DB2 V9.7 for LUW New Feature: Index Compression

Applies to:

All SAP releases running on DB2 for Linux, UNIX, and Windows Version 9.7 or higher.

Summary

This article describes the new feature of index compression available as of DB2 V9.7 and how to use it in an SAP environment. This new feature provides additional storage savings by compressing table indexes.

Author: Marianne Nesiem

Company: IBM Canada

Created on: 28 July 2009

Updated on: June 2015

Author Bio

Marianne Nesiem is a member of the IBM SAP Integration and Support Centre at the IBM Toronto Lab. Her current activities include testing and certification of SAP with every new release of DB2 for Linux, UNIX, and Windows and providing assistance to customers running SAP in a DB2 LUW environment with problem analysis and troubleshooting. She is also a customer advocate, providing custom-tailored support for large customer accounts running SAP and DB2. She has been the sole author of multiple White papers on the topic of SAP and DB2 LUW and has been involved in many customer briefings and presentations and has chaired multiple workshops on the same topic. She is a DB2 Certified Solutions Expert and a UNIX systems administrator, and has experience in application development on Java for databases and other applications.

Table of Contents

1. Introduction

More and more organizations are facing increasing storage cost as more data is being generated and more existing data is being retained for longer periods of time. Since more disk space is consumed, the need for innovations to reduce storage and total cost is becoming crucial for business success.

Data compression was introduced in DB2 V9.1 to allow table data to be compressed and reduce storage space. Customers have seen great success using data compression and many have achieved very high compression rates for some of their tables.

In many database environments, it is common to have many very large indexes that are defined on a large table. Similar to data objects, index objects can now be compressed to further reduce disk consumption and storage cost with minimal effect on the system performance for most cases.

2. How Does Index Compression Work

To compress index objects, multiple compression algorithms can be used. The database manager chooses the best compression algorithms based on the data layout to minimize the storage space.

Contrary to data compression, no dictionary is required to compress indexes as the database manager uses fixed algorithms.

Only data in leaf pages is compressed to reduce CPU and I/O cost to compress, uncompress and fetch the data. A major benefit of compressing leaf pages is that there are less frequent page splits and fewer index levels. Because the number of index levels is reduced, an index search starting from the root page requires fewer I/O corresponding to the number of index levels. Similarly an index scan requires fewer I/O corresponding to the total number of leaf pages. Furthermore, more index keys can be cached in the buffer pool. The buffer pool hit ratios increase and the index page physical reads and writes decrease. On the other hand, since the index keys are now compressed, extra CPU is needed to uncompress the index keys back to the uncompressed format before returning the result back to the users, or to compress the index key to the compressed format before updating an index page.

The degree of compression achieved will vary based on the type of index, as well as the data the index contains. For example, the database manager can compress an index with a large number of duplicate keys by storing an abbreviated format of the record identifier (RID) for the duplicate keys. In an index where there is a high degree of commonality in the prefixes of the index keys, the database manager can apply compression based on the similarities in prefixes of index keys.

The following section describes the different compression algorithms being used.

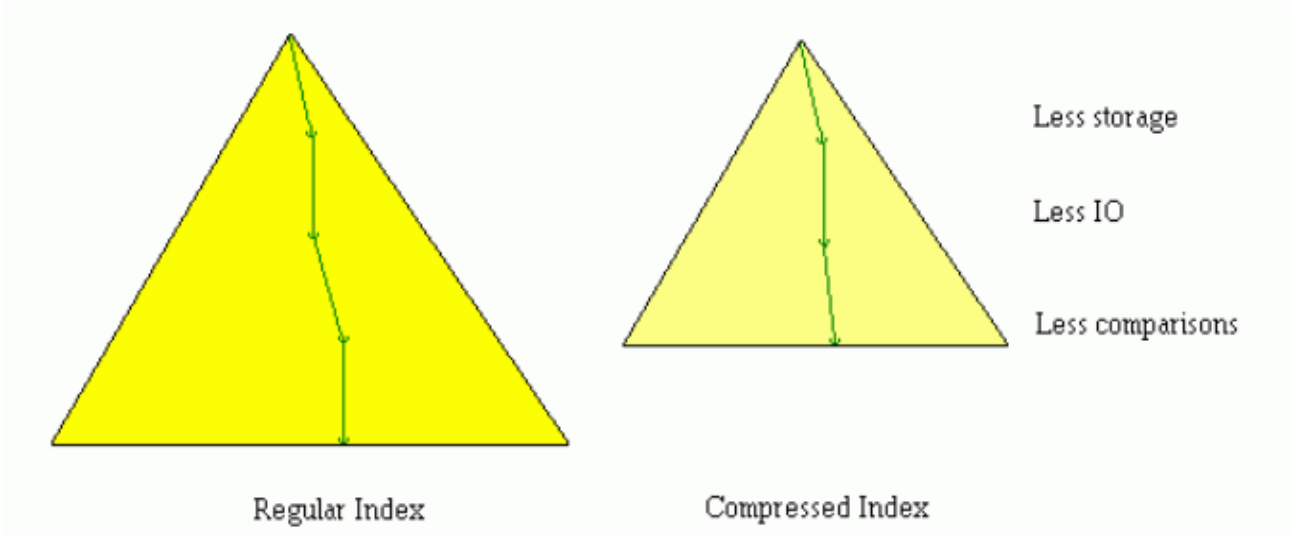
2.1 Compression Algorithms

If index compression is enabled, DB2 considers multiple compression algorithms for index pages:

1. Variable Slot Directory

Each index page begins with a page header, followed by a slot directory. Each two byte entry in the slot directory corresponds to a different index key on the page. The entry itself is the byte-offset into the index page where the index key begins.

In previous releases, a slot directory was pre-allocated and fixed based on the minimum index key size. Normally the slot directory occupies only a little space. However, if an index contains many duplicates or a variable length key part where the actual size of the values are longer than its minimum size, much of the pre-allocated slot directory will not be used. Some slots may have been unused and the unused slot directory space may prevent the insertion of new records on the index page. In DB2 V9.7, a slot directory is dynamically adjusted in order to fit as many index keys into an index page as possible. A slot directory will not be fixed or pre-allocated in a compressed index; as a result, every byte on an index page is used wisely.



2. RID List Compression

An index key contains a key value and a key data list. A key data contains a Record ID (RID) and its flags. DB2 keeps only the RIDs for each duplicate entry in an index page. The RIDs in a RID list are sorted. For indexes on tables in LARGE and TEMPORARY table spaces, the size of a RID is 8 bytes for partitioned tables and 6 bytes for other tables. For indexes on tables in REGULAR tablespaces, the size of a RID is 6 bytes for partitioned tables and 4 bytes for other tables. Instead of saving the full version of a RID, we can save some space by storing the delta between two RIDs. The delta is further compressed if some encoding algorithms are applied. The encoded delta can be as small as 1 byte. If the indexes have low cardinality and a high cluster ratio, the savings will be significant.

Example:

Suppose we have 6 records stored on page 4 for a regular table in a LARGE tablespace.

In an uncompressed index, DB2 saves the following RIDs:

<00 00 00 04, 00 00>,
<00 00 00 04, 00 01>,
<00 00 00 04, 00 02>,
<00 00 00 04, 00 03>,
<00 00 00 04, 00 04>,
<00 00 00 04, 00 05>
<00 00 00 04, 00 06>

In a compressed index, DB2 compresses the RID list:

<00 00 00 04, 00 00>,
<1>,
<1>,
<1>,
<1>,
<1>,
<1>

The following image shows a comparison of the uncompressed and compressed index page:

3. Prefix Compression

Because the key values are sorted on index pages, very often the key values of two adjacent keys are very similar on an index with many different entries. It is common for two adjacent keys to share the same prefixes. During index creation or insertion, DB2 compares the new key with adjacent index keys and finds the longest common prefixes between them. If the page is full, DB2 tries to find the optimal prefixes to use for each page by constructing a representation of the prefixes using some metadata structure. Each prefix is then compared with neighboring keys to find an optimal prefix that uses less space and update the metadata accordingly.

If the cardinality of the initial portion of the index keys is low relative to the number of index records, the prefix compression savings will be significant.

Example:

If we have the following uncompressed index:

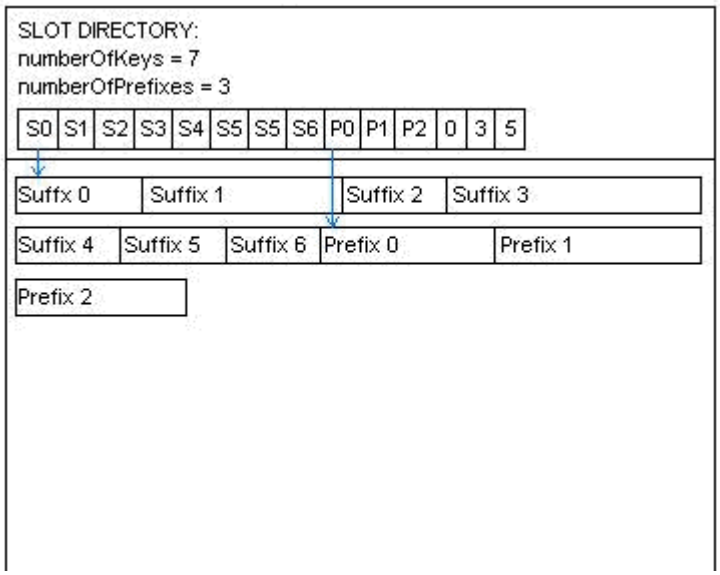
(‘a’, ‘a’, ‘a’),
(‘a’, ‘a’, ‘b’),
(‘a’, ‘a’, ‘c’),
(‘b’, ‘b’, ‘b’),
(‘b’, ‘b’, ‘c’)

DB2 keeps (‘a’, ‘a’) and (‘b’, ‘b’) as the common prefixes in a compressed index. In addition, DB2 knows the first three keys use prefix (‘a’, ‘a’) and will keep (, , ‘a’), (, , ‘b’), (, , ‘c’). Similarly, DB2 knows the fourth and fifth keys use prefix (‘b’, ‘b’) and only keeps (, , ‘b’) and (, , ‘c’) for the fourth and fifth keys.

After compression, the index will look similar to the following:

(‘a’, ‘a’, ‘a’),
(, , ‘b’),
(, , ‘c’),
(‘b’, ‘b’, ‘b’),
(, , ‘c’)

The following image shows how the page looks after prefix compression:



DB2 uses common prefix compression and RID list compression for index leaf pages, which represent the majority of the index space. In some cases, the database manager might choose to use more than one algorithm at the same time to achieve the highest compression rate.

3. How to Enable Index Compression

By default, new indexes that were created on row-based compressed tables are compressed. If a table is created or altered with the COMPRESS attribute set to 'Y', both data and indexes on that table are set for compression and any new indexes created on that table are compressed by default. At that point, it is not required to explicitly compress the indexes upon creation.

As of DB2 V9.7, you can now also compress temporary tables; hence indexes on temporary tables can also be compressed.

3.1 Verify the State of the Indexes

There are two new table functions that you can use to retrieve more detailed information on indexes as described in the following sections.

3.1.1 ADMIN_GET_INDEX_INFO

This new table function displays detailed information about the indexes, such as the compression information as well as the logical and physical size of the indexes.

Syntax

>>-ADMIN_GET_INDEX_INFO--(--objecttype--, --objectschema--, --objectname--)-><

Where:

Objecttype is 'T' for table and 'I' for index.

Objectschema is the name of the schema where the object belongs

Objectname is the name of the object

ADMIN_GET_INDEX_INFO returns, for example, the following important values:

Value	Description
COMPRESS_ATTR	The following values are returned: • *Y* If index compression is enabled • *N* If index compression is not enabled
INDEX_COMPRESSED	The following values are returned: • *Y* If the index is compressed • *N* If the index is not compressed
INDEX_OBJECT_L_SIZE	Returns the logical size of the index object in KB. Note that the same size (which is equivalent to the total size) will be returned for all indexes
INDEX_OBJECT_P_SIZE	Returns the physical size of the index object in KB. Note that the same size (which is equivalent to the total size) will be returned for all indexes

Example:

```
%  db2 " select indname , compress_attr, index_compressed, index_object_p_size from table
(admin_get_index_info ('T', 'SAPLRP', 'MARA')) as t"

INDNAME   COMPRESS_ATTR INDEX_COMPRESSED INDEX_OBJECT_P_SIZE
------------------------------------------------------------------------------------------------------------
MARA~0                Y                 N                   5344
MARA~A                Y                 N                   5344
MARA~L                Y                 N                   5344
MARA~O                Y                 N                   5344
MARA~PMA              Y                 N                   5344
MARA~T                Y                 N                   5344
  6 record(s) selected.
```

The previous example shows that all the indexes belonging to table MARA are set for compression but have not yet been compressed. Note that INDEX_OBJECT_P_SIZE is identical for all indexes as it represents the total size of all the indexes on that table.

3.1.2 ADMIN_GET_INDEX_COMPRESS_INFO

This new function returns information that is specific to index compression, such as the compression savings estimate (before the index is compressed) and the index compression statistics (after the index is compressed) from the system catalog table.

Syntax

--ADMIN_GET_INDEX_COMPRESS_INFO--(--objecttype--,-- objectschema--,--objectname--,--dbpartitionnum--,--datapartitionid--)

Where:

- Objecttype is ‘T’ for table and ‘I’ for index.
- Objectschema is the name of the schema where the object belongs
- Objectname is the name of the object
- Dbpartitionnum is the database partition number. To specify that the data is requested for all partitions use the value -2. For a non-partitioned system, use the value 0.
- Datapartitionid is the database partition ID. To specify that the data is requested for all partition, use the value -2. For a non-partitioned system, use the value 0.

ADMIN_GET_INDEX_COMPRESS_INFO returns, for example, the following important values:

Value	**Description**
COMPRESS_ATTR	The following values are returned: • *Y* If index compression is enabled • *N* If index compression is not enabled
INDEX_COMPRESSED	The following values are returned: • *Y* If the index is compressed • *N* If the index is not compressed

PCT_PAGES_SAVED	Before an index is compressed, it provides an estimated percentage of leaf pages saved based on the existing index keys. After index compression, this parameter provides the percentage of savings after compression. To get an accurate value for PC_PAGES_SAVED, you should perform an RUNSTATS on that table. If the information is unavailable or if the statistics on the table need to be refreshed, this parameter returns *-1*.

If this function is executed in a newly upgraded database, the value for COMPRESS_ATTR as well as INDEX_COMPRESSED will be N by default and the value for PCT_PAGES_SAVED will be -1 until the runstats command is executed on the table.

Example:

```
% db2 “select INDNAME, COMPRESS_ATTR, INDEX_COMPRESSED,  PCT_PAGES_SAVED from table
( admin_get_index_compress_info ('T', 'SAPLRP', 'MARA', 0, 0)) as t”


INDNAME    COMPRESS_ATTR   INDEX_COMPRESSED PCT_PAGES_SAVED
---------------------------------------------------------------------------------------------------------
MARA~0                     N        N                31
MARA~A                     N        N                71
MARA~L                     N        N                57
MARA~O                     N        N                71
MARA~PMA                   N        N                71
MARA~T                     N        N                57
  6 record(s) selected.
```

Note that in this example, all existing indexes do not have index compression enabled. The value of PCT_PAGES_SAVED, which shows the estimated total savings after compression, is displayed per index in this case.

3.2 How to Activate Index Compression

You can explicitly activate or deactivate index compression on a table using the COMPRESS clause of the CREATE INDEX and ALTER INDEX statements

3.2.1 Activating Index Compression for New Indexes:

To activate compression on a new index, use the CREATE INDEX SQL statement:

Syntax

>>-CREATE INDEX--*index-name* ----------→
...
>--+---------+----------->
'--**COMPRESS**--+-**NO** -+-'
'-**YES**-'
...

Example:

```
% db2 create index MARA~6 … compress yes
```

The value of the COMPRESS attribute specifies whether index compression is enabled. To activate index compression on a new index, the value is set to *Yes*- To disable index compression for a new index, the value is set to *No*. Note that if the index is created in a table where data compression is enabled, there is no need to set the compression attribute because it is set to *YES* by default as explained earlier.

3.2.2 Activating Index Compression for Existing Indexes:

For existing indexes, you can use the ALTER INDEX SQL statement as follows:

Syntax

>>-ALTER INDEX--*index-name* ---------- >>

>--COMPRESS--+-NO -+-----------| >
'-YES-'

Example:

```
% db2 alter index MARA~0 … compress yes
```

To activate or deactivate index compression on an existing index , the value of the COMPRESS attribute again is set to *YES* or *NO*.

Another method to modify existing indexes other than using the ALTER INDEX SQL statement is to drop the index and re-create it. In addition, based on the state of the table, the index is created as compressed or uncompressed. Both methods are as effective and use approximately the same amount of time to complete.

To activate compression on multiple indexes, you can use a script similar to the following example:

Example:

To activate index compression for all indexes belonging to table MARA, issue the following command:

```
% db2 -x "select 'alter index ' || CHR(34) || RTRIM(tabschema) || CHR(34) || '.' || CHR(34) || INDNAME || CHR(34) || ' compress  yes;' from syscat.indexes where tabname='MARA'" > alter_indexes_mara.clp
```

A CLP script is generated that you can execute as follows:

```
% db2 –tvf alter_indexes_mara.clp
```

To verify that index compression has been activated, you can also use the admin function ADMIN_GET_INDEX_COMPRESS_INFO.

Example:

```
db2 => select INDNAME, COMPRESS_ATTR, INDEX_COMPRESSED,  PCT_PAGES_SAVED from table
( admin_get_index_compress_info ('T', 'SAPLR1', 'MARA', 0, 0)) as t

INDNAME    COMPRESS_ATTR   INDEX_COMPRESSED PCT_PAGES_SAVED
---------------------------------------------------------------------------------------------------------

MARA~0           Y        N                31

MARA~A           Y        N                71

MARA~L           Y        N                57

MARA~O           Y        N                71

MARA~PMA         Y        N                71

MARA~T           Y        N                57


  6 record(s) selected.
```

Note that in this example the COMPRESS_ATTR value has been changed to *Y*. The value of INDEX_COMPRESSED remains *N* until a table or index reorganization is performed.

3.3 Compressing Indexes

Once you activated compression using one of the previous methods, the indexes are ready to be compressed. To compress the indexes, the REORG command is used. The reorganization can be performed online or offline.

Syntax

>>-REORG
>----TABLE --- *table-name* --| INDEX--*index-name* ---------- >>
> ----INDEXES ALL FOR TABLE – *table-name* ---------- >>
…

***Examples*:**

```
% db2 reorg table MARA index MARA~T
```

In the following example, only index MARA~T is compressed:

```
% db2 reorg indexes all for table MARA
```

In the following example, all the indexes belonging to the table MARA are compressed:

```
% db2 reorg indexes all for table MARA inplace
```

In this example, all indexes in table MARA are compressed while allowing user access.

If you want to compress both data and indexes at the same time, you can run the REORG TABLE command (after setting the compression attribute to Y on both indexes and data). Note that in this case, the REORG operation must be performed offline.
If you want to drop and re-create the indexes as compressed, you do not have to perform the REORG operation.

Example:

```
% db2 reorg table MARA
```

To run the REORG command on multiple tables, you can use a script similar to the example used in the previous section.

Example:

To compress all the indexes in your database, issue the following command:

```
% db2 -x "select 'reorg indexes all for table ' || CHR(34) || RTRIM(tabschema) || CHR(34) || '.' || CHR(34) || TABNAME || CHR(34) || ';' from syscat.tables" > reorg_indexes.clp
```

A CLP script is generated that you can be executed as follows:

```
% db2 –tvf reorg_indexes.clp
```

To verify that the indexes have been compressed, use the same admin function, ADMIN_GET_INDEX_COMPRESS_INFO. To receive an accurate value for PCT_PAGES_SAVED, you have to perform a RUNSTATS.

Example:

```
db2 => select INDNAME, COMPRESS_ATTR, INDEX_COMPRESSED,  PCT_PAGES_SAVED from table
( admin_get_index_compress_info ('T', 'SAPLR1', 'MARA', 0, 0)) as t

INDNAME    COMPRESS_ATTR   INDEX_COMPRESSED PCT_PAGES_SAVED
---------------------------------------------------------------------------------------------------------

MARA~0                          Y       Y     31

MARA~A                          Y       Y     71

MARA~L                          Y       Y     57

MARA~O                          Y       Y     71

MARA~PMA                        Y       Y     71

MARA~T                          Y       Y     57

  6 record(s) selected.
```

NOTE:

To deactivate index compression, you can either re-create the indexes without compression or alter the indexes using the following SQL statement:

ALTER INDEX --- index-name –COMPRESS NO

Afterwards, you should perform an index REORG to disable index compression.

4. Restrictions

Index compression is not supported for the following:

- Catalog indexes
- MDC block indexes
- XML path indexes and meta indexes
- Index specifications

5. Case Study

A case study was performed in the lab on an SAP ERP customer database used for testing. The following table outlines the results:

	Original Size	After Data and Index Compression	Savings
DATA_P_SIZE	567187840	158071552	-72%
INDEX_P_SIZE	130968320	66381600	-49%
Total Size	698156160	224453152	-68%

From the table above, we notice that the total data size has decreased by 72% after data compression and the total index size has decreased by 49% after index compression. The total database size has decreased by 68% after compressing the data and indexes

Related Content

- SAP Note 1555903 - DB6: Supported DB2 Database Features

Copyright