Best Practices: Using the DB2 Compression Feature in an SAP Environment

Applies to:
All SAP releases on DB2 for Linux, UNIX, and Windows Version 9 and higher.

Summary
Disk storage can often be the most expensive components of a database solution. IBM DB2 compression features help reduce space requirements dramatically, resulting in lower storage cost and improved I/O performance. This article explains different compression features delivered in DB2 for LUW V9.1, V9.5, and V9.7 and provides general guidelines on how to use these features efficiently in an SAP environment.

Compression features that were delivered with DB2 10.1 and higher, such as adaptive compression and log archive compression, are not in the scope of this paper.

Author: Lili Zhang
Company: IBM
Created on: July 2010
Updated on: June 2015:

Author Bio
Lili Zhang is a member of the IBM SAP Integration and Support Centre at the IBM Toronto Lab. Her current activities include testing of SAP applications with DB2 LUW and helping customers with problem analysis and troubleshooting. She is also a customer advocate, providing support for large customer accounts running SAP and DB2.
Table of Contents

1.1 Row compression.................................................................3
1.2 Index compression.............................................................6
2.1 DB2 tools to estimate the compression ratio ................................8
  2.1.1 DB2 INSPECT utility ......................................................8
  2.1.2 ADMINTABCOMPRESSINFO view and ADMIN_GET_TAB_COMPRESS_INFO table function ..........9
  2.1.3 ADMINTABCOMPRESSINFO view and ADMIN_GET_TAB_COMPRESS_INFO_V97 table function ..........10
  2.1.4 ADMIN_GET_INDEX_COMPRESS_INFO table function (available in DB2 V9.7) .......................11
2.2 SAP tools to estimate the space savings..................................12
2.3 Calculating the number of tables to be compressed to achieve desired storage savings ...............13
2.4 Tables where compression is less beneficial ................................14
3.1 Enabling the tables for large RIDs (Optional) ................................15
3.2 REORG consideration ..........................................................16
  3.2.1 Reorg using a temporary tablespace ................................16
  3.2.2 Reorg within the same DMS tablespace ................................17
3.3 Managing tablespace fragmentation ..........................................18
4.1 Installation tools ....................................................................20
4.2 Enabling compression for existing systems ................................20
  4.2.3 DB6CONV.................................................................22
  4.2.4 Compression support in BW system .....................................23
  4.2.3 Compressing AS Java tables .............................................24
4.3 Migration tools .......................................................................24
  4.3.1 R3load....................................................................24
5.1 Monitoring the compression quality ...........................................26
  5.1.1 Evaluating compression quality .........................................26
  5.1.2 Rebuilding the Compression Dictionary .................................26
5.2 Reclaiming space ...................................................................27
5.3 Backup Performance ...............................................................27
1. Introduction

Storage cost is one of the major considerations in today’s database solution. Since V8, DB2 has delivered several compression features to help customers better manage their storage growth. One of the most effective compression solutions is row compression which was introduced in DB2 V9.1. It significantly reduces the space requirement by eliminating repetitive data in the tables. In DB2 V9.7, index compression was introduced to further cut down the space consumption for index objects. For a typical SAP ERP system, customers have experienced space savings of more than 50%, along with better application response times.

DB2’s compression features are tightly integrated into SAP software. Customers can easily get compression estimates and enable compression through different SAP tools. This paper explains different compression features delivered in DB2 for LUW V9.1, V9.5, and V9.7, and how to make best use of them in an SAP environment. Additional compression features that were delivered with DB2 10.1 and higher, such as adaptive compression and log archive compression, are not in the scope of this paper.

1.1 Row compression

Row compression is also referred to as “deep compression”. It uses a dictionary-based LZ (Lempel-Z) algorithm to compress data records. When row compression is activated, DB2 first scans the table for repetitive patterns. These patterns could span across multiple columns, or reside within a substring of a column. After analyzing the data, it creates a compression dictionary that maps repeated byte patterns to much smaller symbols. These symbols then replace the longer byte patterns in the table rows.

The following diagram demonstrates how it works:

```
Uncompressed rows
Franklin 211 Grant Road Pierro LA 24554
Franklin 8976 Link Road Pierro LA 24554

(1) (2) (3)
(1) (4) (3)

Compressed rows
Compression dictionary
(1) Franklin
(2) 211 Grant Road
(3) Pierro, LA, 24554
(4) 8976 Link Road
```

The degree of compression depends on the data itself and the quality of the data dictionary. The compression dictionary is physically stored along with the table data and loaded into the database heap when the table is accessed. It is static and remains in the table even if all the data is deleted. The maximum dictionary size is 150 KB whereas in a typical SAP system, the average size of the dictionary is 75 KB. In a multi-partitioned environment, a dictionary is created for each data partition.

With DB2’s row compression feature in V9.1 and V9.5, only data stored in base table rows can be compressed. Indexes, LONGs and LOBs are not eligible for compression. With DB2 V9.7, you can use the LOB inlining feature to include small LOBs in the base table row so that they can be compressed with row compression. In addition, DB2 writes log records in compressed format as well.

**Enabling Row Compression**

In order to use row compression for the table, the following two prerequisites must be fulfilled:

1. The `compress` attribute for the table is set to YES.
By default, the compression flag is set to OFF, and it can be enabled with the CREATE TABLE or ALTER TABLE command using the COMPRESS clause.

Example:

```
% db2 create table MARA... compress yes
% db2 alter table MARA ... compress yes
```

2. The compression dictionary is built.

The classic way to build a compression dictionary is through an offline REORG operation. It first constructs a compression dictionary and then compresses the data in the table. Alternatively you can use the INSPECT utility with the ROWCOMPESTIMATE clause to create the compression dictionary. Any data updated or inserted into the table after dictionary creation will be compressed.

- **Offline REORG**

In DB2 V9.1, the REORG TABLE command has been enhanced with two options related to compression: KEEPDICTIONARY and RESETDICTIONARY. KEEPDICTIONARY is the default setting. If a compression dictionary already exists on the table, the KEEPDICTIONARY option will reorganize the table based on the existing dictionary and the RESETDICTIONARY will replace the existing dictionary with a new one and then compresses the table. If the table is enabled for compression and does not have a dictionary, the KEEPDICTIONARY option will create the compression dictionary if the size of the table is big enough (about 1-2 MB) and if there is enough data in the data. On the other hand, the RESETDICTIONARY option will create the dictionary and compress the table as long as there is at least one row in the table. If the COMPRESS attribute has been turned off, the RESETDICTIONARY option will uncompress the data and also remove the existing dictionary from the table.

- **DB2 INSPECT utility**

The new clause ROWCOMPESTIMATE was added to the INSPECT utility to provide a compression estimate function for the table. When this utility is called, it examines the existing data and creates a compression dictionary which is used to estimate the compression ratio. If the COMPRESS attribute for the table is set to YES and the table doesn't have a dictionary, the dictionary is saved and stored with the table.
**Automatic Dictionary Creation (ADC) as of DB2 V9.5 and higher**

Starting with DB2 V9.5, a compression dictionary is created automatically as part of data growth operations (insert, import, load, and redistribute) when the following three conditions are met:

1. The COMPRESS attribute for the table is set to YES.
2. No compression dictionary exists on the table.
3. The table size grows beyond the threshold (about 2 MB), so that there is sufficient data to construct a compression dictionary.

When ADC is triggered, DB2 scans the table and builds a compression dictionary synchronously. There may be a slight performance impact when the threshold is reached and the dictionary is being created. After the dictionary is created, subsequent data populated into the table is subject to be compressed. The existing data is not compressed unless a record gets changed.

The following diagram demonstrates how it works for a newly created table:

For existing tables with a size greater than the threshold, a compression dictionary is automatically created when a new page is allocated to the table after the compression flag was turned on. The amount of data to be scanned is limited to the first portion of the table (about 2 MB).

**Note:** ADC reduces the need to create a dictionary manually. However, as ADC constructs the dictionary from a subset of data, the quality of the dictionary may not be optimal for large tables.
Performance Impact and Resource Requirement

In addition to the space savings, compression can also improve system performance, especially in I/O bound systems. Queries against compressed data can be performed with fewer I/O operations because each read from disk brings in more data. The bufferpool hit ratio can also increase as more data can be cached in the bufferpool. However, there is a tradeoff in the form of CPU cycles and memory requirements to compress and decompress the data.

- **CPU overhead**

  On average, SAP systems have shown an increase of 0%-10% in CPU usage on a compressed system. In most cases, the extra CPU cycles used for compression can be offset by the efficiency gained through fewer I/O operations. However, if the system is already CPU bound, it may be necessary to increase CPU capacity in order to maintain desirable performance results.

- **Memory consumption**

  - **DBHEAP**
    
    When a compressed table is first accessed, its dictionary is loaded in the database heap and it remains there until the database is deactivated. On average, 75 KB is needed for each compression dictionary in an SAP system. SAP recommends setting database parameter DBHEAP to automatic, and customers may notice a slight increase in the DBHEAP memory usage for systems with compressed tables.

  - **UTIL_HEAP_SZ**
    
    During dictionary creation, DB2 requires up to 10 MB memory from the utility heap in order to hold the data records that are sampled by the algorithm. Once the dictionary is built, this memory will be released.

1.2 Index compression

With DB2 V9.7, index objects can also be compressed to further reduce the storage requirement for index pages. When index compression is enabled, DB2 chooses one of the following algorithms to compress the index pages:

- Variable slot directory
- RID list compression
- Prefix compression

You can find a detailed description of different algorithms used for index compression in the article “DB2 for LUW New Feature: Index Compression” on SCN.

For index compression, our tests showed a reduction of up to 50% in the overall index size. In addition to space savings, index compression also increases disk I/O throughput and bufferpool quality. Depending on the number of indexes compressed and the type of workload, SAP customer systems have shown a 0-5% increase in CPU cycles required for compressing and decompressing the indexes.

**Note:** Index compression is not supported for MDC block indexes.

Enabling Index Compression

- **Activating compression on new indexes**
If an index is created on a compression-enabled table, it will be compressed by default. No additional steps are needed. Compression-enabled table refers to any table with the COMPRESS attribute set to YES. The data in the table does not have to be compressed.

If the table is not enabled for compression, you can use the COMPRESS clause on the CREATE INDEX statement to explicitly create the index in compressed format.

Example:

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

### Activating compression on existing indexes

For existing indexes, you can use the ALTER INDEX command to set the COMPRESS attribute to YES and perform an online or offline REORG on the indexes for index compression.

Example:

```
% db2 alter index MARA~0 … compress yes
% db2 reorg indexes all for table MARA
```

**Note:** When you enable row compression with the ALTER TABLE command, the COMPRESS attribute for the indexes defined on the table does not get changed. You need to set the COMPRESS attribute for each index separately.

The REORG TABLE command can be used to compress data and indexes at the same time. In this case, the REORG has to be run offline.
2. Choosing Candidates for Compression

Before enabling compression, you should get a compression estimate to decide if the table is a suitable candidate for compression. In this section, we will explain different tools available from DB2 and SAP that provide estimates on the space savings with compression.

2.1 DB2 tools to estimate the compression ratio

2.1.1 DB2 INSPECT utility

With DB2 V9.1, the INSPECT utility can be used to get compression estimates with the ROWCOMPESTIMATE clause. This tool takes a sample of the table data, builds a dictionary from it, and estimates how much space can be potentially saved based on the dictionary. The compression result is saved in a binary output file in the db2dump directory. The DB2INSPF utility can then be used to convert the output into readable format.

You will get the following information in the INSPECT report:

- Percentage of bytes saved from compression
- Percentage of pages saved from compression
- Percentage of rows ineligible for compression due to small data size (applicable in DB2 V9.1 only)
- Compression dictionary size
- Expansion dictionary size

If a table's COMPRESS attribute is set to YES and no dictionary exists on the table, the dictionary created by the INSPECT utility will be saved and stored together with the table object. This provides an alternative to the offline REORG procedure to build the compression dictionary.

Example:

```
db2lrp> db2 inspect rowcompestimate table name cosp schema saplrp results keep cosp_estimate
DB20000I  The INSPECT command completed successfully.
db2lrp> cd /db2/LRP/db2dump
db2lrp> db2inspf cosp_estimate cosp_estimate.out
db2lrp> cat cosp_estimate.out

DATABASE: LRP
VERSION : SQL09071
2009-12-16 16:52:29.813034
Action: ROWCOMPESTIMATE TABLE
Schema name: SAPLRP
Table name: COSP
Tablespace ID: 10  Object ID: 1892
Result file name: cosp_estimate

Table phase start (ID Signed: 1892, Unsigned: 1892; Tablespace ID: 10) : SALRP.COSP
  Data phase start. Object: 1892  Tablespace: 10
  Row compression estimate results:
  Percentage of pages saved from compression: 87
  Percentage of bytes saved from compression: 87
  Compression dictionary size: 37760 bytes.
  Expansion dictionary size: 32768 bytes.
  Data phase end.

Table phase end.
```

Note: The INSPECT utility only provides estimates for row compression. For index compression estimates, use the ADMIN_GET_INDEX_COMPRESS_INFO table function as described in 2.1.4.
2.1.2 ADMINTABCOMPRESSINFO view and ADMIN_GET_TAB_COMPRESS_INFO table function

In DB2 V9.5, the new administrative view ADMINTABCOMPRESSINFO and table function ADMIN_GET_TAB_COMPRESS_INFO can be used to get compression information on the tables.

The ADMINTABCOMPRESSINFO administrative view returns a report of compression information for all the tables in a database. You can use a predicate in the query to get the information you are interested in.

Example:

db2lrp> db2 "select * from sysibmadm.admintabcompressinfo where compress_attr='Y'"

<table>
<thead>
<tr>
<th>TABSCHEMA</th>
<th>TABNAME</th>
<th>DBPARTITIONNUM</th>
<th>DATA_PARTITION_ID</th>
<th>COMPRESS_ATTR</th>
<th>DICT_BUILDER</th>
<th>DICT_BUILD_TIMESTAMP</th>
<th>COMPRESS_DICT_SIZE</th>
<th>EXPAND_DICT_SIZE</th>
<th>ROWS_SAMPLED</th>
<th>PAGES_SAVED_PERCENT</th>
<th>BYTES_SAVED_PERCENT</th>
<th>AVG_COMPRESS_REC_LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAPLRP</td>
<td>COSP</td>
<td>0</td>
<td>0</td>
<td>Y</td>
<td></td>
<td>2009-12-17 17.30.14.000000</td>
<td>37760</td>
<td>32768</td>
<td>18799934</td>
<td>87</td>
<td>144</td>
<td></td>
</tr>
</tbody>
</table>

The ADMIN_GET_TAB_COMPRESS_INFO table function allows you to specify a schema, table name, and an execution mode to get the compression information for a specific table.

Syntax

>>> ADMIN_GET_TAB_COMPRESS_INFO --(--tabschema--,--tabname--,--execmode--)-->

The execution mode can be REPORT or ESTIMATE. The REPORT mode displays compression information as of the last generation of the data dictionary and the ESTIMATE mode provides estimates based on the current data.

The ADMINTABCOMPRESSINFO view and the ADMIN_GET_TAB_COMPRESS_INFO table function return the following information on table compression:

- TABSCHEMA
- TABNAME
- DBPARTITIONNUM
- DATA_PARTITION_ID
- COMPRESS_ATTR
- DICT_BUILDER
- DICT_BUILD_TIMESTAMP
- COMPRESS_DICT_SIZE
- EXPAND_DICT_SIZE
- ROW_SAMPLED
- PAGES_SAVED_PERCENT
- BYTES_SAVED_PERCENT
- AVG_COMPRESS_REC_LENGTH

Example:

db2lrp> db2 "select COMPRESS_ATTR, ROWS_SAMPLED, PAGES_SAVED_PERCENT, BYTES_SAVED_PERCENT, AVG_COMPRESS_REC_LENGTH * from table(admin_get_tab_compress_info('SAPLRP','BKPF','ESTIMATE')) as t"

<table>
<thead>
<tr>
<th>COMPRESS_ATTR</th>
<th>ROWS_SAMPLED</th>
<th>PAGES_SAVED_PERCENT</th>
<th>BYTES_SAVED_PERCENT</th>
<th>AVG_COMPRESS_REC_LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>10331351</td>
<td>82</td>
<td>82</td>
<td>86</td>
</tr>
</tbody>
</table>

1 record(s) selected.
Note: The ADMIN_GET_TAB_COMPRESS_INFO table function has been deprecated and replaced by the ADMIN_GET_TAB_COMPRESS_INFO_V97 table function in DB2 V9.7.

2.1.3 ADMINTABCOMPRESSINFO view and ADMIN_GET_TAB_COMPRESS_INFO_V97 table function

In DB2 V9.7, you can use the ADMINTABCOMPRESSINFO administrative view and the ADMIN_GET_TAB_COMPRESS_INFO_V97 table function to get compression estimates for the tables.

The new administrative view and table function adds the additional element OBJECT_TYPE in the result set to support XML compression estimates. The OBJECT_TYPE can be ‘DATA’ or ‘XML’. Here is a list of compression information that can be returned:

- TABNAME
- DBPARTITIONNUM
- DATA_PARTITION_ID
- COMPRESS_ATTR
- DICT_BUILDER
- DICT_BUILD_TIMESTAMP
- COMPRESS_DICT_SIZE
- EXPAND_DICT_SIZE
- ROW_SAMPLED
- PAGES_SAVED_PERCENT
- BYTE_SAVED_PERCENT
- AVG_COMPRESS_REC_LENGTH
- OBJECT_TYPE

Example:

```
db2lrp> db2 "select TABNAME, PAGES_SAVED_PERCENT, AVG_COMPRESS_REC_LENGTH from sysibmadm.admintabcompressinfo where compress_attr='Y'"
TABNAME PAGES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH OBJECT_TYPE
COSP 87 144 DATA
COSP 0 0 XML
2 record(s) selected.
```

The syntax for ADMIN_GET_TAB_COMPRESS_INFO_V97 table function is the same as ADMIN_GET_TAB_COMPRESS_INFO, and this function can be used to retrieve compression estimates for a single table.

Syntax

```
ADMIN_GET_TAB_COMPRESS_INFO_V97(--tabschema--,--tabname--,--execmode--)  
```

The execution mode can be REPORT or ESTIMATE. The REPORT mode will display compression information as of the last generation of the data dictionary. The ESTIMATE mode will estimate space savings based on current data.

Example:

```
db2lrp> db2 "select COMPRESS_ATTR, ROWS_SAMPLED ,PAGES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH OBJECT_TYPE * from table(admin_get_tab_compress_info_v97('SAPLRP','BKPF','ESTIMATE')) as t"  
COMPRESS_ATTR ROWS_SAMPLED PAGES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH OBJECT_TYPE
N 10331351 82 86 DATA
N 0 0 0 XML
2 record(s) selected.
```
2.1.4 ADMIN_GET_INDEX_COMPRESS_INFO table function (available in DB2 V9.7)

This table function returns estimated space savings for uncompressed indexes. If the index is already compressed, it reports the index compression statistics from the catalogs.

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 or null.
- `datapartitionid` is the database partition ID. To specify that the data is requested for all partitions, use the value -2.

The `ADMIN_GET_INDEX_COMPRESS_INFO` table function returns the following information about indexes:

- INDSCHEMA
- INDNAME
- TABSCHEMA
- TABNAME
- DBPARTITIONNUM
- IID
- DATAPARTITIONID
- COMPRESSION_ATTR
- INDEX_COMPRESSED
- PCT_PAGES_SAVED
- NUM_LEAF_PAGES_SAVED

Example:

```
db2lrp> db2 "select indname, compress_attr, index_compressed, pct_pages_saved
  from table(admin_get_index_compress_info('T','SAPLRP','COSP',0,0)) as t"
```

<table>
<thead>
<tr>
<th>INDNAME</th>
<th>COMPRESS_ATTR</th>
<th>INDEX_COMPRESSED</th>
<th>PCT_PAGES_SAVED</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSP~0</td>
<td>Y</td>
<td>Y</td>
<td>0</td>
</tr>
<tr>
<td>COSP~1</td>
<td>N</td>
<td>N</td>
<td>16</td>
</tr>
<tr>
<td>COSP~2</td>
<td>N</td>
<td>N</td>
<td>66</td>
</tr>
</tbody>
</table>

In the above example, index COSP~0 is physically compressed, and COSP~1 and COSP~2 are not compressed. The PCT_PAGES_SAVED value for COSP~1 and COSP~2 represents the estimated percentage of leaf pages saved. If the index is physically compressed (INDEX_COMPRESSED is "Y"), then this value reports the PCTPAGESAVED value from the system catalog view. The value of 0 for index cosp~0 means no statistics have been collected for this index after compression. After we updated the statistics, it showed that the actual space savings for index COSP~0 are 30%.

Example:

```
altena:db2lrp> db2 runstats on table saplrp.cosp with distribution and sampled detailed indexes all
```

```
d2000001 The RUNSTATS command completed successfully.
```

```
altena:db2lrp> db2 "select indname, compress_attr, index_compressed, pct_pages_saved
  from table(admin_get_index_compress_info('T','SAPLRP','COSP',0,0)) as t"
```

<table>
<thead>
<tr>
<th>INDNAME</th>
<th>COMPRESS_ATTR</th>
<th>INDEX_COMPRESSED</th>
<th>PCT_PAGES_SAVED</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSP~0</td>
<td>Y</td>
<td>Y</td>
<td>30</td>
</tr>
<tr>
<td>COSP~1</td>
<td>N</td>
<td>N</td>
<td>16</td>
</tr>
<tr>
<td>COSP~2</td>
<td>N</td>
<td>N</td>
<td>66</td>
</tr>
</tbody>
</table>
2.2 SAP tools to estimate the space savings

As of SAP Basis Release 7.0 Support Package 12 or SAP Basis Release 7.10, row compression support has been integrated into the DBA Cockpit. You can get compression information as well as activate compression for a specific table in the DBA Cockpit under Space → Single Table Analysis.

To run a compression check for a table, execute the following steps:

1. In the DBA Cockpit, click on Space → Single Table Analysis and enter the table name to be analyzed.

2. Click on the Compression Check button at the top of the page and choose your execute option in the dialog box.
3. After the job has finished, click on the *Compression Status* tab to display the compression check result.

![DB2 Compression Feature in an SAP Environment](image)

2.3 Calculating the number of tables to be compressed to achieve desired storage savings

There are thousands of tables in a typical SAP system and usually it is the largest N tables that use the majority of the space. Compressing these large tables can reduce your database size significantly. As the table gets smaller, the benefit of compressing the table gets smaller. The following diagram from a customer ERP system displays the relationship between the number of tables compressed and the overall space savings. Tables are compressed in sequence of their physical size (excluding LOBs).

![Compression vs Number of Tables Diagram](image)

The customer database size is 770G and the largest 20 tables use 360G in total. From the above diagram, we can see the ultimate compression ratio for the database is a little over 45%. After compressing these 20 tables, we get 30% reduction of the total storage space. When the first 40 tables are compressed, the
compression ratio is about 40%, which is very close to the 45% compression ratio we get for compressing the entire database.

As a rule of thumb, tables with relatively small size do not need to be considered for compression as the benefit is negligible.

### 2.4 Tables where compression is less beneficial

In general, the following tables are not optimal candidates for row compression:

1. Tables with a low compression ratio. Always get a compression estimate before you enable compression for the table.
2. Tables with a high frequency of updates. It is hard to maintain a satisfactory compression ratio if the table is constantly going through heavy updates. In addition, the updated row in compressed format may not fit into the current data page even though the uncompressed format has the same length, which leads to more overflow records.
3. Tables with a small size or an average row length that is smaller than the minimum row length of the page.
4. SAP cluster tables. As data is already stored in SAP compressed format in SAP clustered tables, the compression ratio for such tables is typically relatively low.

**Note:** Tables that benefit only little from table compression can still be considered for index compression.
3. Preparing for compression

3.1 Enabling the tables for large RIDs (Optional)

A record Identifier (RID) is used in the database to describe which data page and where in the data page the record is located. It limits the size of the tables and the number of records that can be stored in one page. Prior to DB2 V9.1, DB2 used 4 byte RIDs (3 bytes for page numbers and 1 byte for slots). The maximum number of rows allowed in one page for a 16K pagesize tablespace is 254. When a table gets compressed, the length of each row becomes much smaller. It is possible that the limit for the number of rows per page is reached before the page physically gets full.

With DB2 V9.1, DB2 supports large RIDs (4 bytes for page numbers and 2 bytes for slots) by which the limits are significantly increased. Newly created DMS tablespaces in DB2 V9.1 and higher use large RIDs by default.

For tables with an estimated compressed row length smaller than the minimum row length defined in the following table, the use of large RIDs and large slots will help achieve maximum space savings with compression.

Limit for regular RIDs and large RIDs:

<table>
<thead>
<tr>
<th>Data Page Size</th>
<th>Regular RIDs: Minimum Row Length (Byte) *</th>
<th>Regular RIDs: Maximum Number of Rows/Page</th>
<th>Large RIDs: Minimum Row Length (Byte) *</th>
<th>Large RIDs: Maximum Number of Rows/Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K</td>
<td>14</td>
<td>251</td>
<td>12</td>
<td>287</td>
</tr>
<tr>
<td>8K</td>
<td>30</td>
<td>253</td>
<td>12</td>
<td>580</td>
</tr>
<tr>
<td>16K</td>
<td>62</td>
<td>254</td>
<td>12</td>
<td>1165</td>
</tr>
<tr>
<td>32K</td>
<td>127</td>
<td>253</td>
<td>12</td>
<td>2335</td>
</tr>
</tbody>
</table>

*Minimum row length refers to minimum logical space for a row that is counted in the page.

To find out if the table is enabled for large RIDs and large slots, you can use the administrative table function ADMIN_GET_TAB_INFO.

Example:

```
db2lrp> db2 "select large_rids,large_slots from table(admin_get_tab_info('SAPLRP','COSP'))"
LARGE_RIDS LARGE_SLOTS
--------------------------------------
Y                   Y
```

Alternatively, you can get the table status using the Single Table Analysis screen in the DBA Cockpit:

There are two ways to enable existing tables for large RIDs and large slots:
1. Convert the existing regular tablespace to a large tablespace with the ALTER TABLESPACE command and execute an offline REORG on the table.

2. Move the table into a large tablespace using DB6CONV.

Detailed instructions on how to perform this task are provided in the migration guide. In addition, an ABAP tool is available that can be used to activate large RIDs. You can download this tool from SAP Note 1108956 and use it for any SAP system starting with SAP release 4.6 or higher.

It is recommended to enable all the tables for large RIDs in a large tablespace. Otherwise, when a table with regular RIDs tries to allocate a new page and there is no free page available under the old tablespace limit, an SQL error will be returned.

```
SQL1236N
Table <table-name> cannot allocate a new page because the index with identifier <index-id> does not yet support large RIDs
```

This error could occur even when the size of the table is below the old table size limit, since the free pages that the table with regular RIDs can address are used by the other large RIDs enabled tables.

**Note:** You can set the COMPRESS attribute for the table to YES before implementing one of the above options. In this way, the table can be compressed and enabled for large RIDs at the same time.

## 3.2 REORG Consideration

When you choose to use an offline REORG to compress the existing data, you have the option to use a temporary tablespace or reorganize the table within the same tablespace. These two options have a different impact on the tablespace layout, the high-water mark (HWM), and reorg execution time.

The classic table reorganization uses a shadow copy approach which consists of the following four phases:

1. **SORT** - If an index was specified on the REORG TABLE command or a clustering index was defined on the table, the rows of the table are first sorted according to that index.
2. **BUILD** - A reorganized copy of the entire table is built, either in its tablespace or in a temporary table space that was specified on the REORG TABLE command.
3. **REPLACE** - The original table object is replaced by a copy from the temporary tablespace, or a pointer is created to the newly built object within the tablespace of the table that is being reorganized.
4. **RECREATE ALL INDEXES** - All indexes that were defined on the table are recreated.

**Note:** To avoid the extra time and resources used for sorting, it is recommended not to specify an index on the reorg command when compressing tables unless reclustering is required.

### 3.2.1 Reorg using a temporary tablespace

If a temporary tablespace is used during an offline REORG, the shadow copy is built in the temporary tablespace. This eliminates the space required for the shadow copy on the table's own tablespace, and it is especially useful for large tables. During the replace phase, the shadow copy is moved back to the original table object. The copy process starts from the anchor of the original table and reuses only the extents occupied by the original table. The HWM of the DMS tablespace is not affected with the use of the temporary tablespace.
In the above example, the size of table T2 is reduced from 6 extents to 3 extents. The reorganized table resides in the first 3 extents of the original table, and the higher extents are freed.

### 3.2.2 Reorg within the same DMS tablespace

If the REORG command is executed without the USE clause, the shadow copy is created within the same tablespace that the table resides and a pointer is created to the newly built object afterwards. In general, this option performs faster as the reorganized table does not need to be copied from the temporary tablespace. During the build phase, DB2 will use the free space under the HWM first for the shadow copy. Depending on the amount of free space under the HWM in the tablespace, the HWM after the REORG could be dramatically different.

In the following example, we use the same original tablespace layout as in figure 3.2.1 to reorganize the table T2 within the same tablespace. The size of table T2 is reduced from 6 extents to 3 extents. Since there are not enough free extents under the HWM to store the shadow copy, new extents are allocated and therefore the HWM gets increased.
If there is plenty of free space under the HWM, the shadow copy would fit into the empty space and move the data further down in the tablespace. In this situation, a REORG within the same tablespace not only compresses the table, but is also an effective way to reduce tablespace fragmentation and prepare the system to lower the HWM.

Figure 3.2: Reorg T2 within the same tablespace

You should choose your REORG options based on the size of the table and the amount of free space in the DMS tablespace. For large tables, it is recommended to use temporary tablespaces for offline REORGs to avoid high impact on the HWM.

After the table is compressed with the REORG command, the space freed is shown as free pending space in the tablespace snapshot. You can execute DB2 LIST TABLE SPACE SHOW DETAILS to convert the free pending space to free.

3.3 Managing tablespace fragmentation

In a tablespace where many tables reside, it's very common to have some large tables heavily interleaved with other tables. When such tables get compressed, some extents that have been occupied before become empty in the tablespace. As a result, you may have a tablespace with small groups of used extents separated by small groups of free extents.
While the scattered free extents can be used for the database’s future growth, a severely fragmented tablespace can lead to degraded backup performance. This is because with the current design of the backup process, the prefetcher gets only a block of contiguous data at each request. Each time DB2 moves to a new block of data, it needs to rebuild the tablespace map to identify the next block. When there are many small, non-contiguous blocks, this has to be done many times and can add up to the backup time considerably.

The ultimate solution to this problem is to rebuild the tablespace by unloading and reloading all the tables from the tablespace. This will provide the most contiguous space. Alternatively, you can perform a second set of reorgs inside the same tablespace, starting from the smallest table. In this way, the HWM grows the least, and the space freed by one table can be used by the next table reorganization. However, both options require a relatively long downtime to complete.

To manage the tablespace fragmentation and reduce database downtime, we recommend the following approach for your compression implementation on large tables:

1. Create new tablespaces for data and indexes.
2. Use DB6CONV to move the large tables to the newly created tablespaces and compress the table at the same time (as described in section 4.2.3).
3. Reorganize the rest of the tables in the original tablespace, starting from the smallest table. The REORG should be run within the same tablespace that the table resides. This step can also be replaced by moving all the tables to other tablespaces.

Using DB6CONV eliminates the need to perform an offline REORG. The tables are accessible while being compressed. In addition, the tablespace HWM is not affected and the compressed tables are stored efficiently in the target tablespace. After the large candidate tables have been compressed and moved to the target tablespace, the time required to reorganize the rest of the tables can be significantly reduced.
4. Enabling compression with SAP tools

4.1 Installation tools

Starting with SAP NetWeaver 7.0 SR 3, the SAP installer was enhanced with two options to minimize the database size, and one of them is "Use DB2's Row Compression".

If the option "Use DB2's Row Compression" is selected, all the tables created during installation are compressed. If the database is on DB2 V9.7, indexes will be automatically compressed as well. The only exception is for fact tables, DataStore objects, and PSA tables in a BW system. These tables can later be compressed through the BW interface (see section 4.2.4).

**Note:** Enabling compression during installation is a convenient way to save space without looking at individual tables in detail.

4.2 Enabling compression for existing systems

As of SAP Basis Release 7.10 or SAP Basis Release 7.0 Support Package 12, you can activate row compression for a single table in the DBA Cockpit under Space → Single Table Analysis. Index compression is supported by the DBA Cockpit as of DB2 V9.7.

To enable compression with the DBA Cockpit, take the following steps:

1. Under Space → Single Table Analysis, enter the table schema and name you want to enable compression for.
2. After the table information is displayed, click on the Compression On/Off button at the top of the page.
3. In the dialog box for Compression Options, choose one of the following options to enable compression:

   a: **Enable Compression**
   This option sets the COMPRESS attribute for the table to YES and does not compress the data. If the database is DB2 V9.5 or higher, ADT may be triggered as the table grows.

   b: **Enable Compression and Run REORG**
   This option sets the COMPRESS attribute for the table to YES and performs an offline REORG to compress all the data in the table. The statistics are updated as well after the REORG operation.

4. If Enable Compression and Run REORG is selected, choose the Offline radio button on the next screen.
5. When the job has finished, you can get updated information for the table by clicking on the Refresh button at the top of the page.

Note: SAP uses the default (KEEPDICTIONARY) with the REORG command for the Enable Compression and Run REORG job in the DBA Cockpit. If the table size is very small and there is not sufficient data in it, the table will be enabled for compression, but no dictionary will be built.

4.2.3 DB6CONV

For a production system, you may not have a maintenance window to perform an offline REORG for large tables in order to compress the data. The DB6CONV tool provides the capability to compress or decompress tables through online table move and no offline REORG is required. The latest version of the DB6CONV report including the DB6CONV user guide can be downloaded from SAP Note 1513862.

When DB6CONV is used to move a compression-enabled table, it first creates the target table and loads a sampling of the data from the source table. A compression dictionary is then created based on the sampled data and the full content of the source table is copied to the target table afterwards. With DB2 V9.7, indexes are also compressed.

Note: DB6CONV can only move tables of the AS ABAP schema, and the tables must be known to the ABAP dictionary. This tool should be used when the activity on the table is low.

You can compress the table using the following steps:

1. Enable the compression attribute on the table by using the ALTER TABLE <tabname> COMPRESS YES command or in the DBA Cockpit, choose the Space → Single Table Analysis screen.
2. Run report DB6CONV in SAP transaction SE38. Enter the name of the table to be converted and the target table spaces. Select Update Statistics After Conversion to perform runstats after the move.
Note: It is recommended to create new tablespaces for the target table. In this way, the compressed tables are stored efficiently in the new tablespaces, and it reduces the maintenance effort required on the original tablespace.

4.2.4 Compression support in BW system

Compression is supported in an SAP BW environment. As BW systems are usually large in size and the queries are more complex and I/O intensive, enabling compression on BW tables not only reduces storage cost, but also improves query performance.

To enable compression for PSA tables, DataStore objects, and fact tables, you can set the RSADMIN parameter DB6_ROW_COMPRESSION to YES using the report SAP_RSADMIN_MAINTAIN. In addition, two tests are included in transaction RSRV to check the compression status for InfoProvider and PSA database tables. A repair function is also provided that can be used to trigger the offline reorganization or the execution of INSPECT.
For more information on deep compression support on SAP BW systems, please refer to the following SAP Notes:

- SAP Note 906765 - DB2 9 data row compression for SAP BW 3.x
- SAP Note 926919 – DB2 9 data row compression for SAP NetWeaver BI 2004s.

4.2.3 Compressing AS Java tables

If you have a double stack system, you can use the DBA Cockpit to get compression estimates and perform compression for Java tables. The steps are the same as described in the previous section and you would need to provide the right schema name for Java tables.

The DB6CONV tool cannot be used to compress AS Java tables.

For a single stack Java system, you can set up the DBA Cockpit to remotely connect to the Java system to perform the task. Please note that the user ID provided for the remote connection needs to have the required authority or privileges to compress the specified tables.

4.3 Migration tools

4.3.1 R3load

The SAP R3load tool has been enhanced with several options to deploy DB2 compression features. With the R3load Kernel versions 6.40 and 7.00, the data can be compressed directly when it is loaded into the target system.

R3load 640

To compress tables during an import with R3load, you can use the following commands:

```
R3load -i ... -loadprocedure fast COMPRESS
```

When R3load is called with the -loadprocedure fast COMPRESS option, it performs the following:
1. Insert/load a certain number of rows into the table uncompressed.
2. Execute an offline REORG on the table to generate the compression dictionary.
3. Continue to insert the remaining rows into the table.

By default, 10,000 rows are inserted in step 1 to build the compression dictionary. You can also define a different number with the environment variable DB6LOAD_COMPRESSION_THRESHOLD. Our test showed that a 1% sample is fairly good enough for most of the tables.
Note: With R3load 640, the loadprocedure fast COMPRESS option is effective only if ROW COMPRESSION was already activated before the data was loaded for the target tables. For more information on R3load 640 compression support, please refer to SAP Note 905614 - DB6: R3load -loadprocedure fast COMPRESS.

R3load 700
R3load Version 7.00 and higher offers new options related to DB2 compression:

- **COMPRESS**
  This is compatible with the option in R3load 640 on DB2 V9.1. If the database is DB2 V9.5 or higher, this option is ignored.

- **COMPRESS_ALL**
  R3load creates all tables with the COMPRESS YES attribute. If the database is DB2 V9.1, the COMPRESS option is used. With DB2 V9.5 or higher, the dictionary is created by DB2 using the automatic dictionary creation feature (ADC).

- **FULL_COMPRESS**
  R3load creates all tables with the COMPRESS YES attribute. It loads all of the data into the table and then executes an offline REORG to compress the data.

- **SAMPLED**
  R3load imports each nth row of the export file into the target table first. It then executes a REORG to build a dictionary based on the sampled inserted data. The amount of data to be sampled can be customized with the environment variable DB6LOAD_SAMPLING_FREQUENCY and DB6LOAD_MAX_SAMPLE_SIZE. You need to restart R3load manually to continue loading the rest of the data.

The COMPRESS_ALL option performs best in terms of load time, and the full_compress option provides an optimal compression ratio with the longest runtime. The SAMPLED option is an optimal combination of load time and compression ratio. Its sampling method of retrieving one row out of every n rows also gives you more representative data than using the first n% of the data.

For more information on compression support of R3load 700, please refer to SAP Note 1058437 - DB6: R3load options for compact installation.
5. Post-Compression Considerations

5.1 Monitoring the compression quality

The effectiveness of compression depends on the data and the quality of the dictionary. Once the dictionary is created for the table, whether through ADC or manually, it stays unchanged and is used for all incoming data. This is not a concern if we already have good representatives of the data when the dictionary was built. However, if the data in a table has changed significantly over time, the compression ratio is likely to get worse. We suggest you monitor the effectiveness of compression periodically as data evolves and reset the compression dictionary when needed.

5.1.1 Evaluating compression quality

The compression statistics for table row compression are stored in syscat.tables with the following columns:

- **AVGROWSIZE**: Average row size for compressed and uncompressed rows
- **AVGCOMPRESSEDROWSIZE**: Average length (in bytes) of compressed rows
- **AVGROWCOMPRESSIONRATIO**: Average compression ratio by row; that is, the average uncompressed row length divided by the average compressed row length
- **PCTROWSCOMPRESSED**: Compressed rows as a percentage of the total number of rows in the table
- **PCTPAGESAVED**: Approximate percentage of pages saved in the table as a result of row compression

The statistics are updated as part of runstats processing. You can check the current compression information by querying the syscat.tables statistics view.

Example:

```sql
db2lrp> db2 "select pctpagesaved, pctrowscompressed from syscat.tables where tabname='VBFA'"
PCTPAGESAVED   PCTROWSCOMPRESSED
--------------   ---------------------------
   78             +1.00000E+002
1 record(s) selected.
```

In section 2, we have explained different methods to get a compression estimate on the table. By comparing the PCTPAGESAVED from syscat.tables with the new estimated compression ratio, you can decide whether the existing dictionary needs to be replaced to achieve a higher compression ratio.

5.1.2 Rebuilding the Compression Dictionary

Rebuilding a dictionary requires an offline reorg of the table. This can be performed from the DB2 command line or with the DBA Cockpit.

**Rebuilding the dictionary from the DB2 command line**

To rebuild the compression dictionary, you need to execute the REORG command with the RESETDICTIONARY clause.

```sql
db2lrp> db2 "reorg table saplrp.s961 resetdictionary"
DB20000I The REORG command completed successfully.
```

**Note:** If the reorg is performed from db2 command line, a runstat is needed to refresh the table statistics.

**Rebuilding the dictionary with the DBA Cockpit**

You can also reset the compression dictionary using the DBA Cockpit. Click on the Compression On/Off button on the Single Table Analysis screen, select option “RUN REORG in order to rebuild Dictionary”, and choose offline REORG. The table will be compressed based on a new dictionary, and the statistics are updated as well.
5.2 Reclaiming space

For DMS tablespaces, the space freed after compression is not released to the system automatically. You can choose to leave the space in the tablespace for further growth, or manually reduce the size of the tablespace after compression.

By design, the DMS tablespace can only be reduced to the high-water mark (HWM). After compression, the HWM could be the same or even higher if the REORG is performed in the same tablespace. To reclaim unused storage, you need to take additional actions to lower the HWM first.

- Tablespaces created with DB2 V9.7 are enabled with reclaimable storage. The HWM on a reclaimable storage tablespace can be easily reduced with the ALTER TABLESPACE command. Please refer to the white paper DB2 9.7 New Features – Reducing the High Water Mark on SCN for more information.
- For tablespaces created prior to DB2 V9.7, the HWM cannot safely be lowered.

After the HWM is lowered, you can execute the ALTER TABLESPACE command to reduce the size of the tablespace. This can also be performed in the DBA Cockpit under Space → Tablespaces.

5.3 Backup Performance

As of DB2 V8, the backup image can be compressed with the COMPRESS YES option in the BACKUP command. In addition to data and index compression, backup compression also compresses catalog tables, LOB objects, auxiliary database files, and database metadata. Our test shows that if the size of the database is reduced significantly with compression, the additional space saved through backup compression is very limited. However, the runtime is much longer. In general, it is not recommended to use backup compression on a compressed database unless space saving is of high priority.
In general, backups perform faster when the size of the database is reduced. In some rare cases, compression may affect backup performance negatively because of the tablespace fragmentation (see section 3.3). This problem will be addressed in future DB2 Fix Packs. To reduce tablespace fragmentation, you can follow the same steps to lower the HWM of the tablespace, or use DB6CONV to move the tables to another tablespace.

In addition, reducing the HWM after compression will also help you to get optimal backup performance because a DB2 backup operation processes all pages up to the HWM as part of the backup image.

6. Reference numbers from the SAP system

SAP customers using DB2 compression features have experienced significant reductions in the size of their databases. With row compression only, the overall compression ratio for the database can be over 50%. DB2 index compression provides additional 10-20% savings.
Related Content

SAP Notes ([https://support.sap.com/notes](https://support.sap.com/notes))
- Note 905614 - DB6: R3load -loadprocedure fast COMPRESS
- Note 1513862 - DB6: Table conversion using DB6CONV version 6 or higher
- Note 906765 - DB2 9 data row compression for SAP BW 3.x
- Note 926919 – DB2 9 data row compression for SAP NetWeaver BI 2004s
- Note 1108956 –Large RIDs
- Note 1058437- R3load options for compact installations

Papers and Redbooks
- [Best Practice: Deep Compression](https://support.sap.com/notes)
- Index Compression
- DB2 9.7 New Features – Reducing the High Water Mark
- DB2 Optimization Techniques for SAP Database Migration and Unicode Conversion
- Row Compression in DB2 9.5: Analysis of a DSS Database Environment
- Deep dive into compression
- Introducing DB2 9, Part 1: Data compression in DB2 9
- Enabling DB2 LUW 97’s LOB Inlining in an SAP Environment
- DB2 Information Center
Copyright

© 2015 SAP SE or an SAP SE affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE.

The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE and its affiliated companies ("SAP SE Group") for informational purposes only, without representation or warranty of any kind, and SAP SE Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP SE Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP SE and other SAP SE products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE in Germany and other countries.