# logo-sql08.gif

Guide to Migrating from Sybase ASE to SQL Server 2008

SQL Server Technical Article

**Writers:** Yuri Rusakov (DB Best Technologies), Alexander Pavlov (DB Best Technologies), Alexey Kovaliov (DB Best Technologies)

**Technical Reviewer:** Dmitry Balin (DB Best Technologies)

**Published:** August 2009

**Applies to:** SQL Server 2008 and SQL Server 2008 R2

**Summary:** This white paper covers known issues for migrating Sybase Adaptive Server Enterprise database to SQL Server 2008. Instructions for handling the differences between the two platforms are included. The paper describes how SQL Server Migration Assistant, the best tool for this type of migration, can help resolve various migration issues.

Created by: DB Best Technologies LLC

P.O. Box 7461, Bellevue, WA 98008

Tel.: (408) 202-4567

E-mail: info@dbbest.com

Web: [www.dbbest.com](file:///D%3A%5CWP-JOEL%5COUT%5Cwww.dbbest.com)

# Copyright

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft and SQL Server are registered trademarks of Microsoft Corporation in the United States and other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Contents

[Copyright 2](#_Toc237706157)

[Introduction 8](#_Toc237706158)

[Migration Plan 9](#_Toc237706159)

[SSMA Functionality 10](#_Toc237706160)

[Sybase System Function Migration 12](#_Toc237706161)

[Equivalent Functions 12](#_Toc237706162)

[Nonsupported Functions 12](#_Toc237706163)

[Emulated Functions 14](#_Toc237706164)

[CHAR\_LENGTH 14](#_Toc237706165)

[CURRENT\_DATE 14](#_Toc237706166)

[DATELENGTH 14](#_Toc237706167)

[INTTOHEX 15](#_Toc237706168)

[HEXTOINT 15](#_Toc237706169)

[NEWID 15](#_Toc237706170)

[NEXT\_IDENTITY 16](#_Toc237706171)

[STR\_REPLACE 17](#_Toc237706172)

[CHARINDEX 18](#_Toc237706173)

[REPLICATE 19](#_Toc237706174)

[RTRIM 20](#_Toc237706175)

[LEN 21](#_Toc237706176)

[SUBSTRING 21](#_Toc237706177)

[TO\_UNICHAR 21](#_Toc237706178)

[CONVERT 22](#_Toc237706179)

[PAGESIZE 22](#_Toc237706180)

[UHIGHSURR 22](#_Toc237706181)

[ULOWSURR 23](#_Toc237706182)

[USCALAR 23](#_Toc237706183)

[PATINDEX 23](#_Toc237706184)

[DATEDIFF 24](#_Toc237706185)

[INDEX\_COLORDER 24](#_Toc237706186)

[COL\_NAME 25](#_Toc237706187)

[OBJECT\_NAME 26](#_Toc237706188)

[DATEPART 26](#_Toc237706189)

[INDEX\_COL 27](#_Toc237706190)

[CURRENT\_TIME 28](#_Toc237706191)

[Data Migration Architecture of SSMA for Sybase 29](#_Toc237706192)

[General 29](#_Toc237706193)

[Solution Layers 29](#_Toc237706194)

[Client Application 29](#_Toc237706195)

[Stored Procedures Interface 30](#_Toc237706196)

[Database Layer 30](#_Toc237706197)

[Migration Executable 30](#_Toc237706198)

[Message Handling 31](#_Toc237706199)

[Validation of the Results 31](#_Toc237706200)

[Migrating Applications from CT-Library to ODBC 31](#_Toc237706201)

[Module Mapping 31](#_Toc237706202)

[Command Mapping 32](#_Toc237706203)

[Type and Structure Mapping 40](#_Toc237706204)

[Migration Example 45](#_Toc237706205)

[Source Programming Using CT-Library 45](#_Toc237706206)

[ODBC Equivalent 51](#_Toc237706207)

[Sybase Migration Issues 56](#_Toc237706208)

[Data Types 56](#_Toc237706209)

[Length of String Types 56](#_Toc237706210)

[Expressions 57](#_Toc237706211)

[Conversion of Time to String 57](#_Toc237706212)

[Unicode Literals 60](#_Toc237706213)

[Double-Quoted String Literals 60](#_Toc237706214)

[Empty String 61](#_Toc237706215)

[‘||’ as Concatenation Operator 62](#_Toc237706216)

[Concatenation with NULL 62](#_Toc237706217)

[Comparisons with NULL 63](#_Toc237706218)

[CASE Nesting Limit 63](#_Toc237706219)

[Implicit Conversion of Binary Types 64](#_Toc237706220)

[Implicit Conversion of Datetime as Function Argument 65](#_Toc237706221)

[SQL Commands 67](#_Toc237706222)

[COMMIT 67](#_Toc237706223)

[Different COMMIT Syntax 67](#_Toc237706224)

[Quoted Data Type at CONVERT and CAST Functions 67](#_Toc237706225)

[DEALLOCATE CURSOR 68](#_Toc237706226)

[LOCK TABLE <T> IN SHARE | EXCLUSIVE MODE 68](#_Toc237706227)

[PRINT Command 70](#_Toc237706228)

[RAISERROR 73](#_Toc237706229)

[ROLLBACK 81](#_Toc237706230)

[SELECT / INSERT / DELETE / UPDATE 81](#_Toc237706231)

[AT ISOLATION Clause 81](#_Toc237706232)

[DISTINCT with ORDER BY 82](#_Toc237706233)

[FOR readonly | update Clause 83](#_Toc237706234)

[Different Use of the GROUP BY Clause 83](#_Toc237706235)

[HAVING Clause 87](#_Toc237706236)

[HOLDLOCK Hint 89](#_Toc237706237)

[INDEX <index-name> Hint 90](#_Toc237706238)

[Nonstandard Outer JOIN 91](#_Toc237706239)

[NOHOLDLOCK Hint 95](#_Toc237706240)

[READPAST Keyword 95](#_Toc237706241)

[SHARED Keyword 96](#_Toc237706242)

[Different Behavior of the LIKE Operator in Sybase and SQL Server 2008 96](#_Toc237706243)

[SET ANSINULL 98](#_Toc237706244)

[SET CHAINED 99](#_Toc237706245)

[SET TRANSACTION ISOLATION LEVEL 99](#_Toc237706246)

[UPDATE Aliases 100](#_Toc237706247)

[Different ROLLBACK Syntax 102](#_Toc237706248)

[Sybase Allows Aggregate Functions in UPDATE 102](#_Toc237706249)

[Several Table Hints Are Used at Once 105](#_Toc237706250)

[ORDER BY with Table Name and Column Alias 106](#_Toc237706251)

[CHAR Column Allowing NULLs 107](#_Toc237706252)

[Insertion of Default Values 107](#_Toc237706253)

[Nested Aggregates in a SELECT List 108](#_Toc237706254)

[DELETE Aliases 109](#_Toc237706255)

[Named Constraint on Temporary Table 110](#_Toc237706256)

[Global Variables 112](#_Toc237706257)

[@@ERROR 112](#_Toc237706258)

[@@PAGESIZE 112](#_Toc237706259)

[@@SQLSTATUS 113](#_Toc237706260)

[@@TRANCHAINED 113](#_Toc237706261)

[@@TRANSTATE 114](#_Toc237706262)

[@@UNICHARSIZE 114](#_Toc237706263)

[Data Migration 116](#_Toc237706264)

[Timestamps 116](#_Toc237706265)

[Numeric with Scale > 26 116](#_Toc237706266)

[Constraints and Bound Rules 117](#_Toc237706267)

[Defaults vs. NULLs 117](#_Toc237706268)

[Keeping Identities 118](#_Toc237706269)

[Triggers 118](#_Toc237706270)

[Other Migration Issues 120](#_Toc237706271)

[Cursor Scope 120](#_Toc237706272)

[Case Sensitivity 121](#_Toc237706273)

[Reserved Keywords 121](#_Toc237706274)

[Syb\_identity Pseudocolumn 121](#_Toc237706275)

[Different Syntax of IDENTITY( ) Function 122](#_Toc237706276)

[Login Triggers 122](#_Toc237706277)

[Cross-Database Foreign Key 123](#_Toc237706278)

[Deprecated Equivalents 126](#_Toc237706279)

[Different Scope of Constraint Names 126](#_Toc237706280)

[Dynamic SQL 127](#_Toc237706281)

[Proxy Tables 127](#_Toc237706282)

[Variables in Cursor Declaration 128](#_Toc237706283)

[Different Behavior of the MIN and MAX Functions with Character Columns in Sybase and SQL Server 128](#_Toc237706284)

[Potential Challenges 129](#_Toc237706285)

[Conclusion 130](#_Toc237706286)

[About DB Best Technologies 131](#_Toc237706287)

# Introduction

It's true that Sybase Adaptive Server Enterprise (ASE) and the Microsoft® SQL Server® 2008 database software share common roots. But ever since Sybase and Microsoft began developing their own versions of SQL Server, many differences between the two products have appeared. As a result, some cases of migrating from Sybase ASE to SQL Server 2008 may require certain efforts unless effective migration tools are used.

That's where the Microsoft SQL Server Migration Assistant (SSMA) for Sybase can help. SSMA can migrate tables, views, indexes, triggers, and stored procedures. It frequently converts approximately 99–99.5 percent of source objects and Transact-SQL code. In most cases you need to perform just a few manual patches to produce the final converted result.

This migration guide outlines procedures, problems, and solutions when using SSMA to migrate a Sybase ASE 11.9.2, 12.x or 15.0 database to SQL Server 2008. It has five main sections:

* [**Migration Plan**](#_Migration_Plan). Steps to migrate a Sybase ASE database to SQL Server 2008 by using SSMA 4.0. You will also find hints about manually converting the ASE features that SSMA cannot process automatically.
* [**Sybase System Function Migration**](#_Sybase_System_Function_1). Examines Sybase system function references, grouped by equivalent functions, nonsupported functions, and emulated functions.
* [**Data Migration Architecture of SSMA for Sybase**](#_Data_Migration_Architecture). Explains how SSMA components interact when you migrate data from Sybase to SQL Server 2008.
* [**Migrating Applications from CT-Library to ODBC**](#_Migrating_Applications_from). Outlines the migration of a client application when it calls the Sybase CT-Library to provide database layer access.
* [**Sybase Migration Issues**](#_Sybase_Migration_Issues_1). Explores challenges when migrating from Sybase ASE to SQL Server 2008 and offers possible solutions.

# Migration Plan

Migrating a Sybase ASE database to SQL Server 2008 with SSMA follows a straightforward sequence with the following steps.

**Step 1.** Begin by assessing the source Sybase ASE database by using the SSMA **Create Report** command. The resulting assessment report includes statistics about migration issues and estimates of the working time necessary to manually resolve them. The report also shows the changes SSMA will make during the migration, including a synchronized view of source and target Transact-SQL code showing how SSMA will transform each statement.

**Step 2.** Read the source (ASE) and target (SQL Server 2008) metabases by using the **Connect to Sybase** and **Connect to SQL Server** commands. During this step, SSMA loads the database objects into its workspace, making possible their analysis and processing.

**Step 3.** Decide how to map the source ASE database and schemas to the target SQL Server 2008 database and schemas. By default, SSMA assumes natural schema mapping that preserves the source database and schema names. But SSMA does not create security items by itself; you must manually create all necessary schemas in the target database prior to migration.

**Step 4**. Start the migration by using the **Convert Schema** command. You can find the target objects displayed in the SSMA workspace. To save the results, do one of the following:

* Save the target objects as scripts.
* Load the objects into the SQL Server database.

You can make manual changes in the target SQL window before saving the result code. You can quickly find conversion errors or warnings by looking for special comments that SSMA inserts before the problematic statements.

Note that SSMA does not convert Sybase defaults and rules as separate objects. Instead, they become part of the table definition these defaults and rules are bound to. SSMA replaces Sybase user-defined types by the underlying types, and they also become part of the table definitions.

**Step 5.** After the target tables have been loaded into the target database, you can start data migration.

**Note**Data migration is a remote application on the SQL Server host launched by the SQL Server Agent as a job. That mechanism improves performance by transferring the data directly between two database servers, while not moving information to and from the client workstation where SSMA is executed. For this mechanism to work, Sybase client tools must be installed on both computers, not only on the SSMA workstation. In addition, to use data migration functionality you must install the Microsoft SSMA Extension Pack locally on the target server running SQL Server.

# SSMA Functionality

During Sybase schema conversion, SSMA replaces source objects and Transact-SQL statements, taking into account the differences between the two platforms. The most important changes are:

* PRINT and RAISERROR statements are modified according to SQL Server syntax.
* Nonstandard outer joins are transformed into ANSI format, which is supported by SQL Server.
* Statements with nonstandard usage of the Sybase GROUP BY and HAVING clauses are replaced with SQL Server emulation.
* Double quotes delimiting string literals are changed to single quotes.
* Sybase-specific usage of NULLs and empty string in concatenation and comparison can be optionally emulated.
* SSMA is able to emulate Sybase-style table locking.
* SSMA can change aliases and aggregate functions in the UPDATE statement so that the resulting statement is compatible with SQL Server.
* Incompatible cursor commands, cursor scope, and status are modified to conform to SQL Server standards.

SSMA provides predefined mapping of Sybase data types to SQL Server data types. For example, large objects types (**text** and **image**) are converted to the more advanced SQL Server types **varchar(max)** and **varbinary(max)**. You can customize SSMA type mapping and even specify unique type mapping for each source object independently of others.

SSMA features dozens of useful GUI components, such as the ability to:

* View the entire source-object and target-object trees in all databases and schemas.
* Monitor the progress of any operation in the Output window.
* View the report result for any object in the source tree after the assessment report has been generated.

As for tables, SSMA shows their structures either in column format or as SQL text. You can also view data stored in the tables. In the SQL view, you can see formatted Transact-SQL text both in the source and in the target. The SQL view, moreover, lets you modify the code and save the modifications in the SSMA workspace. In the target SQL view, you can even load the changes to the database, which lets you perform automatic conversions and manual changes simultaneously—without leaving SSMA.

SSMA cannot automatically handle some Sybase features, including:

* Dynamic SQL
* Incompatible system tables and/or procedures
* Proxy tables
* User messages stored in the **sysusermessages** table

Dynamic SQL is a problem because SSMA cannot see the text of a dynamic statement during the conversion. The statement gets its final form only when the generated code is executed. Still, you can use the Statement window to convert ad hoc SQL statements, including dynamic SQL. Try the same approach to convert SQL strings embedded in the user's application code:

1. Pick out the statement from the application (or reconstruct it if the statement is built according to an algorithm).
2. Put the statement in the Statement window.
3. Execute **Convert Schema**.
4. Put the result back into the proper place in the application code.

Note that temporary tables in stored procedures may create problems when their definitions are absent in the module you are converting. Also be alert to duplicated identifiers that can result when you move a case-sensitive Sybase source to a case-insensitive SQL Server. We recommend that the target server collation be case-insensitive.

# Sybase System Function Migration

This section examines what happens to Sybase system function references during migration to SQL Server 2008. Generally speaking, you must pay attention to system function reference migration because:

* Some Sybase system functions cannot be matched to SQL Server system functions.
* Sybase system functions, in many cases, return different results from corresponding SQL Server functions.

This section divides all existing Sybase ASE system functions into three groups:

* **Equivalent functions** that do not require conversion and are usable as is in Transact-SQL code.
* **Nonsupported functions** that cannot be emulated because of physical differences between Sybase ASE and SQL Server 2008.
* **Emulated functions** thatrequire emulation by using SQL Server user-defined functions (UDFs) or that need transformation of their calls to provide full compatibility with the Sybase version.

**Note**   The SSMA Extension Pack creates function emulations. They are implemented as user-defined functions in the **sysdb** database.

## Equivalent Functions

The Sybase system functions are usable as is in SQL Server 2008 code. SSMA currently treats these functions as functional equivalents to their corresponding SQL Server system functions:

Abs, Acos, Ascii\*, Asin, Atan, atn2, avg, ceiling, char, coalesce, col\_length, convert, cos, cot, count, dateadd, datediff, datename, day, db\_id, db\_name, degrees, difference, exp, floor, getdate, host\_name, isnull, left\*\*, ltrim, log, log10, lower, max, min, month, nullif, object\_id, pi, power, radians, rand, reverse\*\*, right\*\*, round, sign, sin, stuff\*\*, soundex, space, square, sqrt, str, sum, suser\_id, suser\_name, tan, textptr, textvalid, upper\*\*, user, user\_id, user\_name, year

\* The results may differ from Sybase if the argument is Unicode data.

\*\* The result may differ from Sybase if the argument contains Unicode surrogate pairs.

**Note**   Be aware that some equivalent functions produce different results when applied to Unicode character data. Such differences matter in some applications, so take this into account.

## Nonsupported Functions

Following is a list of functions that cannot be easy emulated in SQL Server 2008 because of physical organization and security model differences:

curunreservedpgs, data\_pgs, derived\_stat, get\_appcontext\*, host\_id, is\_sec\_service\_on, lct\_admin, license\_enabled, list\_appcontext\*, lockscheme, mut\_excl\_roles, proc\_role, ptn\_data\_pgs, reserved\_pgs, rm\_appcontext\*, role\_contain, role\_id, role\_name, set\_appcontext\*, show\_role, show\_sec\_services, syb\_quit, syb\_sendmsg, tempdb\_id, used\_pgs, valid\_name, valid\_user, rowcnt, tsequal

\* This application context feature can be implemented by using temporary tables, but this is not recommended due to security issues.

## Emulated Functions

This section examines how SSMA 4.0 emulates Sybase functions to produce the same result in SQL Server 2008.

### CHAR\_LENGTH

Sybase syntax:

char\_length(char\_expr | uchar\_expr)

The CHAR\_LENGTH function does not have a full functional equivalent in SQL Server 2008. It is emulated by a UDF, chosen according to the parameter expression type.

If the expression is of the NCHAR or NVARCHAR data type:

 SYSDB.SSMA\_SYB.CHAR\_LENGTH\_NVARCHAR(char\_expr | uchar\_expr)

Otherwise:

 SYSDB.SSMA\_SYB.CHAR\_LENGTH\_VARCHAR(char\_expr | uchar\_expr)

### CURRENT\_DATE

The CURRENT\_DATE function returns the current date. It resembles the SQL Server SYSDATETIME() function but it does not return time information. To emulate it, convert the result of the SYSDATETIME() function to DATE data type:

cast(sysdatetime() as date)

### DATELENGTH

The DATELENGTH function is functionally equivalent to the SQL Server DATELENGTH function, except when it is applied to an empty character string. In most cases, the function call can be migrated as is (the default conversion mode). To provide full compatibility, SSMA includes a DATALENGTH function option (see Figure 1). If it is set to **Replace function**, the call to DATALENGTH is replaced with a call to the **sysdb** function. Otherwise, it is wrapped in a CASE expression as in the following code:

case datalength(expression)

 when 0

 then 1

 else datalength(expression)

end



Figure 1

### INTTOHEX

The INTTOHEX function does not have an equivalent in SQL Server. Use UDF emulation instead:

 sysdb.ssma\_syb.inttohex (<integer\_expression>)

### HEXTOINT

The HEXTOINT function does not have an equivalent in SQL Server. Use UDF emulation instead:

sysdb.ssma\_syb.hextoint (<hexadecimal\_string>)

### NEWID

Sybase syntax:

newid([optionflag])

The NEWID function generates a unique identifier and can be emulated by the SQL NEWID() function with similar functionality. But to provide full compatibility, SSMA converts the function call according to the following rules:

* If *optionflag* is not specified, convert to:

replace(convert(varchar(36), newid()),’-’ , ’ ’)

* If *optionflag* is a constant of **integer** data type:
	+ If it is 0, convert to:

replace(convert(varchar(36), newid()),’-’ ,’’)

* + If it is 1, convert to:

convert(varchar(36), newid())

* If *optionflag* is of **integer** data type and is not a constant, convert to:

CASE optionflag

WHEN 0

THEN replace(convert(varchar(36), newid()),’-’ ,’’))

WHEN 1

THEN convert(varchar(36), newid())

ELSE NULL

END

* If *optionflag* is of **varbinary** data type, convert to:

convert(varbinary, replace(convert(varchar(36), newid()),’-’ ,’’))

### NEXT\_IDENTITY

Sybase syntax:

next\_identity(table\_name)



Figure 2

The NEXT\_IDENTITY function does not have a functional equivalent in SQL Server. It can be emulated by using two functions—IDENT\_CURRENT and IDENT\_INCR—as in the following expression:

ident\_current(table\_name)+ident\_incr(table\_name)

By default, SSMA marks this function with an error because the result can be wrong in multi-user environments. To convert the function as described, check the appropriate conversion option as shown in Figure 2.

### STR\_REPLACE

Sybase syntax:

str\_replace("string\_expression1", "string\_expression2", "string\_expression3")

The STR\_REPLACE function resembles the SQL Server REPLACE function with one exception: Sybase deletes *string\_expression2* from *string\_expression1* if *string\_expression3* is NULL. To emulate this behavior, replace *string\_expression3* with an empty string if *string\_expression 3* is NULL.

REPLACE ( 'string\_expression1' , 'string\_expression2' ,'string\_expression3' )

If *string\_expression3* is a NULL literal, replace it with an empty string (' ').

For full compatibility, you can replace the nonliteral *string\_expression3* with ISNULL(*string\_expression3*, ' ').

### CHARINDEX

Sybase syntax:

charindex(expression1, expression2)

The CHARINDEX function resembles the same SQL Server function except in the treatment of null values. In most cases it can be migrated as is:

charindex(*expression1, expression2*)

To provide full compatibility, SSMA can emulate the function by using a UDF if the appropriate conversion option (Figure 3) is selected. The function choice is based on the parameter’s data type.

If *expression1* is of NVARCHAR or NCHAR data type, convert to:

 SYSDB.SSMA\_SYB.CHARINDEX\_NVARCHAR(expression1, expression2)

Otherwise, convert to:

 SYSDB.SSMA\_SYB.CHARINDEX\_VARCHAR(expression1, expression2)



Figure 3

### REPLICATE

Sybase syntax:

replicate (char\_expr | uchar\_expr, integer\_expr)

In most cases, the REPLICATE function can be migrated as is:

replicate (char\_expr | uchar\_expr, integer\_expr)

To provide full compatibility, SSMA can emulate the function by using a UDF to return a NULL value if an empty string is produced. Select the appropriate SSMA conversion option as shown in Figure 4.

If the expression is of NCHAR or NVARCHAR data type:

SYSDB.SSMA\_SYB.REPLICATE\_NVARCHAR(char\_expr | uchar\_expr, integer\_expr)

Otherwise:

SYSDB.SSMA\_SYB.REPLICATE\_VARCHAR(char\_expr | uchar\_expr, integer\_expr)



Figure 4

### RTRIM

Sybase syntax:

rtrim(char\_expr | uchar\_expr)

The RTRIM function resembles the SQL Server RTRIM function. SSMA can emulate RTRIM by UDF depending on the project setting.

If the expression is of NCHAR or NVARCHAR data type:

SYSDB.SSMA\_SYB.RTRIM\_NVARCHAR(char\_expr | uchar\_expr)

Otherwise:

SYSDB.SSMA\_SYB.RTRIM\_VARCHAR(char\_expr | uchar\_expr)

### LEN

See CHAR\_LENGTH.

### SUBSTRING

Sybase syntax:

substring(expression, start, length )

In most cases you can migrate the SUBSTRING function as is. The result of SQL Server SUBSTRING functions is different if *start* < 0, or if the function produces an empty string. To provide compatibility, SSMA can emulate it by using a UDF, which is chosen depending on the parameter data type. SSMA has an appropriate conversion option.

If *expression* is of NCHAR or NVARCHAR data type, convert to:

SYSDB.SSMA\_SYB.SUBSTRING\_NVARCHAR(expression, start, length)

Otherwise convert to:

SYSDB.SSMA\_SYB.SUBSTRING\_VARCHAR(expression, start, length)

**Note**   The SQL Server SUBSTRING function does not support Unicode surrogate pairs.

### TO\_UNICHAR

Sybase syntax:

to\_unichar (integer\_expr)

You can replace the TO\_UNICHAR function with the SQL Server NCHAR() if *integer\_expr* < 65536. Otherwise, emulate it by UDF:

SYSDB.SSMA\_SYB.TO\_UNICHAR(integer\_expr)

**Note**   SSMA always replaces TO\_UNICHAR by emulation UDF to provide full compatibility.

### CONVERT

Sybase syntax:

convert (datatype [(length) | (precision[, scale])]

[null | not null], expression [, style])

The CONVERT function resembles the corresponding SQL Server function. You can migrate it as is, except when *[null | not null]* is specified. Because SQL Server does not support result nullability, SSMA marks this with an error. Otherwise, the function is converted as is by using the following code:

CONVERT **(** data\_type [ **(** length | (precision[, scale]**)** ] **,** expression [ **,** style ] **)**

### PAGESIZE

Sybase syntax:

pagesize(object\_name [, index\_name])

| pagesize(object\_id [,db\_id [, index\_id]])

In SQL Server, the page size is fixed at 8192. Therefore, convert the PAGESIZE function call to constant 8192.

### UHIGHSURR

Sybase syntax:

uhighsurr(uchar\_expr, start)

The UHIGHSURR function cannot be matched to any SQL Server system function. Emulate it by scalar UDF:

SYSDB.SSMA\_SYB.UHIGHSURR(uchar\_expr, start)

### ULOWSURR

Sybase syntax:

ulowsurr(uchar\_expr, start)

The ULOWSURR function cannot be matched to any Transact-SQL system function. Emulate it by scalar UDF:

SYSDB.SSMA\_SYB.ULOWSURR(uchar\_expr, start)

### USCALAR

Sybase syntax:

uscalar(uchar\_expr)

The USCALAR function is functionally equivalent to the SQL Server UNICODE() function:

UNICODE **(** ncharacter\_expression **)**

### PATINDEX

Sybase syntax:

patindex("%pattern%", char\_expr|uchar\_expr [, using {bytes | characters | chars} ] )

You can migrate the PATINDEX function to the SQL Server PATINDEX function except when *using bytes* is specified:

PATINDEX **(** **'**%pattern%**'** **,** char\_expr|uchar\_expr **)**

If *using bytes* is specified, convert to:

DATALENGTH(SUBSTRING(char\_expr|uchar\_expr, 1, PATINDEX('%pattern%', char\_expr|uchar\_expr) – 1)) + 1

### DATEDIFF

Sybase syntax:

datediff(datepart, date expression1, date expression2)

The DATEDIFF function resembles the SQL Server DATEDIFF function, except for the following dateparts:

* WEEKDAY
* HOUR
* MINUTE
* SECOND
* CALWEEKOFYEAR
* CALDAYOFWEEK
* CALYEAROFWEEK

For all other dateparts, you can use it as is. To provide full compatibility, SSMA always emulates the function by Transact-SQL UDF:

SYSDB.SSMA\_SYB.SSMA\_DATEDIFF(‘datepart’, date expression1, date expression2)

### INDEX\_COLORDER

Sybase syntax

index\_colorder (object\_name, index\_id, key\_# [, user\_id])

The INDEX\_COLORDER function doesn’t match any SQL Server system function. You can emulate it by UDF SYSDB.SSMA\_SYB.INDEX\_COLORDER.

By default, SSMA marks the function call with an error because there is a risk that the INDEX\_ID or USER\_ID parameters differ in Sybase and SQL Server. If the appropriate option is checked (see Figure 5), the function call is converted according to the following rules:

If *user\_id* is specified, convert to:

SYSDB.SSMA\_SYB.INDEX\_COLORDER(object\_name, index\_id, key\_#, user\_id)

Otherwise, convert to:

SYSDB.SSMA\_SYB.INDEX\_COLORDER(object\_name, index\_id, key\_#, DEFAULT)



**Figure 5**

### COL\_NAME

Sybase syntax:

col\_name(object\_id, column\_id[, database\_id])

The COL\_NAME function resembles the SQL Server COL\_NAME function. If *database\_id* is not specified, you can migrate it as is:

COL\_NAME **(** table\_id **,** column\_id **)**

Otherwise, emulate the function by using SQL Server data dictionary system views. SSMA does not currently support the function's *database\_id* parameter.

**Note**   The *object\_id* parameter can be system-specific and have other values in SQL Server databases.

### OBJECT\_NAME

Sybase syntax:

object\_name(object\_id[, database\_id])

The OBJECT\_NAME function resembles the corresponding SQL Server function. If *database\_id* is not specified, convert it to:

OBJECT\_NAME **(** object\_id **)**

Otherwise, emulate the function by using SQL Server data dictionary system views. SSMA does not currently support the function's *database\_id* parameter.

**Note**   The *object\_id* parameter can be system-specific and have other values in SQL Server databases.

### DATEPART

Sybase syntax:

datepart(date\_part, date expression)

The DATEPART function resembles the SQL Server DATEPART function, except for the following dateparts:

* CALDAYOFWEEK
* CALYEAROFWEEK
* CALWEEKOFYEAR

For all other dateparts you can migrate it as is. To support CALDAYOFWEEK, CALYEAROFWEEK and CALWEEKOFYEAR dateparts, use the following emulation UDF:

SYSDB.SSMA\_SYB.SSMA\_DATEPART(‘date\_part’, date expression)

### INDEX\_COL

Sybase syntax:

index\_col (object\_name, index\_id, key\_# [, user\_id])

The INDEX\_COL function resembles the SQL Server INDEX\_COL function, except when using the *user\_id* parameter. If the parameter is not specified, you can migrate INDEX\_COL by using the following code:

INDEX\_COL **(** **'**[ database\_name **.** [ schema\_name ] **.**| schema\_name ] table\_or\_view\_name**',** index\_id **,** key\_id **)**

Otherwise, emulate the function using data-dictionary system views. (SSMA does not support the *User\_id* parameter.)

**Note**   The *index\_id* parameter can have a different value in SQL Server and Sybase. Because of this, SSMA has a conversion option (Figure 6). If the option is not selected, the INDEX\_COL function reference is marked with an error.



**Figure 6**

### CURRENT\_TIME

Sybase syntax:

current\_time()

The CURRENT\_TIME function has no equivalent in SQL Server. You can emulate it by using the following UDF:

SYSDB.SSMA\_SYB.SSMA\_CURRENT\_TIME()

# Data Migration Architecture of SSMA for Sybase

This section describes SSMA components and their interaction during data migration. The components execute on different computers and use SQL Server database objects for communication. This architecture produces the best migration performance and the most flexibility. Understanding this mechanism helps you to set up the proper environment for SSMA data migration, and to control, monitor, and optimize the process.

## General

We based our implementation on the **SqlBulkCopy** class, defined in the .NET Framework 2.0. The functionality of **SqlBulkCopy** is similar to the **bcp** utility, which enables transferring large amounts of data quickly and efficiently. The source database is accessed by either Sybase ADO.NET provider or Sybase OLE DB provider 12.5.*x* or later.

The implementation satisfies the following four requirements:

* The data transfer process must run on SQL Server. This limits the number of Sybase clients that are installed and reduces network traffic.
* The client application controls the process by using SQL Server stored procedures. Therefore, we do not need any additional communication channels with the server and can reuse the existing server connection for this purpose.
* All tables selected for the migration are transferred by a single execution command from the SSMA user.
* The user monitors the data flow progress and can terminate it at any time.

## Solution Layers

Four layers participate in the data migration process:

* The client application, an SSMA executable
* Stored procedures that serve as interfaces to all server actions
* The database layer, which is composed of two tables:
	+ Package information table
	+ Status table
* The server executable, which is started as part of a SQL Server job, and which executes the data transfer and reflects its status

## Client Application

SSMA lets the user choose an arbitrary set of source tables for migration. The batch size for bulk copy operations is a user-defined setting.

When the process starts, the program displays the progress bar and a **Stop** button. If any errors are found, the SSMA shows the corresponding error message and terminates the transfer. The user can click **Stop** to terminate the process. If the transfer is completed normally, SSMA compares the number of rows in each source with the corresponding target table. If they are equal, the transfer is considered to be successful.

Because the client application does not directly control the data migration process, SSMA uses a Messages table to receive feedback on the status of the migration.

## Stored Procedures Interface

The following SQL Server stored procedures control the migration process:

* **bcp\_save\_migration\_package**: Writes package ID and xml parameters into the **bcp\_migration\_packages** table.
* **bcp\_start\_migration\_process**: Creates the SQL Server job that starts the migration executable and returns the ID of the job created.
* **bcp\_read\_new\_migration\_messages**: Returns rows added by the migration executable, filtered by known job ID.
* **stop\_agent\_process**: Stops the migration job, including closing the original connections and killing the migration executable. The data will be migrated partially.
* **bcp\_clean\_migration\_data**: Performs cleanup for a migration job.
* **bcp\_post\_process**: Runs all post-processing tasks for one migrated table.

## Database Layer

SSMA uses a Packages table, named **[ssma\_syb].[bcp\_migration\_packages]**, to store the information about the current package. Each row corresponds to one migration run. It contains a package GUID and xml that represents RSA-encrypted connection strings and the tables that should be migrated.

A Messages table, named **[ssma\_syb].[ssmafs\_bcp\_migration\_messages]**, accumulates messages coming from migration executables during its work.

## Migration Executable

The migration application, SSMA for Sybase Data Migration Assistant.exe, is executed on a SQL Server host. The executable's directory is determined during the Extension Pack installation. When **bcp\_start\_migration\_package** starts the application, it uses hard-coded file names and retrieves the directory name from a server environment variable.

When started, the migration application gets the package ID from a command string and reads all other package-related information from the Packages table. This information includes the source and destination connection strings and a list of the tables to be migrated. Then the tables are processed one at a time. SSMA gets source rows via the **IDataReader** interface and moves them to the target table by using the **WriteToServer** method.

The **BatchSize** setting defines the number of rows in a buffer. When the buffer is full, all rows in it are committed to the target.

Three types of transformation are applied to the processed rows:

* The data of all nullable CHAR, NCHAR, and UNICHAR columns are right-trimmed.
* National characters are converted to Unicode before the transfer.
* Dates that precede 01/01/1753 are converted according to the setting in the project options.

To notify the user about the progress of a bulk copy operation, the data migration executable uses the **SqlRowsCopied** event and **NotifyAfter** property. When the **SqlRowsCopied** event is generated, the application inserts new rows and updates the progress information in the Messages table. The **NotifyAfter** property defines the number of rows that are processed before a **SqlRowsCopied** event is generated. This value equals 25 percent of the row count of the source table.

Another type of output record, the *termination message*, is written to the Messages table when the application terminates successfully or when an exception occurs. In the latter case, the error text is included. If **BatchSize** = 1, SSMA extracts additional information about the columns of the row where the problem occurred, so that the user can locate the problematic row.

## Message Handling

The client application receives feedback from the migration executable by means of the Messages table. During migration, the client is in the loop polling this table and verifying that new rows with the proper package ID appear there. If no new rows are added for a significant period of time, there might be problems with the server executable and the process should terminate with a timeout message.

When the table migration completes, the server executable writes a successful completion message. If the table is large enough, you may see many intermediate messages—these show that the next batch was successfully committed. If there is an error, the client displays the error message that it received from the server process.

## Validation of the Results

Before the migration starts, the client application calculates the numbers of rows in each table that will be migrated. With this data, SSMA can evaluate the correct progress position.

After the migration is complete, the client must calculate the row counts of the target table. If they are equal, the overall migration result is considered to be successful. Otherwise, the user should be aware of the discrepancy and see the source and destination counts.

# Migrating Applications from CT-Library to ODBC

This section outlines the migration of a client application when it calls the Sybase CT-Library to provide database layer access. We suggest that the converted application use ODBC calls to SQL Server 2008. SSMA, however, does not support this transformation. This section has hints on how to map from one library to the other. It includes a simple demo CT-library application, both in the original and the converted form.

## Module Mapping

Following are the necessary headers for the CT-Library and the ODBC API.

|  |  |
| --- | --- |
| **CT-Library** | **ODBC** |
| #include <ctpublic.h> | #include <sql.h>#include <sqlext.h> |

## Command Mapping

The following table lists the function calls that a typical CT-Library application performs, and it shows the ODBC functions that have similar meaning.

| **CT-Library** | **ODBC** | **CT-Library** |
| --- | --- | --- |
| **STEP 1: Setting up the client-library programming environment** | **STEP 1: Connecting to the data source** | **STEP 1: Setting up the client-library programming environment** |
| cs\_ctx\_allocAllocates a context structure. |  | cs\_ctx\_allocAllocates a context structure. |
| cs\_configSets any CS-Library properties for the context. |  | cs\_configSets any CS-Library properties for the context. |
| ct\_initInitializes the Client-Library. | SQLAllocHandle(ENV)Loads the Driver Manager and allocates the environment handle. The Driver Manager allocates a structure in which to store information about the environment, and returns the environment handle. | ct\_initInitializes the Client-Library. |
| ct\_configSets the Client-Library properties for the context. | SQLSetEnvAttrSets attributes that govern aspects of environments. | ct\_configSets the Client-Library properties for the context. |
| **STEP 2: Define Error Handling** |  | **STEP 2: Define Error Handling** |
| cs\_config(CS\_MESSAGE\_CB)Installs a CS-Library error callback. | The cs\_config message is replaced with the functions SQLGetDiagField and SQLGetDiagRec in the return code handling procedures.SQLGetDiagField returns the current value of a field of a record of the diagnostic data structure (associated with a specified handle) that contains error, warning, and status information.SQLGetDiagRec returns the current values of multiple fields of a diagnostic record that contains error, warning, and status information. Unlike SQLGetDiagField, which returns one diagnostic field per call, SQLGetDiagRec returns several commonly used fields of a diagnostic record, including the SQLSTATE, the native error code, and the diagnostic message text. | cs\_config(CS\_MESSAGE\_CB)Installs a CS-Library error callback. |
| ct\_callbackInstalls a client message callback.Installs a server message callback.ct\_callback installs Client-Library callback routines, which are applicationroutines that Client-Library calls automatically when a triggering event of the appropriate type occurs. |  | ct\_callbackInstalls a client message callback.Installs a server message callback.ct\_callback installs Client-Library callback routines, which are applicationroutines that Client-Library calls automatically when a triggering event of the appropriate type occurs. |
| **STEP 3: Connect to a server** |  | **STEP 3: Connect to a server** |
| ct\_con\_allocAllocates a connection structure. | SQLAllocHandle(DBC)Allocates a connection handle. Driver Manager allocates a structure in which to store information about the connection and returns the connection handle. | ct\_con\_allocAllocates a connection structure. |
| ct\_con\_propsSets any properties in the connection structure. | SQLSetConnectAttrSets attributes that govern aspects of connections. Some connection attributes must be set before the application attempts to connect; others can be set after the connection is established. | ct\_con\_propsSets any properties in the connection structure. |
| ct\_connectOpens a connection to a server. | SQLConnect or SQLDriverConnect or SQLBrowseConnectThese functions establish connections to a driver and a data source. | ct\_connectOpens a connection to a server. |
| ct\_optionsSets any server options for this connection. |  | ct\_optionsSets any server options for this connection. |
| **STEP 4: Send a language command to the server** | **STEP 2: Initialize the application** | **STEP 4: Send a language command to the server** |
| ct\_cmd\_allocAllocates a command structure. | SQLAllocHandle(STMT)Driver Manager allocates a structure in which to store information about the statement and calls SQLAllocHandle in the driver with the SQL\_HANDLE\_STMT option. The driver allocates its own structure in which to store information about the statement and returns the driver statement handle to the Driver Manager. The Driver Manager returns the Driver Manager statement handle. | ct\_cmd\_allocAllocates a command structure. |
| ct\_cmd\_propsSets, retrieves, or clears command structure properties. | SQLSetStmtAttrSets attributes related to a statement. | ct\_cmd\_propsSets, retrieves, or clears command structure properties. |
|  | **STEP 3: Build and execute an SQL statement** |  |
| ct\_commandDefines a command. | SQLPreparePrepares an SQL string for execution. | ct\_commandDefines a command. |
| ct\_param or ct\_setparamDefines a command parameter. | SQLBindParameterBinds a buffer to a parameter marker in an SQL statement. | ct\_param or ct\_setparamDefines a command parameter. |
| ct\_sendSends the command text to the server, which parses, compiles, and executes it. | SQLExecuteExecutes a prepared statement by using the current values of the parameter marker variables. | ct\_sendSends the command text to the server, which parses, compiles, and executes it. |
| **STEP 5: Process the results of the command** | **STEP 4: Fetch the results; fetch the row count** | **STEP 5: Process the results of the command** |
| ct\_resultsSets up result data to be processed. Defines the types of a command’s execution result:* Values that indicate command status
* Values that indicate fetchable results
* Values that indicate that information is available
 |  | ct\_resultsSets up result data to be processed. Defines the types of a command’s execution result:* Values that indicate command status
* Values that indicate fetchable results
* Values that indicate that information is available
 |
| ct\_res\_infoRetrieves the current result set or command information.Possible information returned: * The number of the command that generated the current result set
* The number of compute clauses in the current command
* The number of items in the current result set
* The number of columns specified in the current command's ORDER BY clause
* The number of rows affected by the current command

…and so on. | SQLNumResultColsReturns the number of columns in a result set. If 0, the statement did not create a result set; if any other number, the statement did create a result set.SQLRowCountReturns the number of rows affected by an UPDATE, INSERT, or DELETE statement. If a batch of SQL statements is executed, the count of the affected rows might be a total count for all statements in the batch or individual counts for each statement in the batch. | ct\_res\_infoRetrieves the current result set or command information.Possible information returned:* The number of the command that generated the current result set
* The number of compute clauses in the current command
* The number of items in the current result set
* The number of columns specified in the current command's ORDER BY clause
* The number of rows affected by the current command

…and so on. |
| ct\_describeReturns a description of the result data. An application can use ct\_describe to retrieve a description of a regular result column, a return parameter, a stored procedure return status number, or a compute column.An application can call ct\_res\_info to find out how many result items are present in the current result set.An application generally calls ct\_describe to describe a result data item before it binds the result item to a program variable by using ct\_bind. | SQLDescribeColReturns the result descriptor—column name, type, column size, decimal digits, and nullability—for one column in the result set. | ct\_describeReturns a description of the result data. An application can use ct\_describe to retrieve a description of a regular result column, a return parameter, a stored procedure return status number, or a compute column.An application can call ct\_res\_info to find out how many result items are present in the current result set.An application generally calls ct\_describe to describe a result data item before it binds the result item to a program variable by using ct\_bind. |
| ct\_bindBinds server results to program variables. When the application calls ct\_fetch to fetch the result data, it is copied into these variables. | SQLBindColBinds application data buffers to columns in the result set. | ct\_bindBinds server results to program variables. When the application calls ct\_fetch to fetch the result data, it is copied into these variables. |
| ct\_fetchFetches result data. | SQLFetch Fetches the next rowset of data from the result set and returns data for all bound columns.SQLGetDataRetrieves data for a single column in the result set. It can be called multiple times to retrieve variable-length data in parts.The application now calls SQLFetch to retrieve the first row of data and place the data from that row in the variables bound with SQLBindCol. If there is any **long** data in the row, it calls SQLGetData to retrieve that data. The application continues to call SQLFetch and SQLGetData to retrieve additional data. After it has finished fetching data, it calls SQLCloseCursor to close the cursor. | ct\_fetchFetches result data. |
|  | SQLCloseCursorCloses a cursor that has been opened on a statement and discards pending results. |  |
|  | **STEP 5: Commit the transaction**The application performs Step 5 only if it set the transaction commit mode to manual commit; if the transaction commit mode is auto commit, which is the default, the transaction is automatically committed when the statement is executed. |  |
|  | **SQLEndTran**Requests a commit or rollback operation for all active operations on all statements associated with a connection. |  |
| **STEP 6: Finish** | **STEP 6: Disconnect from the data source** | **STEP 6: Finish** |
| ct\_cmd\_dropDeallocates a command structure. | SQLFreeHandle(STMT)Frees the statement. The driver releases the structure used to store information about the statement. | ct\_cmd\_dropDeallocates a command structure. |
| ct\_closeCloses a connection. | SQLDisconnectFrees any statements allocated on the connection and disconnects the driver from the data source. | ct\_closeCloses a connection. |
| ct\_exitExits Client-Library for a specific context. Closes and deallocates any open connections and cleans up internal Client-Library data space. | SQLFreeHandle(DBC)Frees the connection. The driver releases the structure used to store information about the connection. | ct\_exitExits Client-Library for a specific context. Closes and deallocates any open connections and cleans up internal Client-Library data space. |
| cs\_ctx\_dropDeallocates a context structure. | SQLFreeHandle(ENV)Frees the environment handle. The driver releases the structure used to store information about the environment. | cs\_ctx\_dropDeallocates a context structure. |

## Type and Structure Mapping

Some definitions of CT-Library and ODBC have similar meanings. The following table shows the correspondence between them.

| **CT-Library** | **ODBC** |
| --- | --- |
| **Structures** |  |
| CS\_CONTEXTContext structure | HENVEnvironment handle |
| CS\_CONNECTIONConnection structure | HDBCConnection handle |
| CS\_COMMANDCommand structure | HSTMTStatement handle |
| **Types** |  |
| CS\_TINYINTtinyint | SQLSCHAR (SQL\_C\_STINYINT) |
| CS\_SMALLINTsmallint | SQLSMALLINT (SQL\_C\_SSHORT) |
| CS\_USMALLINTusmallint | SQLUSMALLINT (SQL\_C\_USHORT) |
| CS\_INTint | SQLINTEGER (SQL\_C\_SLONG) |
| CS\_UINTuint | SQLUINTEGER (SQL\_C\_ULONG) |
| CS\_BIGINTbigint | SQLBIGINT (SQL\_C\_SBIGINT) |
| CS\_UBIGINTubigint | SQLUBIGINT (SQL\_C\_UBIGINT) |
| CS\_DECIMALdecimal | SQL\_NUMERIC\_STRUCT structure |
| CS\_NUMERICnumeric | SQL\_NUMERIC\_STRUCT structure |
| CS\_MONEYmoney | SQL\_NUMERIC\_STRUCT structure |
| CS\_MONEY4smallmoney | SQL\_NUMERIC\_STRUCT structure |
| CS\_FLOATfloat | SQLDOUBLE, SQLFLOAT (SQL\_C\_DOUBLE) |
| CS\_REALreal | SQLREAL (SQL\_C\_FLOAT) |
| CS\_CHARchar varchar | SQLCHAR (SQL\_C\_CHAR) |
| CS\_UNICHARunichar univarchar | SQLWCHAR (SQL\_C\_WCHAR) |
| CS\_DATEdate | SQL\_DATE\_STRUCT structure |
| CS\_TIMEtime | SQL\_TIME\_STRUCT structure |
| CS\_DATETIMEdatetime | SQL\_TIMESTAMP\_STRUCT structure |
| CS\_DATETIME4smalldatetime | SQL\_TIMESTAMP\_STRUCT structure |
| CS\_BITbit | SQLCHAR (SQL\_C\_BIT) |
| CS\_BINARYbinary varbinary | SQLCHAR (SQL\_C\_BINARY) |
| CS\_TEXTtext | SQLCHAR (SQL\_C\_CHAR) |
| CS\_IMAGEimage | SQLCHAR (SQL\_C\_BINARY) |
| CS\_UNITEXTunitext | SQLWCHAR (SQL\_C\_WCHAR) |
| **Structures** |  |
| CS\_CONTEXTContext structure | HENVEnvironment handle |
| CS\_CONNECTIONConnection structure | HDBCConnection handle |
| CS\_COMMANDCommand structure | HSTMTStatement handle |
| **Types** |  |
| CS\_TINYINTtinyint | SQLSCHAR (SQL\_C\_STINYINT) |
| CS\_SMALLINTsmallint | SQLSMALLINT (SQL\_C\_SSHORT) |
| CS\_USMALLINTusmallint | SQLUSMALLINT (SQL\_C\_USHORT) |
| CS\_INTint | SQLINTEGER (SQL\_C\_SLONG) |
| CS\_UINTuint | SQLUINTEGER (SQL\_C\_ULONG) |
| CS\_BIGINTbigint | SQLBIGINT (SQL\_C\_SBIGINT) |
| CS\_UBIGINTubigint | SQLUBIGINT (SQL\_C\_UBIGINT) |
| CS\_DECIMALdecimal | SQL\_NUMERIC\_STRUCT structure |
| CS\_NUMERICnumeric | SQL\_NUMERIC\_STRUCT structure |
| CS\_MONEYmoney | SQL\_NUMERIC\_STRUCT structure |
| CS\_MONEY4smallmoney | SQL\_NUMERIC\_STRUCT structure |
| CS\_FLOATfloat | SQLDOUBLE, SQLFLOAT (SQL\_C\_DOUBLE) |
| CS\_REALreal | SQLREAL (SQL\_C\_FLOAT) |
| CS\_CHARchar varchar | SQLCHAR (SQL\_C\_CHAR) |
| CS\_UNICHARunichar univarchar | SQLWCHAR (SQL\_C\_WCHAR) |
| CS\_DATEdate | SQL\_DATE\_STRUCT structure |
| CS\_TIMEtime | SQL\_TIME\_STRUCT structure |
| CS\_DATETIMEdatetime | SQL\_TIMESTAMP\_STRUCT structure |
| CS\_DATETIME4smalldatetime | SQL\_TIMESTAMP\_STRUCT structure |
| CS\_BITbit | SQLCHAR (SQL\_C\_BIT) |
| CS\_BINARYbinary varbinary | SQLCHAR (SQL\_C\_BINARY) |
| CS\_TEXTtext | SQLCHAR (SQL\_C\_CHAR) |
| CS\_IMAGEimage | SQLCHAR (SQL\_C\_BINARY) |
| CS\_UNITEXTunitext | SQLWCHAR (SQL\_C\_WCHAR) |

## Migration Example

The sample program in this section demonstrates how a CT-Library-based program can be converted to work on ODBC API. The program executes basic function calls for typical tasks such as establishing connections to a database server, executing SELECT queries, and retrieving the result set. The program also includes error processing.

The first code listing is the sample program in Chapter 1 of the *Sybase Client-library Programmer's Guide*. The sample program connects to a Sybase server, sends a select query, prints the rows, disconnects, and exits.

The second code example is the same program rewritten to work with ODBC API and to connect to Microsoft SQL Server.

### Source Programming Using CT-Library

#include <stdio.h>

#include <ctpublic.h>

#define MAXCOLUMNS 2

#define MAXSTRING 40

#define ERR\_CH stderr

#define OUT\_CH stdout

#define EX\_MAXSTRINGLEN 255

#define EX\_BUFSIZE 1024

#define EX\_CTLIB\_VERSION CS\_VERSION\_125

#define EX\_BLK\_VERSION BLK\_VERSION\_125

#define EX\_ERROR\_OUT stderr

#define EX\_EXIT\_SUCCEED 0

#define EX\_EXIT\_FAIL 1

//place user name here

CS\_CHAR \*Ex\_username = "";

//place user password here

CS\_CHAR \*Ex\_password = "";

//place ODBC DSN here

CS\_CHAR \*Ex\_server = "";

/\*\* Define a macro that exits if a function return code indicates failure.\*/

#define EXIT\_ON\_FAIL(context, ret, str) \

 if (ret != CS\_SUCCEED) \

 { \

 fprintf(ERR\_CH, "Fatal error: %s\n", str); \

 if (context != (CS\_CONTEXT \*) NULL) \

 { \

 (CS\_VOID) ct\_exit(context, CS\_FORCE\_EXIT); \

 (CS\_VOID) cs\_ctx\_drop(context); \

 } \

 exit(EX\_EXIT\_FAIL); \

 }

/\*

\*\* Callback routines for library errors and server messages.

\*/

CS\_RETCODE CS\_PUBLIC csmsg\_callback PROTOTYPE((CS\_CONTEXT \*context,CS\_CLIENTMSG \*clientmsg ));

CS\_RETCODE CS\_PUBLIC clientmsg\_callback PROTOTYPE((CS\_CONTEXT \*context,CS\_CONNECTION \*connection,CS\_CLIENTMSG \*clientmsg ));

CS\_RETCODE CS\_PUBLIC servermsg\_callback PROTOTYPE((CS\_CONTEXT \*context,CS\_CONNECTION \*connection,CS\_SERVERMSG \*servermsg ));

/\* Main entry point for the program.\*/

int main(int argc,char \*\*argv)

{

 CS\_CONTEXT \*context; /\* Context structure \*/

 CS\_CONNECTION \*connection; /\* Connection structure. \*/

 CS\_COMMAND \*cmd;/\* Command structure. \*/

 /\* Data format structures for column descriptions: \*/

 CS\_DATAFMT columns[MAXCOLUMNS];

 CS\_INT datalength[MAXCOLUMNS];

 CS\_SMALLINT indicator[MAXCOLUMNS];

 CS\_INT count;

 CS\_RETCODE ret;

 CS\_RETCODE results\_ret;

 CS\_INT result\_type;

 CS\_CHAR name[MAXSTRING];

 CS\_CHAR city[MAXSTRING];

 /\*environment initialization\*/

 context = (CS\_CONTEXT \*)NULL;

 EXIT\_ON\_FAIL(context,cs\_ctx\_alloc(EX\_CTLIB\_VERSION, &context), "cs\_ctx\_alloc failed");

 EXIT\_ON\_FAIL(context,ct\_init(context, EX\_CTLIB\_VERSION), "ct\_init failed");

 /\*set up error handling: install callback handlers\*/

 EXIT\_ON\_FAIL(context,cs\_config(context, CS\_SET, CS\_MESSAGE\_CB,(CS\_VOID \*)csmsg\_callback,CS\_UNUSED, NULL),"cs\_config(CS\_MESSAGE\_CB) failed");

 EXIT\_ON\_FAIL(context,ct\_callback(context, NULL, CS\_SET, CS\_CLIENTMSG\_CB,(CS\_VOID \*)clientmsg\_callback),"ct\_callback for client messages failed");

 EXIT\_ON\_FAIL(context,ct\_callback(context, NULL, CS\_SET, CS\_SERVERMSG\_CB,(CS\_VOID \*)servermsg\_callback),"ct\_callback for server messages failed");

 /\*connecting to server using server name, user name, and password\*/

 EXIT\_ON\_FAIL(context, ct\_con\_alloc(context, &connection), "ct\_con\_alloc() failed");

 EXIT\_ON\_FAIL(context, ct\_con\_props(connection, CS\_SET, CS\_USERNAME,Ex\_username, CS\_NULLTERM, NULL), "Could not set user name");

 EXIT\_ON\_FAIL(context, ct\_con\_props(connection, CS\_SET, CS\_PASSWORD,Ex\_password, CS\_NULLTERM, NULL), "Could not set password");

 EXIT\_ON\_FAIL(context, ct\_connect(connection, Ex\_server, strlen(Ex\_server)), "Could not connect!");

 /\*execute command\*/

 EXIT\_ON\_FAIL(context,ct\_cmd\_alloc(connection, &cmd), "ct\_cmd\_alloc() failed");

 EXIT\_ON\_FAIL(context,ct\_command(cmd, CS\_LANG\_CMD,"select au\_lname, city from pubs2..authors where state = 'CA'", CS\_NULLTERM, CS\_UNUSED), "ct\_command() failed");

 EXIT\_ON\_FAIL(context,ct\_send(cmd), "ct\_send() failed");

 /\*main process loop\*/

 while ((results\_ret = ct\_results(cmd, &result\_type)) == CS\_SUCCEED)

 {

 switch ((int)result\_type)

 {

 /\*resultset trapped, process it\*/

 case CS\_ROW\_RESULT:

 /\*binding columns\*/

 columns[0].datatype = CS\_CHAR\_TYPE;

 columns[0].format = CS\_FMT\_NULLTERM;

 columns[0].maxlength = MAXSTRING;

 columns[0].count = 1;

 columns[0].locale = NULL;

 EXIT\_ON\_FAIL(context,ct\_bind(cmd, 1, &columns[0],name, &datalength[0],&indicator[0]),"ct\_bind() for au\_lname failed");

 columns[1].datatype = CS\_CHAR\_TYPE;

 columns[1].format = CS\_FMT\_NULLTERM;

 columns[1].maxlength = MAXSTRING;

 columns[1].count = 1;

 columns[1].locale = NULL;

 EXIT\_ON\_FAIL(context,ct\_bind(cmd, 2, &columns[1], city,&datalength[1],&indicator[1]),"ct\_bind() for city failed");

 /\*fetching data\*/

 while (((ret = ct\_fetch(cmd, CS\_UNUSED, CS\_UNUSED,CS\_UNUSED, &count))== CS\_SUCCEED)|| (ret == CS\_ROW\_FAIL))

 {

 if (ret == CS\_ROW\_FAIL) fprintf(ERR\_CH,"Error on row %ld.\n",(long)(count + 1));

 fprintf(OUT\_CH, "%s: %s\n", name, city);

 }

 if (ret == CS\_END\_DATA)

 {fprintf(OUT\_CH,"\nAll done processing rows.\n");}

 else

 {EXIT\_ON\_FAIL(context, CS\_FAIL, "ct\_fetch failed");}

 break;

 case CS\_CMD\_SUCCEED: fprintf(OUT\_CH, "No rows returned.\n"); break;

 case CS\_CMD\_FAIL : break;

 case CS\_CMD\_DONE : break;

 default : EXIT\_ON\_FAIL(context, CS\_FAIL,"ct\_results returned unexpected result type"); break;

 }

 }

 switch ((int)results\_ret)

 {

 case CS\_END\_RESULTS: break;

 case CS\_FAIL:EXIT\_ON\_FAIL(context, CS\_FAIL,"ct\_results() returned CS\_FAIL."); break;

 default: EXIT\_ON\_FAIL(context, CS\_FAIL,"ct\_results returned unexpected return code"); break;

 }

 /\*cleanup\*/

 EXIT\_ON\_FAIL(context,ct\_cmd\_drop(cmd), "ct\_cmd\_drop failed");

 EXIT\_ON\_FAIL(context,ct\_close(connection, CS\_UNUSED), "ct\_close failed");

 EXIT\_ON\_FAIL(context,ct\_con\_drop(connection), "ct\_con\_drop failed");

 EXIT\_ON\_FAIL(context,ct\_exit(context, CS\_UNUSED), "ct\_exit failed");

 EXIT\_ON\_FAIL(context,cs\_ctx\_drop(context), "cs\_ctx\_drop failed");

 exit(EX\_EXIT\_SUCCEED);

}

CS\_RETCODE CS\_PUBLIC

servermsg\_callback(cp, chp, msgp)

CS\_CONTEXT \*cp;

CS\_CONNECTION \*chp;

CS\_SERVERMSG \*msgp;

{

 fprintf(ERR\_CH,"Server message:\n\t");

 fprintf(ERR\_CH,"number(%ld) severity(%ld) state(%ld) line(%ld)\n",(long)msgp->msgnumber, (long)msgp->severity,

 (long)msgp->state, (long)msgp->line);

 if (msgp->svrnlen > 0) fprintf(ERR\_CH, "\tServer name: %s\n", msgp->svrname);

 if (msgp->proclen > 0) fprintf(ERR\_CH, "\tProcedure name: %s\n", msgp->proc);

 fprintf(ERR\_CH, "\t%s\n", msgp->text);

 return (CS\_SUCCEED);

}

CS\_RETCODE CS\_PUBLIC

clientmsg\_callback(context, conn, emsgp)

CS\_CONTEXT \*context;

CS\_CONNECTION \*conn;

CS\_CLIENTMSG \*emsgp;

{

 fprintf(ERR\_CH,"Client Library error:\n\t");

 fprintf(ERR\_CH,"severity(%ld) number(%ld) origin(%ld) layer(%ld)\n",(long)CS\_SEVERITY(emsgp->severity),

 (long)CS\_NUMBER(emsgp->msgnumber),(long)CS\_ORIGIN(emsgp->msgnumber),

 (long)CS\_LAYER(emsgp->msgnumber));

 fprintf(ERR\_CH, "\t%s\n", emsgp->msgstring);

 if (emsgp->osstringlen > 0)

 {

 fprintf(ERR\_CH,

 "Operating system error number(%ld):\n",

 (long)emsgp->osnumber);

 fprintf(ERR\_CH, "\t%s\n", emsgp->osstring);

 }

 return (CS\_SUCCEED);

}

CS\_RETCODE CS\_PUBLIC

csmsg\_callback(context, emsgp)

CS\_CONTEXT \*context;

CS\_CLIENTMSG \*emsgp;

{

 fprintf(ERR\_CH, "CS-Library error:\n");

 fprintf(ERR\_CH,"\tseverity(%ld) layer(%ld) origin(%ld) number(%ld)",

 (long)CS\_SEVERITY(emsgp->msgnumber),(long)CS\_LAYER(emsgp->msgnumber),

 (long)CS\_ORIGIN(emsgp->msgnumber), (long)CS\_NUMBER(emsgp->msgnumber));

 fprintf(ERR\_CH, "\t%s\n", emsgp->msgstring);

 if (emsgp->osstringlen > 0) fprintf(ERR\_CH, "Operating System Error: %s\n",emsgp->osstring);

 return (CS\_SUCCEED);

}

### ODBC Equivalent

#include <windows.h>

#include <stdio.h>

#include <sql.h>

#include <sqlext.h>

//place user name here

CHAR \*Ex\_username = "";

//place user password here

CHAR \*Ex\_password = "";

//place ODBC DSN here

CHAR \*Ex\_server = "";

 SQLHENV ENV = NULL;

 SQLHDBC DBC = NULL ;

 SQLHSTMT STMT = NULL;

void cleanup()

{

 SQLFreeHandle(SQL\_HANDLE\_STMT,STMT);

 SQLFreeHandle(SQL\_HANDLE\_DBC,DBC);

 SQLFreeHandle(SQL\_HANDLE\_ENV,ENV);

}

void EXIT\_ON\_FAIL(int ret,char \*str)

{

 if (SQL\_ERROR== ret )

 {

 fprintf(stderr, "Fatal error: %s\n", str);

 cleanup();

 exit(SQL\_ERROR);

 }

}

int processerrors(int rc)

{

 if (SQL\_SUCCESS == rc) return rc;

 SQLRETURN plm\_retcode = SQL\_SUCCESS;

 SQLCHAR sqlstate[1024], error[1024];

 SQLINTEGER nativerror;

 SQLSMALLINT len = 0;

 while (plm\_retcode != SQL\_NO\_DATA\_FOUND)

 {

 plm\_retcode = SQLError(ENV,DBC,STMT,(SQLCHAR \*)&sqlstate,&nativerror,(SQLCHAR \*)&error,1024,&len);

 if(SQL\_NO\_DATA\_FOUND == plm\_retcode) return rc;

 printf("SqlState: %s, Error: %s\n",sqlstate,error);

 }

 return rc;

}

int main()

{

 SQLCHAR buf[1024];

 char cstr[1024];

 SQLSMALLINT b;

 int rc;

 /\*environment initialization\*/

 EXIT\_ON\_FAIL(processerrors(SQLAllocHandle(SQL\_HANDLE\_ENV,0,&ENV)),"SQLAllocHandle(ENV)");

 EXIT\_ON\_FAIL(processerrors(SQLSetEnvAttr(ENV,SQL\_ATTR\_ODBC\_VERSION,(SQLPOINTER)SQL\_OV\_ODBC3,SQL\_IS\_INTEGER)),"SQLSetEnvAttr()");

 /\*connection initialization\*/

 EXIT\_ON\_FAIL(processerrors(SQLAllocHandle(SQL\_HANDLE\_DBC,ENV,&DBC)),"SQLAllocHandle(DBC)");

 /\*set up error handling: install callback handlers\*/

 /\*omitted: we should use processerrors when proc returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR\*/

 /\*connecting to server using server name, user name, and password\*/

 sprintf(cstr,"DSN=%s;UID=%s;PWD=%s;",Ex\_server,Ex\_username,Ex\_password);

 EXIT\_ON\_FAIL(processerrors(SQLDriverConnect(DBC,NULL,(SQLCHAR \*)&cstr,SQL\_NTS,(SQLCHAR \*)&buf,1024,&b,SQL\_DRIVER\_NOPROMPT)),"SQLDriverConnect()");

 /\*execute command\*/

 EXIT\_ON\_FAIL(processerrors(SQLAllocHandle(SQL\_HANDLE\_STMT,DBC,&STMT)),"SQLAllocHandle(STMT)");

 EXIT\_ON\_FAIL(processerrors(SQLSetStmtAttr(STMT, SQL\_ATTR\_ROW\_ARRAY\_SIZE, (SQLPOINTER)1, 0)),"");

 rc = processerrors(SQLExecDirect(STMT,(SQLCHAR \*)"select au\_lname, city from pubs2..authors where state = 'CA'",SQL\_NTS));

 char city[2000],name[2000];

 SQLINTEGER cityl,namel;

 /\*main process loop\*/

 do

 {

 switch (rc)

 {

 case SQL\_SUCCESS :

 case SQL\_SUCCESS\_WITH\_INFO : if(SQL\_SUCCESS\_WITH\_INFO==rc) processerrors(rc);

 SQLSMALLINT cc;

 EXIT\_ON\_FAIL(processerrors(SQLNumResultCols(STMT,&cc)),"SQLNumResultCols()");

 /\*resultset trapped, process it\*/

 /\*if cc (column count) isn't equal to zero - recordset is returned\*/

 if (0!=cc)

 {

 /\*binding columns\*/

 EXIT\_ON\_FAIL(processerrors(SQLBindCol(STMT,1,SQL\_C\_CHAR,name,2000,&namel)),"SQLBindCol(name)");

 EXIT\_ON\_FAIL(processerrors(SQLBindCol(STMT,2,SQL\_C\_CHAR,city,2000,&cityl)),"SQLBindCol(city)");

 /\*fetching data\*/

 while(SQL\_SUCCESS==SQLFetch(STMT))

 {

 fprintf(stdout, "%s: %s\n", name, city);

 }

 }

 break;

 case SQL\_ERROR : processerrors(rc); break;

 default : EXIT\_ON\_FAIL(SQL\_ERROR,"Unexpected rc");

 }

 /\*move to next recordset\*/

 rc = SQLMoreResults(STMT);

 }

 while (SQL\_NO\_DATA!=rc);

 /\*cleaup\*/

 cleanup();

 return SQL\_SUCCESS;

}

# Sybase Migration Issues

This section examines potential problems you may encounter during migration from Sybase to SQL Server 2008 and their possible solutions. This section covers Sybase 12.5.x and Sybase 15.0 features that work differently, or do not exist, in SQL Server 2008.

**Note**   All references to SSMA mean SSMA for Sybase version 4.0.

Each entry in this section includes three parts:

* Issue
* SSMA support
* Solution

The "SSMA support" section gives the implementation status. If SSMA support is Yes, the solution is the exact description of the SSMA conversion algorithm. If SSMA support is Partial, some manual intervention may be necessary, as explained in the Solution entry.

Be aware that many SSMA Sybase solutions exist in two modes:

* **Optimistic**. For less clumsy target code that resembles the original format; this type of code is likely to have only minimal incompatibilities. If you use this mode, you try to preserve the original Sybase text as much as possible and ignore the probability that this code may work differently from Sybase.
* **Full.** Simulate Sybase behavior as exactly as possible. In this mode, you try to create the best possible emulation, but this is done for code clarity and readability.

## Data Types

This section covers specific migration problems that can arise during data type migration.

### Length of String Types

**Issue**

The maximum length of character data columns and variables can be up to 16 KB, depending on page size. You must distinguish cases when the best conversion is to **char** and when **to varchar(max)**, and decide what to do with **char**(16384).

**SSMA support**

Partial

**Solution**

By default, SSMA maps the **char(n)** data type to **varchar(max)** in the case of *n* > 8000. You can select the following optional type mappings for the **char** data type in SSMA (Figure 7):

* **char**
* **varchar**
* **nchar**
* **nvarchar**
* **varchar(max)**
* **nvarchar(max)**



Figure 7

## Expressions

This section covers specific migration problems that can arise when expressions are migrated.

### Conversion of Time to String

**Issue**

By default, SSMA maps the Sybase **time** data type to the SQL Server **time** data type. But if you convert **time**, either explicitly or implicitly, to character types, you must remove the date part containing the year, month, and day, possibly with a **sysdb** function.

**SSMA support**

None

**Solution**

In all cases when you convert the time variable to character, replace the value with a call to the SSMA\_SYB.TIME\_FROM\_DATETIME function. This function extracts the time part from **datetime**. See the following code example.

**Note**   This replacement is not needed if:

* One time variable or column is assigned to the value of another time variable or column (SET, SELECT, and UPDATE statements).
* The value of the time variable is inserted into a time column (INSERT).
* TIME is the second argument in the DATEPART function.

Here is the code:

create function SSMA.TIME\_FROM\_DATETIME (@dt datetime)

returns varchar(40)

as

begin

 declare @str varchar(40),

 @dt\_len int,

 @last\_space\_pos int

set @str = convert(varchar(40), @dt)

set @dt\_len = len(@str)

set @last\_space\_pos = @dt\_len - charindex(' ', reverse(@str)) + 2

if @last\_space\_pos = @dt\_len return NULL

 return

 substring(@str, @last\_space\_pos, @dt\_len - @last\_space\_pos + 1)

end

### Unicode Literals

**Issue**

Sybase allows Unicode literals in U&’\*xxxx*’ format.

The exact syntax is:

U&<string\_literal>

or

u&<string\_ literal>

where *<string\_literal>* is any string literal that is valid in Sybase, enclosed in single or double quotes. U& indicates that this string contains Unicode escape sequences in \xxxx or \+yyyyyy format. Normal single-byte characters are also allowed.

Valid Unicode literals examples include:

U&’\0041’

u&”\0041KLM\0042XYZ”

U&”ABC\+000041”

**SSMA support**

None

**Solution**

Use binary literals for this purpose.

### Double-Quoted String Literals

**Issue**

Sybase can use double-quoted strings, which by default SQL Server 2008 does not support. You must convert them, accounting for the possibility of single and double quotes inside the string.

**SSMA support**

Yes

**Solution**

Change all double quotes (") that surround string literals to single quotes. If you find a single quote (') inside the string literal, replace it by two single quotes (‘’).

### Empty String

**Issue**

An empty string ("" or '') evaluates to ' ' (single space) in expressions, and is stored as a single blank in variables and table columns.

**SSMA support**

Yes

**Solution**

SSMA uses the following algorithm:

1. Change all empty string literal occurrences in character expressions to ' ' (single space). In Full mode, replace every string variable or string column <s> to:

(CASE <s> WHEN '' THEN ' ' ELSE <s> END)

1. Do not make this replacement in comparisons when the string variable or column are checked against a string literal that is not an empty string, is not a single space, or whose length exceeds 1.
2. Do not make this replacement for IS [NOT] NULL evaluations.
3. Do not apply this replacement to the string variables that receive a value during statement execution, including:
* Output parameters of a procedure call.
* Variables at the left side of an assignment (SET or SELECT).
* Variables from the INTO clause of the FETCH command.
1. Example expressions you should not replace by CASE include:
* @var = 'X'
* a\_col = 'SOME\_LONG\_STRING'
* a\_col IS NOT NULL
* 'U' = @arg1

### ‘||’ as Concatenation Operator

**Issue**

Sybase can use '||' along with '+' as a string concatenation operator.

**SSMA support**

Yes

**Solution**

Replace all '||' in string expressions with '+'.

### Concatenation with NULL

**Issue**

A Sybase string that is concatenated with NULL always yields the original string, which is not the default in SQL Server 2008. In SQL Server 2008 the result is the same only if the option CONCAT\_NULL\_YIELDS\_NULL is set to OFF.

**SSMA support**

Yes

**Solution**

In Full mode, SSMA replaces any variable string expression that participates in concatenation with ISNULL(<*expression*>, ‘’). In Optimistic mode, SSMA ignores the problem and issues a general warning with a proposal to use the SET CONCAT\_NULL\_YIELDS\_NULL OFF command.

**Note**  If CONCAT\_NULL\_YIELDS\_NULL is OFF, it may be impossible to use indexes on computed columns and indexed views in SQL Server 2008.

### Comparisons with NULL

**Issue**

If *<e>* is an expression and *@<v>* is a local variable or procedure parameter, the comparisons in the following table in Sybase will give results that are different from SQL Server 2008 in default ANSI\_NULLS mode.

|  |  |
| --- | --- |
| **Boolean expression** | **Result** |
| <e> = null | TRUE if <e> is NULL, and FALSE otherwise |
| <e> = @<v> | TRUE if both <e> and @<v> are NULLs, and FALSE if NULL is only one |
| <e> != null | TRUE if <e> is not NULL, and FALSE otherwise |
| <e> != @<v> | TRUE if one of <e> and @<v> is not NULL, and FALSE if they are both NULLs |

**SSMA support**

Yes

**Solution**

SSMA can account for the possibility of NULL values if you set the Equality check conversion option to Consider NULL values. This requires Full conversion mode.

### CASE Nesting Limit

**Issue**

CASE expressions in SQL Server 2008 can be nested only to level 10.

**SSMA support**

None

**Solution**

Generally, split long case expressions into several smaller parts.

### Implicit Conversion of Binary Types

**Issue**

Sybase can concatenate **binary** and **varbinary** variables and columns with string expressions. In this case, they are implicitly converted to varchar. Conversely, string expressions are implicitly converted when they are assigned to a **binary**/**varbinary** variable.

**SSMA support**

Yes

**Solution**

If any expression of **binary** or **varbinary** type is connected with a concatenation operator with any **char**, **varchar**, **nchar**, **nvarchar**, **unichar**, or **univarchar** expression, add an explicit conversion to **varchar(max)**.

Here is an example. Convert the following:

DECLARE @bin varbinary(100), @str varchar(20)

SET @str = 'ABC'

SET @bin = 0x41

SET @str = @str + @bin

Into the following:

DECLARE @bin varbinary(100), @str varchar(20)

SET @str = 'ABC'

SET @bin = 0x41

SET @str = @str + CONVERT(varchar(max),@bin)

If a character expression is assigned to a **binary** or **varbinary** variable or appears in an INSERT or UPDATE command that assigns the value to a **binary** or **varbinary** column, add explicit conversion to **varbinary(max)**.

For example, convert the following:

DECLARE @bin binary(10)

SET @bin = 'X'

Into the following:

DECLARE @bin binary(10)

SET @bin = CONVERT(varbinary(max),'X')

### Implicit Conversion of Datetime as Function Argument

**Issue**

If a system function has a **string** parameter, Sybase can accept datetime expressions as an actual value by implicitly converting them to **varchar**.

**SSMA support**

Yes

**Solution**

Cast the datetime value *<dt>* to the parameter type used in the function definition. The default is CAST (*<dt>* as nvarchar(4000)).

**Example**

Sybase

DECLARE @dt datetime

SET @dt = getdate()

select substring(@dt, 1, 10)

select len(@dt)

select rtrim(dateadd(month, 2, @dt))

SQL Server 2008

DECLARE @dt datetime

SET @dt = getdate()

SELECT substring(CAST(@dt AS nvarchar(max)), 1, 10)

SELECT sysdb.ssma\_syb.char\_length\_nvarchar(CAST(@dt AS nvarchar(4000)))

SELECT rtrim(CAST(dateadd(month, 2, @dt) AS nvarchar(max)))

## SQL Commands

This section covers specific migration problems that can arise when Sybase commands are converted to SQL commands.

### COMMIT

**Issue**

The COMMIT command can be executed without a prior BEGIN TRANSACTION statement.

**SSMA support**

Yes

**Solution**

Replace COMMIT … with IF (@@TRANCOUNT > 0) COMMIT …

### Different COMMIT Syntax

**Issue**

Sybase can use COMMIT *transaction\_name* and COMMIT WORK *transaction\_name* syntax that does not exist in SQL Server 2008.

**SSMA support**

Yes

**Solution**

Change all occurrences of COMMIT <word> or COMMIT WORK <word> to COMMIT TRANSACTION <word>, where *<word>* is a single word that is not equal to TRAN[SACTION].

### Quoted Data Type at CONVERT and CAST Functions

**Issue**

Sybase allows writing the target data type in CONVERT and CAST functions in single quotes and double quotes, as in this example:

 select convert('datetime',getdate())

 select cast(7.89/3.45 as ''float'')

**SSMA support**

None

**Solution**

Ignore these single quotes (or double quotes) during conversion.

### DEALLOCATE CURSOR

**Issue**

SQL Server 2008 does not support the DEALLOCATE CURSOR command.

**SSMA support**

Yes

**Solution**

Change DEALLOCATE CURSOR to DEALLOCATE.

### LOCK TABLE <T> IN SHARE | EXCLUSIVE MODE

**Issue**

SQL Server 2008 does not support the LOCK TABLE <T> IN SHARE | EXCLUSIVE MODE command.

**SSMA support**

Yes

**Solution**

The statement's full syntax is:

LOCK TABLE <T> IN {SHARE | EXCLUSIVE} MODE [ WAIT [ *NumSecs* ] | NOWAIT ]

If the source statement contains *NumSecs* or a NOWAIT parameter, it generates an error. Otherwise, implement table locking by using the DELETE statement.

If the mode is SHARE:

/\* Lock table <T> \*/

DELETE TOP (0) FROM <T> WITH (TABLOCK) WHERE 0=1

If the mode is EXCLUSIVE:

/\* Lock table <T> \*/

DELETE TOP (0) FROM <T> WITH (TABLOCKX) WHERE 0=1

**Example 1**

Sybase:

LOCK TABLE TableA IN SHARE MODE

Emulation in SQL Server 2008 is:

DELETE TOP (0) FROM TableA WITH (TABLOCK) WHERE 0=1

**Example 2**

Sybase:

LOCK TABLE TableA IN EXCLUSIVE MODE

Emulation in SQL Server 2008 is:

DELETE TOP (0) FROM TableA WITH (TABLOCKX) WHERE 0=1

### PRINT Command

**Issue**

SQL Server 2008 cannot use the PRINT command with a format string and arguments. In addition, Sybase interprets double percent (%%) in a format string as a single percent.

**SSMA support**

Yes

**Solution**

You have two options for converting a PRINT statement. The Optimistic option offers the simplest solution for converting commonly used statements. The more complicated Full option offers the most accuracy.

**Case 1. PRINT without arguments**

PRINT <format>

Assume that the *<format>* string is already converted according to the SSMA string conversion algorithm. In literal strings, replace all double percents (%%) with single percent (%).

In Full mode, if <*format*> is a variable, SSMA makes the replacement by using the REPLACE( ) function:

PRINT REPLACE (<format>, '%%', '%')

In Optimistic mode, SSMA does not include REPLACE even for the variable *<format>*, so the result remains unchanged: PRINT *<format>*. There is a risk that the variable contains unsupported format characters.

**Case 2. PRINT with argument list**

PRINT <format>, <arg1>, <arg2>, . . . <argN>

When <*format*> is a string literal, convert the statement to:

PRINT <new\_str>

To generate *<new\_str>*, locate all placeholders, that is, occurrences of %n! substrings in *<format>*, where *n* is an integer number from 1 to 20.

Break the format string at each placeholder and insert the argument there. If the argument is not **char** and not **varchar**, add cast to **varchar(max)** as in the following:

<left\_part\_of format> + <argn> + <right part of format>

or

<left\_part\_of format> + CAST(<argn> AS varchar(max)),'') + <right part of format>

Here *<argn>* is the converted *n*th argument from the argument list, where *n* is the number used in the placeholder.

Replace the double percent (%%) in the *<format>* literal with a single percent (%).

When <format> is a variable:

To convert the statement, create an intermediate format variable, *@print\_format\_<X>*, of varchar(max) type. *<X>* is the sequential number of PRINT in the subprogram. Adjust the percent signs by assignment:

SET @print\_format\_<X> = REPLACE (<format>, '%%', '%')

Insert the value of each argument into a format variable by using the REPLACE( ) function. If the argument type is not **varchar** and not **char**, add a conversion to **varchar(max)**.

When <format> is a string literal or a variable:

In Optimistic mode, SSMA assumes that the SQL Server option CONCAT\_NULL\_YIELDS\_NULL is OFF, and therefore does not check for NULL arguments.

In Full mode, SSMA wraps the argument with the ISNULL(…, '') function. Do not apply this if the argument is a non-NULL literal.

**Example 1: Format is a string literal**

Sybase:

PRINT “Yours are %1! %%”, @my\_percent

SQL Server 2008:

Optimistic mode:

PRINT 'Yours are '+ CAST(@my\_percent AS varchar(max)) + ' % '

Full mode:

PRINT 'Yours are '+ ISNULL(CAST(@my\_percent AS varchar(max)), '') + ' % '

Example 2: Format is a variable (Full mode)

Sybase:

PRINT @fmt, @arg1, @arg2, @arg3

SQL Server 2008:

DECLARE @print\_format\_1 varchar(max)

SET @print\_format\_1 = REPLACE (@fmt, '%% ', '% ')

SET @print\_format\_1 = REPLACE (@print\_format\_1, '%1! ', ISNULL(CAST (@arg1 AS varchar(max)), ''))

SET @print\_format\_1 = REPLACE (@print\_format\_1, '%2! ', ISNULL(CAST (@arg2 AS varchar(max)), ''))

SET @print\_format\_1 = REPLACE (@print\_format\_1, '%3! ', ISNULL(CAST (@arg3 AS varchar(max)), ''))

PRINT @print\_format\_1

### RAISERROR

**Issue**

RAISERROR has different syntax and error code numbering.

**SSMA support**

Partial

**Solution**

There can be several possible RAISERROR formats in the source code. Following are the cases covered for RAISERROR in this section.

[Case 1](#Case1): RAISERROR <num>

[Case 2](#Case2): RAISERROR *<num> <format> [, <arg\_list>]*

[Case 3](#Case3): RAISERROR <num>, <arg1>, <arg2>, . . . <arg3>

[Case 4](#Case4): RAISERROR WITH ERRORDATA <list>

**Case 1: RAISERROR <*num*>**

In this format, *<num>* is a numeric constant or a variable that represents an error number. Error numbers in the range 17000..19999 are stored in **sysmessages**, while numbers 20000 and greater are stored in a **sysusermessages** system table.

SQL Server 2008 uses numbers starting from 50001 to encode custom error messages. In our emulation, you can store such messages in **sys.messages** by running the stored procedure **sp\_addmessage**.

Whereas Sybase stores the **sysusermessage** table separately for each database, SQL Server 2008 uses a single **sys.messages** view for the entire server. So you need to differentiate the messages from different database. For this purpose SSMA provides the parameter *Base Messages Number* with a default and minimal value equal to 30001 (Figure 8).



**Figure 8**

This parameter finds appropriate user messages as described here.

Use the following approach to migrate Sybase user messages:

1. Use a Transact-SQL linked server to retrieve Sybase user messages into a Transact-SQL temporary table.
2. Use a Transact-SQL cursor to select from the temporary table to execute an **sp\_addmessage** system stored procedure using specific base-message numbers for each database.

SSMA makes the conversion in the following way:

1. If *<num>* is in the range 17000..19999, SSMA generates an error message because it does not convert system messages. The main problem is that SQL Server 2008 does not support raising system exceptions by using RAISERROR.
2. If *<num>* is a constant, SSMA changes the number to *<num> + Base Number*. *<num>* should be kept with the statement as a comment for history maintenance.
3. If *<num>* is a variable or expression, SSMA creates an intermediate local variable.

DECLARE @raiserror\_<seq\_n> int

SET @raiserror\_<seq\_n> = <num> + Base Messages Number

RAISERROR (@raiserror\_<seq\_n>, 16, 1)

*<seq\_n>* is a generated number, unique within the converted subprogram.

**Case 2: RAISERROR *<num> <format> [, <arg\_list>]***

*<format>* is a literal string or a string variable. If trimmed, *<format>* is an empty string literal, and RAISERROR logic becomes identical to Case 3, which follows this case.

*<arg\_list>* is comma-separated list of variable length, containing *<arg1>, <arg2>, … <argN>* arguments.

Like the PRINT statement, this kind of RAISERROR statement is converted differently in Full and Optimistic mode.

**Constant format**

If *<arg\_list>* is empty, SSMA converts the statement to:

RAISERROR (<new\_str>, 16, 1)

where *<new\_str>* is a converted *<format>* string.

Otherwise, an intermediate variable is created:

DECLARE @error\_format\_<X> varchar(max)

SET @error\_format\_<X> = <new\_str>

RAISERROR (@error\_format\_<X>, 16, 1)

To generate *<new\_str>*, locate all placeholders (that is, occurrences of '%n!' substrings in *<format>*), where *n* is an integer number from 1 to 20.

SSMA then breaks the *<format>* string at each placeholder and inserts the argument there. If the argument is not of **string** type, casting to **varchar(max)** is added:

<left\_part\_of format> + <argn> + <right part of format>

or

<left\_part\_of format> + CAST(<argn> AS varchar(max)),'') + <right part of format>

*<argn>* is a converted *n*th argument from the argument list, where *n* is the number used in the placeholder.

**Variable format**

If *<arg\_list>* is empty, the conversion result is:

RAISERROR (<format>, 16, 1)

Otherwise, create an intermediate variable:

DECLARE @error\_format\_<X> varchar(max)

SET @error\_format\_<X> = <ms\_format>

Here *<X>* is a sequential number of RAISERROR in the module, and *<ms\_format>* is the original *<format>*, already converted according to the SSMA string conversion algorithm.

For each argument, replacement commands are added:

SET @error\_format\_<X> = REPLACE (@error\_format\_<X>, '%1! ', ISNULL(CAST (<arg1> AS varchar(max)), '')

SET @error\_format\_<X> = REPLACE (@error\_format\_<X>, '%2! ', ISNULL(CAST (<arg2> AS varchar(max)), '')

. . .

SET @error\_format\_<X> = REPLACE (@error\_format\_<X>, '%,<N>! ', ISNULL(CAST (<argN> AS varchar(max)), '')

If the argument is non-NULL literal, it is never wrapped into the ISNULL(…,’’) function. If the argument has a **varchar** or **char** type, casting to **varchar(max)** is not applied.

Finally, the SQL Server RAISERROR statement is added:

RAISERROR (@error\_format\_<X>, 16, 1)

Because this solution does not set @@ERROR = <num>, SSMA generates a warning in the comments.

**For both constant and variable format**

In Optimistic mode, assume that the SQL Server option CONCAT\_NULL\_YIELDS\_NULL is OFF; therefore, no checks for NULL argument are made. Otherwise, the result of all string expressions will be NULL if any NULL argument is used.

In Full mode, SSMA generates ISNULL checks for arguments.

**Case 2 Examples**

**Example 1**: Constant format, optimistic mode

Sybase:

DECLARE @a1 char(10)

SET @a1 = "your SSN "

RAISERROR 315000 " %2! is not found in %1!", @a1, 666

SQL Server 2008:

DECLARE @a1 char(10)

SET @a1 = 'your SSN '

DECLARE @error\_format\_1 varchar(max)

SET @error\_format\_1 = ' ' + CAST(666 AS varchar(max)) + 'is not found in ' + @a1

RAISERROR (@error\_format\_1, 16, 1)

**Example 2**: Variable format, full mode

Sybase:

DECLARE @a1 char(10), @fmt varchar(100)

SET @a1 = "your SSN "

SET @fmt = " %2! is not found in %1!"

RAISERROR 315000 @fmt, @a1, 666

SQL Server 2008:

DECLARE @a1 char(10), @fmt varchar(100)

SET @a1 = 'your SSN'

SET @fmt = ' %2! is not found in %1! '

DECLARE @error\_format\_1 varchar(max)

SET @error\_format\_1 = REPLACE(@fmt, '%%', '%')

SET @error\_format\_1 = REPLACE (@error\_format\_1, '%1!', ISNULL(CAST(@a1 AS varchar(max)),''))

SET @error\_format\_1 = REPLACE (@error\_format\_1, '%2!', CAST(666 AS varchar(max)))

RAISERROR (@error\_format\_1,16,1)

**Case 3: RAISERROR <num>, <arg1>, <arg2>, . . . <arg3>**

Optimistic mode:

If *<num>* is in the range 17000..19999, SSMA generates an error message. System messages are not supported.

If *<num>* is a constant, SSMA changes the number to *<num> + Base Message Number*. SSMA keeps the number *<num>* with the statement as a comment for history maintenance.

If *<num>* is a variable or expression, create an intermediate local variable:

DECLARE @raiserror\_<seq\_n> int

SET @raiserror\_<seq\_n> = <num> + Base Messages Number

RAISERROR (@raiserror\_<seq\_n>, 16, 1, <arg1>, <arg2>,…<argN>)

*<seq\_n>* is a generated number, unique within the converted subprogram.

Full mode

SSMA creates intermediate variables for message text and severity, and retrieves them from the **sys.messages** system view by using the *Base Message Number* parameter.

DECLARE @error\_format\_<X> varchar(max),

 @severity\_<X> int

SELECT TOP 1 @error\_format\_<X> = sm.text, @severity\_<X> = sm.severity

 FROM sys.messages sm, sys.syslanguages sl

    WHERE (sl.langid = @@langid or sl.langid = 0)

    AND sl.msglangid = sm.language\_id

    AND sm.message\_id = <num> + Base Messages Number

 ORDER BY sl.langid DESC

Here *<X>* is a sequential number of RAISERROR in the module.

For each argument, replacement commands are added:

SET @error\_format\_<X> = REPLACE (@error\_format\_<X>, '%1! ', ISNULL(CAST (<arg1> AS varchar(max)), '')

SET @error\_format\_<X> = REPLACE (@error\_format\_<X>, '%2! ', ISNULL(CAST (<arg2> AS varchar(max)), '')

. . .

SET @error\_format\_<X> = REPLACE (@error\_format\_<X>, '%<N>! ', ISNULL(CAST (<argN> AS varchar(max)), '')

If the argument is a non-NULL literal, it is not wrapped into the ISNULL(…,'') function. If the argument has a **varchar** or char type, casting to **varchar(max)** is not applied.

Finally, a SQL Server RAISERROR statement is added:

RAISERROR (@error\_format\_<X>, @severity\_<X>, 1)

Because this solution does not set @@ERROR = <num>, SSMA generates a warning in the comments.

**Case 4: RAISERROR WITH ERRORDATA <list>**

RAISERROR WITH ERRORDATA <list> is a Sybase-specific feature (additional information for CT client), which SSMA cannot convert. To emulate this feature, you can use a SELECT result set to pass the information to the client.

### ROLLBACK

**Issue**

The ROLLBACK command can be executed without a prior BEGIN TRANSACTION statement.

**SSMA support**

Yes

**Solution**

Replace ROLLBACK … with IF (@@TRANCOUNT > 0) ROLLBACK …

### SELECT / INSERT / DELETE / UPDATE

### AT ISOLATION Clause

**Issue**

SQL Server 2008 does not support the SELECT/INSERT/DELETE/UPDATE

AT ISOLATION clause.

**SSMA support**

Yes

**Solution**

Add isolation hints to all tables that participate in the query by using the scheme in the following table.

|  |  |
| --- | --- |
| **at isolation parameter (Sybase)** | **SQL Server 2008 table hint** |
| 0 | read uncommitted | WITH (READUNCOMMITTED) |
| 1 | read committed | WITH (READCOMMITTED) |
| 2 | repeatable read | WITH (REPEATABLEREAD) |
| 3 | serializable | WITH (SERIALIZABLE) |

### DISTINCT with ORDER BY

**Issue**

SQL Server 2008 requires that all ORDER BY items appear in a SELECT list if ORDER BY is specified in a query with the DISTINCT keyword.

**SSMA support**

Yes

**Solution**

The fields missing in the SELECT list are added to the original SELECT statement. This statement is inserted as a subquery into a FROM clause of the SELECT with the original SELECT list and no DISTINCT. All WHERE and HAVING clauses of the original SELECT statement are copied to the subquery without changes.

If an item in the original SELECT list does not have an alias, generate the alias as a string that is unique within the statement. Also, create this type of alias for all ORDER BY items.

Move all SELECT items that have existing or generated aliases to the subquery. For each item, leave only the alias in the outer SELECT list. Replace ORDER BY items with this alias if an ORDER BY expression is identical to the one in the original SELECT list.

If the source ORDER BY list contains an expression or column that is not present in the SELECT list, add this item by using a generated unique alias to the subquery. Replace the item in ORDER BY with this alias.

Note that:

* Original item names are not preserved in the ORDER BY list.
* All items in the SELECT list lose their original names unless they already had aliases before conversion.

**Example**

This Sybase statement:

 select distinct name from table\_a order by id

Is converted to:

 select name

 from (select distinct name, id from table\_a) as subquery

 order by id

### FOR readonly | update Clause

**Issue**

The FOR clause is not part of SELECT query syntax in SQL Server 2008.

**SSMA support**

Partial

**Solution**

SQL Server 2008 can recognize the FOR readonly | update clause when the SELECT statement is used in the DECLARE CURSOR command. In that case, the statement is not changed. In all other cases, ignore the clause.

### Different Use of the GROUP BY Clause

**Issue**

In a SELECT list, Sybase can use nonaggregated columns not included in the GROUP BY clause.

Sybase Transact-SQL extensions to standard SQL make displaying data more flexible because they allow references to columns and expressions that are not used for creating groups or summary calculations:

* A SELECT list that includes aggregates can include additional columns that are not arguments of aggregate functions and are not included in the GROUP BY clause. An additional column affects the display of final results, because the result may contain more rows.
* The HAVING clause can include columns or expressions that are not in the SELECT list and not in the GROUP BY clause. When the Sybase Transact-SQL extensions add rows and columns to the result, or if GROUP BY is omitted, the query results can be difficult to interpret.

**SSMA support**

Partial. Queries with outer joins and unresolved identifiers, and queries with aggregates and without a GROUP BY clause are not supported.

**Solution**

Create two subqueries, one to calculate aggregates and the other to retrieve nonaggregated columns. Join the derived tables by the columns contained in the GROUP BY clause. If there are expressions in the SELECT list, calculate them in super-SELECT, because subqueries should return only regular columns and calculated aggregate functions.

To do this, follow these steps:

1. Create a subquery:
* Include a SELECT list that contains all nonaggregated columns and expressions that do not contain aggregates from the SELECT list of the original query. All GROUP BY expressions are added here.
* Use the same FROM as in the original query.
* Do not include WHERE, DISTINCT, GROUP BY, HAVING, or ORDER BY clauses.
1. Create another subquery:
* Include a SELECT list containing aggregate functions from the original query's SELECT list and all expressions from its GROUP BY clause.
* Use the same FROM as in the original query.
* Use the same WHERE as in the original query.
* Use the same GROUP BY as in the original query.
* Do not include DISTINCT, HAVING, or ORDER BY clauses.
1. Build a result query:
* Include a SELECT list that is the same as the original query's SELECT list.
* Use the FROM of the two subqueries that you created, joined by expressions from a GROUP BY clause.
* Do not use a WHERE clause, except if there is a HAVING clause (see the handling of this case in the next section).
* Use the same DISTINCT and ORDER BY clauses as in the original query.
* Assign aliases to the aggregates that are calculated by the subquery you created in step 2. Rename these generated aliases back to the original column names in the embracing query.

If multiple tables are joined in the original query, use the following rules:

* If a table does not take part in a GROUP BY, SELECT list, HAVING, or ORDER BY, do not use it in the FROM clause of the subquery you created in step 1. The same rule applies if the table columns appear in the SELECT list, HAVING, or ORDER BY, but their use is limited to aggregate functions.
* Otherwise, include the table in the FROM clauses in both subqueries.

For joined tables, move the join condition only to the subquery you created in step 2.

**In Full mode:**

The join between the two subqueries should be different from a standard query, because it allows linking by NULL columns. Therefore, check for NULLs in join expressions.

For example, if the condition is:

ON Q\_A.x = Q\_B.y

Extend it to:

ON (Q\_A.x = Q\_B.y OR (Q\_A.x IS NULL AND Q\_B.y IS NULL))

**Example 1 (Optimistic mode)**

Sybase:

 select id, subid, value, sum(value)

 from table\_c

 where value>10.00

 group by id order by id, subid

SQL Server 2008:

 select sq1.id, subid, value, sum\_value

 from

 ( select id, subid, value

 from table\_c ) as sq1

 inner join

 ( select id, sum(value) as sum\_value

 from table\_c

 where value > 10.00

 group by id ) as sq2

 on sq1.id = sq2.id or (sq1.id is null and sq2.id is null)

 order by sq1.id, subid

**Example 2**

Sybase:

 select a.name, min(a.value)

 from table\_gh a, table\_gj b

 where a.id = b.id and a.subid > 3

 group by a.subid

 having a.id>=1 and a.subid = avg(a.subid)

SQL Server 2008:

 select aname, min\_a\_value

 from

 ( select a.name as aname, a.id as aid, a.subid as asubid

 from table\_gh a

 ) as non\_group\_query

 inner join

 (

 select a.subid as asubid, min(a.value) as min\_a\_value,

 avg(a.subid) as avg\_a\_subid

 from table\_gh a, table\_gj b

 where a.id = b.id and a.subid>3

 group by a.subid

 ) as group\_query

 on non\_group\_query.asubid = group\_query.asubid

 where (non\_group\_query.aid >= 1

 and non\_group\_query.asubid = avg\_a\_subid)

### HAVING Clause

**Issue**

Sybase can use the HAVING clause without a GROUP BY, or a HAVING clause with nonaggregate columns missing in GROUP BY.

**SSMA support**

Partial. Queries with outer joins and unresolved identifiers are not converted.

**Solution**

The solution resembles the solution from the previous section ([Different Use of the GROUP BY Clause](#_Different_Use_of)). First, add regular columns from the HAVING clause to the SELECT list of the subquery. Next, calculate aggregate functions in the subquery as described in the previous section and move the HAVING clause to the outer WHERE predicate.

If HAVING is used without GROUP BY, replace INNER JOIN with CROSS JOIN and remove the joining condition, because you do not have any columns to make the join with.

**Example 1**

Sybase:

 select id, min(subid)

 from table\_gh

 having name='NameBC'

SQL Server 2008:

 select id, min\_subid

 from

 ( select id, name

 from table\_gh ) as non\_group\_query

 cross join

 ( select min(subid) as min\_subid

 from table\_gh ) as group\_query

 where name = 'NameBC'

**Example 2**

Sybase:

 select a.name, min(a.value)

 from table\_gh as a join table\_gj as b on a.id=b.id

 where a.subid>3

 group by a.subid, b.id

 having a.id>=1 and a.subid=avg(a.subid)

 order by b.value

SQL Server 2008:

 select aname, min\_a\_value

 from

 ( select a.name as aname, a.id as aid, a.subid as asubid,

 b.id as bid, b.value as bvalue

 from table\_gh as a, table\_gj as b

 ) as non\_group\_query

 inner join

 (

 select a.subid as asubid, min(a.value) as min\_a\_value,

 avg(a.subid) as avg\_a\_subid, b.id as bid

 from table\_gh as a join table\_gj as b on a.id=b.id

 where a.subid>3

 group by a.subid, b.id

 ) as group\_query

 on non\_group\_query.asubid = group\_query.asubid

 and non\_group\_query.bid = group\_query.bid

 where (non\_group\_query.aid >= 1

 and non\_group\_query.asubid = avg\_a\_subid)

 order by bvalue

### HOLDLOCK Hint

**Issue**

SQL Server 2008 does not use the HOLDLOCK hint syntax.

**SSMA support**

Yes

**Solution**

Change HOLDLOCK to WITH (HOLDLOCK).

### INDEX <index-name> Hint

**Issue**

SQL Server 2008 uses the INDEX hint differently from Sybase.

**SSMA support**

Yes

**Solution**

If *<index\_name>* is an integer number, SSMA ignores this hint. A possible source format is:

(INDEX <integer>)

The following format is not processed by SSMA:

(<integer>)

In other cases, *<index\_name>* can be a character name. SSMA can convert it to the SQL Server 2008 equivalent as follows:

WITH (INDEX (<index\_name>))

### Nonstandard Outer JOIN

**Issue**

Sybase queries can use non-ANSI outer join syntax (\*= or =\*).

**SSMA support**

Partial. Complex expressions in join conditions or join conditions that are connected with the OR operator are not supported.

**Solution**

To rewrite these joins to ANSI format, SSMA converts all non-ANSI joins to LEFT OUTER JOIN. Therefore, if you have an (=\*) condition, change the order of the tables to the opposite. The order of columns in the target ON expression does not matter.

Follow these steps of the conversion algorithm:

1. Find tables that have only left joins (all outer join conditions have an asterisk on the side of such tables). We call these *starting tables*.
2. Combine all starting tables with the CROSS JOIN operator. A CROSS JOIN is not needed if there is only one table.
3. For each starting table <A>:
4. Find table <B> that is directly left-joined to starting table <A>. Add LEFT OUTER JOIN <B> ON <*condition*>, where <*condition*> includes <A> to <B> outer links.
5. Find table <C> that is left-joined to <A> and/or <B>. Add LEFT OUTER JOIN <C> ON <*condition*>, where <*condition*> includes <A> – <C> and <B> – <C> links.
6. Continue searching until another joined table is not found. At each step, add joins that connect the new table with the tables already in the list.

Additional remarks:

* There may be several outer join conditions between two tables at once (for example, A.id1 \*= B.id2 and A.id3 \*= B.id4 and …). Move them all to the ON clause. If the join condition includes expressions, convert the expressions and use the result under ON.
* Inner join conditions are not moved.
* All filtering conditions for starting tables remain in WHERE. If any filtering condition is found for the <B>, <C>, … tables, add the condition to the ON clause where the corresponding table is introduced. Link this type of condition by using the AND operator.
* Any join condition added to the ON clause is removed from WHERE. If all expressions in WHERE are removed, delete the WHERE clause.
* Non-ANSI outer joins can be used in UPDATE or DELETE statements. The conversion algorithm is the same as for SELECT. The only exception is when the UPDATE statement does not contain the updated table in the FROM list. In this case, add it to the FROM list.
* If a column is used in a join condition without a table qualifier, locate the table by column name.
* When searching for tables that you add at each step, exclude tables that have right outer joins with the last table.
* Generate an error if two tables are both right joined and left joined. (Sybase does not allow this).
* Parallel usage of ANSI and non-ANSI constructions for a single table is not converted and leads to an error.
* The word *table* as it is used in this section may be applied to views and subqueries as well.

**Example 1**

Sybase:

SELECT \* FROM A, B WHERE A.id \*= B.id

SQL Server 2008:

SELECT \* FROM A LEFT OUTER JOIN B ON A.id = B.id

**Example 2**

Sybase:

SELECT \* FROM A, B WHERE A.id =\* B.id

SQL Server 2008:

SELECT \* FROM B LEFT OUTER JOIN A ON A.id = B.id

**Example 3**

Sybase:

SELECT \* FROM A, B, C

WHERE A.id \*= B.id

AND A.id \*= C.id

AND B.id2 \*= C.id2

SQL Server 2008:

SELECT \* FROM A

 LEFT OUTER JOIN B ON A.id = B.id

 LEFT OUTER JOIN C ON A.id = C.id AND B.id2 = C.id2

**Example 4**

*Sybase*

SELECT \* FROM A, B, C, D

 WHERE A.id =\* B.id

AND B.id \*= C.id

AND C.id2/10 \*= D.id2

SQL Server 2008:

SELECT \* FROM B

 LEFT OUTER JOIN A ON A.id = B.id

 LEFT OUTER JOIN C ON B.id = C.id

 LEFT OUTER JOIN D ON C.id2/10 = D.id2

**Example 5**

Sybase:

SELECT \* FROM A, B, C, D

 WHERE B.id \*= C.id

AND A.id2/5 \*= D.id2

AND C.id = 3

SQL Server 2008:

SELECT \* FROM A CROSS JOIN B

 LEFT OUTER JOIN D ON (A.id2/5 = D.id2)

 LEFT OUTER JOIN C ON B.id = C.id AND C.id = 3

**Example 6**

Sybase:

SELECT \* FROM A, B, C

 WHERE A.id + B.id \*= C.id

SQL Server 2008:

SELECT \* FROM A CROSS JOIN B

 LEFT OUTER JOIN C ON A.id + B.id = C.id

**Example 7**

Sybase:

UPDATE A SET x.id2 = isnull(B.id2,0) + isnull(C.id2,0)

 FROM A x, B, C

WHERE x.id \*= B.id AND x.id \*= C.id AND B.id2 \*= C.id2

SQL Server 2008:

UPDATE x SET x.id2 = isnull(B.id2,0)+isnull(C.id2,0)

 FROM A x

 LEFT OUTER JOIN B ON x.id = B.id

 LEFT OUTER JOIN C ON x.id = C.id AND B.id2 = C.id2

### NOHOLDLOCK Hint

**Issue**

SQL Server 2008 does not use NOHOLDLOCK hint syntax.

**SSMA support**

Yes

**Solution**

Change NOHOLDLOCK to WITH (NOLOCK).

### READPAST Keyword

**Issue**

SQL Server 2008 does not support READPAST.

**SSMA support**

Yes

**Solution**

Change to WITH (READPAST).

### SHARED Keyword

**Issue**

SQL Server 2008 does not support SHARED.

**SSMA support**

Yes

**Solution**

Change to WITH (TABLOCK).

### Different Behavior of the LIKE Operator in Sybase and SQL Server 2008

**Issue**

The LIKE operator in Sybase and SQL Server treats the trailing blanks differently:

Example syntax: *left\_exp* LIKE *right\_exp*

*left\_exp*: Sybase always treats *left\_exp* the same is does CHAR (that is, Sybase always right-pads it with enough spaces). In contrast, SQL Server 2008 uses *left\_exp* without changes.

*right\_exp*: Sybase replaces all trailing blanks (if any) with a single blank, while SQL Server does not modify *right\_exp*.

During comparison, both Sybase and SQL Server 2008 ignore trailing blanks in *left\_exp* if they do not match the *right\_exp* value.

Sample code 1

declare @a char(50), @b char(50)

*-- Sybase returns single row, while SQL Server doesn’t*

SET @a = ' acd'

SET @b = ' a%'

select 1 where @a like @b

*-- returns single row in both Sybase and SQL Server*

SET @a = ' ac '

SET @b = ' a%'

select 1 where @a like @b

Sample code 2

declare @a varchar(50), @b char(50)

*-- Sybase returns single row, while SQL Server doesn’t*

SET @a = 'a'

SET @b = 'a'

select 1 where @a like @b

Sample code 3

declare @a varchar(50), @b varchar(50)

*-- Sybase returns single row, while SQL Server does not*

SET @a = ' a '

SET @b = ' a% '

select 1 where @a like @b

Sample code 4

declare @a char(2),@b char(5)

*-- Sybase returns single row, while SQL Server does not*

set @a = ' a'

set @b = ' a%'

select 1 where @a like @b

**SSMA support**

Partial

**Solution**

**Optimistic mode**

Source code:

left\_exp LIKE right\_exp

Target code:

left\_exp LIKE rtrim(right\_exp)

**Full mode**

Source code:

left\_exp LIKE right\_exp

Target code:

cast(left\_exp as char(8000)) LIKE rtrim(right\_exp)

If *right\_exp* is a string literal, you do not need to use the RTRIM function in the target code. Instead, during conversion replace *right\_exp* with the right-trimmed version of *right\_exp*.

### SET ANSINULL

**Issue**

SQL Server 2008 does not support the SET ANSINULL command.

**SSMA support**

Yes

**Solution**

Replace SET ANSINULL with one of the following two commands:

SET ANSI\_NULLS

SET ANSI\_WARNINGS

### SET CHAINED

**Issue**

SQL Server 2008 does not support SET CHAINED.

**SSMA support**

Yes

**Solution**

Change SET CHAINED to SET IMPLICIT\_TRANSACTIONS.

### SET TRANSACTION ISOLATION LEVEL

**Issue**

Sybase uses transaction-level identifiers that are different from those used in SQL Server 2008.

**SSMA support**

Yes

**Solution**

Replace Sybase level numbers with Transact-SQL keywords, as in the following table.

|  |  |
| --- | --- |
| Sybase | SQL Server 2008 |
| 0 | READ UNCOMMITTED |
| 1 | READ COMMITTED |
| 2 | REPEATABLE READ |
| 3 | SERIALIZABLE |

### UPDATE Aliases

**Issue**

In an UPDATE statement, SQL Server 2008 does not allow the use of a table name if that name is duplicated in a FROM clause with an alias.

**SSMA support**

Yes

**Solution**

**Case 1**

If a FROM clause contains different tables, replace all references to the original table name in UPDATE, SET, and WHERE with the alias of the table name.

**Example**

Convert source query:

UPDATE TableA

SET a.val=b.dval

FROM TableA a, TableB b

WHERE a.id=b.id

To:

UPDATE a

SET a.val = b.dval

FROM TableA a, TableB b

WHERE a.id = b.id

**Case 2**

A FROM clause contains duplicates of the table name. If the first occurrence of the table has an alias, change the table name in UPDATE and SET to the name of this first alias.

**Example**

Convert the Sybase statement:

UPDATE TableA

SET val = ’X ’

FROM TableA a, TableA b

WHERE a.id >= 1 AND a.id <= 3 and b.id > 5

To:

UPDATE a

SET val = ‘X’

FROM TableA a, TableA b

WHERE a.id >= 1 AND a.id <= 3 and b.id > 5

### Different ROLLBACK Syntax

**Issue**

Sybase can use ROLLBACK *transaction\_name* and ROLLBACK WORK *transaction\_name* syntax that does not exist in SQL Server 2008.

**SSMA support**

Yes

**Solution**

Change all occurrences of ROLLBACK *<word>* or ROLLBACK WORK *<word>* to ROLLBACK TRANSACTION *<word>* where *<word>* is a single word that is not equal to TRAN[SACTION].

### Sybase Allows Aggregate Functions in UPDATE

**Issue**

Sybase can use aggregate functions in the SET clause of an UPDATE statement, which is invalid in SQL Server 2008.

**SSMA support**

Yes

**Solution**

SSMA constructs the converted statement according to the following algorithm:

1. Copy the UPDATE clause from the original query without changing it.
2. Add the following subqueries to the FROM clause after the comma (that is, cross-joined), each one with a generated alias:
3. The first subquery calculates aggregated values. The SELECT clause in this subquery includes the aggregate functions used in the SET clause of the original query. If a table has an alias, add this alias to each column of the table. For each aggregate function, add an alias. Move FROM and WHERE clauses from the original query. If the table specified in the UPDATE clause is missing in the FROM clause of the subquery, add it to the subquery's FROM clause with a comma delimiter (even if this table's fields do not participate in a subquery).
4. If the SET clause contains nonaggregated fields from one or more other tables that are not updated, for each of these tables add the following subquery to the target query's FROM clause:

SELECT TOP 1 List\_of\_fields\_participating\_in\_SET\_clause FROM Table

Add the subquery for the second, third, and so on occurrence of the updated table in FROM, if the nonaggregated columns from the table show in the SET clause.

1. In the SET clause, replace aggregate functions and fields from nonupdated tables with aliases from subqueries as follows:
* If a field of an updated table has an alias at the left side of a SET clause, remove this alias.
* If a field of an updated table has an alias at the right side of SET, replace the alias with the table name.
* If a field of an updated table does not have an alias at the right side of SET, add the table name prefix to it.
1. The original WHERE clause is removed from the UPDATE statement and remains only in the subquery in step 2a.

**Note**   Check this issue before checking conditions for the UPDATE aliases problem described in [UPDATE Aliases](#_UPDATE_Aliases_1) earlier in this document. If this solution is applied, there is no need to perform the transformations for UPDATE aliases.

**Example 1**

**Sybase**

UPDATE tb\_u

SET id = sum(subid)

WHERE subid>2

**SQL Server 2008 equivalent**

UPDATE tb\_u

SET id = ssma\_aggr.sum\_subid

FROM

(

SELECT sum(subid) as sum\_subid FROM tb\_u WHERE subid>2

) ssma\_aggr

**Example 2**

**Sybase**

UPDATE tb\_u

SET a.id = sum(a.subid) + a.id,

 subid = value

FROM tb\_u a

WHERE subid>2

**SQL Server 2008 equivalent**

UPDATE tb\_u

SET id = sum\_subid + id,

 subid = value

FROM

(

SELECT sum(a.subid) as sum\_subid FROM tb\_u a WHERE subid>2

) ssma\_aggr

**Example 3**

**Sybase**

UPDATE tb\_u

SET tb\_u.id = sum(tb\_u.value+tb\_j.value),

 tb\_u.subid = sum(tb\_u.value) + tb\_u.value + tb\_j.value,

 tb\_u.value=tb\_j.value

FROM tb\_u, tb\_j

WHERE tb\_u.id=tb\_j.id and tb\_u.subid>1 and tb\_j.id>=2

**SQL Server 2008 equivalent**

UPDATE tb\_u

SET tb\_u.id=sum\_tb\_u\_value\_tb\_j\_value,

 tb\_u.subid=sum\_tb\_u\_value + tb\_u.value + ssma\_tb\_j.value,

 tb\_u.value=ssma\_tb\_j.value

FROM

(

SELECT sum(tb\_u.value+tb\_j.value) as sum\_tb\_u\_value\_tb\_j\_value,

 sum(tb\_u.value) as sum\_tb\_u\_value

FROM tb\_u, tb\_j

WHERE tb\_u.id=tb\_j.id and tb\_u.subid>1 and tb\_j.id>=2

) ssma\_aggr,

( SELECT TOP 1 tb\_j.value as value FROM tb\_j ) ssma\_tb\_j

### Several Table Hints Are Used at Once

**Issue**

Sybase can use several hints with one table. If each hint is converted independently, SQL Server 2008 cannot support the result.

**SSMA support**

Yes

**Solution**

Combine several hints in a single pair of parentheses and delimit them with commas.

**Example**

**Sybase**

SELECT \* FROM QQI (INDEX QQII) READPAST

**SQL Server 2008**

SELECT \* FROM QQI WITH (INDEX (QQII), READPAST)

### ORDER BY with Table Name and Column Alias

**Issue**

In Sybase, it is possible to specify the ordering when a table name (or alias) is used together with a column alias. SQL Server 2008 does not support this syntax. For example, the following Sybase code specifies the order:

SELECT c AS c\_alias FROM tab\_a ORDER BY tab\_a.c\_alias

**SSMA support**

Yes

**Solution**

Remove the table name or alias from ORDER BY. Sybase does not allow duplication of column names within a single SELECT statement anyway.

### CHAR Column Allowing NULLs

**Issue**

If a Sybase column is of **char** type and is defined with a null specifier, it is treated as **varchar**. The column value is right-trimmed every time you retrieve it. That is not applicable to Sybase variables. This is also true for **nchar** and **unichar** column types.

**SSMA support**

None

**Solution**

This problem can be solved by one of the following two approaches:

* Map all char NULL table columns to the **varchar** data type.
* Adjust business and application logic to support this feature.

### Insertion of Default Values

**Issue**

In Sybase if all columns have defaults, the following INSERT command can add a row to the table without specifying any value:

INSERT INTO <a\_table> VALUES ( )

SQL Server 2008 does not support this syntax.

**SSMA support**

Yes

**Solution**

SSMA converts statements such as these to:

INSERT INTO <a\_table> DEFAULT VALUES

### Nested Aggregates in a SELECT List

**Issue**

Sybase allows nesting aggregate functions like this:

SELECT x, SUM(y), MAX(SUM(y)) FROM tab GROUP BY x

In this case, SUM(*y*) is the sum per each value of *x*, and MAX(SUM(*y*)) is maximum for the entire table replicated in each row of the result set.

**SSMA support**

None

**Solution**

Emulate this type of SELECT statement by using subquery or common table expressions to calculate nested aggregates.

**Example**

**Sybase**

select x, sum(y) as s,

 ms = (select max(s) from (select sum(y) as s

 from table\_a group by x) in\_query)

 from table\_a group by x

**SQL Server**

 with base\_query (x,s)

 as (select x, sum(y) as s from table\_a group by x)

 select x, s, (select max(s) from base\_query) as ms

 from base\_query

### DELETE Aliases

**Issue**

In a DELETE statement, SQL Server 2008 does not allow the use of a table name if the FROM clause contains duplicates of the table and the table's first occurrence in the FROM clause has an alias.

**SSMA support**

None

**Solution**

If the first mention of the table in the FROM clause is referred to by an alias, change the table name in the DELETE clause to the name of the alias.

**Example**

Convert the source code:

 delete dt from dt a, dt b where b.i>1 -- deleted from a

To:

 delete a from dt a, dt b where b.i>1 -- deleted from a

### Named Constraint on Temporary Table

**Issue**

SQL Server 2008 does not allow multiple PK (PRIMARY KEY) constraints with the same name, even on different tables on different user sessions.

**SSMA support**

Yes

**Solution**

When a PK (PRIMARY KEY) constraint name is defined manually, migrate its name in this format: *table\_name$pk\_name*. Also replace all references to the constraint name.

**Example**

Convert the following Sybase code:

CREATE TABLE prices\_subgroup\_daily

(fosg\_c char(32) NOT NULL,

investment\_id char(24) NOT NULL,

CONSTRAINT prices\_subgroup\_daily\_pk PRIMARY KEY NONCLUSTERED (fosg\_c, investment\_id))

To:

CREATE TABLE prices\_subgroup\_daily

(fosg\_c char(32) NOT NULL,

investment\_id char(24) NOT NULL,

CONSTRAINT prices\_subgroup\_daily$prices\_subgroup\_daily\_pk

PRIMARY KEY NONCLUSTERED (fosg\_c, investment\_id)

)

## Global Variables

This section covers specific migration problems that can arise during migration of global variables.

### @@ERROR

**Issue**

Generally, Sybase error codes differ from those of SQL Server 2008.

**SSMA support**

Partial

**Solution**

SSMA does not change anything in expressions where @@ERROR is compared with 0, such as @@ERROR = 0, @@ERROR != 0.

If @@ERROR is used in other contexts, SSMA writes the warning, "Microsoft SQL Server may use different error code." Separately investigate each occurrence of a specific number because many error codes are identical on both platforms. Some have different numbers and some Sybase errors may not exist in SQL Server 2008.

### @@PAGESIZE

**Issue**

The page size may vary in Sybase.

**SSMA support**

Yes

**Solution**

SSMA substitutes an 8192 constant in place of @@PAGESIZE, which reflects the actual page size in SQL Server 2008.

## @@SQLSTATUS

**Issue**

Sybase error codes differ from SQL Server 2008.

**SSMA support**

Yes

**Solution**

@@SQLSTATUS is equivalent to:

CASE @@FETCH\_STATUS WHEN -1 THEN 2 ELSE 0 END

In comparisons, you can avoid the CASE expression as shown in the following table.

|  |  |
| --- | --- |
| Source | Target |
| @@SQLSTATUS = 0 | @@FETCH\_STATUS = 0 |
| @@SQLSTATUS > 0 | @@FETCH\_STATUS < 0 |
| @@SQLSTATUS != 0 | @@FETCH\_STATUS < 0 |
| @@SQLSTATUS = 2 | @@FETCH\_STATUS = -1 |

### @@TRANCHAINED

**Issue**

@@TRANCHAINED is not available in SQL Server 2008.

**SSMA support**

Yes

**Solution**

Replace with (@@OPTIONS & 2).

### @@TRANSTATE

**Issue**

@@TRANSTATE is not available in SQL Server 2008.

**SSMA support**

Partial

**Solution**

SSMA can make the conversion when @@TRANSTATE is compared with 1:

* Replace (@@TRANSTATE > 1) with (@@ERROR != 0)
* Replace (@@TRANSTATE <=1) with (@@ERROR = 0)

### @@UNICHARSIZE

**Issue**

@@UNICHARSIZE is not available in SQL Server 2008.

**SSMA support**

Yes

**Solution**

Replace it with the numeric literal ‘2’.

# Data Migration

This section covers specific migration problems that can arise during data migration.

## Timestamps

**Issue**

When you migrate a Sybase **timestamp** to a SQL Server 2008 **timestamp**, the target field receives values that are generated automatically, and not the original values. If the application logic includes timestamp comparisons on different rows, it can fail after migration.

**SSMA support**

Partial

**Solution**

Map **timestamp** to **binary**(8) type for table columns. During SSMA conversion, add a new column named **SSMA\_timestamp** that has a Transact-SQL **timestamp** type. Add the default @@DBTS for the old **timestamp** column to ensure that this column receives proper timestamp values after the migration finishes.

The **SSMA\_timestamp** column is necessary to advance the @@DBTS value after the row is updated.

**Example**

Convert the Sybase table:

CREATE TABLE (ts timestamp, id int, name varchar(100))

To:

CREATE TABLE (ts binary(8) default @@DBTS, id int, name varchar(100), SSMA\_timestamp timestamp)

## Numeric with Scale > 26

**Issue**

If the source decimal or numeric column is defined with a scale greater than 26, a ‘Number too large’ error is generated and the transfer terminates.

**SSMA support**

None

**Solution**

To migrate this type of data you can use SQL Server Integration Services (SSIS) instead of SSMA data migration, or change the column's data type to a numeric data type that has a scale that is less than 26.

### Constraints and Bound Rules

**Issue**

If a constraint or bound rule is not satisfied, an error is displayed and the transfer terminates.

**SSMA support**

Partial

**Solution**

By default, SSMA data migration does not check constraints. You can change this by setting the Check Constraint property in the project settings to true.

If you migrate data via SQL Server Integration Services, you can turn constraint checks on or off by switching the BulInsertCheckConstraints property in the data flow destination component.

### Defaults vs. NULLs

**Issue**

If a table has a column or bound default, all NULLs in that column are replaced by default values.

**SSMA support**

Partial

**Solution**

By default, SSMA keeps nulls while transferring data (see the **KeepNulls** option in the project options).

## Keeping Identities

**Issue**

Identity columns are regenerated when inserting data in the SQL Server 2008 table. If a foreign key is referenced in the identity column, the original link is broken.

**SSMA support**

Partial

**Solution**

By default, SSMA data migration does not fire triggers (see the Fire Triggers property in the project settings). If you migrate data via SQL Server Integration Services, you can set the **FireTriggers** property of the OLE DB destination to false.

## Triggers

**Issue**

INSERT triggers may be executed during the transfer, which can greatly decrease performance.

**SSMA support**

Partial

**Solution**

By default, SSMA data migration does not fire triggers (see the Fire Triggers property in the project settings). If you migrate data via SSIS, you can set the **FireTriggers** property of the OLE DB destination to false.

# Other Migration Issues

## Cursor Scope

**Issue**

Sybase limits the cursor scope to the stored procedure where it is declared and to nested stored procedures. The cursor is deallocated automatically on exiting the procedure.

SQL Server 2008 supports two types of cursors—local and global. Local cursor scope is limited to the stored procedure where it is declared and the cursor is deallocated automatically on exiting the procedure. Global cursor scope is limited to the user session, and the cursor still exists after the procedure in which it was declared exits.

To emulate this behavior you can try two possible solutions:

* Declare all cursors as global so they are deallocated explicitly before each exit point of the procedure. The cursor will also be visible to nested procedures.
* Automatically deallocate the cursor by declaring it as local. While the cursor is not visible to nested procedures, this may be enough for most applications.

**SSMA support**

Yes—implemented in the second solution.

**Solution**

Try one of the following two solutions:

* At the end of the procedure but before each RETURN statement, add the following statements for each cursor that is declared in the procedure:

IF CURSOR\_STATUS(‘GLOBAL’, ‘*<cursor\_name>*’) > -3

DEALLOCATE *<cursor\_name>*

* (Preferred) Convert all Sybase cursors to Transact-SQL LOCAL cursors.

DECLARE *<cursor\_name>* CURSOR LOCAL FOR …

## Case Sensitivity

**Issue**

Sybase identifiers and object names are case sensitive. If you transfer the source to a case-insensitive target, name conflicts may arise.

**SSMA support**

None

**Solution**

The only solution for this problem is to rewrite the code using distinct objects and identifier names. You could script an entire database into a single file and then use a text editor to find and replace ambiguous names.

## Reserved Keywords

**Issue**

A Sybase object or table column may have a name that is identical to SQL Server 2008 reserved keywords.

**SSMA support**

None

**Solution**

Enclose every occurrence of the identifier in Transact-SQL code in square brackets, such as:

[FUNCTION]

## Syb\_identity Pseudocolumn

**Issue**

To reference the IDENTITY column, Sybase can use the syb\_identity keyword.

**SSMA support**

Yes

**Solution**

IDENTITYCOL is the proper SQL Server 2008 equivalent.

## Different Syntax of IDENTITY( ) Function

**Issue**

Sybase syntax is identity(*precision*), but Transact-SQL uses identity *(data\_type [,seed, increment])*.

**SSMA support**

Yes

**Solution**

Replace Sybase identity(*precision*) with identity*(numeric(precision))*

## Login Triggers

**Issue**

By using the sp\_modifylogin system procedure, in Sybase you can use login triggers, which are procedures that are executed every time a user logs in.

**SSMA support**

None

**Solution**

Handle this problem on the application level if the application provides a call to the trigger code before any other activity on the database server.

## Cross-Database Foreign Key

**Issue**

Sybase allows the creation of foreign key references to other databases, which SQL Server 2008 cannot do.

For example, this statement is valid in Sybase, but it fails in SQL Server 2008:

ALTER TABLE dbo.MyTable ADD CONSTRAINT fk\_MyTable\_Id

FOREIGN KEY (parent\_id)

REFERENCES Outerbase.dbo.parentlist (id)

**SSMA support**

None

**Solution**

Change the schema mapping so that the referenced tables go to the same target database. If that is not allowed, resolve the problem by using the INSTEAD OF trigger on the master table to ensure that no subordinate records exist in subordinate tables. Use a check constraint on slave tables to ensure that there is an appropriate record in the master table.

**Example:**

Suppose there are two tables: a master (named **header** and stored in database **A**) and a subordinate (named **detail** and stored in database B). The following code creates the tables:

CREATE TABLE header(id int IDENTITY PRIMARY KEY, name varchar(25))

CREATE TABLE detail(parent\_id int NOT NULL, detail\_name varchar(25))

The **Parent\_id** field of the **detail** table refers to the **id** field of the **header** table:

Alter table detail, add a constraint called fk\_main, which is defined as:

FOREIGN KEY (parent\_id)

REFERENCES A.dbo.header(id)

To implement this functionality, first create an INSTEAD OF trigger:

CREATE TRIGGER iodHeader

ON dbo.header INSTEAD OF DELETE

AS BEGIN

 SET NOCOUNT ON

IF EXISTS(SELECT \* FROM deleted a join dbo.detail b on a.id = b.parent\_id) begin

 RAISERROR('Cannot delete from "master" table - corellated records in "detail" ',16,-1) WITH SETERROR

 return

 END

 DELETE h

 FROM dbo.header h join deleted d on h.id = d.id

 RETURN

END

Next, create a function that checks the **detail** table:

CREATE FUNCTION dbo.ckdetail(@parent\_id int)

RETURNS int

AS BEGIN

 DECLARE @ret int set @ret = 0

 SELECT TOP 1 @ret = 1 from dbo.header WHERE id = @parent\_id

 RETURN @ret

END

Lastly, add a check constraint to the **detail** table:

ALTER TABLE dbo.detail ADD CONSTRAINT

CK\_detail CHECK (dbo.ckDetail(parent\_id)=1)

As a result:

* The INSTEAD OF trigger prevents you from deleting records in the **header** table if dependent records in the **detail** table exist.
* The check constraint **ck\_detail** prevents you from storing (via an INSERT or UPDATE statement) incorrect (nonexisting) values into the **parent\_id** field.
* The scalar function **ckDetail** makes check constraints possible. (In check constraints you can use only simple expressions—subqueries are not allowed.)

**Note**   This solution doesn’t take into account cases when the primary key on the master table can be modified. Because of the behavior of calling ROLLBACK inside of a trigger in SQL Server, this is not a common case solution. Assume that you have the following code:

delete header where id = 47

select @@error

If the DELETE statement in your trigger fails, all modifications made by the trigger are rolled back and the batch is terminated. Therefore, the next statement (select @@error) will never execute.

For more information, see [CREATE TRIGGER](http://technet.microsoft.com/en-us/library/ms189799.aspx) (http://technet.microsoft.com/en-us/library/ms189799.aspx) and [Rollbacks and Commits in Stored Procedures and Triggers](http://technet.microsoft.com/en-us/library/ms187844.aspx) (http://technet.microsoft.com/en-us/library/ms187844.aspx) in SQL Server 2008 Books Online.

## Deprecated Equivalents

**Issue**

SQL Server 2008 fully supports Sybase object categories such as DEFAULTs and RULEs. However, SQL Server 2008 Books Online describes these categories as deprecated and not recommended for use in any new development.

**SSMA support**

None

**Solution**

During conversion SSMA will:

* Replace RULE with the appropriate CHECK constraint.
* Ignore RULE when applied to user data types.
* Convert DEFAULTS as the column default.

## Different Scope of Constraint Names

**Issue**

Named constraints in different Sybase tables can use the same name. In SQL Server 2008, constraint names must be unique in the database.

**SSMA support**

Yes

**Solution**

Generate SQL Server 2008 constraint names by concatenating the table name and the Sybase constraint name.

**Note**   If you use hard-coded constraint names (for example, a primary key name) in your application, SSMA will not handle this automatically. Manually replace old constraint names in your code with new ones.

## Dynamic SQL

**Issue**

The EXECUTE statement in both Sybase and SQL Server 2008 is identical in syntax and semantic, but the dynamic SQL string may need conversion.

**SSMA support**

No

**Solution**

SSMA adds a warning that reads, "Dynamic SQL string was not converted" each time the converter encounters an EXECUTE statement. Usually, you must manually rewrite the code.

But in some (not so rare) cases, you can copy the source code to the Statements node in the SSMA Sybase Metadata Explorer, and then convert it. (For detailed instructions about how to work with the Statements node, see the SSMA Help.)

Note that using dynamic SQL is not good practice; it can hurt SQL Server performance because of query recompilation and optimization, and it can lead to security issues. Try, therefore, to avoid using dynamic SQL.

## Proxy Tables

**Issue**

Sybase proxy tables look like normal tables, but their content is synchronized with a date on a remote server.

**SSMA support**

None

**Solution**

A solution requires information about the remote server—including plans to migrate objects on that platform. Generally, you should create a linked server at the local server that is running SQL Server and that points to the remote database. Proxy tables are emulated as views on OPENQUERY( ), or as four-part object identifiers. In both cases, create synonyms on the converted objects so that the original code is not changed.

Note that four-part identifiers may not work on Oracle servers.

## Variables in Cursor Declaration

**Issue**

In SQL Server 2008, cursor variable values do not change after a cursor is declared. In Sybase (as in SQL Server version 6.5 and earlier), variable values are refreshed every time a cursor is reopened.

**SSMA support**

None

**Solution**

Move the cursor declaration statement to directly before the cursor open statement, and then add the DEALLOCATE CURSOR statement before the iterative cursor declaration statement.

## Different Behavior of the MIN and MAX Functions with Character Columns in Sybase and SQL Server

**Issue**

The Sybase min and max functions implicitly convert the **char** data type to **varchar** and the **unichar** data type to **univarchar**, stripping all trailing blanks.

**SSMA support**

None

**Solution**

Wrap the expression containing the min or max function in the RTRIM function if the expression in the target code has a **char** or **nchar** data type.

## Potential Challenges

There are features of Sybase ASE that cannot be automatically handled by SSMA 4.0 and require manual intervention. Those features include:

* Dynamic SQL
* Incompatible system tables and procedures
* Proxy tables
* User messages stored in the **sysusermessages** table
* User-defined functions that appeared in Sybase ASE 15.0.2
* Java data types
* Identifiers that differ only by letter case in a case-sensitive Sybase database
* Some cases of temporary tables whose definition is out of scope of the stored procedure being converted

Here are our suggestions about handling some of these features.

*Dynamic SQL* is problematic because it is not generally possible to see the real text of a dynamic statement at conversion time. The statement gets its final form only when the generated code is executed. Still, you can use the Statement window in SSMA to convert ad hoc SQL statements, including dynamic SQL. Try the same approach to convert SQL strings embedded in the user's application code by using the following steps:

* Cut the statement from the application (or reconstruct it if the statement is built according to an algorithm).
* Paste it into the Statement window.
* Execute **Convert Schema**.
* Paste the result back into the proper place in the application code.

The declaration syntax of Sybase *user-defined functions* (UDFs) is very similar to the syntax of SQL Server functions, but their conversion is not implemented in SSMA yet. We recommend the following steps for migrating these functions.

1. Start SSMA 2008 for Sybase.
2. Establish connections to the Sybase ASE and SQL Server databases.
3. Open Sybase Central and copy the SQL script of the user-defined function to be converted.
4. In SSMA Sybase Metadata Explorer, expand the ASE server node, and then expand database in which the UDF is stored. Then expand the schema to which the UDF belongs, and then select the Statements category.
5. Right click the Statements category, and then click **Add Statement**. To the right of the Metadata Explorer an area to enter an SQL statement appears.
6. Paste the copied Sybase user-defined function into this area. SSMA will ask to save metadata. Click **Yes**. The entered statement is displayed in the Metadata Explorer as Statement # *the-number-of-statement*.
7. Right click the Statement #... object in the Statements category, and then click **Convert** **Schema**. The converted script appears in the SQL Server Metadata Explorer below.
8. Right click the converted Statement #... object in the SQL Server Metadata Explorer, and then click **Save as Script**.
9. Edit all the saved scripts as follows by putting the user-defined function parameters in parentheses. Save the changes.
10. If the function script contains the WITH RECOMPILE statement, delete this statement from the script.
11. Execute the saved script in SQL Server Management Studio.

Sybase *Java data types* can be migrated by converting the Java code to C# and creating the corresponding SQL CLR type based on the converted code. For more information, see [CLR User-Defined Types](http://technet.microsoft.com/en-us/library/ms131120.aspx) (http://technet.microsoft.com/en-us/library/ms131120.aspx) in SQL Server Books Online.

Duplicated identifiers can result when you move a *case-sensitive Sybase source* to a case-insensitive SQL Server object. To resolve this issue, we recommend that the target server collation be case-sensitive.

*Temporary tables* in stored procedures can create problems when their definitions are absent in the module you are converting. Generally, you should convert the module where a temporary table is created before converting the modules from which the temporary table is referenced. That way, SSMA is able to remember the table definition and to use it properly.

# Conclusion

This migration guide covers the differences between Sybase ASE and SQL Server 2008 database platforms, and it describes the steps necessary to convert a Sybase database to SQL Server. It explains the algorithms that SSMA Sybase uses to perform this conversion so that you can better understand the processes that are executed when you run the SSMA Convert Schema and Migrate Data commands. For those cases when SSMA does not handle a particular migration issue, approaches to manual conversion are included.

## About DB Best Technologies

DB Best Technologies is a leading provider of database and application migration services and custom software development. We have been focused on heterogeneous database environments (SQL Server, Oracle, Sybase, DB2, MySQL) since starting at 2002 in Silicon Valley. Today, with over 75 employees in the United States and Europe, we develop database tools and provide services to customers worldwide.

DB Best developed migration tools to automate conversion between SQL dialects. In 2005 Microsoft acquired this technology, which later became a family of SQL Server Migration Assistant (SSMA) products. We continue to develop new versions of SSMA, and support Microsoft customers who are migrating to SQL Server.

We also provide migration services covering all major steps of a typical migration project: complexity assessment, schema conversion, data migration, application conversion, testing, integration, deployment, performance tuning, training, and support.

For more details, visit us at <http://www.dbbest.com>, e-mail us at info@dbbest.com, or call 1-408-202-4567.

**For more information:**

<http://www.microsoft.com/sqlserver/>: SQL Server Web site

<http://technet.microsoft.com/en-us/sqlserver/>: SQL Server TechCenter

<http://msdn.microsoft.com/en-us/sqlserver/>: SQL Server DevCenter

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

* Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
* Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

Send feedback.