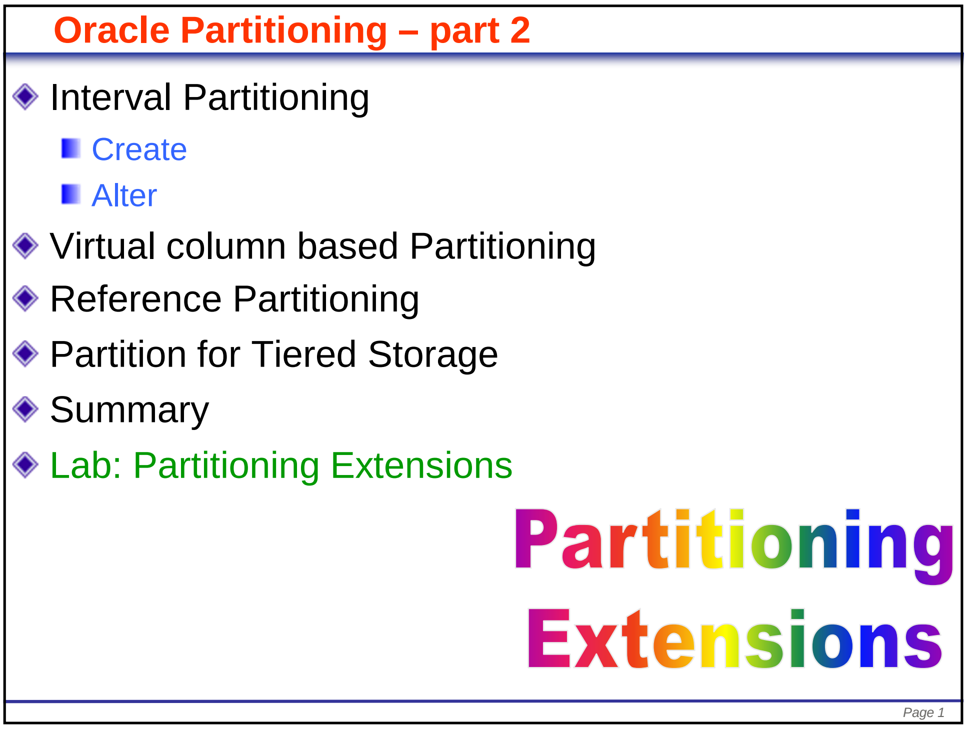
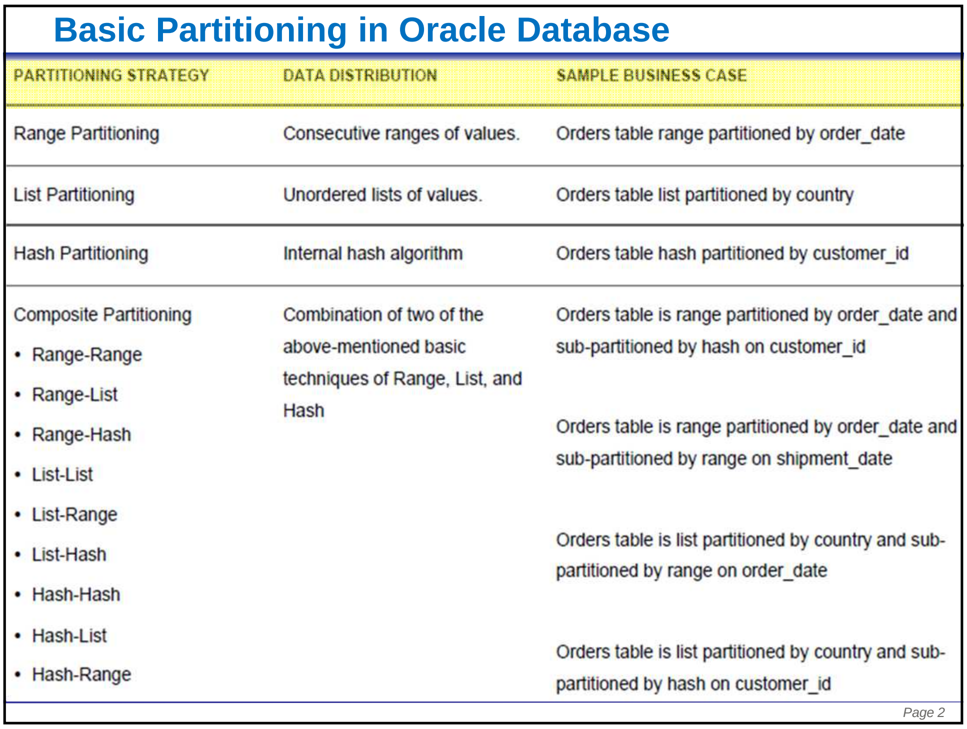


Oracle Partitioning – part 2

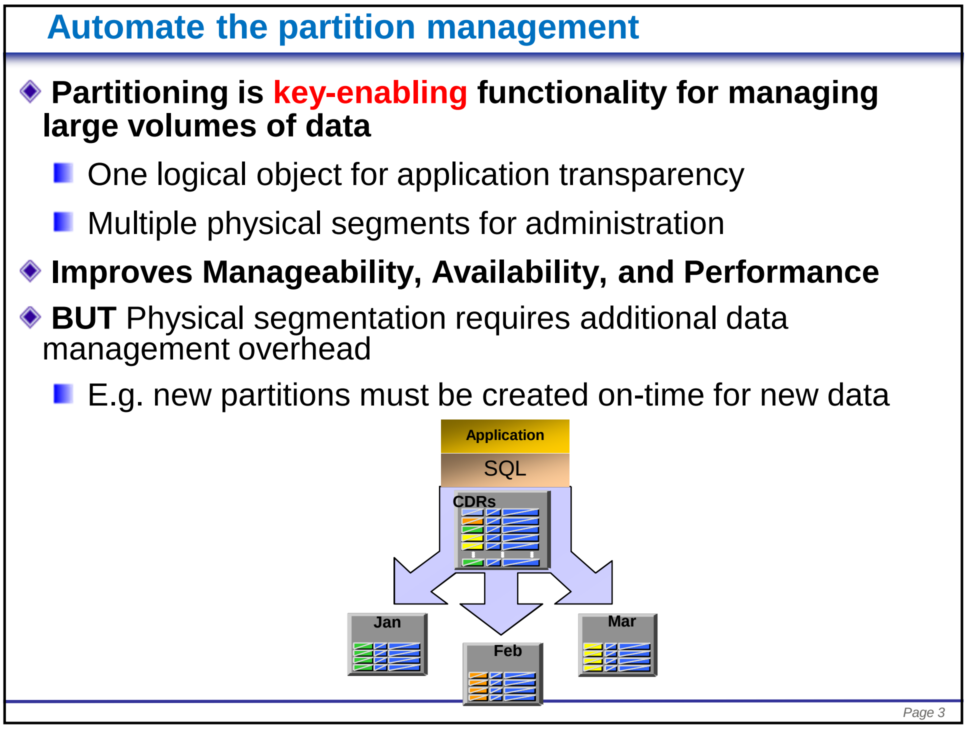
- Interval Partitioning
  - Create
  - Alter
- Virtual column based Partitioning
- Reference Partitioning
- Partition for Tiered Storage
- Summary
- Lab: Partitioning Extensions

Partitioning Extensions



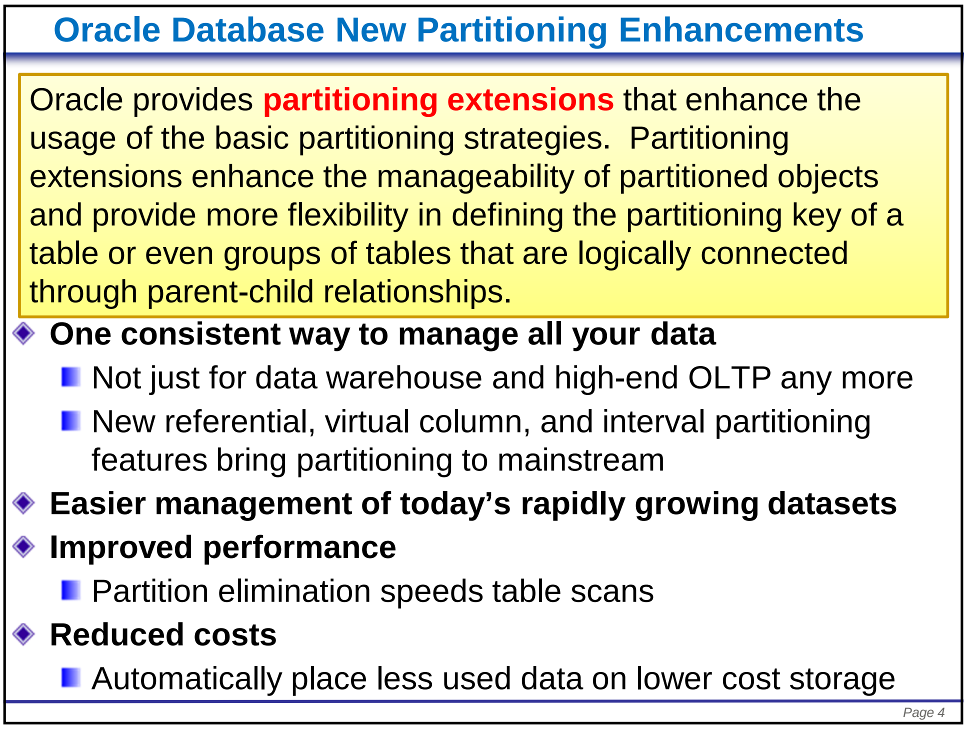
Basic Partitioning in Oracle Database

PARTITIONING STRATEGY	DATA DISTRIBUTION	SAMPLE BUSINESS CASE
Range Partitioning	Consecutive ranges of values.	Orders table range partitioned by order_date
List Partitioning	Unordered lists of values.	Orders table list partitioned by country
Hash Partitioning	Internal hash algorithm	Orders table hash partitioned by customer_id
Composite Partitioning • Range-Range • Range-List • Range-Hash • List-List • List-Range • List-Hash • Hash-Hash • Hash-List • Hash-Range	Combination of two of the above-mentioned basic techniques of Range, List, and Hash	Orders table is range partitioned by order_date and sub-partitioned by hash on customer_id Orders table is range partitioned by order_date and sub-partitioned by range on shipment_date Orders table is list partitioned by country and sub-partitioned by range on order_date Orders table is list partitioned by country and sub-partitioned by hash on customer_id



Automate the partition management

- **Partitioning is key-enabling functionality for managing large volumes of data**
  - One logical object for application transparency
  - Multiple physical segments for administration
- **Improves Manageability, Availability, and Performance**
- **BUT** Physical segmentation requires additional data management overhead
  - E.g. new partitions must be created on-time for new data



Oracle Database New Partitioning Enhancements

Oracle provides partitioning extensions that enhance the usage of the basic partitioning strategies. Partitioning extensions enhance the manageability of partitioned objects and provide more flexibility in defining the partitioning key of a table or even groups of tables that are logically connected through parent-child relationships.

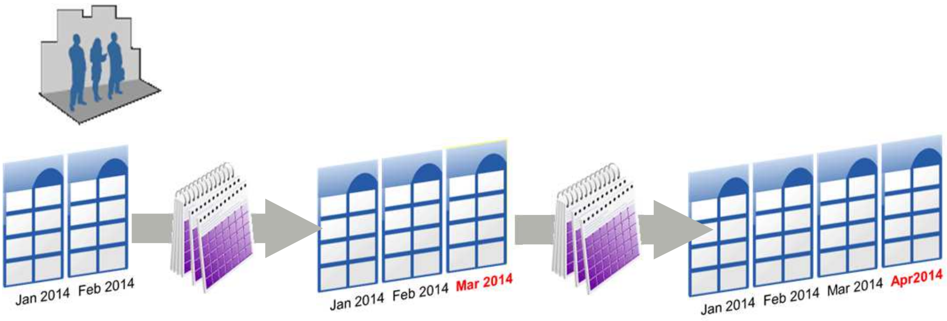
- **One consistent way to manage all your data**
  - Not just for data warehouse and high-end OLTP any more
  - New referential, virtual column, and interval partitioning features bring partitioning to mainstream
- **Easier management of today's rapidly growing datasets**
- **Improved performance**
  - Partition elimination speeds table scans
- **Reduced costs**
  - Automatically place less used data on lower cost storage

❑ Interval Partitioning

- Managing the creation of new partitions can be a cumbersome and highly repetitive task. This is especially true for predictable additions of partitions covering small ranges, such as adding new daily partitions.
- Interval partitioning automates this operation by creating partitions **on-demand**.
- This new partitioning feature fully automates the partition creation for **range**.
  - New partitions will be created when they are needed.
  - By defining the interval criteria, the database knows when to create new partitions for new or modified data.
- You can create single-level interval partitioned tables as well as the following composite partitioned tables:
  - **Interval-range**
  - **Interval-hash**
  - **Interval-list**

Interval Partitioning

- **Automatic interval partitioning**
  - Partitions are created automatically as data arrives
  - Example: Create new partition every month

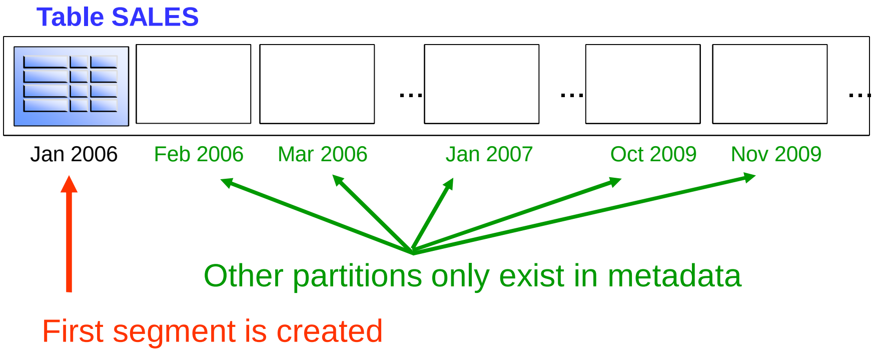


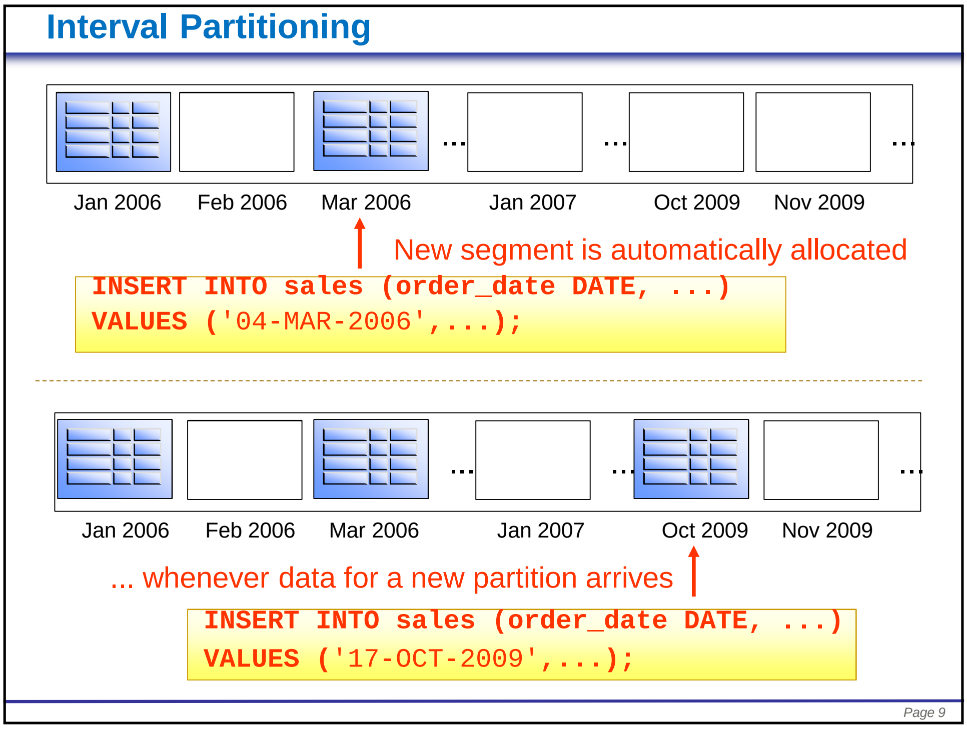
Extension of range partitioning

- Interval partitioning is an **extension of range partitioning** which instructs the database to automatically create partitions of a specified interval when data inserted into the table exceeds all of the existing range partitions.
- You must specify at least **one range partition**. The range partitioning key value determines the high value of the range partitions, which is called the *transition point*, and the database creates interval partitions for data beyond that transition point.
- The lower boundary of every interval partition is the non-inclusive upper boundary of the previous range or interval partition.
- For example, if you create an interval partitioned table with monthly intervals and the transition point at January 1, 2007, then the lower boundary for the January 2007 interval is January 1, 2007. The lower boundary for the July 2007 interval is July 1, 2007, regardless of whether the June 2007 partition was already created.
- You can only specify **one partitioning key column**, and it must be of **NUMBER** or **DATE** type.
- Interval partitioning is not supported for index-organized tables.
- You cannot create a domain index on an interval-partitioned table.

Interval Partitioning

```
CREATE TABLE sales (order_date DATE, ...)
PARTITON BY RANGE (order_date)
INTERVAL(MonthInt(1,'month')
(PARTITION p_first VALUES LESS THAN ('01-JAN-2006');
```





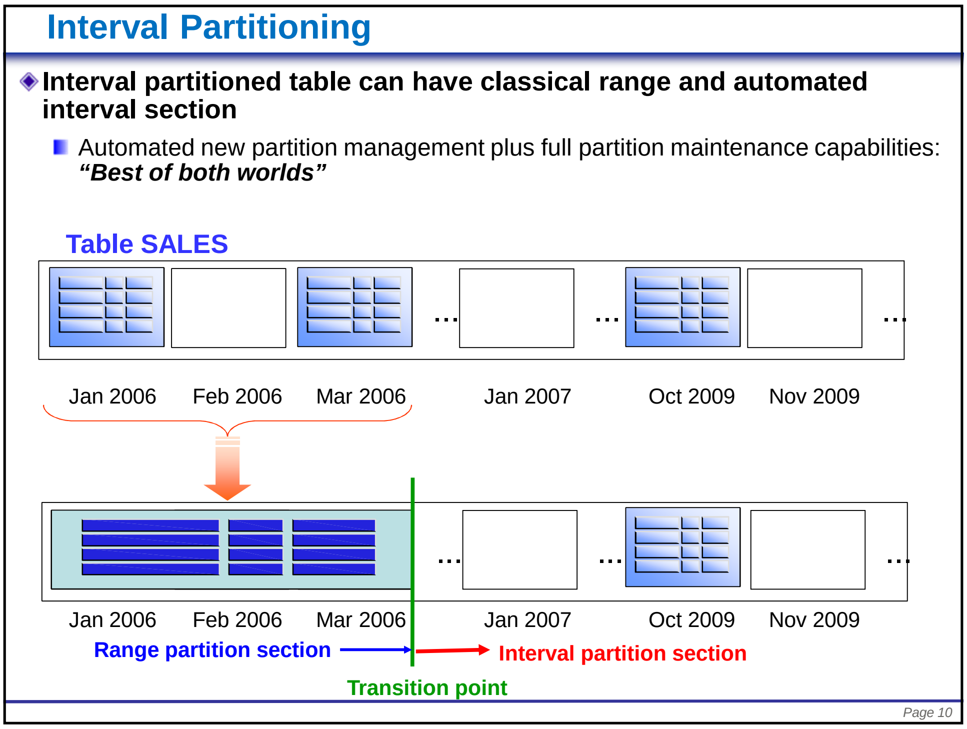
Interval Partitioning

New segment is automatically allocated

```
INSERT INTO sales (order_date DATE, ...)
VALUES ('04-MAR-2006',...);
```

... whenever data for a new partition arrives

```
INSERT INTO sales (order_date DATE, ...)
VALUES ('17-OCT-2009',...);
```



Interval Partitioning

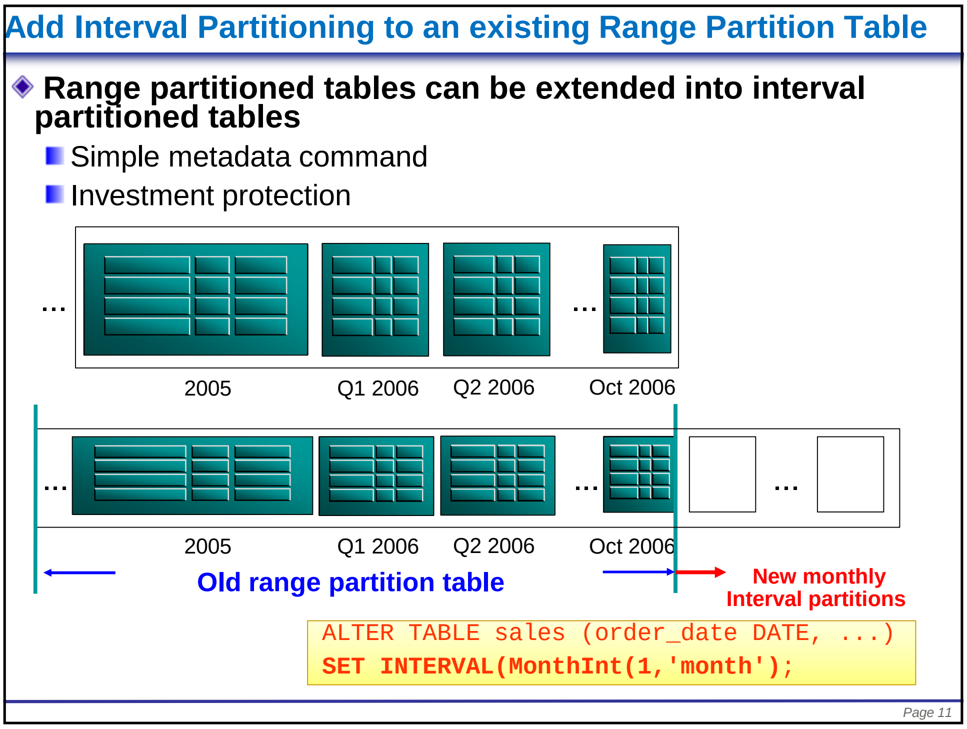
- Interval partitioned table can have classical range and automated interval section
  - Automated new partition management plus full partition maintenance capabilities: ***"Best of both worlds"***

Table SALES

Range partition section

Interval partition section

Transition point



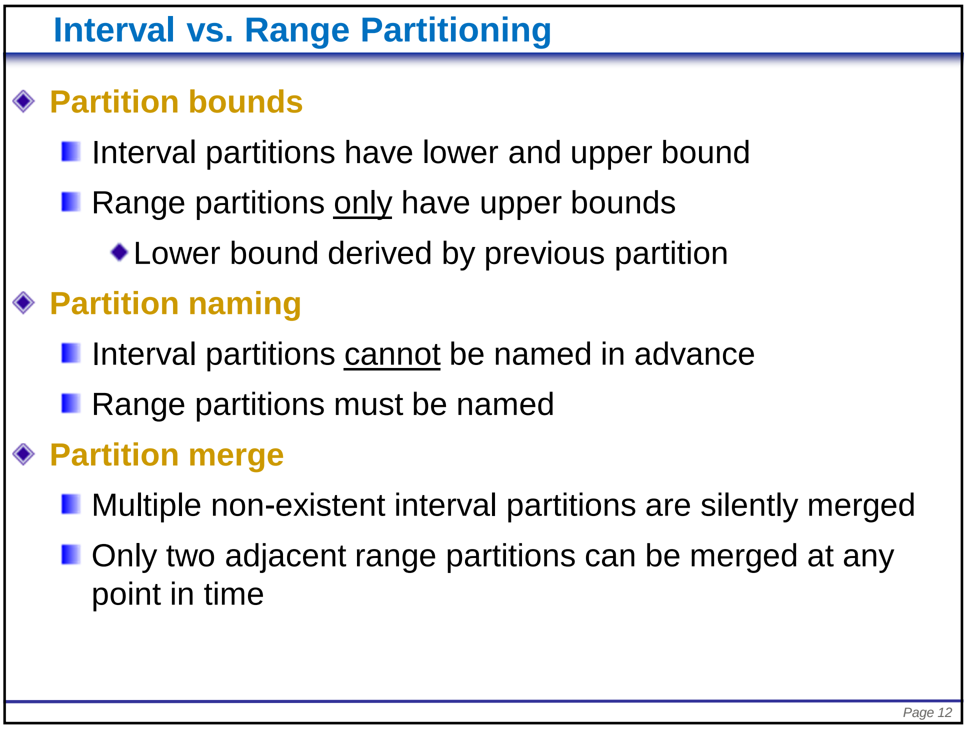
Add Interval Partitioning to an existing Range Partition Table

- **Range partitioned tables can be extended into interval partitioned tables**
  - Simple metadata command
  - Investment protection

Old range partition table

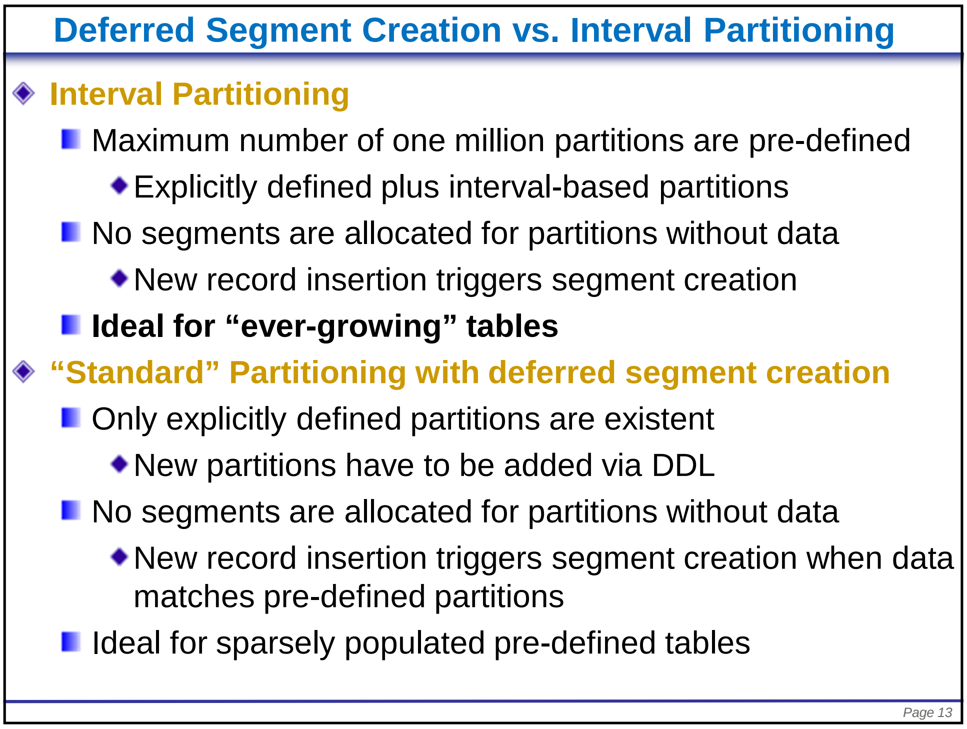
New monthly Interval partitions

```
ALTER TABLE sales (order_date DATE, ...)
SET INTERVAL(MonthInt(1,'month');
```



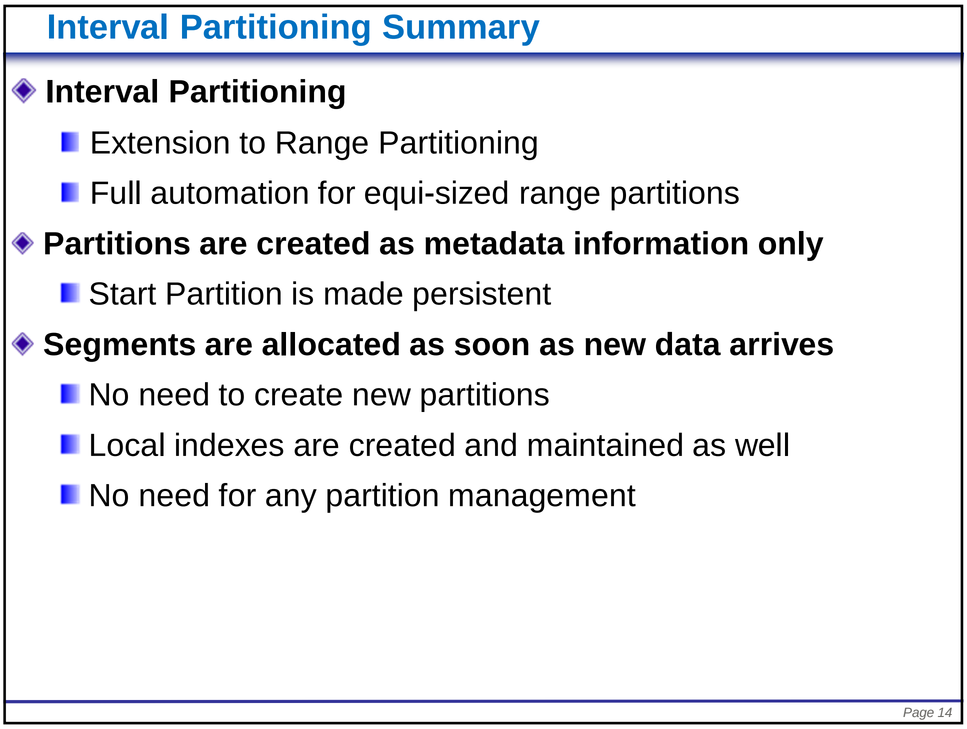
Interval vs. Range Partitioning

- **Partition bounds**
  - Interval partitions have lower and upper bound
  - Range partitions only have upper bounds
    - Lower bound derived by previous partition
- **Partition naming**
  - Interval partitions cannot be named in advance
  - Range partitions must be named
- **Partition merge**
  - Multiple non-existent interval partitions are silently merged
  - Only two adjacent range partitions can be merged at any point in time



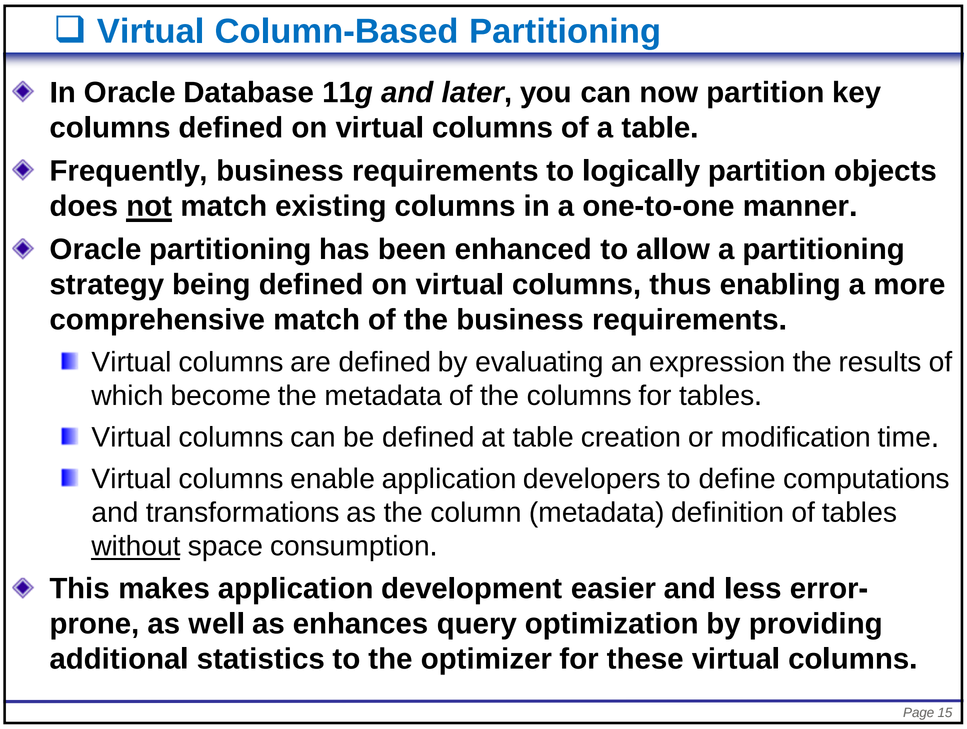
Deferred Segment Creation vs. Interval Partitioning

- **Interval Partitioning**
  - Maximum number of one million partitions are pre-defined
    - Explicitly defined plus interval-based partitions
  - No segments are allocated for partitions without data
    - New record insertion triggers segment creation
  - **Ideal for “ever-growing” tables**
- **“Standard” Partitioning with deferred segment creation**
  - Only explicitly defined partitions are existent
    - New partitions have to be added via DDL
  - No segments are allocated for partitions without data
    - New record insertion triggers segment creation when data matches pre-defined partitions
  - Ideal for sparsely populated pre-defined tables



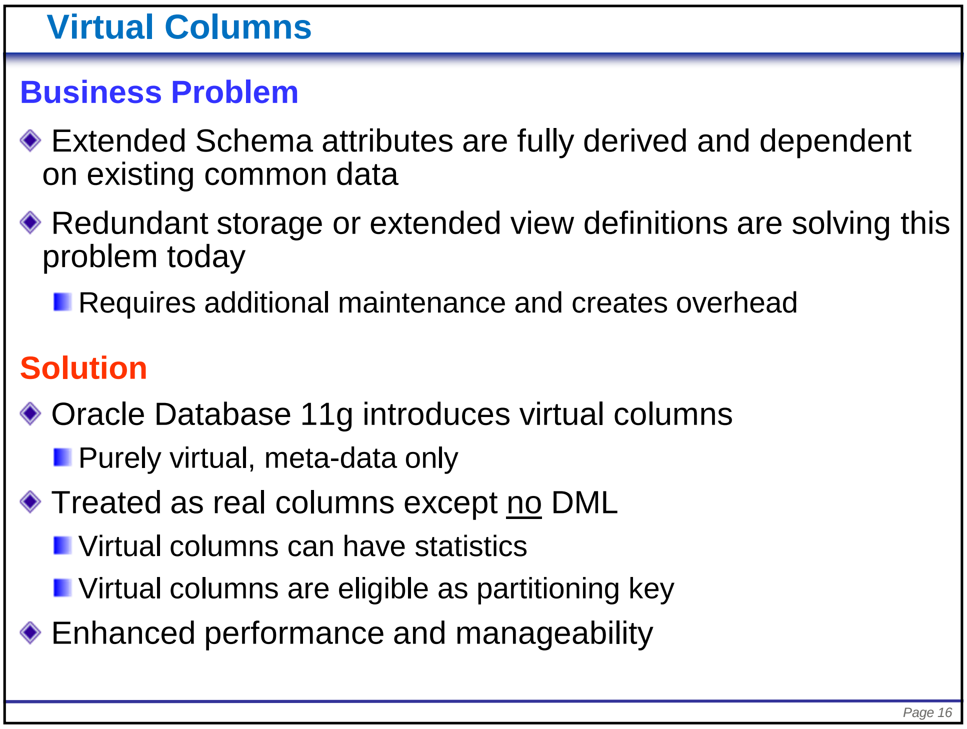
Interval Partitioning Summary

- **Interval Partitioning**
  - Extension to Range Partitioning
  - Full automation for equi-sized range partitions
- **Partitions are created as metadata information only**
  - Start Partition is made persistent
- **Segments are allocated as soon as new data arrives**
  - No need to create new partitions
  - Local indexes are created and maintained as well
  - No need for any partition management



❑ Virtual Column-Based Partitioning

- **In Oracle Database 11*g and later*, you can now partition key columns defined on virtual columns of a table.**
- **Frequently, business requirements to logically partition objects does not match existing columns in a one-to-one manner.**
- **Oracle partitioning has been enhanced to allow a partitioning strategy being defined on virtual columns, thus enabling a more comprehensive match of the business requirements.**
  - Virtual columns are defined by evaluating an expression the results of which become the metadata of the columns for tables.
  - Virtual columns can be defined at table creation or modification time.
  - Virtual columns enable application developers to define computations and transformations as the column (metadata) definition of tables without space consumption.
- **This makes application development easier and less error-prone, as well as enhances query optimization by providing additional statistics to the optimizer for these virtual columns.**



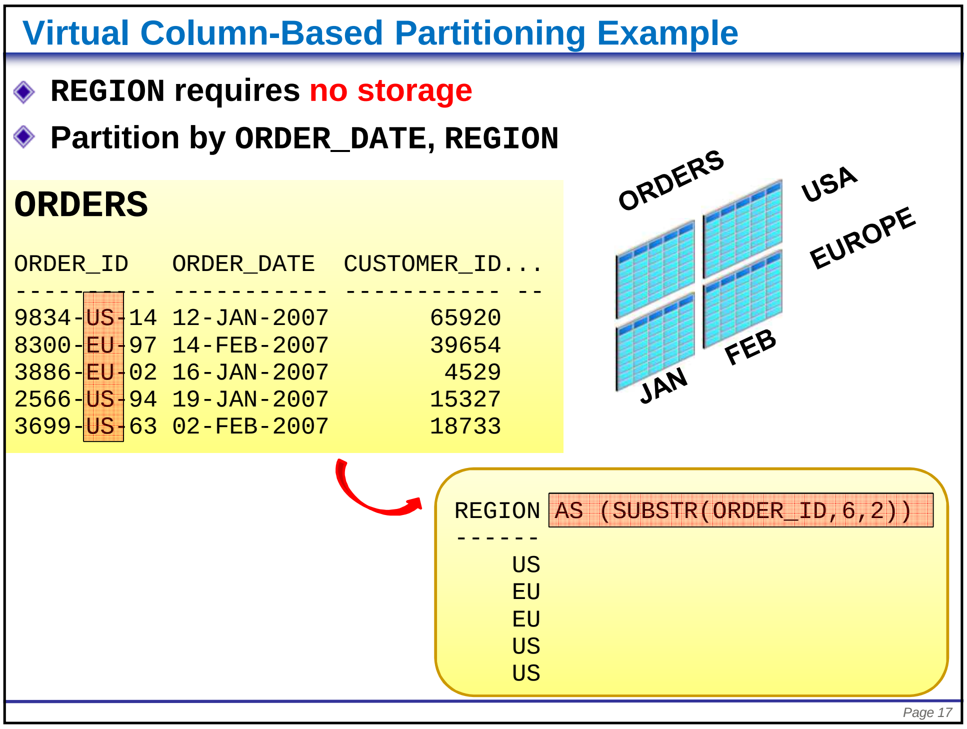
Virtual Columns

Business Problem

- Extended Schema attributes are fully derived and dependent on existing common data
- Redundant storage or extended view definitions are solving this problem today
  - Requires additional maintenance and creates overhead

Solution

- Oracle Database 11g introduces virtual columns
  - Purely virtual, meta-data only
- Treated as real columns except no DML
  - Virtual columns can have statistics
  - Virtual columns are eligible as partitioning key
- Enhanced performance and manageability



Virtual Column-Based Partitioning Example

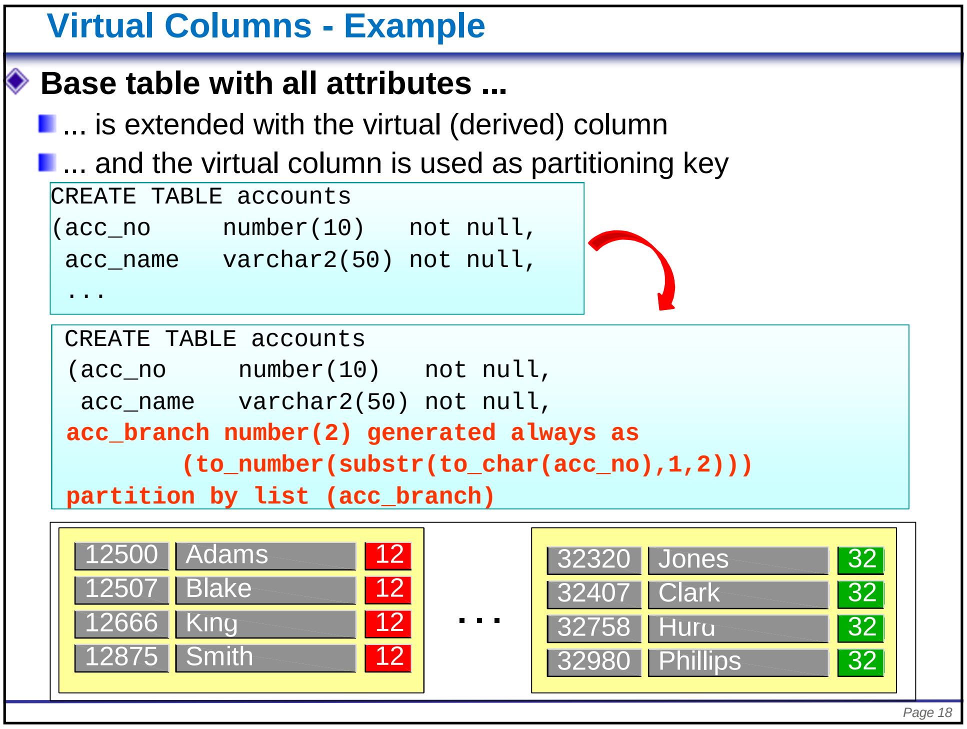
- **REGION requires no storage**
- **Partition by ORDER_DATE, REGION**

ORDERS

```
ORDER_ID    ORDER_DATE  CUSTOMER_ID...
----------- ----------- ----------- --
9834-US-14 12-JAN-2007       65920
8300-EU-97 14-FEB-2007       39654
3886-EU-02 16-JAN-2007        4529
2566-US-94 19-JAN-2007       15327
3699-US-63 02-FEB-2007       18733
```

ORDERS
USA
EUROPE
JAN
FEB

```
REGION AS (SUBSTR(ORDER_ID,6,2))
------
    US
    EU
    EU
    US
    US
```



Virtual Columns - Example

- **Base table with all attributes ...**
  - ... is extended with the virtual (derived) column
  - ... and the virtual column is used as partitioning key

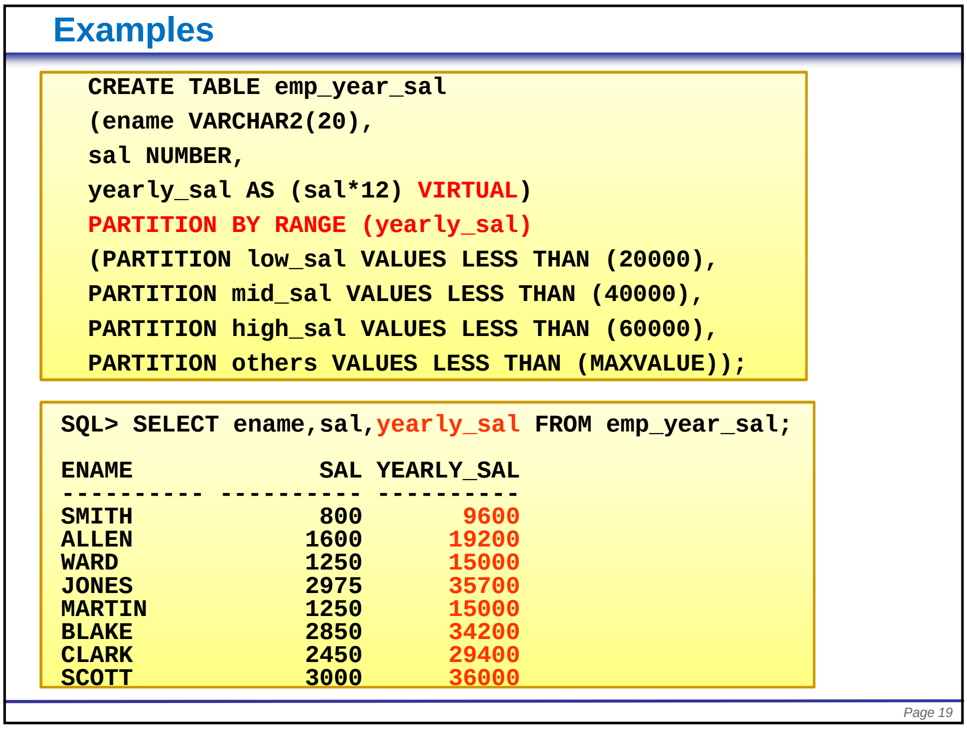
```
CREATE TABLE accounts
(acc_no     number(10)   not null,
 acc_name   varchar2(50) not null,
 ...
```

```
CREATE TABLE accounts
(acc_no     number(10)   not null,
 acc_name   varchar2(50) not null,
acc_branch number(2) generated always as
        (to_number(substr(to_char(acc_no),1,2)))
partition by list (acc_branch)
```

12500	Adams	12
12507	Blake	12
12666	Kıng	12
12875	Smith	12

...

32320	Jones	32
32407	Clark	32
32758	Huru	32
32980	Phillips	32

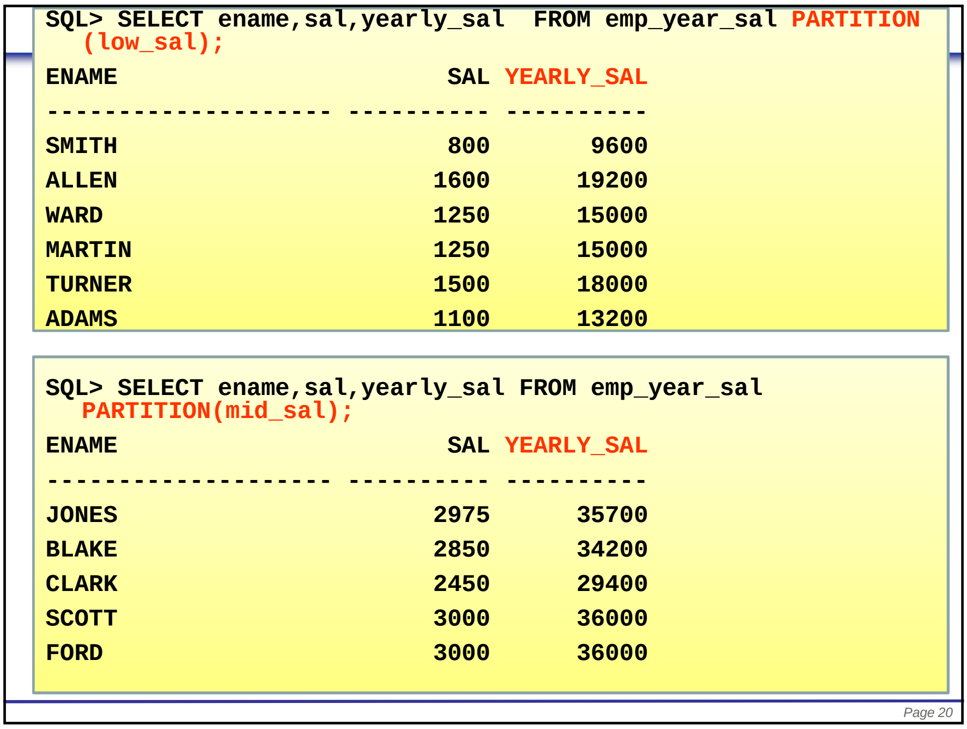


Examples

```
CREATE TABLE emp_year_sal
(ename VARCHAR2(20),
sal NUMBER,
yearly_sal AS (sal*12) VIRTUAL)
PARTITION BY RANGE (yearly_sal)
(PARTITION low_sal VALUES LESS THAN (20000),
PARTITION mid_sal VALUES LESS THAN (40000),
PARTITION high_sal VALUES LESS THAN (60000),
PARTITION others VALUES LESS THAN (MAXVALUE));
```

```
SQL> SELECT ename,sal,yearly_sal FROM emp_year_sal;

ENAME             SAL YEARLY_SAL
---------- ---------- ----------
SMITH             800       9600
ALLEN            1600      19200
WARD             1250      15000
JONES            2975      35700
MARTIN           1250      15000
BLAKE            2850      34200
CLARK            2450      29400
SCOTT            3000      36000
```



```
SQL> SELECT ename,sal,yearly_sal  FROM emp_year_sal PARTITION
   (low_sal);
ENAME                       SAL YEARLY_SAL
-------------------- ---------- ----------
SMITH                       800       9600
ALLEN                      1600      19200
WARD                       1250      15000
MARTIN                     1250      15000
TURNER                     1500      18000
ADAMS                      1100      13200
```

```
SQL> SELECT ename,sal,yearly_sal FROM emp_year_sal
   PARTITION(mid_sal);
ENAME                       SAL YEARLY_SAL
-------------------- ---------- ----------
JONES                      2975      35700
BLAKE                      2850      34200
CLARK                      2450      29400
SCOTT                      3000      36000
FORD                       3000      36000
```

❑ REF Partitioning

Business Problem

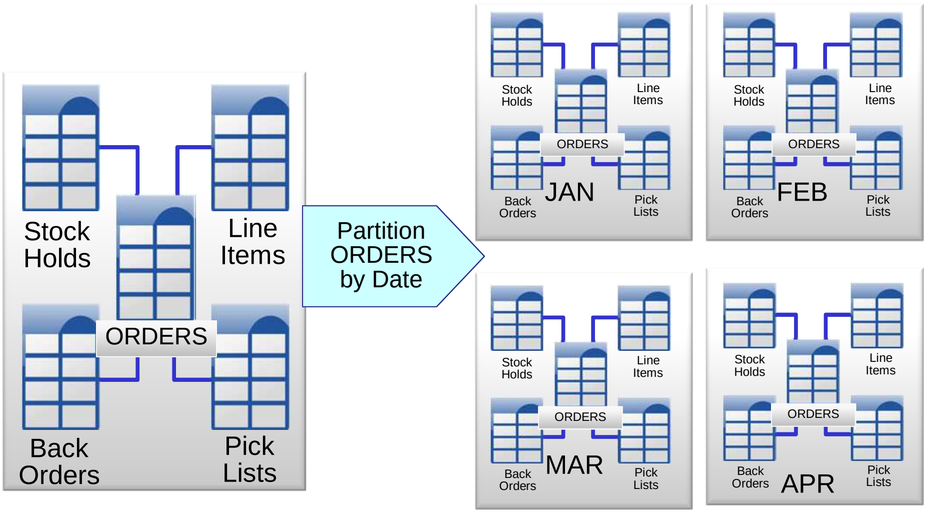
- **Related tables benefit from same partitioning strategy**
  - Sample order – lineitem
- **Redundant storage of the same information solves this problem**
  - Data overhead
  - Maintenance overhead

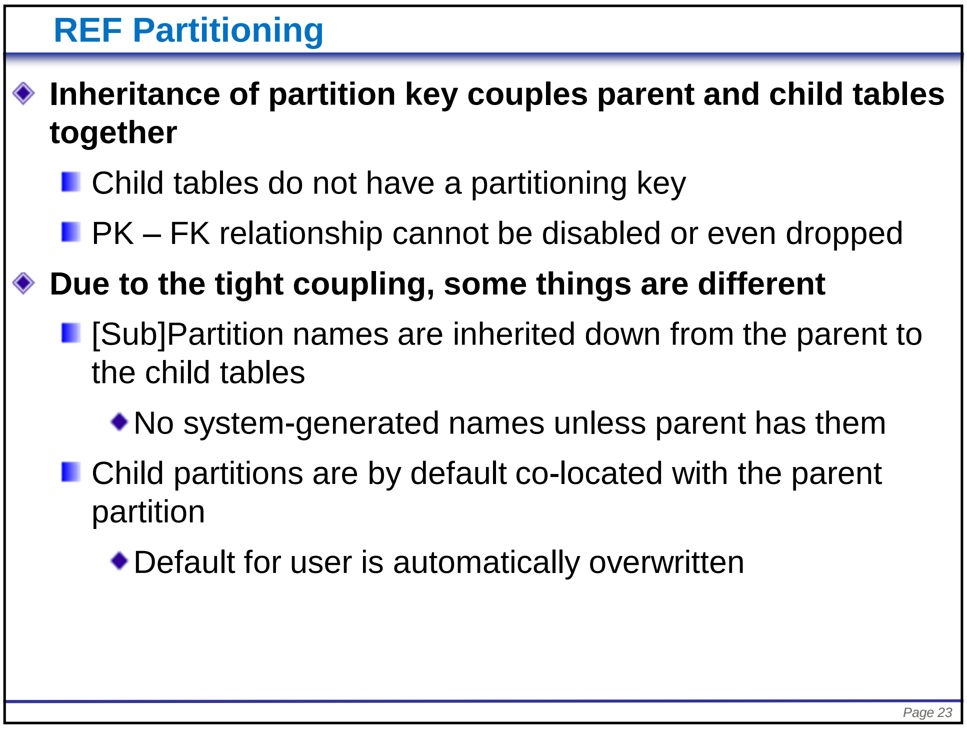
Solution: REF Partitioning

- **Intuitive modelling**
  - Child table inherits the partitioning strategy of parent table through PK-FK relationship
  - The partitioning key is resolved through an existing parent-child relationship, enforced by active primary key or foreign key constraints.
- **Enhanced Performance and Manageability**
  - The benefit of this feature is that tables with a parent-child relationship can be logically equi-partitioned by inheriting the partition key from the parent table without duplicating the key columns.
  - The logical dependency will also automatically cascade partition maintenance operations, thus making application development easier and less error-prone.

REF Partitioning

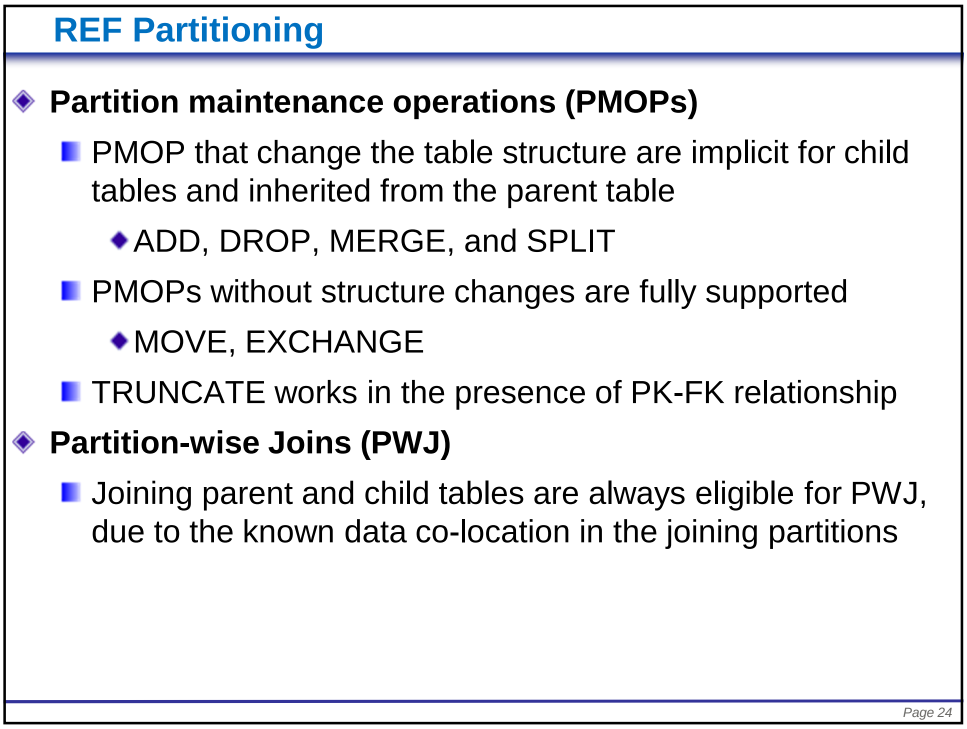
- **Inherit partitioning strategy**





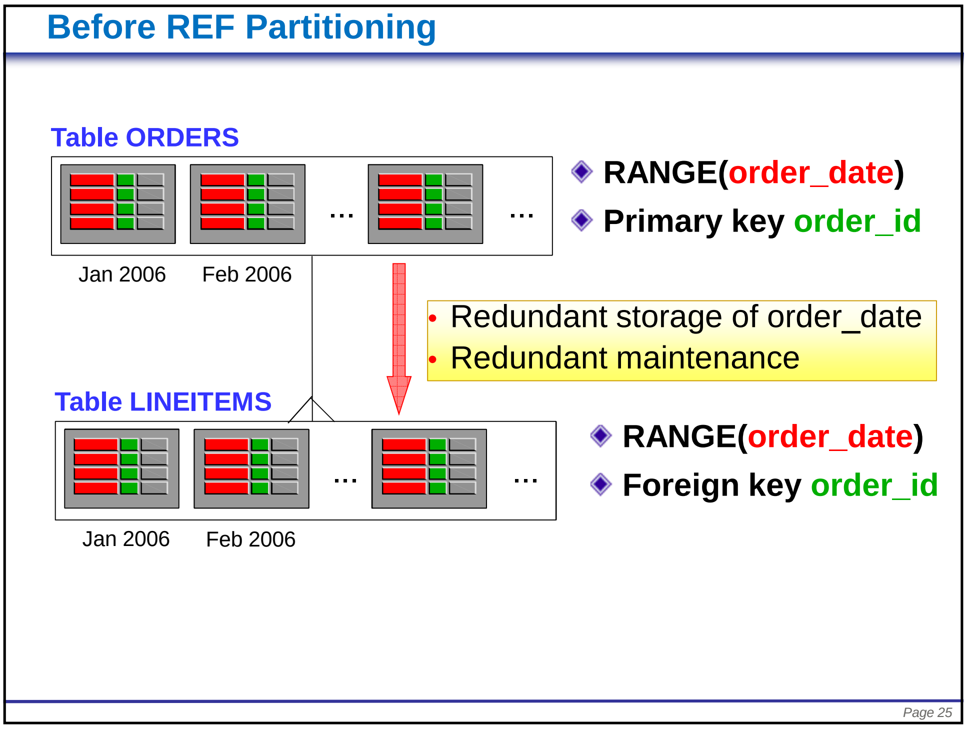
REF Partitioning

- **Inheritance of partition key couples parent and child tables together**
  - Child tables do not have a partitioning key
  - PK – FK relationship cannot be disabled or even dropped
- **Due to the tight coupling, some things are different**
  - [Sub]Partition names are inherited down from the parent to the child tables
    - No system-generated names unless parent has them
  - Child partitions are by default co-located with the parent partition
    - Default for user is automatically overwritten



REF Partitioning

- **Partition maintenance operations (PMOPs)**
  - PMOP that change the table structure are implicit for child tables and inherited from the parent table
    - ADD, DROP, MERGE, and SPLIT
  - PMOPs without structure changes are fully supported
    - MOVE, EXCHANGE
  - TRUNCATE works in the presence of PK-FK relationship
- **Partition-wise Joins (PWJ)**
  - Joining parent and child tables are always eligible for PWJ, due to the known data co-location in the joining partitions



Before REF Partitioning

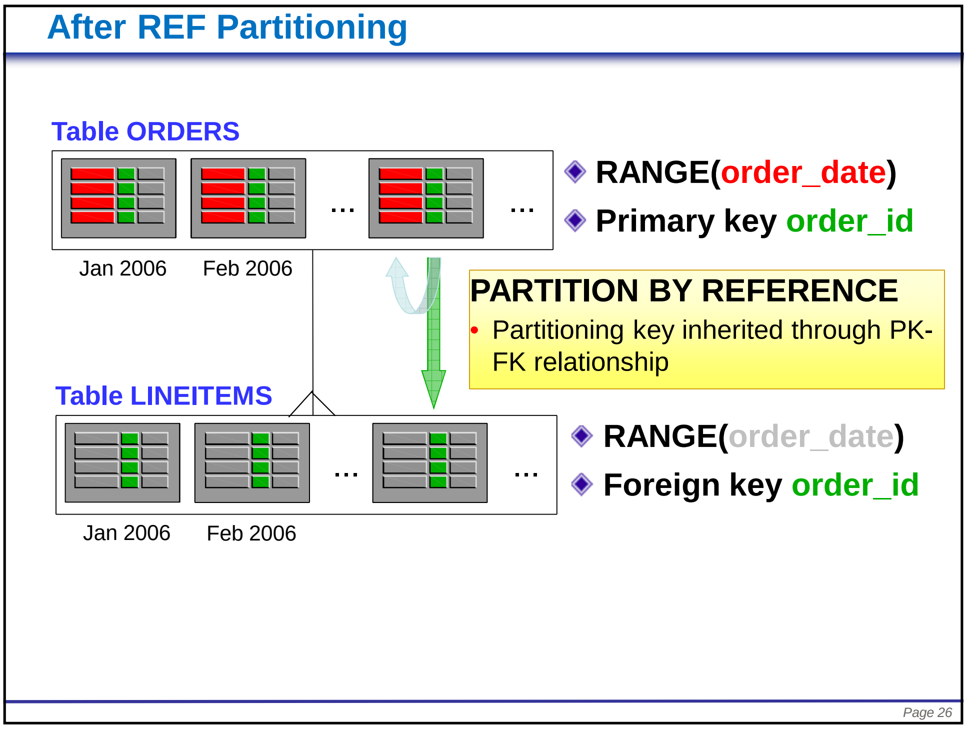
Table ORDERS

- RANGE(order_date)
- Primary key order_id

- Redundant storage of order_date
- Redundant maintenance

Table LINEITEMS

- RANGE(order_date)
- Foreign key order_id



After REF Partitioning

Table ORDERS

- RANGE(order_date)
- Primary key order_id

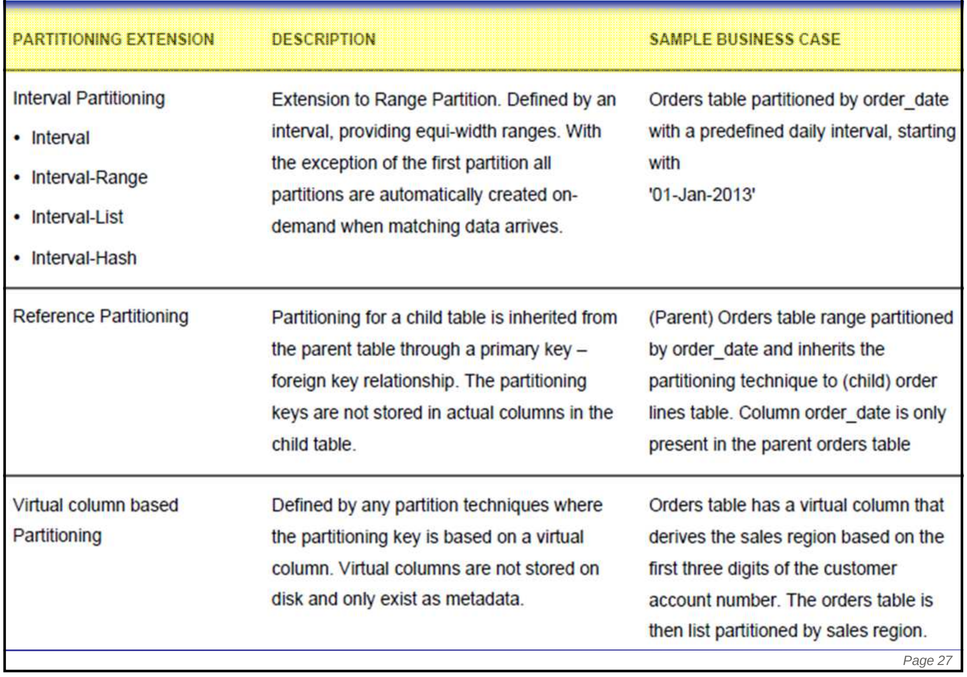
PARTITION BY REFERENCE

- Partitioning key inherited through PK-FK relationship

Table LINEITEMS

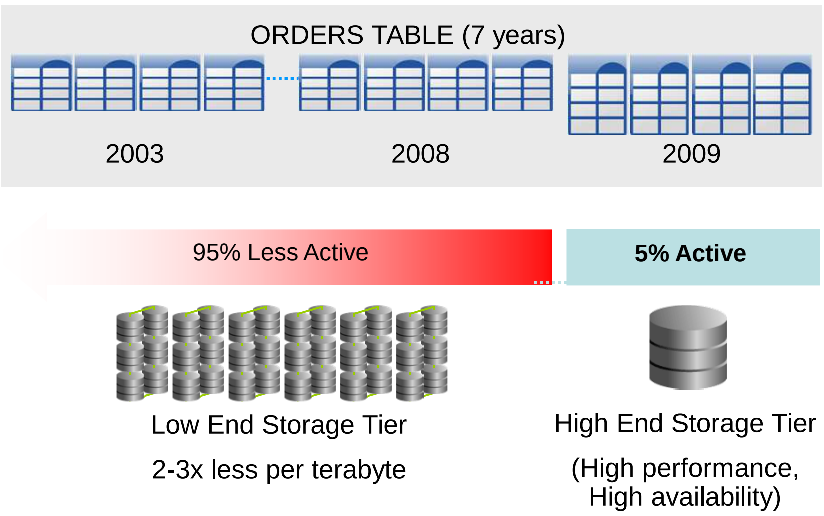
- RANGE(order_date)
- Foreign key order_id

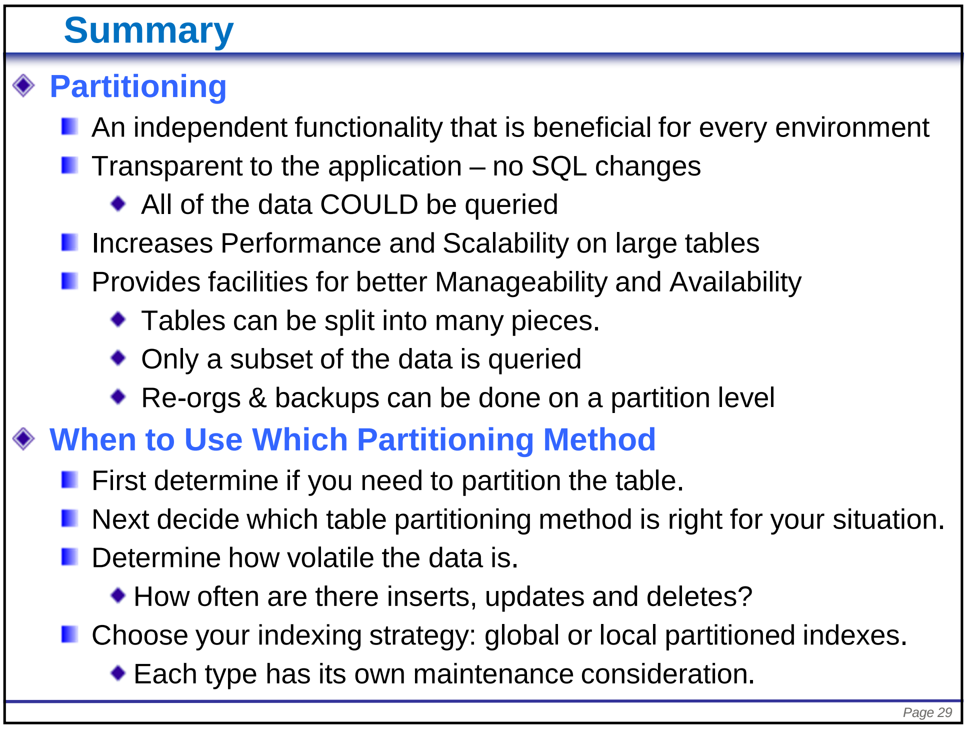
New Partitioning Extensions In Oracle 11g/12c



PARTITIONING EXTENSION	DESCRIPTION	SAMPLE BUSINESS CASE
Interval Partitioning • Interval • Interval-Range • Interval-List • Interval-Hash	Extension to Range Partition. Defined by an interval, providing equi-width ranges. With the exception of the first partition all partitions are automatically created on-demand when matching data arrives.	Orders table partitioned by order_date with a predefined daily interval, starting with '01-Jan-2013'
Reference Partitioning	Partitioning for a child table is inherited from the parent table through a primary key – foreign key relationship. The partitioning keys are not stored in actual columns in the child table.	(Parent) Orders table range partitioned by order_date and inherits the partitioning technique to (child) order lines table. Column order_date is only present in the parent orders table
Virtual column based Partitioning	Defined by any partition techniques where the partitioning key is based on a virtual column. Virtual columns are not stored on disk and only exist as metadata.	Orders table has a virtual column that derives the sales region based on the first three digits of the customer account number. The orders table is then list partitioned by sales region.

Partition for Tiered Storage





Summary

- **Partitioning**
  - An independent functionality that is beneficial for every environment
  - Transparent to the application – no SQL changes
    - All of the data COULD be queried
  - Increases Performance and Scalability on large tables
  - Provides facilities for better Manageability and Availability
    - Tables can be split into many pieces.
    - Only a subset of the data is queried
    - Re-orgs & backups can be done on a partition level
- **When to Use Which Partitioning Method**
  - First determine if you need to partition the table.
  - Next decide which table partitioning method is right for your situation.
  - Determine how volatile the data is.
    - How often are there inserts, updates and deletes?
  - Choose your indexing strategy: global or local partitioned indexes.
    - Each type has its own maintenance consideration.

Page 29