<https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/row-compression-implementation>

# Row Compression Implementation

2016-6-30 6 min to read Contributors

* [BYHAM](https://github.com/BYHAM)

* [Craig Guyer](https://github.com/craigg-msft)

**THIS TOPIC APPLIES TO:** yesSQL Server (starting with 2016)yesAzure SQL DatabasenoAzure SQL Data Warehouse noParallel Data Warehouse

This topic summarizes how Database Engine implements row compression. This summary provides basic information to help you plan the storage space that you need for your data.

Enabling compression only changes the physical storage format of the data that is associated with a data type but not its syntax or semantics. Application changes are not required when one or more tables are enabled for compression. The new record storage format has the following main changes:

* It reduces the metadata overhead that is associated with the record. This metadata is information about columns, their lengths and offsets. In some cases, the metadata overhead might be larger than the old storage format.
* It uses variable-length storage format for numeric types (for example **integer**, **decimal**, and **float**) and the types that are based on numeric (for example **datetime** and **money**).
* It stores fixed character strings by using variable-length format by not storing the blank characters.

Note

NULL and 0 values across all data types are optimized and take no bytes.

## How Row Compression Affects Storage

The following table describes how row compression affects the existing types in SQL Server and Azure SQL Database. The table does not include the savings that can be achieved by using page compression.

| Data type | Is storage affected? | Description |
| --- | --- | --- |
| **tinyint** | No | 1 byte is the minimum storage needed. |
| **smallint** | Yes | If the value fits in 1 byte, only 1 byte will be used. |
| **int** | Yes | Uses only the bytes that are needed. For example, if a value can be stored in 1 byte, storage will take only 1 byte. |
| **bigint** | Yes | Uses only the bytes that are needed. For example, if a value can be stored in 1 byte, storage will take only 1 byte. |
| **decimal** | Yes | This storage is exactly same as the vardecimal storage format. |
| **numeric** | Yes | This storage is exactly same as the vardecimal storage format. |
| **bit** | Yes | The metadata overhead brings this to 4 bits. |
| **smallmoney** | Yes | Uses the integer data representation by using a 4-byte integer. Currency value is multiplied by 10000 and the resulting integer value is stored by removing any digits after the decimal point. This type has a storage optimization similar to that for integer types. |
| **money** | Yes | Uses the integer data representation by using an 8-byte integer. Currency value is multiplied by 10000 and the resulting integer value is stored by removing any digits after the decimal point. This type has a larger range than **smallmoney**. This type has a storage optimization similar to that for integer types. |
| **float** | Yes | Least significant bytes with zeros are not stored. **float** compression is applicable mostly for nonfractional values in mantissa. |
| **real** | Yes | Least significant bytes with zeros are not stored. **real** compression is applicable mostly for nonfractional values in mantissa. |
| **smalldatetime** | No | Uses the integer data representation by using two 2-byte integers. The date takes 2 bytes. It is the number of days since 1/1/1901. This needs 2 bytes starting from 1902. Therefore, there is no savings after that point.  The time is the number of minutes since midnight. Time values that are slightly past 4AM start to use the second byte.  If a **smalldatetime** is only used to represent a date (a common case), the time is 0.0. Compression saves 2 bytes by storing the time in most significant byte format for row compression. |
| **datetime** | Yes | Uses the integer data representation by using two 4-byte integers. The integer value represents the number of days with base date of 1/1/1900. The first 2 bytes can represent up to the year 2079. Compression can always save 2 bytes here until that point. Each integer value represents 3.33 milliseconds. Compression exhausts the first 2 bytes in first five minutes and needs the fourth byte after 4PM. Therefore, compression can save only 1 byte after 4PM. When **datetime** is compressed like any other integer, compression saves 2 bytes in the date. |
| **date** | No | Uses the integer data representation by using 3 bytes. This represents the date from 1/1/0001. For contemporary dates, row compression uses all 3 bytes. This achieves no savings. |
| **time** | No | Uses the integer data representation by using 3 to 6 bytes. There are various precisions that start with 0 to 9 that can take 3 to 6 bytes. Compressed space is used as follows:  **Precision = 0. Bytes = 3**. Each integer value represents a second. Compression can represent time up to 6PM by using 2 bytes, potentially saving 1 byte.  **Precision = 1. Bytes = 3**. Each integer value represents 1/10 seconds. Compression uses the third byte before 2AM. Results in little savings.  **Precision = 2. Bytes = 3**. Similar to the previous case, it is unlikely to achieve savings.  **Precision = 3. Bytes = 4**. Because the first 3 bytes are taken by 5AM, achieves little savings.  **Precision = 4. Bytes = 4**. The first 3 bytes are taken in the first 27 seconds. No savings are expected.  **Precision = 5, Bytes = 5**. Fifth byte will be used after 12-noon.  **Precision = 6 and 7, Bytes = 5**. Achieves no savings.  **Precision = 8, Bytes = 6**. Sixth byte will be used after 3AM.    Note that there is no change in storage for row compression. Overall, not much savings can be expected from compressing the **time** data type. |
| **datetime2** | Yes | Uses the integer data representation by using 6 to 9 bytes. The first 4 bytes represent the date. The bytes taken by the time will depend on the precision of the time that is specified.  The integer value represents the number of days since 1/1/0001 with an upper bound of 12/31/9999. To represent a date in year 2005, compression takes 3 bytes.  There is no savings on time because it allows for 2 to 4 bytes for various time precisions. Therefore, for one-second time precision, compression uses 2 bytes for time, which takes the second byte after 255 seconds. |
| **datetimeoffset** | Yes | Resembles **datetime2**, except that there are 2 bytes of time zone of the format (HH:MM).  Like **datetime2**, compression can save 2 bytes.  For time zone values, MM value might be 0 for most cases. Therefore, compression can possibly save 1 byte.  There are no changes in storage for row compression. |
| **char** | Yes | Trailing padding characters are removed. Note that the Database Engine inserts the same padding character regardless of the collation that is used. |
| **varchar** | No | No effect. |
| **text** | No | No effect. |
| **nchar** | Yes | Trailing padding characters are removed. Note that the Database Engine inserts the same padding character regardless of the collation that is used. |
| **nvarchar** | No | No effect. |
| **ntext** | No | No effect. |
| **binary** | Yes | Trailing zeros are removed. |
| **varbinary** | No | No effect. |
| **image** | No | No effect. |
| **cursor** | No | No effect. |
| **timestamp** / **rowversion** | Yes | Uses the integer data representation by using 8 bytes. There is a timestamp counter that is maintained for each database, and its value starts from 0. This can be compressed like any other integer value. |
| **sql\_variant** | No | No effect. |
| **uniqueidentifier** | No | No effect. |
| **table** | No | No effect. |
| **xml** | No | No effect. |
| User-defined types | No | This is represented internally as **varbinary**. |
| FILESTREAM | No | This is represented internally as **varbinary**. |

## See Also

[Data Compression](https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/data-compression)  
[Page Compression Implementation](https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/page-compression-implementation)