DATABASE ADMINISTRATION

TI. Database Server Architecture

C3: Oracle Storage Structures

Plan

- 2. Operating Oracle Database Server Internals
  - 2.1 Structures of Oracle Instance
    - Server Processes, Background Processes, Internal Memory
  - 2.2 Oracle Instance life cycle
    - Starting and stopping stages, Oracle Instance Data Query Processing
  - 2.3 Oracle Database Storage Structures
    - Data Files, Tablespaces, Blocks and extents, Data Segments
    - Database creation process
  - 2.4 Table storage
    - Table space allocation, Table segments, Cluster Segments, Partitions
  - **2.5 Index storage**
    - **Index organization, Index space allocation, IO-Tables, Index Partitions**

2.4 Storing SQL Tables
Table storage structures

- Data Storage Types/Data Segment Types for SQL Data in tables and indexes:
  - Table Data Storage:
    - TABLE segments (conventional data storage);
    - CLUSTER segments;
    - TABLE PARTITION segments;
    - INDEX segments;
  - Index Data Storage:
    - INDEX segments;
    - INDEX PARTITION segments.

CLUSTER Segments to store SQL table rows

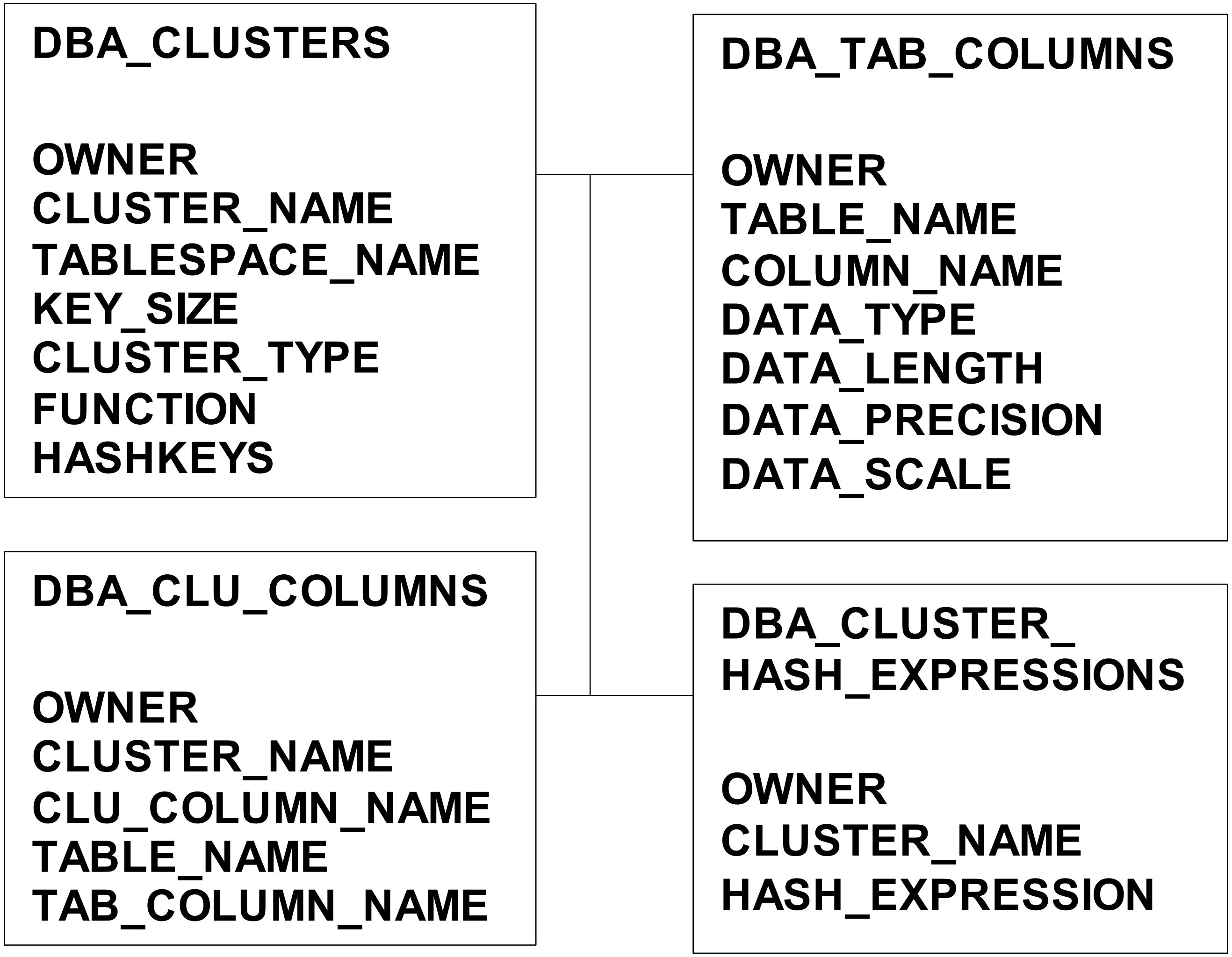
- Rows from a clustered table will be physically grouped by a cluster-key within table physical storage.
- Rows from different logical tables (usually related by a foreign key - parent key relationship) could be located the same table-block by sharing the same grouping key.
- The rows of clustered tables could be grouped:
  - by some indexing (and sorting) key resulting an *index cluster*;
  - by some hash-based key resulting a *hash cluster*: the same hash function will be used:
    - at INSERT-time to determine physical location (data block id) to store the new rows and_
    - at SELECT-time to access physical rows by filtering predicates based on the cluster key.

INDEX and HASH Cluster CREATE commands

```
CREATE CLUSTER cluster
      ( column datatype
            [ column datatype ] …)
      [PCTFREE integer]
      [PCTUSED integer]
      [INITRANS integer]
          [MAXTRANS integer]
          [SIZE integer [ K|M ]
            [STORAGE  storage-clause]
            [LOGGING | NOLOGGING]
          [NOSORT]
      [TABLESPACE tablespace]
      [INDEX];
CREATE INDEX ... ON CLUSTER …;
CREATE TABLE … CLUSTER ...(key);
```

```
CREATE CLUSTER cluster
      ( column datatype
            [ column datatype ] …)
      HASHKEYS integer
      [SINGLE TABLE]
      [HASH IS expression]
      [PCTFREE integer]
      [PCTUSED integer]
      [INITRANS integer]
          [MAXTRANS integer]
          [SIZE integer [ K|M ]
            [STORAGE  storage-clause]
            [LOGGING | NOLOGGING]
          [NOSORT]
            [TABLESPACE tablespace];
CREATE TABLE … CLUSTER ...(key);
```

Dictionary Views on Clusters



DBA_CLUSTERS

OWNER
CLUSTER_NAME
TABLESPACE_NAME
KEY_SIZE
CLUSTER_TYPE
FUNCTION
HASHKEYS

DBA_TAB_COLUMNS

OWNER
TABLE_NAME
COLUMN_NAME
DATA_TYPE
DATA_LENGTH
DATA_PRECISION
DATA_SCALE

DBA_CLU_COLUMNS

OWNER
CLUSTER_NAME
CLU_COLUMN_NAME
TABLE_NAME
TAB_COLUMN_NAME

DBA_CLUSTER_HASH_EXPRESSIONS

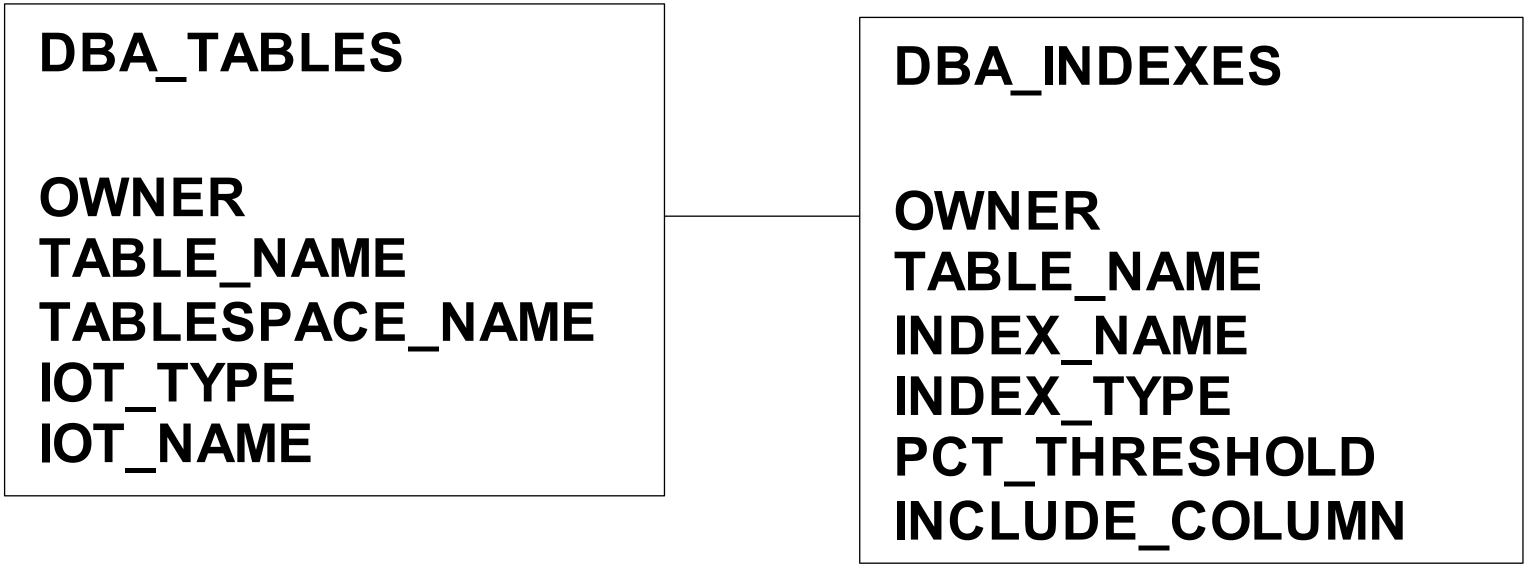
OWNER
CLUSTER_NAME
HASH_EXPRESSION

IOT: Index Organized Tables

- The ROWs within IOTs will be entirely *ordered* by using a sort-key that will finally define the B*Tree *index* segment structure.
- An IOT has the same root-to-branches-to-leafs structure as a regular index, the difference consists in storing all row values and not just the ROWID into the leaf entries.

```
CREATE TABLE table
    (column datatype [,column datatype ] …)
    ORGANIZATION INDEX
    [TABLESPACE tablespace]
    [PCTFREE integer]
    [INITRANS integer]
               [MAXTRANS integer]
               [STORAGE  storage-clause]
               [PCTTHRESHOLD integer
    [INCLUDING column]]
               [OVERFLOW segment_attributes_clause ]
```

Dictionary Views on IOTs



DBA_TABLES

OWNER
TABLE_NAME
TABLESPACE_NAME
IOT_TYPE
IOT_NAME

DBA_INDEXES

OWNER
TABLE_NAME
INDEX_NAME
INDEX_TYPE
PCT_THRESHOLD
INCLUDE_COLUMN

TABLE PARTITIONS

- **Splitting** tables into partitions:
  - assumes a grouping strategy for tables rows based on:
    - range partitioning or interval partitioning;
    - list partitioning, where partition key values will explicitly be declared;
    - hash partitioning, using hash functions on partition-keys to determine partition location for table rows.
  - produces a storage distribution for a single table as a set of data segments (of PARTITION type); those segments could eventually be distributed into different cluster nodes (by using TABLESPACE clause and TABLESPACE flexible mechanism to manage storage allocation).

RANGE PARTITION TYPE

```
CREATE TABLE tbl_range_partitioned (
    range_key_col INTEGER,
    misc_data_col VARCHAR2(100)
)
PARTITION BY RANGE (range_key_col)
(
    PARTITION ptbl_1 VALUES LESS THAN (1000)
        TABLESPACE TS1,
    PARTITION ptbl_2 VALUES LESS THAN (2000)
        TABLESPACE TS2,
    PARTITION ptbl_3 VALUES LESS THAN (3000)
        TABLESPACE TS3,
    PARTITION ptbl_4 VALUES LESS THAN (maxvalue)
        TABLESPACE TS4
);
```

Practice C3_P1

Script	Topics

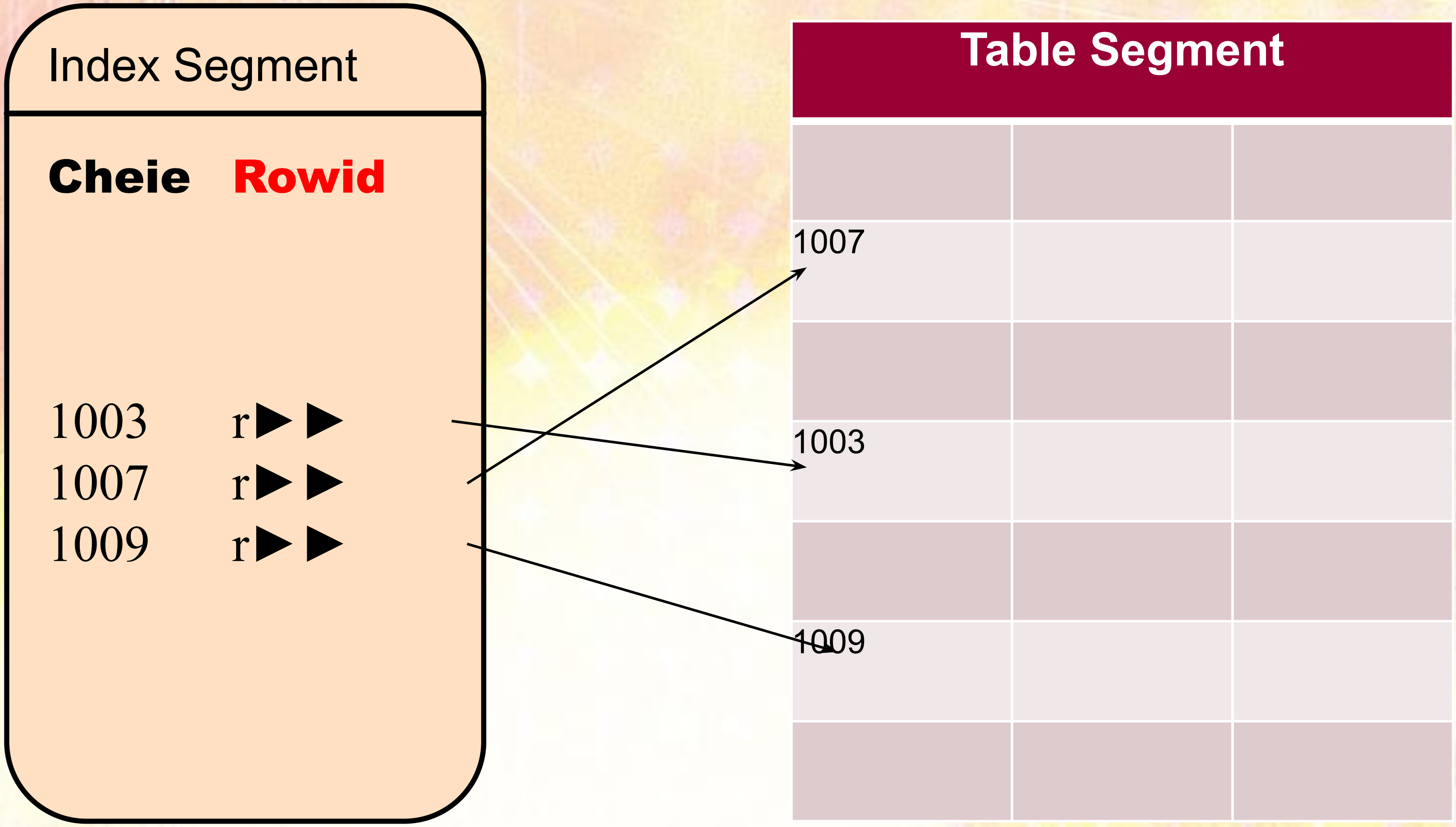
C3_P1.1.Base_Schema_CREATE_TABLE_STORAGE_TYPES.sql	Recreate tables for OFA with different storage types:
	CREATE TABLE INDEX ORGANIZED
	CREATE CLUSTER
	CREATE PARTITION

C3_P1.2.TABLE_STORAGE_Stats.sql	Stats

2.5 Storing Indexes
INDEX Storage Structures

- **Indexes** represent the main access paths to table data:
  - index blocks are loaded before table blocks;
  - then index blocks are scanned by using SQL phrase *predicates* (WHERE clauses) to get the entries with the ROWIDs targeting the original table blocks.
- Index internal organization:
  - (unlike “heap” tables) each new entry will have a pre-computed location: index entries will be physically ordered by the index key;
  - index blocks are hierarchically organized in order to efficiently find/locate index entries containing the key-values coming matching the access predicates (from SELECT-WHERE clauses).

Index Role



Index Segment

Cheie **Rowid**

1003 r▶ ▶

1007 r▶ ▶

1009 r▶ ▶

Table Segment		
1007		
1003		
1009		

ORACLE Index Classification

- 1. The logical perspective:
  - *number of columns criteria*:
    - single-column indexes;
    - composite indexes;
  - *uniqueness criteria*:
    - uniques indexes;
    - non-uniques indexes.

- 2. The internal implementation (physical) perspective:
  - B-Tree and bitmap index types;
  - Partitioned and non-partitioned indexes.

ORACLE B*TREE and Bitmap Indexes

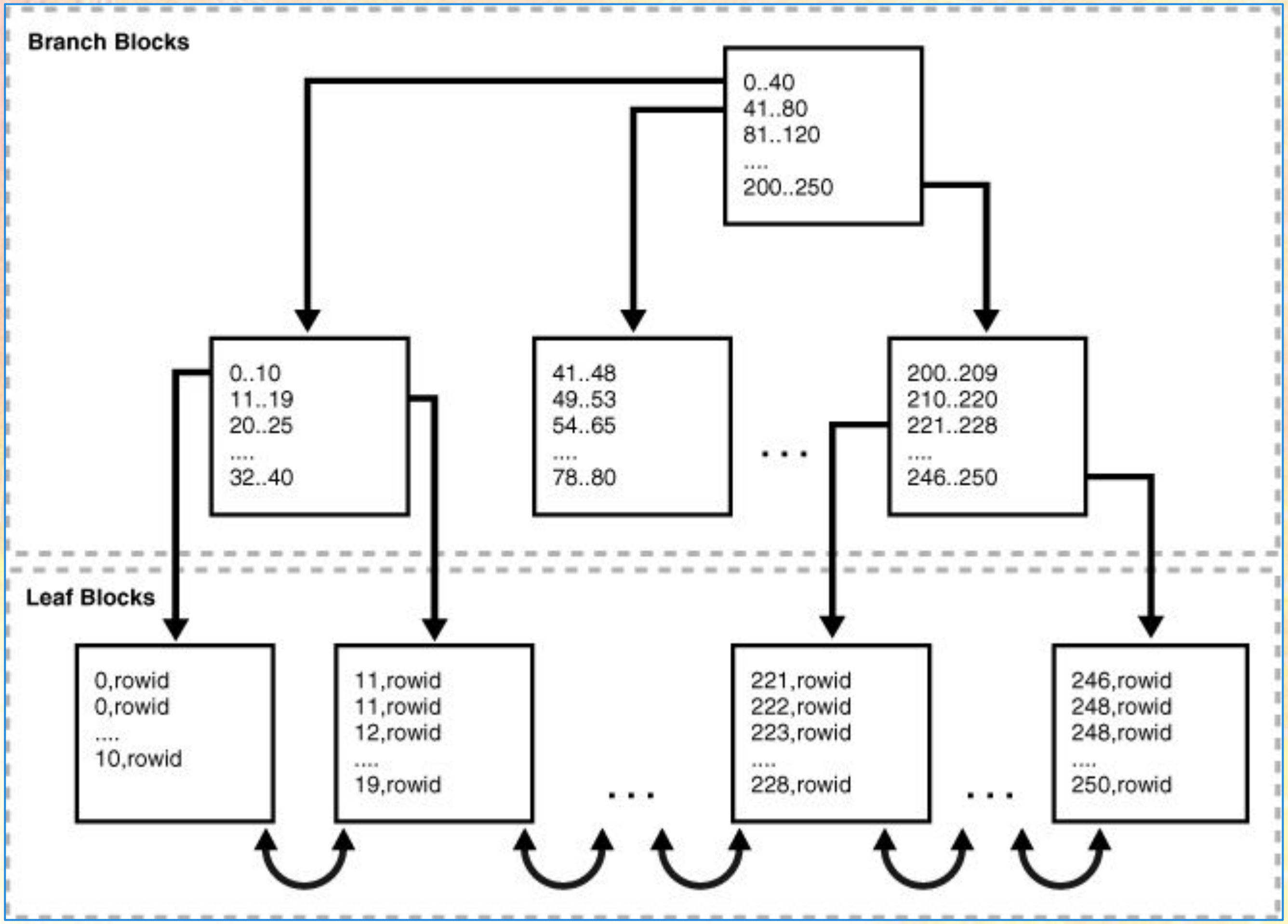
- The index Structure (B*tree or bitmap) relies on:
  - a specific inter-connected system of branch blocks:
    - starting from a unique root node containing a set of *pointers* to a chained set of branch nodes →
    - forming equally sized search paths with the same number of levels to scan; any key-value could start the search operation;
  - an ordered set of leaf-nodes containing associations of index-key-value and table ROWIDs.

ORACLE B*TREE and Bitmap Indexes
Leaf Nodes

- Leaf entries of B*TREE indexes:
    - associate each index-key-value to a single-ROWID (uniques indexes) or with a ROWID-set (non-unique indexes), each ROWID contains the location information (the file and table block) where the table row is stored.

- Leaf entries of BITMAP indexes:
    - associate each index-key-value to a BITMAP (set of bites) where every bit will correspond to a single table ROW:
        - if the bit is set to 1: the corresponding row contains the key-value;
        - if the bit is set to 0: the corresponding row do not contains the key-value.

B-Tree Index Structure



Branch Blocks

0..40
41..80
81..120
....
200..250

0..10
11..19
20..25
....
32..40

41..48
49..53
54..65
....
78..80

...

200..209
210..220
221..228
....
246..250

Leaf Blocks

0,rowid
0,rowid
....
10,rowid

11,rowid
11,rowid
12,rowid
....
19,rowid

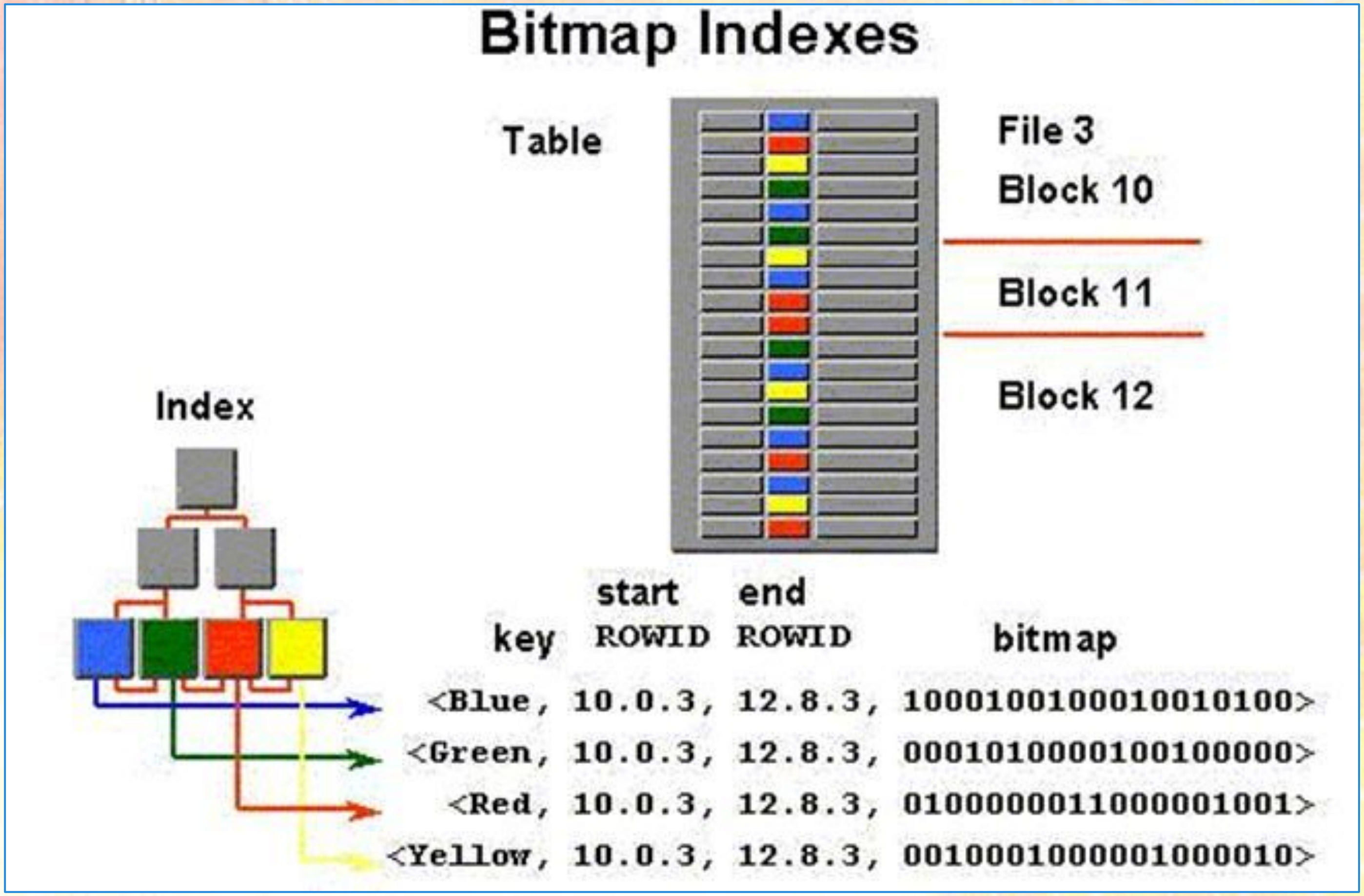
...

221,rowid
222,rowid
223,rowid
....
228,rowid

...

246,rowid
248,rowid
248,rowid
....
250,rowid

Oracle BITMAP Index



```
                 start     end
         key     ROWID     ROWID          bitmap
     <Blue,  10.0.3,  12.8.3,  1000100100010010100>
    <Green,  10.0.3,  12.8.3,  0001010000100100000>
      <Red,  10.0.3,  12.8.3,  0100000011000001001>
   <Yellow,  10.0.3,  12.8.3,  0010001000001000010>
```

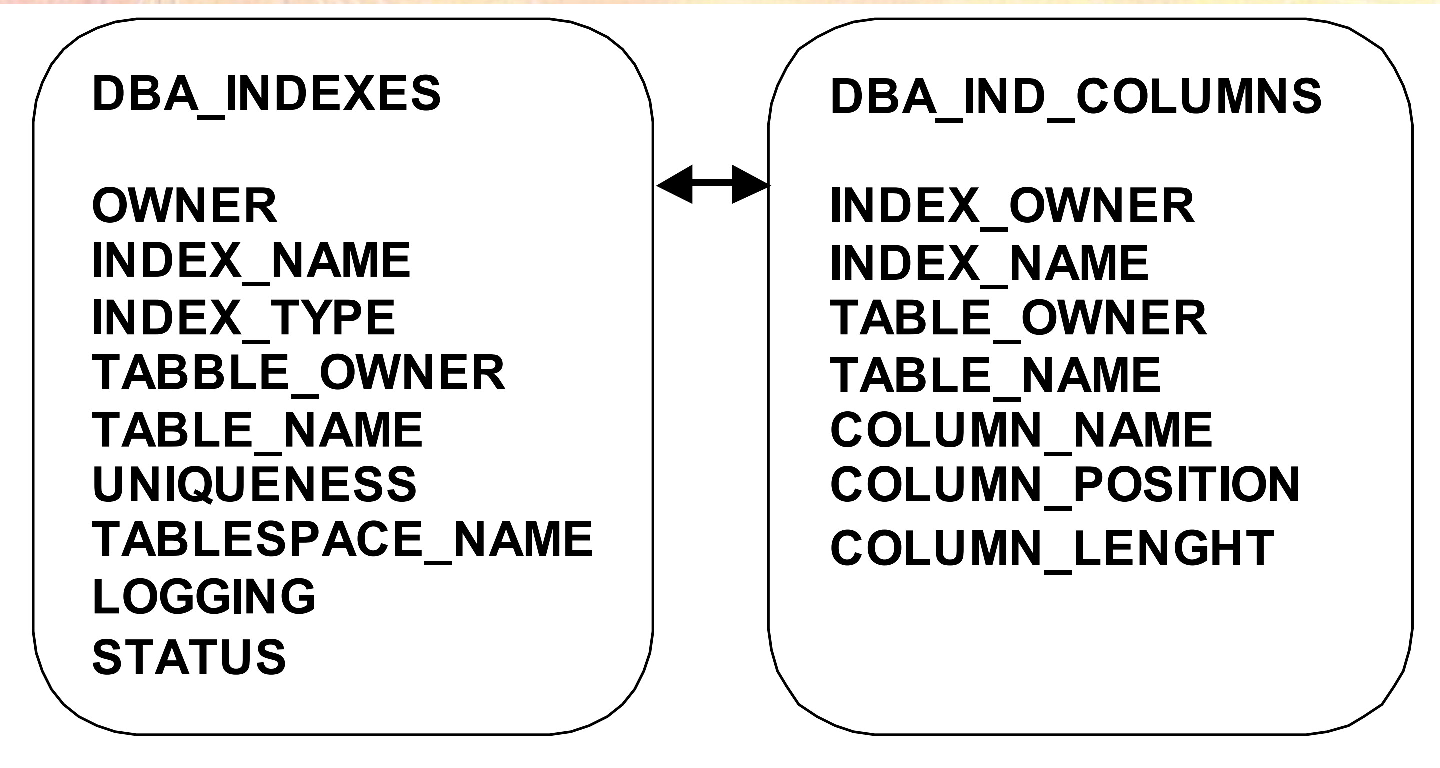
INNOVATION

CREATE INDEX and ALTER INDEX Commands

```
CREATE [UNIQUE] INDEX
    index ON table
    ( column [ column ] …)
    [TABLESPACE tablespace]
        [PCTFREE integer]
        [INITRANS integer]
        [MAXTRANS integer]
        [STORAGE  storage-clause]
        [LOGGING | NOLOGGING]
        [NOSORT]
```

```
ALTER INDEX index REBUILD
    TABLESPACE tablespacename
    [PCTFREE integer]
    [INITRANS integer]
    [MAXTRANS integer]
    [STORAGE  storage-clause]
    [LOGGING | NOLOGGING]
    [NOSORT]
    [ONLINE];
```

Dictionary Views on Indexes



DBA_INDEXES

OWNER
INDEX_NAME
INDEX_TYPE
TABBLE_OWNER
TABLE_NAME
UNIQUENESS
TABLESPACE_NAME
LOGGING
STATUS

DBA_IND_COLUMNS

INDEX_OWNER
INDEX_NAME
TABLE_OWNER
TABLE_NAME
COLUMN_NAME
COLUMN_POSITION
COLUMN_LENGHT

INDEX PARTITIONS

- Unpartitioned indexes:
  - will be created by default for unpartitioned tables;
  - could be created for partitioned tables ignoring the actual table partitioning policy.

- Partitioned Indexes come from a partitioning policy:
  - specified at the table level and reflected at the index level in order to locate in the same storage space both: each index partition is associated with one table partition (local indexes);
  - specified at the index level: independent from table partitioning rule (global indexes).

GLOBAL INDEXES

- Index partitioning clause (PARTITION BY) not related with table partitioning clause:

```
CREATE TABLE tbl_part (t_id INTEGER, t_date DATE)
        PARTITION BY RANGE (t_date)
        ( PARTITION ptbl_1
                VALUES LESS THAN (TO_DATE(‘01/01/2019’, ‘DD/MM/YYYY’)),
          PARTITION ptbl_2
                VALUES LESS THAN (TO_DATE(‘01/01/2020’, ‘DD/MM/YYYY’))
       PARTITION ptbl_3 VALUES LESS THAN (MAXVALUE));

CREATE INDEX tbl_idx_g ON tbl_part (t_date) GLOBAL
        PARTITION BY RANGE (t_date)
        ( PARTITION pidx_1
                VALUES LESS THAN (TO_DATE(‘15/10/2019’, ‘DD/MM/YYYY’)),
          PARTITION pidx_2
                VALUES LESS THAN (TO_DATE(‘15/10/2020’, ‘DD/MM/YYYY’))
       PARTITION pidx_3 VALUES LESS THAN (MAXVALUE));
```

LOCAL Indexes

- Without using any specific index partitioning clause, the actual partitioning rule for the indexes of a partitioned table will depend on the PARTITION table policy:

```
CREATE TABLE tbl_part (t_id INTEGER, t_date DATE)
    PARTITION BY RANGE (t_date)
    ( PARTITION ptbl_1 VALUES LESS THAN TO_DATE('01/08/2019',
   'DD/MM/YYYY'),
      PARTITION ptbl_2 VALUES LESS THAN TO_DATE('01/01/2020',
   'DD/MM/YYYY'));

CREATE INDEX tbl_idx _loc ON tbl_part(t_date) LOCAL;
```

Practice C3_P2

Script	Topics

C3_P1.1.Base_Schema_CREATE_TABLE_STORAGE_TYPES.sql
C1_P2.3.Base_Schema_INSERT.sql
	Recreate indexes for OFA tables:
	CREATE INDEX
	CREATE INDEX LOCAL

C3_P2.1.INDEX_STORAGE_REBUILD.sql	ALTER INDEX REBUILD
C3_P2.2.INDEX_STORAGE_Stats.sql	Index stats

References

Titles
Craig S. Mullins, *Database Administration: the complete guide to practices and procedures*, Second Edition, Addison-Wesley, 2013
Thomas Kyte and Darl Kuhn, *Expert Oracle Database Architecture*, Third Edition*, Apress, 2015*
Lahdenmaki, Tapio, Leach, Michael, *Relational database index design and optimizers: DB2, Oracle, SQL server et al*, John Wiley & Sons, 2005
Bob Bryla, Kevin Loney *Oracle Database 11g DBA Handbook*, (Oracle Press), McGraw-Hill Osborne Media, 2008