**Why Oracle uses B-tree as the default index type?**

Oracle has a lot of index types available but B-tree index is its only one default index type – if you have created an index without specifying anything, then it’s a B-tree index.

Wikipedia just summarizes its advantages and disadvantages:

* 1. They are very fast when you are selecting just a small very subset of the index data (5%-10% max typically)
  2. They work better when you have a lot of distinct indexed values.
  3. Combining several B-Tree indexes can be done, but simpler approaches are often more efficient.
  4. They are not useful when there are few distinct values for the indexed data, or when you want to get a large (>10% typically) subset of the data.
  5. Each B-Tree index imposes a small penalty when inserting/updating values on the indexed table. This can be a problem if you have a lot of indexes in a very busy table.

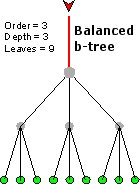
These characteristics make B-Tree indexes very useful for speeding searches in OLTP applications, when you are working with very small data sets at a time, most queries filter by ID, and you want good concurrent performance. I think that is why B – tree become the most common index type and the type that most people mean when they say database index.

Next, I will give more details from concepts to structures to explain the reason why Oracle chooses B-tree as its default index type.

**What is a B – tree?**

A B – tree index is a data structure in the form of a tree. In a tree, records are stored in locations called **leaves**. The starting point is called the **root**. The maximum number of children per node is called the **order** of the tree. The maximum number of access operations required to reach the desired leaf (data stored on the leaf) is called the **depth** (level).

|  |  |
| --- | --- |
| **real tree (in nature)** | **b-tree** |
| grows up | grows down |
| main trunk | Root |
| branch | Node |
| leaf | Leaf |

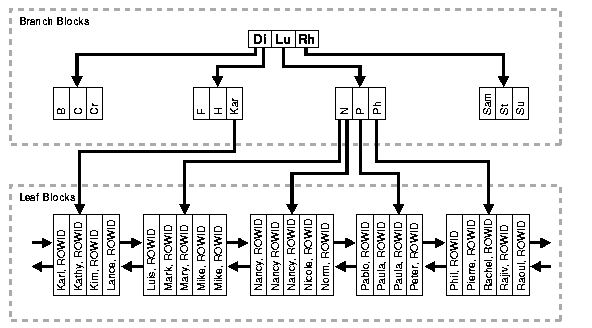


Contrary to popular belief, B is not for binary; it’s balanced. The order is the same at every at every node and the depth is the same for every leaf. The bigger the order, the more leaves and nodes you can put at a certain depth. This means that there are fewer levels to traverse to get to the leaf (which contains the data you want).

**How does a b-tree help with database access?**

Most indexes are too large to fit into memory, which means that they are going to be stored on disk. Since I/O is usually the most expensive thing you can do in a computer system, these indexes need to be stored in an I/O efficient way.

A b-tree is a good way to do this. If we make the nodes the size of a physical I/O block, it would take one I/O to move to a lower depth in the tree. In the example below, an index was created on a first name kind of field.

**[](http://mattfleming.com/node/196)**

If every level were an I/O it would take 3 I/Os to find Mary (or any other leaf). From this point, B-index is a high efficient index type.

Moreover, Oracle offers several options when creating an index using the default b-tree structure. It allows you to index on multiple columns (concatenated indexes) to improve access speeds. Also, it allows for individual columns to be sorted in different orders. For example, we could create a b-tree index on a column called employee\_id and have a second column within the index that displays the salary column.

*CREATE INDEX employee\_salary\_idx ON employees*

*(employee\_id, salary);*

A multicolumn index can be used by the database but only from the first or lead column. Our employee\_salary\_idx index can be used in the following query.

*SELECT first\_name, last\_name, salary, department\_id*

*FROM employees*

*WHERE employee\_id > 120;*

Note that the lead column of the index is the employee\_id, so the database can use the index in the query above. I can also use the employee\_salary\_idx index in the queries below.

*SELECT first\_name, last\_name, salary, department\_id*

*FROM employees*

*WHERE employee\_id > 120*

*AND salary = 9000;*

As you can see, b-tree indexes are very powerful. You must remember that a multicolumn index cannot skip over columns, so the lead index column must be in the WHERE clause filters. Oracle has used b-tree indexes for many years, and they are appropriate from most of your indexing needs. However, the Oracle database provides specialized indexes that can provide additional capabilities; the bit-mapped index and the function-based index.

While b-tree indexes are great for simple queries, they are not very good for the following situations:

* Low-cardinality columns: columns with less than 200 distinct values do not have the selectivity required in order to benefit from standard b-tree index structures.
* No support for SQL functions: B-tree indexes are not able to support SQL queries using Oracle's built-in functions. Oracle9i provides a variety of built-in functions that allow SQL statements to query on a piece of an indexed column or on any one of a number of transformations against the indexed column.