# Indexes

*INT206 2/2020*

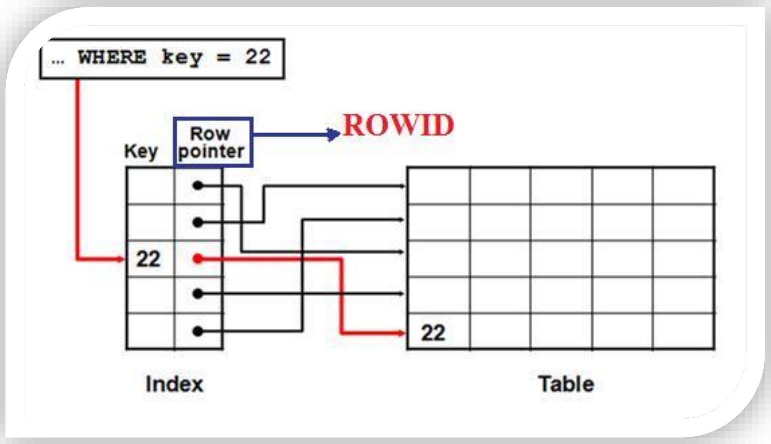
## Indexes

Indexes are database objects that you can create to improve the performance of some queries.

Advantages

- It can be used by the DB server to speed up the retrieval of rows by using a pointer.
- It can reduce disk I/O by using a rapid path access method to locate data quickly.
- It is used and maintained automatically by the database server.

ROWID



For each row in the database, the ROWID pseudocolumn returns the address of the row. You can query to see its address as below:

```
SELECT   ROWID, column_name
FROM     table_name ;
```

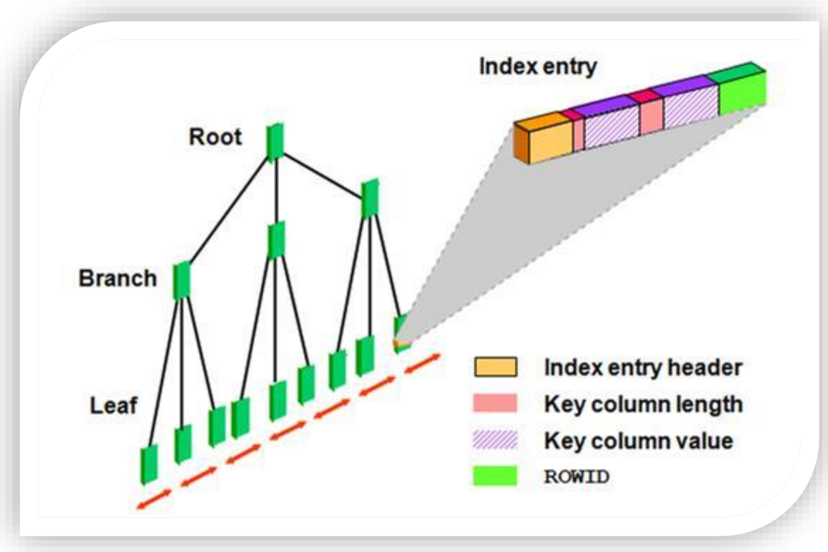
## There are 2 physical types of indexes.

### *B-Tree Index*

Although all the indexes use a B-tree structure, the term B-tree index is usually associated with an index that stores a list of ROWIDs for each key.

#### *Structure of a B-tree index*

At the top of the index is the **root**, which contains entries that point to the next level in the index. At the next level are **branch blocks**, which in turn point to blocks at the next level in the index. At the lowest level are **the leaf nodes**, which contain the index entries that point to rows in the table. The leaf blocks are doubly linked to facilitate the scanning of the index in an ascending as well as descending order of key values.



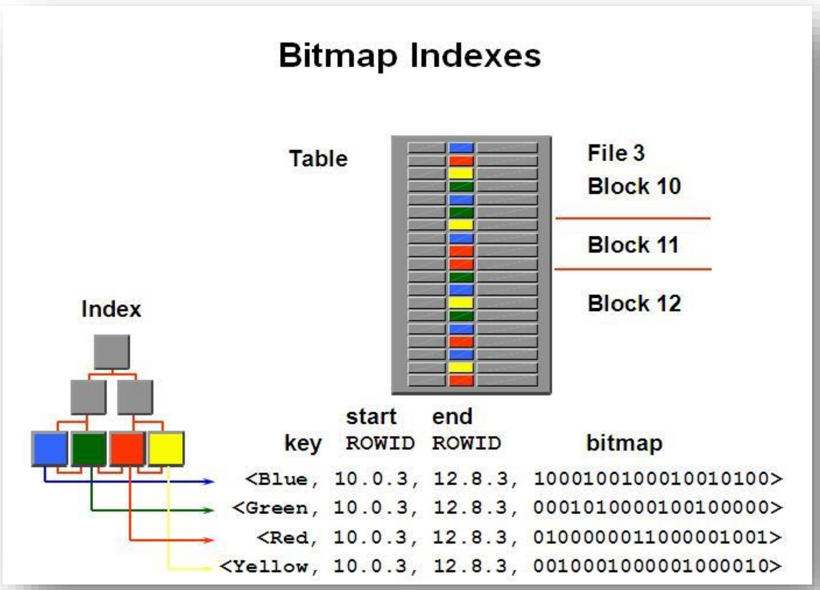
## *Bitmap Indexes*

Bitmap indexes are more advantageous than B-tree indexes in certain situations:

- When a table has millions of rows and the key columns have **low cardinality**—that is, there are very few distinct values for the column. For example, bitmap indexes may be preferable to B-tree indexes for the gender and marital status columns of a table containing passport records
- When queries often use a combination of multiple WHERE conditions involving the OR operator
- When there is **read-only** or **low update activity** on the key columns

***Structure of a bitmap index:***

A bitmap index is also organized as a B-tree, **but the leaf node** stores a **bitmap** for each key value instead of a list of ROWIDs. Each bit in the bitmap corresponds to a possible ROWID, and if the bit is set, it means that the row with the corresponding ROWID contains the key value.



```
<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>
```

## *Comparing B-Tree and Bitmap Indexes:*

| B-Tree | Bitmap |
|---|---|
| • Suitable for high-cardinality columns | • Suitable for low-cardinality columns |
| • Updates on keys relatively inexpensive | • Updates to key columns very expensive |
| • Inefficient for queries using OR predicates | • Efficient for queries using OR predicates |
| • Useful for OLTP | • Useful for data warehousing |

## ***When to Create an Index:***

Therefore, you should create indexes only if:

- The column contains a wide range of values
- The column contains a large number of null values
- One or more columns are frequently used together in a WHERE clause or join condition
- The table is large and most queries are expected to retrieve less than 2% to 4% of the rows