Data Compression Overview

Data compression lets you use less storage space for the same amount of data, reduce cache memory consumption, and improve performance because of lower I/O demands.

You can compress large object (LOB) and regular data.

**Note:** Regular data and LOB data use separate compression syntax and options. In this documentation, the phrase "data compression" indicates compression for data other than LOB columns, while "LOB compression" indicates compression for LOB columns.

Adaptive Server® provides different levels of compression for regular and LOB data. Generally, higher compression ratios use more CPU when you decompress the data. Select compression levels based on how the data is accessed. Data that you access frequently ("hot data") may be best suited for compression levels that have smaller CPU overhead.

After you create a compressed table or partition, Adaptive Server compresses any subsequently inserted or updated data (that is, existing data is not already compressed). If Adaptive Server cannot efficiently compress the inserted data, the original row is retained. If newly inserted or updated LOB data occupies space that is smaller than or equal to a single data page, Adaptive Server does not compress this data.

Tables can be a mixture of compressed and uncompressed data. For example, if you create a compressed table, load data, then disable data compression for the table, previously inserted data is compressed, but rows added after you disable compression are not compressed.

You need not uncompress data to run queries against it. You can insert, update, and delete compressed data; running `select` or `readtext` statements on the compressed column returns decompressed rows. Because there is less data for Adaptive Server to search, there are fewer I/Os, improving the efficiency of data storage.

Enabling Data Compression

To compress data, you must obtain a current ASE_COMPRESSION license, then set the system-wide configuration parameter `enable compression`.

1. Obtain an ASE_COMPRESSION license from the SPDC download site (see the *SySAM Users Guide* or your Sybase® representative).
2. Enable data compression using:
   
   ```sql
   sp_configure 'enable compression', 1
   ```

   See *System Administration Guide: Volume 1 > Setting Configuration Parameters.*
Select Data into Compressed Tables

Use `select into ... compression` to select regular and LOB data directly into a compressed table.

The destination table does not inherit anything from the original table. That is, if the table from which you are selecting data is page-level compressed, the table into which you select the data can be row-level compressed, or not compressed.

You must indicate compression levels if you are selecting large object data into a table.

The behavior of `select into` on target tables or columns depends on the type of compression you are using.

<table>
<thead>
<tr>
<th>Compression Type</th>
<th>Behavior of select into on Target Tables</th>
<th>Source Table or Column</th>
<th>Database-wide Setting for compression</th>
<th>Target Table or Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Compression</td>
<td>Target table or columns do not inherit any properties from the source table. If you do not specify <code>compression</code>, tables other than temporary tables inherit the database-wide setting for compression. Temporary tables do not inherit any compression settings from the source table, source column, or from the target databases’s attributes.</td>
<td>Table can be compressed or uncompressed, and may include one or more compressed columns.</td>
<td><code>none</code></td>
<td>Target table and all columns are uncompressed.</td>
</tr>
<tr>
<td>LOB compression</td>
<td>LOB columns in the target table do not inherit any properties from the source columns. If you do not specify <code>compression</code>, LOB columns in target tables other than temporary tables inherit the database-wide setting.</td>
<td>Source LOB columns may be compressed.</td>
<td><code>lob_compression = 0</code>, unset for the database</td>
<td>All LOB columns in the target table are uncompressed.</td>
</tr>
</tbody>
</table>
This example selects all rows from the `titles` table, and creates a new table named `titles_2` with row-level compression:

```sql
select * into titles2
with compression = row
from titles
```

See the *Reference Manual: Commands*.

## Administering Compressed Databases

Administration duties for compressed databases include enabling or disabling session compression, bulk-copying, and dumping and loading compressed data.

Use the

- `compression info pool size` configuration parameter to check the memory pool for compression.
- `capture compression statistics` to enable the `monTableCompression` monitoring table to begin capturing compression statistics.

See the *System Administration Guide: Volume 1*.

### Session-Level Data Compression

Enable and disable compression for a session with the `set` command.

To enable compression for the current session, use:

```sql
set compression {on | off | default}
```

This command has no effect on uncompressed tables. When you enable compression for a session, Adaptive Server compresses all subsequent data inserted in the table that uses the

<table>
<thead>
<tr>
<th>Compression Type</th>
<th>Behavior of select into on Target Tables</th>
<th>Source Table or Column</th>
<th>Database-wide Setting for compression</th>
<th>Target Table or Column</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For the <code>lob_compression</code> attribute. LOB columns in temporary tables inherit nothing from the source table, source column, or from the target database's attributes.</td>
<td>lob_compression = compression_level</td>
<td>lob_compression = compression_level</td>
<td>All LOB columns in the target table are created using the database-wide setting for <code>lob_compression = compression_level</code>.</td>
</tr>
</tbody>
</table>
appropriate datatype. If you set compression off, Adaptive Server disables compression for the duration of the session. When you set compression to `default`, Adaptive Server uses the compression configuration you established when you created the table.

Adaptive Server does not support session-level compression for LOB compression.

Stored or system procedures inherit a session’s compression settings. Subprocedures inherit the `set compression` command settings executed in the parent procedure. When the procedure ends, Adaptive Server restores the compression level of the outer session or parent procedure.

`set compression` changes included with login triggers apply to the session established when you first log in until you explicitly change the compression level. You need not enable `set export_options` in the login trigger to export `set compression` changes. Once the compression level is exported to a session, it applies to individual tables. However, `set compression` is not exported to the immediate parent procedure’s context if you issue `set export_options` in a nested procedure before setting issuing `set compression`.


Copy, Dump, and Load Compressed Data

Use `bcp` to bulk-copy compressed data in and out of tables.

Pages in a compressed table may have a combination of row-compressed, page-compressed, or uncompressed rows. Even tables or partitions marked as uncompressed can include data that is a mixture of different states of compression.

- **bcp out** – any compressed rows (including those with text data) are decompressed and returned to the client, either in native or character form.
- **bcp in** – uncompressed data received from the client is compressed during the insert. `bcp in` selects the appropriate compression scheme, which depends on the compression level of the partition into which you are inserting the row.

When you bulk-copy data out (using `bcp out`), followed by a `bcp in` to a compressed table (or partition), all newly loaded data is compressed, even when the extracted data was stored as uncompressed.

See Utility Guide > Utility Commands Reference > `bcp`

`dump database` dumps compressed data directly from disk to archive. If the transaction log contains compressed LOB data, recover the compressed LOB data with `load tran` (see the System Administration Guide: Volume 2 > Developing a Backup and Recovery Plan).
Limits for Database Compression

Database compression includes limitations on replicating compressed data and in-memory databases.

- Generally, compression is restricted for in-memory databases. Loading and recovering compressed objects in disk-resident or relaxed-durability in-memory databases is permitted. However, Adaptive Server often restricts access to compressed objects in the target in-memory database. Adaptive Server provides minimal support for disabling compression in the target database or in tables defined for compression, so you may revert to using uncompressed data.

- Compressed LOB columns do not support replication. Issue the following to indicate that a column is not to be replicated before you compress columns with LOB data that are part of a replicated database:

  ```sql
  sp_setrepcol table_name, lob_column_name, 'do_not_replicate'
  ```

See the Replication Server Reference Manual.
Data Compression Overview
Levels of Data Compression

You can compress data at the row level and the page level.

Row-Level Compression

Row-level compression compresses individual rows in a table.

Row-level compression is intended for fixed-length, regular data. For most fixed-length columns, data does not completely occupy the space reserved for the row. For example, a 32-bit integer with a value of 2 is represented by 0x10 in hexadecimal. Adaptive Server requires 1 byte to represent this value, but fills the other 3 bits of the row with zeros. Similarly, if a 50-byte fixed-length character column includes the character data “a”, Adaptive Server requires 1 byte for the character data, but completes the column with zeros.

Some fixed-length datatypes are not compressed because there is no benefit in doing so. For example, Adaptive Server uses only 1 byte to store a tinyint, so compressing a row using this datatype is not beneficial.

For example, if you create this uncompressed table:

```
create table t1 (col1 char(1) not null,
    col2 char(50) not null,
    col3 tinyint not null,
    col4 int not null,
    col5 varchar(20))
lock datapages
```

After changing the compression level to row:

```
alter table t1
set compression = row
```

Adaptive Server does not compress `col1` and `col3` because their length is 1 byte. Adaptive Server compresses `col2` and `col4` and stores required information about decompression for each column using the minimum space, if required.

If you insert these values into `t1`:

```
insert t1 values (
    "a", "aaaaa", 1, 100, "NineBytes")
```

The compressed version of the columns comprises 17 bytes, nearly one-third the size of the uncompressed columns:

- When uncompressed, the value of `col2`, char(50) is “aaaaa” with 45 blanks to fill out the rest of the column. After compression, the value of `col2` is “aaaaa”, using one byte for each “a”. 

• The value of col4 is 100, and is represented with a single byte.
• Trailing blanks are truncated from the value of col5; 9 bytes to store the value.

Page-Level Compression

Use page-level compression to compress the amount of data redundancy on a page.

When you specify page-level compression for regular data, Adaptive Server performs row-level compression first, then page-level compression.

Data pages often include repeated information (for example, the same date, time, or department ID). Instead of storing the same value multiple times, page-level compression lets you store these values in a single place and use a symbol on the data page to refer to them.

Adaptive Server includes a number of techniques for page-level compression:
• Extracting repetitive information from variable-length byte strings and replacing them with shorter symbols.
  When you insert a new row into a data page, Adaptive Server compares the data in the columns with the symbols in the page dictionary. If it finds a match in the dictionary for the new data, Adaptive Server stores the dictionary symbol instead of the data, and the row is compressed. When Adaptive Server retrieves the data, the symbol indicates the appropriate data. A page dictionary can include multiple entries, each with a different symbol that compresses a different piece of information.
• Extracting and removing short, duplicate values that use fixed-length columns from the rows.
  If a fixed-length column includes a high number of duplicates, Adaptive Server stores the duplicate value in the page index, and uses a status bit in the row to indicate that this value is stored in the page index and is available for compression. When you retrieve data from the row, the status bit indicates the value that Adaptive Server includes in the result set.
  A page index may contain multiple entries for different duplicate values in the page.
  For example, if you create this table:

```
create table order_line (  
onder_id int,  
disp_id tinyint,  
width_id smallint,  
number tinyint,  
info_id int,  
supply smallint,  
delivery datetime,  
quantity smallint,  
amount float,  
dist_info char(24))  
lock datapages
```

And insert this data:

```
682, 1, 7, 11, 30000, 7, 'Dec 2 2008 1:19PM', 5, 290, 'Houston')
748, 1, 7, 12, 93193, 7, 'Sep 27 2009 1:15PM', 5, 9900,
```
The `disp_id`, `width_id`, `supply`, and `quantity` columns all contain duplicate values (1, 7, 7, and 5), that are all short fixed-length columns, and candidates for page index compression.

- For `char` and `varchar` columns, frequently used characters are encoded with a representation that takes less storage.

If the row length after compression exceeds the original row length, Adaptive Server uses the original row instead of the compressed row.

Adaptive Server analyzes the data and automatically selects the appropriate method of page-level compression.

Compression does not automatically occur on a table configured for page-level compression until you insert a row that causes the page to become full.
Levels of Data Compression
Creating Databases for Data Compression

Compressed databases can include compressed and uncompressed tables or partitions.

Note: The default setting for compression in the model database is none, so unless you specify otherwise, compression is off when you create a database.

To create databases with data compression, use:

```sql
create database database_name
[...]
with dml_logging = { minimal | full }
, durability =
{ no_recovery | at_shutdown | full }
, compression = {none | row | page}
```

The compression parameter indicates that all tables in the database inherit the specified level of compression, unless you explicitly state otherwise. See the Reference Manual: Commands.

This example creates the emaildb database with row-level compression on the emaildb_dev device:

```sql
create database emaildb
on emaildb_dev = '50M'
with compression = row
```

Altering the Compression Level of a Database

Changing a database's compression level does not change the compression level of existing tables in the database; only tables you create after alter the database inherit the new compression level.

Alter the compression level of existing databases using:

```sql
alter database database_name
[...]
set
[[,] compression = {none | row | page}]
```

See the Reference Manual: Commands.

To alter the pubs2 database to use page-level compression, use:

```sql
alter database pubs2
set compression = page
```
Creating a Compressed Table

You can compress all tables except system and worktables.

Use `create table` to create a compressed table or partition. You need not compress all columns in a table. When designing your table, select the columns that offer the greatest benefit from compression. Partitions, and tables can use row- and page-level compression. Partitions for which you do not specify the compression level inherit the table-level compression.

The partial syntax for compression is:

```sql
create table [database.[owner].]table_name
(column_name datatype ...
    [not compressed ],
    [, next_column...])
[with {max_rows_per_page = num_rows,
    ...
    compression = {none | page | row }]
[on segment_name]
[partition clause]
```

```sql
partition_clause::=
partition by partition_type [(column_name[, column_name]...)]
    ([partition_name] ...)
    [with compression = {none | page | row }]
    [on segment_name],
    [, next_partition...])
```

The `create table... with compression` parameter overrides the database-wide setting. That is, if you create a database with row-level compression, then issue a `create table` command that indicates page-level compression, Adaptive Server creates the table using page-level compression.

To compress all columns in the `sales` table, use:

```sql
create table sales
    (store_id int not null,
     order_num int not null,
     date datetime not null)
    with compression = row
```

To compress only the `order_num` column, specify the other columns as `not compressed`:

```sql
create table sales
    (store_id int not null not compressed,
     order_num int not null,
     date datetime not null not compressed)
    with compression = row
```

To use page-level compression on the Y2008 partition and row-level compression on the Y2009 partition, enter:

```sql
create table sales_date
    (store_id int not null,
     order_num int not null,
     date datetime not null)
    with compression = row
```
Creating a Compressed Table

```
order_num int not null,
    date datetime not null)
partition by range (date)
(Y2008 values <= ('12/31/2008') with compression = page on seg1,
Y2009 values <= ('12/31/2009') with compression = row on seg2,
Y2010 values <= ('12/31/2010') on seg3)
```

Use `sp_help` to view a table's compression level. This is the `sp_help` compression information for the `mail` table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Owner</th>
<th>Object_type</th>
<th>Object_status</th>
</tr>
</thead>
<tbody>
<tr>
<td>mail</td>
<td>dbo</td>
<td>user table</td>
<td>row level compressed, contains compressed data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Apr 8 2011 2:55PM</td>
</tr>
</tbody>
</table>

Disabling Compression

Set the compression level to `none` to remove data compression from a table or partition.

**Note:** Modifying a database's compression level, or enabling and disabling compression at table or partition level does not affect existing data; it affects only data you add or update after the change. However, changing whether a column is compressed or not performs a data copy, and therefore affects existing data.

Disable compression using:

```
alter table table_name set compression = none
```

See the *Reference Manual: Commands*.

To set the compression to `none` for the sales table, use:

```
alter table sales set compression = none
```

To disallow compression for the `order_num` column:

```
alter table sales modify order_num int not compressed
```

To remove compression from the `Y2008` and `Y2009` partitions:

```
alter table sales_date modify partition Y2008, Y2009 set compression = none
```
Altering the Compression Level of a Table or Partition

`alter table` does not affect the compression level of existing data, but affects the compression level of new or changed data rows produced by subsequent DML operations.

`alter table` lets you:

- Enable compression on uncompressed tables or partitions, and disable compression on already compressed tables or partitions.
- Change the compression type (**row** or **page**) for compressed tables.
- Alter a column in a compressed table to allow or disallow compression.

**Note:** You must set the compression level for a table before you can modify a column for compression.

Alter the compression level of existing tables or partitions using:

```sql
alter table table_name
{
 modify column [not] compressed
},
modify partition partition_name, [partition_name . . .]
set compression = {default | none | row | page}
},
{set compression = {none | page | row}}
```

See the *Reference Manual: Commands*.

This example alters the `sales_data` table for compression:

```sql
alter table sales_data
set compression = row
```

This example modifies the `isbn` column for compression:

```sql
alter table sales_data
modify isbn compressed
```
Creating a Compressed Table
Datatypes Available for Compression

Not all datatypes are eligible for data compression.

**Exact Numeric Integer Datatypes Eligible for Compression**

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Compression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigint</td>
<td>8</td>
<td>Row and page dictionary</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>Row and page dictionary</td>
</tr>
<tr>
<td>smallint</td>
<td>2</td>
<td>Page index</td>
</tr>
<tr>
<td>tinyint</td>
<td>1</td>
<td>Page index</td>
</tr>
<tr>
<td>ubigint</td>
<td>8</td>
<td>Row and page dictionary</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4</td>
<td>Row and page dictionary</td>
</tr>
<tr>
<td>unsigned smallint</td>
<td>2</td>
<td>Page index</td>
</tr>
</tbody>
</table>

- All exact numeric datatypes are compressed.
- Platform-specific big-endian and little-endian (most- and least-significant bytes) storage for exact numeric integers is in the specified number of bytes.

**Exact Numeric Decimal Datatypes Eligible for Compression**

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Compression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>numeric (precision, scale)</td>
<td>User-specified</td>
<td>Row and page dictionary</td>
</tr>
<tr>
<td>decimal (precision, scale)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- All exact numeric decimal datatypes are compressed.
- Storage format for exact numeric decimals is a byte stream storing 1 byte for precision, 1 byte for scale, and \( n \) number of bytes for data.

**Approximate Numeric Datatypes Eligible for Compression**

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Compressed?</th>
<th>Compression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>float (precision)</td>
<td>4 bytes if precisions &lt; 16, 8 if ( \geq 16 )</td>
<td>No</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Datatypes Available for Compression

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Compressed?</th>
<th>Compression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>double precision</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>real</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Money Datatypes Eligible for Compression

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Storage format</th>
<th>Compressed?</th>
<th>Compression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>money</td>
<td>8</td>
<td>Two 4-byte values: one signed int and the other an unsigned int</td>
<td>Yes</td>
<td>Row and page dictionary</td>
</tr>
<tr>
<td>smallmoney</td>
<td>4</td>
<td>1 signed 4-byte integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Date and Time Datatypes Eligible for Compression

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Storage Format</th>
<th>Compressed?</th>
<th>Compression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigdatetime</td>
<td>8</td>
<td>Represented as an unsigned 64-bit integer. Using a base date of 1/1/0001, bigdatetime holds the number of microseconds between midnight of the base date and a point in time. Stores fractions of a second to 6 decimal places.</td>
<td>Yes</td>
<td>Row and page dictionary</td>
</tr>
<tr>
<td>bigtime</td>
<td>8</td>
<td>8-byte unsigned integer holding the number of microseconds since midnight. Stores fractions of a second to 6 decimal places.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Datatype</td>
<td>Length, in Bytes</td>
<td>Storage Format</td>
<td>Compressed?</td>
<td>Compression Type</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
<td>----------------</td>
<td>-------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>date</td>
<td>4</td>
<td>Stores the number of days, backward or forward, from January 1, 1900.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>datetime</td>
<td>8</td>
<td>Two 4-byte parts. First part stores the number of days forward or backward from 1/1/1900. Second part stores the number of 1/300th seconds since midnight.</td>
<td>Yes</td>
<td>Page dictionary (date portion) and row compressed (time portion)</td>
</tr>
<tr>
<td>smalldate-time</td>
<td>4</td>
<td>Two 2-byte unsigned smallint values. First stores the number of days since 1/1/1900. Second stores the number of minutes since midnight.</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>time</td>
<td>4</td>
<td>The number of milliseconds since midnight.</td>
<td>Yes</td>
<td>Row page dictionary</td>
</tr>
</tbody>
</table>

**Character Datatypes Eligible for Compression**

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Storage Format</th>
<th>Compressed?</th>
<th>Compression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>char(n)</td>
<td>User-specified</td>
<td>Single or multiple byte or character stream, depending on the character type</td>
<td>Yes</td>
<td>Row, page dictionary if length ≥4. Page index, if length &lt; 4.</td>
</tr>
<tr>
<td>unichar(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nchar(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>varchar(n)</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Page dictionary if length ≥4. Page index, if length &lt; 4.</td>
</tr>
</tbody>
</table>
### Datatypes Available for Compression

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Storage Format</th>
<th>Compressed?</th>
<th>Compression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>uni-varchar-(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nvarchar-(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Binary Datatypes Eligible for Compression

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Storage Format</th>
<th>Compressed?</th>
<th>Compression Type</th>
</tr>
</thead>
</table>

### Other Datatypes Eligible for Compression

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Length, in Bytes</th>
<th>Storage Format</th>
<th>Compressed?</th>
<th>Compression Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit</td>
<td></td>
<td></td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>timestamp</td>
<td>8</td>
<td>Byte stream; binary data</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>xtype_token</td>
<td>User-specified</td>
<td></td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>text pointer</td>
<td>16 bytes of binary data</td>
<td>Byte stream, 8 bytes of RID, 8 bytes of first text page's database timestamp value.</td>
<td>No</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Compressed Data Storage Strategies

Pages in a compressed table may have a combination of row-compressed, page-compressed, and uncompressed data.

For example, if you create the `sales_data` table:

```sql
create table sales_data
(isbn bigint not null,
au_id varchar(11) not null,
total_sales int not null)
```

And insert this data:

<table>
<thead>
<tr>
<th>ISBN</th>
<th>AU ID</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>4750984443</td>
<td>903-94-9344</td>
<td>34733</td>
</tr>
<tr>
<td>2385837442</td>
<td>346-94-5593</td>
<td>50945</td>
</tr>
<tr>
<td>2388347442</td>
<td>346-94-5593</td>
<td>50945</td>
</tr>
</tbody>
</table>

`sales_data` is uncompressed:

```
Uncompressed rows [ ]
```

```
Compressed rows [ ]
```

However, if you alter `sales_data` for compression:

```sql
alter table sales_data
set compression = row
```

And insert this data:

<table>
<thead>
<tr>
<th>ISBN</th>
<th>AU ID</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>4783023685</td>
<td>887-49-9984</td>
<td>45009</td>
</tr>
<tr>
<td>3894350422</td>
<td>776-45-9045</td>
<td>89667</td>
</tr>
<tr>
<td>3349580094</td>
<td>884-59-9983</td>
<td>84855</td>
</tr>
</tbody>
</table>

Only the new data is compressed:
Compressed Data Storage Strategies

However, if you alter sales_data again to be uncompressed:

```
alter table sales_data set compression = none
```

And insert this data:

```
6590345093, '439-49-9943', 485844
3458940330, '559-40-3999', 21003
4859390403, '884-30-0200', 790499
```

Adaptive Server does not compress the new data, but retains the older data in a compressed state:
Compressed Data Storage Strategies

**sp_help** reports whether a table has ever contained compressed data. This is the **sp_help** output for `sales_data`:

```
Name       Owner Object_type Object_status            Create_date
---------- ----- ----------- ------------------------
-------------------
sales_data dbo   user table  contains compressed data Apr  8 2011 4:36PM
```

Compressed Data Storage Strategies
Compressed Columns with Large Objects

Adaptive Server lets you create databases and compress columns that use the text, image, unitext, and java large object (LOB) datatypes.

LOB columns can contain up to 2,147,483,647 (or $2^{31} -1$) bytes of character or binary data. Adaptive Server stores LOB values on a text page chain. Adaptive Server compresses only text pages.

Adaptive Server uses the FastLZ (with LZO) and ZLib (with LZW,26) algorithm to compress LOB data. Both are dictionary-based compression techniques; that is, they replace repeated words on the data page with a status bit that points to the actual word in an index. The differences are:

- FastLZ – lower CPU usage and execution time.
- ZLib – higher compression ratio.

Adaptive Server automatically determines the algorithm to use when you select the compression level. Levels 1 – 9 use the ZLib technique, and levels 100 and 101 use the FastLZ technique.

Generally, the higher the compression level, the more the LOB is compressed. However, the amount of compression depends on the content of the LOB. The higher the compression level, the more CPU-intensive the process, so a compression_level of 9 provides the best compression ratio, but also the heaviest CPU usage.

You can combine table-level and column-level compression.

<table>
<thead>
<tr>
<th>Compression Level</th>
<th>No Column Compression</th>
<th>Column is not compressed</th>
<th>Column uses compression_level Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>No table-level compression</td>
<td>Uncompressed</td>
<td>Uncompressed</td>
<td>Column-level compression</td>
</tr>
<tr>
<td>lob_compression = 0</td>
<td>Uncompressed</td>
<td>Uncompressed</td>
<td>Column-level compression</td>
</tr>
<tr>
<td>lob_compression is the same as the table-level compression</td>
<td>Column level compression</td>
<td>Uncompressed</td>
<td>Column-level compression</td>
</tr>
</tbody>
</table>

Adaptive Server alters the page layout when it compresses LOB columns.
Creating a Compressed Database with LOB Datatypes

All tables in a database inherit the compression level you specify for LOB columns.

1. Select a compression level to determine the database's compression strategy:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression level</td>
<td>Strategy</td>
</tr>
<tr>
<td>1 - 9, where 9 provides the best compression ratio but heaviest CPU usage</td>
<td>Higher compression ratio (ZLib algorithm)</td>
</tr>
<tr>
<td>100 or 101</td>
<td>Lower CPU usage and execution time (FastLZ algorithm)</td>
</tr>
</tbody>
</table>

2. Create a database with LOB datatypes using:

```sql
create database database_name
[...]
with dml_logging = { minimal | full }
, durability =
{ no_recovery | at_shutdown | full }
, lob_compression = off | compression_level
```

The `lob_compression` parameter indicates that all tables in the database inherit the specified level of compression for LOB columns.

This creates the `email_lob_db`, which is configured for a LOB compression level of 101:

```sql
create database email_lob_db
on email_lob_dev = '50M'
with lob_compression = 101
```

Creating Compressed Tables with LOB Datatypes

You need not compress all columns in the table.

1. Select a compression level to determine the table's compression strategy:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression level</td>
<td>Strategy</td>
</tr>
<tr>
<td>1 - 9, where 9 provides the best compression ratio but heaviest CPU usage</td>
<td>Higher compression ratio (ZLib algorithm)</td>
</tr>
<tr>
<td>100 or 101</td>
<td>Lower CPU usage and execution time (FastLZ algorithm)</td>
</tr>
</tbody>
</table>
2. Create a table with LOB compression using:

```sql
create table table_name (  
  column_name data_type  
  [compressed = compression_level | not compressed]  
  ...  
)  
[with lob_compression = compression_level]
```

The `compressed =` parameter controls column-level compression; `with lob_compression =` controls table-level compression.

This example creates a compressed table that includes LOB data:

```sql
create table mail(user_name char(10),  
  mailtxt text compressed = 5,  
  photo image compressed = 1,  
  reply_mail text compressed = 9,  
  attachment image compressed = 100)  
lock datarows  
with lob_compression = 0
```

### Altering Tables with Compressed LOB Datatypes

Use `alter table` command to enable or disable a table's compression.

1. Select a compression level to determine the compression strategy for the table:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compression level</strong></td>
<td><strong>Strategy</strong></td>
</tr>
<tr>
<td>1 - 9, where 9 provides the best compression ratio but heaviest CPU usage</td>
<td>Higher compression ratio (ZLib algorithm)</td>
</tr>
<tr>
<td>100 or 101</td>
<td>Lower CPU usage and execution time (FastLZ algorithm)</td>
</tr>
</tbody>
</table>

2. Alter a LOB table's compression level using:

```sql
alter table table_name  
add column_name datatype ...  
[compressed = compression_level | not compressed]  
| set  
  [, lob_compression = off | compression_level ]  
| modify column_name ...  
  [compressed = compression_level | not compressed ]
```

This alters the compression level of the titles table to row:

```sql
alter table titles set compression = row
```
Compressed Columns with Large Objects
Select Data into Compressed Tables

Use `select into ... compression` to select regular and LOB data directly into a compressed table.

The destination table does not inherit anything from the original table. That is, if the table from which you are selecting data is page-level compressed, the table into which you select the data can be row-level compressed, or not compressed.

You must indicate compression levels if you are selecting large object data into a table.

The behavior of `select into` on target tables or columns depends on the type of compression you are using.

<table>
<thead>
<tr>
<th>Compression Type</th>
<th>Behavior of select into on Target Tables</th>
<th>Source Table or Column</th>
<th>Database-wide Setting for compression</th>
<th>Target Table or Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Compression</td>
<td>Target table or columns do not inherit any properties from the source table. If you do not specify <code>compression</code>, tables other than temporary tables inherit the database-wide setting for <code>compression</code>. Temporary tables do not inherit any compression settings from the source table, source column, or from the target databases’s attributes.</td>
<td>Table can be compressed or uncompressed, and may include one or more compressed columns.</td>
<td>none</td>
<td>Target table and all columns are uncompressed.</td>
</tr>
<tr>
<td>LOB compression</td>
<td>LOB columns in the target table do not inherit any properties from the source columns. If you do not specify <code>compression</code>, LOB columns in target tables other than temporary tables inherit the database-wide setting for <code>compression</code>. LOB columns in the target table may be compressed.</td>
<td>Source LOB columns may be compressed.</td>
<td><code>lob_compression = 0</code>, unset for the database</td>
<td>All LOB columns in the target table are uncompressed.</td>
</tr>
<tr>
<td>Compression Type</td>
<td>Behavior of select into on Target Tables</td>
<td>Source Table or Column</td>
<td>Database-wide Setting for compression</td>
<td>Target Table or Column</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------</td>
<td>------------------------</td>
<td>--------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lob_compression = compression_level</td>
<td>All LOB columns in the target table are created using the database-wide setting for lob_compression = compression_level.</td>
</tr>
</tbody>
</table>

This example selects all rows from the `titles` table, and creates a new table named `titles_2` with row-level compression:

```
select * into titles2
with compression = row
from titles
```

See the *Reference Manual: Commands*. 
Administering Compressed Databases

Administration duties for compressed databases include enabling or disabling session compression, bulk-copying, and dumping and loading compressed data.

Use the

- **compression info pool size** configuration parameter to check the memory pool for compression.
- **capture compression statistics** to enable the monTableCompression monitoring table to begin capturing compression statistics.

See the *System Administration Guide: Volume 1.*

Session-Level Data Compression

Enable and disable compression for a session with the `set` command.

To enable compression for the current session, use:

```
set compression {on | off | default}
```

This command has no effect on uncompressed tables. When you enable compression for a session, Adaptive Server compresses all subsequent data inserted in the table that uses the appropriate datatype. If you set compression off, Adaptive Server disables compression for the duration of the session. When you set compression to `default`, Adaptive Server uses the compression configuration you established when you created the table.

Adaptive Server does not support session-level compression for LOB compression.

Stored or system procedures inherit a session’s compression settings. Subprocedures inherit the `set compression` command settings executed in the parent procedure. When the procedure ends, Adaptive Server restores the compression level of the outer session or parent procedure.

`set compression` changes included with login triggers apply to the session established when you first log in until you explicitly change the compression level. You need not enable `set export_options` in the login trigger to export `set compression` changes. Once the compression level is exported to a session, it applies to individual tables. However, `set compression` is not exported to the immediate parent procedure’s context if you issue `set export_options` in a nested procedure before setting issuing `set compression`.

Copy, Dump, and Load Compressed Data

Use `bcp` to bulk-copy compressed data in and out of tables.

Pages in a compressed table may have a combination of row-compressed, page-compressed, or uncompressed rows. Even tables or partitions marked as uncompressed can include data that is a mixture of different states of compression.

- **bcp out** – any compressed rows (including those with text data) are decompressed and returned to the client, either in native or character form.
- **bcp in** – uncompressed data received from the client is compressed during the insert. `bcp in` selects the appropriate compression scheme, which depends on the compression level of the partition into which you are inserting the row.

When you bulk-copy data out (using `bcp out`), followed by a `bcp in` to a compressed table (or partition), all newly loaded data is compressed, even when the extracted data was stored as uncompressed.

See *Utility Guide > Utility Commands Reference > bcp*

`dump database` dumps compressed data directly from disk to archive. If the transaction log contains compressed LOB data, recover the compressed LOB data with `load tran` (see the *System Administration Guide: Volume 2 > Developing a Backup and Recovery Plan*).
Limits for Database Compression

Database compression includes limitations on replicating compressed data and in-memory databases.

- Generally, compression is restricted for in-memory databases. Loading and recovering compressed objects in disk-resident or relaxed-durability in-memory databases is permitted. However, Adaptive Server often restricts access to compressed objects in the target in-memory database. Adaptive Server provides minimal support for disabling compression in the target database or in tables defined for compression, so you may revert to using uncompressed data.

- Compressed LOB columns do not support replication. Issue the following to indicate that a column is not to be replicated before you compress columns with LOB data that are part of a replicated database:

  ```
  sp_setrepcol table_name, lob_column_name, 'do_not_replicate'
  ```

See the *Replication Server Reference Manual*. 
Limits for Database Compression
Index

A
alter database 11
alter table
    altering the compression level 15
    disabling compression 14
    examples 27
    for compressed tables with LOB datatypes 27
altering LOB compression 27
altering tables with fixed-length data 7
approximate numeric datatypes, compressing 17
ASE_COMPRESSION license 1

B
bcp in and bcp out 4, 32
big-endian 17
binary, compressing 17
bit, compressing 17

C
character datatypes, compressing 17
compressing large objects (LOBs) 25
compression level
    viewing with sp_help 13, 21
compression, enabling 1
CPU and data compression 1
create database
    creating compressed databases with LOB datatypes 26
    examples 11
create table
    creating compressed tables with LOB datatypes 26
creating 13
creating tables with LOB compression 26

D
data compression
    alter tables for 7
    altering the compression level 11, 15
    and CPU 1
    copy data in and out 4, 32
creating compressed tables and partitions 13
creating database for 11
datatypes available 17
disabling 14
enabling 1
fixed-length data and page-level compression 8
for fixed-length data 7
hot data 1
in-memory databases 5, 33
license required 1
limits for 5, 33
mix of compressed and uncompressed data 1
model database 11
overview 1
page-level 8
querying 1
replicating compressed data 5, 33
row-level 7
select data into compressed tables 2, 29
select into 2, 29
setting enable compression 1
setting session-level compression 3, 31
storage strategies 21
trailing blanks 7
datatypes available for compression 17
date and time datatypes, compressing 17
dump database 4, 32

E
enable compression, setting 1
enabling compression 1
exact numeric datatype, compressing 17
exact numeric decimal, compressing 17

F
FastLZ (with LZO) compression 25–27
fixed-length data
    altering tables for 7
    page-level compression 8
    trailing zeros 7
    uncompressed data types 7
H
hot data 1

I
image data, compressing 25
in-memory databases and data compression 5, 33

L
large object (LOB) compression
  altering tables 27
  creating databases 26
  creating tables 26
  datatypes supported 25
  FastLZ (with LZO) algorithm 25
  select data into compressed LOB tables 2, 29
  selecting the database's compression level 26
  selecting the table's compression level 26, 27
  setting session-level compression 3, 31
  supported compression levels 25
  ZLib (with LZW.26) algorithm 25
license for compression 1
little-endian 17
load tran 4, 32
LOB compression
  See large object (LOB) compression

M
model database default compression level 11
money datatypes, compressing 17

P
page index 8
page-level compression
  altering 15
  and tables 13
  examples 8
  fixed-length data 8
  page dictionary 8
  page index 8
  techniques for compression 8
  when avoided 8
partitions
  altering the compression level 15
partitions, compressed
  creating 13
disabling 14

Q
querying compressed data 1

R
replicating compressed data 5, 33
row-level compression
  altering 15
  and fixed-length data 7
  and tables 13
  and trailing blanks 7
  examples 7
  uncompressed data types 7

S
select data into compressed tables 2, 29
select into
  example 2, 29
session, setting the compression level 3, 31
set command
  setting the compression level for the session 3, 31
sp_help 13, 21
storage strategies, compression 21

T
tables, compressed 13
  disabling compression 14
  text data, compressing 25
  text pointer 17
timestamp, compressing 17
  trailing blanks 7

U
unitext data, compressing 25

X
 xtype_token 17
Z
ZLib (with LZW.26) compression 25–27