Improvements in SQL Server 2008 T-SQL command (Part 2)

In the previous section we explored some of the improvements in SQL Server 2008 T-SQL statements, including IntelliSense, Grouping Set, and FORCESEEK hint tables.

In the previous section, we learned some improvements in SQL Server 2008 T-SQL statements, including IntelliSense, Grouping Set, FORCESEEK hint tables, etc. In this section we will continue to explore New data types introduced in SQL Server 2008.

>> Improvements in SQL Server 2008 T-SQL command (Part 1)

SQL Server 2008 introduced a number of new data types that help extend the audience and improve the performance of SQL Server. For example, the User-Defined Table type and Table-Valued Parameter allow users to put a result group into a procedure and store multiple return values ??on the server, while the data type Date and Time can help save memory resources because of less memory usage in case users only need to save the day or time, and make operation easier when working with them. In this section we will focus on understanding these two data types, in terms of scope of use and applicability.

User-Defined Table Type (UDTT) and Table-Value Parameter (TVP)

With SQL Server 2008 we can create a UDTT (user-defined table style) according to the table structure definition. To ensure that all data in the UDTT meets all criteria, we can create separate constraints and primary keys on this type of table. In addition, we can use a UDTT to declare multiple TVPs (parameters by value table) for storage tools or procedures to send multiple data records to a stored procedure or a public tool without creating a temporary table or multiple parameters.

TVP is now much more flexible, and in some cases it is even more capable of executing temporary tables or providing many different methods to avoid using parameters. Using TVP has the following benefits, do not use the key for the original data form from the workstation, no need to re-edit the command, reduce access to the server, allow the client to specify the sort order and the Main courses, …

When table variables are manipulated as parameters, this table will be actualized in the tempdb system database rather than transmitting the entire data set in memory, which helps to deal with the amount Big data more efficient. Every server that performs the transformation of the table's parameters is manipulated by the reference, using this reference as a pointer to that table in tempdb to avoid making copies for the input data.

Application

Programmers are always having trouble transferring multiple records in the database to make the most of their performance. For example, when a programmer needs to program a page that accepts a multi-item order, they will have to write in their own logic to group all the insertion instructions into an XML or gender string

(OPENXML in SQL Server 2000) and then move those text values ??to a procedure or command. This operation requires that the procedure or command must have the logic necessary to remove the group of values ??and make the data structures valid, then proceed to insert the records. Too many operations to perform but not optimal, in this case we can use TVP to transfer a data table consisting of multiple records from .NET application to SQL Server and insert directly into the table without Any additional operations must be performed on this server.

Limit

1. The UDTT exists some limitations, for example it cannot be used as a column in the table, the table format cannot be changed after it has been created, a default value cannot be specified in the format of UDTP, .

2. SQL Server does not maintain the column statistics of TVP.

3. TVP must be transferred as input READONLY parameters to T-SQL commands. We cannot perform DML tasks such as UPDATE, DELETE, or INSERT (insert) on a TVP in the main part of the command. If we need to change the data to be transferred to a stored procedure or parameter expression command in the TVP, we must insert that data into a temporary table to fly a variable of the table. In addition, we cannot use table variables as OUTPUT parameters, which can only be used as INPUT (input) parameters.

For example

In this example we will learn how to create a user-defined table type, create a variable of this type of table, insert records into the table and pass it to the stored procedure as a TVP parameter.

First we will create a table and insert the tables there. The command structure creates this table as follows:

```
--Create a table to store customer information
CREATE TABLE [Customers]
(
[ID] [int] NOT NULL PRIMARYKEY IDENTITY,
[FirstName] [varchar] (100) NOT NULL,
[LastName] [varchar] (100) NOT NULL,
[Email] [varchar] (200) NOTNULL
)
GO
- Insert the record into the Customer table
INSERT INTO [Customers] (FirstName, LastName, Email)
VALUES ('AAA', 'XYZ', 'aaa@test.com')
INSERT INTO [Customers] (FirstName, LastName, Email)
VALUES ('BBB', 'XYZ', 'bbb@test.com')
INSERT INTO [Customers] (FirstName, LastName, Email)
VALUES ('CCC', 'XYZ', 'bbb@test.com')
GO
```

	ID	FirstName	LastName	Email
1	1	AAA	XYZ	aaa@test.com
2	2	BBB	XYZ	bbb@test.com
3	3	CCC	XYZ	bbb@test.com

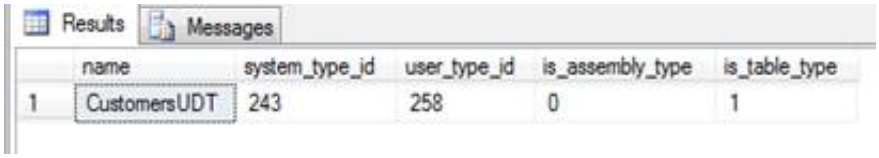
Next we will create a UDTT when successfully created we can view the table details using two **System Catalog View** :

- Create a UDTT that stores customer records

```
CREATE TYPE [CustomersUDT] AS TABLE
(
[FirstName] [varchar] (100) NOT NULL,
[LastName] [varchar] (100) NOT NULL,
[Email] [varchar] (200) NOTNULL
)
GO
```

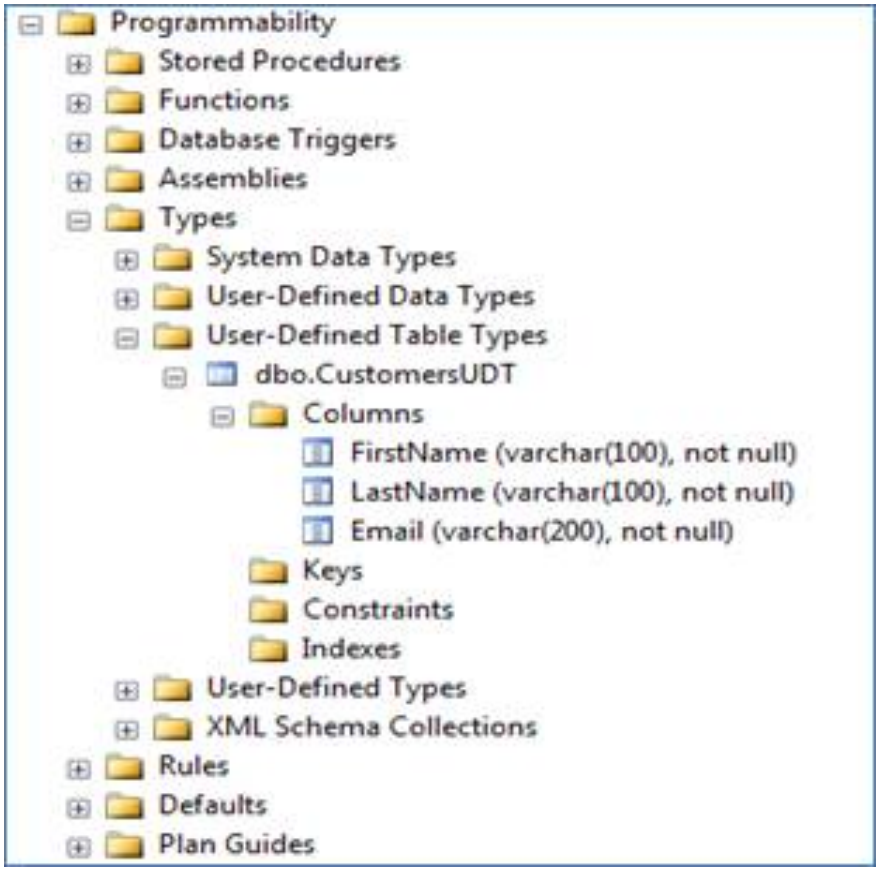
- We can use the Catalog View to view the created table

```
SELECT name, system_type_id, user_type_id, is_assembly_type, is_table_type FROM
SYS.TYPES WHERE is_table_type = 1
SELECT name, system_type_id, user_type_id, is_assembly_type, is_table_type FROM
SYS.TABLE_TYPES
GO
```



	name	system_type_id	user_type_id	is_assembly_type	is_table_type
1	CustomersUDT	243	258	0	1

We can also use **SQL Server Management Studio** (SSMS) to see user-defined tables created in a database, access the **User-Defined Table Types** in the node **Types** window of the **Object Explorer** panel.



Then create a stored procedure to convert the variable of the UDTT as a TVP. Remember the scope rule for a variable to be applied in this case as well as the UDTT variable because this variable will automatically exceed the scope when the table is created.

Results | Messages

	ID	FirstName	LastName	Email
1	1	AAA	XYZ	aaa@test.com
2	2	BBB	XYZ	bbb@test.com
3	3	CCC	XYZ	bbb@test.com
4	4	DDD	XYZ	ddd@test.com
5	5	EEE	XYZ	eee@test.com
6	6	FFF	XYZ	fff@test.com

Transfer TVP with .NET application

First we need to install the .NET Framework 3.5 software, which provides a new type of SQL database called Structure inside the *System.Data.SQLClient namespace* .

Make sure that the *DataTable* we create in the application. *NET* matches the schema of the UDTT, in other words, the name of the column, the number of columns and the data types must be the same. Although in some cases, if the data type is not the same, if it is still compatible, it is still allowed to operate.

```
// Create a local data table to store customer records
DataTable dtCustomers = new DataTable ("Customers");
DataColumn dcFirstName = new DataColumn ("FirstName", typeof (string));
DataColumn dcLastName = new DataColumn ("LastName", typeof (string));
DataColumn dcEmail = new DataColumn ("Email", typeof (string));
dtCustomers.Columns.Add (dcFirstName);
dtCustomers.Columns.Add (dcLastName);
dtCustomers.Columns.Add (dcEmail);
// Insert customer 1
DataRow drCustomer = dtCustomers.NewRow ();
drCustomer ["FirstName"] = "AAA";
drCustomer ["LastName"] = "XYZ";
drCustomer ["Email"] = "aaa@test.com";
dtCustomers.Rows.Add (drCustomer);
// Insert customer 2
drCustomer = dtCustomers.NewRow ();
drCustomer ["FirstName"] = "BBB";
drCustomer ["LastName"] = "XYZ";
drCustomer ["Email"] = "bbb@test.com";
dtCustomers.Rows.Add (drCustomer);
// Insert customer 3
drCustomer = dtCustomers.NewRow ();
drCustomer ["FirstName"] = "CCC";
drCustomer ["LastName"] = "XYZ";
drCustomer ["Email"] = "ccc@test.com";
dtCustomers.Rows.Add (drCustomer);
// Create the Connection object to connect to the server / database
SqlConnection conn = new SqlConnection ("Data Source = ARALI-LAPTOP; Initial Catalog
= tempdb; Integrated Security = true");
conn.Open ();
// Create a Command object called the stored procedure
SqlCommand cmdCustomer = new SqlCommand ("AddCustomers", conn);
cmdCustomer.CommandType = CommandType.StoredProcedure;
// Create a parameter using SQL DB type viz. Structured new to convert as parameter table values
SqlParameter paramCustomer = cmdCustomer.Parameters.Add ("@ CustomersTVP",
SqlDbType.Structured);
paramCustomer.Value = dtCustomers;
// Run query
cmdCustomer.ExecuteNonQuery ();
```

New Date and Time data type

Real application case

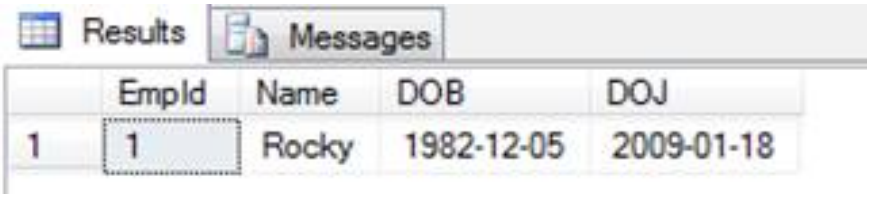
No one can deny the importance of a data type that can only store dates without time or time without dates. For example, to store the birth date of an employee we will only need to store the date, while time is not needed. Similarly, to store information from time to time during the day such as from 01:00 to 08:00 (stage A), from 8: 1 to 16:00 (stage B), and from 16:01 to 24:00 (stage C), we only need to store the time of the unrelated date in this case. Until SQL Server 2005, we do not have a separate storage option; instead, we have to choose either a data type or DATETIME or SMALLDATETIME. Saving both days and hours not only causes trouble at work but also wastes storage space. For example, to store birthdays of 100 million customers we will have to use about 770MB if using the DATETIME data type. However, in SQL Server 2008 this problem has been fixed, now we can store the same information with much less capacity (about 290MB in the above example, only storing the date) by pointing store date or time. Not only that, these new data types will have a wider area, the time is broken down to nanoseconds (one billionth of a second), and allows storing time zone blanks with that data.

SQL Server 2008 introduces four new DATETIME data types including:

DATE

In previous versions of SQL Server, we must use the *DATETIME* or *SMALLDATETIME* data types even if we only need to store *Date* . These data types store time as part of it. We then need to format the output data to show only the date component. SQL Server 2008 introduces *DATE* data type that is very useful for storing dates. It supports Gregorian calendar and uses 3 bytes to store the date. Area of *DATE* data type from *01-01-0001* to *12-31-9999* , while *DATETIME* data *type* has area from *01-01-1753* to *31-12-9999* and *SMALLDATETIME* from *01-01-1900* to *06- 06-2079* .

```
CREATE TABLE Employee
(
EmpId INT IDENTITY,
Name VARCHAR (100),
DATE DOB, - Date c?a phân vùng sinh s? l?u ch? ch? ngày và không có ph?n th?i gian
DOJ DATE DEFAULT GETDATE () - Date of joining will be default, again ch? ngày, không
th?i gian
)
GO
- Create a DATE variable and assign a date value
DECLARE @DOB DATE = CONVERT (DATE, '12 / 05/1982 ')
INSERT INTO Employee (Name, DOB)
VALUES ('Rocky', @DOB)
GO
SELECT * FROM Employee
GO
```



TIME

Similar to the Date data type, in SQL Server 2008, a data type *TIME* is used in case only the time is stored. *TIME* data area between *00:00: 00.0000000* and *23:59: 59.9999999* . The accuracy of data type *TIME* up to 100 nanoseconds, however, we can reduce this value to save memory (about 3 to 5 bytes). This data type does not recognize the time zone, and it uses a 24-hour system.

```
CREATE TABLE ShiftMaster
(
ShiftName VARCHAR (100),
StartTime TIME, - StartTime without storing date component
EndTime TIME - EndTime without storing component date
)
GO
--Delete variable TIME to store time
DECLARE @StartTime TIME = '00: 01 '
DECLARE @EndTime TIME = '08: 00 '
INSERT INTO ShiftMaster (ShiftName, StartTime, EndTime)
VALUES
('A Shift', @StartTime, @EndTime),
('Shift B', '08:01', '04: 00 '),
('Shift C', '04:01', '00: 00 ')
GO
SELECT * FROM ShiftMaster
GO
```

	ShiftName	StartTime	EndTime
1	Shift A	00:01:00.0000000	08:00:00.0000000
2	Shift B	08:01:00.0000000	04:00:00.0000000
3	Shift C	04:01:00.0000000	00:00:00.0000000

DATETIMEOFFSET

This is another new data type of SQL Server 2008. Thanks to this type of data, we can store dates / times with very high accuracy. Although we cannot store time zones like Eastern Time, Central Time, etc., but we can store offset -5: 00 for Eastern Standard Time and -6: 00 for Central Standard Time, . *Date* data range in approx. from *01-01-0001* to *12-31-9999* , *Time* range between *00:00:00* and *23:59: 59.9999999* . *Offset* region ranges from *-14: 00* to *+14: 00* . The accuracy of data can be set manually according to Gregorian schedule. *DATETIMEOFFSET* defines a date that is associated with a period of another day that recognizes the time zone and is based on a 24-hour system.

```
CREATE TABLE OrderMaster
(
```

```
OrderID INT IDENTITY,
CustomerID INT,
OrderDateTIme DATETIMEOFFSET - DATETIMEOFFSET data type to date store and
time offset with time-zone
)
GO
--Register DATETIMEOFFSET variable to store date and time together with Offset time zone
DECLARE @OrderDateTIme DATETIMEOFFSET = '2009-01-16 00:24 +05: 30'
INSERT INTO OrderMaster (CustomerID, OrderDateTIme)
VALUES
(1, @OrderDateTIme),
(1, '2009-01-18 00:24 +05: 30'),
(2, '2009-01-18 00:24 -04: 00')
GO
SELECT * FROM OrderMaster
GO
```

	OrderID	CustomerID	OrderDateTIme
1	1	1	2009-01-16 00:24:00.0000000 +05:30
2	2	1	2009-01-18 00:24:00.0000000 +05:30
3	3	2	2009-01-18 00:24:00.0000000 -04:00

DATETIME2

The reason SQL Server 2008 introduces the *DATETIME2* data *type* is because the *DATETIME* data *type is* not compatible with the SQL standard, and DATETIME is not fully compatible with the .NET *DATETIME* data *type* . *DATETIME2* is basically a combination of the new *DATE* and *TIME* data types and uses 6 to 8 bytes. The actual size is determined by the accuracy of the data stored, for example, DATETIME (0) will use 6 bytes, DATETIME (3) will use 7 bytes while DATETIME (7) will default. Use 8 bytes (the number inside the brackets indicates the accuracy of the information, and you can change this value between 0 and 7).

DATETIME2 accepts many string formats. This data type also uses the Gregorian calendar, and we cannot specify the time zone in this data type.

In addition to changes in data types, SQL Server 2008 introduces five new functions, including *SYSDATETIME* , *SYSDATETIMEOFFSET* , *SYSUTCDATETIME* , *SWITCHOFFSET* and *TODATETIMEOFFSET* .

The **SYSDATETIME** function returns the current system time without the time zone with an accuracy of 10 milliseconds.

The **SYSDATETIMEOFFSET** function is the same as the **SYSDATETIME** function, but this function includes the time zone.

SYSUTCDATETIME returns the Universal Coordinated Time date and time (similar to Greenwich Mean Time) with an accuracy of 10 milliseconds. This time is taken from the current local time and the time zone setting of the server where SQL Server is running.

Both **SYSDATETIME** and **SYSUTCDATETIME** functions **return the DATETIME2** data *type* , while the **SYSDATETIMEOFFSET** function returns the **DATETIMEOFFSET** data **type** .

SWITCHOFFSET function returns a **DATETIMEOFFSET** value that is changed from the Offset storage time zone to a specific new Offset time zone. The **TODATETIMEOFFSET** function converts a local or time-based date and a time-off time zone to a **DATETIMEOFFET** value.

```
SELECT
GETDATE () AS [GETDATE],

- Returns the current time of the system without the time zone with an accuracy of 10 milliseconds.
SYSDATETIME () AS [SYSDATETIME],

- Returns Coordinated Time and Date and Time with an accuracy of 10 milliseconds.
SYSUTCDATETIME () AS [SYSUTCDATETIME]
```

	GETDATE	SYSDATETIME	SYSUTCDATETIME
1	2009-01-18 01:51:39.330	2009-01-18 01:51:39.3340000	2009-01-17 20:21:39.3340000

```
SELECT

- Returns the current time of the system with the time zone with an accuracy of 10 milliseconds
SYSDATETIMEOFFSET () AS [SYSDATETIMEOFFSET],

- Returns the DATETIMEOFFSET value changed from the stored time zone Offset to a stored time zone of the newly assigned Offset and maintains the original value.
SWITCHOFFSET (SYSDATETIMEOFFSET (), '+05: 00') AS [SWITCHOFFSET],

- Convert the local date or time value and an Offset time zone specified to a DATETIMEOFFSET value
TODATETIMEOFFSET (GETDATE (), '+ 05:30') AS [TODATETIMEOFFSET]
```

	SYSDATETIMEOFFSET	SWITCHOFFSET	TODATETIMEOFFSET
1	2009-01-18 01:50:55.6320000 +05:30	2009-01-18 01:20:55.6320000 +05:00	2009-01-18 01:50:55.630 +05:30

Conclude

UDTT and TVP allow users to transfer a result group to a procedure and save many return values ??to the server to use in earlier versions of SQL Server.

New Date and Time data types in SQL Server 2008 save memory because it takes less time to save the date or time (more accurately), making it easier to work with these data, and allows working with DATETIMEOFFSET.

In the next section we will look at the new HIERARCHYID data type which allows to store hierarchies in the database.