MS SQL

May 28, 2003

Choosing SQL Server 2000 Data Types

By Alexander Chigrik

Introduction

Choosing an appropriate data type is very important, because the errors made in a table design can result in large performance degradation. These problems often surface later, when a large amount of data is inserted. In this article, I want to tell you about built-in and user-defined data types, how SQL Server 2000 stores data on a data page, and show some general tips to choose an appropriate data type.

Built-in data types

In Microsoft SQL Server 2000, each object (such as column, variable, or parameter) has a related data type, which is an attribute that specifies the type of data that the object can hold.

SQL Server 2000 ships with 27 built-in (system) data types. They are:

Data Types	Description
bigint	Integer data from -2^63 through 2^63-1
int	Integer data from -2^31 through 2^31 - 1
smallint	Integer data from -2^15 through 2^15 - 1
tinyint	Integer data from 0 through 255
bit	Integer data with either a 1 or 0 value
decimal	Fixed precision and scale numeric data from -10^38 +1 through 10^38 -1
numeric	Fixed precision and scale numeric data from -10^38 +1 through 10^38 -1
money	Monetary data values from -2^63 through 2^63 - 1
smallmoney	Monetary data values from -214,748.3648 through +214,748.3647
float	Floating precision number data from -1.79E + 308 through 1.79E + 308
real	Floating precision number data from -3.40E + 38 through 3.40E + 38
datetime	Date and time data from January 1, 1753, through December 31, 9999, with an accuracy of 3.33 milliseconds
smalldatetime	Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute
char	Fixed-length character data with a maximum length of 8,000 characters

varchar	**Variable-length data with a maximum of 8,000 characters**
text	**Variable-length data with a maximum length of 2^31 - 1 characters**
nchar	**Fixed-length Unicode data with a maximum length of 4,000 characters**
nvarchar	**Variable-length Unicode data with a maximum length of 4,000 characters**
ntext	**Variable-length Unicode data with a maximum length of 2^30 - 1 characters**
binary	**Fixed-length binary data with a maximum length of 8,000 bytes**
varbinary	**Variable-length binary data with a maximum length of 8,000 bytes**
image	**Variable-length binary data with a maximum length of 2^31 - 1 bytes**
cursor	**A reference to a cursor**
sql_variant	**A data type that stores values of various data types, except text, ntext, timestamp, and sql_variant**
table	**A special data type used to store a result set for later processing**
timestamp	**A database-wide unique number that gets updated every time a row gets updated**
uniqueidentifier	**A globally unique identifier**

Some of these data types (bigint, sql_variant, and table) are only available in SQL Server 2000, while some were supported under the previous SQL Server versions.

User-defined data types

SQL Server 2000 supports user-defined data types too. User-defined data types provide a mechanism for applying a name to a data type that is more descriptive of the types of values to be held in the object. Using user-defined data type can make it easier for a programmer or database administrator to understand the intended use of any object defined with the data type. The user-defined data types are based on the system data types and can be used to predefine several attributes of a column, such as its data type, length, and whether it supports NULL values. To create a user-defined data type, you can use the **sp_addtype** system stored procedure or you could add one using the Enterprise Manager. When you create a user-defined data type, you should specify the following three properties:

- Data type's name.
- Built-in data type upon which the new data type is based.
- Whether it can contain NULL values.

The following example creates a user-defined data type based on money data type named **cursale** that cannot be NULL:

```
EXEC sp_addtype cursale, money, 'NOT NULL'
GO
```

Both system and user-defined data types are used to enforce data integrity. It is very important that we put forth a lot of effort while designing tables: the better you design your tables, the more time you can work without any performance problems. In an ideal case, you never will update the structure of your tables.

Tips to choose the appropriate data types

SQL Server 2000 stores data in a special structure called data pages that are 8Kb (8192 bytes) in size. Some space on the data pages is used to store system information, which leaves 8060 bytes to store user's data. So, if the table's row size is 4040 bytes, then only one row will be placed on each data page. If you can decrease the row size to 4030 bytes, you can store two rows within a single page. The less space used, the smaller the table and index, and the less the I/O SQL Server has to perform when reading data pages from disk. So, you should design your tables in such a way as to maximize the number of rows that can fit into one data page. To maximize the number of rows that can fit into one data page, you should specify the narrowest columns you can. The narrower the columns, the less data that is stored, and the faster SQL Server is able to read and write data.

Try to use the following tips when choose the data types:

- **If you need to store integer data from 0 through 255, use *tinyint* data type.** The columns with tinyint data type use only one byte to store their values, in comparison with two bytes, four bytes and eight bytes used to store the columns with smallint, int and bigint data types accordingly. For example, if you design tables for a small company with 5-7 departments, you can create the departments table with the DepartmentID tinyint column to store the unique number of each department.
- **If you need to store integer data from -32,768 through 32,767, use *smallint* data type.** The columns with smallint data type use only two bytes to store their values, in comparison with four bytes and eight bytes used to store the columns with int and bigint data types accordingly. For example, if you design tables for a company with several hundred employees, you can create an employee table with the EmployeeID smallint column to store the unique number of each employee.
- **If you need to store integer data from -2,147,483,648 through 2,147,483,647, use *int* data type.** The columns with int data type use only four bytes to store their values, in comparison with eight bytes used to store the columns with bigint data types. For example, to design tables for a library with more than 32,767 books, create a books table with a BookID int column to store the unique number of each book.
- **Use *smallmoney* data type instead of money data type, if you need to store monetary data values from 214,748.3648 through 214,748.3647.** The columns with smallmoney data type use only four bytes to store their values, in comparison with eight bytes used to store the columns with money data types. For example, if you need to store the monthly employee payments, it might be possible to use a column with the smallmoney data type instead of money data type.
- **Use *smalldatetime* data type instead of datetime data type, if you need to store the date and time data from January 1, 1900 through June 6, 2079, with accuracy to the minute.** The columns with smalldatetime data type use only four bytes to store their values, in comparison with eight bytes used to store the columns with datetime data types. For example, if you need to store the employee's hire date, you can use a column with the smalldatetime data type instead of datetime data type.
- **Use varchar/nvarchar columns instead of text/ntext columns whenever possible.** Because SQL Server stores text/ntext columns on the

Text/Image pages separately from the other data, stored on the Data pages, it can take more time to get the text/ntext values.

- **Use char/varchar columns instead of nchar/nvarchar if you do not need to store unicode data.** The char/varchar value uses only one byte to store one character, the nchar/nvarchar value uses two bytes to store one character, so the char/varchar columns use two times less space to store data in comparison with nchar/nvarchar columns.