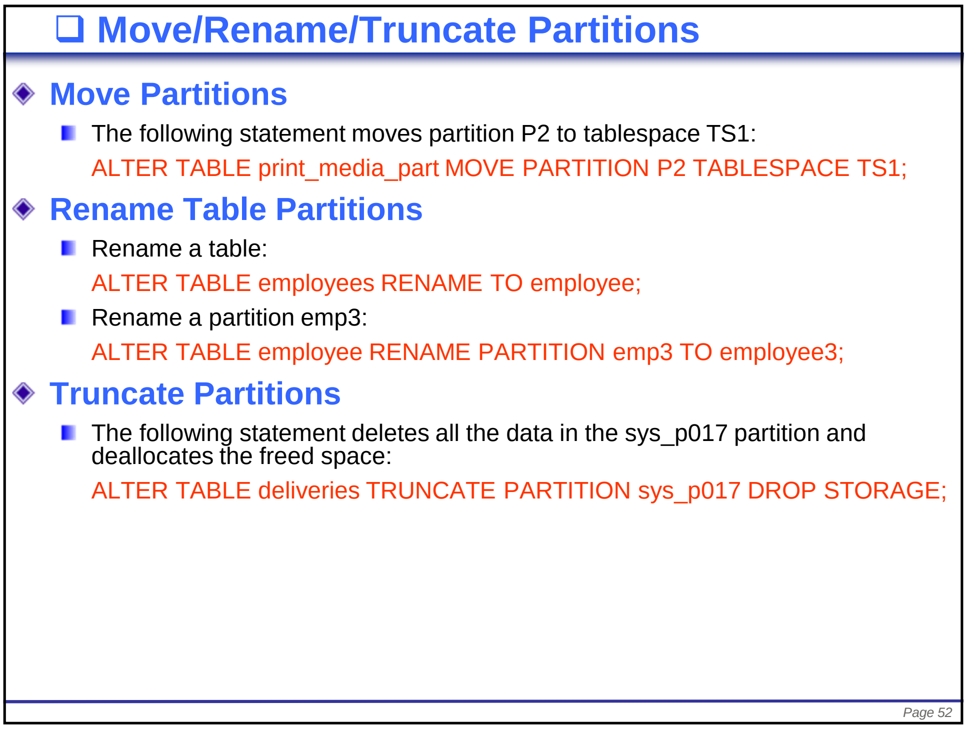


❑ Partition Exchange (After)

```
select * from load_sales;

SALES_DAT STORE_ID SALES_AMT
--------- -------- ---------
01-FEB-00        1      3333
01-FEB-00        2      1865
01-FEB-00        3      1000
01-FEB-00        4      3222
```

```
select * from sales_transactions
   partition (sales_feb_2000);

SALES_DAT STORE_ID SALES_AMT
--------- -------- ---------
01-FEB-00        1      3333
01-FEB-00        2      1865
01-FEB-00        3      1000
01-FEB-00        4      3222
01-FEB-00        5      5666
01-FEB-00        6      3171
01-FEB-00        7      1700
01-FEB-00        8      5477

8 rows selected.
```

Page 51



❑ Move/Rename/Truncate Partitions

Move Partitions

- The following statement moves partition P2 to tablespace TS1:

  ALTER TABLE print_media_part MOVE PARTITION P2 TABLESPACE TS1;

Rename Table Partitions

- Rename a table:

  ALTER TABLE employees RENAME TO employee;

- Rename a partition emp3:

  ALTER TABLE employee RENAME PARTITION emp3 TO employee3;

Truncate Partitions

- The following statement deletes all the data in the sys_p017 partition and deallocates the freed space:

  ALTER TABLE deliveries TRUNCATE PARTITION sys_p017 DROP STORAGE;

Page 52

❑ Split/Merge Partitions

- The following statement splits the old partition sales_q4_2000 in the sample table sh.sales, creating two new partitions, naming one sales_q4_2000b and reusing the name of the old partition for the other:

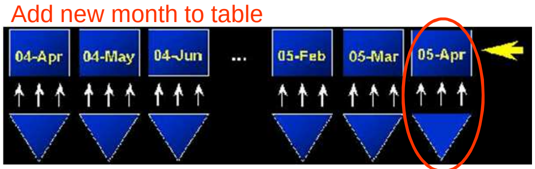
```
ALTER TABLE sales SPLIT PARTITION SALES_Q4_2000
AT (TO_DATE('15-NOV-2000','DD-MON-YYYY'))
INTO (PARTITION SALES_Q4_2000, PARTITION SALES_Q4_2000b);
```

- The following statement merges back into one partition

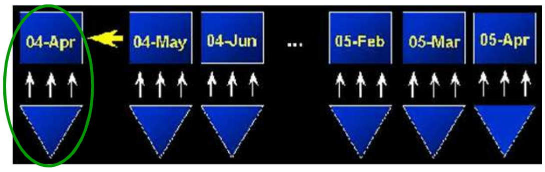
```
ALTER TABLE sales MERGE PARTITIONS SALES_Q4_2000,
SALES_Q4_2000b
INTO PARTITION SALES_Q4_2000;
```

❑ Rolling Windows Operations

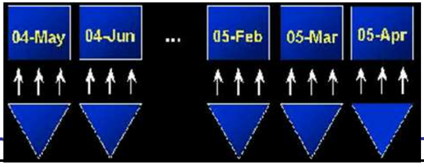
- **Partitioned Tables with Local Indexes:**
  - Load and index new month
  - Add new month to table



  - Remove old month from table



  - New data has been loaded with virtually no disruption
  - Powerful methodology for managing time-based updates to the Warehouse


04-May
04-Jun
...
05-Feb
05-Mar
05-Apr