Implement Partitioning

Henry Zhou
Oracle Support Services

Agenda

1. Introduction to Partitioning
2. Implementing Partitioned Tables
3. Implementing Partitioned Indexes
4. Maintenance of Partitioned Tables and Indexes

1. Introduction to Partitioning

Objectives

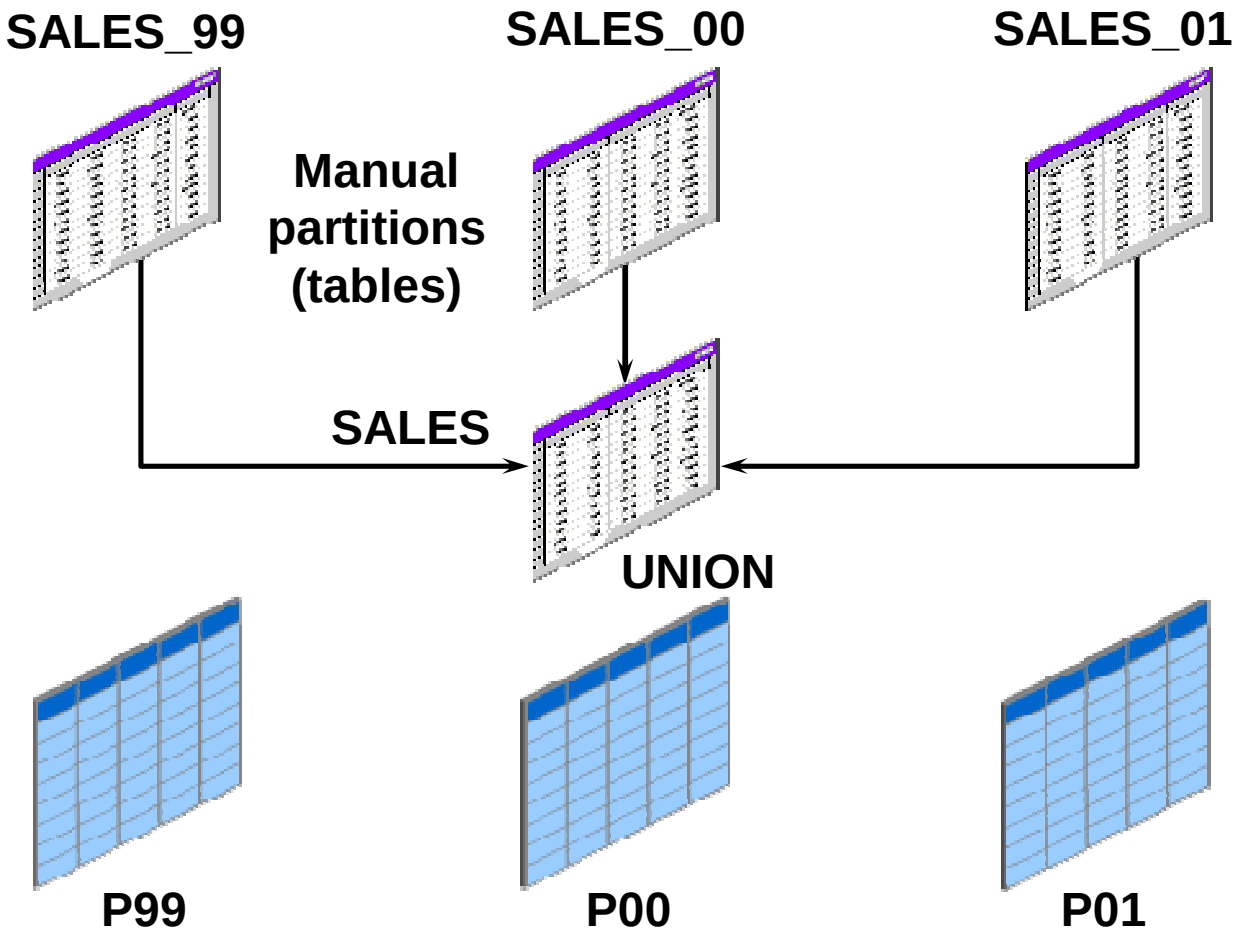
After completing this lesson, you should be able to do the following:

- **Describe the partitioning architecture, uses, and advantages**
- **Describe the partition types supported by Oracle RDBMS**

VLDB Manageability and Performance Constraints

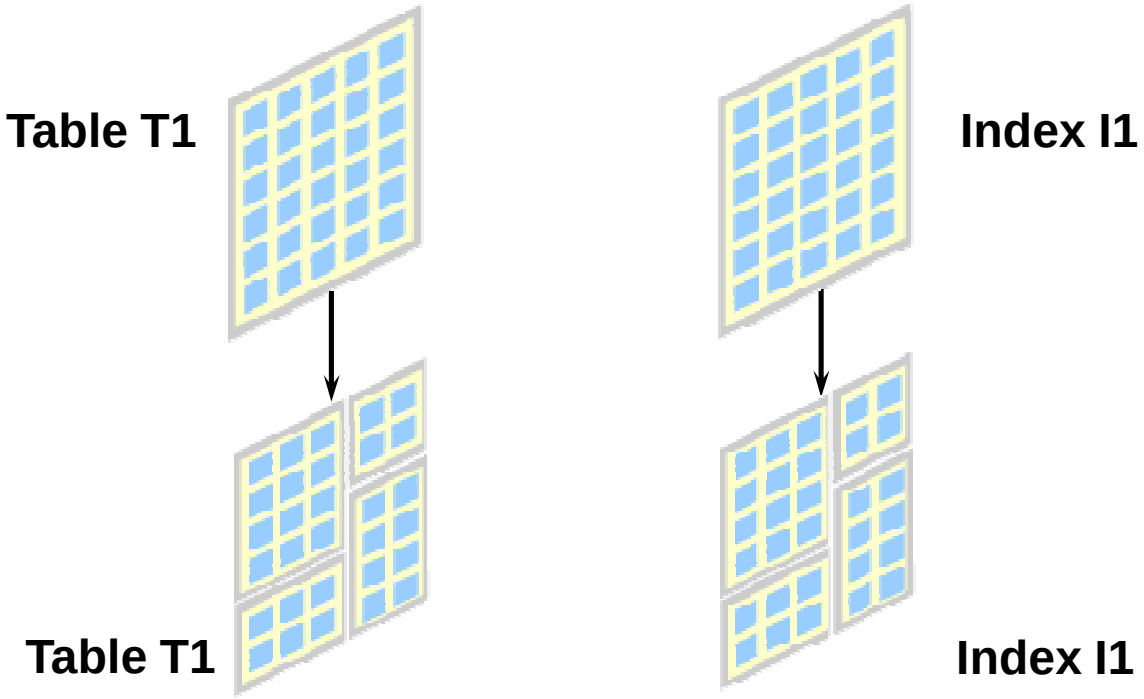
- **Table availability:**
  - **Large tables are more vulnerable to disk failure.**
  - **It is too costly to have a large table inaccessible for hours due to recovery.**
- **Large table manageability:**
  - **They take too long to be loaded.**
  - **Indexes take too long to be built.**
  - **Partial deletes take hours, even days.**
- **Performance considerations:**
  - **Large table and large index scans are costly.**
  - **Scanning a subset improves performance.**

Manual Partitions



Partitioned Tables and Indexes

Large tables and indexes can be partitioned into smaller, more manageable pieces.



Benefits of Partitioning: Table Availability

- Partitions can be independently managed.
- Backup and restore operations can be done on individual partitions.
- Partitions that are unavailable do not affect queries or DML operations on other partitions that use the same table or index.

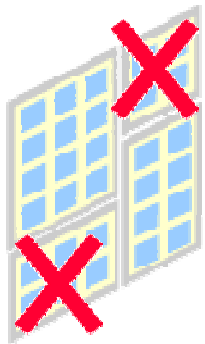
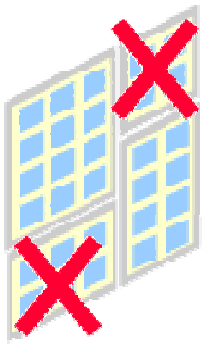


Table T1



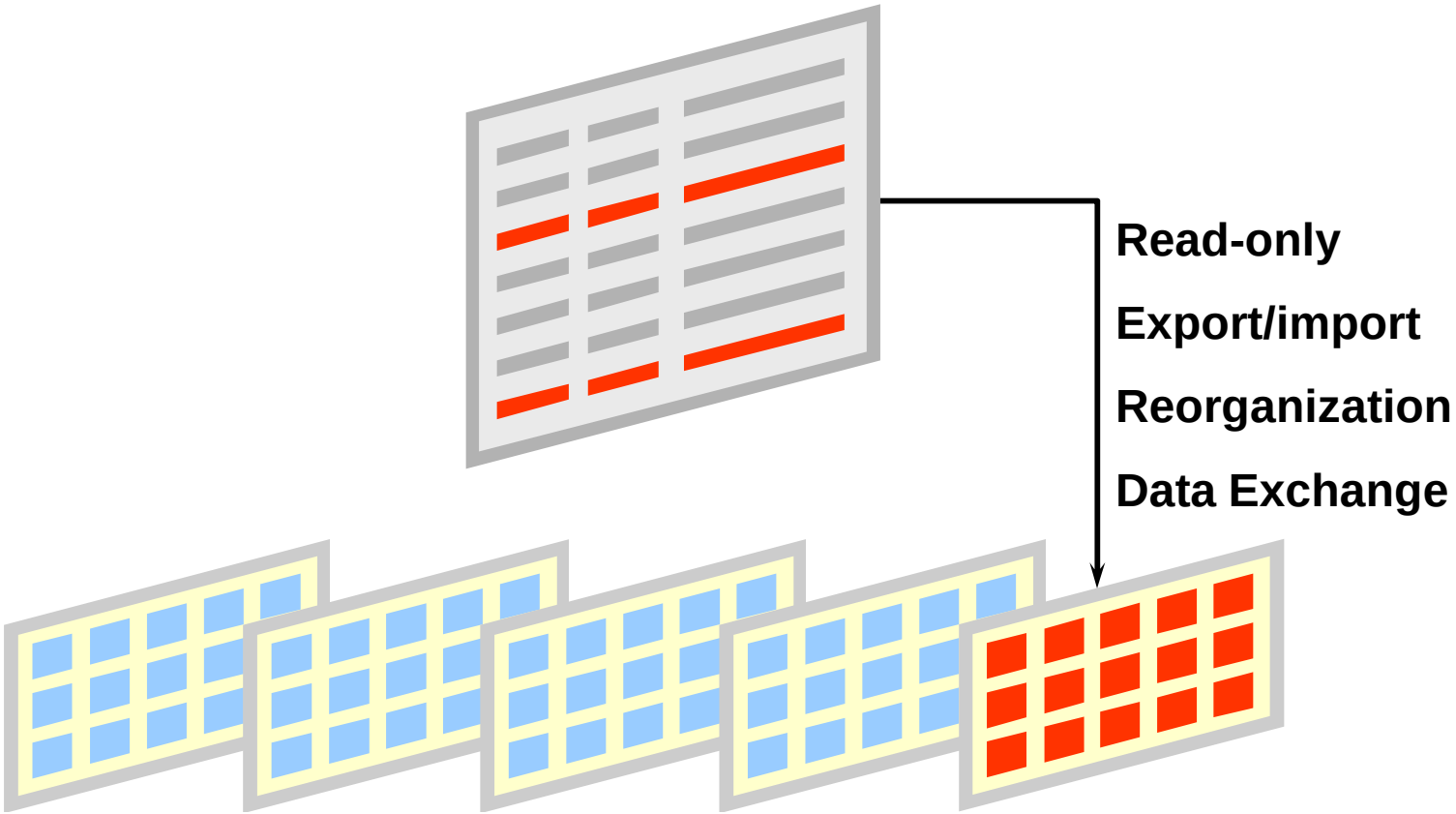
Index I1

Benefits of Partitioning: Large Table Manageability

Oracle provides a variety of methods and commands to manage partitions:

- **A partition can be moved from one tablespace to another.**
- **A partition can be divided at a user-defined value.**
- **Partitioning can isolate subsets of rows that must be treated individually.**
- **A partition can be dropped, added, or truncated.**
- **`SELECT`, `UPDATE`, `INSERT`, and `DELETE` operations can be applied on a partition level instead of a table level.**

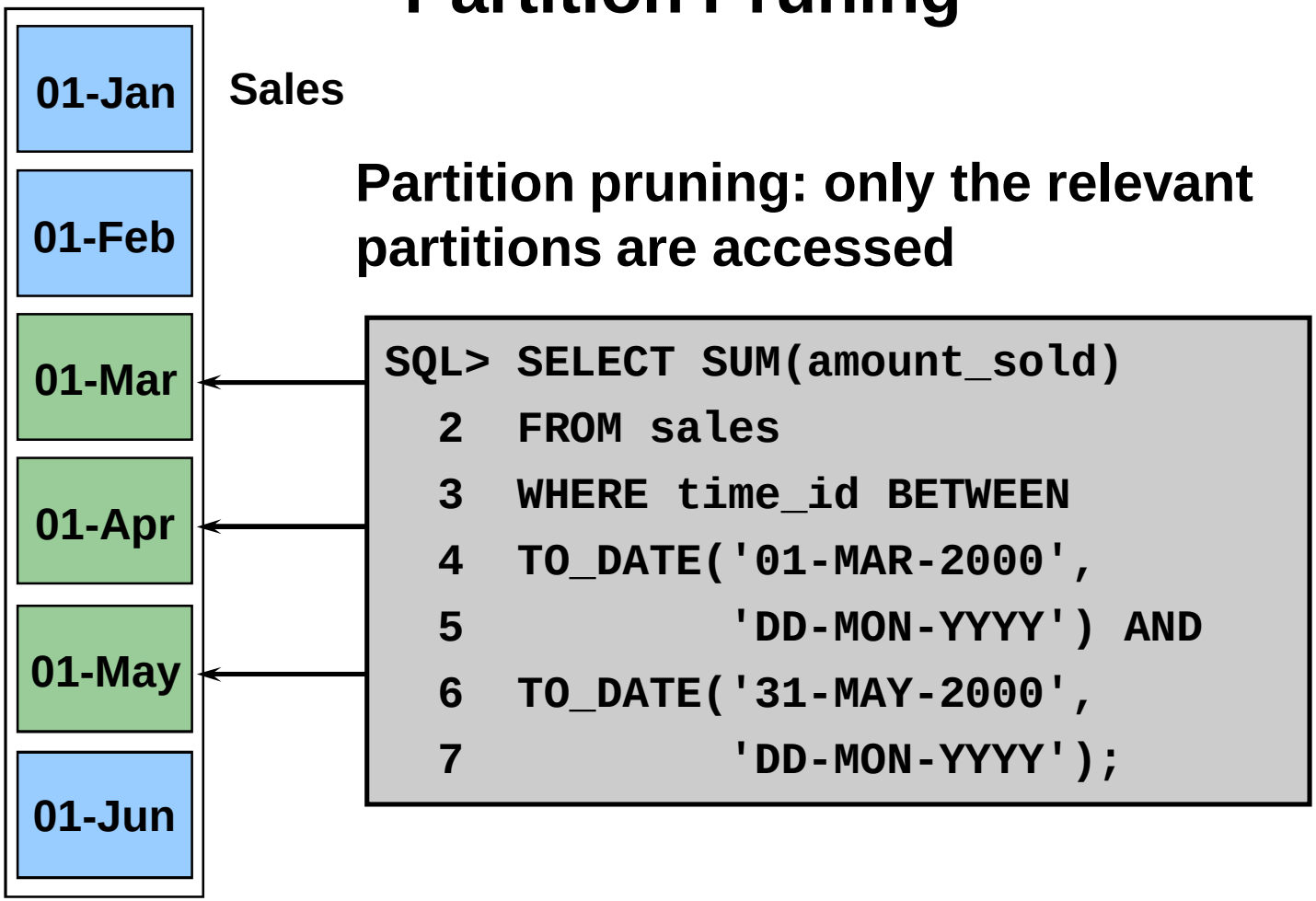
Manageability:
Clearly Defined Record Subsets



Benefits of Partitioning: Performance Considerations

- The optimizer eliminates (prunes) partitions that do not need to be scanned.
- Partitions can be scanned, updated, inserted, or deleted in parallel.
- Join operations can be optimized to join “by the partition”.
- Partitions can be load-balanced across physical devices.
- Large tables within Real Application Clusters environments can be partitioned.

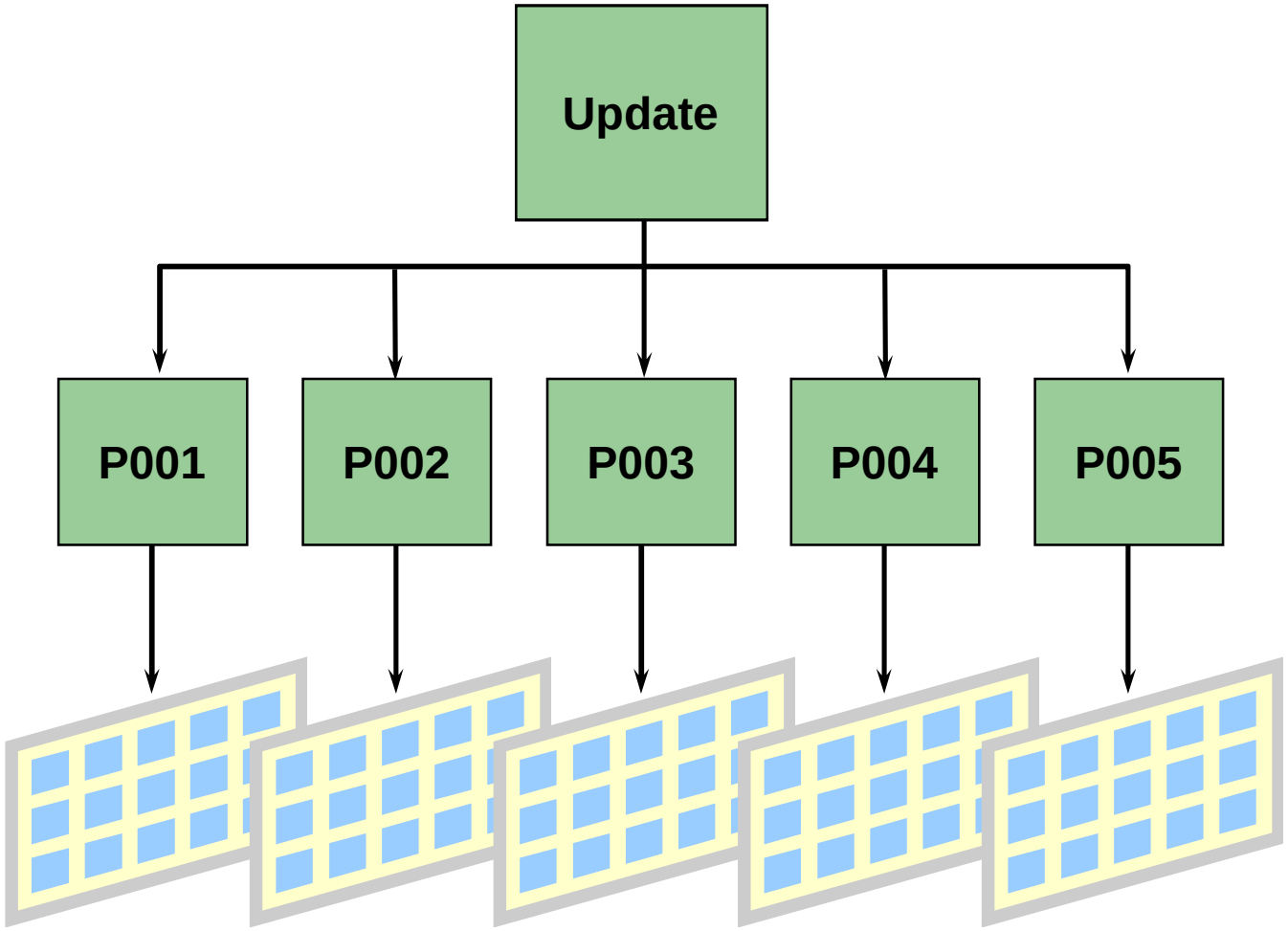
Performance Consideration: Partition Pruning



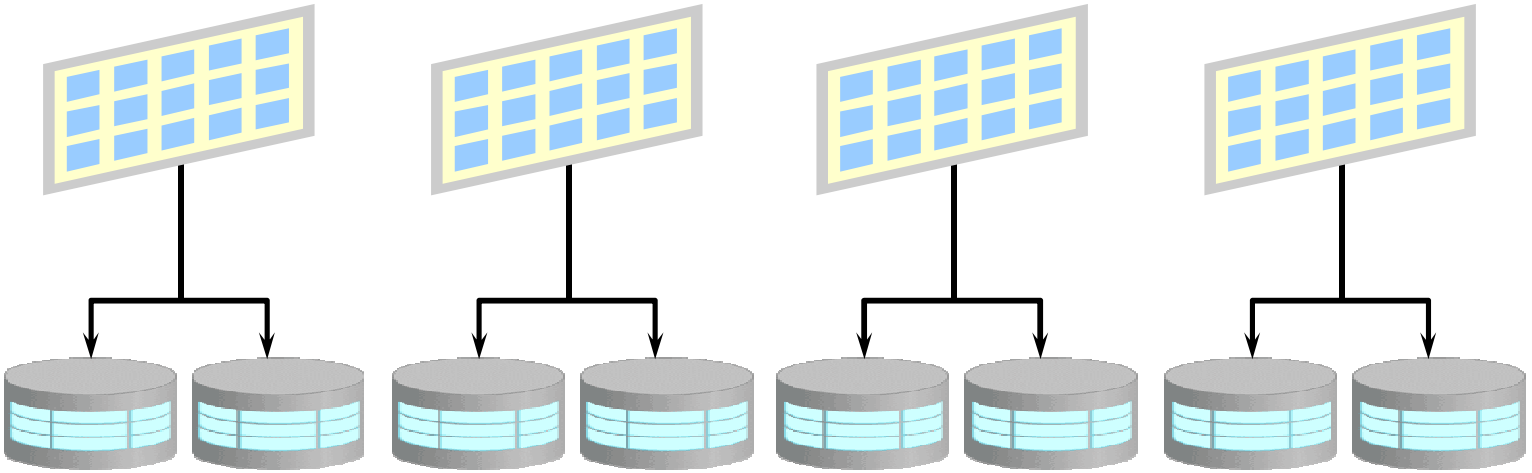
Partition pruning: only the relevant partitions are accessed

```
SQL> SELECT SUM(amount_sold)
  2  FROM sales
  3  WHERE time_id BETWEEN
  4  TO_DATE('01-MAR-2000',
  5          'DD-MON-YYYY') AND
  6  TO_DATE('31-MAY-2000',
  7          'DD-MON-YYYY');
```

Performance Consideration: Parallel DML



Performance Consideration: Device Load Balancing



Performance Improvement: Real Application Clusters

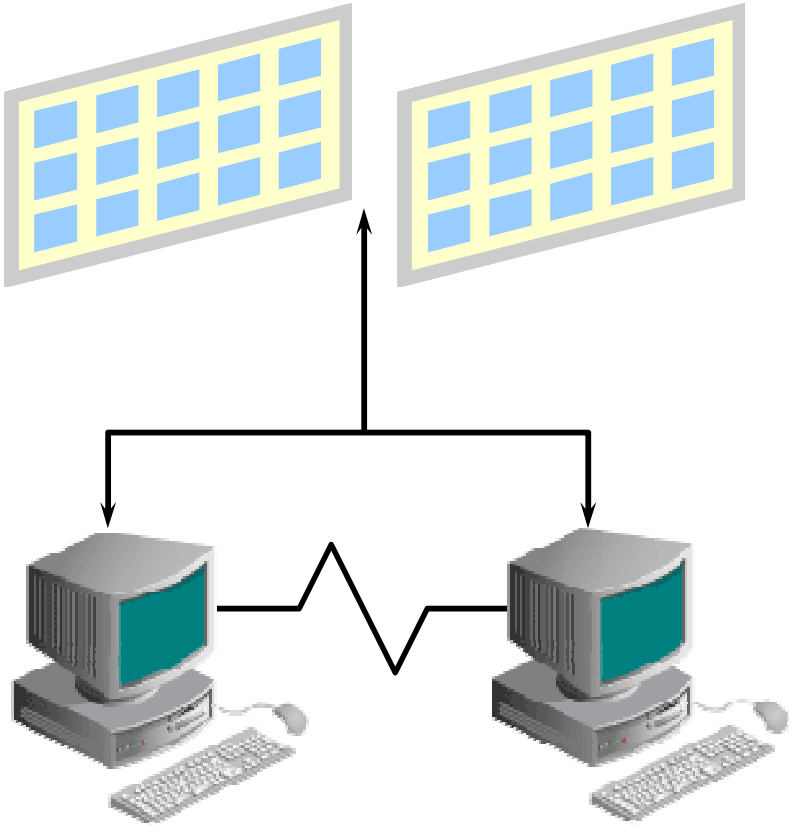
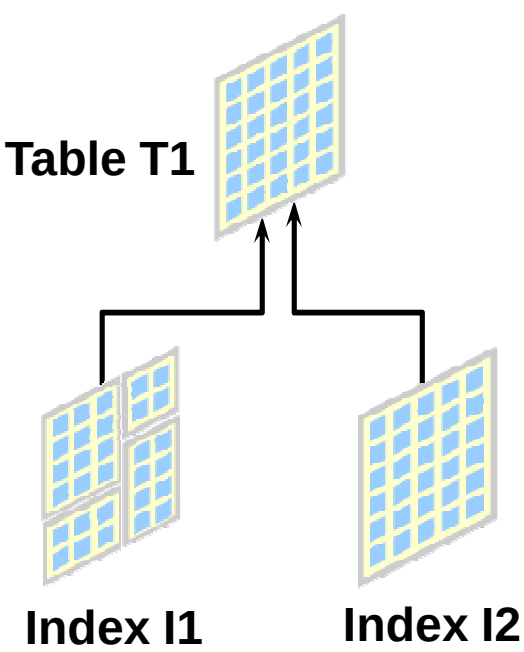
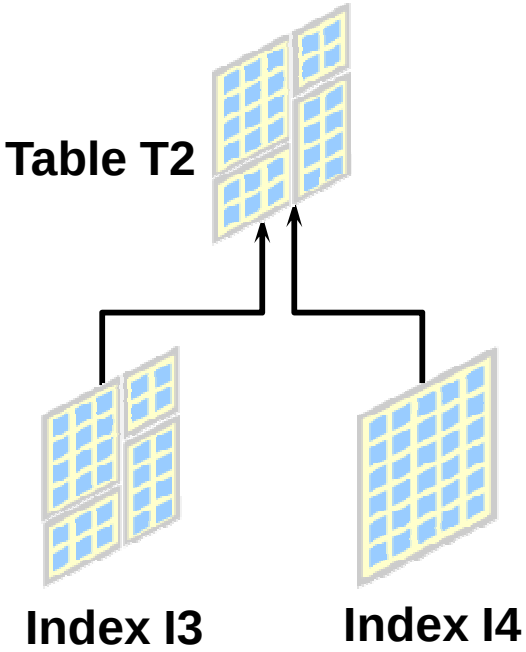


Table Versus Index Partitioning

A nonpartitioned table can have partitioned or nonpartitioned indexes.

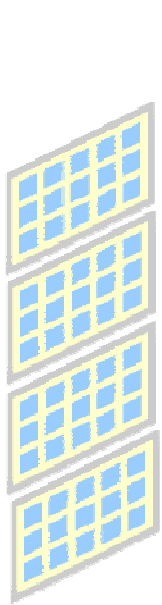


A partitioned table can have partitioned or nonpartitioned indexes.

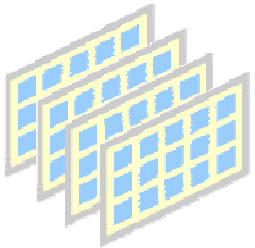


Partitioning Methods

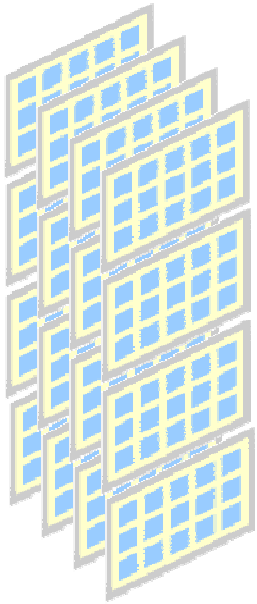
The following partitioning methods are available:



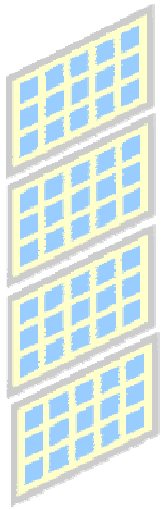
Range partitioning



Hash partitioning



Composite partitioning



List partitioning

Partitioned Indexes

- Indexes can be either partitioned or nonpartitioned.
- Choice of indexing strategy allows greater flexibility to suit database and application requirements.
- Indexes can be partitioned with the exception of cluster indexes.
- The same rules apply for indexes as for tables.
  - Type of access to data through the applications
  - Performance in accessing data
  - Availability in case of disk failure
  - Are parallel operations possible?

Verifying Partition Use

- **Examining execution plans will confirm partition pruning.**

Proof of Pruning

Proof of partition elimination or pruning may be obtained:

- **By using `tkprof`**
- **Through the explain plan**

SQL*Loader and Partitioned Objects

- You can load a partitioned table through the conventional path.
- You can sequentially load a partitioned table through the direct path.
- You can parallel load a single table partition through the direct path.
- Parallel direct path load in a single partition, neither local or global indexes can be maintained.

Summary

In this lesson, you should have learned how to:

- **Describe the partitioning architecture, uses, and advantages**
- **Describe the partition types supported by Oracle RDBMS**

2. Implementing Partitioned Tables

Objectives

After completing this lesson, you should be able to do the following:

- **Describe the partitioning types**
- **List all of the options for creating a partitioned table**
- **Create partitioned tables**
- **Use the data dictionary to verify the partitioned table structure**

The CREATE TABLE Statement with Partitioning

An example:

```
SQL> CREATE TABLE simple
  2   ( idx NUMBER, txt VARCHAR2(20) )
  3   PARTITION BY RANGE ( idx )
  4   (  PARTITION VALUES LESS THAN ( 0 )
  5        TABLESPACE data01
  6   ,  PARTITION VALUES LESS THAN ( MAXVALUE )
  7   ) ;
```

Logical and Physical Attributes

Logical attributes:

- **Normal table structure (columns, constraints)**
- **Partition type**
- **Keys and values**
- **Row movement**

Physical attributes:

- **Tablespace**
- **Extent sizes, block attributes**

Partitioning Type

The Partitioning type is declared in the `PARTITION` clause.

```
SQL> CREATE TABLE ( … column … )
  2  PARTITION BY RANGE ( column_list )
  3  ( PARTITION specifications ) ;
```

```
  2  PARTITION BY HASH ( column_list )
```

```
  2  PARTITION BY LIST ( column )
```

```
  2  PARTITION BY RANGE ( column_list )
   SUBPARTITION BY HASH/LIST ( column_list2 )
```

Specifying Partition Attributes

Each Partition is specified in a partition value clause.

```
   …
    PARTITION simple_p1 VALUES ( 'HIGH', 'MED' )
        TABLESPACE data01 PCTFREE 5
  , PARTITION simple_p2 VALUES ( 'LOW' )
        TABLESPACE data02 STORAGE ( INITIAL 1M )
   …
```

You can specify up to a total of 1024K-1 partitions and subpartitions

Partition Key Value

- **The partition key value must be a literal.**
- **Constant expressions are not allowed, with the exception of `TO_DATE` conversion.**
- **The partition key can consist of up to 16 columns**

Range Partitioning

Specify the columns to be partitioned, and the break values for each partition.

- **Each partition must be defined.**
- **The `MAXVALUE` value includes `NULL` values.**
- **How does oracle store null**

Range Partitioning Example

```
SQL> CREATE TABLE simple
  2   ( idx NUMBER, txt VARCHAR2(20) )
  3   PCTFREE 20 TABLESPACE data04
  4   PARTITION BY RANGE ( idx )
  5   (  PARTITION VALUES LESS THAN ( 0 )
  6        TABLESPACE data01
  7   ,  PARTITION VALUES LESS THAN ( MAXVALUE )
  8        PCTFREE 5 ) ;
```

Multicolumn Partitioning

You can specify multiple columns for a composite partitioning key.

- **The order is significant.**
- **The second column will be examined only after the first column values are equal to the limit specification.**

Multicolumn Example

If this is the partition definition,

```
SQL> CREATE TABLE multicol
  2    ( unit  NUMBER(1), subunit CHAR(1) )
  3  PARTITION BY RANGE ( unit, subunit )
  4  (  PARTITION P_2b VALUES LESS THAN (2,'B')
  5  ,  PARTITION P_2c VALUES LESS THAN (2,'C')
  6  ,  PARTITION P_3b VALUES LESS THAN (3,'B')
  7  ,  PARTITION P_4x VALUES LESS THAN (4,'X') );
```

which partition do the rows then go into?

#	Values	#	Values	#	Values
01	1,'A'	05	1,'Z'	09	1,NULL
02	2,'A'	06	2,'B'	10	2,'C'
03	2,'D'	07	2,NULL	11	3,'Z'
04	4,'A'	08	4,'Z'	12	4,NULL

List Partitioning

Specify the column to partition on, and list the values for each partition.

- **Each partition and each value must be defined.**
- **NULL can be specified.**

```
SQL> CREATE TABLE simple
  2    ( idx NUMBER, txt VARCHAR2(20) )
  3  PARTITION BY LIST ( txt )
  4  ( PARTITION s_top VALUES ( 'HIGH', 'MED' )
  5       TABLESPACE data01
  6  , PARTITION s_bot VALUES ( 'LOW', NULL )
  7       TABLESPACE data02
  8  ) ;
```

Hash Partitioning, Named Partitions

Specify the columns to be partitioned, and the number of partitions:

- **Partition may be defined, or just quantified**
- **NULL is placed in the first partition**
- **Number should be power of two**

```
SQL> CREATE TABLE simple
  2   (idx NUMBER, txt VARCHAR2(20) PRIMARY KEY)
  3  ORGANIZATION INDEX
  4  PARTITION BY HASH ( txt )
  5  ( PARTITION s_h1 tablespace data01
  6  , PARTITION s_h2 tablespace data03
  7  ) ;
```

Hash Partitioning: Quantity of Partitions

```
SQL> CREATE TABLE simple
  2    ( idx NUMBER, txt VARCHAR2(20) )
  3  PARTITION BY HASH ( idx )
  4    PARTITIONS 4
  5    STORE IN ( data01, data02 ) ;
```

Composite Partitioning

Composite Partitioning is a partitioning of the partitions.

Hash/list subpartitioning of a Range Partitioned table:

```
SQL> CREATE TABLE simple
  2   ( idx NUMBER, txt VARCHAR2(20) )
  3  PARTITION BY RANGE ( idx )
  4   SUBPARTITION BY HASH ( txt )
  5    SUBPARTITIONS 4 STORE IN (data01, data02)
  6  ( PARTITION ns_lo VALUES LESS THAN ( 0 )
  7  , PARTITION ns_hi VALUES LESS THAN ( 1E99 )
  8  , PARTITION ns_mx
  9               VALUES LESS THAN ( MAXVALUE )
 10     SUBPARTITIONS 2 STORE IN ( data03 ) ) ;
```

Composite Partitioning: Another Example

```
SQL> CREATE TABLE simple
  2   ( idx NUMBER, txt VARCHAR2(20) )
  3  PARTITION BY RANGE ( idx )
  4   SUBPARTITION BY HASH ( txt )
  5  ( PARTITION ns_lo VALUES LESS THAN ( 0 )
  6    ( SUBPARTITION ns_lo1 TABLESPACE data01
  7    , SUBPARTITION ns_lo2 TABLESPACE data02
  8    , SUBPARTITION ns_lo3 TABLESPACE data01
  9    , SUBPARTITION ns_lo4 TABLESPACE data02 )
 10  , PARTITION ns_hi VALUES LESS THAN ( 1E99 )
 11    ( SUBPARTITION ns_hi1 TABLESPACE data01
 12    , SUBPARTITION ns_hi2 TABLESPACE data02 )
 13  , PARTITION ns_mx
 14              VALUES LESS THAN ( MAXVALUE )
 15     SUBPARTITIONS 2 STORE IN (data03)
 16   ) ;
```

LOB Partitioning

- LOB segments are equipartitioned with the table partition.
- Storage attributes are specified for each LOB in each partition.

```
  …
PARTITION s_10 VALUES LESS THAN ( 10 )
   TABLESPACE data01
   LOB ( txt ) STORE AS st_10
       ( DISABLE STORAGE IN ROW
         TABLESPACE data03 )
  …
```

LOB Segment Example

```
SQL> CREATE TABLE simple
  2    ( idx NUMBER, txt CLOB )
  3  LOB ( txt ) STORE AS s_lob
  4      ( TABLESPACE data04 )
  5  PARTITION BY RANGE ( idx )
  6  ( PARTITION s_10 VALUES LESS THAN ( 10 )
  7      TABLESPACE data01
  8      LOB ( txt ) STORE AS st_10
  9          ( DISABLE STORAGE IN ROW
 10            TABLESPACE data03 )
 11  , PARTITION s_20 VALUES LESS THAN ( 20 )
 12      TABLESPACE data02
 13   ) ;
```

Updateable Partition Keys

Because performing `UPDATE` on a row alters the value of the columns that define which partition the row belongs to, the following can happen:

- **The update results in the row still being mapped to the same partition.**
- **The update makes the row map to another partition, and therefore is disallowed.**
- **The update makes the row map to another partition, and therefore the row is moved to the new partition.**

- **Demo it.**

Partition Extended Table Names

Specify the partition in a table to limit an operation:

```
SQL> SELECT idx
  2   FROM simple PARTITION ( s_neg ) ;
```

```
SQL> DELETE  FROM simple SUBPARTITION ( s_h2 ) ;
```

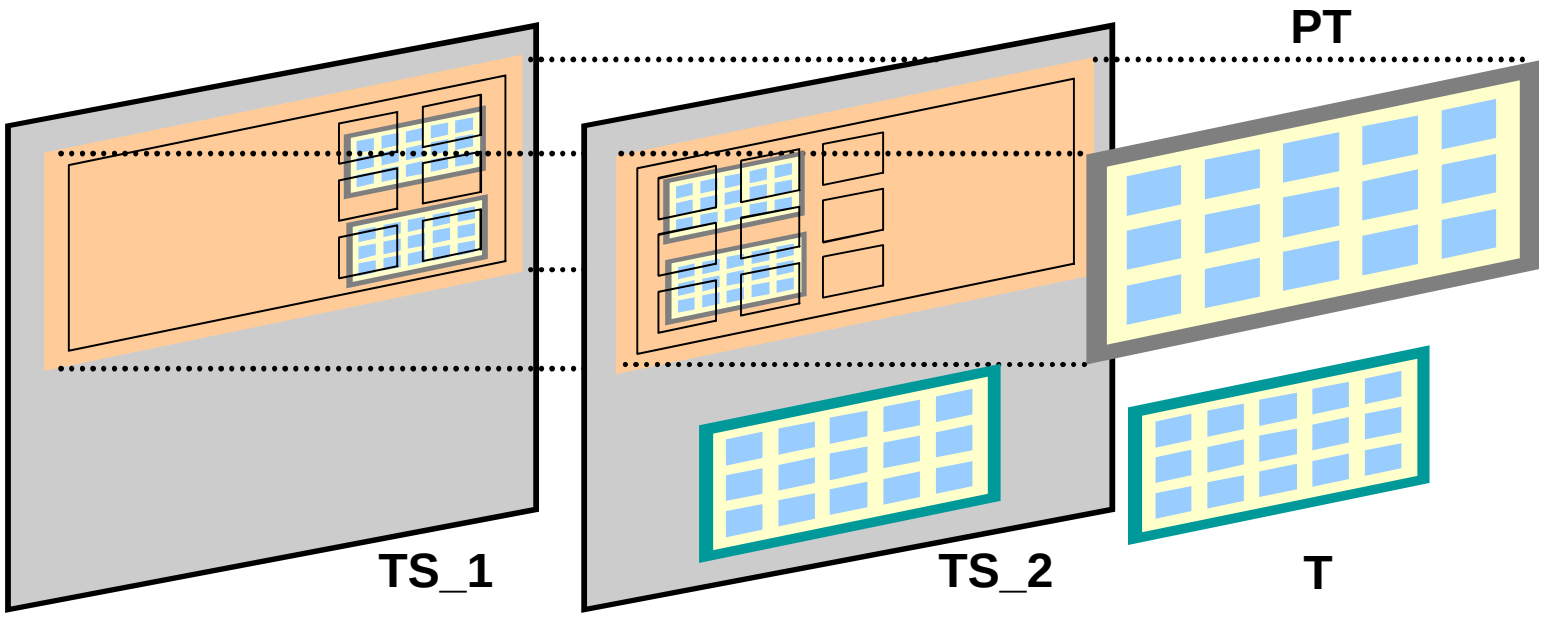
```
SQL> CREATE TABLE sim2
  2   AS SELECT * FROM simple PARTITION ( p3 ) ;
```

General Restrictions

- **Partitioned tables cannot contain `LONG` data.**
- **All partitions must reside in tablespaces with the same block size.**
- **You cannot partition on LOBs.**
- **Comparison of partition keys is done by binary values.**

Table, Partition, and Segment Relations

- A partitioned table is an object consisting of subobjects, the partitions.
- The table is “virtual,” and consists of physical partitions.



Data Dictionary Views
Tables

Name	Purpose	N
DBA_TABLES	Table structure, Partition Y/N	T
DBA_PART_TABLES	Partition type, default values	T
DBA_TAB_*PARTITIONS	Partitions detail	P
DBA_*PART_KEY_COLUMNS	Partition keys	P

* SUB variation

T = per Table
P = per Partition

Demo it.

Data Dictionary Views
Segments

Name	Columns to show
DBA_SEGMENTS	PARTITION_NAME, SEGMENT_TYPE
DBA_EXTENTS	PARTITION_NAME, SEGMENT_TYPE
DBA_OBJECTS	SUBOBJECT_NAME, OBJECT_TYPE

Summary

In this lesson, you should have learned how to:

- Create the four different partition types
- Specify storage attributes for partitions
- Apply partitioning to tables, IOTs, and tables with LOBs
- Examine the data dictionary to verify how the partitions are defined

3. Implementing Partitioned Indexes

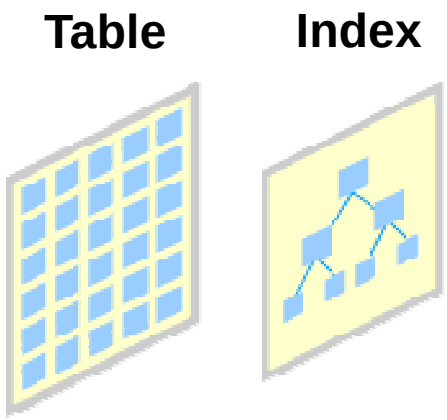
Objectives

After completing this lesson, you should be able to do the following:

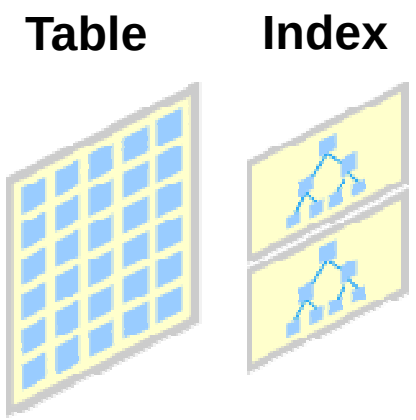
- **Describe the table and index partition relationships**
- **List all the options of partitioned indexes**
- **Create some partitioned indexes**
- **Use the data dictionary to verify the partitioned index structure**

Partitioned Indexes

- **Indexes can be partitioned like tables.**
- **Partitioned or nonpartitioned indexes can be used with partitioned or nonpartitioned tables.**



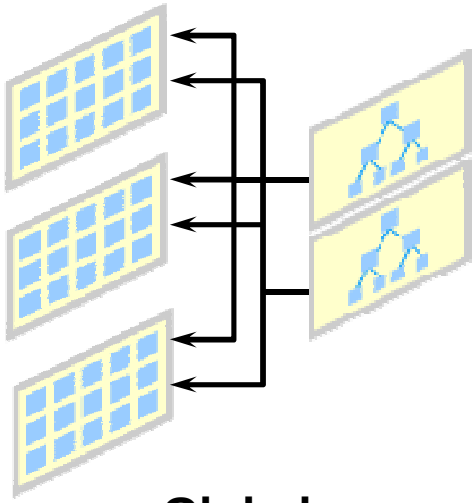
Normal



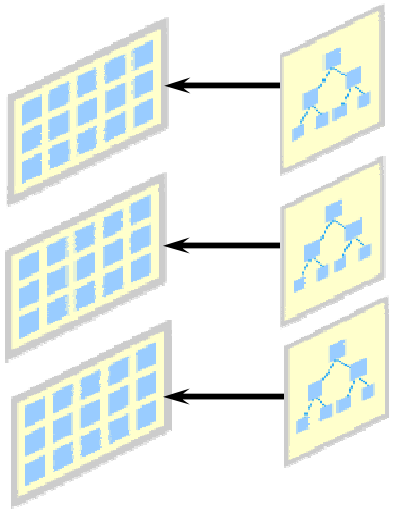
Partitioned

Partitioned Index Attributes: Global or Local

- **The partitions of a Global Partitioned Index are defined independently from the table that it indexes. A Local Partitioned Index corresponds in partitioning to the table.**



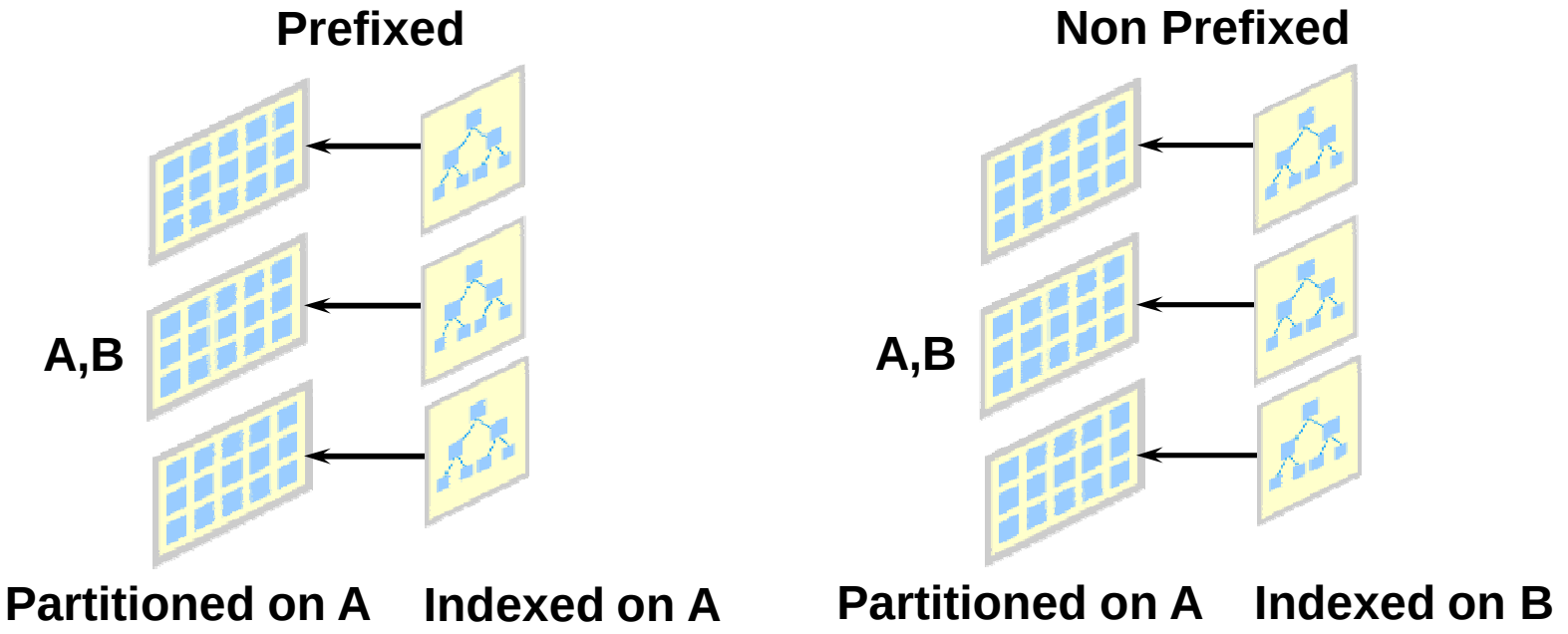
Global



Local

Partitioned Index Attributes: Prefixed or Nonprefixed

In a prefixed index, all leftmost columns of the index key are the same as all the columns in the partition key of the index.



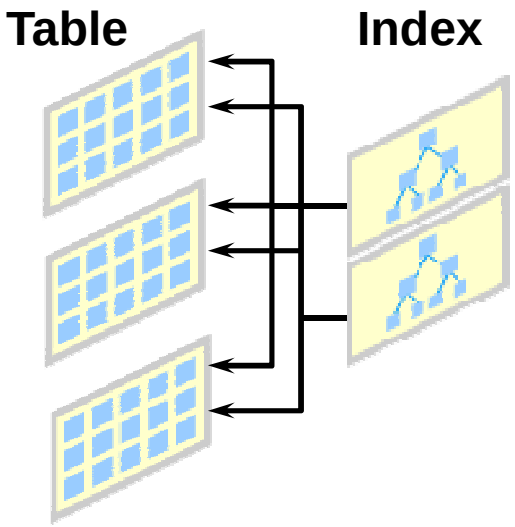
Index Partitioning Types

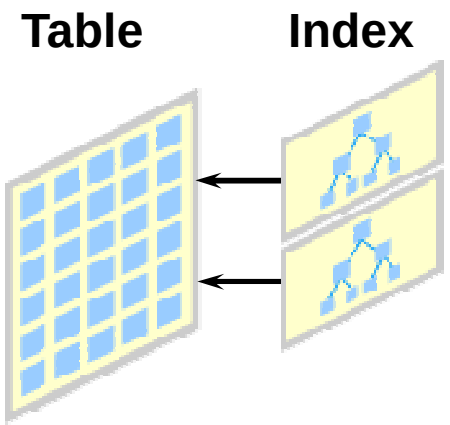
- Partitioned indexes can be:
  - Global or local
  - Prefixed or nonprefixed
- Allowed partitioning types are:
  - Global, not equipartitioned, and prefixed
  - Local, equipartitioned, and prefixed
  - Local, equipartitioned, and nonprefixed
- A normal nonpartitioned index is also a “partition type.”
- All index types can be partitioned.

Global Indexes

Global indexes:

- **Must be prefixed**
- **Only allow `RANGE` partitioning**
- **Must include `MAXVALUE` on all columns**





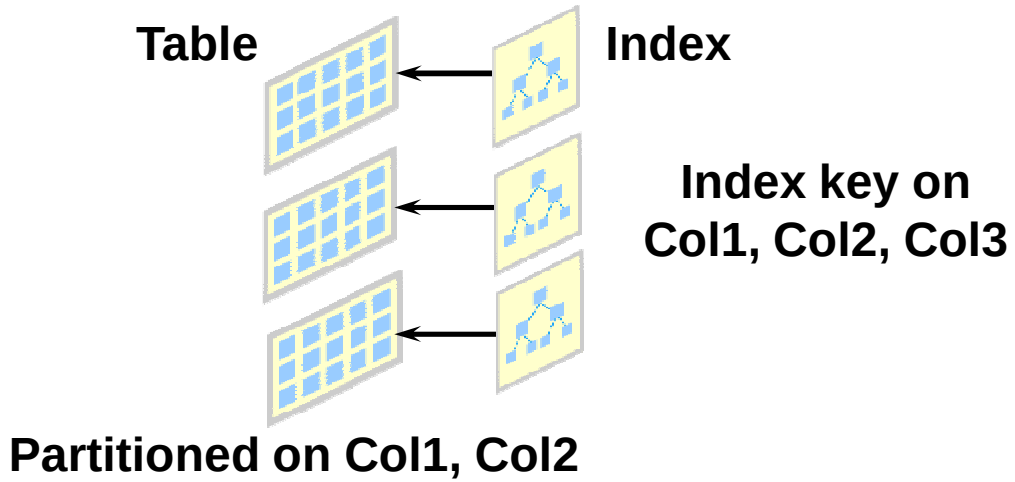
Global Index Example

```
SQL> CREATE INDEX idx ON emp ( first_name )
  2  GLOBAL PARTITION BY RANGE ( first_name )
  3  (  PARTITION x1 VALUES LESS THAN ( 'H' )
  4       TABLESPACE data01
  5  ,  PARTITION x2 VALUES LESS THAN
  6                                ( MAXVALUE )
  7  ) ;
```

Local Prefixed Index

Local prefixed indexes:

- **Only possible on partitioned tables**
- **Both the partition key of the index partitions and the leading columns of the index key are the same as the table partitioning key.**



Local Prefix Index Examples

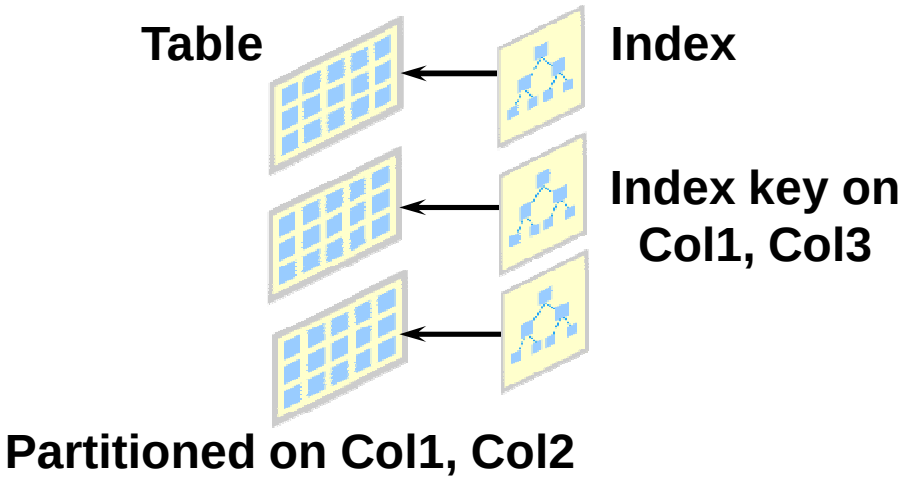
```
SQL> CREATE INDEX idx ON hr_emp( first_name )
  2    LOCAL ;
```

```
SQL> CREATE INDEX idx ON hr_emp( first_name )
  2   TABLESPACE indx04
  3   LOCAL
  4  ( PARTITION ex1 TABLESPACE indx01
  5  , PARTITION ex2 TABLESPACE indx02
  6  , PARTITION ex3
  7  ) ;
```

Local Nonprefixed Index

Local nonprefixed indexes:

- **Are possible only on partitioned tables**
- **Although the index partition key is the same as the table partition key, the index key is not the same.**



Local Nonprefix Index Example

```
SQL> CREATE INDEX idx ON hr_emp( last_name )
  2    LOCAL ;
```

Index Partitioning and Type Matrix

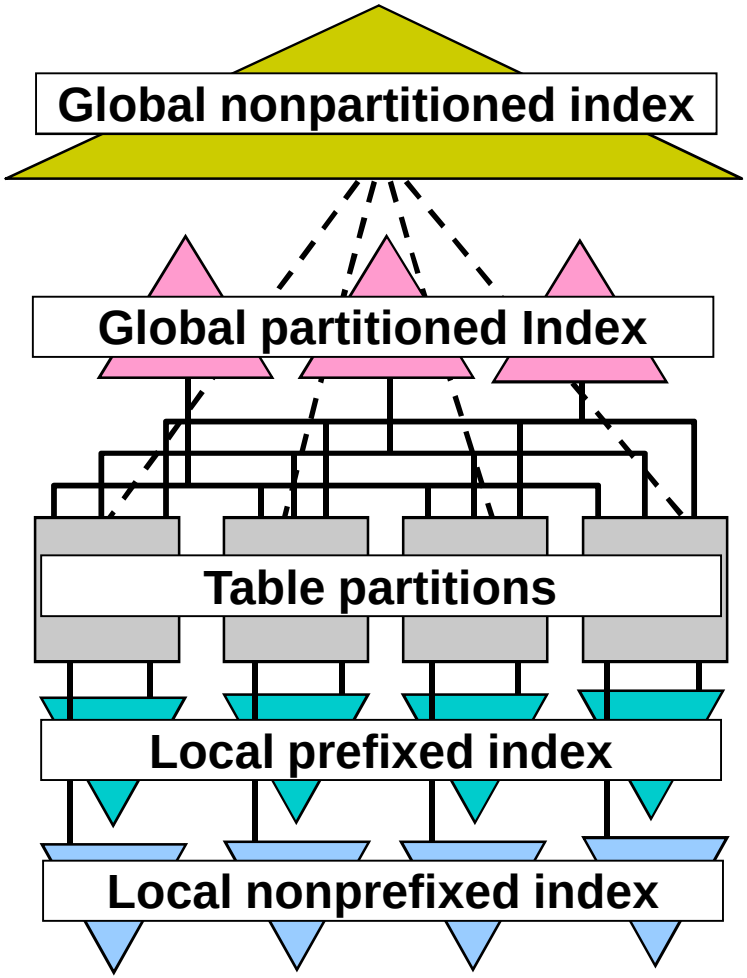
Index types	Global (Range)	Local (all)
B*Tree	Yes	Yes
Bitmap	No	Yes
Bitmap Join	No	Yes
Secondary IOT	No	Yes
Cluster*	No	No

Specifying Index with Table Creation

The partition structure of an index that is used for primary key constraint can be specified together with the partitioned table creation.

```
SQL> CREATE TABLE nonsimple
  ( idx number, txt varchar2(10),
    CONSTRAINT s_pk PRIMARY KEY ( idx ) )
  TABLESPACE data04 PARTITION BY HASH ( txt )
  ( PARTITION s1, PARTITION s2 )
ENABLE CONSTRAINT s_pk USING INDEX
 GLOBAL PARTITION BY RANGE ( idx )
 ( PARTITION spk1 VALUES LESS THAN ( 0 )
       TABLESPACE indx02 ,
  PARTITION spk2 VALUES LESS THAN (MAXVALUE)
       TABLESPACE indx03 ) ;
```

Graphic Comparison of Partitioned Index Types

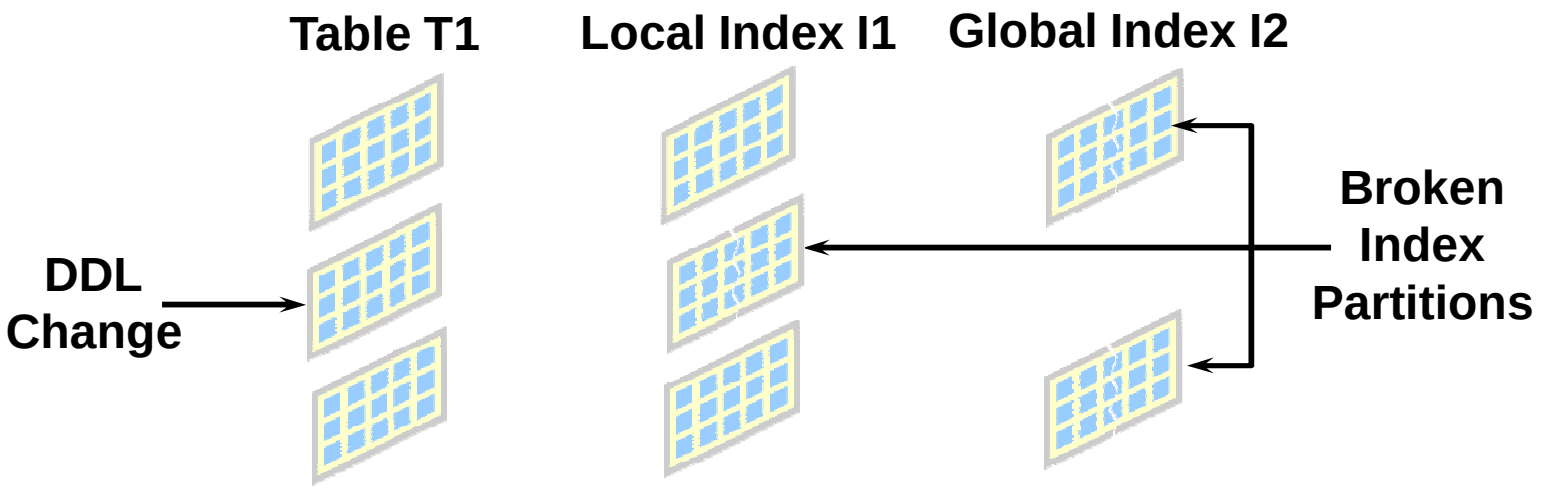


- Global nonpartitioned index
- Global partitioned prefixed index
- Local prefixed index
- Local nonprefixed index

Index Partition Status

A table partition can be altered:

- With DML - The index is maintained.
- With DDL - The index might become `UNUSABLE`.
  - Usually only one partition for local indexes
  - The whole index for global or nonpartitioned indexes



Index Partition UNUSABLE

- The index remains defined.
- If partitioned, other partitions remain fully usable.
- The index will block DML on the corresponding table.
- Queries can fail or bypass UNUSABLE index partitions depending on the session SKIP_UNUSABLE_INDEX setting.
  - TRUE, use another execution plan
  - FALSE, report ORA-1502

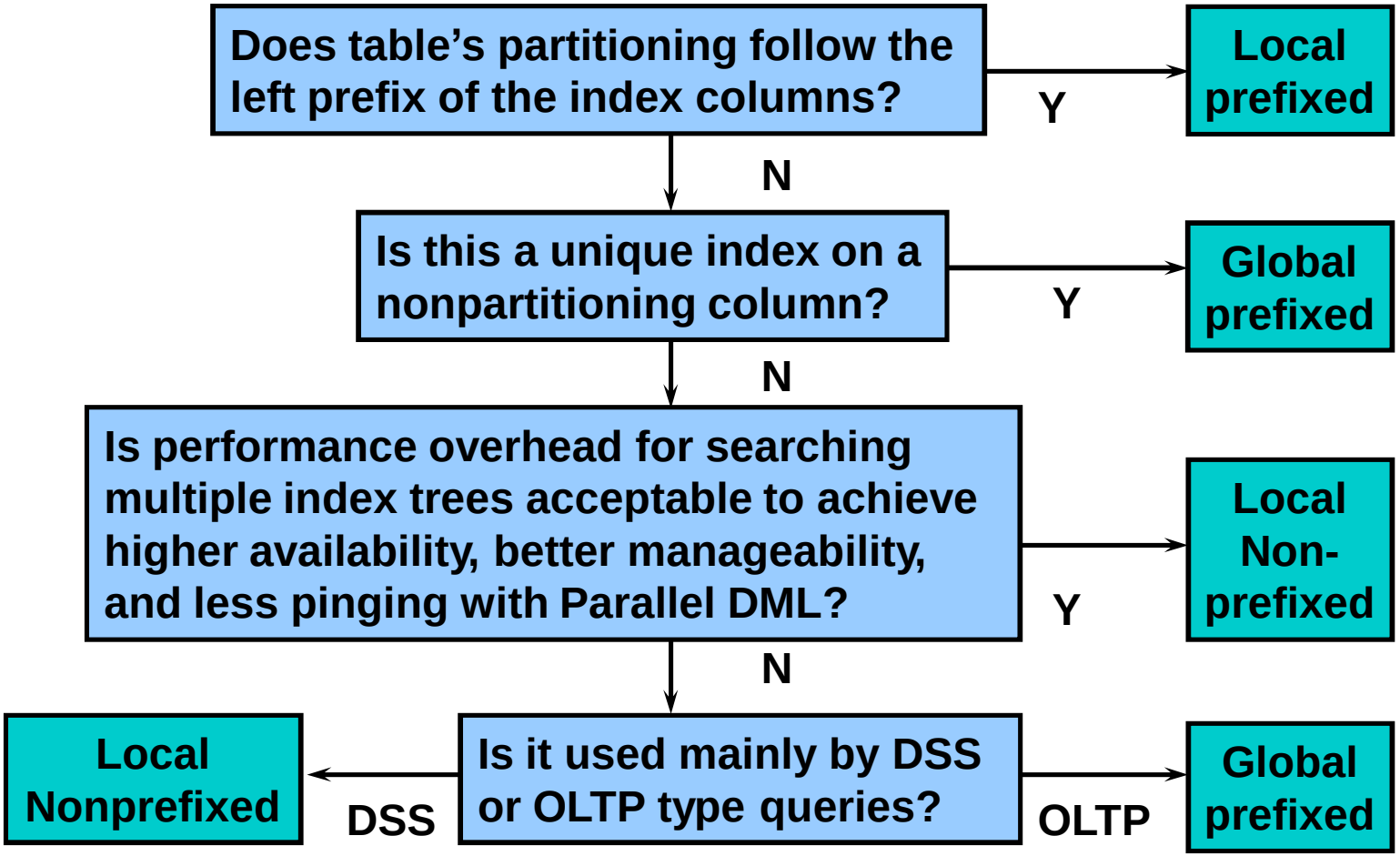
Data Dictionary Views Indexes

Name	Purpose	N
DBA_INDEXES	Index structure, Partition Y/N	I
DBA_PART_INDEXES	Partition type, default values	I
DBA_IND_*PARTITIONS	Partitions detail	P
DBA_*PART_KEY_COLUMNS	Partition keys	P
DBA_IND_COLUMNS	Index keys	I

* SUB variation

I = per index
P = per partition

Guidelines for Partitioning Indexes



Summary

In this lesson, you should have learned how to:

- **Describe the different index partitioning types**
- **Create partitioned indexes**

4. Maintenance of Partitioned Tables and Indexes

Objectives

After completing this lesson, you should be able to do the following:

- **List all of the alterable partitioned table and index attributes**
- **Describe the overhead associated with each maintenance command**

Maintenance Overview

With the `ALTER TABLE` or `ALTER INDEX` statements, you can modify:

- **Logical attributes of table - Column data types**
- **A single partition:**
  - **Allocate, truncate, rename**
  - **Exchange with a table**
- **Table or index partitions:**
  - **Add, drop, move, split, coalesce, merge**
  - **Row movement**

You cannot simply alter the partition type.

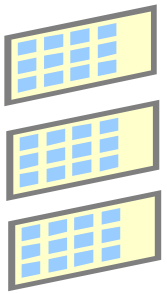
DBMS_REDFENITION

Table and Index Interaction During Partition Maintenance

- Altering a table partition will affect indexes on the table:
  - Local indexes are added, dropped, or set to `UNUSABLE`.
  - Global indexes are marked `UNUSABLE`.
  - Global partitioned indexes are marked `UNUSABLE`.
- Adding the `UPDATE GLOBAL INDEXES` clause will maintain global indexes.
- Altering an index partition does not affect other indexes or tables.

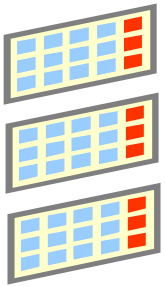
Modifying a Table or Indexing Logical Properties

- You can modify the name of a partitioned table or index, just like you can modify a nonpartitioned one.
- You can add, modify, or drop columns in a table, like a nonpartitioned one.
- There are restrictions on modifying the columns used for the partition key or if not all partitions are available.



```
SQL> RENAME name TO newname
```

```
SQL> ALTER TABLE …
  2   ADD ( column type … )
```



Modifying Partition Properties on the Table

- The row migration property can be enabled and disabled.

```
SQL> ALTER TABLE simple ENABLE ROW MOVEMENT ;
```

- The default storage attributes of the table/(new partitions) can be altered.

```
SQL> ALTER TABLE simple MODIFY
  2    DEFAULT ATTRIBUTES PCTFREE 50 ;
```

Using the ALTER TABLE or INDEX Commands

For RENAME, TRUNCATE, ADD, DROP, SPLIT, COALESCE, MERGE, and MOVE commands, use the following:

```
SQL> ALTER TABLE table_name
  2    operation PARTITION partition_name …

SQL> ALTER INDEX index_name
  2    operation PARTITION partition_name …
```

```
SQL> ALTER TABLE table PARTITION ( name ) …

ORA-14052: partition-extended table name
syntax is disallowed in this context
```

Renaming a Partition

```
SQL> ALTER TABLE tab_hash
  2    RENAME PARTITION SYS_P451 TO HASH_1 ;
```

Partition Storage Changes

- **TRUNCATE (DROP) . Update global indexes**

```
SQL> ALTER TABLE tab TRUNCATE PARTITION px ;
```

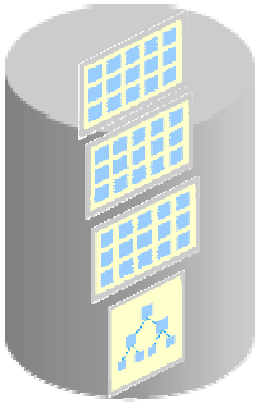
- **MODIFY: Partition storage**

```
SQL> ALTER TABLE tab MODIFY PARTITION px
  2   ALLOCATE EXTENT (SIZE 100M) ;
```

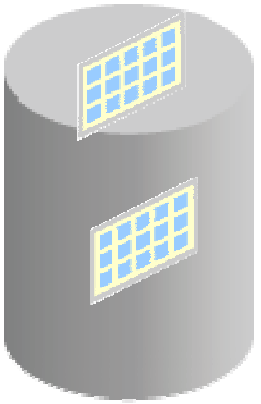
Moving a Partition

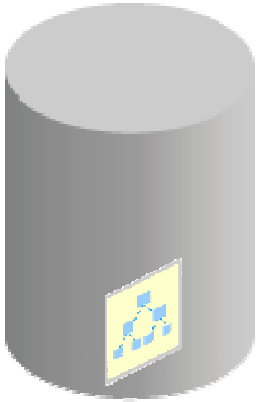
- Moving a partition places it in a new tablespace.
  - All storage attributes can be modified.
  - The partition is reorganized.
- All partition types can be moved: range, list, hash, and subpartitions
- Both table and index partitions can be moved:
  - Use `MOVE` for table partitions
  - Use `REBUILD` for index partitions

Moving a Partition: Example

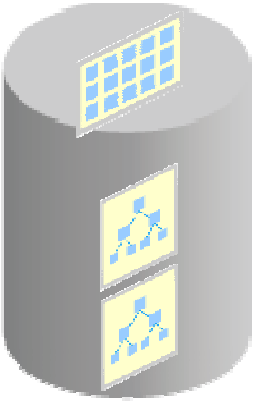


```
SQL> ALTER TABLE simple
  2   MOVE PARTITION p2
  3    TABLESPACE data03
  4    PCTFREE 95 ;
```





```
SQL> ALTER INDEX s_glo
  2   REBUILD PARTITION sg_1
  3    TABLESPACE data03 ;
```



Adding a Partition

- For Range Partition, a new partition is added *at the end*.
  - Specify a new high end value
  - Cannot add if `MAXVALUE` partition exists
  - Does not mark global indexes UNUSABLE
- For Hash Partition and Hash Subpartition, a added partition will receive rows redistributed from other partitions. Global/Local index partitions unusable.
- For List Partition, a added partition is added as specified.
- Cannot add a partition to a global index
  - Local indexes follow the table

When to Add a Partition

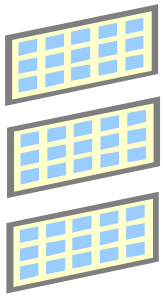
A partition is added when:

- Changes in data require it:
  - Rolling window (range partition)
  - New key values (list partition)
- The quantity of data increases

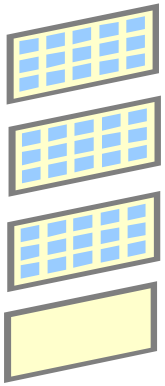
  Spread over more storage (hash partition)

Adding a Partition: Examples

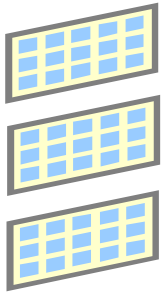
- **List-partitioned table:**



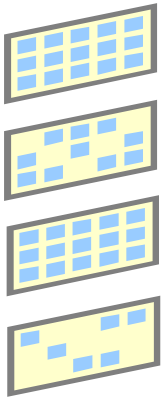
```
SQL> ALTER TABLE tab_list ADD
  2    PARTITION p3
  3    VALUES ( 'NEW' )
  4    TABLESPACE data03 ;
```



- **Hash-partitioned table:**

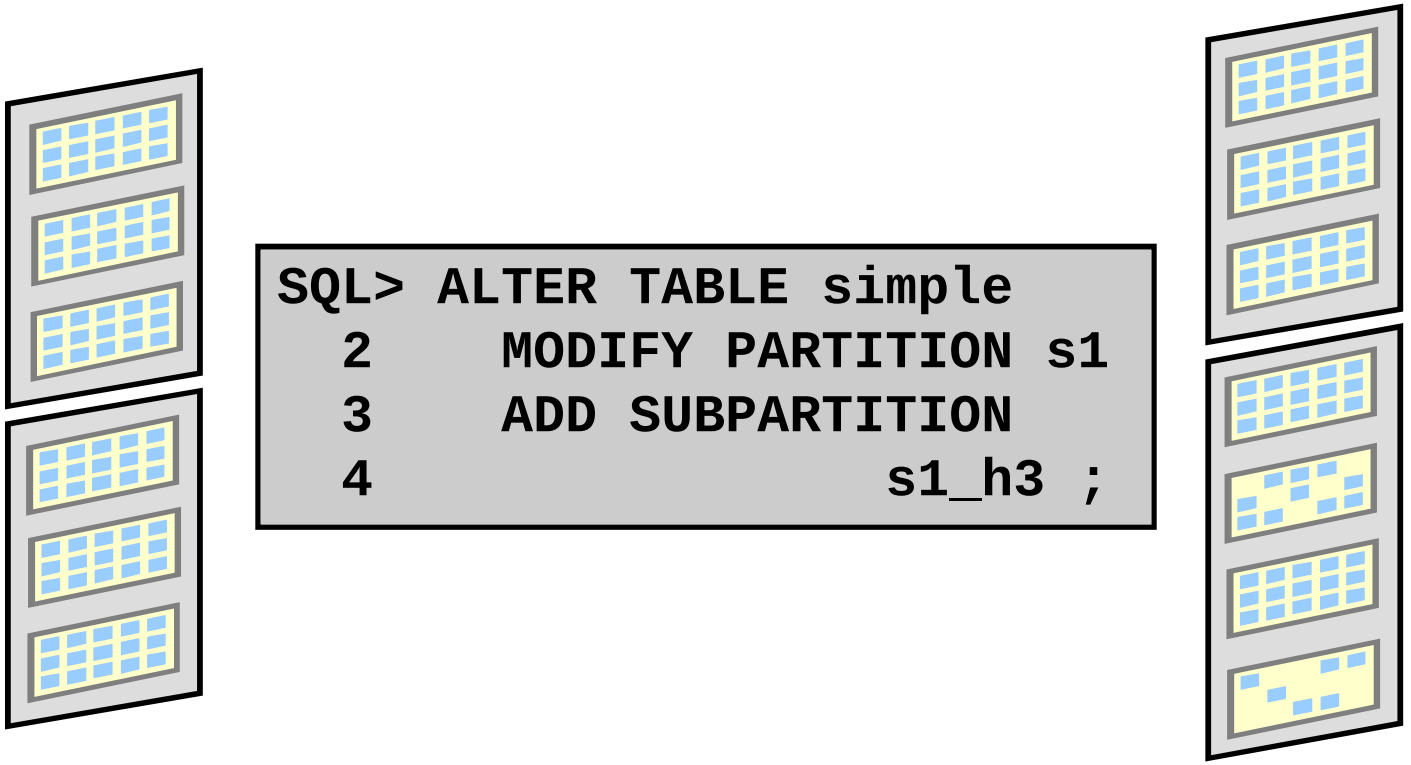


```
SQL> ALTER TABLE tab_hash ADD
  2    PARTITION p3
  3    TABLESPACE data03
  4  UPDATE GLOBAL INDEXES ;
```



Adding a Subpartition: Example

Composite partitioned table:



```
SQL> ALTER TABLE simple
  2    MODIFY PARTITION s1
  3    ADD SUBPARTITION
  4                    s1_h3 ;
```

Dropping a Partition

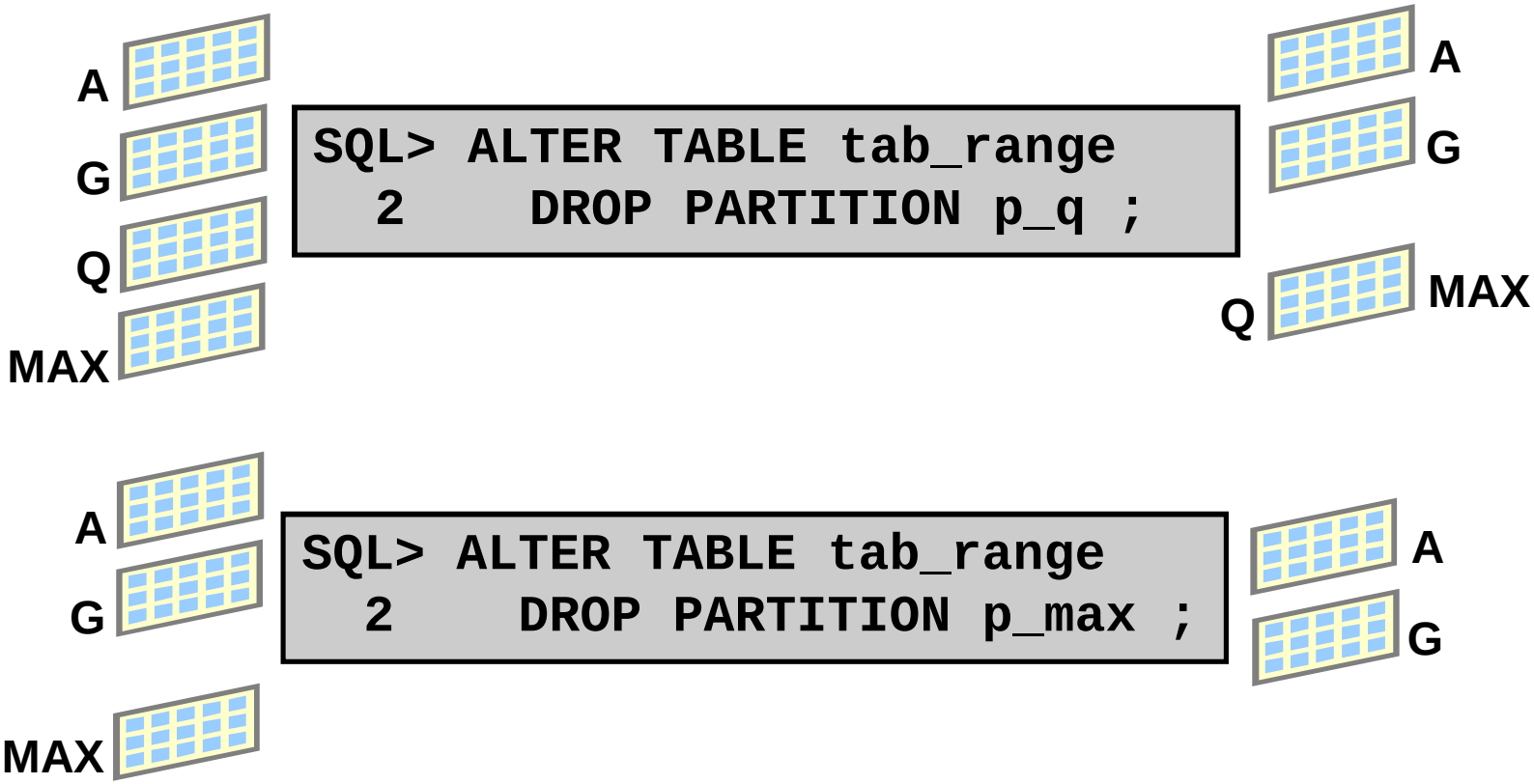
- Discards the rows contained quickly, without rollback
- Only Range and List Partitions can be dropped.
- One partition must remain. Or drop the table.
- You can drop a partition from a global index.
  - Cannot drop the last partition
  - Dropped index entries are recreated on rebuilding
  - The previous partition is marked UNUSABLE, unless the dropped partition is empty
- Local indexes partitions follow the table partitions.

When to Drop a Partition

When changes in data require it:

- **Rolling window (range partition)**
- **Obsolete key values (list partition)**

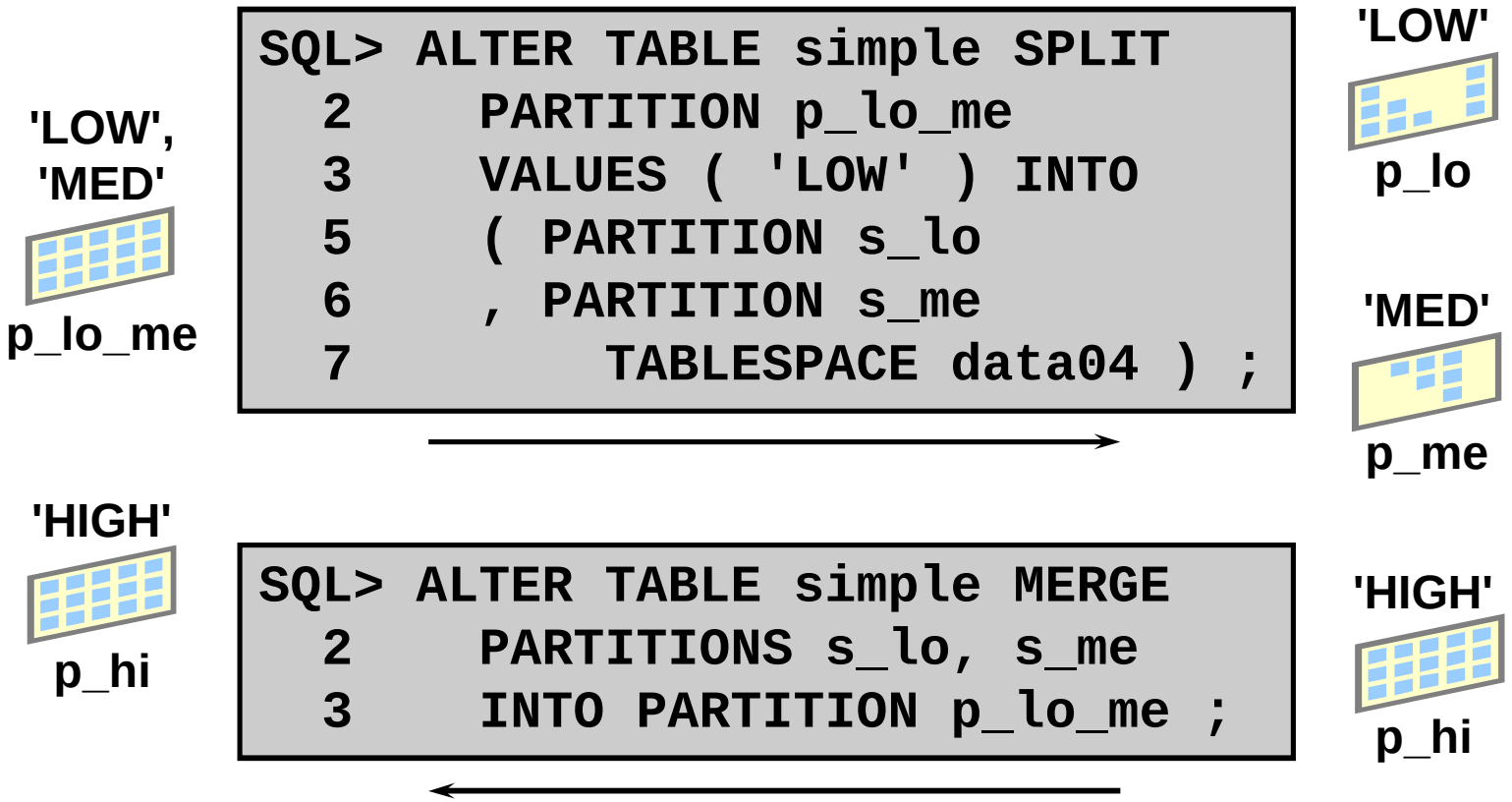
Dropping a Partition: Examples



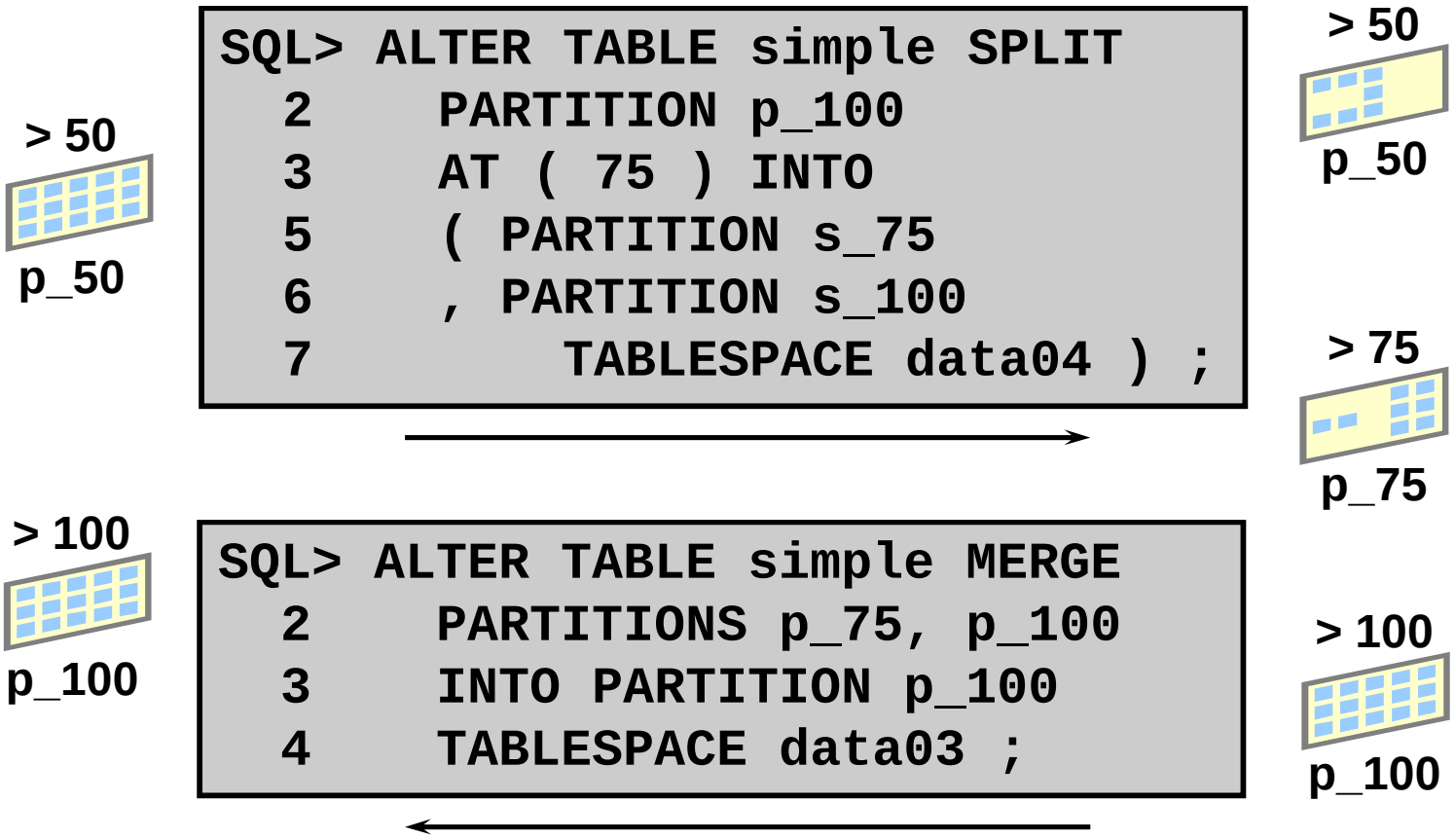
Splitting and Merging a Partition

- Splitting a partition creates two new partitions filled with rows of the split partition, which is discarded.
- Merging a partition collects the rows from two partitions and drops one of them.
- For range partitions, it involves two consecutive partitions.
- Hash partitions or subpartitions cannot be split or merged.
- You can split a partition on a global index.
- Local index result from split be marked unsuable.
- Global indexs are marked unusable.

Splitting and Merging: List Partitions



Splitting and Merging: Range Partitions



Altering List Partition Key Values

The key list in a list partition can be altered, as long as no rows are affected.

```
SQL> ALTER TABLE simple MODIFY
  2   PARTITION p_high
  3   ADD VALUES ( 'ULTRA', 'EXTREME' ) ;
```

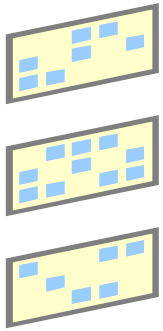
```
SQL> ALTER TABLE simple MODIFY
  2   PARTITION p_high
  3   DROP VALUES ( 'ULTRA' ) ;
```

Coalescing a Partition

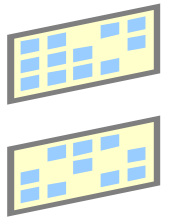
- The `COALESCE` command for hash and subpartitions has the same effects as the `MERGE` command on non-hash partitions:
  - Rows are distributed to other partitions.
  - The partition is dropped.
- Using the `COALESCE` command for a partition of an IOT table will reorganize the IOT.

Coalescing a Partition: Examples

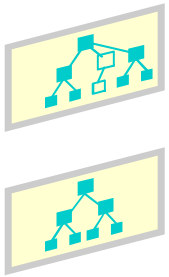
- Coalesce (merge hash partition)



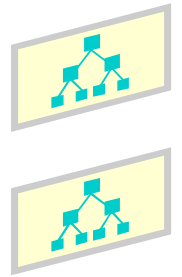
```
SQL> ALTER TABLE simple COALESCE
  PARTITION ;
```



- Coalesce (reorganize IOT)



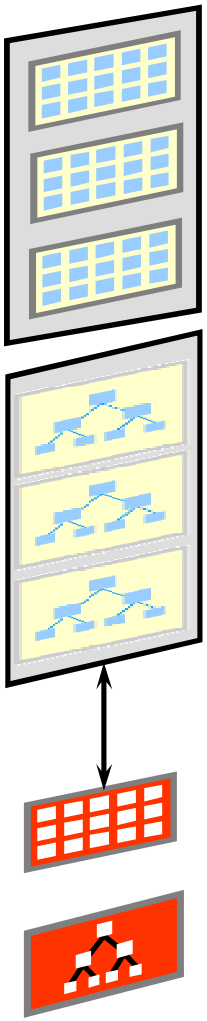
```
SQL> ALTER TABLE simple MODIFY
  PARTITION p1 COALESCE ;
```



Exchanging a Partition with a Table

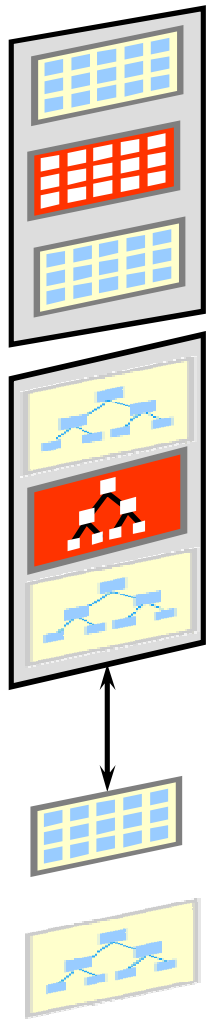
- A range or hash partition can be exchanged with a nonpartitioned table.
  - This is done by *swapping the names*.
  - You can work offline on the swapped data.
  - A hash subpartition can be swapped with a hash partition.
- The nonpartitioned table must have the same structure as the partitioned table.
- The exchange operation will verify partition key conformity by default.

Exchanging a Partition: Example



```
SQL> ALTER TABLE simple
  2   EXCHANGE PARTITION s_h1
  3   WITH TABLE tiny
  4   INCLUDING INDEXES
  5   WITHOUT VALIDATION ;
```

```
SQL> ANALYZE TABLE simple
  2   PARTITION (s_h1)
  3   VALIDATE STRUCTURE
  4      INTO INVALID_ROWS ;
```



Rebuilding Indexes

- If the partition operation has made an index unusable, it must be rebuilt.
- If the index is invalid, it must be dropped and re-created.
- Partitioned indexes must have each affected partition processed separately. You cannot rebuild a partitioned index as one whole index.

Rebuilding an Index: Examples

```
SQL> ALTER INDEX s_glo
  2    REBUILD PARTITION s_g1 ;
```

```
SQL> ALTER TABLE simple
  2    MODIFY PARTITION s_h1
  3    REBUILD UNUSABLE LOCAL INDEXES ;
```

```
SQL> ALTER INDEX s_cmp_idx
  2    REBUILD SUBPARTITION sys_subp453 ;
```

```
SQL> ALTER TABLE simple
  2    MODIFY SUBPARTITION sys_subp453
  3    REBUILD UNUSABLE LOCAL INDEXES ;
```

Benefits and Costs of UPDATE GLOBAL INDEXES

When using the UPDATE GLOBAL INDEXES clause:

- \+ Global indexes remain useable and available, even during the partition operation.
- \+ You do not have to perform a number of rebuild operations.
- – The partition operation will take longer.
- – The resultant global index may be larger.
- – You can not specify NOLOGGING.

IOT Overflow and LOB Segments

- **When altering table partitions:**
  - **any LOB partitions will correspondingly change.**
  - **Any `OVERFLOW` partitions will correspondingly change.**
  - **Storage attributes of LOB or `OVERFLOW` segments can be explicitly specified.**

Summary

In this lesson, you should have learned how to:

- Modify attributes of a partitioned table or index
- Drop, add, split, merge, coalesce, move, exchange, and truncate partitions on tables
- Drop, split, merge, and rebuild partitions on indexes
- List the index invalidations that occur with separate table partition operations