Data Compression

- The benefit of using such compression is that disk space is greatly reduced as is memory consumption (since less buffers need to be written into an instance's buffer cache).
- The downside is that there is additional CPU overhead for loading as well as DML operations.
- Oracle
- SQL Server
- Lab: Compression

store more with less

Page 1

A history of database compression

- Data compression algorithms (such as the Huffman algorithm), have been around for nearly a century, but only today are they being put to use within mainstream information systems processing.
- All of the industrial strength database offer some for of data compression (Oracle, SQL Server, DB2).
- There are several places where data can be compressed, either external or internal within the DBMS software.
- **Physical database compression**
  - **Hardware assisted compression:** IMS, the first commercially available database offers Hardware Assisted Data Compression (HDC) which interfaces with the 3380 DASD to compress IMS blocks at the hardware level, completely transparent to the database engine.
  - **Block/page level compression:** Historical database compression uses external mechanisms that are invisible to the database. As block are written from the database, user exits invoke compression routines to store the compressed block on disk. In 1993, the popular DB2 offers built-in tablespace level compression using a COMPRESS DDL keyword.

http://www.dba-oracle.com/oracle11g/sf_Oracle_11g_Data_Compression_Tips_for_the_DBA.html#history Page 2

Logical, External database compression

- **Logical database compression:** Internal Database compression operates within the DBMS software, and write pre-compressed block directly to DASD:
  - **Table/segment-level compression:** A database administrator has always had the ability to remove excess empty space with table blocks by using Oracle Data Pump or Oracle's online reorganization utility (dbms_redefinition). By adjusting storage parameters, the DBA can tightly pack rows onto data blocks. In 2003, Oracle9i release 2 introduced a table-level compression utility using a table DDL COMPRESS keyword.
  - **Row level compression:** In 2006, DB2 extended their page-level compression with row-level compression. Oracle 11g offers a row-level compression routine.
- **External database compression:** Legacy mainframe databases such as IDMS and DB2 allowed the DBA to choose any data compression algorithm they desired. One popular database compression program was offered by Clemson University (still a leader in data compression technologies), and their compression programs were very popular in the early 1980's with their CLEMCOMP and CLEMDCOM programs. In IDMS, a user exit in the DMCL allowed for the data compression routine to be invoked "before put" (writes), and "after get" (reads).

Read / Write Compressed Data

- Internally, all database compression routines try to avoid changing their internal software and rely on user exits to compress the data outbound and decompress the incoming data before it enters the database buffers. In general, data compression techniques follows this sequence:
- ***Read compressed data (decompress):***

  1 - The database determines that a physical read is desired and issues an I/O request.

  2 - Upon receipt of the data block from disk, Oracle un-compresses the data. This happens in RAM, very quickly.

  3 - The uncompressed data block is moved into the Oracle buffer.
- ***Write an compressed Oracle block (compress):***

  1 - The database determines that a physical write is desired and issues an I/O request.

  2 - The database reads the data block from the buffer and calls a compression routine to quickly compresses the data. This happens in RAM, very quickly.

  3 - The compressed data block is written to disk.

❑ Oracle Compression Overview

- Database compression has evolved dramatically since it was first introduced in the early 1980's, and Oracle 11g claims to have one of the best database compression utilities ever made.
- Over the past decade Oracle introduced several types of compression so we must be careful to distinguish between the disparate tools. The 11g data compression is threshold-based and allows Oracle to honor the freelist unlink threshold (PCTFREE) for the compress rows, thereby allowing more rows per data block.
  - Simple index compression in Oracle 8i
  - Table-level compression in Oracle9iR2 (create table mytab COMPRESS)
  - LOB compression (utl_compress) in Oracle 10g
  - Row-level compression in Oracle 11g, even for materialized views (create table mytab COMPRESS FOR ALL OPERATIONS;)
- The historical external compression (blocks are compressed outbound and uncompressed before presenting to the database) are far simpler because all index objects are treated the same way, whereas with the 11g table compression, a data block may contain both compressed and uncompressed row data.

Oracle Table Compression

- **As your database grows in size, consider using table compression.**
- **Compression saves disk space, reduces memory use in the database buffer cache, and can significantly speed query execution during reads.**
- **Compression has a cost in CPU overhead for data loading and DML. However, this cost may be offset by reduced I/O requirements.**
- **Table compression is completely transparent to applications. It is useful in both decision support systems (DSS) and online transaction processing (OLTP) systems.**
- **You can specify compression for a tablespace, a table, or a partition. If specified at the tablespace level, then all tables created in that tablespace are compressed by default.**
- **Compression can occur while data is being inserted, updated, or bulk loaded into a table. Operations that permit compression include:**
  - Single-row or array inserts and updates
  - The following direct-path insert methods:
    - Direct path SQL*Loader
    - CREATE TABLE AS SELECT statements
    - Parallel INSERT statements
    - INSERT statements with an APPEND or APPEND_VALUES hint

- **Oracle Database support two methods of table compression.**

Table Compression Method	Applications	CREATE/ALTER TABLE Syntax	Direct-Path Insert	DML
Basic compression	DSS	COMPRESS [BASIC]	Yes	Yes
OLTP compression	OLTP, DSS	COMPRESS FOR OLTP	Yes	Yes

- You specify table compression with the **COMPRESS** clause of the **CREATE TABLE** statement.
- You can enable compression for an existing table by using these clauses in an **ALTER TABLE** statement.
- In this case, only data that is inserted or updated after compression is enabled is compressed.
- You can disable table compression for an existing compressed table with the **ALTER TABLE...NOCOMPRESS** statement.
- In this case, all data that was already compressed remains compressed, and new data is inserted uncompressed.
- To enable OLTP table compression, you must set the COMPATIBLE initialization parameter to 11.1.0 or higher.

Examples

- **Example: Enables OLTP table compression on the table orders:**
  - CREATE TABLE orders ... COMPRESS FOR OLTP;
- **Data for the orders table is compressed during both direct-path insert and conventional DML.**
- **The next two examples, which are equivalent, enable basic table compression on the sales_history table, which is a fact table in a data warehouse. Frequent queries are run against this table, but no DML is expected.**
  - CREATE TABLE sales_history ... COMPRESS BASIC;
  - CREATE TABLE sales_history ... COMPRESS;
- **Example: Using the APPEND hint to insert rows into the sales_history table using direct-path insert.**
  - INSERT /*+ APPEND */ INTO sales_history SELECT * FROM sales WHERE year=2008;

Compression and Partitioned Tables

- **A table can have both compressed and uncompressed partitions, and different partitions can use different compression methods.**
- **If the compression settings for a table and one of its partitions disagree, the partition setting has precedence for the partition.**
- **To change the compression method for a partition, do one of the following:**
  - To change the compression method for new data only, use
    ALTER TABLE ... MODIFY PARTITION ... COMPRESS ...
  - To change the compression method for both new and existing data, use
    ALTER TABLE ... MOVE PARTITION ... COMPRESS ...

Determining If a Table Is Compressed

- **In the *_TABLES data dictionary views, compressed tables have ENABLED in the COMPRESSION column.**
- **For partitioned tables, this column is null, and the COMPRESSION column of the *_TAB_PARTITIONS views indicates the partitions that are compressed. In addition, the COMPRESS_FOR column indicates the compression method in use for the table or partition.**
- **SELECT table_name, compression, compress_for FROM user_tables;**

TABLE_NAME	COMPRESSION	COMPRESS_FOR
T1	DISABLED	
T2	ENABLED	BASIC
T3	ENABLED	OLTP

- **SELECT table_name, partition_name, compression, compress_for FROM user_tab_partitions;**

TABLE_NAME	PARTITION_NAME	COMPRESSION	COMPRESS_FOR
SALES	Q4_2008	ENABLED	OLTP
SALES	Q1_2009	ENABLED	OLTP
SALES	Q2_2009	ENABLED	OLTP

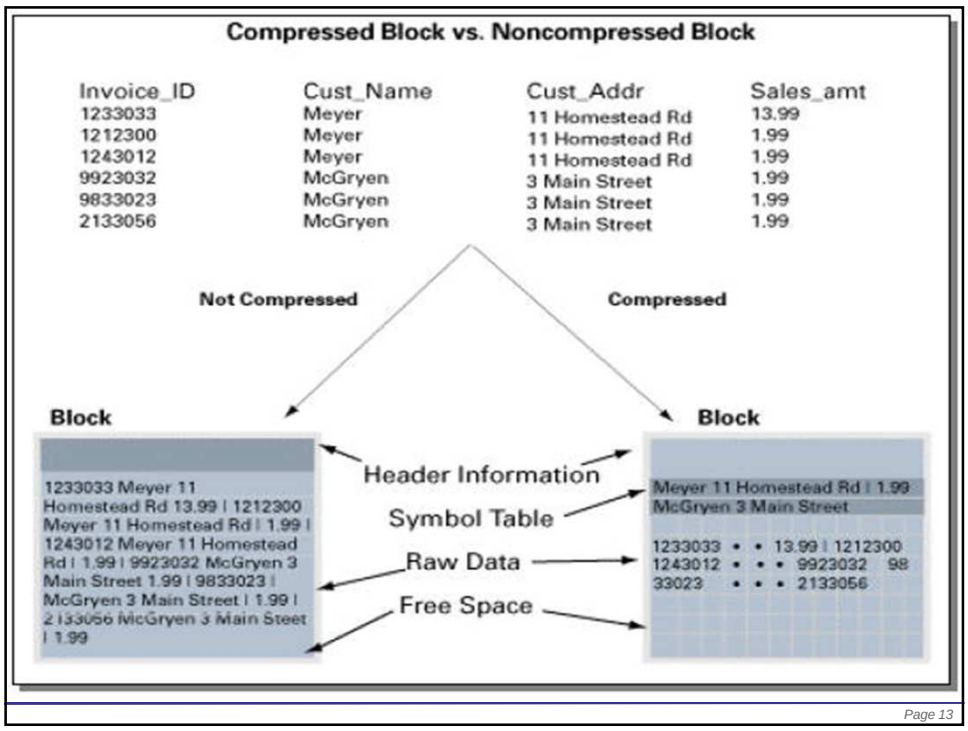
Adding and Dropping Columns in Compressed Tables

- **The following restrictions apply when adding columns to compressed tables:**
  - **Basic compression** - You cannot specify a default value for an added column.
  - **OLTP compression** - If a default value is specified for an added column, the column must be NOT NULL. Added nullable columns with default values are not supported.
- **The following restrictions apply when dropping columns in compressed tables:**
  - **Basic compression** - Dropping a column is not supported.
  - **OLTP compression** - DROP COLUMN is supported, but internally the database sets the column UNUSED to avoid long-running decompression and recompression operations.

Compressed Tablespaces

- **You can specify that all tables created in a tablespace are compressed by default.**
- **You specify the type of table compression using the DEFAULT keyword, followed by one of the compression type clauses used when creating a table.**
- **Example: All tables created in the tablespace are to use OLTP compression, unless otherwise specified:**

  CREATE TABLESPACE ... DEFAULT COMPRESS FOR OLTP ... ;
- **You can override the default tablespace compression specification when you create a table in that tablespace.**



Page 13

Costs and Benefits of Oracle compression

One of the exciting new features of Oracle 11g is the new inline data compression utility that promises these benefits:

- **Up to a 3x disk savings:** Depending on the nature of your data, Oracle compression will result in huge savings on disk space.
- **Cheaper solid-state disk:** Because compressed tables reside on fewer disk blocks, shops that might not otherwise be able to afford solid-state flash disk can now enjoy I/O speed faster than platter disk.
- **Faster full scan/range scan operations:** Because tables will reside on less data blocks, full table scans and index range scans can retrieve the rows with less disk I/O.
- **Reduced network traffic:** Because the data blocks are compressed/decompressed only within Oracle, the external network packets will be significantly smaller.

Page 14

Oracle compression syntax

- The new COMPRESS keyword works for tables, table partitions and entire tablespaces. Oracle has implemented their data compression at the table level, using new keywords within the "*create table*" DDL:
  - create table test (col1 number) NOCOMPRESS;
  - create table test (col1 number) COMPRESS FOR DIRECT_LOAD OPERATIONS;
  - create table test (col1 number) COMPRESS FOR ALL OPERATIONS;

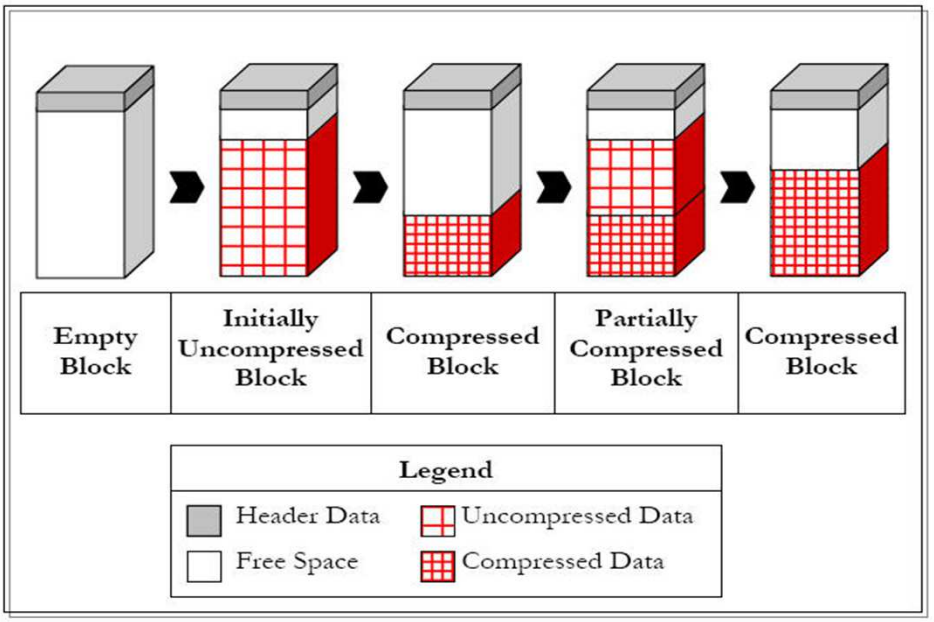
```
CREATE TABLESPACE MYSTUFF . . . DEFAULT{ COMPRESS [ FOR { ALL |
DIRECT_LOAD } OPERATIONS ] | NOCOMPRESS }
```

- Several DBA views have been enhanced in 11g and later version to show compression attributes. The *dba_tables* view has added the new columns COMPRESSED (*enabled, disabled*) and COMPRESSED_FOR (*nocompress*, *compress for direct_load operations, compress for all operations*).
- Oracle has made some important fundamental decisions about mixed-mode compression, a feature whereby it is possible for rows within the same table to be either compressed or uncompressed!

Oracle Compression Options

- The compression clause can be specified at the **tablespace, table** or **partition** level with the following options:
- **NOCOMPRESS:** The table or partition is not compressed. This is the default action when no compression clause is specified.
- **COMPRESS:** This option is considered suitable for **data warehouse** systems. Compression is enabled on the table or partition during **direct-path inserts** only.
- **COMPRESS FOR DIRECT_LOAD OPERATIONS:** This option has the same affect as the simple COMPRESS keyword.
- **COMPRESS FOR ALL OPERATIONS:** This option is considered suitable for **OLTP** systems.
  - This option enables compression for all operations, including regular DML statements.
  - This option requires the COMPATIBLE initialization parameter to be set to 11.1.0 or higher.
  - In 11gR2 this option has been renamed to **COMPRESS FOR OLTP** and the original name has been deprecated.

Oracle's multi-state compression



Page 17

Baseline

- How many rows and blocks are in a table plus how many rows are currently stored per block.

```
SQL> select count(*) from sh.customers;

COUNT(*)
   55500
```

```
select count (distinct dbms_rowid.rowid_block_number(rowid))
"TABLE BLOCK COUNT“ from sh.customers;

TABLE BLOCK COUNT
          1454
```

```
select avg(count(*)) "AVERAGE RECORDS PER BLOCK"
from sh.customers group by dbms_rowid.rowid_block_number(rowid);

AVERAGE RECORDS PER BLOCK
        38.170564
```

Page 18

Compression Results

- Log on as sh
- Create a copy of the table and then observe the table compression results.

```
create table customers_compressed compress for all operations
as select * from customers;

Table created.
```

- Query the data dictionary to confirm the newly created segments have the desired attributes:

```
select table_name, compression, compress_for from user_tables where
table_name in ('CUSTOMERS','CUSTOMERS_COMPRESSED');

TABLE_NAME           COMPRESS   COMPRESS_FOR
CUSTOMERS            DISABLED
CUSTOMERS_COMPRESSED ENABLED    OLTP
```


Compression Results

- **Query the compressed table's storage characteristics:**

```
select count (distinct dbms_rowid.rowid_block_number(rowid))
"TABLE BLOCK COUNT“ from sh.customers_compressed;
TABLE BLOCK COUNT
          763
```

```
select avg(count(*)) "AVERAGE RECORDS PER BLOCK"
from customers_compressed group by
dbms_rowid.rowid_block_number(rowid);
AVERAGE RECORDS PER BLOCK
        72.7391874
```

- **The resulting compressed table stores the same data in about half the space:**

```
select segment_name,  round(bytes/1024/1024) "MB“ from
dba_segments where owner='SH‘ and segment_name in
('CUSTOMERS','CUSTOMERS_COMPRESSED');
SEGMENT_NAME           MB
CUSTOMERS_COMPRESSED    7
CUSTOMERS              12
```

❑ SQL Server Data Compression

- **Only available on SQL Server Enterprise Edition & Developer Edition**
- **There are three types of data compression:**
  - **Row compression**
  - **Page compression**
  - **Backup compression**
- **What is Data Compression?**
  - It is a way of compressing the data within your database so that you can reduce greatly the amount of storage space required to host the data.
  - There is a caveat with this, depending upon the amount of stored data within a table, the allocation unit size of your disk and the datatypes you could in fact end up using MORE storage.

What is Allocation Unit Size (AUS)?

- The blocksize of your disk. This size is set when you format your disk and can range in size from 512 Bytes to 64 KB.
- Larger file allocations will provide performance improvements for applications such as SQL Server, particularly the 64 KB AUS.
- The size of a SQL Server extent (8 pages) is 64 KB, and so optimizes performance.
- The downside of a larger AUS is that is takes a great deal more space to hold a file on the disk, so if you have a lot of smaller files you could end up using a far more disk space than you need to as the disk has to allocate 64 KB of space for even a 2 KB file.

The Row Compression (1)

- **The row compression can compress columns of data to use only the minimum amount of space required.**
- **Row Compression can be enabled when creating a table or index or when altering an index or table.**
- **Row level compression will provide savings by not storing blank characters within fixed character strings, such as a char(10) with a 5 character value.**

```
CREATE TABLE NotCompressed
(id int, FName varchar(100), LName varchar(100))
```

```
declare @n int
set @n=1
while @n<=10000
begin
insert into NotCompressed
values
(1,'Adam','Smith'),(2,'Maria','Carter'),(3,'Walter','Shoemaker')
set @n=@n+1
end
```

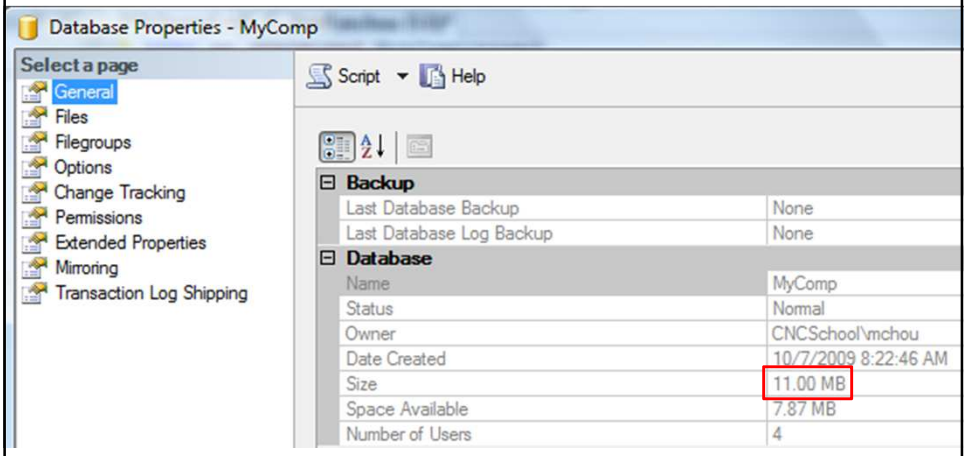
30000

id	FName	LName
1	Adam	Smith
2	Maria	Carter
3	Walter	Shoemaker
1	Adam	Smith
2	Maria	Carter
3	Walter	Shoemaker
1	Adam	Smith
2	Maria	Carter

Page 23

The Row Compression (2)

Name	Type	Size
MyComp_data	MDF File	10,240 KB
MyComp_log	SQL Server Database Transaction Log File	1,024 KB



Page 24

The Row Compression (3) – Without Compression

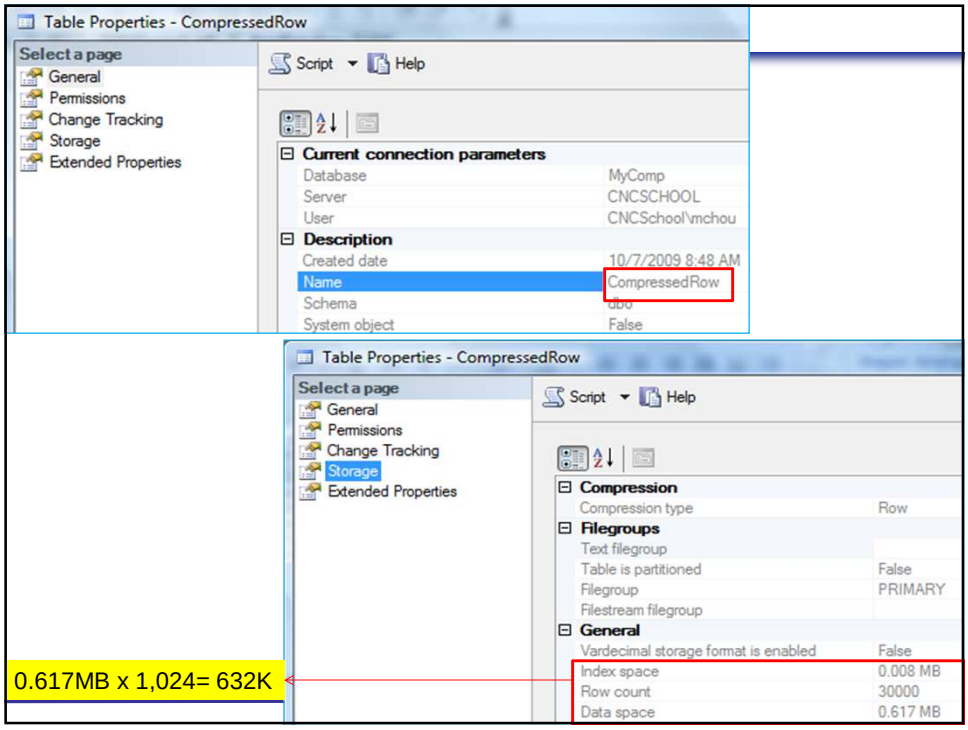
- Now let's query the space used by this table using the following transact SQL statement.

```
EXEC sp_spaceused NotCompressed
```

Name	Rows	Reserved	Data	Index_size	Unused
NotCompressed	30000	968 KB	960 KB	8 KB	0 KB

Column name	Data type	Description
name	**nvarchar(128)**	Name of the object for which space usage information was requested.
rows	**char(11)**	Number of rows existing in the table. If the object specified is a Service Broker queue, this column indicates the number of messages in the queue.
reserved	**varchar(18)**	Total amount of reserved space for *objname*.
data	**varchar(18)**	Total amount of space used by data in *objname*.
index_size	**varchar(18)**	Total amount of space used by indexes in *objname*.
unused	**varchar(18)**	Total amount of space reserved for *objname* but not yet used.

Page 25



The Row Compression (4) – With Compression

```
CREATE TABLE CompressedRow
(id int, FName varchar(100), LName varchar(100))
with (Data_compression = ROW)
```

```
declare @n int
set @n=1
while @n<=10000
begin
insert into CompressedRow
values (1,'Adam','Smith'),(2,'Maria','Carter'),(3,'Walter','Shoemaker')
set @n=@n+1
end
```

```
EXEC sp_spaceused CompressedRow
```

Name	Rows	Reserved	Data	Index_size	Unused
NotCompressed	30000	968 KB	960 KB	8 KB	0 KB
CompressedRow	30000	648 KB	632 KB	8 KB	8 KB

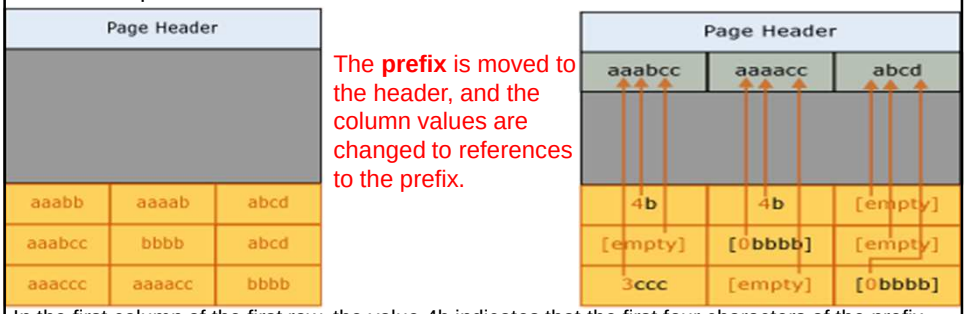
Page Compression

- **Page compression uses a far more complex algorithm to minimize the storage space of data, known as dictionary compression.**
- **SQL Server looks at all of the data stored on a page and builds a dictionary based upon that data which can be referenced for repeated values, and only the dictionary id and changes of the dictionary value are stored.**
- **This provides great savings for similar patterned data. Page compression includes row compression, so you get the benefit of both.**
- **Page compression is similar for tables, table partitions, indexes, and index partitions.**
- **Compressing the leaf level of tables and indexes with page compression consists of three operations in the following order:**
  - Row compression
  - Prefix compression
  - Dictionary compression

Prefix Compression

For each page that is being compressed, prefix compression uses the following steps:

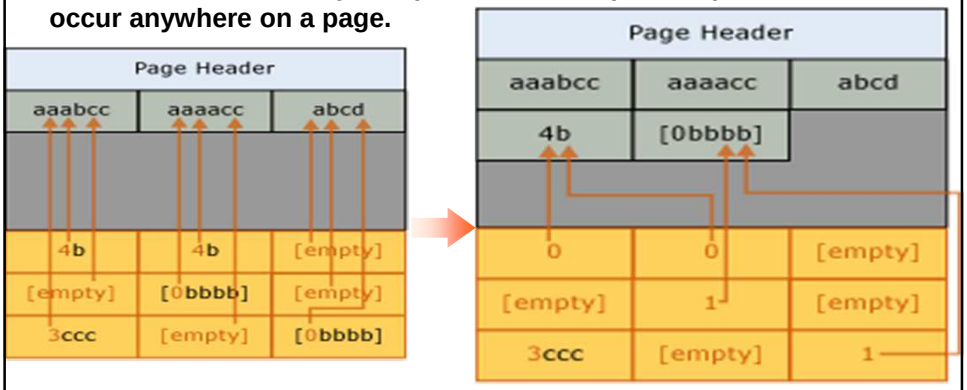
- For each column, a value is identified that can be used to reduce the storage space for the values in each column.
- A row that represents the prefix values for each column is created and stored in the compression information (CI) structure that immediately follows the page header.
- The repeated prefix values in the column are replaced by a reference to the corresponding prefix. If the value in a row does not exactly match the selected prefix value, a partial match can still be indicated.



In the first column of the first row, the value 4b indicates that the first four characters of the prefix (aaab) are present for that row, and also the character b. This makes the resultant value aaabb, which is the original value.

Dictionary Compression

- **After prefix compression has been completed, dictionary compression is applied.**
- **Dictionary compression searches for repeated values anywhere on the page, and stores them in the CI area.**
- **Unlike prefix compression, dictionary compression is not restricted to one column. Dictionary compression can replace repeated values that occur anywhere on a page.**



Note that the value 4b has been referenced from different columns of the page.

The Page Compression (1) – With Compression

```
CREATE TABLE CompressedPage
(id int, FName varchar(100), LName varchar(100))
with (Data_compression = Page)
```

```
declare @n int
set @n=1
while @n<=10000
begin
insert into CompressedPage
values (1,'Adam','Smith'),(2,'Maria','Carter'),(3,'Walter','Shoemaker')
set @n=@n+1
end
```

```
EXEC sp_spaceused CompressedPage
```

Name	Rows	Reserved	Data	Index_size	Unused
NotCompressed	30000	968 KB	960 KB	8 KB	0 KB
CompressedRow	30000	648 KB	632 KB	8 KB	8 KB
CompressedPage	30000	648 KB	632 KB	8 KB	8 KB

Page 31

When Page Compression Occurs

- **When a new table is created that has page compression, no compression occurs.**
  - The metadata for the table indicates that page compression should be used.
- **As data is added to the first data page, data is row-compressed. Because the page is not full, no benefit is gained from page compression.**
- **When the page is full, the next row to be added initiates the page compression operation. The whole page is reviewed; each column is evaluated for prefix compression, and then all columns are evaluated for dictionary compression.**
- **If page compression has created enough room on the page for an additional row, the row is added, and the data is both row- and page-compressed.**
- **If the space gained by page compression minus the space that is required for the CI structure is not significant, page compression is not used for that page.**
  - Future rows either fit onto the new page or, if they do not fit, a new page is added to the table.
  - Similar to the first page, the new page is not at first page-compressed.
- **When an existing table that contains data is converted to page compression, each page is rebuilt and evaluated. Rebuilding all the pages causes the rebuilding of the table, index, or partition.**

Page 32

Add Clustered Index

- **There are also situations where you need to create an Index using data compression.**
  - Clustered Index
  - Nonclustered Index

```
CREATE TABLE CompressedCIX
(id int, FName varchar(100), LName varchar(100))
```

```
declare @n int
set @n=1
while @n<=10000
begin
insert into CompressedCIX
values
(1,'Adam','Smith'),(2,'Maria','Carter'),(3,'Walter','Shoemaker')
set @n=@n+1
end
```

```
create clustered index CIX on CompressedCIX (ID)
```


Alter Table and Clustered Index with Compression

```
EXEC sp_spaceused CompressedCIX
```

Uncompressed

```
ALTER TABLE CompressedCIX REBUILD
  WITH (DATA_COMPRESSION = PAGE);
```

```
ALTER INDEX CIX on CompressedCIX REBUILD
WITH (DATA_COMPRESSION = PAGE);
```

```
EXEC sp_spaceused CompressedCIX
```

Compressed

Name	Rows	Reserved	Data	Index_size	Unused
CompressedCIX – With Clustered Index w/o Compression	30000	1240 KB	1096 KB	48 KB	96 KB
CompressedCIX – With Clustered Index with Compression	30000	456 KB	328 KB	16 KB	112 KB

Add NonClustered Index

```
CREATE TABLE CompressedIX
(id int, FName varchar(100), LName varchar(100))
```

```
declare @n int
set @n=1
while @n<=10000
begin
insert into CompressedIX
values
(1,'Adam','Smith'),(2,'Maria','Carter'),(3,'Walter','Shoemaker')
set @n=@n+1
end
```

```
create nonclustered index IX on CompressedIX (ID)
```


Alter Table and NonClustered Index with Compression

```
EXEC sp_spaceused CompressedIX
```

Uncompressed

```
ALTER TABLE CompressedIX REBUILD
  WITH (DATA_COMPRESSION = PAGE);
```

```
ALTER INDEX IX on CompressedIX REBUILD
WITH (DATA_COMPRESSION = PAGE);
```

```
EXEC sp_spaceused CompressedIX
```

Compressed

Name	Rows	Reserved	Data	Index_size	Unused
CompressedIX – With NonClustered Index w/o Compression	30000	1616 KB	960 KB	560 KB	96 KB
CompressedIX – With NonClustered Index with Compression	30000	920 KB	344 KB	424 KB	152 KB

Summary

Reserved = Data + Index_size + Unused

Name	Rows	Reserved	Data	Index_size	Unused
NotCompressed	30,000	968 KB	960 KB	8 KB	0 KB
CompressedRow	**30,000**	**648 KB**	**632 KB**	**8 KB**	**8 KB**
CompressedPage	**30,000**	**648 KB**	**632 KB**	**8 KB**	**8 KB**
CompressedCIX – with Clustered Index w/o Compression	30,000	1240 KB	1096 KB	48 KB	96 KB
CompressedCIX – with Clustered Index w/o Compression	**30,000**	**456 KB**	**328 KB**	**16 KB**	**112 KB**
CompressedIX – with NonClustered Index w/o Compression	30,000	1616 KB	960 KB	560 KB	96 KB
CompressedIX – with NonClustered Index with Compression	**30,000**	**920 KB**	**344 KB**	**424 KB**	**152 KB**

The results show that the size of the **reserved and data columns** **are much less in Compressed table** when compared to None Compressed table.

Page 37

Potential Issues with Data Compression

- **Data compression is not for everybody.**
- **Depending upon the workload of your system, the performance requirements, and whether or not you use encryption this might not be the right thing for you.**
- **There is a CPU overhead associated with using data compression, and this may adversely impact your system.**
- **High volume OLTP systems could be significantly impacted by attempting to implement data compression.**

Page 38

How Do I know if Data Compression is Right for Me?

- **Estimate the potential storage savings that you could get by implementing *(you could actually end up using MORE storage within certain circumstances)***
- **Complete a baseline performance analysis of your database server and reproduce this in a development or staging environment.**
- **Enable compression and evaluate the performance against that baseline.**
- **Look at which tables can provide you the biggest benefit.**
- **Tables which have a lot of repetitive or numerical data or CHAR columns that are not fully populated are usually excellent candidates for compression.**
- **Check and recheck your analysis against the baseline, and seek feedback from the users if you do implement it in a production environment.**

How Do I know if a Table is Worth Compressing?

sp_estimate_data_compression_savings is available only in the SQL Server Enterprise and Developer editions.

- **@schema_name** allows you to estimate the savings against tables on a different schema (by default the procedure only looks at tables within your own schema).
- **@object_name** refers to the table that you wish to evaluate
- **@index_id** to specify the estimated savings for a single index on a table based upon the index id, left with the default value of NULL it will assess all of the indexes on that table.
- **@partition_number** define particular partitions on a table to evaluate potential savings.
- **@data_compression** can have one of three values (NONE, ROW, PAGE). Depending upon the value passed this will perform estimations for the two compression types and strangely for no compression.
- This can be very useful for estimating the savings on older data, rarely changed living on a partition, which you may want to compress as opposed to more volatile data within a different partition which may not be a good candidate for compression.

```
EXEC sp_estimate_data_compression_savings 'Production', 'WorkOrderRouting', NULL, NULL, 'ROW' ;
```

object_name	schema_name	index_id	partition_number	size_with_current_compression_setting(KB)	size_with_requested_compression_setting(KB)
WorkOrderRouting	Production	1	1	5592	4040
WorkOrderRouting	Production	2	1	920	792

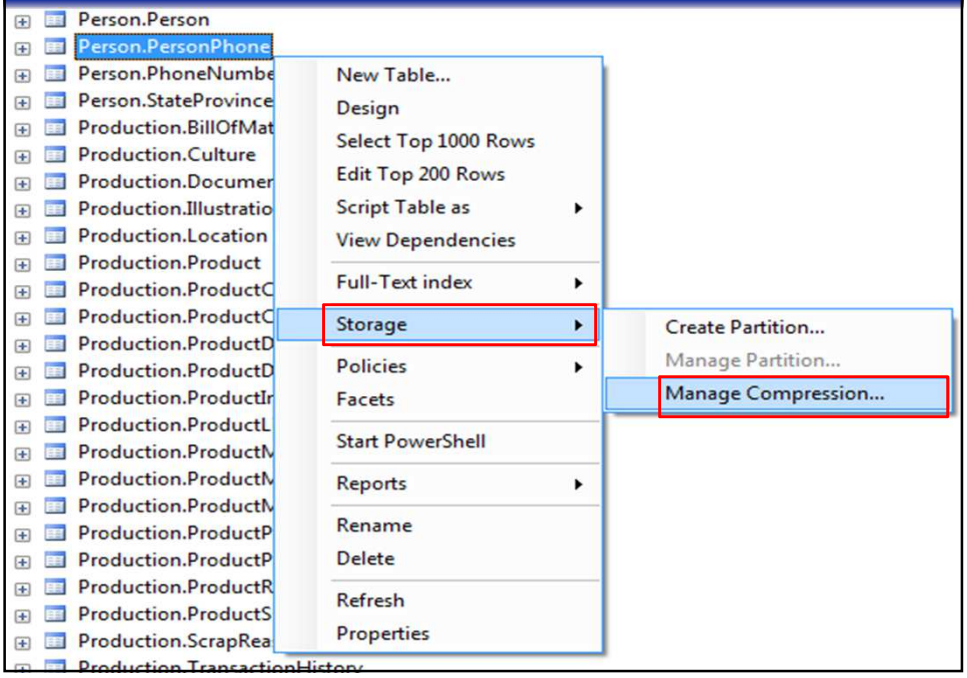
sample_size_with_current_compression_setting(KB)	sample_size_with_requested_compression_setting(KB)
5592	4040
920	792

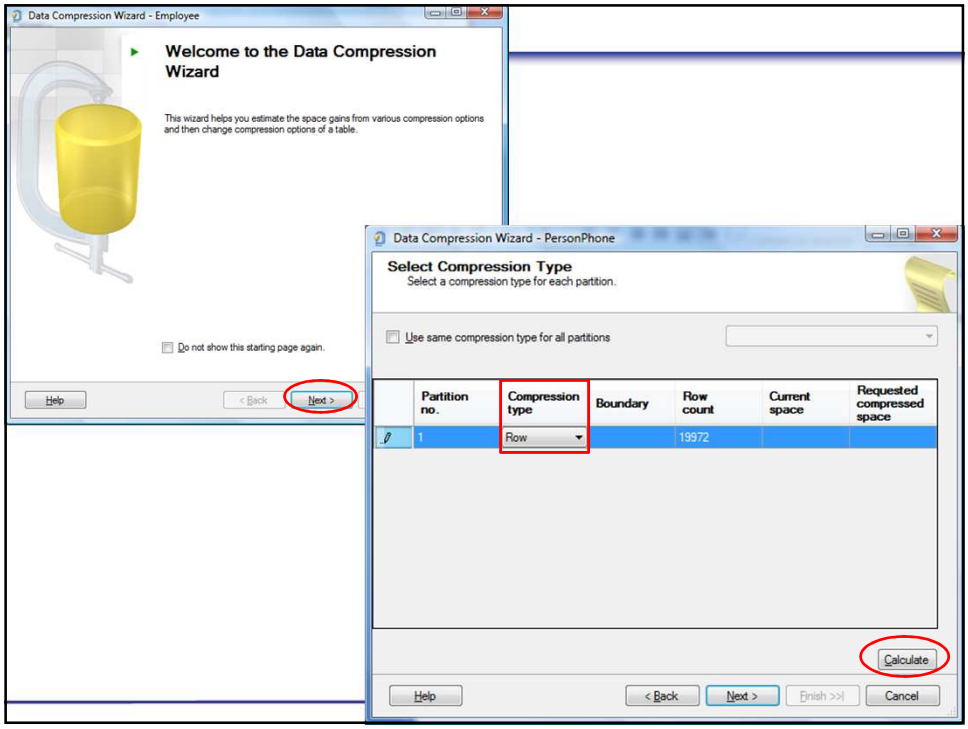
Results from sp_estimate_data_compression_savings

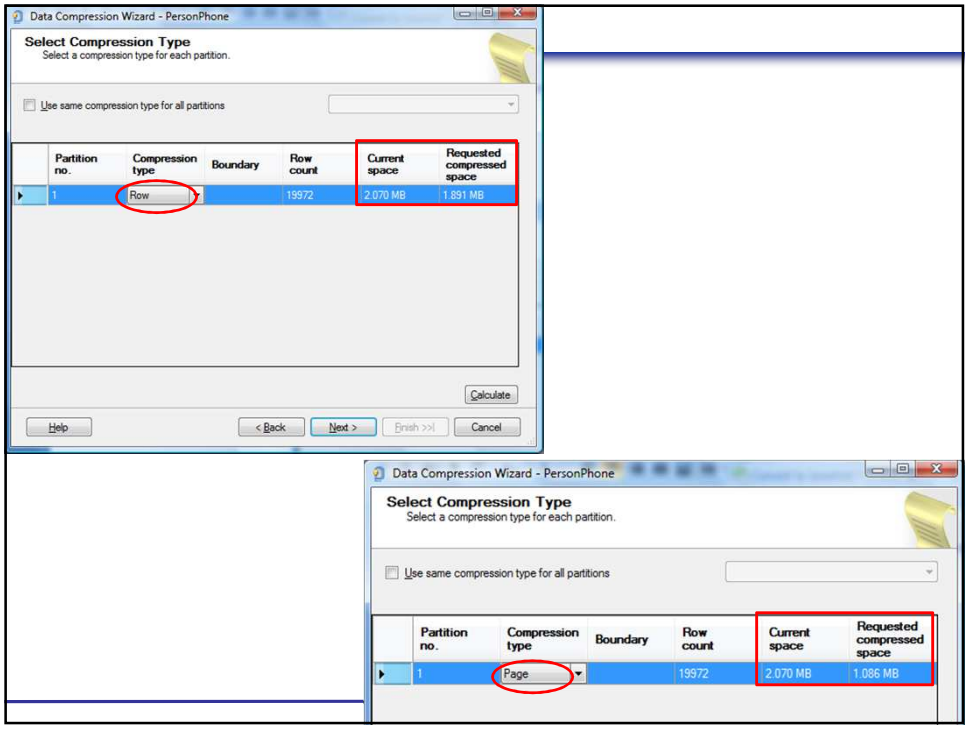
Column name	Data type	Description
object_name	**sysname**	Name of the table or the indexed view.
schema_name	**sysname**	Schema of the table or indexed view.
index_id	**int**	Index ID of an index:
		0 = Heap
		1 = Clustered index
		> 1 = Nonclustered index
partition_number	**int**	Partition number. Returns 1 for a nonpartitioned table or index.
size_with_current_compression_setting (KB)	**bigint**	Size of the requested table, index, or partition as it currently exists.
size_with_requested_compression_setting (KB)	**bigint**	Estimated size of the table, index, or partition that uses the requested compression setting; and, if applicable, the existing fill factor, and assuming there is no fragmentation.
sample_size_with_current_compression_setting (KB)	**bigint**	Size of the sample that is created by using the existing compression setting; and, if applicable, the existing fill factor and no fragmentation. Because this rowset is created from scratch, there is no fragmentation.
sample_size_with_requested_compression_setting (KB)	**bigint**	Size of the sample that is created by using the requested compression setting; and, if applicable, the existing fill factor and no fragmentation.

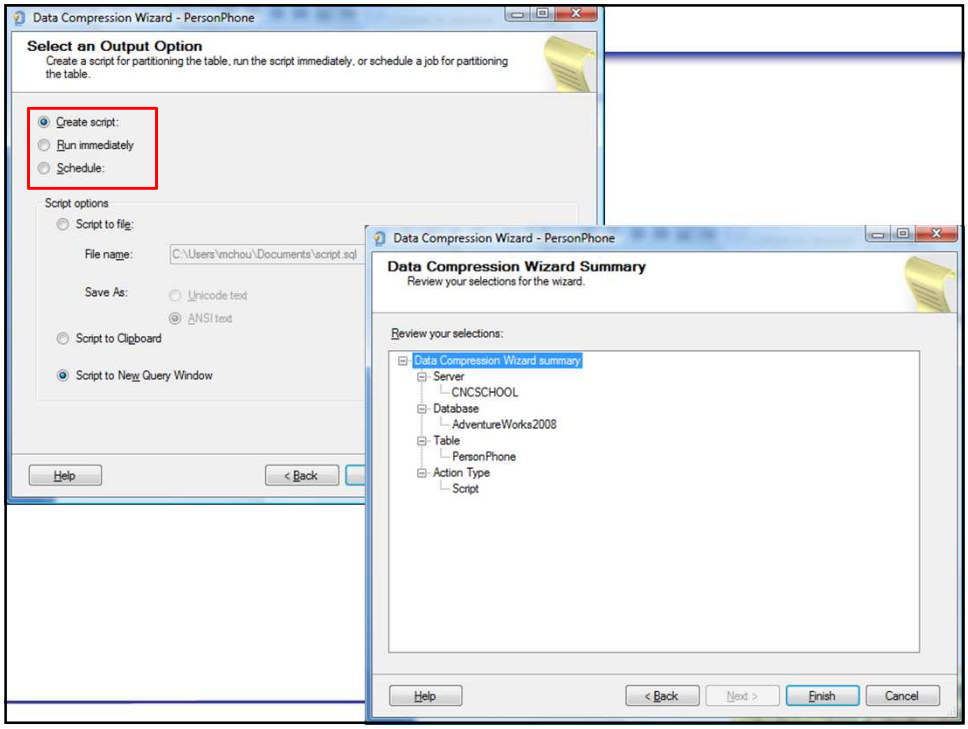
Page 41

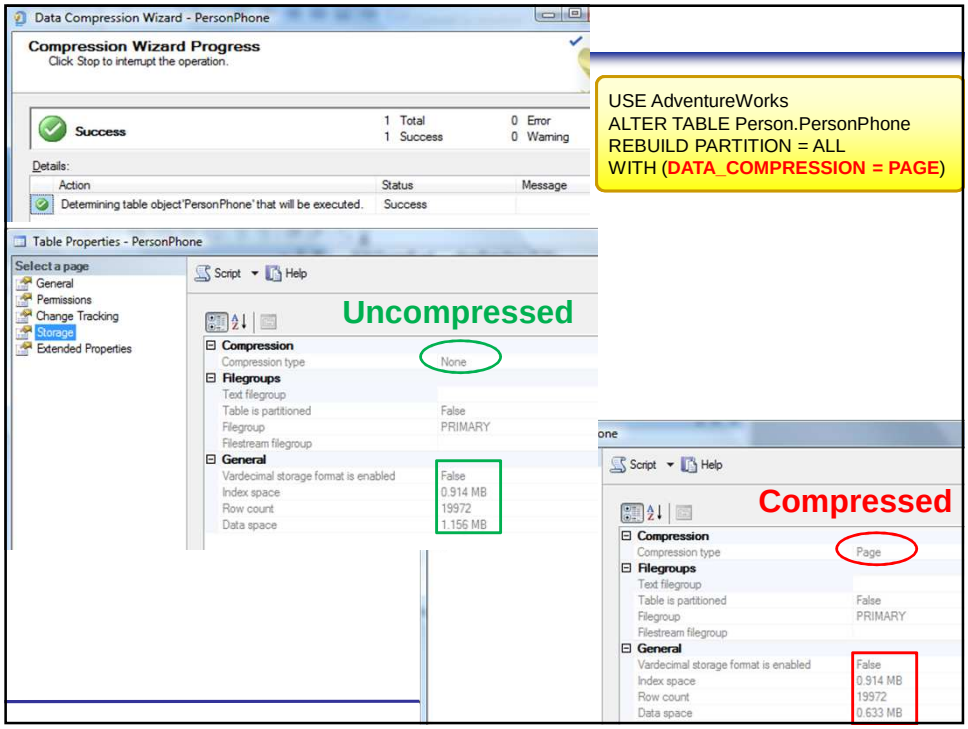
Management Studio → Storage → Manage Compression




Data Compression Wizard - Employee
Welcome to the Data Compression Wizard
This wizard helps you estimate the space gains from various compression options and then change compression options of a table.
Do not show this starting page again.
Help
< Back
Next >
Data Compression Wizard - PersonPhone
Select Compression Type
Select a compression type for each partition.
Use same compression type for all partitions
Partition no.
Compression type
Boundary
Row count
Current space
Requested compressed space
1
Row
19972
Calculate
Help
< Back
Next >
Finish >>|
Cancel


Data Compression Wizard - PersonPhone
Select Compression Type
Select a compression type for each partition.
Use same compression type for all partitions
Partition no.
Compression type
Boundary
Row count
Current space
Requested compressed space
1
Row
19972
2.070 MB
1.891 MB
Calculate
Help
< Back
Next >
Finish >>|
Cancel
Data Compression Wizard - PersonPhone
Select Compression Type
Select a compression type for each partition.
Use same compression type for all partitions
Partition no.
Compression type
Boundary
Row count
Current space
Requested compressed space
1
Page
19972
2.070 MB
1.086 MB


Data Compression Wizard - PersonPhone
Select an Output Option
Create a script for partitioning the table, run the script immediately, or schedule a job for partitioning the table.
Create script:
Run immediately
Schedule:
Script options
Script to file:
File name:
C:\Users\vnchou\Documents\script.sql
Save As:
Unicode text
ANSI text
Script to Clipboard
Script to New Query Window
Help
< Back
Data Compression Wizard - PersonPhone
Data Compression Wizard Summary
Review your selections for the wizard.
Review your selections:
Data Compression Wizard summary
Server
CNCSCHOOL
Database
AdventureWorks2008
Table
PersonPhone
Action Type
Script
Help
< Back
Next >
Finish
Cancel


Data Compression Wizard - PersonPhone
Compression Wizard Progress
Click Stop to interrupt the operation.
Success
1 Total
1 Success
0 Error
0 Warning
Details:
Action
Status
Message
Determining table object 'PersonPhone' that will be executed.
Success
USE AdventureWorks
ALTER TABLE Person.PersonPhone
REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = PAGE)
Table Properties - PersonPhone
Select a page
General
Permissions
Change Tracking
Storage
Extended Properties
Script
Help
Uncompressed
Compression
Compression type
None
Filegroups
Text filegroup
Table is partitioned
False
Filegroup
PRIMARY
Filestream filegroup
General
Vardecimal storage format is enabled
False
Index space
0.914 MB
Row count
19972
Data space
1.156 MB
Script
Help
Compressed
Compression
Compression type
Page
Filegroups
Text filegroup
Table is partitioned
False
Filegroup
PRIMARY
Filestream filegroup
General
Vardecimal storage format is enabled
False
Index space
0.914 MB
Row count
19972
Data space
0.633 MB

Comparison

```
EXEC sp_spaceused 'person.personphone'
```

Uncompressed

name	rows	reserved	data	index_size	unused
PersonPhone	19972	2320 KB	1184 KB	936 KB	200 KB

Compressed

name	rows	reserved	data	index_size	unused
PersonPhone	19972	1808 KB	648 KB	936 KB	224 KB

Uncompress Data

```
USE [AdventureWorks]
ALTER TABLE [Person].[PersonPhone]
REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = NONE)
```