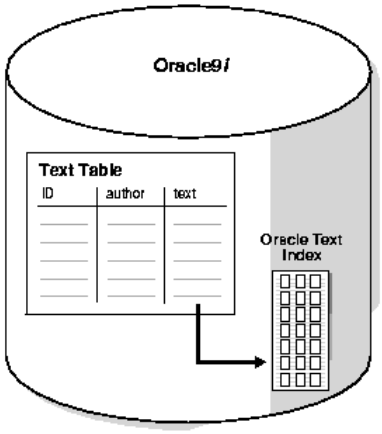
INDEXES

What is Index?

SQL Indexes are nothing, but optional structure associated with the table which may or may not improve the performance of Query.

"In simple words suppose we want to search the topic in to book we go to index page of that book and search the topic which we want. Just like that, to search the values from the table when indexing is there you need not use the full table scan."

SQL Indexes are nothing but a way of reducing the cost of the query. The more the cost of the query, the less the performance of the query. The main task of query tuner is to reduce the cost of the query using indexing, Reduce the Full table scans, reduce the time to fetch the records from the query.

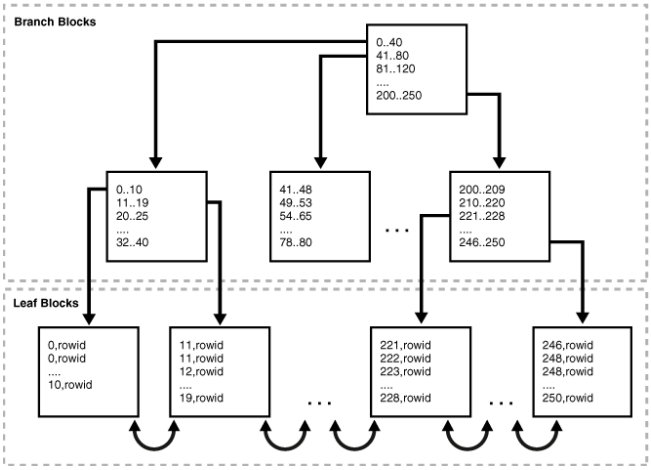


Types of SQL Indexes:

1. B-Tree
2. Bitmap

B-Tree Index:

- A B-tree index is an ordered list of values divided into ranges.
- A B-tree index has two types of blocks: **branch blocks** for searching and **leaf blocks** that store values.
- The upper-level branch blocks of a B-tree index contain index data that points to lower-level index blocks.
- The leaf blocks contain every indexed data value and a corresponding rowid used to locate the actual row.
- The purpose of an index is to provide pointers to the rows in a table that contain a given key value.
- B-tree indexes are most effective for high-cardinality data.



Oracle has no. of indexes which follow b-tree structure. They are as follows.

1. Normal index
2. Unique Index
3. Composite Index
4. Function Based Index
5. Clustered Index
6. Non-Clustered Index

- ***Normal Indexes:***

Before creating these indexes, you need to check the cost of the query. If you are using SQL developer, you can use the direct statement 'Explain Plan' before SQL Query. After checking the cost of query user needs to check whether any indexes are there on the table or not. 'ALL_INDEXES' is a data dictionary which gives user the information about indexes on the table. If Indexing is not there on table level, you can create indexes on column of the table.

Syntax of Normal Index:

```
CREATE index Index_name
ON Table_Name (Column_Name);
```

Example:

```
CREATE index NI_EMP_NAME
ON EMP(EMP_NAME);
```

- ***Unique Index:***

User needs to check the values of the table to create unique index. If the table contains uniquely identified values in specified column, then you should use unique index. Especially while creating the table if we specify the primary key then unique index is automatically created on that column. Also, for Unique key constraint columns indexing is created implicitly. Kindly make sure that Unique key indexes are created on the columns which have unique values only.

Syntax for Unique Index:

```
CREATE Unique index Index_name
ON Table_name(Unique column name);
```

Example:

```
CREATE UNIQUE INDEX UI1_EMP
ON EMP(EMP_ID);
```

- ***Composite Index:***

When 2 or more columns in a single table are related which each other and used in where condition of select statement then user should create composite index on the columns which are created. If all columns selected in the query are in composite index, then oracle will return the values from the index without accessing the table. Composite indexes should be avoided as they are large in size.

Sample Query:

```
Select e.Emp_name,d.Dept_name
from Emp e, Dept d
where e.Dept_ID = d.Dept_ID;
```

Here in the above example emp_id and dept_id is related. So, we can create an index on emp_id and dept_id.

Syntax for Composite Index:

```
CREATE index CI_ENO_DEPTNO
ON emp (emp_id, dept_id);
```

- ***Function Based Indexes:***

Function based indexes allow user to index on the functional columns so that oracle engine will take the index and improve the performance of the query. As per requirements we are using lot of SQL functions to fetch the results. Function based indexes give ability to index the computed columns. These indexes speeds up the application without changing application code or query.

Syntax:

```
CREATE index indexname
ON tablename (Function_name(column_name));
```

Example:

```
CREATE index FI_Employee
ON emp (trunc (Hire_date));
```

- ***Clustered Indexes:***

1. The clustered indexes are indexes which are physically stored in order means it stores in ascending or descending order in Database.
2. Clustered indexes are created one for each table.
3. When primary key is created then clustered index has been automatically created in the table.
4. If table is under heavy data modifications the clustered indexes are preferable to use.

Syntax:

- ***First We Have To Create A Cluster***

```
CREATE CLUSTER Cluster_Name(Column_Name Data_Type);
```

- ***Calling Cluster Key Into Table***

```
CREATE TABLE Table_Name (
        …
)
CLUSTER Cluster_Name(Column_Name);
```

- ***Then We Have To Create Index On The Cluster It Self***

```
CREATE INDEX Index_Name
ON CLUSTER Cluster_Name;
```

Example:

```
CREATE CLUSTER emp_clu(deptno NUMBER);

CREATE TABLE emp_index_clu (
      empno number,
      ename varchar(20),
      job varchar(20),
      deptno number
   )
CLUSTER emp_clu(deptno);

CREATE INDEX emp_index
ON cluster emp_clu;
```

- ***Non-Clustered Indexes:***

1. The clustered indexes are used for searching purposes as we can create clustered indexes where primary is is defined. But non clustered indexes are indexes which will be created on the multiple joining conditions, multiple filters used in query.
2. We can create 0 to 249 non-clustered indexes on a single table.
3. Foreign keys should be non-clustered. When the user wants to retrieve heavy data from fields other than primary key the non-clustered indexes are useful.

Bitmap Index:

If Table contains low cardinality value, then user should go for Bit map indexes. Users should avoid indexing on every row and do the indexing only on distinct records of the table column. We should be able to check drastic change in query cost after changing the normal index to Bit map index. The bit map indexes are very useful in data ware housing where there is a low level of concurrent transactions. Bit map index stores row_id as associated key value with bitmap and did the indexing only distinct values. Means If in 1 million records only 20 distinct values are there so Bitmap index only stores 20 values as bitmap and fetches the records from those 20 values only.

Syntax:

Create bitmap index Index_name on Table_name(Columns which have distinct values);
Example:
CREATE BITMAP index BM_DEPT_NAME on DEPT(Department_name);

Example:

"Suppose There are 2 tables which have millions of records. We need to improve the performance of Query. Now it is taking 4 mins to fetch 1 million Records."

Step 1:

```
EXPLAIN PLAN FOR
SELECT *
FROM dept d, emp e
WHERE d.dept_id = e.dept_id;

SELECT * FROM (dbms_xplan.display());
```

Output:

Cost of DEPT Table-20000 --- table is Full Scanned
Cost of EMP Tabe-20000 --- table is Full Scanned

Step 2:

Check whether there are SQL Indexes on table columns:

```
SELECT *
FROM ALL_INDEXES
WHERE table_name in ('EMP', 'DEPT');
```

Step 3:
Check description of the table and check whether the normal index where the Unique index and where bitmap indexes are applicable.

Step 4:
Creation of normal index on EMP table name column.

```
CREATE INDEX NI_EMP_NAME
ON emp(emp_name);
```

Step 5:
EMP_ID has unique values so kindly create UNIQUE INDEX ON that column. DEPT_ID has also unique values so for DEPT_ID column we need to create unique index.

```
CREATE UNIQUE INDEX UI1_EMP
ON emp(emp_id);

CREATE UNIQUE INDEX UI2_DEPT
ON dept(dept_id);
```

Step 6:
Check for Distinct values.So dept_name has 20 distinct departments so on dept_name column create Bit-map index.

```
CREATE BITMAP index  BM_DEPT_NAME
ON DEPT(dept_name);
```

Step 7:
Check the cost of Query

```
EXPLAIN PLAN FOR
SELECT *
FROM dept d, emp e
WHERE d.dept_id =  e.dept_id;

SELECT * FROM (dbms_xplan.display());
```

Output:
Cost of Emp Table :-> 20 --- Fast Unique Scan
Cost of DEPT Table:-> 10 --- Bit-map Scan
and Results will come in 10 Seconds

Interview Questions Related to Index

1. **What is an index?**
   Indexes are used in Oracle to provide quick access to rows in a table. Indexes provide faster access to data for operations that return a small portion of a table's rows. Although Oracle allows an unlimited number of indexes on a table, the indexes only help if they are used to speed up queries.

2. **How does a database know when to use an index?**
   In Oracle, the database optimizer is responsible for determining when to use an index during query execution. The optimizer's primary goal is to create an execution plan that retrieves the requested data as efficiently as possible. It evaluates various factors to decide whether to use an index or perform a full table scan.

3. **Can you force the database to use an index on a query?**
   Yes, in Oracle we can force the database to use the index by using ORACLE HINTS. These HINTS will redirect the path of execution to use the index.

4. **How to create a multi-column index in SQL?**
   It is called composite index. If the columns are used together frequently then we create composite indexes.
   Syntax:
   CREATE INDEX index_name
   ON TABLE_NAME (COLUMN_NAME1, COLUMN_NAME2,.. COLUMN_NAMEN);

5. **How to check a table has indexes or not?**
   You can check if a table has indexes in Oracle by querying the data dictionary. The primary data dictionary views that provide information about indexes are DBA_INDEXES, USER_INDEXES, and ALL_INDEXES.
   Syntax:
   SELECT INDEX_NAME
   FROM ALL_INDEXES
   WHERE TABLE_NAME = 'your_table_name'
   AND OWNER = 'schema_owner';

6. **What is the difference between B-Tree and Bitmap Index?**
   B-Tree Index:
   - B-Tree indexes are organized as balanced tree structures with nodes that contain keys and pointers to data.
   - They are designed for efficient searching, insertion, and deletion of data.
   - B-Tree indexes maintain the sorted order of data for quick retrieval.
   - B-Tree indexes are suitable for columns with high cardinality (many distinct values) and are effective for equality and range queries.

   Bitmap Index:
   - Bitmap indexes use a bitmap representation, where each bit corresponds to a row in the table and represents the presence or absence of a specific value.
   - Bitmap indexes are designed for set-based operations (e.g., OR, AND) and are highly efficient for such operations.
   - They are ideal for columns with low cardinality (few distinct values) and categorical or binary data.
   - Bitmap indexes may not maintain sorted order and are less suitable for range queries.

7. **When we use Function based index?**
   Function-based indexes in Oracle are used to improve query performance when you frequently use expressions or functions in your queries. These indexes are based on the result of applying a function or expression to one or more columns in a table.

Example:
CREATE INDEX idx_upper_name ON emp(UPPER(emp_name));

8. **What is Clustered index?**
   A clustered index determines the physical order of data rows within a table for faster data retrieval. In other words, the data rows in the table are stored on disk in the same order as the index key. Each table can have only one clustered index because the physical order of rows can't be maintained in multiple ways simultaneously.

9. **When would using a clustered index make sense?**
   These types of indexes are useful when index column has multiple related values. If the same column is used in multiple tables acting as entity relationship, then clustered index makes joins fasted when querying.

10. **A table can have multiple non-clustered indexes, but only one clustered index. Why?**
    If a table has clustered index, then its records are sorted based on the column on which clustered index is created. Therefore, if we create multiple clustered indexes, we cannot sort the table record for all those columns. Only one time sorting based one column for which clustering is done is valid.

11. **What is the difference between a Clustered and Non-Clustered Index?**
    - A clustered index determines the order in which the rows of the table will be stored on disk and it actually stores row level data in the leaf nodes of the index itself. A non-clustered index has no effect on which the order of the rows will be stored.
    - A clustered index can be a disadvantage because any time a change is made to a value of an indexed column, the subsequent possibility of re-sorting rows to maintain order is a definite performance hit.
    - A table can have multiple non-clustered indexes. But, a table can have only one clustered index.