**Legal Notice:** This document contains privileged and/or confidential information and may not be disclosed, distributed or reproduced without the prior written permission of Datamola.

U1M3.Database Types of Tables, Indexes

#### Kiryl Bucha

Contents

[1. Overview: Types of Tables 3](#_Toc43816914)

[2. Overview: Types of Indexes 6](#_Toc43816915)

[3. Source Books and Articles 9](#_Toc43816916)

# 1. Overview: Types of Tables

We will define each type of table before getting into the details. There are nine major types of tables in

Oracle, as follows:

1. **Heap organized tables:** These are normal, standard database tables. Data is managed in a heap-like fashion. As data is added, the first free space found in the segment that can fit the data will be used. As data is removed from the table, it allows space to become available for reuse by subsequent INSERTs and UPDATEs. This is the origin of the name “heap” as it refers to this type of table. A heap is a bunch of space, and it is used in a somewhat random fashion.
2. **Index organized tables:** These tables are stored in an index structure. This imposes physical order on the rows themselves. Whereas in a heap the data is stu ffed wherever it might fit, in index-organized tables (IOTs) the data is stored in sorted order, according to the primary key.
3. **Index clustered tables:** Clusters are groups of one or more tables, physically stored on the same database blocks, with all rows that share a common cluster key value being stored physically near each other. Two goals are achieved in this structure. First, many tables may be stored physically joined together. Normally, you would expect data from only one table to be found on a database block, but with clustered tables, data from many tables may be stored on the same block. Second, all data that contains the same cluster key value, such as DEPTNO = 10, will be physically stored together. The data is clustered around the cluster key value. A cluster key is built using a B\*Tree index.
4. **Hash clustered tables:** These tables are similar to index clustered tables, but instead of using a B\*Tree index to locate the data by cluster key, the hash cluster hashes the key to the cluster to arrive at the database block the data should be on. In a hash cluster, the data is the index (metaphorically speaking). These tables are appropriate for data that is read frequently via an equality comparison on the key.
5. **Sorted hash clustered tables:** This table type is new in Oracle 10g and combines some aspects of a hash-clustered table with those of an IOT. The concept is as follows: you have some key value that rows will be hashed by (say, CUSTOMER\_ID), and then a series of records related to that key that arrive in sorted order (timestamp-based records) and are processed in that sorted order. For example, a customer places orders in your order entry system, and these orders are retrieved and processed in a first in, first out (FIFO) manner. In such a system, a sorted hash cluster may be the right data structure for you.
6. **Nested tables:** These are part of the object-relational extensions to Oracle. They are simply system-generated and maintained child tables in a parent/child relationship. They work much in the same way as EMP and DEPT in the SCOTT schema with the EMP table being the nested table. EMP is considered to be a child of the DEPT table, since the EMP table has a foreign key, DEPTNO, that points to DEPT. The main difference is that they are not stand-alone tables like EMP.
7. **Temporary tables:** These tables store scratch data for the life of a transaction or the life of a session. These tables allocate temporary extents, as needed, from the current user’s temporary tablespace. Each session will see only the extents that session allocates; it will never see any of the data created in any other session.
8. **Object tables:** These tables are created based on an object type. They have special attributes not associated with non-object tables, such as a system-generated REF (object identifier) for each row. Object tables are really special cases of heap, index organized, and temporary tables, and they may include nested tables as part of their structure as well.
9. **External tables:** The data in these tables are not stored in the database itself; rather, they reside outside of the database in ordinary operating system files. External tables in Oracle9i and above give you the ability to query a file residing outside the database as if it were a normal table inside the database. They are most useful as a means of getting data into the database (they are a very powerful data-loading tool). Furthermore, in Oracle 10g, which introduces an external table unload capability, they provide an easy way to move data between Oracle databases without using database links.

**NOTE:** To get more details about each type of table please read next books: Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming; Techniques and Solutions, Second Edition; Thomas Kyte ; 2010 (**Chapter 10: Database Tables**).

**NOTE:** To get experience with common table tasks please continue you work with Lab works: *MTN.NIX.07.Oracle DB.DWH\_labwork03\_Database Types of Tables, Indexes.docx*

# 2. Overview: Types of Indexes

Oracle provides many different types of indexes for us to use. Briefly, they are as follows:

* **B\*Tree indexes:** They are, by far, the most common indexes in use in Oracle and most other databases. Similar in construct to a binary tree, B\*Tree indexes provide fast access, by key, to an individual row or range of rows, normally requiring few reads to find the correct row. It is important to note, however, that the “B” in “B\*Tree” does not stand for binary but rather for balanced.. The B\*Tree index has Several subtypes:
* **Index organized tables:** These are tables stored in a B\*Tree structure. Whereas rows of data in a heap table are stored in an unorganized fashion (data goes wherever there is available space), data in an IOT is stored and sorted by primary key. IOTs behave just like “regular” tables as far as your application is concerned; you use SQL to access them as normal. IOTs are especially useful for information retrieval, spatial, and OLAP applications.
* **B\*Tree cluster indexes:** These are a slight variation of conventional B\*Tree indexes. They are used to index the cluster. Rather than having a key that points to a row, as for a conventional B\*Tree, a B\*Tree cluster has a cluster key that points to the block that contains the rows related to that cluster key.
* **Descending indexes:** Descending indexes allow for data to be sorted from big-to-small (descending) instead of small-to-big (ascending) in the index structure.
* **Reverse key indexes:** These are B\*Tree indexes whereby the bytes in the key are reversed. Reverse key indexes can be used to obtain a more even distribution of index entries throughout an index that is populated with increasing values. For example, if I am using a sequence to generate a primary key, the sequence will generate values like 987500, 987501, 987502, and so on. These sequence values are monotonic, so if I were using a conventional B\*Tree index, they would all tend to go the same right-hand side block, thus increasing contention for that block. With a reverse key index, Oracle will logically index 205789, 105789, 005789, and so on instead. Oracle will reverse the bytes of the data to be stored before placing them in the index, so values that would have been next to each other in the index before the byte reversal will instead be far apart. This reversing of the bytes spreads out the inserts into the index over many blocks.
* **Function-based indexes:** These are B\*Tree or bitmap indexes that store the computed result of a function on a row’s column(s), not the column data itself. You can consider them an index on a virtual (or derived) column—in other words, a column that is not physically stored in the table. These may be used to speed up queries of the form SELECT \* FROM T WHERE FUNCTION(DATABASE\_COLUMN) = SOME\_VALUE, since the value FUNCTION(DATABASE\_COLUMN) has already been computed and stored in the index.
* **Bitmap indexes:** Normally in a B\*Tree, there is a one-to-one relationship between an index entry and a row: an index entry points to a row. With bitmap indexes, a single index entry uses a bitmap to point to many rows simultaneously. They are appropriate for highly repetitive data (data with few distinct values relative to the total number of rows in the table) that is mostly read-only. Consider a column that takes on three possible values—Y, N, and NULL—in a table of 1 million rows. This might be a good candidate for a bitmap index if, for example, you need to frequently count how many rows has a value of Y. That is not to say that a bitmap index on a column with 1,000 distinct values in that same table would not be valid—it certainly can be. Bitmap indexes should never be considered in an OLTP database for concurrency-related issues.
* **Bitmap join indexes:** These provide a means of denormalizing data in an index structure, instead of in a table. For example, consider the simple EMP and DEPT tables. Someone might ask the question, “How many people work in departments located in the city of Boston?” EMP has a foreign key to DEPT, and in order to count the employees in departments with a LOC value of Boston, we would normally have to join the tables to get the LOC column joined to the EMP records to answer this question. Using a bitmap join index, we can instead index the LOC column against the EMP table. The same caveat in regard to OLTP systems applies to a bitmap join index as a regular bitmap index.
* **Application domain indexes:** These are indexes you build and store yourself, either in Oracle or perhaps even outside of Oracle. You tell the optimizer how selective your index is and how costly it is to execute, and the optimizer will decide whether or not to use your index based on that information. The Oracle text index is an example of an application domain index; it is built using the same tools you may use to build your own index. It should be noted that the index created here need not use a traditional index structure. The Oracle text index, for example, uses a set of tables to implement its concept of an index.

**NOTE:** To get more details about each type of table please read next books: Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming; Techniques and Solutions, Second Edition; Thomas Kyte ; 2010 (**Chapter 11: Indexes)**.

# 3. Source Books and Articles

1. Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming; Techniques and Solutions, Second Edition; Thomas Kyte ; 2010;
2. Pro Oracle SQL; Karen Morton, Kerry Osborne, Robyn Sands, Riyaj Shamsudeen, Jared Still ; 2010;