CHAPTER 12

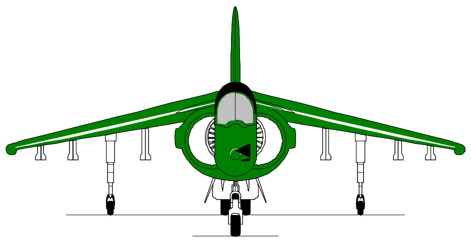
OBJECTIVES

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

- **Identify index types**
- **Identify basic access methods**
- **Monitor index usage**

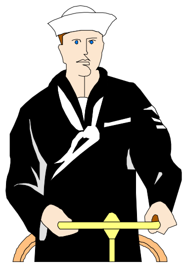
SYNONYMS

Designer


AV-8b

Pilot

Soldier


Technician

Create synonym AV8b for Harrier;
Create synonym NC8801 for Harrier;
Create synonym jumpjet for Harrier

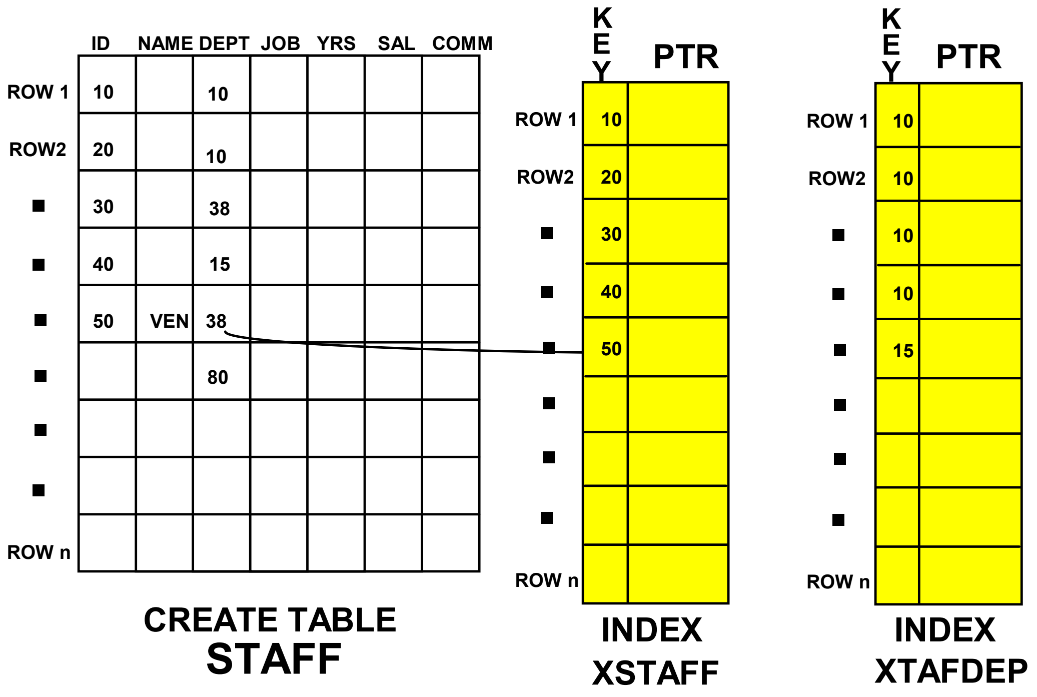
A synonym is an alternative name for a table (Base or View)

In the example, the table Harrier can now be referred to as an AV8b for the soldier, or a jumpjet to a designer of aircraft, or as NC8801 for aircraft technician who need to work on a particular craft. Each customer has his or her own name for the aircraft, which is easily identifiable by each.

```
SELECT * FROM jumpjet
```

The name jumpjet is completely private and local to the user issuing the create statement

INDEXES



1) CREATE UNIQUE INDEX XSTAFF
   ON STAFF (ID ASC)

2) CREATE INDEX XTAFDEP ON STAFF (DEPT ASC)

- Only one index per table can be designated cluster
- Indexes exist strictly to improve performance by eliminating extensive table scans
- ORACLE makes the decision to use an index for a particular query, not the user
- Example 2 would improve performance for STAFF queries based on DEPT or ORDERED BY DEPT

NON-UNIQUE INDEXES

TASK: **Create an Index so that ORACLE can retrieve data based upon an Index search versus a table scan**

SQL:

```
CREATE INDEX xname
ON STAFF (name)
```

RESULT: **Index xname is built with the key field of NAME.**

NOTES:

- A general guideline is to create an index on anything you want to search frequently.
- Primary keys should always be indexed uniquely.
- Foreign keys should also be indexed, to help you find the information they point to more efficiently.
- In the case of joins , the system can perform the join much faster if the columns are sorted.
- Must be the table owner in order to create or have Index privilege on table .

UNIQUE INDEXES

TASK: Create a unique index on the primary key field DEPTNUMB of the ORG table

SQL:

```
CREATE UNIQUE INDEX XORG
ON ORG (DEPTNUMB)
```

RESULT: Index XORG is created which will not allow duplicate Department numbers in the ORG table.

NOTES:

1. Enforces uniqueness of Rows
2. Speeds Joins (Referential checks)
3. Speeds Data Retrieval
4. Speeds ORDER BY and GROUP BY

COMPOSITE INDEXES

TASK: **Create an index on name and department on the STAFF table**

SQL:

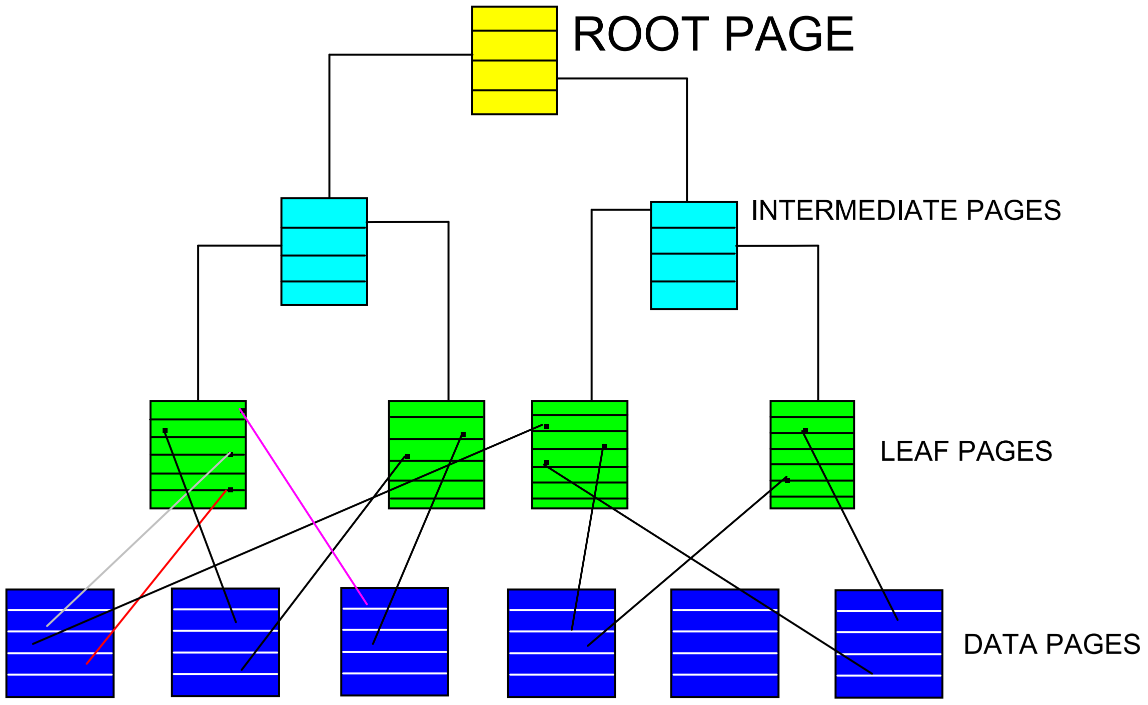
```
CREATE INDEX xstaff_dept
ON STAFF (NAME,DEPT)
```

RESULT: Index xstaff_dept is created which will allow a duplicate name and dept column in the STAFF table.

NOTES:

- A composite index specifies two or more columns as the index
- Composite columns are searched as a unit
- Column order in the CREATE INDEX statement doesn't have to match column order in the CREATE TABLE statement
- May create either a unique, non-unique or even a clustered index using a composite index.

NONCLUSTERED INDEXES



NOTES:

- Row keys are searched starting with a high key search in the root page
- The leaf level contains pointers to the rows on the data page
- The pointers add a level between the index and the data

INDEXES WITH PCTFREE

TASK: Limit the number of Index pages which are filled with rows at index creation time

SQL:

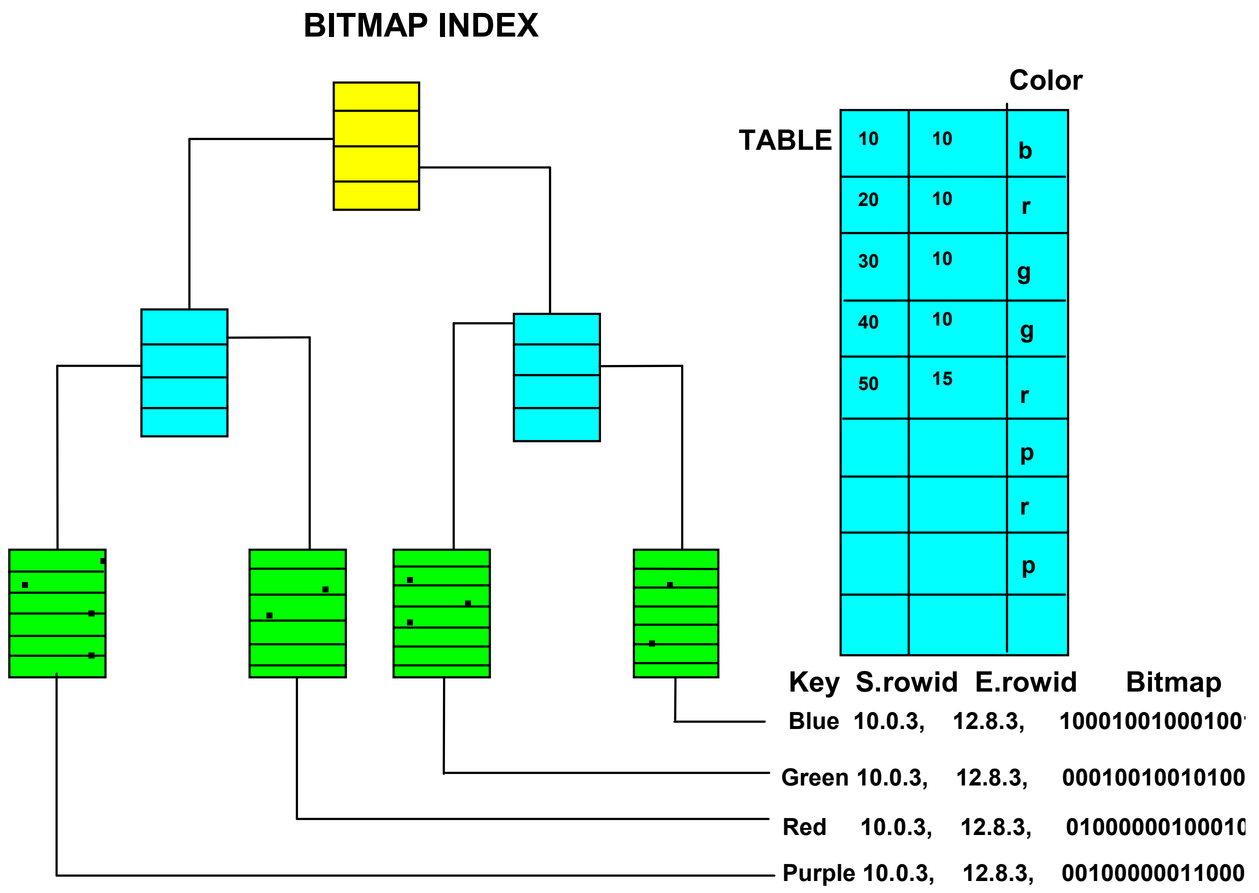
```
CREATE UNIQUE INDEX xstaff_indx
ON STAFF (id,name)
PCTFREE 10
```

RESULT: 90% of each Index page is filled at creation with rows from the STAFF table. This minimizes index page splitting and allows for fine-tuning for performance.

NOTES:

PCTFREE of 10% will keep 10% of the index page available for updates and inserts. This would be useful for tables where the index values will not change a great deal through inserts,deletions, or updates..

BITMAP INDEXES



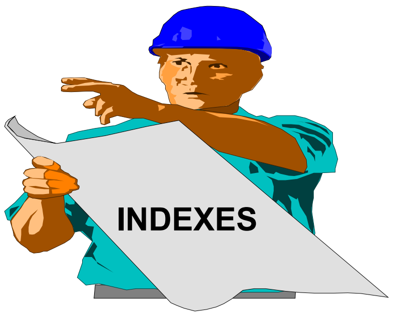
		Color
10	10	b
20	10	r
30	10	g
40	10	g
50	15	r
		p
		r
		p

Key	S.rowid	E.rowid	Bitmap
Blue	10.0.3,	12.8.3,	10001001000100
Green	10.0.3,	12.8.3,	00010010010100
Red	10.0.3,	12.8.3,	0100000010001
Purple	10.0.3,	12.8.3,	00100000011000

Bitmap Indexes are used for:

- Used for Low-cardinality columns
- Good for multiple predicates
- Use minimal storage space
- Best for read-only systems
- Good for very large tables

CREATING AND MAINTAINING BITMAP INDEXES

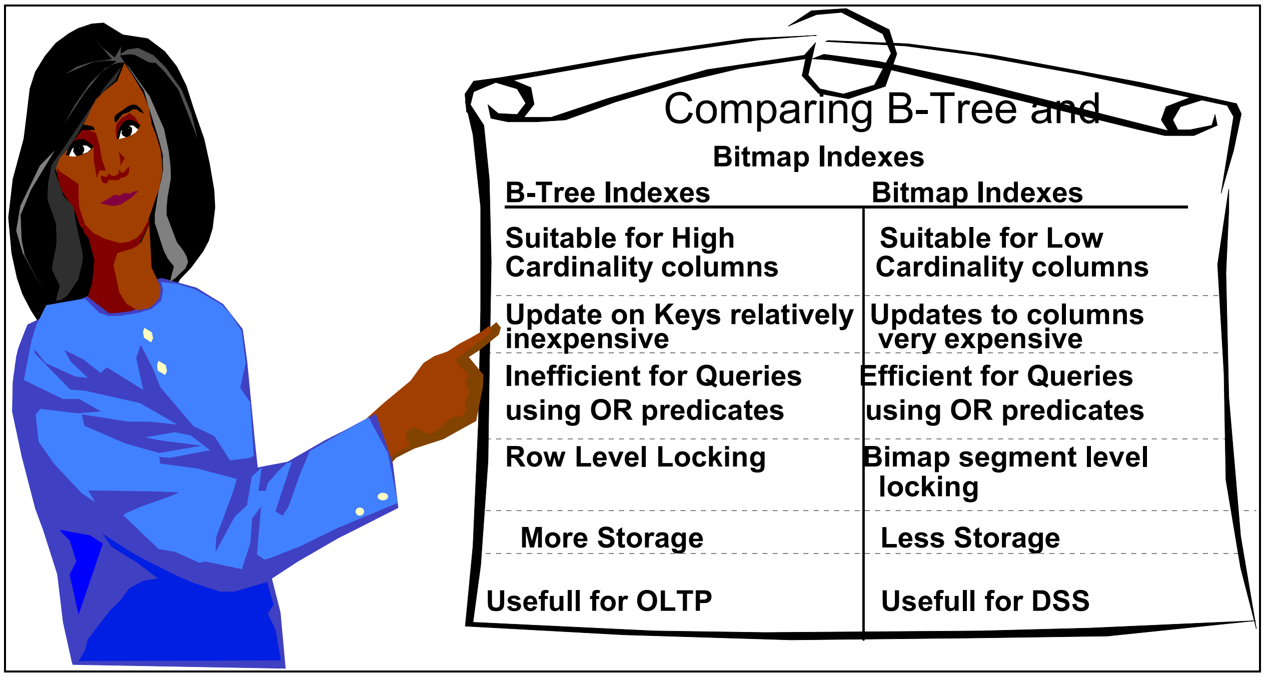


```
CREATE BITMAP INDEX INV_BIKE_COLOR
        ON  INVENTORY (COLOR)
        STORAGE (INITIAL 200K  NEXT 200K
        PCTINCREASE 0)  TABLESPACE ORASERVE;
```

- CREATE_BITMAP_AREA_SIZE defines the amount of memory for an index creation. The default is 8mb. The higher the cardinality the more memory you will need.
  - BITMAP_MERGE_AREA_SIZE defines the amount of memory used to merge bitmaps retrieved from a range scan of an index. The default is 1 MB.

In a DSS environment, data is usually maintained by using bulk inserts and updates. Index maintenance is deferred until the end of each DML operation. For example, if you insert 1,000 rows into a table that has a bitmapped index, the bitmapped column information and the ROWID information from the inserted rows are placed into the reference to the sort buffer, and then the updates of all 1,000 index entries are batched. The SORT_AREA_SIZE parameter will need to be monitored when using these indexes.

INDEX COMPARISONS



Comparing B-Tree and Bitmap Indexes

B-Tree Indexes	Bitmap Indexes
Suitable for High Cardinality columns	Suitable for Low Cardinality columns
Update on Keys relatively inexpensive	Updates to columns very expensive
Inefficient for Queries using OR predicates	Efficient for Queries using OR predicates
Row Level Locking	Bimap segment level locking
More Storage	Less Storage
Usefull for OLTP	Usefull for DSS

Bitmap indexes should be investigated and tested for Data Warehousing applications, Financials and all low-cardinality accesses to tables where the predicates are not widely dispersed.

CREATING REVERSE KEY INDEXES

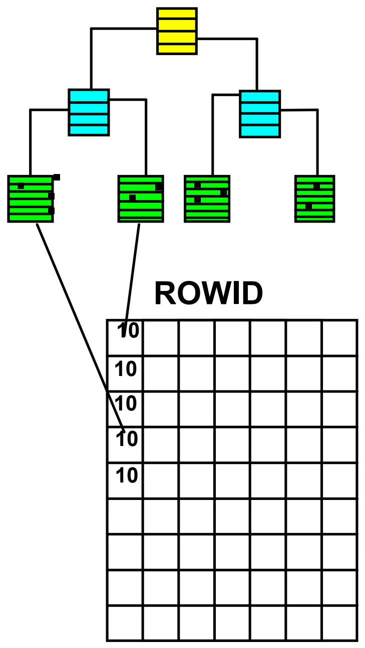
Reducing HOT SPOT Hits

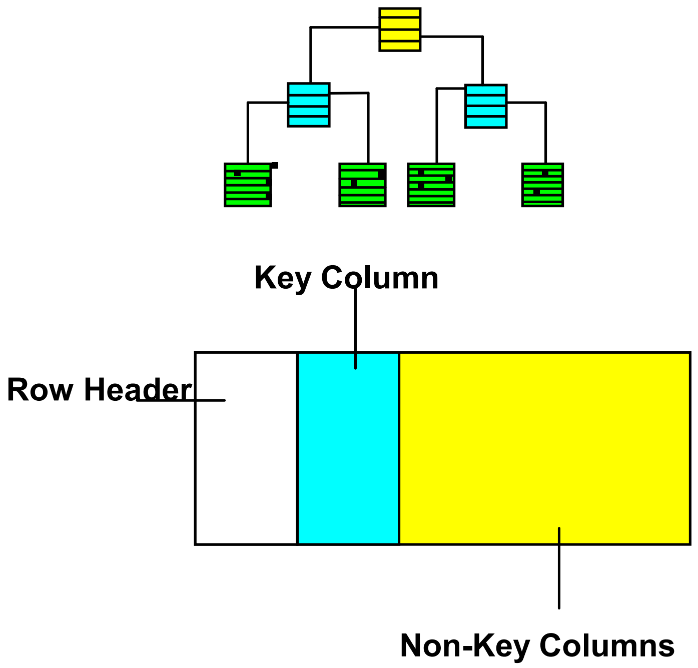
```
Create unique index po_no_idx on puorder (po_no)

REVERSE pctfree 30

storage (initial 400k  next 400k   pctincrease 0)

 tablespace jerindx.
```

Creating a reverse key index reverses the bytes of each column key value, keeping the column order in case of a composite key.

Use a Reverse key index when an ever-increasing key will repetitively degrade the index height level and cause the HOT SPOT syndrome.

INDEX-ORGANIZED TABLES

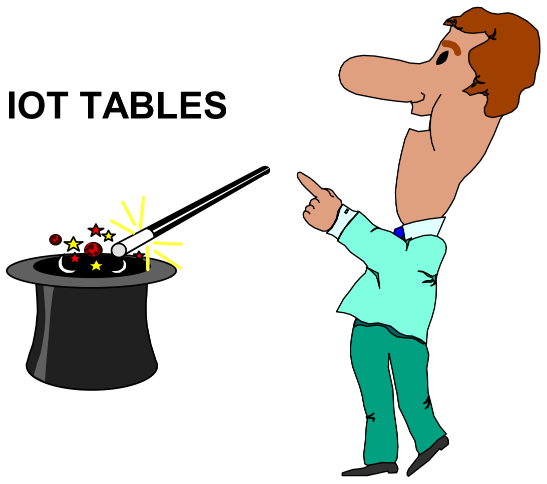




- Index contains both the primary key and other column values
- No duplication of the values for the primary key column (less storage)
- Faster key-based access for queries involving exact matches or range searches or both.
  - Logical Rowid used instead of a physical ROWID. Access to this value is through a UROWID.

CREATING INDEX-ORGANIZED TABLES

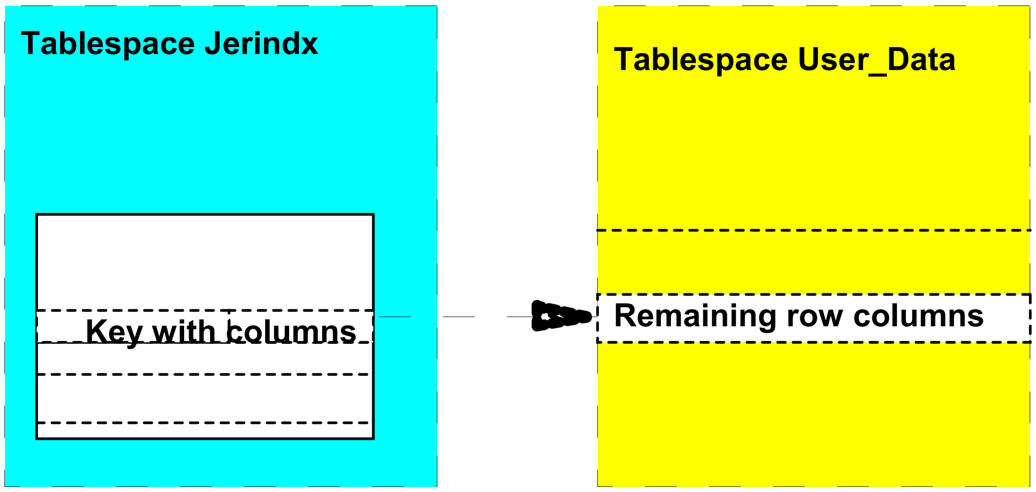
IOT TABLES



```
CREATE TABLE SALES
(OFFICE_ID           NUMBER(3) NOT NULL,
  QTR_END            DATE,
  REVENUE            NUMBER(11,2),
  REVIEW             VARCHAR2(200)
   CONSTRAINT  SALES_PK PRIMARY KEY (OFFICE_ID,QTR_END)
)
ORGANIZATION INDEX TABLESPACE JERINDX
PCTTHRESHOLD  20
INCLUDING REVENUE
OVERFLOW TABLESPACE USER_DATA;
```

IOT-TABLES store as many rows in a leaf page as PCTTHRESHOLD limit will allow. The IOT-Table will store the most frequently used columns with the primary key value. Non key column values that are not used frequently are kept in the overflow tablespace. This is called the overflow area.

IOT ROW-OVERFLOW STRATEGY



- PCTTHRESHOLD Clause

This clause specifies the percentage of space reserved in the index for an index organized table row. If a row exceeds the size calculated based upon this value, all columns after the column named in the INCLUDING clause are moved to the overflow segment. If OVERFLOW is not specified the rows exceeding the threshold are rejected. The default PCTTHRESHOLD = 50.

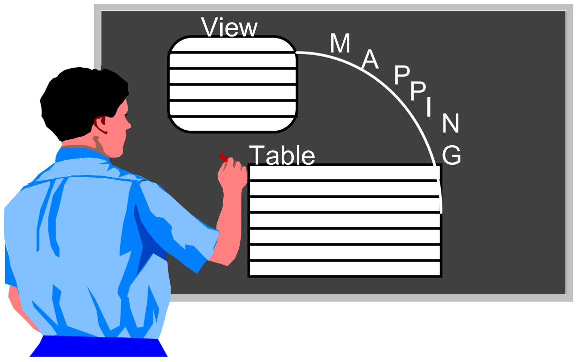
- INCLUDING Clause

This clause specifies a column to be placed into the IOT Table and all other row columns to be placed in overflow. Oracle accommodates all non-key columns up to the column specified in the INCLUDING clause in the LEAF block provided it does not exceed the specified threshold.

- OVERFLOW Clause and Segment

This clause specifies the non-key columns exceeding the specified THRESHOLD and are subsequently placed in the OVERFLOW segment.

BITMAP INDEXES ON IOT

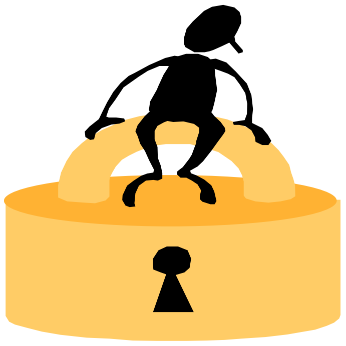


In 12c, you can create a bitmap index on an IOT table. A mapping table is used to resolve ROWIDs needed for the IOT, and the physical ROWIDs needed by the bitmap index. The mapping table is built by specifying the MAPPING TABLE clause when creating the IOT.

Performance problems occur for IOT's when:

- Leaf blocks of an IOT split and cause performance hits
- Analyze indexes and query PCT_DIRECT_ACCESS in the DBA_INDEXES table to see percentage of rows gotten through A correct guess on the IOT
- Rebuild mapping tables with the ALTER Table command, including The MAPPING TABLE UPDATE BLOCK REFERENCES clause.

FUNCTION-BASED INDEXES


One of the keys to using indexes

Consider the following query of the STAFF table that identifies information about a specific staff record, based on the uppercase version of the name field.

```
SELECT ID, NAME, JOB, SALARY
FROM staff
WHERE UPPER(NAME) = ‘RICH’;
```

In this case, a normal index on the NAME field would not be optimal because the index contains each staff record’s NAME field as it exists in the record itself (in other words, in all uppercase letters, in all lowercase letters, or in a mixed case of letters). Consequently, to do a comparison with an index on the NAME column Oracle needs to convert each key in the index to uppercase letters and then compare the resulting string to the search criteria.

To overcome this overhead processing we create a function-based index:

```
SQL>  CREATE INDEX staff_upper_name
        ON STAFF (UPPER (NAME));
```

In 12c an index-only scans required the index column to be NOT NULL in order to be used. This meant expression used in Function -based indexes could not be used because expression are always defined as NULL. Oracle12c corrects that.

Example: Function-Based Index for Case-Insensitive Searches

The following command allows faster case-insensitive searches in table EMP_TAB.

```
CREATE INDEX Idx ON Emp_tab (UPPER(Ename));
```

The SELECT command uses the function-based index on UPPER(e_name) to return all of the employees with name like :KEYCOL.

```
SELECT * FROM Emp_tab WHERE UPPER(Ename) like :KEYCOL;
```

Example: Precomputing Arithmetic Expressions with a Function-Based Index

The following command computes a value for each row using columns A, B, and C, and stores the results in the index.

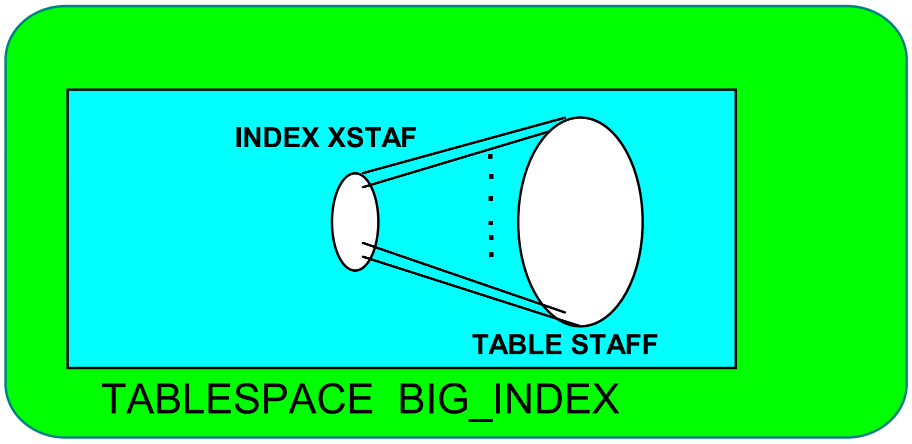
```
CREATE INDEX Idx ON Fbi_tab (A + B * (C - 1), A, B);
```

The SELECT statement can either use index range scan (since the expression is a prefix of index IDX) or index fast full scan (which may be preferable if the index has specified a high parallel degree).

```
SELECT a FROM Fbi_tab WHERE A + B * (C - 1) < 100;
```

INDEX BEFORE OR AFTER LOADING DATA

When to create Indexes?



Issues to consider when creating indexes and loading data:

- Indexes slow down data loading
- Usually more efficient to index data after loading it
- The sort area may not be large enough for extremely large tables

Consider using the NOSORT option if data has been loaded in ascending key order:

```
ORACLE>    CREATE INDEX BIG_BIGNO ON BIG(BIGNO) NOSORT
           TABLESPACE  BIG_INDEX
```

CLUSTERING TABLES

1. Use when sql joins using primary foreign keys are creating significant io problems and sorting. Ensure that the applications using the tables are usually joined together in all the applications hitting these tables. They should not have heavy DML activity on them.

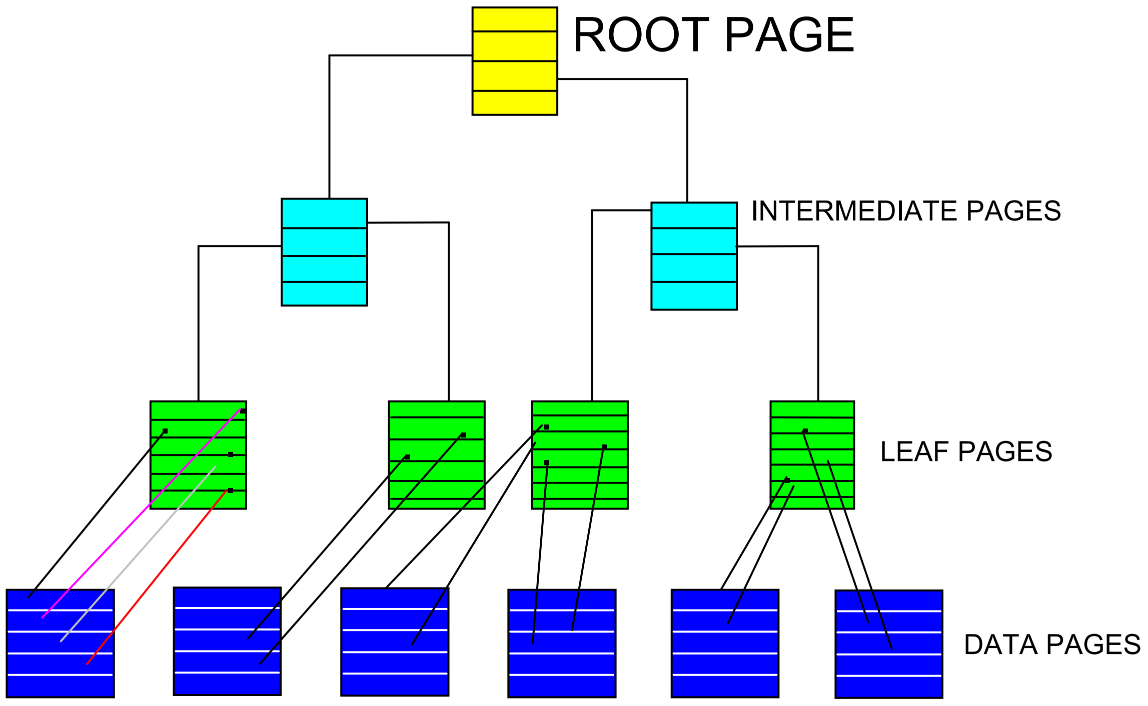
Clustering tables are used for:

- JOIN operations
- reduced storage requirements
- 'clustering' of key row values

CLUSTER BLOCK

DEPTNO	
01	**MYDEPT table row**
01	**MYEMP table row**
01	**MYEMP table row**
02	**MYDEPT table row**
02	**MYEMP table row**
02	**MYEMP table row**

CLUSTERED INDEXES



NOTES:

1. Rows from more than one table are grouped in order in a data page based upon the Cluster Index

2. The leaf level contains pointers to the rows on the data page

3. The pointers add a level between the index and the data

CREATING A CLUSTER INDEX

Task: Create the CLUSTER INDEX for the clus_dept CLUSTER and ORG and STAFF tables

******Important: You must have created the cluster before you can create the cluster index.**

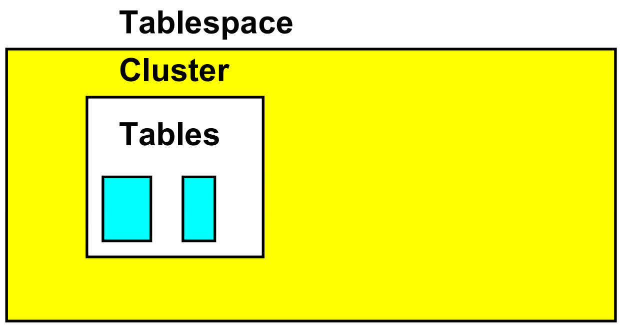
SQL:

```
CREATE INDEX staff_deptindx
on CLUSTER clus_dept
```

Notes:

To create the cluster, tables, and indexes you must have the appropriate privileges to do so. You must have either UNLIMITED TABLESPACE system privilege or CREATE ANY CLUSTER system privilege. To create a table you must have CREATE TABLE privilege or be a system administrator. To create indexes you must have the CREATE INDEX privilege at the very least.

CREATING CLUSTERS



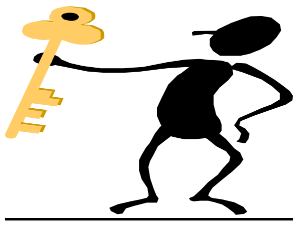
Clusters are created independently from your base tables

- Based upon common key from base tables
- Base tables clustered column must be NOT NULL
- Need to create an index on cluster key
- Typically created for empty tables

Format

```
Create cluster cluster-name
(cluster_key1 datatype, cluster_key2 datatype,....)
[space space_name]
{size logical_block_size}
[compress |no compress]
```

EXAMPLES:



1.) CREATE CLUSTER CUST_CLUS (custid number(4))
- custid is the cluster key column

2.) CREATE INDEX custndx
on CLUSTER CUST_CLUS

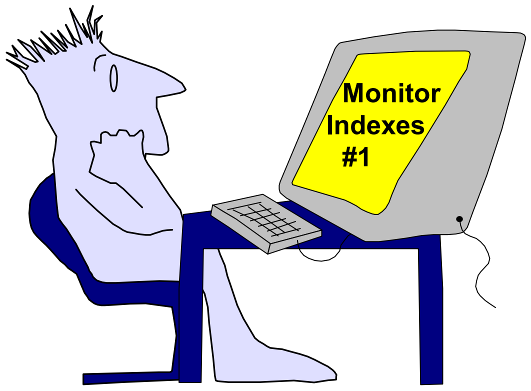
3.)
```
CREATE TABLE CUSTOMER
CLUSTER CUST_CLUS (custid)
AS
SELECT * FROM CUST

CREATE TABLE ORDERS
CLUSTER CUST_CLUS(custid)
AS
SELECT * FROM ORD
```

3.) If you do not need the previous unclustered tables and want the same names:

```
DROP TABLE CUST
RENAME CUSTOMER TO CUST
DROP TABLE ORD
RENAME ORDER TO ORD
```

4.) You cannot query a cluster, only the base tables

MONITORING INDEX SPACE IN BLOCKS



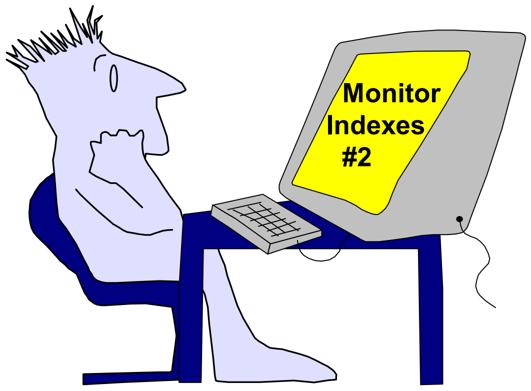
Index space usage can be examined using several methods. Lets assume we have an primary key index called YEIDX which we want to evaluate:

```
SQL> ANALYZE INDEX YEIDX VALIDATE STRUCTURE

        Index analyzed.

SQL> SELECT HEIGHT, BLOCKS, NAME, DISTINCT_KEYS,
MOST_REPEATED_KEY  MRP,PCT_USED FROM SYS.INDEX_STATS
        WHERE NAME = 'YEIDX';

HEIGHT      BLOCKS    NAME      DISTINCT_KEYS   MRP  PCT_USED
2            130      YEIDX     10000            1         88

1 ROW SELECTED

SQL> DROP INDEX  YEIDX;
SQL> CREATE INDEX YEIDX ...;
```

The method above should be performed several times to establish a base average efficiency of space usage. Once this base average is calculated it should be compared against a new percentage. When the new percentage falls below the base average, then drop and recreate to conserve space.

ORACLE stores the information from method one in a SYS view which is based on dynamic tables, therefore, the view is initialized every time a new session begins.

MONITORING INDEXES



Method two for monitoring indexes can be performed to gather certain information that is not generated in method one, e.g. index level, average number of leaf blocks/key, average number of data blocks/key, clustering factor, minimum key value, and maximum key value.

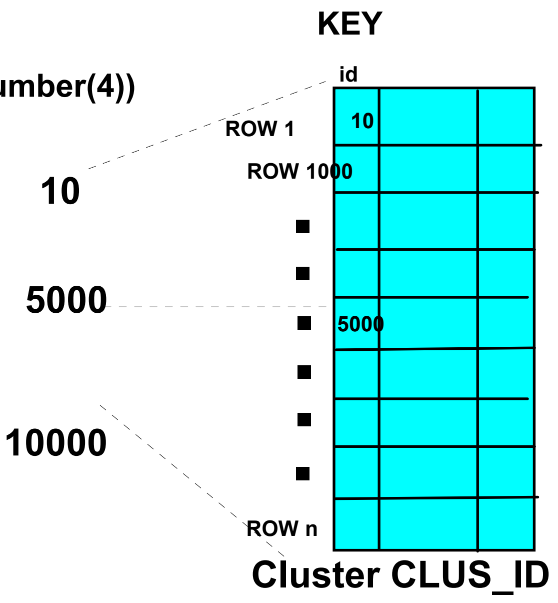
ORACLE stores the information from method two in several data dictionaries:

- ALL_INDEXES
- DBA_INDEXES
- USER_INDEXES

Information for a particular index in these data dictionary views are regenerated every time the ANALYZE INDEX command is executed

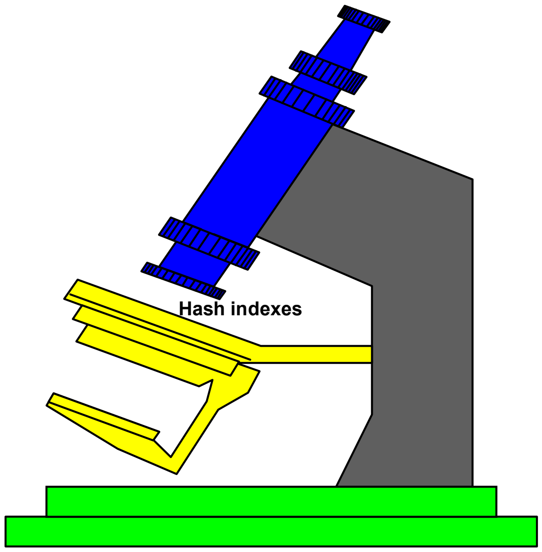
HASH CLUSTERS

```
1. Create cluster clus_id (id  number(4))
   size 70
   hash is id HASHKEYS 10000

2. Create table empid
   (id   number(4) not  null,
    name    varchar(15))
    cluster clus_id;

3. insert into empid (id, name)
   as select id,name
   from staff
```



- A hash cluster is a way of storing and accessing table data
- Applies a hash function to one or more columns
- Rows are stored as a result of the hash function
- Hashing typically reduces the number of I/Os to return data blocks
- Rows are stored as a result of the hash function
- Used when it is acceptable to pre-allocate storage space for a table
- Used when query optimization is very important
- Used when the hashed columns are frequently referenced in the where clause with an equal sign (=)
- Tables must be clustered before they are hashed

ANALYZING A HASH CLUSTER



Hash clusters can be used in place of indexes many times because their search for a key is done through an algorithm versus a key scan. To ensure your has index is working properly you can use the following steps.

```
ORACLE>  Analyze cluster clus_id compute statistics;

ORACLE>    SELECT CLUSTER_NAME,KEY_SIZE,
                        AVG_BLOCKS_PER_KEY   BLOCKS,
                        CLUSTER_TYPE,FUNCTION,
                       HASHKEYS   HASH
               FROM USER_CLUSTERS;

CLUSTER_NAME   KEY_SIZE    BLOCKS      CLUS          FUNCTION   HASH
------------------------   --------------   ------------   --------          --------------   -------
CLUS_ID                     20               50      HASH          DEFAULT     10000
```