

ORACLE®



ORACLE®

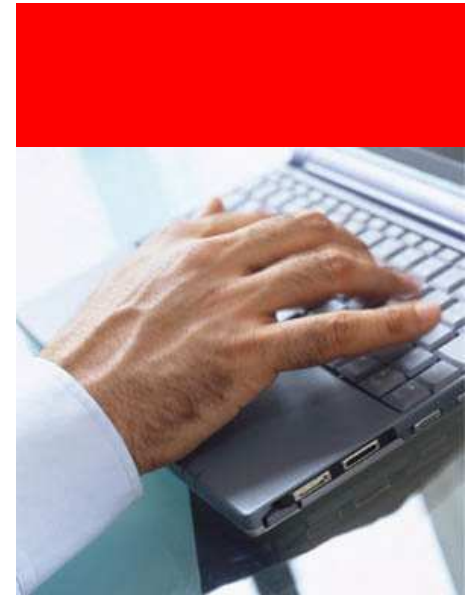
Sorted Ranked Revisited

Ian Smith
Oracle Rdb Product Architect



Presentation **Agenda**

- Overview of SORTED RANKED indices
- Case studies
- Final comments



Sorted Indices

General comments





Sorted Indices

- Selected columns are stored in the index
- Values are collated (ordered)
- Utilized by various SQL clauses to reduce I/O
 - ORDER BY
 - DISTINCT
 - GROUP BY
 - UNION DISTINCT
 - Various JOIN clauses
 - EXISTS
 - COUNT



Special Optimizations

- MAX
- MIN
- COUNT
 - MIN, MAX and COUNT perform special index traversal
 - Use small I/O path to fetch well define values
- Index Only Retrieval
- Range Retrieval

Query showing MIN and MAX optimization

```
SQL> select min(last_name) from employees;
```

```
Tables:
```

```
0 = EMPLOYEES
```

```
Aggregate: 0:MIN (0.LAST_NAME) Q2
```

```
Index only retrieval of relation 0:EMPLOYEES
```

```
Index name EMPLOYEE_LAST_NAME_INDEX [0:0] Min key lookup
```

```
Ames
```

```
1 row selected
```

```
SQL> select max(last_name) from employees where last_name < 'Smith';
```

```
Tables:
```

```
0 = EMPLOYEES
```

```
Aggregate: 0:MAX (0.LAST_NAME) Q2
```

```
Index only retrieval of relation 0:EMPLOYEES
```

```
Index name EMPLOYEE_LAST_NAME_INDEX [0:1] Max key lookup
```

```
Keys: 0.LAST_NAME < 'Smith'
```

```
Silver
```

```
1 row selected
```

Query showing COUNT lookup

```
SQL> begin
cont> if exists (select *
cont>                from candidates c
cont>                where c.last_name = :last_name)
cont> then
cont>     trace 'found';
cont> else
cont>     trace 'not found';
cont> end if;
cont> end;
```

Tables:

0 = CANDIDATES

Aggregate-F1: 0:COUNT-ANY (<subselect>) Q1

Index only retrieval of relation 0:CANDIDATES

Index name CANDIDATE_LAST_NAME_INDEX [1:1]

Keys: 0.LAST_NAME = <var0>

Index counts lookup

...



Restrictions on the key columns

- Total index key length is 255 octets
- Some encoding disables Index Only Retrieval
 - VARCHAR is stored as CHAR (space padded) in the index
 - CHAR with some collating sequences requires special encoding. i.e. for German collating sequence the character sequence “ss” is encoded as a single character, as is “ch” in the Spanish collating sequence
- Partial text fields (SIZE IS clause) can be useful to reduce index key size



Other Management Notes

- SORTED index which allow duplicates
- Pointers to duplicate rows held in a chain of duplicate nodes
 - Size of duplicate nodes is small (relative to index node)
 - there is no control over sizing, placement or order of DBKEY in the chain
- Many applications suffer from “pages discarded”
 - Page is selected as valid target
 - Not enough room exists on the page for whole node
 - The solution is THRESHOLD definitions for the index
 - Monitor using *Record Statistics* in RMU /Show Statistics



Sorted Ranked

Improved management





Sorted Ranked

- New index type added in Rdb V7.0
- Benefits include:
 - No duplicates chain (no need for THRESHOLD calculations)
 - Duplicates stored as compressed bit map
 - For low duplicate count – stored with index key
 - For high duplicate count – stored in overflow node
 - Includes exact leaf node cardinality
 - Higher nodes store approximate cardinality
 - Overflow nodes are the same size as primary nodes



Sorted Ranked and the Optimizer

- Designed with the Rdb Dynamic Optimizer in mind
- Strategy generation “Estim” phase uses the cardinalities in the index develop query cost
 - COUNT can be solved using the exact leaf cardinalities
- Duplicates are sorted in DBKEY order
 - Allows serial page retrieval (best buffer usage)
 - Index prefetch loads duplicates asynchronously
- Bitmaps can be used directly
 - Requires SET FLAGS ‘BITMAPPED_SCAN’



Unique versus Duplicates

- Should SORTED RANKED not be used for UNIQUE?
- Some database administrators believe space is saved by using SORTED for UNIQUE indices
 - For UNIQUE indices the cardinality field is compressed to a single bit
 - For cardinality 2 to 255 one byte is used
 - For cardinality 256 to 65535 two bytes (a word) is used
 - For cardinality 65536 to four bytes (a longword) is used
 - For cardinality greater than this eight bytes (a quadword is used)
- Overall optimizer savings out weigh small space saving of SORTED indices

Case Study #1

Should you use MIXED areas?





The problem statement

- Database administrator chooses to map SORTED or SORTED RANKED index to a MIXED page format area
- Good or bad?



Let us create an example database

```
SQL> create database
cont>   filename ABC
cont>
cont>   create storage area
cont>       page format MIXED
cont>       page size 4 blocks
cont>
cont>   create storage area U_AREA
cont>       page format UNIFORM
cont>       page size 4 blocks
cont> ;
```



A closer look at the database

```
$ rmu/dump/header/option=debug/out=abc.dump  
_Root: abc
```

Search for some interesting values

Mixed areas reserve
space for system record

```
$ search abc.dump "Storage area ",MAX_NEW_SEG_LEN/exact
Storage area "RDB$SYSTEM"
MAX_FREE_LEN = 982.    MAX_NEW_SEG_LEN = 964.    MAX_PNO = 703.
Storage area "M_AREA"
MAX_FREE_LEN = 1988.    MAX_NEW_SEG_LEN = 1980.    MAX_PNO = 704.
Storage area "U_AREA"
MAX_FREE_LEN = 2006.    MAX_NEW_SEG_LEN = 1988.    MAX_PNO = 701.
...
```

Uniform areas give you
8 extra bytes



Notes and Suggestions

- If mixed not used for HASHED index then waste the system record space (RDB\$SYSTEM_RECORD)
- Used to locate HASH bucket on the page
 - Key hashes to page
 - Page system record has dbkey of hash bucket
 - Hash bucket has key value and list of row dbkeys
- Must reserve space for future use
- GLOBAL BUFFERS can not use a small but significant amount of memory
- I/O from disk is slight less effective
- It all depends on the size of the index node

Case Study #2



Should you use default node size?



The problem statement

- Probably the answer is NO!
- The default node size is 430
- No idea why, the reason is lost in history
- Many reported performance problems are solved by using a larger node size



Create a simple database to test theory

```
SQL> create database
cont>   filename abc
cont>   create storage area U1
cont>     page format uniform
cont>     page size 1 block
cont>   create storage area U2
cont>     page format uniform
cont>     page size 2 block
cont>   create storage area U3
cont>     page format uniform
cont>     page size 4 block
cont> ;
```

- Use different page size to see what we could have used

Locate MAX_NEW_SEG_LEN

Storage area "RDB\$SYSTEM"

MAX_FREE_LEN = 982. MAX_NEW_SEG_LEN = 964. MAX_PNO = 703.

Storage area "U1"

MAX_FREE_LEN = 470. MAX_NEW_SEG_LEN = 452. MAX_PNO = 703.

Storage area "U2"

MAX_FREE_LEN = 982. MAX_NEW_SEG_LEN = 964. MAX_PNO = 703.

Storage area "U3"

MAX_FREE_LEN = 2006. MAX_NEW_SEG_LEN = 1988. MAX_PNO = 701.

- We can see that 430 isn't even a best choice for 1 block page (waste 32 bytes)
- Page size 2, two on a page (waste 106 bytes)
- Page size 4, four on a page (waste 254 bytes)



What about SORTED indices?

- When you have SORTED indices then these gaps may be used by duplicate nodes
- Typically these duplicate nodes will be 112 bytes long
- Too large for the unused space on the default page size
- So space will be needed on other pages. This implies extra I/O to find free space (INSERT) and retrieval of duplicates (SELECT, UPDATE, DELETE)
- Describe percentage of node size versus duplicate node using THRESHOLDS ARE (78, 94)



Futures

- The next release of Oracle Rdb plans to change this behavior
- Different default page size
- Different default node size
- If you already specify NODE SIZE in your index then you'll see no changes
- Only affects plain CREATE INDEX commands

Case Study #3

Simple errors to avoid?





The problem statement

- Recent problem report
- Noticed curious definitions
- Page sizes change over time, but node size setting maybe be forgotten

One of the indices looked like this

```
SQL> show index IX_1
Indexes on table ...:
IX_1                                with column COL1
  No Duplicates allowed
  Type is Sorted Ranked
  Key suffix compression is DISABLED
  Node size 1940
  Percent fill 100
  Store clause:          STORE
                    in INDEX_AREA
```

What does this area look like?

INDEX_AREA

Access is: Read write
Page Format: Uniform
Page Size: 4 blocks
Area File: RAID1:[TESTING]TESTINDEX.RDA;1
Area Allocation: 28800 pages
Extent: Enabled
Area Extent ...
Snapshot File: RAID1:[TESTING]TESTINDEX.SNP;1
Snapshot Allocation: 1000 pages
Snapshot Extent ...
Locking is Row Level
Using Cache INDEX_CACHE



We could do better?

- We saw this type of area in case study #1
- For UNIFORM, page size 4 blocks
- MAX_NEW_SEG_LEN = 1988
- But this index, SORTED RANKED, uses 1940
- SORTED RANKED therefore no tiny duplicates node to fill in the 48 byte gaps
- Why not use 1988 as the NODE SIZE?
- Fill the whole page!

What about that **CACHE** we were using?

```
SQL> show cache index_cache
```

INDEX_CACHE

Cache Size:	200000 rows
Row Length:	2000 bytes
Row Replacement:	Enabled
Shared Memory:	Process
Large Memory:	Disabled
Window Count:	100
Working Set Count:	10
Location:	\$1\$DGA1:[TESTING]

...

Snapshot in Cache:	Enabled
Snapshot Cache Size:	1000 row



Summary

- The cache can hold the 1988 NODE SIZE already
- The 48 bytes can be reclaimed for real work
- The larger node will contribute to reduced I/O to read the index
- Special attention needs to be taken:
 - Page size changes
 - Cache changes (physical area cache)
 - Index node size changes

Case Study #4

Making use of *Index Only* retrieval





The problem statement

- How many databases have small lookup tables?
- Consider the DEPARTMENTS table in the PERSONNEL database
- We have a DEPARTMENT_CODE column that is stored as a foreign key in various other tables
- Reports or data display wants to translate this to the full DEPARTMENT_NAME
- Add an index on both columns



Creating the index

```
SQL> create unique index DEPARTMENTS_LOOKUP_INDEX  
cont> on DEPARTMENTS (DEPARTMENT_CODE, DEPARTMENT_NAME)  
cont> type is sorted ranked  
cont> enable compression;
```

- Make the DEPARTMENT_CODE the leading segment to allow lookup
- Enable compression to reduce the size of the stored data
- Trailing spaces in the DEPARTMENT_NAME in particular



This is what we have


```
SQL> show index DEPARTMENTS_LOOKUP_INDEX
Indexes on table DEPARTMENTS:
DEPARTMENTS_LOOKUP_INDEX          with column DEPARTMENT_CODE
                                   and column DEPARTMENT_NAME

No Duplicates allowed
Type is Sorted Ranked
  Duplicates are Compressed Bitmaps
Key suffix compression is ENABLED (Min run length 2)
```

- The compression default is RUN LENGTH 2, which means two or more spaces get compressed
- Compression characters are spaces (based on character set) and zeros
- Zeros apply to numeric, date/time and interval types



Using this index



```
SQL> select DEPARTMENT_NAME from DEPARTMENTS
cont> where DEPARTMENT_CODE = 'ADMN';
Tables:
  0 = DEPARTMENTS
Index only retrieval of relation 0:DEPARTMENTS
  Index name  DEPARTMENTS_LOOKUP_INDEX [1:1]
    Keys: 0.DEPARTMENT_CODE = 'ADMN'
  DEPARTMENT_NAME
  Corporate Administration
1 row selected
```

More complexity

```
SQL> select last_name, first_name, department_name
cont> from employees inner join job_history using (employee_id)
cont> inner join departments using (department_code)
cont> where employee_id = '00164'
cont> and job_end is null;
Tables:
  0 = EMPLOYEES
  1 = JOB_HISTORY
  2 = DEPARTMENTS
Cross block of 3 entries  Q1
Cross block entry 1
  Get      Retrieval by index of relation 0:EMPLOYEES
          Index name  EMPLOYEES_HASH [1:1]          Direct lookup
          Keys: 0.EMPLOYEE_ID = '00164'
Cross block entry 2
  Conjunct: MISSING (1.JOB_END)
  Get      Retrieval by index of relation 1:JOB_HISTORY
          Index name  JOB_HISTORY_HASH [1:1]
          Keys: 0.EMPLOYEE_ID = 1.EMPLOYEE_ID
          Bool: 1.EMPLOYEE_ID = '00164'
Cross block entry 3
  Index only retrieval of relation 2:DEPARTMENTS
          Index name  DEPARTMENTS_LOOKUP_INDEX [1:1]
          Keys: 1.DEPARTMENT_CODE = 2.DEPARTMENT_CODE
EMPLOYEES.LAST_NAME    EMPLOYEES.FIRST_NAME    DEPARTMENTS.DEPARTMENT_NAME
Toliver                Alvin                    Board Manufacturing North
1 row selected
```



Discussion

- Adding a row cache can help performance for this type of index
- Only need to cache the index, and depending on the size of the table, make the cache large enough for all the index nodes

Case Study #5

Bitmapped Scan





Bit maps

- Bitmap represents the dbkeys of rows
- Operations such as AND and OR can be solved by using bitmap operations on the bitmaps themselves
- Resulting bitmap is the dbkeys that match all criteria
- Contrast: traditional lookup will be required to read index on one attribute and then filter fetched rows



Support

- Support built into SORTED RANKED
 - No special index type
- Can be enabled per query
 - SET FLAGS 'BITMAPPED_SCAN'
- Not suitable for every query
- Best used when managing indices with many duplicates
- Supported as part of the Dynamic Optimizer



Our sample database

```
SQL> create table car  
cont> (lplate char(6)  
cont> ,make char(10)  
cont> ,ctype char(10)  
cont> ,colour char(10)  
cont> ,cyear integer  
cont> ,price integer);
```

- Fact table with many attributes
- Probably the database has 100's of million rows



Indices

```
SQL> create index ilp on car(lplate) type sorted ranked;  
SQL> create index imake on car(make) type sorted ranked;  
SQL> create index itype on car(ctype) type sorted ranked;  
SQL> create index icolour on car(colour) type sorted ranked;  
SQL> create index iyear on car(cyear) type sorted ranked;
```

- Use SORTED RANKED on each attribute
- We will have lots of duplicates
- Don't try this on regular SORTED indices

Query without using Dynamic Optimizer

```
SQL> select lplate from car  
cont> where make = 'holden' and ctype = 'sedan'  
cont> and cyear = 1978 and colour = 'red';
```

Tables:

0 = CAR

Conjunct: (0.MAKE = 'holden') AND (0.CTYPE = 'sedan')
AND (0.COLOUR = 'red')

Get Retrieval by index of relation 0:CAR

Index name IYEAR [1:1]

Keys: 0.CYEAR = 1978

- Static optimizer chooses one (best?) index and filters remaining columns



Enable the Dynamic optimizer

Tables:

0 = CAR

Leaf#01 FFirst 0:CAR Card=6047

Bool: (0.MAKE = 'holden') AND (0.CTYPE = 'sedan')
AND (0.CYEAR = 1978) AND (0.COLOUR = 'red')

BgrNdx1 IYEAR [1:1] Fan=17

Keys: 0.CYEAR = 1978

BgrNdx2 ICOLOUR [1:1] Fan=14

Keys: 0.COLOUR = 'red'

BgrNdx3 ITYPE [1:1] Fan=14

Keys: 0.CTYPE = 'sedan'

BgrNdx4 IMAKE [1:1] Fan=14

Keys: 0.MAKE = 'holden'



Dynamic Optimizer

- Uses many indexes
- Ordered by effectiveness (based on estimations)
- Competes the indices and abandons those that are unproductive
- Still produces better retrieval
- Now enable bitmap scan
 - SET FLAGS 'BITMAPPED_SCAN'



Now with Bitmapped Scan enabled

Tables:

0 = CAR

Leaf#01 FFirst 0:CAR Card=6047 **Bitmapped scan**

Bool: (0.MAKE = 'holden') AND (0.CTYPE = 'sedan')
AND (0.CYEAR = 1978) AND (0.COLOUR = 'red')

BgrNdx1 IYEAR [1:1] Fan=17

Keys: 0.CYEAR = 1978

BgrNdx2 ICOLOUR [1:1] Fan=14

Keys: 0.COLOUR = 'red'

BgrNdx3 ITYPE [1:1] Fan=14

Keys: 0.CTYPE = 'sedan'

BgrNdx4 IMAKE [1:1] Fan=14

Keys: 0.MAKE = 'holden'



Scanning

- Now reads bitmaps from each index and performs an in-memory bitmap operation
- Resulting bitmap should result in far fewer dbkeys
- End result is less I/O to fetch the data rows



Comments

- See if this meets your needs
- Use `SHOW STATISTICS` to compare I/O and CPU
- Processing the bitmap may use more CPU but should significantly reduce I/O
- Not all queries will need or use bit maps



Final Comments

Much more to say...





What will we talk about next time?

- We discussed
 - Sorted ranked indices
 - Node sizing
 - THRESHOLD ARE clause
 - BITMAPPED SCAN
- Didn't get to
 - Partitioning
 - Management of large indices
 - Query tuning
 - Reverse scan



Questions? & Answers!



ORACLE IS THE INFORMATION COMPANY



ORACLE®