

Oracle9i Index-Organized Tables Technical Whitepaper

An Oracle White Paper
September 2001

Oracle9i Index-Organized Tables Features Overview

EXECUTIVE OVERVIEW	3
INTRODUCTION.....	3
WHAT IS AN INDEX-ORGANIZED TABLE?	4
Flexible Storage and Column Placement Options	5
Access Alternatives for Performance.....	5
THINK IOTS FOR AVAILABILITY	6
SCALABLE PARTITIONING AND PARALLEL OPERATIONS.....	6
ADDITIONAL FEATURES OF IOTS	7
Key Compression.....	7
LOB Columns.....	8
Object Support	8
Distributed Database and Replication Support.....	8
Export/Import	8
SQL*Loader.....	8
APPLICATIONS SUITABLE FOR IOTS	8
CONCLUSION.....	9

Oracle9i Index-Organized Tables Features Overview

EXECUTIVE OVERVIEW

Organizations have always demanded rapid access to data that is always available for unlimited use to support applications necessary for daily business operations. Now that demand is fueled by the growth of internet, e-commerce, and B2B as these newer web-based applications take on increasing importance to the business landscape.

Since many web applications (as well as traditional ones) involve data with access based on single column primary keys, Oracle has implemented a unique storage organization, index-organized tables, to increase performance, scalability and availability for this type of data. Oracle9i extends this feature offering even greater benefits in each of these areas.

INTRODUCTION

Storage organization within the database is one of the most critical factors for data access performance. Traditionally, most data has been stored in heap-organized tables. Rows in heap-organized tables are stored in the order that they are inserted, then the user creates an index on the primary key to enable rapid access. Although heap-organized tables have advantages, they also have a cost – access must be duplicated against the index and the data itself. Furthermore, actual storage requirements increase as the key columns are duplicated from the table to the index.

An index-organized table (IOT) is an alternative to the heap organization that was introduced in Oracle8. IOTs are also unique to the Oracle server and provide substantial performance, availability and scalability benefits over conventional tables. An IOT is a variant of a B-tree index that allows the table data itself to be

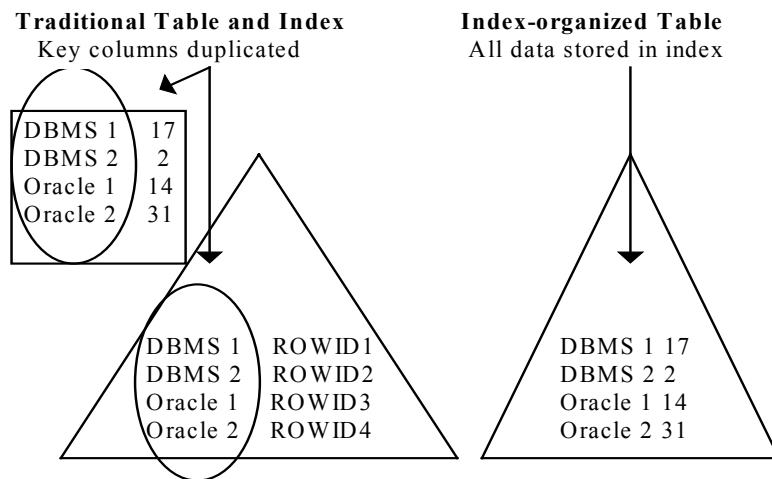
kept in its associated primary key index, with the rows placed in a key-sequenced order, thus eliminating duplication between the table and the index. This approach enables extremely fast access to table data for primary key based queries, making IOTs an excellent choice for a wide variety of applications.

WHAT IS AN INDEX-ORGANIZED TABLE?

An IOT has its own structure, storage, and indexing model that is quite different from a conventional table. In a conventional table, the original physical address or ROWID is stable and this physical ROWID does not change as long as the row exists. When creating an index on a column in a conventional table, the index stores the column data as well as the ROWID as the provides the actual physical location.

IOTs are not tied to stable physical locations. Data for an IOT is stored in sorted order in the leaves of a b-tree index created by the table's primary key. Rows may move around to keep the sorted order, For example an INSERT operation can cause an index leaf to split and the existing row may be moved to a different slot or even block. Simply put, changes to the table data result in an update to the index only as the B-tree index holds the actual row data with the primary key.

Since all the data is stored in the B-tree structure, IOTs provide a fast, primary key-based access to table data for queries involving exact match and/or range search. Once the search has located the key values, it also locates the data so there is no need to trace a physical ROWID back to table data, thus an I/O operation, namely the read of the table (illustrated below). In addition, the storage requirements are reduced as key columns are not duplicated in the table and the index and there is no additional storage needed for physical ROWIDs.



Flexible Storage and Column Placement Options

B-tree index entries are usually small since they only consist of a key and a ROWID. But B-tree index entries in an IOT can be very large since they contain a key and non-key column values. A large index entry means that the leaf node may store one row or row-piece which destroys the dense clustering property of the B-tree index.

The use of an overflow segment with an IOT handles this problem. With the OVERFLOW option users can specify physical storage attributes such as a tablespace. So in an IOT with an overflow segment, the index row consists of a key and a row head which contains the first few non-key columns and a ROWID. And in this case, the ROWID points to an overflow area where the remaining column values are located.

The threshold value, specifies as a percentage of the leaf block size, determines the last non-key column included in the index row head piece. If the row size is greater than the specified threshold value, then the non-key column values for the row that exceeds the threshold are stored in the specified overflow tablespace. Vertical partitioning of a row between the index and data segments allows for higher clustering of rows in the index, which results in better query performance for the columns stored in the index.

Setting a threshold value does not allow for the same set of columns to be included in the index for all rows in the table. However, there is an INCLUDING option provided for this purpose. This option ensures that all columns after the specified including column are stored in the overflow segment. If the specified including column has a corresponding index row size that exceeds the specified threshold, then the last non-key column to be included is determined according to the threshold value.

Access Alternatives for Performance

Secondary indexes on IOTs grant efficient access to the IOT via columns that are not primary key-based. Secondary indexes are constructed on IOTs using logical ROWIDs based on the table's primary key. The logical ROWID of a row does not change as long as the primary key value does not change.

A logical ROWID differs from a physical one because it has no permanent physical address and can move across data blocks when new rows are inserted. Furthermore, if the physical location of a row changes, its logical ROWID remains valid giving advantages for data availability during maintenance operations. Additionally, the access through the logical ROWID is the fastest possible way to get to a specific row, even if it takes more than one block access to get it.

A logical ROWID includes the table's primary key and a guess which identifies the block location of a row at the time the guess is made. These guesses can be used to

probe directly into the leaf block of the index-organized table, bypassing the primary key search. Because rows in an IOT. The value of the guess is that it makes ROWID access to non-volatile IOTs comparable to access of conventional tables.

Oracle9i also supports bitmap indexes on IOTS. In a bitmap index, a bitmap for each key value is used instead of a list of ROWIDs. Each bit in the bitmap corresponds to a possible ROWID. If the bit is set, then it means that the row with the corresponding ROWID contains the key value. If the number of different key values is small, then bitmap indexes are very space efficient. Bitmap indexing efficiently merges indexes that correspond to several conditions in a WHERE clause found in applications involving ad hoc queries over large amounts of data. Rows that satisfy some, but not all, conditions are filtered out before the table itself is accessed. This improves response time, often dramatically.

A bitmap index is accessed using a search key and when the key is found, the bitmap entry is converted to a physical ROWID. For an IOT, the physical ROWID accesses a mapping table containing the logical ROWIDs. The logical ROWID is then used to access the IOT. A bitmap index is logical in nature and so movement of the rows in an IOT does not invalidate its bitmap indexes. Some of the guesses in the mapping table's logical ROWID entries are invalid but the IOT can still be accessed by the primary key.

Oracle9i also supports indexes on UROWID datatypes of an IOT. A single datatype called the universal row identifier, or UROWID, supports both logical and physical ROWIDs. A column of the UROWID datatype can store all kinds of ROWIDs including the logical primary key-based ROWIDs identifying rows of IOTs.

THINK IOTS FOR AVAILABILITY

Online reorganization of database objects such as tables and indexes is of prime importance as business applications today focus on high availability requirements. An IOT can be rebuilt online to reduce fragmentation from incremental updates while keeping downtime to a minimum. The COALESCE operation on the underlying primary key B-tree merges leaf blocks within the same branch by locking a few blocks at a time, then quickly frees them once the operation is completed. The ALTER TABLE...MOVE ONLINE method is also supported which allows DML and query operations to proceed during the actual move operations. These options make IOTS very suitable for high availability applications.

SCALABLE PARTIONING AND PARALLEL OPERATIONS

Queries on IOTs with a primary key index scan can be executed in parallel. The following parallel scan methods are supported:

- Parallel fast full scan of a nonpartitioned IOT.

- Parallel fast full scan of a partitioned IOT.
- Parallel index range scan of a partitioned IOT.

These scan methods can also be used for IOTs with overflow areas and for IOTs containing LOBs.

An IOT can be partitioned by RANGE or HASH on column values. The partitioning columns must form a subset of the primary key columns. Local partitioned (prefixed and non-prefixed) index as well as global partitioned (prefixed) indexes are supported for partitioned IOTs the same as it is for conventional tables.

Partitioning was designed to split large tables into a smaller set of partitions for easier management. As a result, there are tables now with thousands of partitions. When a table consists of many partitions, time needed to parse a SQL statement increases – quite dramatically in some cases -- because the parsing process retrieves information about each partition. In Oracle 9i, the parsing algorithms on partitioned tables, including IOTs, have been redesigned to correct this.

ADDITIONAL FEATURES OF IOTS

Even though an IOT has its own index structure, it can be accessed via SQL just as conventional tables are. However, the database system performs all operations by manipulating the corresponding B-tree index. Applications use standard SQL statements for query, insert, update, or delete operations on an IOT. SELECT FOR UPDATE statements can also be used to lock rows of an IOT. In addition, other Oracle programmatic interfaces such as PL/SQL, OCI, and JDBC can be used to access an IOT as well as a conventional table. This consistency eliminates the need to learn separate, different interfaces, thus saving time when choosing to implement IOTs.

IOTs are fully functional and support the standard features available with conventional tables. These include secondary indexes, constraints, triggers, composite columns such as nested tables and VARRAYSs, Object and REF columns, and LOB columns. Key compression is supported on secondary indexes and the index-organized table itself, because the IOT is stored in an index.

Key Compression

Key compression eliminates repeated occurrences of key column prefixes in IOTs and indexes. Key compression breaks an index key into a prefix entry and suffix entry. Compression is achieved by sharing the prefix entries among all the suffix entries in an index block. Only keys in the leaf blocks of a B-tree are compressed.

When an IOT contains a VARRAY and/or a NESTED TABLE, the storage overhead of 16 bytes per element for the object ID (OID) is unnecessary, because OIDs are repeated for all elements in a collection. Key compression allows the compression of the repeating OID values in the leaf blocks of an IOT.

LOB Columns

Internal and external LOB columns can be created in IOTs to store large unstructured data such as audio, video, and images. The SQL DDL, DML, and piece-wise operations on LOBs in an IOT have the same behavior as in conventional tables. By default, the LOB's data and index segments are created in the tablespace in which the primary key index segment of the IOT is created. LOBs in IOTs with overflow segments have the same characteristics as those in conventional tables but those without an overflow segment are stored out-of-line.

Other LOB features--such as BFILEs, temporary LOBs, and varying character width LOBs--are also supported in IOTs and are used in the same fashion as conventional tables.

Object Support

Most of the object features are supported on IOTs, including Object Type, VARRAYs, Nested Table, and REF Columns. The one exception is object tables which cannot be created as IOTs.

Distributed Database and Replication Support

Both non-partitioned and partitioned IOTs can be replicated.

Export/Import

Export/Import supports export (both ordinary and direct path) and import of non-partitioned and partitioned index-organized tables.

SQL*Loader

SQL*Loader supports both ordinary and direct path load of IOTs and their associated indexes (including partitioning support). Direct path parallel load to an IOT is not supported but the same result can be obtained by doing a parallel load to an ordinary table using SQL*Loader, then using the parallel CREATE TABLE AS SELECT option to build the IOT.

APPLICATIONS SUITABLE FOR IOTS

In general, applications the major category of very large database (VLDB) and online transaction processing (OLTP) application benefit greatly from IOTs. The logical nature of an IOT's secondary indexes as global and local indexes remain usable after an ALTER TABLE MOVE operation which is not the case with conventional tables. Additionally, this operation can be done online satisfying 24x7 availability requirements common to VLDB and OLTP applications.

All types of information retrieval applications derive benefits from using IOTS. These applications support content-based searches on document collection by maintaining an inverted index containing tokens.. The application performs a content-based search by scanning the inverted index looking for tokens of interest. By using an IOT to model the inverted index, only the non-key column values are

stored with the key in the index so that there is no duplication of data. Also, storage requirements are reduced because the additional ROWID required by an index is maintained on an ordinary table is avoided.

On-line analytical processing (OLAP) applications typically manipulate multi-dimensional blocks which, for performance reasons, maintain an inverted index to map a set of dimension values to a set of pages. As described above, the inverted index can easily be modeled as an IOT.

Ongoing efforts to increase performance for data warehousing applications is universal. IOTS adds performance advantages when used to implement fact tables common to data warehousing applications because the fact table lookup will result in an index only scan for an IOT. Support for materialized views and bitmapped secondary indexes added in Oracle9i further enhance the benefits of IOT for these applications.

Internet and web applications also gain advantages from IOTs. Electronic order processing shares characteristics of similar to OLTP applications described earlier, and thus can derive availability and performance benefits as well. Using an IOT as the storage structure for the orders table is where the query and DML is predominantly primary key based is an excellent choice.

A related category is electronic catalogs for both manufacturing and retail purposes. Manufacturing catalogs are usually indexed by product attributes based on primary key and a retailers catalog may have a multi-column primary key matching the hierarchy of products offered. Both types benefit from using IOTs as key compression can be used to avoid column value repetitions, thus increasing performance and reducing storage.

Internet search applications maintain lists of keywords, users, or URLs, suitable for storage in an IOT, where each row holds a primary key with some additional information. An IOT storing URLs and their associated links can considerably speed up access time.

A prevailing feature of web portals and auction sites is databases of users names with a subset of this available user information accessed more frequently than the rest. The flexible column placement within an IOT provides options for increasing performance of these applications.

CONCLUSION

Unlike conventional tables, an IOT contains ALL the data -- both the indexed and additional columns -- stored in the primary key index. This means that once a search has found the key values, it also finds the data located there, thus eliminating the any additional search for the data itself via a ROWID reference back to a table. This enables faster access to table data for queries involving exact match and/or range search on a primary key. Because key columns are not duplicated in both the table and the index, the use of IOTs results in reduced storage requirements.

IOTs also provide advantages for high availability applications. An IOT, or one of its partitions, can be reorganized without rebuilding its indexes, which results in a shorter reorganization maintenance window. Also, an IOT and its secondary indexes can be reorganized online thus eliminating the reorganization maintenance window. In short, Oracle9i IOTs is an important contributor to the continuous availability and tremendous performance and scalability of the Oracle9i server.



White Paper Title

September 2001

Author: Shirley Ann Stern

Contributing Authors:

Oracle Corporation

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

www.oracle.com

**Oracle Corporation provides the software
that powers the internet.**

**Oracle is a registered trademark of Oracle Corporation. Various
product and service names referenced herein may be trademarks
of Oracle Corporation. All other product and service names
mentioned may be trademarks of their respective owners.**

Copyright © 2000 Oracle Corporation

All rights reserved.