SOUG

Suncoast Oracle Users' Group

Technology Day 2006

Indexing & Partitioning

Paul Phillips
Cendant

Choose (And Optimize) The Best Index

To Promote
Better SQL Writing Skills
And More Efficient Systems
Among Central Florida
Oracle User
Group Members

Presentation Summary

This presentation will discuss the many current index types now available in Oracle 10g and in addition some of the less commonly used features

About Paul D. Phillips:

- Sr. Database Administrator @ Cendant Timeshare Resort Group
- Oracle Certified (8, 8i, 9i) DBA
- IT Industry Subject Matter Expert for over 20 years and published author
- Officer of CFOUG since 2002

Types of Indexes as of 10g

- Clustered
- Bit Map
- Bit Map Join
- Reverse
- Function Based
- Domain
- B-Tree

What is a Clustered Index?

- A **cluster** is a schema object that contains data from one or more tables
- Data is effectively “pre-joined”
- Commonly used by OLTP systems
- Can do Index Cluster or Hash Cluster
- Research implies Hash is faster, but has restrictions

What is a Bitmap Index?

A **bitmap** for each unique column value is used instead of a list of rowids

Each bit in the bitmap corresponds to a possible rowid

If bit is set, then it means that the row with the corresponding rowid contains the key value

Mapping function converts the bit position to an actual rowid

Why use a Bitmap Index?

- In cases where there is low cardinality of the column data
- Each key value is used instead of a list of rowids
- Can also index NULLS
- Size is orders of magnitude smaller than standard B-Tree
- In combinations, speed can be blistering fast (BITMAP And)

Why use a Bitmap Index (cont.)?

- “Minimize Records Per Block” sounds odd but works great by eliminating “assumption” of how many rows are in each block
- Must be applied after table is built but before Bitmap Indexes are created
- Tests show space saving from 2 – 15 %.

Downside to the use of Bitmap Indexes

CONCURRENCY: When many updates need to occur at the same time

Due to the potential for a large number of blocks to be involved in a single BITMAP INDEX value (I.E., low cardinality,) a single update has the potential to lock a very large portion of a table.

What is a Bitmap Join Index?

- An index for the join of two or more tables
- Much more efficient in storage than materialized join views
- Restrictions:
  - Update only one table in the join at a time
  - Must be unique or PK
  - No IOTs
  - Others

What is a Reverse Index?

Example: a standard B-Tree Index would have the following order: 537, 538, 539, 540, 541

A reverse index, though also a B-Tree structure, would contain: 045, 145, 735, 835, 935

What's the advantage? For items that are sequential, the more even distribution prevents an unbalanced leaf block structure

Disadvantage? Mainly, can't do range scans, only unique key or FFS

What is a Function Based Index?

Useful for queries that have a qualified value in the predicate

Value is precomputed and stored in the index

Classic example:

```
WHERE UPPER( NAME ) = ‘SMITH’;
```

Can create more powerful sorts: UPPER, LOWER, DESC, and NLSSORT functions

What is a Function Based Index (cont.)?

- Increase the potential for Optimizer to do RANGE SCANS:
  - CREATE INDEX idx ON T1 (col_a + col_b);
  - SELECT * FROM T1 WHERE col_a + col_b < 10;
- True DESCending order indexes are special case of FBIs
- Must have the following initialization parameters defined to create a function-based index:
  - QUERY_REWRITE_INTEGRITY set to TRUSTED
  - QUERY_REWRITE_ENABLED set to TRUE
  - COMPATIBLE set to 8.1.0.0.0 or a greater value
- Table must have statistics

What is a Domain Index?

The use of an application-specific index of type ***indextype***

An ***indextype*** is an object that specifies the routines that manage a domain (application-specific) index

Exist in the same namespace as tables, views, and other schema objects

Similar to Function Based Indexes, but is more extensible; can have many user defined objects (Functions) to precompute “value of interest”

The Classic: Binary Tree Indexes (B-Trees)

- Branches and Leafs:
  - Upper Blocks (Branches) point to lowest level (Leaf) blocks
  - Branches are for searching
  - Leaf blocks contain every indexed, non-null data value and a corresponding rowid
  - Rowid is used to locate the actual row
  - All leaf blocks are at the same depth from the root branch block
- Advantages are numerous and well documented
- New option in Version 8: **COMPRESS N**

IOT: the INDEX ORGANIZED TABLE

Introduced in Oracle version 8.

Standard tables are called Organization HEAP: data is stored as it is entered

New structure, the IOT, is defined as Organization INDEX: Data is stored in Order

Currently, must have a Primary Key definition

Redundant data storage is eliminated by having the data in the table also be the data in the Primary Key and carrying the associated non-PK data with the PK

IOTs (cont.)

Pros:

- Table can be updated without space impact, unlike Compressed Data Segments
- Useful when PK is primary access mode (lookup tables, etc.)

Cons:

- Table will always be in LOGGING mode.
- Secondary Indexes are somewhat less than efficient.

Compressed IOTs: Serendipity

Any non-unique or unique partitioned index can use the COMPRESS feature.

Syntax of this additional clause is quite simple:

COMPRESS *integer*

Where *integer* can be from 1 to the number of columns in a non-unique index, or 1 to the number of columns - 1 in a unique index.

Performance Example & Autotrace:

Query is of the form:

```
SELECT  SUM( d.Col5 )
FROM    Master m,
          Detail d
WHERE   m.Col1 = 'Time1'
AND     m.Col2 = 'AttributeValue'
AND     m.Col3 = 'CustomerValue'
AND     m.Col1 = d.Col1
AND     m.Col3 = d.Col3
AND     m.Col4 = d.Col4
```

Compression Reduces PIO and LIO

Autotrace #1: Standard tables with separate primary key indexes.

```
 Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=48 Card=1 Bytes=43)
   1    0   SORT (AGGREGATE)
   2    1     SORT* (AGGREGATE)
   3    2       TABLE ACCESS* (BY INDEX ROWID) OF 'DETAIL' (Cost =3 Card=68 Bytes=1292)
   4    3         NESTED LOOPS* (Cost=48 Card=12437 Bytes=534791)
   5    4           INDEX* (FAST FULL SCAN) OF 'MASTER_PK' (UNIQUE) (Cost=1 Card=182 Bytes=4368)
   6    4           INDEX* (RANGE SCAN) OF 'DETAIL_PK' (UNIQUE) (Cost=2 Card=68)
 Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
      8318  consistent gets (BASELINE)
          0  physical reads
          0  redo size
        390  bytes sent via SQL*Net to client
        430  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
```

IOT Reduces PIO and LIO

Autotrace #2: Non Compressed IOTs:

```
 Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=32 Card=1 Bytes =43)
   1    0   SORT (AGGREGATE)
   2    1     SORT* (AGGREGATE
   3    2       NESTED LOOPS* (Cost=32 Card=1961350 Bytes=84338050)
   4    3         INDEX* (FAST FULL SCAN) OF 'MASTER_PK' (UNIQUE) (Cost=2 Card=182 Bytes=4368)
   5    3         INDEX* (RANGE SCAN) OF 'DETAIL_PK' (UNIQUE) (Cost=2 Card=10800 Bytes=205200)
 Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
____3217  consistent gets  ( 39% less I/O or 61% REDUCTION! )
          0  physical reads
          0  redo size
        390  bytes sent via SQL*Net to client
        430  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Compressed IOT Reduces PIO and LIO

Autotrace #3: Compressed IOTs:

```
 Execution Plan
----------------------------------------------------------
  0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=26 Card=1 Bytes=41)
  1    0   SORT (AGGREGATE)
  2    1     SORT* (AGGREGATE
  3    2       NESTED LOOPS* (Cost=26 Card=35573 Bytes=1458493)
  4    3         INDEX* (FAST FULL SCAN) OF 'MASTER_PK' (UNIQUE) (Cost =2 Card=182 Bytes=4368)
  5    3         INDEX* (RANGE SCAN) OF 'DETAIL_PK' (UNIQUE) (Cost=2 Card=196 Bytes=3332)
 Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
____2940  consistent gets  ( 35 % less I/O or 65% REDUCTION! )
          0  physical reads
          0  redo size
        390  bytes sent via SQL*Net to client
        430  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Conclusion:

New index structures continue to enhance performance and flexibility of the Oracle RDBMS.

More choices, more complexity.

DBAs as well as Developers should become familiar with them to be able to choose the “best practices”

SUNCOAST ORACLE USERS' GROUP

Monthly
4th Thursday
6pm – 8pm

IBM Center
Rocky Point

WWW.SOUG.NET