Training Resources

Oracle8i Indexing Choices: Best of Breed

John Jay King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627 USA
www.kingtraining.com
800.252.0652 or 303.798.5727

Objectives

- Learn about the many Oracle8i indexing features
- Be aware of the differences between index types available in Oracle8i:
  - B-Tree indexes
  - Reverse-key indexes
  - Bit-map indexes
  - Hash indexes
  - Index-Organized Tables
- Understand the performance implications of choosing one index type over another
- Be familiar with Oracle8i's ability to use indexes when functions or expressions are used

Types of Indexes

- B-Tree (traditional) indexes
- Hash-cluster indexes
- Bitmap indexes
- Index-Organized Tables
- Reverse-Key indexes
- Both B-Tree and Bitmap indexes allow Function/Expression-based indexes

Purpose of Indexes

- Except for Index-Organized Tables, indexes translate a key value into a rowid
- Indexes reduce cost of obtaining rows to the I/O or calculations necessary to find the rowid, followed by direct access using the rowid
- This is often faster than reading all possible rows looking for a match (table scan

Indexing Methods, 1

- B-Tree indexes store key values sequentially and are traversed from: root block, to branch block (sometimes multiple levels of branch blocks), to leaf block, to data block containing the row
- Hash-cluster indexes convert the key value using an algorithm to determine which data block to read

Indexing Methods, 2

- Bitmap indexes contain a bit (0 or 1) for each key value that corresponds to every rowid in the table
- Index-Organized Tables actually contain the table row data; once an index value is found in the index, the data is immediately available

B-Tree Indexes

- The original type of index supported by Oracle is the B-tree index providing a linked-list type access using key and the rowid(s)
- A B-tree index is stored in a hierarchy of pages:
  - The first index page is called the "root"
  - The "root" points to lower-level "branch" (non-leaf) pages
  - Multiple levels of "branch" (non-leaf) pages might exist
  - The lowest level index page is called a "leaf" page containing addresses of data blocks
- The actual key value is stored (unless using key compression)
- Concatenated keys can increase the number of index levels necessary, so, some organizations create keys
- Indexes require overhead when updating, deleting, or inserting rows

B-Tree Index Example

		Root Block	<Clark	idxblkid		
			Clark	idxblkid		
			James	idxblkid		
Branch Blocks	<Brown	idxblkid	Clark	idxblkid	James	idxblkid
	Brown	idxblkid	Deng	idxblkid	King	idxblkid
	Charles	idxblkid	Hotke	idxblkid	Samo	idxblkid
Leaf Blocks	Clark	rowid	Deng	rowid	Hotke	rowid
	Craig	rowid	Feingold	rowid	Izzard	rowid
	Davis	rowid	Garbo	rowid	Jabbo	rowid
Data Blocks	Jabbo		Clark		Deng	
	Davis		Izzard		Smith	
	Garbo		Feingold		Hotke	

Key Design

- Key design can cause dense key ranges, especially when key values are geographic
- Index paths more densely or sparsely populated others can lead to deeper levels of index than would otherwise be needed
- Index “trees” that are unbalanced may inhibit performance
  - Rebuilding the index will redistribute the levels
  - Reverse-Key indexes offer another solution

Reverse Key Indexes

- If keys and usage are heavily clustered in a table, Reverse Key Indexes might speed things up
- Oracle says savings will likely be restricted to Parallel-processing environments, most who have tried it find it causes problems in non-Parallel environments
- A Reverse Key Index is simply a standard index with the key values stored in reversed form (e.g. ‘1234’ becomes ‘4321’, ‘1235’ becomes ‘5321’), table data is not changed
- By reversing key values index blocks might be more evenly distributed reducing the likelihood of densely or sparsely populated index paths
- **Carefully** test Reverse-Key indexes to verify benefits, here’s a direct quote from the Oracle8i Concepts manual: “Under some circumstances using a reverse-key index can make an OLTP Oracle Parallel Server application faster.”

Reverse Key Index: Syntax

- Syntax to create a Reverse-Key index is to simply add the word REVERSE after the column specification
- Use of Reverse-Key indexes eliminates the possibility of index range-scan processing (sequential key values are no longer stored sequentially in the index)

```
CREATE INDEX employee_ssn_rev
          ON employee_table (ssn) REVERSE
          /* … rest of index definition … */;
```

Function/Expression-based Indexes (Oracle8i only)

- Until Oracle8i index columns always represented the actual value of the column and a WHERE clause needed to specify the original (unadulterated) column value to use the index
- Now, an index can represent a column after some function or expression has been applied
- If a WHERE clause uses a column value with a function or expression exactly as specified (case-insensitive and blanks are ignored) during index creation, an index may be used

Creating/Using Function/Expression Based Index

- Creating the index:

```
 CREATE INDEX … ON EMP (UPPER(ENAME)) …
 CREATE INDEX … ON INVENTORY (IN_STOCK + ON_ORDER) …
```

- Using the index (if the optimizer agrees…):

```
SELECT … FROM EMP
      WHERE UPPER(ENAME) = UPPER(:hostvar) …
SELECT … FROM INVENTORY
      WHERE IN_STOCK + ON_ORDER > :LARGE_QTY_ITEMS …
```

- These indexes require:
  - Cost-based optimization
  - Statistics on the function/expression-based index
  - Alter session set query_rewrite_enabled = true
  - Alter session set query_rewrite_integrity = trusted (when enabling user-defined functions)

Bitmap Indexes

- For B-Tree indexes columns with few values over many rows (low-cardinality) should be avoided
- Country Code of a customer or Gender of a customer would make poor index columns due to the small number of different values in the table
- Bitmap indexes offer performance improvement for columns with relatively few values
- Bitmap indexes contain the key value and a bitmap listing the value of 0 or 1 (yes/no) for each row indicating whether the row contains that key value of not

Bitmap Example

- For an index of customers in a particular country (limited in example to: United States (US), United Kingdom (UK), Japan (JA), and Australia (AU)):

COUNTRY_CD=AU	COUNTRY_CD=JA	COUNTRY_CD=UK	COUNTRY_CD=US
0	0	0	1
1	0	0	0
0	1	0	0
0	0	1	0
0	0	1	0
0	0	0	1

- Each entry in a bitmap corresponds to a row, a value of 1 indicates which value that row contains
- Bitmaps include all rows, even those with NULL values (unlike B-Tree indexes)
- Bitmaps are usually smaller than B-Tree indexes

Bitmap Issues

- Bitmap indexes work best for equality-type tests (= or IN)
- Bitmap indexes are best when used with other indexes
- This query improves given bitmap indexes on columns COUNTRY_CODE, GENDER, and CREDIT_CARD

```
SELECT CUSTOMER_ID, LAST_NAME, BALANCE
      FROM NON_GOV_CUSTOMERS
      WHERE COUNTRY_CODE IN ('AU','UK')
            AND GENDER = 'M'
            AND CREDIT_CARD = 'AX';
```

- Bitmap index maintenance can be expensive; an individual bit may not be locked, a single update may lock large portions of the index
- Bitmap indexes are best in read-only situations like data warehouses or where concurrent transactions are unlikely

Hash-Cluster Indexing

- B-Tree and Bitmap index keys are used to find rows requiring I/O to process the index
- Hash clusters get rows with a key-based algorithm
- Rows are stored together based upon hash value
- Oracle or user hashing algorithms may be used
- Index size should be known at index creation, it should allow for distribution of rows with few (no) collisions when hashing a specific key
  - Keys with unique hash values are optimal
  - Keys with the same hash value (a collision) may cause chaining, reducing the benefit of hashing

Hash Index Issues

- Hash clusters ***can be*** the fastest access if:
  - Very-high-cardinality columns are used
  - Only equal (=) tests are used
  - Index values do not change
  - Number of rows and rows/index values are known and specified via HASHKEYS at cluster creation time
  - Only minimal insert/delete activity will occur
  - Key values hash well
- Only one Hash-cluster is allowed per table
- To reorganize a Hash-cluster, the index cluster must be dropped and recreated
- **Carefully** test Hash-clusters to verify benefits...

Index-Organized Tables

- CREATE TABLE’s ORGANIZATION INDEX clause causes table data to be incorporated into a B-tree index using the table’s Primary Key
- Table data is always in order by Primary Key and many sorts can be avoided by the optimizer
- Oracle8i adds the ability to create secondary indexes for Index-Organized tables
- Especially useful for “lookup” type tables
- Sequential scan of an index-organized table yields all values in sequence by key

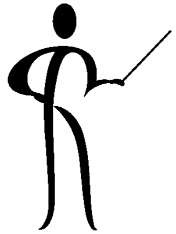
Index-Organized Table Terms

- To differentiate from Index-Organized Tables, use “heap organized” to describe traditional tables
- The entire Index-Organized Table is stored in the index and has no specific rowid, so, Oracle8i uses a “virtual rowid” to provide secondary indexing capability
- Secondary indexes using “virtual rowid” are quicker than a scan of the Index-Organized Table, but, not quite as fast as a traditional B-Tree secondary index

Index-Organized Table Issues

- Index-Organized Tables work best when:
  - There are few columns in the table/index other than the key (a “narrow” table)
  - Size of a row is small compared to the size of a block
- Index-Organized columns may not contain LONG columns, but may contain BLOB, CLOB, or BFILE data (will probably use index overflow area mitigating much of the advantage of IOTs)
- Index-Organized Tables may not be used in a CLUSTER

Comparing Index Strengths and Weaknesses, 1



- For high-cardinality key values, B-tree indexes are usually best and Hash-clusters might be:
  - B-Tree indexes work with all types of comparisons and gracefully shrink and grow as table data changes
  - Hash-clusters work only with equal tests and table growth is a significant problem
- For low-cardinality key values that are not changed by concurrent transactions, Bitmap indexes are often superior to B-tree indexes
- Hash-clusters are not a good choice for low-cardinality data (many collisions)

Comparing Index Strengths and Weaknesses, 2

- If a key design causes dense or sparse population of index values in a Parallel-processing environment, test using Reverse-key indexes to see if overall performance is improved
- Index-Organized tables are a good choice if:
  - Tables have few non-key columns
  - Tables have relatively small rows
  - Results are frequently sorted by Primary Key

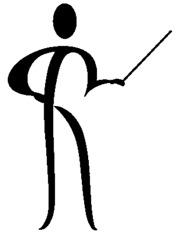
Using Hints to Suggest Indexes

- Sometimes, the optimizer may not choose to use an index, or might not choose to use it as desired
- Use Hints to control the processing of the SQL by “improving” the optimizer’s decisions
- Use trace information and Explain output when determining impact of hint
- Be careful! Test statements thoroughly before and after adding Hints
- Revisit decisions to use Hints regularly

User-Defined Index Types

- Oracle8i allows creation of a user-defined index type to index complex data such as documents, images, video clips, spatial data, or audio clips
- Creation uses object features introduced in Oracle8 to build indexes specifically designed for complex applications such as On-Line Analytical Processing (OLAP)
- User-defined indexes may be used with user defined operators (CREATE OPERATOR)
- Specifics concerning User-Defined Index Types may be found in the Oracle8i Concepts manual, no further discussion is provided in this paper

Conclusion



- Indexing capabilities of Oracle8i are powerful and system developers should decide not just what columns to index but how to index them
- This paper presented various indexing options available and suggested when choosing a particular type of index might be the best choice
- As with all performance related issues, test, test, and test again
- Any performance oriented decision must be revisited periodically to make sure that the best choice is still being made

To contact the author:

John King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627 USA
1.800.252.0652 - 1.303.798.5727
Email: john@kingtraining.com