Rdb Journal
technical corner

Guide to Performance and Tuning: Space Management and Thresholds

A feature of Oracle Rdb

By Ian Smith
Oracle Rdb Relational Technology Group
Oracle Corporation

The Rdb Technical Corner is a regular feature of the Oracle Rdb Web Journal. The examples in this article use SQL language from Oracle Rdb release 7.1 and later versions.

This is a revised version of the article titled: *Rdb Technical Notes #17 - Space Management and Thresholds in Rdb, 29 September 1997.*

Guide to Performance and Tuning: Space Management and Thresholds

Space on a database page can be managed in Rdb by defining storage thresholds. These thresholds encode simple rules so that the database system can efficiently determine candidate pages for storing new data.

Storage areas can be defined as either **page format is mixed** in which case each page may contain a variety of table rows, index nodes and hash index buckets, or as **page format is uniform** that allows just one type of row or index to be present on the page. These two types of storage areas support threshold definitions in different ways; however, the general principles described in this article can be applied to either type of storage area.

Oracle Rdb supports the definition of thresholds for storage areas, storage maps and indices. The values for thresholds can be defined and changed using the following commands:

- The **create storage area** clause of the **create database** statement, or the **add storage area** clause of the **alter database** statement allow the specification of thresholds for new **page format is mixed** storage area.

- The **create index** and **alter index** statements allow the **thresholds are** clause as a storage area option for any **page format is uniform** storage areas. In the case of **alter index** the thresholds can be specified for all new storage areas referenced by the index. Oracle Rdb V7.2.1 or later is required to modify the **thresholds** setting for an existing partition using the **alter index** statement.[1]

- The **create storage map** and **alter storage map** statements allow the **thresholds are** clause as a storage area option for any **page format is uniform** storage areas. In the case of **alter**

[1] Please refer to the document *Guide to Performance and Tuning: Managing the AIP Data Structure.*

storage map the thresholds can be specified for all new storage areas referenced by the table. Oracle Rdb V7.2.1 or later is required to modify the **thresholds** setting for an existing partition using the **alter storage map** statement.

- RMU/REPAIR/SPAM/INITIALIZE=LAREA_PARAMETERS can be used to set the thresholds for a logical area. The thresholds are specified within an options file supplied to the LAREA_PARAMETERS keyword[2]. This is an offline operation.

- RMU/SET AIP can be used to set the thresholds for logical areas. This is an online operation. This command requires Rdb V7.2.1 and later versions. Please refer to article *Guide to Database Performance and Tuning: Managing the AIP Structure* and also the *Oracle Rdb Release Notes* for more details of this Rdb V7.2.1 feature.

- RMU/MOVE_AREA can be used to relocate a MIXED format storage area and modify the thresholds using the /THRESHOLD qualifier.

To understand what this feature does, and how to define it appropriately requires an understanding of space management in Oracle Rdb. This article describes space management, the THRESHOLDS feature and provides a simple tutorial using the MF_PERSONNEL database.

Space Management

As applications **insert** rows into a table Oracle Rdb stores those rows and any associated new index nodes into various storage areas. Rdb uses the storage map for the table and the associated **store** clause for the index to allocate rows and index nodes to different physical storage areas. Eventually pages will be filled and can no longer be used as a target for the **insert** statement.

So now we come to the main focus of this article. How does Rdb know that a page is full when looking for free space to insert a new row? When first inserting data the database system could read each page to see if there was room to store the new data, but surely this would be excessive (in I/O and elapsed time). The more rows already exist in the table the longer it would take to store a row.

To avoid this overhead Rdb uses a special structure page called a *Space Management Page*, or SPAM page, that is placed at fixed intervals in each storage area. The first page in each storage area is a SPAM page and each manages a range of pages known as the *SPAM interval*. For the

[2] An example appears in the section **Repairing Thresholds**.

examples in this article we use the default page size of 2 blocks for a MIXED format area, which yields a default SPAM interval of 216 pages[3]. This means that the database system can easily compute the location of SPAM pages in the example database at every 216 pages in the storage area. SPAM intervals differ for areas with different page sizes and formats. The SPAM interval is fixed for **uniform** areas (calculated by Rdb based on the page size), but may be adjusted in **mixed** format areas.

The SPAM page is used for various management tasks, but we will look at its usage for free space management. Each SPAM page contains a bit vector that describes every page in its SPAM interval. Two bits are reserved for each page and these bits encode the values 0, 1, 2, and 3. We will describe the use for all these coded values later but a value of 3 indicates that a page is at full capacity, i.e. cannot be used to store more data.

Filling a page with data can be visualized by imagining a glass being filled with water a little at a time. The example page size will allow up to 964 octets of data to be stored with the rest of the pages reserved for control information[4]. After adding seven EMPLOYEES rows (at 126 octets each) and accounting for the page level overhead the page only 26 free octets remain. Therefore, the page has insufficient space for an additional EMPLOYEES row. If this is the only row type being stored on the page then the page is effectively full. Rdb now marks the page to indicate that it is no longer considered a candidate when inserting new data.

> ***Note**: The actual free space remaining on the page (the 26 octets in the previous example) can still be used when an UPDATE statement expands a row. For example, if the value in the column LAST_NAME is changed from "SMITH" to "KIRKPATRICK" then the compressed row will be longer than it was prior to the UPDATE. In many cases the row is able to grow into the remaining free space on the page. If there were insufficient space Rdb would fragment the row storing the new fragment on another page (or pages) and modify the existing row to reference the fragment chain.*[5]

When a table is mapped to a **uniform** format area Rdb defaults the thresholds to (0, 0, 0). In this case the threshold bits will only indicate full or not full using the values 0 and 3. The algorithm uses the row length stored in the *Area Inventory Page*, or AIP.

[3] Use RMU/DUMP/HEADER or SQL SHOW STORAGE AREA commands to see the SPAM interval used for any storage area.
[4] The page overhead is fixed in size so using a larger page size will reduce the overhead per row. Larger page sizes also have other benefits – such as using less I/O to read table data.
[5] In general fragmentation should be avoided because it will require extra I/O and CPU time to re-assemble the row, and such rows are never saved in a ROW CACHE.

The AIP pages are linked together in a chain that is stored in the RDB$SYSTEM storage area. Each AIP page describes several logical areas. The entries in the AIP chain describe the mapping to the physical, the nominal record length of the record (row or index node), the storage thresholds, the type of object being stored, and other management information. The mapping to the physical area is used to translate DBKEY references to physical storage areas. The AIP record length is stored in the AIP when a table, storage map or index is created.

The command RMU/DUMP/LAREA=RDB$AIP can be used to examine the AIP record length for any table or index. The following extract from an RMU/DUMP of the AIP pages for the MF_PERSONNEL database shows an entry for EMPLOYEES and an entry for JOB_HISTORY. When a table is partitioned across multiple areas there will exist multiple AIP entries for a storage map, or index – one for each partition.

```
                                        entry #13
                          000002EA  0323  first area bitmap page 746
                         0001 003E  0327  logical area 62, physical area 1
                                09  032B  area name length 9 bytes
00000000000000534545594F4C504D45  032C  area name 'EMPLOYEES.......'
  00000000000000000000000000000000  033C  area name '................'
                          00000035  034B  snaps enabled TSN 53
                              007E  034F  record length 126 bytes
                          00000000  0351  MBZ '....'
                                01  0355  entry is in use
                              0000  0356  MBZ '..'
                            000000  0358  thresholds are (0,0,0)
                                00  035B  MBZ '.'
                                01  035C  record type data table
                                00  035D  MBZ '.'

                                        entry #0
                          00000317  0024  first area bitmap page 791
                         0001 0041  0028  logical area 65, physical area 1
                                0B  002C  area name length 11 bytes
000000000059524F545349485F424F4A  002D  area name 'JOB_HISTORY.....'
  00000000000000000000000000000000  003D  area name '................'
                          0000003C  004C  snaps enabled TSN 60
                              002A  0050  record length 42 bytes
                          00000000  0052  MBZ '....'
                                01  0056  entry is in use
                              0000  0057  MBZ '..'
                            000000  0059  thresholds are (0,0,0)
                                00  005C  MBZ '.'
                                01  005D  record type data table
                                00  005E  MBZ '.'
```

***Note**: When examining the output for RDB$AIP you will notice some entries with the name of the table that was just created but marked as deleted. This occurs because Rdb processes first a **create table** statement and creates a default logical area, and later when the **create storage map** statement is processed this original logical area is no longer need and so is discarded.*

The RMU/SHOW AIP command[6] shows this same information uncluttered by dump annotations.

```
$ rmu/show aip sql$database jobs

Logical area name JOBS
Type: TABLE
Logical area 96 in mixed physical area 7
Physical area name JOBS
Record length 41
AIP page number: 154
ABM page number: 0
Snapshot Enabled TSN: 128
```

The AIP length is automatically calculated and stored when you perform a **create table**, **create storage map** or **create index** statement. The length used for the table or storage map is calculated from the sum of the column lengths, plus overhead for NULL bit vectors and row overhead. The AIP length for an index is derived from the **node size** clause used when creating a **sorted** or **sorted ranked** index. If the index is created without a node size clause then Rdb will calculate one based size of the index key.

A more complex page storage arrangement exists in **mixed** format storage areas. In areas of this type more than one row type may be stored on a page along with sorted index nodes and hash index buckets. For instance, my application might want to store JOB_HISTORY rows that are related to a particular EMPLOYEES row on the same page and cluster these rows around related hashed indices (one for EMPLOYEES and one for JOB_HISTORY). This enables faster retrieval of the employee and all their job history data because much of the data has already been fetched into memory when the hashed index was referenced.

In the example database MF_PERSONNEL the EMPLOYEES table requires 134 octets (including overhead) to store an uncompressed row, and the JOB_HISTORY table requires 50 octets per row[7].

[6] Requires Oracle Rdb V7.2.1 and later.

After storing four rows in EMPLOYEES and seven rows in JOB_HISTORY the page is left with 78 octets free. At this point it is now too full to hold another EMPLOYEES row, however, we have room for a single JOB_HISTORY row[8].

How does Rdb know that a page that is too full for an EMPLOYEES row has sufficient space for a JOB_HISTORY row? This knowledge is provided by the THESHOLDS settings in the SPAM pages. The database administrator describes the fullness rules using the THRESHOLDS clause when creating a MIXED format storage area, or when mapping tables and indices to uniform storage areas.

The thresholds clause allows up to three percentage values that describe the high water mark (using a partly full glass analogy) for various row types. Too full for an EMPLOYEES row means the free space on the page falls below 134 octets, and too full for a JOB_HISTORY row means it falls below 50 octets free, and so on.

These octet lengths are expressed as percentages relative to the page size to make the bit encoding simple and compact.

When using a page size of 2 blocks for a mixed format area the amount of free space on the page is 964 octets. The RMU/DUMP/HEADER command can be used to display information for each storage area. The /OPTION=DEBUG qualifier causes dump to display an extra section containing useful details. In the partial output from RMU shown below you can see the lines that say MAX_FREE_LEN = 964 and MAX_SEG_LEN = 964.

```
Storage area "EMPIDS_LOW"
    Area ID number is 3
    Filename is "DISK1: [DATABASES]EMPIDS_LOW.RDA;1"
    Access mode is READ/WRITE
    Pages...
      - Page format is mixed
      - Page size is 2 blocks
      - Initial data page count was 51
      - Current physical page count is 52
      - Row level locking is enabled
    Extension...
      - Extends are enabled
      - Extend area by 20%, minimum of 99 pages, maximum of 9999 pages
      - Volume set spreading is enabled
      - Area has never been extended
    Snapshots...
      - Snapshots are enabled
      - Snapshot area ID number is 12
    Space Management...
```

[7] We will calculate the lengths later in a worked example.
[8] Please note that we have assumed no compression on the row to simplify this discussion.

```
      - SPAMS are enabled
      - Interval is 216 data pages
      - Thresholds are 70%, 85%, and 95%
      - Current SPAM page count is 1
    Status...
      - Area has never been backed up
      - Area has never been incrementally restored

FILID_ENT[3.] @004201B0   2PPL_ENABLED = 00          ACCESS_MODE = 0.
ACTIVE_FLG = 01           AIJ_RCVR_VNO = 0.
BACKUP_TAD = 17-NOV-1858 00:00:00.00                BACKUP_TSN = 0.
CHECKSUM = 0.             CLC_PNO = 51.             CLUMP_PAGCNT = 3.
COMMIT_TSN = 0.           CORRUPT_FLG = 00          COUPLED = 00
CSM_FLG = 01              DBID = 3.                 EXT_COUNT = 0.
EXT_ENABLED = 01          EXT_PERCENT = 20.
FILNAM = "DISK1:[DATABASES]EMPIDS_LOW.RDA;1"
INC_RES_TAD = 17-NOV-1858 00:00:00.00               INCONSISTENT_FLG = 00
MAX_EXT_PAGCNT = 9999.    MAX_FREE_LEN = 964.       MAX_NEW_SEG_LEN = 956.
MAX_PNO = 52.             MAX_SEG_LEN = 964.        MIN_EXT_PAGCNT = 99.
MIXED_FMT_FLG = 01        P0_VBN = 1.               PAGES_PER_SPAM = 216.
PAG_BLKCNT = 2.           PAG_DBID = 3.             PAG_LEN = 1024.
PAG_PAD_LEN = 102.        PPSP1 = 217.              RELATED = 00
RESTRUCTURED = 00         SNAPS_ALLOWED = 01        SNAP_DBID = 12.
SNAP_FLG = 00             SNAPS_ENABLED = 01        SNAPS_ENABLED_TSN = 0.
SPAMS = 01                SPAMS_ENABLED = 01        SPAMVEC_LEN = 54.
SPAM_T1 = 674.            SPAM_T2 = 818.            SPAM_T3 = 914.
SPREAD_FLG = 01           STAREA_NAME = "EMPIDS_LOW"
THRESHOLD = (70., 85., 95.)
```

The value for MAX_FREE_LEN could also be calculated by hand using the data structure definitions described in the *Oracle Rdb Guide to Database Performance and Tuning*. However, this information can easily be displayed using the optional INFORMATION table Rdb$STORAGE_AREAS which contains the column Rdb$MAX_PAGE_FREE_SPACE.

In addition to the data itself, each page has two arrays used to locate the row on the page. Together these arrays provide the **line** used in a DBKEY. This LDX/TDX vector requires eight octets per row. Please refer to the Rdb documentation for a description of these arrays.

Calculating the Thresholds

Now continuing with the example, the octet length for EMPLOYEES and JOB_HISTORY can be encoded as percentages:

- Fullness-percentage = 100 – (row-length * 100 / max-free-len)
- For JOB_HISTORY this will be: 100 - (50 * 100 / 964) = 95%
- for EMPLOYEES this will be: 100 - (134 * 100 / 964) = 86%

As there are no other row types we can set the initial threshold to 86% as it serves no purpose in this area and we want to avoid an extra threshold level that would cause the SPAM to be updated for no useful purpose. Thus we have the thresholds (86, 86, 95). Working backwards we can see that if 5% of the page was free then that would leave 48 octets available which is too small for a JOB_HISTORY row, but if 6% is free (i.e. below the upper limit) then that would leave at least 57 octets free for use by a JOB_HISTORY row.

As rows are stored on the page and the percentage at each level (threshold) is exceeded an indication is recorded in the SPAM bit mask to indicate the level reached by each page. Likewise, as rows are deleted, or updated (rows may shrink or expand across the threshold) the threshold bits are updated when the level changes.

Threshold Values		
Threshold	**Binary Mask**	**Meaning**
3	11	The page is full, or at least cannot accommodate more data because the remaining free space is too small for any row type written to this page. For example not even the small JOB_HISTORY row will fit on the page.
2	10	The page is less than threshold three. For example the larger EMPLOYEES row will not fit on this page, however, a JOB_HISTORY row will fit.
1	01	The page is less than threshold two.
0	00	The page is not at threshold one. This is not an indicator of an empty page, however, there exists space for at least one of the record types.

During insert of new data the threshold values as percentages are used to calculate a suitable set of threshold bits for the row. For instance to store an EMPLOYEES row in the area we need look only at pages at threshold level 0. Any page at threshold 1 or 2 indicates that there is only room for

a JOB_HISTORY row, and if it is at threshold 3 then the page is considered full. This can be an enormous help during insert because now Rdb can examine a bit map to see which pages need to be read from disk, and skip over those that are marked as full.

When a table is full this allows Rdb to quickly skip through the storage area avoiding pages (or possibly entire SPAM intervals) in which no suitable space is available. By defining multiple threshold values will ignore pages that are marked as full and also those marked as threshold 1 and 2 because they still have too little space to store an EMPLOYEES row.

Once a target page has been found future inserts in the same session will remember the location and so need not scan the SPAM pages again. In Oracle Rdb V7.2 and later this knowledge is shared across the cluster with other users of this storage area.

Other Examples

Although the examples so far have used fixed length rows by disabling compression, this same technique is useful for variable length rows, even for the same table. Oracle Rdb uses the actual stored length when looking for space on the page, which means the compressed table row or the variable length segments of a LIST OF BYTE VARYING column. Examples of this type of data include:

Data rows from a table with COMPRESSION ENABLED

When compression is enabled the length of rows may varying in size. Variability will depend on the compression characteristics of the data. Oracle Rdb uses a form of run-length encoding to compress sequences of identical octets into small counted strings[9]. The database administrator needs to know a little about the characteristics of the data. This may be derived from the current database being tuned, or can be determine by educated guessing base on knowledge of the table. After examining the data the database administrator could set the thresholds to the MINIMUM, AVERAGE and MAXIMUM row lengths expected.

[9] If the data does not compress then the compression overheads in the data might actually grow the stored data. If this occurs for many rows then compression should be disabled to save space and CPU cost.

Segments of a LIST OF BYTE VARYING column

LIST data by nature is often variable length. For example, in a medical records database the LIST might contain the comments entered by the physician, each line of comment will be variable in length. Based on the usage of the LIST column it might be easy to predict the MINIMUM, AVERAGE and MAXIMUM segment lengths.

These values can be calculated using a simple application that reads each row of the table and opens a **list cursor** for each list column. The SQLCA contains information about the number of segments and the maximum segment size. While the cursor could fetch each segment to compute the average length, it might be sufficient to use the data provided in the SQLCA. Alternatively, RMU/EXTRACT/ITEM=VOLUME/OPTION=COLUMN can do some sampling for you. The except below shows the output for the RESUMES table:

```
Table RESUMES all is 1;
      List RESUME
         Cardinality IS 1
            Number of segments is 19
            Average length of segments is 30;
```

LIST OF BYTE VARYING data can be, and is most often, mapped to a shared area. In this case the lengths of all the segments from different LIST columns (possibly from different tables) must be taken into consideration.

Prior to Oracle Rdb release 7.1 a **page format is mixed** area was the only way to assign THRESHOLDS for LIST data. The default **list storage area** or a **list** storage map was used to direct **list of byte varying** data to a mixed format storage areas. With Rdb V7.1 and later a **create storage map** … **store lists** statement can be used to map **list** data to individual uniform logical areas, and assign a pertinent **threshold** clause that represents the expectations for the data.

The following example shows the use of storage area attributes in a LIST storage map. The storage area attributes must immediately follow the storage area name (as in table storage maps).

```
SQL> create database
cont>     filename 'DB$:MULTIMEDIA'
cont>
cont>     create storage area PHOTO_AREA1
cont>         filename 'DB$:PHOTO_AREA1'
```

```
cont>          page format UNIFORM
cont>
cont>      create storage area PHOTO_AREA2
cont>          filename 'DB$:PHOTO_AREA2'
cont>          page format UNIFORM
cont>
cont>      create storage area TEXT_AREA
cont>          filename 'DB$:TEXT_AREA'
cont>          page format UNIFORM
cont>
cont>      create storage area AUDIO_AREA
cont>          filename 'DB$:AUDIO_AREA'
cont>          page format UNIFORM
cont>
cont>      create storage area DATA_AREA
cont>          filename 'DB$:DATA_AREA'
cont>          page format UNIFORM
cont> ;
SQL>
SQL> create table EMPLOYEES
cont>      (name        char(30),
cont>       dob         date,
cont>       ident       integer,
cont>       photograph list of byte varying (4096) as binary,
cont>       resume     list of byte varying (132) as text,
cont>       review     list of byte varying (80) as text,
cont>       voiceprint list of byte varying (4096) as binary
cont>      );
SQL>
SQL> create storage map EMPLOYEES_MAP
cont>      for EMPLOYEES
cont>      enable compression
cont>      store in DATA_AREA;
SQL>
SQL> create storage map LISTS_MAP
cont>      store lists
cont>          in AUDIO_AREA
cont>                  (thresholds are (89, 99, 100)
cont>                  ,comment is 'The voice clips'
```

```
cont>                   ,partition AUDIO_STUFF)
cont>               for (employees.voiceprint)
cont>           in TEXT_AREA
cont>                   (thresholds is (99)
cont>                   ,partition TEXT_DOCUMENTS)
cont>               for (employees.resume, employees.review)
cont>           in (PHOTO_AREA1
cont>                   (comment is 'Happy Smiling Faces?'
cont>                   ,threshold is (99)
cont>                   ,partition PHOTOGRAPHIC_IMAGES_1)
cont>               ,PHOTO_AREA2
cont>                   (comment is 'Happy Smiling Faces?'
cont>                   ,threshold is (99)
cont>                   ,partition PHOTOGRAPHIC_IMAGES_2)
cont>               )
cont>               for (employees.photograph)
cont>               fill randomly
cont>           in RDB$SYSTEM
cont>                   (partition SYSTEM_LARGE_OBJECTS);
SQL>
SQL> show storage map LISTS_MAP;
     LISTS_MAP
 For Lists
 Store clause:           STORE lists
        in AUDIO_AREA
                (thresholds are (89, 99, 100)
                ,comment is 'The voice clips'
                ,partition AUDIO_STUFF)
            for (employees.voiceprint)
        in TEXT_AREA
                (thresholds is (99)
                ,partition TEXT_DOCUMENTS)
            for (employees.resume, employees.review)
        in (PHOTO_AREA1
                (comment is 'Happy Smiling Faces?'
                ,threshold is (99)
                ,partition PHOTOGRAPHIC_IMAGES_1)
            ,PHOTO_AREA2
                (comment is 'Happy Smiling Faces?'
```

```
                ,threshold is (99)
                ,partition PHOTOGRAPHIC_IMAGES_2)
            )
            for (employees.photograph)
            fill randomly
        in RDB$SYSTEM
                (partition SYSTEM_LARGE_OBJECTS)

 Partition information for lists map:
 Vertical Partition: VRP_P000
  Partition: (1) AUDIO_STUFF
    Fill Randomly
   Storage Area: AUDIO_AREA
         Thresholds are (89, 99, 100)
 Comment:       The voice clips
  Partition: (2) TEXT_DOCUMENTS
    Fill Randomly
   Storage Area: TEXT_AREA
         Thresholds are (99, 100, 100)
  Partition: (3) PHOTOGRAPHIC_IMAGES_1
    Fill Randomly
   Storage Area: PHOTO_AREA1
         Thresholds are (99, 100, 100)
 Comment:       Happy Smiling Faces?
  Partition: (3) PHOTOGRAPHIC_IMAGES_2
   Storage Area: PHOTO_AREA2
         Thresholds are (99, 100, 100)
 Comment:       Happy Smiling Faces?
  Partition: (4) SYSTEM_LARGE_OBJECTS
    Fill Randomly
   Storage Area: RDB$SYSTEM
SQL>
SQL> commit;
```

SORTED index nodes, and duplicate nodes

Sorted indices use fixed size nodes based on the **node size** clause. However, the created duplicate nodes will vary in size. When the index is created with data in the table Rdb minimizes the duplicate node size to reduce wasted space. Therefore, a logical area containing an index allowing duplicates needs to be tuned as though there were several record types.

SORTED RANKED index nodes

Sorted ranked indices store the duplicates information within the standard index node alongside the key value. Only when the number of duplicates for a key grows very large will a duplicate node be created. The duplicate nodes are always the same size as the other nodes in the index, which makes **thresholds** definition simple. In reality the default **thresholds** for the uniform area may be best for this type index.

When Can Thresholds Go Wrong?

Many reports of excessive I/O to insert a row involve the incorrect, or untuned SPAM thresholds. Correction of these types of problems is important in production applications.

Threshold values are often untuned, namely the default uniform algorithm is used (i.e. thresholds are (0, 0, 0) and the AIP length is used), or the default storage area thresholds are (70, 85, 95) and are not appropriate for the stored data.

The remainder of this article will describe the various problems with excessive pages checked and look at corrective actions in each case.

Large Objects

The LIST OF BYTE VARYING column data (variously called BLOB - binary large object, and segmented strings) is stored as a list of data segments. Each segment can vary in size and so presents a similar problem to differently sized table rows.

If an application is managing image data then it probably uses a few different sizes for the data. For example, a photographic image may be stored as many large fixed sized data segments

followed by smaller thumbnail images. Other applications may store fixed sized segments based on the size of the LIST OF BYTE VARYING definition. For example

```
create domain IMAGE_DATA list of byte varying (32000);
```

Here it is assumed that all segments are a maximum of 32000 octets long. However, Rdb doesn't use this value for the AIP length because many different list segments from different tables may share the same logical area. In fact Rdb defaults the AIP length for list logical areas to 162 octets.

> ***Note**: in some earlier versions of Rdb the AIP length for all LIST's stored using the LIST storage map were using an incorrect AIP length of 5. Check your databases for this value as it means that only a page with less 5 octets free will be marked as FULL and will most likely cause extra I/O to check its fullness.*

Sorted Indices

Prior to Rdb release 4.1 all SORTED indices had their AIP length set to 215 (half the smallest Index node). In later versions the NODE SIZE supplied with the index definition of UNIQUE indices is used to set the AIP record length. This has greatly improved performance for unique indices. However, non-unique indices and indices which do not provide a NODE SIZE still retain the AIP record size of 215.

When an index is created with no STORE clause it is mapped to the default storage area (usually RDB$SYSTEM). If there already exists an index for the same table similarly mapped then it will share the same logical area and hence use the same AIP record length. This means that the specified NODE SIZE will not be used for the AIP length (although it is used for the index construction) and any specified THRESHOLDS are ignored in such cases.

> ***Note**: Oracle recommends that indices use an explicit STORE clause to map the index. The storage area named can be the default storage area if desired. The use of the STORE clause will ensure a separate AIP entry for each index and allow full use of the NODE SIZE and THRESHOLDS settings.*

In Oracle Rdb, sorted UNIQUE indices and all SORTED RANKED indices defined without a NODE SIZE clause will have a default set based on the index key size, either 430 or 860.

However, all SORTED indices allowing duplicates will default to an AIP record length of 215. This value was chosen as a good compromise between the size of the index node and the duplicate nodes. However, defining THRESHOLDS can greatly reduce the time and I/O required to search for free space for such an index.

Hashed Indices

Normally pages used by HASHED indices will have the small SYSTEM record, and hash bucket written before user data. Both these structure are variable in length. Therefore, tune the thresholds for the MIXED format area using the data row lengths (this assumes PLACEMENT VIA INDEX), or the Hashed Duplicates nodes if there are many duplicates.

A common practice is to use PLACEMENT VIA INDEX to mix hash bucket, data rows (possibly from multiple tables and multiple indices) and system records on the same page. If the sizes vary by a large amount then some THRESHOLD tuning should be used to guarantee good space utilization and therefore optimised I/O. Appropriately sized pages are essential to make this type of placement or clustering work effectively.

Compressed Rows in a Uniform Area

When storing a row in a UNIFORM area the compressed length is used to locate free space. However, once the non-compressed length (as stored in the AIP record length) is no longer available the page is marked as full. This enables the unused space on the page to be used as a reserve for row expansion.

However, in cases where the uncompressed length is large relative to the average compressed row size, there might be a lot of wasted space in the area. Appropriate choices for thresholds might allow the data to be more tightly packed and so ultimately reduce the number of pages fetched when reading this data.

Row Length Greater Than the Page Size Allows

What happens when the AIP record length indicates that the row (or index node) is longer than the storage area's MAX_NEW_SEG_LEN? In this case Rdb must fragment a full sized row during INSERT. Fragmentation involves storing a leading portion on the target page in the available

space with a pointer to the next segment on a different page. Special information is stored in each fragment to allow the components to be materialized when fetched.

If there are no thresholds defined Rdb makes the assumption that thresholds were defined in terms of the MAX_NEW_SEG_LEN.

T1 = T2 = T3 = 100 - (MAX_NEW_SEG_LEN * 100 / MAX_FREE_LEN)

Thus forcing them to appear as (1,1,1) for our area. This means that almost anything being stored on the page will cause the page to be set FULL. This is done because the AIP record length is assumed to be accurate for the stored row.

However, consider the case where the actual compressed length of the row is smaller than the MAX_FREE_LEN. After the row is stored, the database system detects that the page will no longer store a new row and sets it full. Thus, for any extra long row that compresses well, space in the database will be wasted. This problem can be overcome by specifying appropriate thresholds for the area.

A notable example of this can be seen in the Replication Option for Rdb. The RDB$CHANGES table is defined with one VARCHAR column of 10,000 octets. This system table is a journal of changes made to selected user tables that are used by the replication transfers scheduled against the database. By default the table is mapped to the RDB$SYSTEM area which often has a page size of 2 blocks. The RDB$CHANGES rows are always sized and compressed to fit the available free space. How well these rows compress is a function of the user data being journaled. Therefore, there may be wasted space on pages which store rows for the RDB$CHANGES table.

To correct this involves the same calculations discussed in this article. How well do the rows in RDB$CHANGES compress? As this is specific to the application and transfers defined on the database we will use RMU/ANALYZE to see what the correct average rows size is for RDB$CHANGES, we assume there exists some good sampling of rows in the table.

```
Logical area: RDB$CHANGES for storage area : RDB$SYSTEM
 Larea id: 50,  Record type: 28, Record length: 10022, Compressed

 Data records: 69, bytes used: 14645 (1%)
   average length: 212, compression ratio:  .02
```

Here we see that the average length is just 212 octets, so with appropriate threshold settings we can fit four of these rows on a page, and thus reduce the space required to store the Replication Option journal information, and also the I/O required reading those records when the transfer is in progress.

Repairing Thresholds

RMU/REPAIR can be used to change the AIP length or the THRESHOLDS in a logical area and recalibrate the SPAM thresholds. RMU Repair is an offline command, so any management will require additional planning.

$ RMU/REPAIR/INITIALIZE=LAREA=option.opt/SPAM

The OPTION.OPT file contains the specification of the logical area and the new length. For example:

```
jobs/areas=job_area_1/length=37/thresholds=(55,55,88)
```

In this example options file, JOBS is the logical area name (usually the name of the table or index), and the /AREAS qualifier identifies the physical area in which these changes are made, you may provide a list of areas if the table or index is partitioned across multiple storage areas.

The new AIP record length is specified using /LENGTH, and the new thresholds are specified using /THRESHOLDS. Please see the *Oracle Rdb RMU Reference Manual* for more information.

> ***Note**: The /SPAM qualifier is the default for RMU/REPAIR. These RMU/REPAIR options are not journaled to the AIJ (after image journals) so you must take a FULL database backup when performing these database restructuring. Both the target area(s) and the RDB$SYSTEM areas will be altered by this command.*

Starting Oracle Rdb V7.2.1 this functionality is superseded by a new simpler RMU/SET AIP command. This tool can be used online, the actions are journaled and the command accepts wildcard logical area names. Oracle recommends that this tool be used instead of RMU/REPAIR if possible. Please refer to the article *Guide to Database Performance and Tuning: Managing the AIP Structure* for more information on the RMU/SET AIP and RMU/SHOW AIP commands.

Threshold Tutorial

This tutorial looks at calculating the THRESHOLD for the EMPLOYEES and JOB_HISTORY rows. The aim is to tightly pack the data in the database (for archival purposes) and minimize the number of pages read when an employee's history is retrieved.

Some tools which can be used:

- RMU/DUMP/HEADER/OPTION=DEBUG
  In the output from this command can be found two pieces of information: MAX_NEW_SEG_LEN that is the maximum size of a new segment written to the page. This includes the overhead at the page level for the LDX/TDX vectors, and MAX_FREE_LEN which is the maximum amount of space available on the page after space for page header and page tail are subtracted from the page size.
- We can use the SQL functions shown below to calculate the length in octets of the interesting rows. This query gives us the raw size - adjusting for VARCHAR types, and adding overhead octets for the version number and null bit vector. We can then calculate the threshold for given page size and row length.

```
create module SYSTEM_QUERIES
    language SQL

function SHOW_ROW_LENGTH
        (in :rn RDB$OBJECT_NAME)
    returns integer
    comment is  'This function calculates the row length for the '
    /           'named table.';
    begin
    declare :rc, :column_sizes, :column_count integer default 0;
    declare :rdb_header constant integer default 5;
    declare :rdb_version constant integer default 2;

    -- collect some base statistics
    select sum(case when f.rdb$field_type = 37 -- VARCHAR type
                    then f.rdb$field_length + 2
                    else f.rdb$field_length
                end),
           max(rf.rdb$field_id)
    into :column_sizes, :column_count
    from rdb$relations r, rdb$relation_fields rf, rdb$fields f
    where r.rdb$relation_name = :rn
          and rf.rdb$relation_name = r.rdb$relation_name
          and rf.rdb$field_source = f.rdb$field_name
```

```
      and r.rdb$view_blr is null;   -- exclude VIEWS

if :column_sizes is null
then
    -- the table was not found
    SIGNAL ('22023');
end if;

trace 'Using column lengths: ',
      coalesce(:column_sizes, -1), ' and column count: ',
      coalesce(:column_count, -1);

-- Now calculate the full row length
--  + rdb_header is the fixed size row header
--  + rdb_version is the SMALLINT version number
--          for the stored row
--  + column_sizes is the sum of all column sizes
--  + column_count is used to calculate the length
--          of the NULL bit vector (rounded up to next byte)
return :rdb_header + :rdb_version + :column_sizes +
       CAST(TRUNC ((:column_count+7)/8) as INTEGER);
end;

function SHOW_THRESHOLD
    (in :free_len integer, in :t1_len integer)
returns integer
comment is  'This procedure calculates the threshold for '
/           'a given length';
begin
-- account for ldx/tdx index entries for the row
-- also allow for rows longer than the page size
declare :req_len integer = LEAST (:t1_len + 8, :free_len);
declare :end_pct integer = (:req_len * 100 / :free_len);

-- trace the end percent calculated and also the range
-- sometimes the rounding up causes the threshold to be lower than it could
-- be, so check and use the appropriate percentage
trace 'ep=',:end_pct,
      ' v1=', cast(:free_len * :end_pct / 100 as integer(2)),
      ' v2=', cast(:free_len * (:end_pct - 1) / 100 as integer(2));

return 100 - case
                when (:free_len * (:end_pct - 1) / 100) >= :req_len
                then -- correct for rounding error
                    (:end_pct - 1)
                else
                    :end_pct
```

```
                end;
    end;

end module;
```

These can be used to report the row lengths and threshold for the tables of interest:

```
SQL> -- These are the thresholds for the examples
select 'EMPLOYEES  ',
cont>         SHOW_ROW_LENGTH ('EMPLOYEES'),
cont>         SHOW_THRESHOLD (964, SHOW_ROW_LENGTH ('EMPLOYEES'))
cont> from rdb$database;

 EMPLOYEES             126            86
1 row selected
SQL> select 'JOB_HISTORY',
cont>         SHOW_ROW_LENGTH ('JOB_HISTORY'),
cont>         SHOW_THRESHOLD (964, SHOW_ROW_LENGTH ('JOB_HISTORY'))
cont> from rdb$database;

 JOB_HISTORY            42            95
1 row selected
SQL>
SQL> -- These are the thresholds for a duplicates index with node size 430
SQL> select SHOW_THRESHOLD (964, 112), SHOW_THRESHOLD (964, 430)
cont> from rdb$database;

          88            55
1 row selected
SQL>
```

Extra Notes

1. There are two storage algorithms used by Rdb, one for data stored in UNIFORM areas when no thresholds are specified, and one for MIXED and UNIFORM areas when thresholds are specified.

2. Thresholds can be defined for MIXED storage areas, or storage maps and indices mapped to UNIFORM areas.

3. The threshold values are based on the ratio of the row size to the page size. The thresholds may no longer be valid if the page size changes (such as can be done using IMPORT, RMU/MOVE_AREA or RMU/RESTORE), or if the row size changes (due to ALTERing the table and adding or dropping columns, or changing the data type).

4. The THRESHOLDS clause can be used at the storage map or index level. In this case it is applied to each storage area. However, it can also be applied to some or all of the storage areas specified by the IN clauses. This overrides any default threshold setting for the map. You would probably only do this if you knew that data written to a particular partition had different characteristics. For example, product information stored prior to 1995 did not include nutritional information as required by the government, so this data compresses more than more recently stored data that contains this extra information. Thus the thresholds can be tuned to specific requirements of the partition.

5. Data can be vertically as well as horizontally partitioned. That is, sets of columns from the table can be mapped to different storage areas. Therefore, you can specify a default THRESHOLD for the map, default thresholds for the vertical partitions (which will each have different lengths), or thresholds for the storage areas.

6. The thresholds are stored in the AIP structure in the database, use RMU/SHOW AIP or RMU/DUMP/LAREA=RDB$AIP to display this structure.

7. When RMU/EXTRACT displays the storage map or index it will apply the THRESHOLD clause to each UNIFORM storage area named in the map. i.e. it no longer knows that the thresholds were originally specified just once as a map default.

Frequently asked Questions

What does THESHOLDS ARE (0,0,0) mean? Logical areas within **page format is uniform** storage areas may have this value. In such cases Oracle Rdb derives the threshold percentages at runtime using the free space on the page and the record length stored in the AIP. Many customer databases successfully use this setting to manage their data. It has the benefit of adapting to changes in the page size after an RMU/RESTORE or RMU/MOVE_AREA command.

What happens if only a partial THRESHOLD clause is specified, such as THRESHOLDS ARE (60)? Any missing threshold values are filled with 100%. Therefore, (60) is equivalent to (60,100,100).

What happens when I use RMU/MOVE_AREA and enlarge the page size for the area? This can be a problem because the thresholds are proportional to the page size. That is, the percentages represent rations of the row size to the original page length. Therefore, these ratios will no longer represent an accurate description of the table rows. For **mixed** areas the /THRESHOLD qualifier on RMU/MOVE_AREA should be used to redefine the thresholds - relative to the new page size. For **uniform** areas the thresholds should be redefined using either RMU/SET AIP along with /REBUILD_SPAMS, or in older versions RMU/REPAIR.

How are INFORMATION tables added to my database? The database administrator can add these tables by executing the script in RDM$DEMO:INFO_TABLE.SQL. These table definitions may not be modified, nor can these tables be updated by SQL. They provide a table-like interface to several Rdb internal data structures.

Oracle Rdb
Guide to Performance and Tuning: Space Management and Thresholds
August 2008
Original publication: 29 September 1997

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.