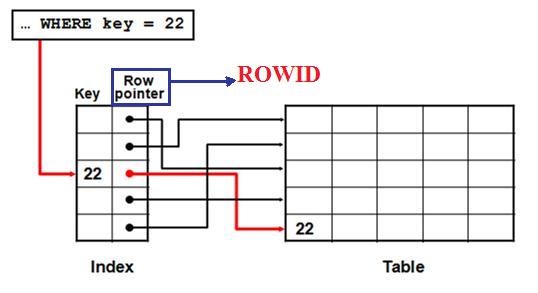
**Lab 2 Creating Secondary Indexes**

## Indexes

****

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

## Can be used by the DB server to speed up the retrieval of rows by using a pointer

## Can reduce disk I/O by using a rapid path access method to locate data quickly

## 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 ;

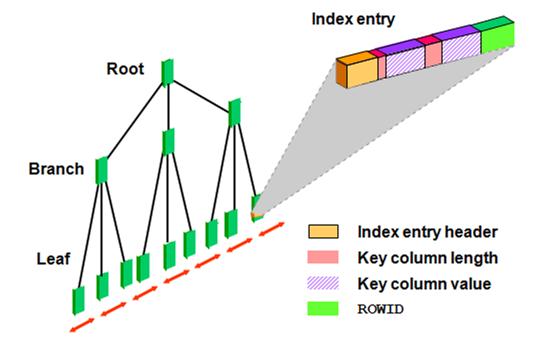
## There are 2 physical types of indexes.

## 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.



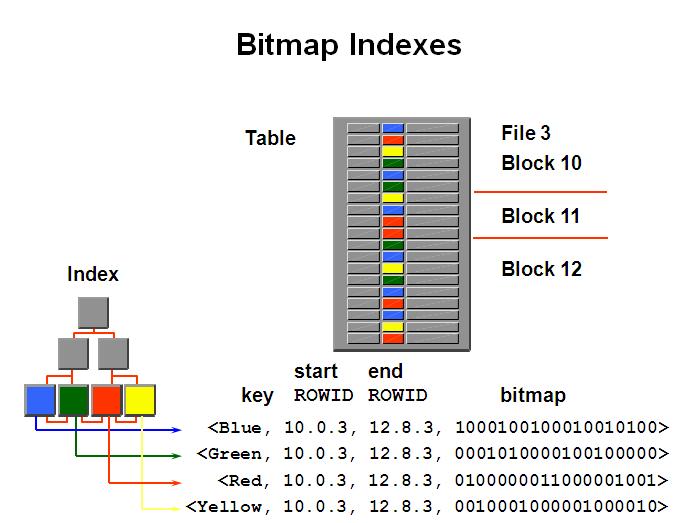
## Bitmap Indexes

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.



## 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

## CREATE INDEX Syntax:

**CREATE [UNIQUE][BITMAP] INDEX *index\_name***

**ON *table\_name* (column [, column]);**

## DROP INDEX Syntax:

**DROP INDEX *index\_name*;**

## Note: Indexes can also be created automatically by the server when you create a primary key or unique constraint.

* To create an index on one or more columns
* When creating an index without BITMAP option, the B-Tree index is the default type for the index
* Specify UNIQUE to indicate that the value of the column (or columns) upon the index is based must be unique
* Specify BITMAP to indicate that the index is to be created with a bitmap for each distinct key, rather than indexing each row separately

**Index Naming Guideline**: => *Table\_Column\_*IDX

For example: => CC\_C1\_IDX

**Example:**

* You want to improve the speed of query access to LAST\_NAME column in the EMPLOYEES table:

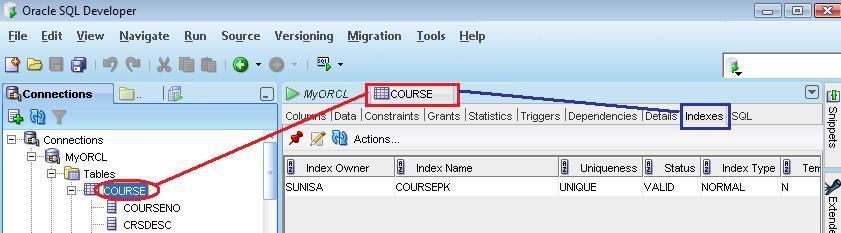
CREATE INDEX emp\_last\_name\_idx ON employees ( last\_name) ;

***Order Index columns for Performance***

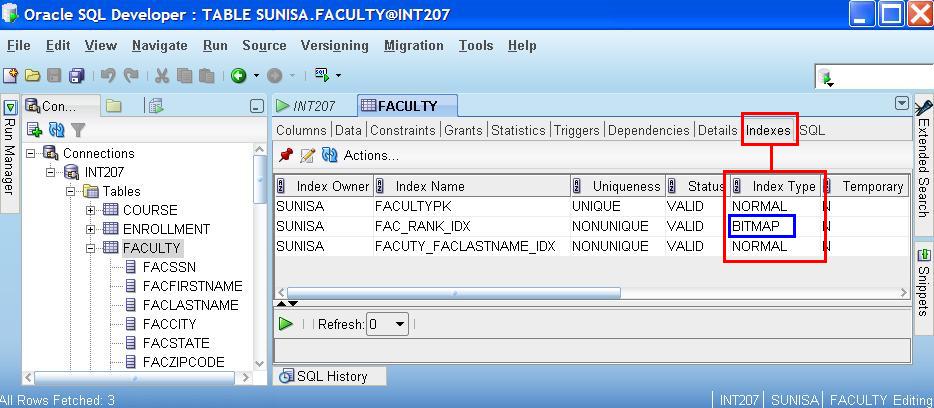
* The order of columns in the CREATE INDEX statement can affect query performance. In general, specify the most frequently used columns first.
* If you create a single index across columns to speed up queries that access
  + For example, col1, col2, and col3;
  + Then queries that access just col1, or that access just col1 and col2, are also speeded up.
  + But a query that accessed just col2, just col3, or just col2 and col3 does not use the index.

**Using SQL Developer:**

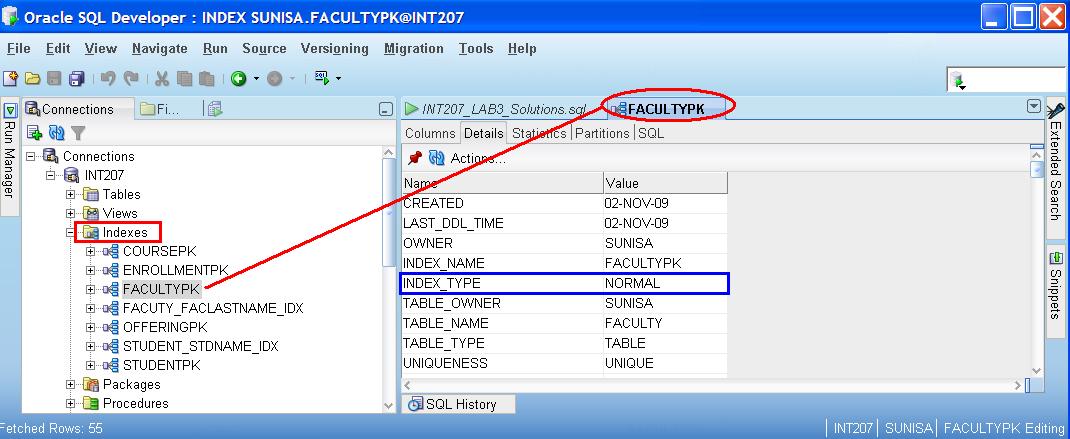
To view the existing indexes in the tables:



To view type of the existing indexes in the tables:



**OR**

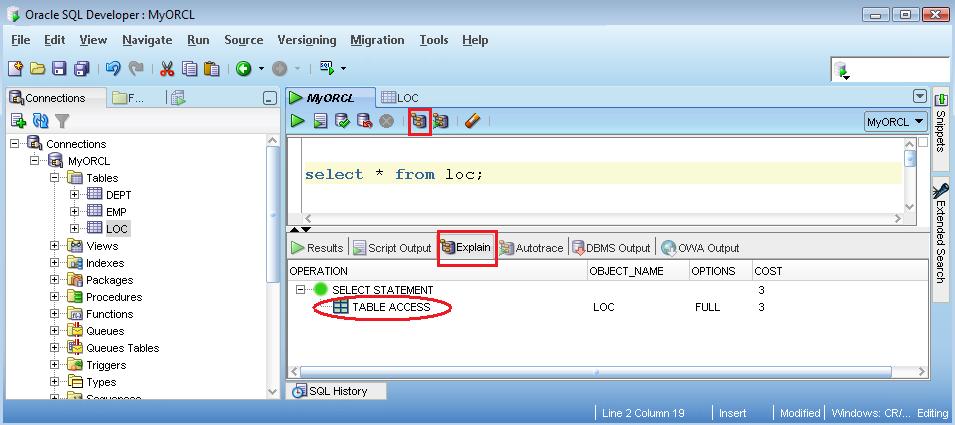


**Tip:** When you write CREATE statement to create an index, you should click ***RunScript(F5)*** RunScript.JPG to execute the statement because of DDL statement.

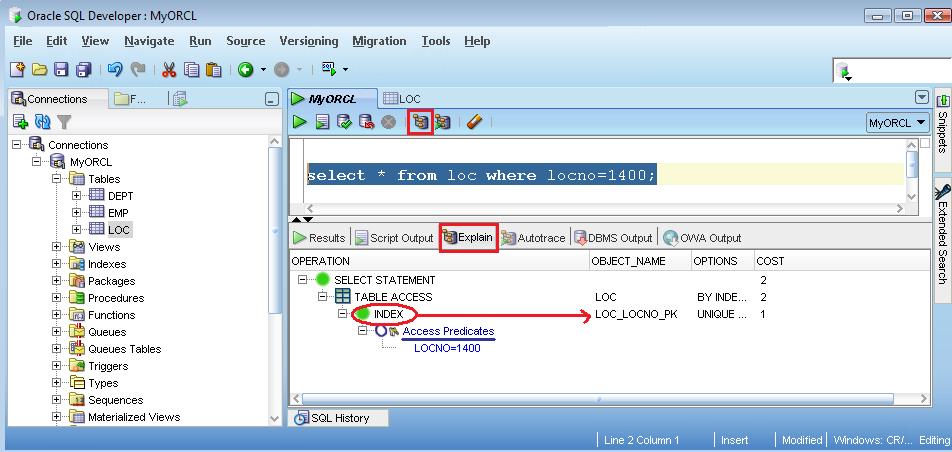
To monitor an index usage, Please click the **Execute Explain Plan** button to execute a statement.

F6.jpg **Execute Explain Plan (F6)**

**Example 1:** Table Access

****

**Example 2:** Index Access

****

***Practices***

## Run the script CC.sql to create and populate the CC table.

## How many records are there in the CC tables?

## Query the table CC to find the number of distinct values and the number of Null values in each column of the table.

|  |  |  |
| --- | --- | --- |
| **Column** | **# Distinct values** | **# Null Values** |
| **CID** |  |  |
| **C1** |  |  |
| **C2** |  |  |
| **C3** |  |  |
| **C4** |  |  |

1. View the existing indexes of the CC table. How many indexes?

|  |  |  |
| --- | --- | --- |
| **Index Name (s)** | **Index Type (s)** | **Column Name (s)** |
|  |  |  |
|  |  |  |

## Checking whether the query is executed using an index or not:

* 1. select \* from cc where cid = 1000 ;
  2. select \* from cc where cid BETWEEN 1000 and 1005 ;
  3. select \* from cc where cid in (1001, 1100, 1200) ;
  4. select \* from cc where c1 = 888 ;
  5. select \* from cc where c2 =29 ;
  6. select \* from cc where c2 =30 ;
  7. select \* from cc where c3 = 1 ;
  8. select \* from cc where c3 = 4 ;
  9. select \* from cc where c4 = 'z' ;
  10. select \* from cc where upper(c4) = 'Z' ;
  11. select \* from cc where c4 in ('a','b','c') ;
  12. select \* from cc where c4 not in ('a','b','c') ;
  13. select \* from cc where c2 = 15 and c3 =3 and c4='z' ;
  14. select \* from cc where c3 =3 and c4='z' ;

1. Create indexes on the columns of CC table and evaluate the types of index are appropriate for the specified column?

|  |  |  |
| --- | --- | --- |
| **Column Name (s)** | **Index Name (s)** | **Index Type (s)** |
| C1 |  |  |
| C2 |  |  |
| C3 |  |  |
| C4 |  |  |
| C2,C3,C4 |  |  |

View the existing indexes of the CC table. How many indexes? Which index is a composite index?

1. Checking whether the query is executed using an index or not:
   1. select \* from cc where cid = 1000 ;
   2. select \* from cc where cid BETWEEN 1000 and 1005 ;
   3. select \* from cc where cid in (1001, 1100, 1200) ;
   4. select \* from cc where c1 = 888 ;
   5. select \* from cc where c2 =29 ;
   6. select \* from cc where c2 =30 ;
   7. select \* from cc where c3 = 1 ;
   8. select \* from cc where c3 = 4 ;
   9. select \* from cc where c4 = 'z' ;
   10. select \* from cc where upper(c4) = 'Z' ;
   11. select \* from cc where c4 in ('a','b','c') ;
   12. select \* from cc where c4 not in ('a','b','c') ;
   13. select \* from cc where c2 = 15 and c3 =3 and c4='z' ;
   14. select \* from cc where c3 =3 and c4='z' ;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_