



Special Data Types



Rowversion Or Timestamp Data Type



Rowversion or Timestamp ?

- ROWVERSION and TIMESTAMP are synonyms
- ROWVERSION is available since SQL Server 2005
- TIMESTAMP is deprecated and will be removed in a future version of SQL Server



Rowversion Data Type

- Automatically generated, unique binary numbers within a database
 - Has nothing to do with Dates or times
- **Binary numbers** that indicate the relative sequence in which data modifications have occurred in a database
- Generally used as a mechanism for version-stamping table rows
- Never use timestamp columns in keys, especially primary keys
 - because the timestamp value changes every time that the row is modified.



Rowversion Data Type (cont.)

- Rowversion or Timestamp is NOT a date or a time
 - It's a binary number that gets incremented by SQL whenever the row is updated
 - It's a row version, not a date
 - It cannot go into a datetime column
- ROWVERSION data types represent automatically generated binary numbers, unique within the database
 - It is just an incrementing number and does not preserve a date or a time



Rowversion Data Type (cont.)

- The storage size is 8 bytes
- Only One ROWVERSION column is allowed per table
- When you insert or update a row
 - a new unique number across the database is generated and assigned to the ROWVERSION column
 - This property makes a rowversion column a poor candidate for keys, especially primary keys



Example

```
CREATE TABLE testDataType
(  
    theID          INT identity PRIMARY KEY  
    , rVer          ROWVERSION  
    , intVal        INT  
)  
GO
```

```
INSERT testDataType (intVal) VALUES( 11)  
INSERT testDataType (intVal) VALUES( 12)
```

```
SELECT * FROM testDataType
```

<u>theID</u>	<u>rVer</u>	<u>intVal</u>
1	0x0000000000000007D6	11
2	0x0000000000000007D7	12

Example (cont.)

```
SELECT * FROM testDataType
```

<u>theID</u>	<u>rVer</u>	<u>intVal</u>
1	0x0000000000000007D6	11
2	0x0000000000000007D7	12

```
UPDATE testDataType  
SET intVal = 99  
WHERE theID = 2
```

```
SELECT * FROM testDataType
```

<u>theID</u>	<u>rVer</u>	<u>intVal</u>
1	0x0000000000000007D6	11
2	0x0000000000000007D8	99



sql_variant Data Type



sql_variant Data Type

- sql_variant is a data type that stores values of various SQL Server-supported data types
- A column of type sql_variant may contain rows of different data types
- For example, a column defined as sql_variant can store int, binary, and char values.
- sql_variant can have a maximum length of 8016 bytes.
- A table can have any number of sql_variant columns



sql_variant - Example

- Using sql_variant data type in defining a column

```
CREATE TABLE testSqlDataType  
(  
    theID          INT identity PRIMARY KEY  
    , myData       sql_variant  
)  
GO
```

```
INSERT testSqlDataType (myData)  
VALUES( 12)
```

```
INSERT testSqlDataType (myData)  
VALUES('myName')
```



sql_variant - Example

- Using sql_variant data type in a script

```
DECLARE @v1 sql_variant
```

```
SET @v1 = 44
```

```
SELECT @v1
```

```
SET @v1 = 'ABC'
```

```
SELECT @v1
```



sql_variant - Example

```
DECLARE @v1 sql_variant  
SET @v1 = 44
```

```
SELECT    SQL_VARIANT_PROPERTY(@v1, 'BaseType') AS theType,  
          SQL_VARIANT_PROPERTY(@v1, 'MaxLength') AS theLength
```

```
SET @v1 = 'ABC'
```

```
SELECT    SQL_VARIANT_PROPERTY(@v1, 'BaseType') AS theType,  
          SQL_VARIANT_PROPERTY(@v1, 'MaxLength') AS theLength
```

OUTPUT

theType	theLength
int	4
varchar	3



Sysname Data Type



Sysname Data Type

- Sysname is a built in datatype that is used primarily to store object names when creating scripts
- The sysname data type can also be used for table columns, variables, and stored procedure parameters that store object names
- The exact definition of sysname is related to the rules for identifiers and can vary between instances of SQL Server.
 - sysname is functionally the same as nvarchar(128).
 - In earlier versions of SQL Server, sysname is defined as varchar(30)



Sysname

- If you want to store object names in a table, then while designing the table you can specify datatype of the column as sysname instead of nvarchar(128)
- Thus the datatype sysname will act as:
 - Datatype of nvarchar(128)
 - NOT NULL constraint
 - Length constraint to limit object name to 128 characters



Example

```
CREATE TABLE testSysNameDataType
(
    theID          INT identity PRIMARY KEY
    , myData       SYSNAME
    , myInt        INT
)
```

```
INSERT testSysNameDataType (myData, myInt)
VALUES( 'abc' , 234 )
```

```
INSERT testSysNameDataType (myInt)
VALUES( 234 )                -- ERROR
```



Sysname

- Some of the features of SYSNAME are:
 - It can store object names from the SQL database, such as functions, stored procedures, tables, users, server names, databases, login, and views.
 - When declaring the SYSNAME data type, the object names assigned to it must be quoted.
 - It is used internally by SQL Server to define certain sys tables columns and stored procedures
 - It is usually used in dynamic SQL to store the name names of database objects



Sysname - Example

- When using dynamic sql it is appropriate to use sysname as data type for variables holding table names, column names and server names.

```
DECLARE      @TableName SYSNAME
```

```
DECLARE      @SQL NVARCHAR(4000)
```

```
SET      @TableName = 'Employee'
```

```
SET      @SQL = 'SELECT * FROM ' + @TableName
```

```
EXEC      (@SQL)
```



Sysname - Example

- You can pass a table name to useful system SP's

```
DECLARE    @Table sysname;  
SET        @Table = 'TableName';
```

```
EXEC       sp_fkeys @Table;  
EXEC       sp_help @Table;
```



Sysname - Example

```
DECLARE @TableName SYSNAME
DECLARE @SQL NVARCHAR(4000)
SET @TableName = 'Employee'

SET @SQL = '
CREATE TRIGGER [dbo].[trigDeleteEmployee3] ON ' + @TableName + '
AFTER DELETE
AS
BEGIN
    UPDATE      E2
    SET          E2.Salary = 13000
    FROM         Employee2 E2 JOIN Deleted D
               ON      E2.EmpID = D.EmpID
    WHERE        E2.EmpID = 3
END'
EXEC (@SQL)
GO
```



Sysname - Example

- Using sysname as a parameter in a stored procedure

```
CREATE PROC getINFO
    @tblName sysname
AS
BEGIN
    DECLARE @SQL NVARCHAR(4000)

    SET @SQL = 'SELECT * FROM ' + @tblName
    EXEC (@SQL)
END
GO

EXEC getINFO    'Employee'
```



User-Defined Data Type



User Defined Data Types

- There are some situations when a customized data type is needed for consistency across an application
- They can be considered customized SQL Server data types
- User defined types are also called Alias data types
- Examples could be phone numbers in a specific format, alpha numeric employee ID's, IP addresses, etc.
- User defined type allow developers to use the same type for the same type of columns



User Defined Data Types (cont.)

- We can create SQL Server user defined data types and bind them to an existing default and rule
- Using your own defined data types with rules and defaults will provide a powerful mechanism to implement uniformity and consistency in your database design.
- A user defined data type is only accessible in a single database as such
 - Store the definition in a script so that it can be used whenever required
 - Create the objects in the model database, so as new databases are created the user defined data types, rules, defaults, etc. will automatically be available



CREATE Type Statement

- Creates a user-defined type in the current database
- Syntax

```
CREATE TYPE type_name  
    FROM base_type  
    [ ( precision [ , scale ] ) ]  
    [ NULL | NOT NULL ]
```

- base_type: Is the SQL Server supplied data type on which the alias data type is based
- To modify a user-defined type, you must drop the type by using a DROP TYPE statement and then re-create it.



Examples

- Creating an user-defined type based on the varchar data type

```
CREATE TYPE TypeName  
FROM VARCHAR(100)
```

```
CREATE TYPE StudCode  
FROM CHAR(6) NOT NULL
```



sp_addtype

■ Syntax

```
sp_addtype      Typename ,  
system_data_type  
[ 'null_type' ]
```

■ Examples

```
USE master;
```

```
GO
```

```
EXEC sp_addtype ssn, 'varchar(11)', 'NOT NULL';
```

```
EXEC sp_addtype birthday, datetime, 'NULL';
```



CREATE DEFAULT

- Creates an object called a default
- Default specifies a value to be inserted into the column to which the object is bound (or into all columns, if of an alias data type), when no value is explicitly supplied during an insert.
- Default can be bound to a column or an alias data type
- Syntax

```
CREATE DEFAULT default_name  
AS constant_expression
```



CREATE DEFAULT - Examples

- A. Creating a simple character default

- The following example creates a character default called unknown.

```
CREATE DEFAULT phonedflt AS 'unknown';
```

- Binding a default

```
sp_bindefault 'phonedflt', 'PersonPhone.PhoneNumber';
```



sp_bindefault

- Binds a default to a column or to an alias data type
- Syntax

`sp_bindefault        defname , 'object_name' , [ 'futureonly_flag' ]`

- Binding a default to a column

`EXEC sp_bindefault 'defaultXX', 'Employee.ColumnXY'`

- Binding a default to an alias data type

`EXEC sp_bindefault 'def_ssn', 'ssn';`



CREATE RULE

- Creates an object called a rule
- When bound to a column or an alias data type, a rule specifies the acceptable values that can be inserted into that column
- A rule can be any expression valid in a WHERE clause
- It can include
 - elements such as arithmetic operators
 - predicates (for example, IN, LIKE, BETWEEN)



CREATE RULE (cont.)

■ Syntax

```
CREATE RULE    rule_name  
AS condition_expression
```

- The **condition_expression** includes one variable
- The expression refers to the value entered with the UPDATE or INSERT statement via this variable
- Any name or symbol can be used to represent the value when creating the rule
 - But the first character must be the at sign (@)



CREATE RULE - Examples

■ Creating a rule with a range

- A rule that restricts the range of integers inserted into the column

```
CREATE RULE range_rule  
AS
```

```
@range >= $1000 AND @range < $20000;
```

■ Creating a rule with a list

- Creates a rule that restricts the values entered into a column (to which this rule is bound) to only those listed in the rule.

```
CREATE RULE list_rule  
AS
```

```
@list IN ('1389', '0736', '0877');
```



CREATE RULE - Examples

- Creating a rule with a pattern

- Creates a rule to follow a pattern of any two characters followed by a hyphen (-), any number of characters or no characters, and ending with an integer from 0 through 9.

```
CREATE RULE pattern_rule  
AS  
@value LIKE '__-%[0-9]'
```



sp_bindrule

- Binds a rule to a column or to an alias data type

`sp_bindrule 'rule', 'object_name' , ['futureonly_flag' ]`

- Binding a rule to a column

`EXEC sp_bindrule 'ruleXX', 'Employee.ColumnXY'`

- Binding a rule to an alias data type

`EXEC sp_bindrule 'rule_ssn', 'ssn'`

- Using the futureonly_flag

- ☐ When futureonly is specified, no existing columns of type ssn are affected

`EXEC sp_bindrule rule_ssn, 'ssn', 'futureonly';`



sp_droptype

- Deletes an alias data type from systypes

`sp_droptype 'typeName'`

- Example

- ☐ The following example drops the alias data type studCode.

`EXEC sp_droptype 'studCode'`



User-defined Table Type

- A user-defined table type can be used
 - as a table-valued parameters
 - In procedures and functions
 - as table variables
- A user-defined table type is not an object
 - You can't add rows to this table
 - It is a TYPE
- You can use the user-defined table type to create table-valued parameters in stored a procedure or function
 - Table-valued parameters can only be INPUT parameters and must be accompanied by the READONLY keyword



User-defined Table Type - Example

```
CREATE TYPE dbo.MyName AS TABLE
(fName varchar(20), LName varchar(20) );
GO
```

```
CREATE PROCEDURE dbo.GetFullNames
    @nm dbo.MyName READONLY
AS
BEGIN
    SELECT fName + ' ' + LName FROM @nm;
END
GO
```

```
DECLARE          @mm dbo.MyName
INSERT           @mm(fName, LName)
                VALUES('Genet ', 'Tesfaye')

INSERT           @mm(fName, LName)
                VALUES('Hailu ', 'Getachew')

EXEC            dbo.GetFullNames @mm
```



Exercise

1. Create a user defined data type for the **studCode** to be used at HiLCoE
 - ☐ Format: AX2359
2. Create a user defined data type to be used for the **status** column in the employees table
 - ☐ Set the default to be “ACTIVE”
 - ☐ Enforce a rule to ensure that the acceptable values are one of the following:
 - Active
 - Not Active
 - On Leave