



Index Organized Tables: When should they be used?

By Alon Peled

Since the early versions of Oracle, table data organization was done in one way: heap. All rows were piled into the data blocks with no order or connection between them. In order to escalate retrieval, time DBAs were able to add indexed as required.

Queries that search a specific row or range or rows had to go through the following steps:

- A. Read the index segment.
- B. Search the correct key through the B*tree index tree.
- C. Get the relevant Rowids
- D. Access the table's correct data block using Rowids.

Figure 1 shows the index and heap organization.

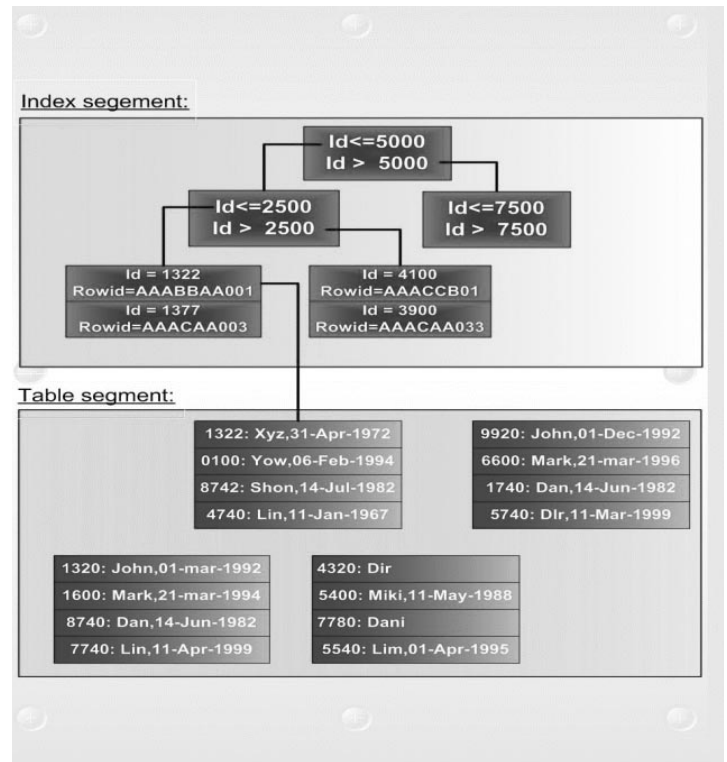


Figure 1: Index and Heap organization

Drawbacks of index usage

There are two drawbacks associated with index usage. First, index creation uses additional storage space. The index segment has to store the key value and the rowid that would point to the correct data block.

The second drawback involves performance. Each DML operation (Insert, Delete or Update of the key column) or a query using the index has to perform the following two tasks:

- Read the index segment into memory and then get the data block from the disk.
- Update the table block and the index block (this does not apply to query operations).

Index Organized Tables (IOT)

In order to solve some of the above-mentioned issues, Oracle has developed a new way to organize tables.

The data will be stored in the table in the same way as it would be stored in an index. The table behaves like an ordinary table with an additional index, except that there is no need for an external index to speed up row search.

Disclaimer: The following information reflects a personal opinion and should not be used on production systems without adequate testing. The author takes no responsibility for any damages that may occur while implementing any of the features described in this document.

continued on page 34



Indexed Organized Tables: When Should They Be Used? *continued from page 33*

The rows are already in the correct order and interlinked. Theoretically, the IOT needs less storage space and less physical I/O for exact or range searches. DBAs who want to speed up performance have the ability to direct some of the non-key columns to another tablespace. By doing so, the number of blocks storing the key values can be reduced.

Figure 2 shows the IOT organization method.

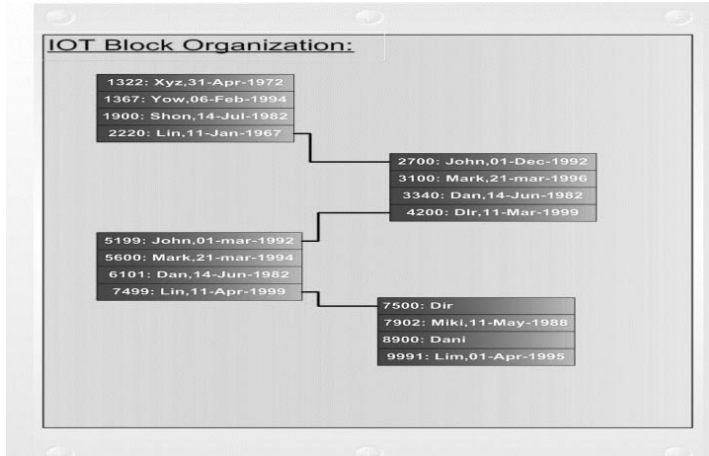


Figure 2: IOT Organization

Feature Comparison

Table 1 summarizes the comparison between Heap and IOT features.

Feature	Heap Table	Index organized table
Unique row identification	Rowid	Primary key
Row duplication	Allowed	A primary key is a must
Can be part of a cluster	Yes	No
Contain LONG and LOB datatypes	LONG and LOB	LOB only
Manipulation by applications	Normal	Normal, Same as Heap
Distributed SQL	Allowed	Only from Oracle9i
Replication	Allowed	Only from Oracle9i
Partitioning	Allowed	Only from Oracle8i

Table 1: Heap and IOT feature comparison

Testing Heap and IOT Table Organization: Which one is faster?

A series of tests were performed to compare performance using different scenarios.

Test description

Several tests were made with more than 500,000 rows inserted into both table types. These rows were then retrieved, updated and deleted more than 10,000 times. Table sizes varied from 10M to over 100M.

Tests were done on short and long rows ranging from 18 to 170 bytes as the average row length. Primary and secondary indexes were created as well.

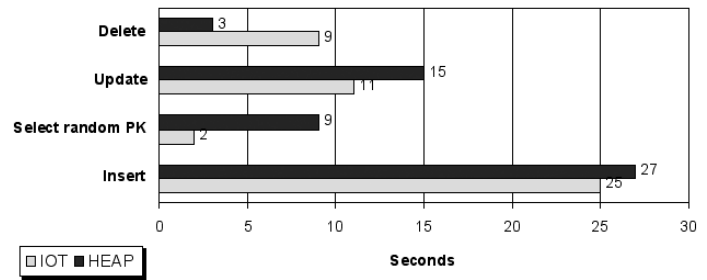
Criteria that were tested

The following factors were tested:

- A. Data Load (via insert)
 - B. Row manipulation – Update and delete
 - C. Exact fetch via primary index
 - D. Range scan via secondary index
- All of the above were tested on various row lengths.

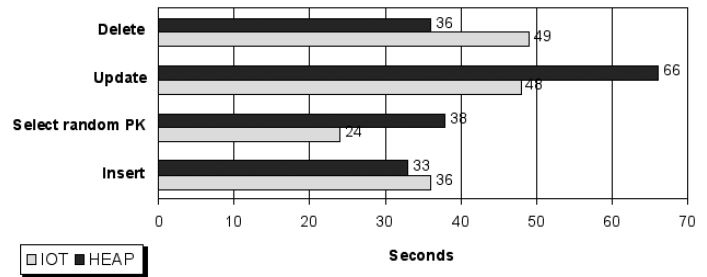
Test results

In the case of a short column length, the IOT query time is more than 400% faster. This is the classic case where IOT would bring the most benefits.



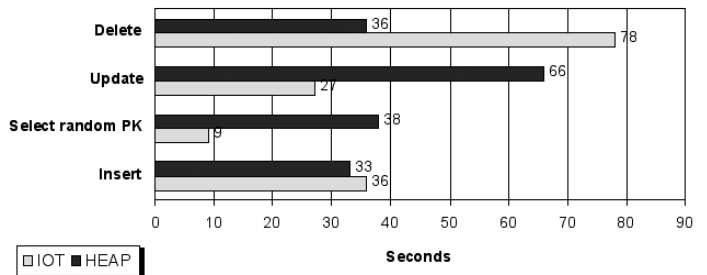
A. Table with two columns + only primary key

When the table has more than two or three columns, the IOT advantage is small and the focus should be on the query time, which is about 40% faster than HEAP tables.



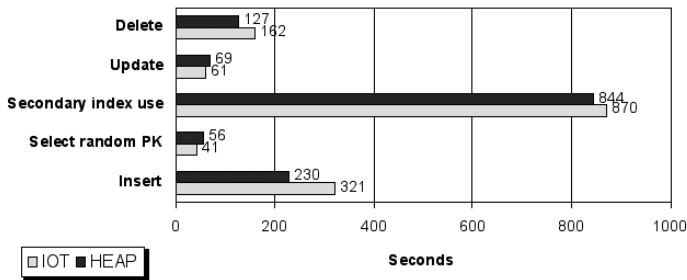
B. Table with many columns + single index as primary key

Using the PCTTHRESHOLD, the IOT advantage in the query criteria increases dramatically to be 400% faster than HEAP tables.



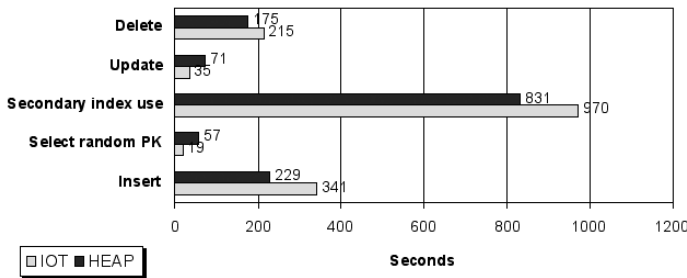
C. Table with Many columns + Primary key + use of PCTTHRESHOLD

The IOT query time, using the secondary index has no advantage over heap secondary index.



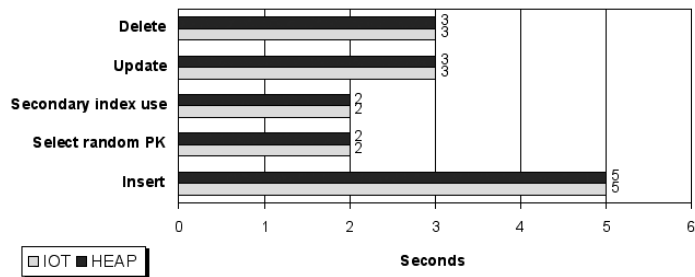
D. Table with Many columns + Primary key + secondary index

When the IOT table has the PCTTHRESHOLD option, even with a secondary index, the results change dramatically. Although secondary index queries are 15% slower, IOT PK is 400% faster.



E. Row length – long, Primary key + secondary index + use PCTTHRESHOLD

This test may be the most important one: If your tables are small, the organization method is irrelevant. Both options produce the same results.



F. Same as E above but with only 5000 rows

Conclusions

IOT tables may significantly speed up performance in certain types of applications. Best suited for this type are large tables where primary key exact fetch is time critical and there is a low volume of DML operations. In most cases, in the category of exact fetch via primary key, IOT organization is much faster than HEAP. This advantage varies between 15% and 400%, depending upon the row length, number of columns and the use of the PCTTHRESHOLD option.

If your application is more transaction intensive, then more tests would be in order. The same goes for secondary index usage.

In the case of tables with a small number of rows (less than 5,000) - I have found no apparent performance difference.

As with any other option encountered during application development, both organization methods should be tested in their true environments. My conclusions should never be implemented without thorough testing and you should never assume that someone else knows more about your systems than you do.

Syntax

```
CREATE TABLE table_name
(column_spec... PRIMARY KEY,
 column_spec...,
 column_spec... )
ORGANIZATION INDEX
TABLESPACE tablespace_name
PCTTHRESHOLD percentage_no INCLUDING column_name
OVERFLOW TABLESPACE another_tablespace_name
```

You must specify a primary key for an index-organized table, because the primary key uniquely identifies a row. Use the primary key instead of the Rowid for directly accessing index-organized rows.

PCTTHRESHOLD integer

Specifies the percentage of space reserved in the index block for an index-organized table row. Any portion of the row that exceeds the specified threshold is stored in the overflow segment. PCTTHRESHOLD must be a value from 1 to 50.

Restrictions:

- PCTTHRESHOLD must be large enough to hold the primary key.
- You cannot specify PCTTHRESHOLD for individual partitions of an index-organized table.

OVERFLOW

Specifies that index-organized table data rows exceeding the specified threshold be placed in the data segment listed in this clause.

- When you create an IOT, Oracle evaluates the maximum size of each column to estimate the largest possible row. If an overflow segment is needed but you have not specified OVERFLOW, Oracle raises an error and does not execute the CREATE TABLE statement. This checking function guarantees that subsequent DML operations on the index-organized table will not fail because an overflow segment is lacking.
- All physical attributes and storage characteristics you specify in this clause after the OVERFLOW keyword apply only to the overflow segment of the table. Physical attributes and storage characteristics for the index-organized table itself, default values for all its partitions, and values for individual partitions must be specified before this keyword.
- If the index-organized table contains one or more LOB columns, the LOBs will be stored out-of-line unless you specify OVERFLOW, even if they would otherwise be small enough to be stored inline.

continued on page 36

INCLUDING *column_name*

Specifies a column at which to divide an index-organized table row into index and overflow portions. All non-primary-key columns that follow *column_name* are stored in the overflow data segment. A *column_name* is either the name of the last primary-key column or any subsequent nonprimary-key column.

Restriction: You cannot specify this clause for individual partitions of an index-organized table.

Compression_clause enables or disables key compression. This is useful when using more than one column for the primary key.

COMPRESS - Enables key compression, which eliminates repeated occurrence of key column values in index-organized tables. Use *integer* to specify the prefix length (number of prefix columns to compress).

The valid range of prefix length values is from 1 to the number of primary key columns minus 1. The default prefix length is the number of primary key columns minus 1.

Restriction: At the partition level, you can specify COMPRESS, but you cannot specify the prefix length with *integer*.

NOCOMPRESS - disables key compression in index-organized tables. This is the default.

Example:

```
Create table emp_details (
  id          number          not null constraint emp_det_pk
  primary key,
  Name        varchar2 (50)   ,
  Address     varchar2(250)   ,
  Phone       varchar2 (50)   ,
  Sum1        number         ,
  Sum2        number         ,
  Date1       date           ,
  Date2       date           ,
  Qty1        number         ,
  Qty2        number         ,
  Text        char (50)
)
ORGANIZATION INDEX
PCTTHRESHOLD 2 OVERFLOW TABLESPACE ts_data;
Create index emp_details_i2 on emp_details (name) tablespace ts_index;
```

References and Acknowledgments

Oracle Technology Network: www.otn.com

Special thanks to Mr. Craig A. Shallahamer, President of OraPub Inc (www.orapub.com).

Another thank you to Mr. Geerlings Mark, chief DBA at Gentex.



About the Author

Alon Peled's 14 years of experience in the IT marketplace brings a distinct balance of creativity to any production site. As a senior consultant, Alon's specializations include designing, tuning and teaching about Oracle based systems. Peled also teaches DBA courses in "Michelet Hi-Tech." More details or white papers can be found at his Web site: www.peled.com