Migrating from Sybase to SQL Server

Table of Contents:

Migrating from Sybase to SQL Server

Projects involving database migration are common. In this article, we share experiences in migrating from Sybase to SQL Server.

Although some differences will be quite noticeable, such as a Sybase stored procedure that refuses to compile in SQL Server, other differences are much more subtle. It will be necessary to heavily test the behavior and result sets of all programming logic in script files and stored procedures prior to completing the conversion.

In the following section we provide a set of core differences between the two databases that must be explored during the critical planning stage.

Migrating from Sybase to SQL Server - Data Compatibility Mode Behavior

A temporary solution to some of the compatibility differences between SQL Server 2000 and Sybase is to change the databases compatibility level in SQL Server to match that of Sybase. To make this change, use the sp_dbcmptlevel stored procedure.

The following statements and results are listed in this table to show the difference in the versions:

Statement	Sybase	SQL Server
Select au_fname Fromauthors Group BY au_fname	All distinctau_fname in Ascending order.	All distinct au_fname inInserted order.
UPDATE authors SETa.au_fname = "John"	Au_fname in all rows updated to"John"	Error
CREATE TABLE malls(ID int, test bit)	Test column(bit) defaults to not NULL.	Test column(bit) determined by session or database settings.
ALTER TABLE authorsALTER COLUMN au_lname Varchar(60) NULL	Invalid syntax.	Changes the au_lname column from varchar(40) to varchar(60).
CREATE TRIGGERtu_authors on authors for update as print 'Update tu_authors' go	tu_authors2overwrites tu_authors	Both tu_authors and tu_authors2 are created and fire when the authors table is updated.

CREATE TRIGGERtu_authors2 on authors for update as print 'Update tu_authors2' go		
CREATE PROC test_procAs SELECT au_lname From #archive	Sybase-error if#archive does not exist	No warning.
SELECT DATALENGTH(' ')	Returns 1.	Returns 0.
SELECT DATALENGTH(N' ')	Returns 1.	Returns 0.
SELECT LTRIM(' ')	Returns NULL.	Returns an empty string.
SELECT LTRIM(N' ')	Returns NULL.	Returns an empty string.
select REPLICATE ('123' ,0)	Returns NULL.	Returns an empty string.
select REPLICATE (N' 123' ,0)	Returns NULL.	Returns an empty string.
select RIGHT(N '123' ,0)	Returns NULL.	Returns an empty string.
select RIGHT('123' ,0)	Returns NULL.	Returns an empty string.
select RIGHT('123' , -1)	Returns NULL.	Error.
select RIGHT(N '123' , -1)	Returns NULL.	Error.
select RTRIM(' ')	Returns NULL.	Returns an empty string.
select RTRIM(N' ')	Returns NULL.	Returns an empty string.
Select space(0)	Returns NULL.	Returns an empty string.
select SUBSTRING('123',1,0)	Returns NULL.	Returns an empty string.
select SUBSTRING(N'123',1,0)	Returns NULL.	Returns an empty string.
select CHARINDEX ('SQLServer' ,NULL)	Return 0.	Return NULL.
INSERT x SELECT 1INTO y FROM authors	Error.	Error.

Notes:

1. When the compatibility mode is set to 70, the following words cannot be used for object names and identifiers: BACKUP, DENY, PRECENT, RESTORE, and TOP.
2. When the compatibility mode is set to 65, the following words cannot be used for object names and identifiers: AUTHORIZATION, CASCASE, CROSS, DISTRIBUTED, ESCAPE, FULL, INNER, JOIN, LEFT, OUTER, PRIVILEGES, RESTRICT, RIGHT, SCHEMA, and WORK.

Here is the syntax for sp_dbcmptlevel:

sp_dbcmptlevel [[@dbname=] name][,[@new_cmptlevel=]version]

@dbname is the name of the database for checking or changing the compatibility

level. @new_cmptlevel determines which compatibility level the database is set at (set it to 70, 65, or 60 with a default of NULL)

Example:

sp_dbcmptlevel pubs

This code returns the following:

The current compatibility level is 70.

Now take a look at another example

sp_dbcmptlevel pubs, 65

It returns this:
DBCC execution completed. If DBCC printed error messages, contact your system administrator. At this point, you can rerunsp_dbcmptlevel to verify that the pubs database was changed correctly:

sp_dbcmptlevel pubs

It returns the following:

The current compatibility level is 65

In addition to the examples in the previous table, the differences in the compatibility levels extend to reserved words too. Both Sybase and SQL Server have lists of reserved words that cannot be used to name objects in the database. The lists for the two products are similar, but not exactly the same.

This issue could make the conversion from Sybase to SQL Server a little more difficult because objects that can be created in Sybase might not be able to be created in SQL Server. The following is a list of SQL Server reserved words that are not reserved words in Sybase:

Note: Any objects in your Sybase database with names in this list must be renamed before an SQL Server conversion.

BACKUP	COLUMN	COMMITTED	CONTAINS	CONTAINSTABLE
CROSS	CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_USER
DENY	DISTRIBUTED	FILE	FLOPPY	FREETEXT
FREETEXTTABLE	FULL	IDENTITYCOL	INNER	JOIN

Sybase	SQL Server
Set chained [on : off]	Set implicit_transactions [on : off]

To determine the current transaction mode using Sybase:

```
SELECT @@tranchained
GO
```

Here are the results:
0 indicates that the default unchained mode is being used.
1 indicates that the connection is running in chained mode

To determine the current transaction mode using SQL Server:

```
IF (@@options & 2)  >  0
PRINT on
ELSE
PRINT off
```

Here are the results:
0 off
>0 on

Isolation levels

In a multithreaded application, such as a relational database, it is important for the database engine to manage how data is isolated between running processes. The syntax is different for Sybase and SQL Server when referring to the isolation levels with the SET statement.

Here in the next table I explain the isolation levels differences between Sybase and SQL Server:

Sybase	SQL Server
0	READ UNCOMMITTED
1	READ COMMITTED
2	REPEATABLE READ
3	SERIALIZABLE

Cursor syntax

Creating and executing a stored procedure in both products remains similar, but a few exceptions in the cursor statements should be noted when converting.

The following is an example:

```
CREATE PROCEDURE sql_cursor AS
DECLARE  @lname char(20), @fname char(20)
DECLARE mycursor CURSOR FOR
SELECT au_lname, au_fname FROM authors
OPEN mycursor
FETCH FROM mycursor INTO @lname, @fname
```

```
WHILE @@ FETCH_STATUS = 0
/*  Sybase uses  @SQLSTATUS instead of @@ FETCH_STATUS */
BEGIN
FETCH FROM  mycursor INTO @lname, @fname
/*
** SOME BUSINESS LOGIC GOES HERE
*/
END
CLOSE mycursor
DEALLOCATE /* Sybase needs the word CURSOR right here * / mycursor
```

	Sybase	SQL Server
Fetch was successful	0	0
Fetch statement failed	1	-2
No more rows available	2	-1

Rollback Triggers

This command does not exist in SQL Server, so all Sybase stored procedures that incorporate the ROLLBACK TRIGGER command must be converted before a successful migration to SQL Server. By using the ROLLBACK TRIGGER command, it could be misleading when modifying data in tables with triggers. A single ROLLBACK TRIGGER rolls back only the trigger and the modification that fired the trigger. The remainder of the transaction continues and, if committed, is written to the database without the single command that was rolled back. Therefore, all statements in the transaction might not have completed successfully, but the data was committed anyway.

Here I give a sample trigger using ROLLBACK TRIGGER in Sybase:

```
CREATE TABLE table1 (a int, b int)
GO
CREATE TRIGGER trigger1 on table1 FOR  INSERT
AS
IF EXISTS  (SELECT  1 FROM inserted WHERE a = 100)
BEGIN
ROLLBACK TRIGGER  with RAISERROR  50000 Invalid value for column a
END
INSERT INTO table2
SELECT a, GETDATE() from inserted
RETURN
GO
```

In this code, all inserts into table1 also insert as audit rows into table2 unless a = 100. If a = 100, the ROLLBACK TRIGGER command is fired and the INSERT is not fired. The rest of the batch continues, and a raiserror occurs, stating that there was an error in one of the INSERT commands.

The INSERT commands are shown here:

```
BEGIN TRAN
INSERT INTO table1 VALUES (1, 1)
INSERT INTO table1 VALUES (100,2)
INSERT INTO table1 VALUES (3, 3)
GO
SELECT  *  FROM table1
```

After issuing these commands, table1 and table2 each have two rows. Table1 has the values 1,1 and 3,3 and the second INSERT isn't committed because of the ROLLBACK TRIGGER. Table2 has the values 1, (currentdate) and 3, (currentdate) and the 100 isn't inserted because all processing in the trigger halts when a = 100 and the ROLLBACK TRIGGER is fired.
Mimicking this behavior in SQL Server requires some additional code. The outer transaction must now be accompanied with savepoints, as shown here:

```
CREATE trigger1 on table1 FOR INSERT
AS
SAVE TRAN trigger1
IF EXISTS (SELECT * FROM inserted WHERE a = 100)
BEGIN
ROLLBACK TRAN trigger1
RAISERROR  50000 ROLLBACK
END
INSERT INTO table2
SELECT a,  GETDATE() FROM inserted
GO
```

This trigger now begins with a savepoint and ROLLBACK TRANSACTION rolls back only the trigger logic, not the entire transaction (which is similar to Sybases ROLLBACK TRIGGER statement). The changes to the batch job are shown here:

```
BEGIN TRAN
SAVE TRAN save1
INSERT INTO table1 VALUES (1, 1)
IF @@error = 50000
ROLLBACK TRAN save1
SAVE TRAN save2
INSERT INTO table1 VALUES (100, 1)
IF @@error = 50000
ROLLBACK TRAN save2
SAVE TRAN save3
INSERT INTO table1 VALUES (3, 3)
IF @@error = 50000
ROLLBACK TRAN save3
COMMIT TRAN
```

As you can see, the changes are not trivial. Because the ROLLBACK TRIGGER command can allow any single batch statement to fail, the additional logic must be included in the migrated SQL Server stored procedure code. Depending on the use of ROLLBACK TRIGGER, this could be a big, but necessary, job. There are no shortcuts here. The behavior of the trigger changes if all the ROLLBACK TRIGGER statements are changed to ROLLBACK TRANSACTION after converting, so be careful.

Migrating from Sybase to SQL Server - Optimizer Hints

SQL Server allows for optimizer hints on SELECT, INSERT, UPDATE, and DELETE statements. Sybase allows optimizer hints only on SELECT statements. Here's the SQL Server vs. Sybase GUI approach:

Sybase	SQL Server
Use a text-based query analysis tool called SHOWPLAN	Use Query Analyzer
Command to enable SHOWPLAN from within ISQL **SET SHOWPLAN ONGO**	Command to enable SHOWPLAN_ALL or SHOWPLAN_TEXT from query analyzer **SET SHOWPLAN_ALLGO**

Temporary Table Names

Type of table name	Maximum length
SQL Server table name	128
SQL Server temporary table name	116
Sybase table name	30
Sybase temporary table name	13

Data Types

Data types	Sybase	SQL Server
char(n)	255	8000
varchar(n)	255	8000
nchar(n)	255	4000
Nvarchar(n)	255	4000
Binary	255	8000
Varbinary	255	8000

Notes:
1) The bit data type in SQL Server can be set to 0, 1, or NULL.
2) The bit data type in Sybase does not allow nulls.

Identity Columns

Sybase	SQL Server
Numeric(x,0)	Tinyint,smallint,int,decimal(x,0) or numeric(x,0)

Print Syntax

All PRINT statements that use substitution syntax must be changed to RAISERROR statements during the conversion process.

Migrating from Sybase to SQL Server - Conclusion

Converting a Sybase database to an SQL Server database is a feat that can be accomplished, but there are a number of differences between the two products that must be addressed. Depending on the size of your application, this conversion could turn into a lengthy process. You don't have to rewrite the whole application, but there are a fair amount of show-stoppers.

However, the positive side is: there cannot be an easier conversion between two major database packages than this one, and a successful conversion is highly probable because of the products many similarities. Happy migrating!