

# Guide to Migrating from MySQL to SQL Server 2005

SQL Server Technical Article

Writers: Alexander Pavlov, Yuri Rusakov

Technical Reviewers: Irena Balin, Dmitry Balin

Published: March 2008

Applies To: SQL Server 2005 Service Pack 2

Summary: In this migration guide you will learn the differences between the MySQL and SQL Server 2005 database platforms, and the steps necessary to convert a MySQL database to SQL Server.

### Copyright

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.

 2008 Microsoft Corporation. All rights reserved.

Microsoft, SQL Server, and Visual C++are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

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

### Table of Contents

[Introduction 4](#_Toc193451386)

[MySQL to SQL Server 2005 Migration 4](#_Toc193451387)

[Main Migration Steps 4](#_Toc193451388)

[Converting Database Objects 4](#_Toc193451389)

[Migrating MySQL Data Types 5](#_Toc193451390)

[Type Mapping 6](#_Toc193451391)

[Data Type Migration Issues 9](#_Toc193451392)

[Numeric data types 9](#_Toc193451393)

[Date and time types 12](#_Toc193451394)

[String types 15](#_Toc193451395)

[ENUM and SET data types 17](#_Toc193451396)

[Other types 20](#_Toc193451397)

[Implicit data type conversion 21](#_Toc193451398)

[Data type default values 22](#_Toc193451399)

[MySQL Migration Issues 23](#_Toc193451400)

[Operators 23](#_Toc193451401)

[Comparison operators 23](#_Toc193451402)

[Bit operators 25](#_Toc193451403)

[Assignment operators 26](#_Toc193451404)

[Variables 27](#_Toc193451405)

[Utility Statements 29](#_Toc193451406)

[Data Definition Statements 30](#_Toc193451407)

[IF NOT EXISTS, IF EXISTS, OR REPLACE clauses 30](#_Toc193451408)

[Temporary tables 36](#_Toc193451409)

[SCHEMA keyword in DATABASES statements 40](#_Toc193451410)

[CHARACTER SET and COLLATE clauses in DDL statements 40](#_Toc193451411)

[CREATE INDEX statement 41](#_Toc193451412)

[CREATE TABLE statement 43](#_Toc193451413)

[ALTER TABLE statement 49](#_Toc193451414)

[RENAME DATABASE statement 54](#_Toc193451415)

[RENAME TABLE statement 55](#_Toc193451416)

[CREATE VIEW, ALTER VIEW, DROP VIEW statements 56](#_Toc193451417)

[CREATE EVENT, ALTER EVENT, DROP EVENT statements 58](#_Toc193451418)

[CREATE, ALTER, DROP PROCEDURE/FUNCTION statements 58](#_Toc193451419)

[Data Manipulation Statements 63](#_Toc193451420)

[LIMIT clause 63](#_Toc193451421)

[DELETE statement 65](#_Toc193451422)

[UPDATE statement 66](#_Toc193451423)

[INSERT statement 67](#_Toc193451424)

[REPLACE statement 70](#_Toc193451425)

[SELECT statement 71](#_Toc193451426)

[SELECT…INTO and LOAD DATA INFILE statements 72](#_Toc193451427)

[GROUP BY, HAVING, and ORDER BY clauses 73](#_Toc193451428)

[JOINs 76](#_Toc193451429)

[Subqueries 79](#_Toc193451430)

[Pepared statements 80](#_Toc193451431)

[DO command 80](#_Toc193451432)

[HANDLERs 81](#_Toc193451433)

[MODIFIERs 82](#_Toc193451434)

[Transactional and Locking Statements 82](#_Toc193451435)

[BEGIN TRANSACTION statements 82](#_Toc193451436)

[END TRANSACTION statements 83](#_Toc193451437)

[Named transaction SAVEPOINT statements 84](#_Toc193451438)

[SET AUTOCOMMIT statements 85](#_Toc193451439)

[LOCK TABLES and UNLOCK TABLES statements 87](#_Toc193451440)

[SET TRANSACTION ISOLATION LEVEL statement 87](#_Toc193451441)

[XA transaction statements 87](#_Toc193451442)

[Database Administration Statements 88](#_Toc193451443)

[Account management statements 88](#_Toc193451444)

[Table maintenance statements 88](#_Toc193451445)

[SET statement 88](#_Toc193451446)

[SHOW statement 90](#_Toc193451447)

[Other administrative statements 91](#_Toc193451448)

[Stored Procedures and Functions (Routines) 91](#_Toc193451449)

[CALL statements 91](#_Toc193451450)

[Compound statements block 92](#_Toc193451451)

[Local variables 93](#_Toc193451452)

[Conditions and handlers 96](#_Toc193451453)

[Cursors 97](#_Toc193451454)

[Flow control constructs 99](#_Toc193451455)

[Routines 102](#_Toc193451456)

[Triggers 103](#_Toc193451457)

[SQL Mode (SQL\_MODE system variable) 106](#_Toc193451458)

[Migrating MySQL System Functions 107](#_Toc193451459)

[Equivalent Functions 107](#_Toc193451460)

[Nonsupported Functions 107](#_Toc193451461)

[Emulated Functions 107](#_Toc193451462)

[Conclusion 114](#_Toc193451463)

#### Introduction

This migration guide outlines the procedures, problems, and solutions for migrating from MySQL 5 to Microsoft® SQL Server™ 2005.

Inside you will find three main sections:

[Migrating MySQL Data Types](#DataTypes). Explains the data type mapping and adds remarks about the related conversion issues.

[MySQL Migration Issues](#_MySQL_Migration_Issues). Explores the challenges you might encounter when migrating from MySQL to SQL Server 2005 and offers possible solutions.

[Migrating MySQL System Functions](#_MySQL_System_Functions). Examines MySQL system function references, divided into equivalent functions, nonsupported functions, and emulated functions.

#### MySQL to SQL Server 2005 Migration

Following are the basic, high-level steps for migrating a MySQL database to SQL Server 2005 and what you must know about converting database objects.

##### Main Migration Steps

To migrate a MySQL database

1. Decide how you will map MySQL databases to SQL Server 2005. You have two main options:
* Map each MySQL database to a separate SQL Server database. For example, you could map the MyDB MySQL database to MyDB SQL Server database.
* Map each MySQL database to a single SQL Server database but a separate schema. For example, you could map the MyDB MySQL database to MySQLDatabases SQL Server database, schema MyDB.

In SQL Server, schemas are not necessarily linked to a specific user or a login, and one server contains multiple databases.

1. Convert database objects; these are tables, tables constraints, indexes, view, procedures, functions, and triggers.
2. Map data types from the MySQL data type to a SQL Server data type.
3. Rewrite your views, procedures, and functions according to SQL Server syntax.
4. Change your applications as necessary so that they can connect and work with SQL Server 2005.

After a successful database conversion, migrate your data from the old MySQL database to the newly created SQL Server 2005 database. For this task you could use SQL Server Integration Service (SSIS), for example.

##### Converting Database Objects

This section contains considerations that you must know when converting database objects.

Schema object names

In SQL Server 2005, an object name can be up to 128 chars long.

Non-quoted identifier names must follow these rules:

* First character must be alphanumeric, underscore (\_), “at” sign (@), or number sign (#)
* Subsequent characters can include alphanumeric characters, underscore, at (@) sign, number sign, dollar sign
* Identifier must not be a Transact-SQL reserved word
* Embedded spaces or special characters are not allowed

Identifiers that start with @or a number sign have special meanings. Identifiers starting with @ are local variable names. Those that start with a number sign are temporary table names.

To quote an identifier name in Transact‑SQL, you must use square brackets ([]).

Tables, constraints, indexes, and views

Convert tables by using column data type mapping (see [Type Mapping](#_Type_Mapping) later in this guide).

SQL Server 2005 supports the following table (column) constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK. Convert each type of constraint according to Transact‑SQL syntax.

SELECT statements with VIEW should also be converted according to Transact‑SQL SELECT syntax.

Stored procedures and user defined functions

Convert stored procedures and functions by using Transact‑SQL syntax.

SQL Server 2005 does not support DML statements in user-defined functions. This means that you cannot change any data from within the function.

Triggers

SQL Server 2005 does not have BEFORE triggers.

Convert multiple BEFORE triggers to a single INSTEAD OF trigger.

#### Migrating MySQL Data Types

This section explains mappings and differences between MySQL and SQL Server 2005 data types, specific data type handling, and provides solutions for problems related to data types.

##### Type Mapping

Following are the recommended type mappings for converting table columns, subroutine arguments, returned values, and local variable data types.

|  |  |  |  |
| --- | --- | --- | --- |
| MySQL type | SQL Server 2005 mapping | Conversion remarks | Possible mappings |
| BIT (N) | varbinary (8) | Binary value has N bits. N = 1..64 |  |
| TINYINT (M)BOOL, BOOLEAN = TINYINT (1) | tinyint | M is the number of decimal places in the output for this value. | tinyint, smallint, int, bigint, numeric(p,s), decimal(p,s), float(p), double precision, real, smallmoney, money |
| SMALLINT (M) | smallint | M is the number of decimal places in the output for this value. | tinyint, smallint, int, bigint, numeric(p,s), decimal(p,s), float(p), double precision, real, smallmoney, money |
| MEDIUMINT (M) | int | M is the number of decimal places in the output for this value. | tinyint, smallint, int, bigint, numeric(p,s), decimal(p,s), float(p), double precision, real, smallmoney, money |
| INT (M)INTEGER (M) | int | M is the number of decimal places in the output for this value. | tinyint, smallint, int, bigint, numeric(p,s), decimal(p,s), float(p), double precision, real, smallmoney, money |
| BIGINT (M) | bigint | M is the number of decimal places in the output for this value. | tinyint, smallint, int, bigint, numeric(p,s), decimal(p,s), float(p), double precision, real, smallmoney, money |
| FLOAT (P) | float (P) |  | numeric(p,s), decimal(p,s), float(p), double precision, real, smallmoney, money |
| FLOAT [(P, S)] | float (24) | MySQL allows a non-standard syntax: FLOAT(P,S) or REAL(P,S) or DOUBLE PRECISION(P,S). Here, “(P,S)” means that values are displayed with up to P digits total, of which S digits may be after the decimal point. MySQL performs rounding when storing values.If M and D are omitted, values are stored up to the size limits allowed by the hardware. | numeric(p,s), decimal(p,s), float(p), double precision, real, smallmoney, money |
| DOUBLE [(P, S)]DOUBLE PRECISION [(P, S)]REAL [(P, S)] | float (53) | numeric(p,s), decimal(p,s), float(p), double precision, real, smallmoney, money |
| DECIMAL [(P [, S])]DEC [(P [, S])]NUMERIC [(P [, S])]FIXED [(P [, S])] | decimal [(P [, S])]numeric [(P [, S])] | Decimal types can have up to 65 digits. For a decimal with a precision of more than 38, use float or double data type. | numeric(p,s), decimal(p,s), float(p), double precision, real, smallmoney, money |
| DATETIME | datetime | MySQL can store dates from 0000-00-00 to 9999-12-31. MySQL can store zero-value of year, month and year. | smalldatetime, datetime |
| DATE | datetime | smalldatetime, datetime |
| TIME | datetime | Range is '-838:59:59' to '838:59:59'. | smalldatetime, datetime, varchar, nvarchar |
| TIMESTAMP | smalldatetime | Range is '1970-01-01 00:00:00' to partway through the year 2037. If not defined during conversion, this type gets the current datetime value. | datetime, rowversion, timestamp, varbinary(8), binary(8)  |
| YEAR [(2| 4)] | smallint | In four-digit format, allowable values are 1901 to 2155, and 0000. In two-digit format, allowable values are 70 to 69, representing years from 1970 to 2069. | datetime, varchar(4) |
| [NATIONAL] CHAR (N) | nchar (N)nchar | Range of N is 0 to 255 characters. | char, varchar, nchar, nvarchar |
| [NATIONAL] CHAR |
| [NATIONAL] VARCHAR (N)CHARACTER VARYING (N) | nvarchar (N | max) | Range of N is 0 to 65,535.If N<=8000 then nvarchar(N) else nvarchar(max). | char, varchar, nchar, nvarchar |
| TINYTEXT | nvarchar (255) |  | char, varchar, nchar, nvarchar |
| TEXT (N) | nvarchar (N | max) | A TEXT column with a maximum length of 65,535 characters.If N<=8000 then nvarchar(N) else nvarchar(max). | char, varchar, nchar, nvarchar, varchar(max), nvarchar(max) |
| MEDIUMTEXT | nvarchar (max) |  | char, varchar, nchar, nvarchar, varchar(max), nvarchar(max) |
| LONGTEXT | nvarchar (max) |  | char, varchar, nchar, nvarchar, varchar(max), nvarchar(max) |
| BINARY (N) | binary (N) |  | binary, varbinary, char, varchar, nchar, nvarchar |
| VARBINARY (N) | varbinary (N) |  | binary, varbinary, char, varchar, nchar, nvarchar |
| TINYBLOB | varbinary (255) |  | binary, varbinary, varbinary(max) |
| BLOB (N) | varbinary (N | max) | A BLOB column with a maximum length of 65,535 bytes.If N<=8000 then nvarchar(N) else nvarchar(max). | binary, varbinary, varbinary(max) |
| MEDIUMBLOB | varbinary (max) |  | binary, varbinary, varbinary(max) |
| LONGBLOB | varbinary (max) |  | binary, varbinary, varbinary(max) |
| ENUM |  | See [ENUM and SET Data Types](#_ENUM_and_SET) in this guide. |  |
| SET |  | See [ENUM and SET Data Types](#_ENUM_and_SET) in this guide. |  |

**Note**: MySQL numeric types can have an UNSIGNED flag. These should be converted to the bigger numeric type.

##### Data Type Migration Issues

This section describes data type conversion issues. Each issue is caused by a MySQL feature that is not supported in SQL Server.

###### Numeric data types

Issue: Unsigned data types

All integer types in MySQL (TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT) can have the UNSIGNED optional attribute. Unsigned values can be used to allow only non-negative numbers in a column when you need a large upper numeric range for the column.

Unsigned values can also be used to allow only non-negative values in a column with floating-point (FLOAT, DOUBLE) and fixed-point (DECIMAL) data types.

**Example:**

create table numeric\_unsigned (t tinyint unsigned, s smallint unsigned,

m mediumint unsigned, i int unsigned, b bigint unsigned);

insert numeric\_unsigned values

(255, 65535, 16777215, 4294967295, 18446744073709551615);

create table point\_unsigned (f float unsigned, d double unsigned);

insert point\_unsigned values (-1.1234567890,-1.12345678901234567890);

insert point\_unsigned values ( 5.1234567890, 5.12345678901234567890);

select \* from point\_unsigned;

-- 0 0

-- 5.12346 5.12345678901235

Solution:

To avoid negative values, use CHECK constraints. This is the simplest method but it has one disadvantage—you get an exception if you try to assign an invalid value.

Another way to avoid negative values is to use an INSERT or UPDATE trigger. This method allows the correction of invalid values before storing them in database.

Issue: Operations with unsigned values

When you use subtraction between integer values where one is of type UNSIGNED, the result is unsigned.

Example:

create table unsing (a int unsigned, b int);

insert unsing values (1,2),(4,3),(10,100);

select a-b from unsing;

-- 18446744073709551615

-- 1

-- 18446744073709551526

Solution:

Use the CASE function to calculate the result of an operation that uses unsigned values.

Issue: Display width of integer values and ZEROFILL attribute

MySQL supports specifying the display width of an integer (TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT) value in parentheses following the base keyword for the type (for example, INT(4)). This optional display width specification is used to left-pad the display of values having a width that is less than the width specified for the column.

The display width does not constrain either the range of values that can be stored in the column or the number of digits that are displayed for values having a width that exceeds that specified for the column.

**See also:** [ZEROFILL attribute](#ZEROFILL)

Solution:

Ignore these attributes during the conversion. Format the output data by using functions such as STR and CONVERT.

Issue: ZEROFILL attribute

When used in conjunction with the optional extension attribute ZEROFILL, the default padding of spaces is replaced with zeros in MySQL.

If you specify ZEROFILL for a numeric column, MySQL automatically adds the UNSIGNED attribute to the column.

Example:

create table table\_zerofill (a int(2) zerofill, b int(4) zerofill,

c int(8) zerofill, d decimal(5,2) zerofill);

insert table\_zerofill values (2,4,8,1.23);

select concat('BEGIN',a,b,c,d,'END') from table\_zerofill

-- BEGIN02000400000008001.23END

create table point\_zerofill (f float zerofill, d double zerofill);

insert point\_zerofill values (-1.1234567890,-1.12345678901234567890);

insert point\_zerofill values ( 5.1234567890, 5.12345678901234567890);

select \* from point\_zerofill;

-- 000000000000 0000000000000000000000

-- 000005.12346 0000005.12345678901235

Solution:

Ignore these attributes during the conversion. Format the output data by using functions such as STR, REPLICATE, and REPLACE.

Issue: FLOAT and DOUBLE data type precision and scale

In MySQL, FLOAT and DOUBLE data types can have precision and scale.

Example:

create table table\_float (f2 float|double(10,2),

 f5 float|double(10,5), f7 float|double(10,7));

insert into table\_float values (1.1234567,1.1234567,1.1234567);

insert into table\_float values (12345.1234567,12345.1234567,12345.1234567);

select \* from table\_float;

-- 1.12 1.12346 1.1234567

-- 12345.12 12345.12305 1000.0000000

Solution:

FLOAT and DOUBLE values with precision and scale can be rounded in triggers or in DML statements by using the ROUND function.

Issue: Maximum number of digits for DECIMAL data type

In MySQL, the maximum number of digits for the DECIMAL data type is 65.

Example:

create table table\_decimal (d decimal(65), m decimal(65,30));

insert table\_decimal values

(1234567890123456789012345678901234567890123456789012345678901234567890,

 1234567890123456789012345678901234567890123456789012345678901234567890);

select \* from table\_decimal;

-- 99999999999999999999999999999999999999999999999999999999999999999

-- 99999999999999999999999999999999999.999999999999999999999999999999

Solution:

For decimals with a precision more than 38, use the **float** or **double** data types.

###### Date and time types

Issue: "Zero" values

MySQL allows you to store '0000-00-00' as a “dummy date” if you are not using the NO\_ZERO\_DATE SQL mode.

Illegal DATETIME, DATE, YEAR, or TIMESTAMP values are converted to the “zero” value of the appropriate type ('0000-00-00 00:00:00', '0000-00-00' or '0000').

For date and time types other than TIMESTAMP in MySQL, the default is the appropriate zero value for the type. For the first TIMESTAMP column in a table, the default value is the current date and time.

Example:

create table date\_zero (dt datetime not null, d date not null,

 t time not null, y year not null, ts timestamp not null)

insert date\_zero values ();

select \* from date\_zero;

-- 0000-00-00 00:00:00 | 0000-00-00 | 00:00:00 | 0000 | 2006-12-19 18:24:49

insert date\_zero values

('20060229150000','20060229','900:15:20','0321','19000101140000');

select \* from date\_zero;

-- 0000-00-00 00:00:00 | 0000-00-00 | 838:59:59 | 0000 | 0000-00-00 00:00:00

Solution:

Replace zero date values with "1753 January 01" date.

Another method is to use a string or number data type to store zero dates.

Issue: Zeros in year, day, or month

In MySQL, you can store dates where the year, day, or month is zero in a DATE or DATETIME column.

Example:

create table date\_zeropart (d datetime null);

insert date\_zeropart values

('00001215'),('20060015'),('20061200'),('20060000'),('20060229');

-- 0000-12-15 00:00:00

-- 2006-00-15 00:00:00

-- 2006-12-00 00:00:00

-- 2006-00-00 00:00:00

-- 0000-00-00 00:00:00

Solution:

Use string or number data types to store these values.

Issue: Invalid dates

MySQL accepts invalid dates in ALLOW\_INVALID\_DATES SQL mode.

In ALLOW\_INVALID\_DATES mode, MySQL verifies only that the month is in the range from 0 to 12 and that the day is in the range from 0 to 31.

Example:

create table date\_inval (d datetime null);

set sql\_mode='ALLOW\_INVALID\_DATES';

insert date\_inval values ('20061131');

insert date\_inval values ('20061132');

set sql\_mode='';

insert date\_inval values ('20061131');

select \* from date\_inval;

-- 2006-11-31 00:00:00

-- 0000-00-00 00:00:00

-- 0000-00-00 00:00:00

Solution:

Use string or number data types to store these values.

Issue: Supported range of the DATETIME data type

The supported range of the MySQL DATETIME data type is '0000-00-00 00:00:00' to '9999-12-31 23:59:59'.

Example:

create table datetime\_range (d datetime);

insert datetime\_range values ('0000-00-00 00:00:01');

insert datetime\_range values ('0000-02-28 23:00:01');

insert datetime\_range values ('0170-04-30 08:05:01');

insert datetime\_range values ('9999-12-31 23:59:59');

Solution:

Use string or number data types to store these values.

Issue: MySQL YEAR, DATE, TIME data types

MySQL supports the YEAR, DATE, and TIME data types, which are not present in SQL Server.

The range of the YEAR data type is 1901 to 2155. Illegal YEAR values are converted to 0000.

The range of the DATE data type is '0000-00-00' to '9999-12-31'. Illegal DATE values are converted to '0000-00-00'.

TIME values may range from '-838:59:59' to '838:59:59'. By default, values that lie outside the TIME range but are otherwise legal are clipped to the closest endpoint of the range. For example, '-850:00:00' and '850:00:00' are converted to '-838:59:59' and '838:59:59'. Illegal TIME values are converted to '00:00:00'.

Example:

create table time\_range (t time);

insert time\_range values ('2 01:30:54'); -- 49:30:54

insert time\_range values ('201:03:45'); -- 201:03:45

insert time\_range values ('900:42:14'); -- 838:59:59

insert time\_range values ('-900:42:14'); -- -838:59:59

insert time\_range values ('-1 05:15:20'); -- -29:15:20

create table year2 (y year(2));

insert year2 values (20),(1920),(80),(2080);

select \* from year2; -- 20 20 80 80

create table year4 (y year);

insert year4 select y from year2;

select \* from year4; -- 2020 1920 1980 2080

Solution:

String or number data types can be used to store these values.

Issue: TIMESTAMP and DATETIME data types

The TIMESTAMP data type is identical to the DATETIME data type and can have duplicate values.

Example:

create table table\_ts (

id int auto\_increment not null, d datetime null,

t timestamp not null default current\_timestamp on update urrent\_timestamp,

key(id));

insert table\_ts (d) values (now()),(now()),(now()),(now()),(now());

select t, count(\*) from table\_ts group by t

-- 2006-12-22 19:20:38 | 5

Solution:

The TIMESTAMP type is easily emulated by using a trigger on INSERT and UPDATE that saves the current datetime in a **datetime** field.

###### String types

Issue: VARCHAR and VARBINARY maximum size

The maximum size of the MySQL VARCHAR and VARBINARY data types is 65,535.

MySQL VARCHAR data longer than 65,535 is transformed into MEDIUMTEXT or LONGTEXT.

MySQL VARBINARY data longer than 65,535 is transformed into MEDIUMBLOB or LONGBLOB.

Example:

create table t\_varchar (v varchar(65532));

describe t\_varchar; -- v varchar(65532) ...

create table t\_varchar (v varchar(65536));

describe t\_varchar; -- v mediumtext ...

create table t\_varbinary (v varbinary(65532));

describe t\_varbinary; -- v varbinary(65532) ...

create table t\_varbinary (v varbinary(65536));

describe t\_varbinary; -- v mediumblob ...

Solution:

Use the **varchar(max)** and **varbinary(max)** data types to store character and binary data that is longer than 8,000 bytes.

Issue: BINARY attribute for fields with CHAR and VARCHAR data types

The MySQL BINARY attribute causes the binary collation for the column character set to be used. For example, CHAR(5) BINARY in MySQL is treated as CHAR(5) CHARACTER SET latin1 COLLATE latin1\_bin, assuming that the default character set is latin1.

Example:

create table char\_binary\_ci (v varchar(8));

insert char\_binary\_ci values ('a'),('A'),('C'),('B');

select \* from char\_binary\_ci order by v; -- 'a' 'A' 'B' 'C'

create table char\_binary\_cs (v varchar(8) binary);

insert char\_binary\_cs values ('a'),('A'),('C'),('B');

select \* from char\_binary\_cs order by v; -- 'A' 'B' 'C' 'a'

Solution:

No solution yet.

Issue: CHAR, VARCHAR, and TEXT data types can have Unicode character sets

MySQL has two Unicode character sets: ucs2 (UCS-2 Unicode) and utf8 (UTF-8 Unicode).

SQL Server contains the **nchar** and **nvarchar** data types to store Unicode data and uses the Unicode UCS-2 character set.

Example:

create table unicode\_ucs2 (v varchar(10) character set ucs2);

create table unicode\_utf8 (v varchar(10) character set utf8);

create table collation\_cp (v varchar(10) charset cp1251);

insert unicode\_ucs2 values ('Привет!');

insert unicode\_ucs2 values ('您好您');

insert unicode\_utf8 values ('Привет!');

insert unicode\_utf8 values ('您好您');

insert collation\_cp values ('Привет!');

select length(v) from unicode\_ucs2; -- 14 6

select length(v) from unicode\_utf8; -- 13 9

select length(v) from collation\_cp; -- 7

select char\_length(v) from unicode\_ucs2; -- 7 3

select char\_length(v) from unicode\_utf8; -- 7 3

select char\_length(v) from collation\_cp; -- 7

Solution:

Convert CHAR, VARCHAR, and TEXT data types with Unicode character sets to the SQL Server **nchar** and **nvarchar** data types.

Issue: BLOB and TEXT data types can be indexed

For indexes on BLOB and TEXT columns, you must specify an index prefix length in MySQL. Prefixes can be up to 1,000 bytes long (767 bytes for InnoDB tables).

Example:

create table blob\_index (blob\_col blob, index(blob\_col(20)));

Solution:

No solution yet.

Issue: String constants can contain ESCAPE sequences

Each ESCAPE sequence in a string constant begins with a backslash (‘\’) in MySQL.

Example:

select 'This is \'Quoted string\'';

select 'This is ''Quoted string''';

-- This is 'Quoted string'

Solution:

No solution yet.

###### ENUM and SET data types

Issue: ENUM (*enumerate*) data type

MySQL supports the ENUM (*enumerate*) data type. An ENUM is a string object with a value chosen from a list of allowed values that are enumerated explicitly in the column specification at table creation time.

If you insert an invalid value into an ENUM, the empty string is inserted instead as a special error value. If an ENUM column is declared to allow NULL, the NULL value is a legal value for the column, and the default value is NULL. If an ENUM column is declared NOT NULL, its default value is the first element of the list of allowed values.

Each enumeration value has an numeric index.

ENUM values are sorted according to the order in which the enumeration members were listed in the column specification.

Example:

create table table\_enum (e enum ('a','b','c') not null);

insert into table\_enum values ('a');

insert into table\_enum values ('d');

insert into table\_enum values ('a,c');

insert into table\_enum values ('b,b,b');

insert into table\_enum values ('b');

insert into table\_enum values ();

select \* from table\_enum; -- 'a','','','','b','a';

select \* from table\_enum where e=1 -- 'a', 'a'

--------------------------------------------------

 create procedure proc\_enum (e enum ('a','b','c'))

 begin

 if e!=''

 then select e;

 else select 'Invalid argument';

 end if;

 end

call proc\_enum ('a'); -- 'a'

call proc\_enum ('t'); -- 'Invalid argument'

Solution:

Try to emulate ENUM data type as a lookup table, such as in the following example code:

create table someenumtype (\_id integer, \_value varchar(max))

The original table will have a reference to this hash-table by \_id.

You must add joins to all queries where the ENUM field value is used.

Issue: SET data type

MySQL supports the SET data type. A SET is a string object that can have zero or more values, each of which must be chosen from a list of allowed values specified when the table is created.

If you set a SET column to an unsupported value, the value is ignored.

MySQL stores SET values numerically, with the low-order bit of the stored value corresponding to the first set member.

Example:

create table table\_set (s set('a','b','c') not null);

insert into table\_set values ('a');

insert into table\_set values ('d');

insert into table\_set values ('a,c');

insert into table\_set values ('b,b,b');

insert into table\_set values ('b');

insert into table\_set values ();

select \* from table\_set; -- 'a','','a,c','b','b',''

select \* from table\_set where s='a,c' -- 'a,c'

--------------------------------------------------

 create procedure proc\_set (p char(1), s set ('a','b','c'))

 begin

 if find\_in\_set(p,s)>0

 then select p;

 else select 'Invalid argument';

 end if;

 end

call proc\_set ('a','b,c,a'); -- 'a'

call proc\_set ('a','b,c'); -- 'Invalid argument'

Solution:

The SET data type has dual nature—it is both an integer (up to 64 bit) and a string. Each bit in SET corresponds to a string description. The string representation of a SET value consists of appropriate strings, concatenated by commas.

Data manipulation is possible with both integer and string representations of SET.

Internally, SET is stored as integer; the size depends on the number of SET values (from 1 to 8 bytes). SQL Server emulation of the SET data type should be based on **bigint**, the largest possible integer data type.

To hold the string representation of bits in a SET value and to define all possible bits, create a “lookup table” as in the following example code:

create table lookup\_set(

schemaname sysname not null, -- schema name

tablename sysname not null, -- table name

colname sysname not null, -- column name

bitmask bigint not null, -- bitmask for value

position int not null, -- position in list

description varchar(512) not null, -- chararcter description of value

constraint pk\_lookup\_set

 primary key clustered (schemaname,tablename,colname,bitmask)

)

In addition, create a set of UDFs to support operations that use the SET data type.

|  |  |
| --- | --- |
| UDF | Description |
| char\_to\_set | Converts a string representation to an integer representation.**Note:** char\_to\_set always acts as if “strict mode” is disabled or the IGNORE word in an INSERT or UPDATE clause is present. |
| set\_to\_char | Converts an integer representation to a char representation.  |
| clean\_set | Removes invalid bits from an integer representation.**Note:** Usage of clean\_set may depend on “strict mode” or the IGNORE word in INSERT and UPDATE clauses. |
| check\_set  | Test to see if a given integer is a valid SET value.  |
| find\_in\_set | Emulates the MySQL function FIND\_IN\_SET. Check the FIND\_IN\_SET function for possible name clash with the second emulation of FIND\_IN\_SET.  |

###### Other types

Issue: MySQL spatial data types

MySQL has data types that correspond to **OpenGIS** classes (MySQL spatial data types).

Example:

create table spatial\_type (g geometry, p point,

 l linestring, pg polygon, mp multipoint)

Solution:

No solution yet.

###### Implicit data type conversion

Issue: Implicit data type conversion in MySQL

When a value of one type is used in a context that requires a value of another type, MySQL automatically performs extensive type conversion according to the kind of operation that is performed.

Examples:

select 100+'ABC' -- 100

select 100+'23ABC' -- 123

select concat('ABC',345,now(),50.4789) -- ABC3452006-11-08 19:00:0050.4789

drop table if exists table\_date;

create table table\_date

(d datetime, b smallint, i int(10) zerofill, f float, s varchar(64));

set @d=19980514;

insert into table\_date values (@d, @d, @d, @d, @d);

select \* from table\_date;

-- 1998-05-14 00:00:00 32767 0019980514 1.99805e+007 19980514

set @d=now();

insert into table\_date values (@d, @d, @d, @d, @d);

select \* from table\_date;

-- 2006-11-08 19:24:25 2006 0000002006 2006 2006-11-08 19:26:27

Solution:

No solution yet.

###### Data type default values

Issue: Implicit DEFAULT values

If a column definition does not include an explicit DEFAULT value, MySQL determines the default value as follows:

* If the column can take NULL as a value, the column is defined with an explicit DEFAULT NULL clause.
* If the column cannot take NULL as a value, MySQL defines the column without an explicit DEFAULT clause. For data entry, if an INSERT or REPLACE statement does not include a value for a column, MySQL handles the column according to the SQL mode that is in effect at the time:
* If strict SQL mode is not enabled, MySQL sets the column to the implicit default value for the column data type.
* If strict mode is enabled, an error occurs for transactional tables and the statement is rolled back. For non-transactional tables, an error occurs, but if this happens for the second or subsequent row of a multiple-row statement, the preceding rows will have been inserted.

Example:

create table table\_default (i int not null, d datetime not null,

 s varchar(64) not null, e enum ('a','b','c') not null, n int null);

insert table\_default values ();

insert table\_default values (default,default,'ABC',default,default);

select \* from table\_default;

-- 0 0000-00-00 00:00:00 a NULL

-- 0 0000-00-00 00:00:00 ABC a NULL

-- DEFAULT function example

create table table\_defaultfunc (a int not null default 1,

 b int not null default 2);

insert table\_defaultfunc values ();

insert table\_defaultfunc values (default(b),default(a));

select \* from table\_defaultfunc;

-- 1 2

-- 2 1

Solution:

No solution yet.

#### MySQL Migration Issues

This section identifies problems that may occur when migrating from MySQL 5 to SQL Server 2005 and suggests ways to handle them.

##### Operators

This section explains the differences between operators in MySQL and SQL Server 2005.

###### Comparison operators

Issue: Comparison operators in DML statements

Unlike SQL Server, MySQL allows comparison operators in DML statements.

Example:

create table table\_logic (id int not null,

 v varchar(64) not null, b int not null);

insert table\_logic values (1,'1=2',1=2);

insert table\_logic values (2,'1>2',1>2);

insert table\_logic values (3,'1<2',1<2);

select \* from table\_logic; -- 1 1=2 0 | 2 1>2 0 | 3 1<2 1

select 1=2, 1>2, 1<2 from dual; -- 0 0 1

update table\_logic set v='2=3', b=2=3 where id=3;

select \* from table\_logic; -- 3 2=3 0

update table\_logic set v='NULL IS UNKNOWN', b=NULL IS UNKNOWN where id=3;

select \* from table\_logic; -- 3 NULL IS UNKNOWN 1

select @a is unknown, @a is null, @a is not null; -- 1 1 0

set @a=5-1=3+1

select @a -- 0

select 'a' in ('a','b','c'), 'a' not in ('a','b','c'); -- 1 0

select 1=2=0=5=0, 2>1=1<7=1<0 -- 1 0

Solution:

Emulate the comparison operators in DML statements by using the CASE function.

Issue: NULL-safe equal comparison operator <=>

This MySQL operator performs an equality comparison like the = operator, but returns 1 rather than NULL if both operands are NULL, and 0 rather than NULL if one operand is NULL. SQL Server does not have an identical operator.

Example:

select 1 <=> 1, null <=> null, 1 <=> null, @d <=> null;

-- 1 1 0 1

select 1 = 1, null = null, 1 = null, @d = null;

-- 1 NULL NULL NULL

Solution:

No solution yet.

Issue: IS [NOT] *boolean\_value* comparison operator

This MySQL operator tests a value against a Boolean value, where *boolean\_value* can be TRUE, FALSE, or UNKNOWN. SQL Server does not have a similar operator.

Example:

create table table\_is\_int (i int);

insert table\_is\_int values (-1),(0),(1),(2),(3),(null);

select i is true from table\_is\_int; -- 1 0 1 1 1 0

select i is false from table\_is\_int; -- 0 1 0 0 0 0

select i is unknown from table\_is\_int; -- 0 0 0 0 0 1

select i=0 is true from table\_is\_int; -- 0 1 0 0 0 0

select \* from table\_is\_int where (i is true) is false; -- 0 NULL

select 'A' is false, 'A' is true,

 '7A' is false, '7A' is true, now() is true;

-- 1 0 0 1 1

Solution:

Emulate this comparison operator by using the CASE function.

Issue: IS NULL comparison operator extra features

MySQL supports extra features for the IS NULL comparison operator.

In MySQL, you can find the row that contains the most recent AUTO\_INCREMENT value by issuing a statement of the following form immediately after generating the value:

SELECT \* FROM tbl\_name WHERE auto\_col IS NULL

For DATE and DATETIME columns that are declared as NOT NULL, you can find the special date '0000-00-00' by using a statement such as the following:

SELECT \* FROM tbl\_name WHERE date\_column IS NULL

Example:

create table auto\_inc (id int not null auto\_increment,

 v varchar(64) not null, key(id));

insert auto\_inc (v) values ('ABC');

insert auto\_inc (v) values ('DEF');

insert auto\_inc (v) values ('GHI');

select \* from auto\_inc where id is null;

-- 3 'GHI'

create table auto\_date (d datetime not null, v varchar(64) not null);

insert auto\_date set v='A';

insert auto\_date set v='B';

insert auto\_date set v='C', d=now();

select \* from auto\_date where d is null;

-- 0000-00-00 00:00:00 A

-- 0000-00-00 00:00:00 B

Solution:

No solution yet.

###### Bit operators

Issue: Bit shift operators

MySQL has bit shift operators (<< and >>), which are not supported in SQL Server.

Example:

create procedure bit\_shift (count int)

begin

declare v bigint; declare i int;

set v:=1; set i:=1;

while i<=count do

 set v := v << 1;

 select v, i;

 set i := i+1;

end while;

end;

call bit\_shift (70);

-- 2 1

-- 4611686018427387904 62

-- 9223372036854775807 63

-- 9223372036854775807 64

Solution:

No solution yet.

###### Assignment operators

Issue: Variable assignment in SET statements

In MySQL, variables can be assigned in a SET statement by using the := or = operators.

MySQL can also assign a value to a user variable in statements other than SET. In this case, the assignment operator must be := and not = because = is treated as a comparison operator in non-SET statements.

Unlike SQL Server, if a variable is assigned in a MySQL SELECT statement, the recordset is returned.

Example:

set @a=1; set @b:=2;

select @a, @b, @a=@b; -- 1 2 0

select @a:=@b; -- 2

select @a, @b, @a=@b; -- 2 2 1

create table assign\_var (i int not null);

insert assign\_var values (1),(2),(3);

select @i, @i:=i from assign\_var order by i;

-- NULL 1

-- 1 2

-- 2 3

select @i:=i, @i from assign\_var order by i;

-- 1 1

-- 2 2

-- 3 3

select @i; -- 3

Solution:

No solution yet.

##### Variables

This section explains differences between variables in MySQL and SQL Server 2005.

Issue: Types of variables supported

MySQL supports two types of variables:

* User-defined variables @var\_name
* Local variables (variables in stored routines)
DECLARE var\_name[,...] type [DEFAULT value];

In MySQL, user-defined variables do not use the DECLARE statement for initialization. They are initialized implicitly at the moment of first set (with SET or SELECT statement) or use. If you refer to a variable that has not been initialized with a SET or SELECT statement, the variable has a value of NULL and a type of string.

User-defined variables are connection-specific. SQL Server does not have connection-specific variables.

Example 1:

 create procedure proc ()

 begin

 select @a;

 end

set @a=100;

call proc2 ();

Example 2:

 create procedure proc (inout par\_a int)

 begin

 set par\_a=200;

 end

call proc2 (@b);

select @b;

Solution:

No solution yet.

Issue: Case sensitivity of user-defined variables

User-defined variables names are case sensitive in versions earlier than MySQL 5.0 and not case sensitive in MySQL 5.0 and later.

This should be considered when choosing the SQL Server collation.

Solution:

No solution yet.

Issue: Default value of local variables

MySQL local variables can have a default value.

Example:

create procedure ProcA ()

begin

declare var\_a int default 100;

declare var\_b varchar(8) default 'ABCDEFGHIJKLMN';

declare var\_c datetime default now();

declare var\_d int;

select var\_a, var\_b, var\_c, var\_d;

-- 100 ABCDEFGH 2006-11-08 15:05:04 (NULL)

end

Solution:

No solution yet.

##### Utility Statements

Issue: DELIMITER command

The MySQL DELIMITER command allows changing the statements delimiter.

Example:

create table table\_a (id int);

select \* from table\_a;

delimiter //

select \* from table\_a//

drop table table\_a//

Solution:

No solution yet.

Issue: HELP command (HELP syntax)

The HELP statement returns online information from the MySQL Reference manual.

HELP 'search\_string'

Example:

HELP 'replace'

Syntax:

REPLACE(str,from\_str,to\_str)

Returns the string str with all occurrences of the string from\_str

replaced by the string to\_str. REPLACE() performs a case-sensitive

match when searching for from\_str.

Solution:

No solution yet.

##### Data Definition Statements

This section explains differences between the MySQL and SQL Server 2005 Data Definition Languages and provides common solutions for particular migration issues. It covers creation of tables, schemas, and views, conversion of temporary tables, and other DDL specific issues.

###### IF NOT EXISTS, IF EXISTS, OR REPLACE clauses

Issue: IF NOT EXISTS clause in CREATE DATABASE, CREATE TABLE, CREATE EVENT statements

The keywords IF NOT EXISTS prevent an error from occurring if the table (database, event) exists.

**Note:** If you use IF NOT EXISTS in a CREATE TABLE...SELECT statement, any rows selected by the SELECT part are inserted regardless of whether the table already exists.

MySQL Example:

create database db\_exists;

create database db\_exists;

-- Error Code : 1007 Can't create database 'db\_exists'; database exists

**A:** create database if not exists db\_exists;

-- No Action

create table exists\_a (i int not null);

create table exists\_a (i int not null);

-- Error Code : 1050 Table 'exists\_a' already exists

**B:** create table if not exists exists\_a (i int not null, v varchar(64) null);

-- No Action

show create table exists\_a; -- create table exists\_a (i int(11) not null)

**C:** create table if not exists exists\_a select now() as d from dual;

show create table exists\_a; -- create table exists\_a (i int(11) not null)

select \* from exists\_a; -- 2007

Solution:

* CREATE DATABASE

Replace the IF NOT EXISTS clause by following condition:

IF NOT EXISTS (SELECT name FROM sys.databases

    WHERE name = N'<db\_name>')

BEGIN

<create\_database\_statement\_without\_if\_not\_exists>

END

* CREATE TABLE

If CREATE TABLE...SELECT syntax is not used, replace the IF NOT EXISTS clause with the following condition:

* For permanent tables:

IF NOT EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'<table\_name>') AND type in (N'U'))

BEGIN

<create\_table\_statement\_without\_if\_not\_exists>

END

* For temporary tables:

IF NOT EXISTS (SELECT \* FROM tempdb.sys.objects WHERE object\_id =

 OBJECT\_ID(N'tempdb..<#table\_name>') AND type in (N'U'))

BEGIN

<create\_#table\_statement\_without\_if\_not\_exists>

END

If CREATE TABLE...SELECT syntax is used, replace the IF NOT EXISTS clause with the following condition:

* For permanent tables:

IF NOT EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'<table\_name>') AND type in (N'U'))

BEGIN

<create\_table\_statement\_without\_if\_not\_exists>

<insert\_select\_statement>

END

ELSE BEGIN

<insert\_select\_statement>

END

* For temporary tables:

IF NOT EXISTS (SELECT \* FROM tempdb.sys.objects WHERE object\_id =

 OBJECT\_ID(N'tempdb..<#table\_name>') AND type in (N'U'))

BEGIN

<create\_#table\_statement\_without\_if\_not\_exists>

<insert\_select\_statement>

END

ELSE BEGIN

<insert\_select\_statement>

END

* CREATE EVENT

No solution yet.

SQL Server Example:

**A:** IF NOT EXISTS (SELECT name FROM sys.databases WHERE name = N'db\_exists')

BEGIN

create database db\_exists

END

**B:** IF NOT EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'exists\_a') AND type in (N'U'))

BEGIN

create table exists\_a (i int not null, v varchar(64) null)

END

**C:** IF NOT EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'exists\_a') AND type in (N'U'))

BEGIN

create table exists\_a (d datetime not null default '1753-01-01 00:00:00')

insert exists\_a select getdate() as d

END

ELSE BEGIN

insert exists\_a select getdate() as d

END

Issue: IF EXISTS clause in DROP DATABASE, DROP TABLE, DROP VIEW, DROP EVENT, DROP PROCEDURE, DROP FUNCTION statements

IF EXISTS is used to prevent an error from occurring if the database (table, view, event, procedure, function) does not exist.

MySQL Example:

**A:** drop database if exists db\_exists;

 drop database db\_exists;

 -- Error Code : 1008 Can't drop database 'db\_exists'; database doesn't exist

**B:** drop table if exists exists\_a;

**C:** drop view if exists exists\_view;

**D:** drop procedure if exists exists\_proc;

**E:** drop function if exists exists\_func;

Solution:

* DROP DATABASE. Replace the IF EXISTS clause with the following condition:

IF EXISTS (SELECT name FROM sys.databases WHERE name = N'<db\_name>')

BEGIN

<drop\_database\_statement\_without\_if\_exists>

END

* DROP TABLE. Replace the IF EXISTS clause with the following condition:

IF EXISTS (SELECT \* FROM tempdb.sys.objects WHERE object\_id =

 OBJECT\_ID(N'tempdb..<#table\_name>') AND type in (N'U'))

BEGIN

<drop\_#table\_statement\_without\_if\_exists>

END

ELSE BEGIN

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'<table\_name>') AND type in (N'U'))

BEGIN

<drop\_table\_statement\_without\_if\_exists>

END

* DROP TEMPORARY TABLE. Replace the IF EXISTS clause with the following condition:

IF EXISTS (SELECT \* FROM tempdb.sys.objects WHERE object\_id =

 OBJECT\_ID(N'tempdb..<#table\_name>') AND type in (N'U'))

BEGIN

<drop\_#table\_statement\_without\_if\_exists>

END

* DROP VIEW. Replace the IF EXISTS clause with the following condition:

IF EXISTS (SELECT \* FROM sys.views WHERE object\_id =

 OBJECT\_ID(N'<view\_name>'))

BEGIN

<drop\_view\_statement\_without\_if\_exists>

END

* DROP EVENT. No solution yet.
* DROP PROCEDURE. Replace the IF EXISTS clause with the following condition:

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'<proc\_name>') AND type in (N'P', N'PC'))

BEGIN

<drop\_procedure\_statement\_without\_if\_exists>

END

* DROP FUNCTION. Replace the IF EXISTS clause with the following condition:

IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'<func\_name>')

 AND type in (N'FN', N'IF', N'TF', N'FS', N'FT'))

BEGIN

<drop\_function\_statement\_without\_if\_exists>

END

SQL Server Example:

**A:** IF EXISTS (SELECT name FROM sys.databases WHERE name = N'db\_exists')

BEGIN

drop database db\_exists

END

**B:** IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'exists\_a') AND type in (N'U'))

BEGIN

drop table exists\_a

END

**C:** IF EXISTS (SELECT \* FROM sys.views WHERE object\_id =

 OBJECT\_ID(N'exists\_view'))

BEGIN

drop view exists\_view

END

**D:** IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'<exists\_proc>') AND type in (N'P', N'PC'))

BEGIN

drop procedure exists\_proc

END

**E:** IF EXISTS (SELECT \* FROM sys.objects WHERE object\_id =

 OBJECT\_ID(N'exists\_func')

 AND type in (N'FN', N'IF', N'TF', N'FS', N'FT'))

BEGIN

drop function exists\_func;

END

Issue: OR REPLACE clause in CREATE VIEW statements

The MySQL OR REPLACE clause replaces an existing view.

MySQL Example:

create or replace view repl\_view as select now();

create or replace view repl\_view as select version();

Solution:

Replace the OR REPLACE clause with the following condition:

IF EXISTS (SELECT \* FROM sys.views WHERE object\_id =

 OBJECT\_ID(N'<view\_name>'))

BEGIN

<drop\_view\_statement>

exec dbo.sp\_executesql @statement

 = N'<create\_view\_statement\_without\_or\_replace>'

END

ELSE BEGIN

exec dbo.sp\_executesql @statement

 = N'<create\_view\_statement\_without\_or\_replace>'

END

SQL Server Example:

IF EXISTS (SELECT \* FROM sys.views WHERE object\_id =

 OBJECT\_ID(N'repl\_view'))

BEGIN

drop view repl\_view

exec dbo.sp\_executesql @statement

 = N'create view repl\_view as select getdate() as d'

END

ELSE BEGIN

exec dbo.sp\_executesql @statement

 = N'create view repl\_view as select getdate() as d'

END

###### Temporary tables

Issue: MySQL TEMPORARY tables not dropped when they go out of scope

SQL Server temporary tables created by the CREATE TEMPORARY TABLE statement are visible only to the current connection, and are dropped automatically when the connection is closed. However, MySQL temporary tables are not dropped when they go out of scope.

In SQL Server:

* A local temporary table created in a stored procedure is dropped automatically when the stored procedure is finished. The table can be referenced by any nested stored procedures executed by the stored procedure that created the table. The table cannot be referenced by the process that called the stored procedure that created the table.
* A local temporary table created within a stored procedure or trigger can have the same name as a temporary table that was created before the stored procedure or trigger is called. However, if a query references a temporary table and two temporary tables with the same name exist at that time, it is not defined which table the query is resolved against. Nested stored procedures can also create temporary tables with the same name as a temporary table that was created by the stored procedure that called it. However, for modifications to resolve to the table that was created in the nested procedure, the table must have the same structure and the same column names as the table that was created in the calling procedure.

MySQL Example 1:

create procedure proctemptable ()

begin

create temporary table table\_temp (d datetime);

insert table\_temp values (now());

end

call proctemptable();

select \* from table\_temp; -- 2006-11-20 11:18:58

call proctemptable(); -- Error Code : 1050 Table 'table\_temp' already exists

MySQL Example 2:

create procedure test2 ()

begin

create temporary table t (x int);

insert into t values (2);

select x as test2col from t;

end

create procedure test1 ()

begin

create temporary table t (x int);

insert into t values (1);

select x as test1col from t;

call test2 ();

end

call test1 ();

-- test1col = 1 (1 row)

-- ERROR 1050 (42S01): Table 't' alredy exists

Solution:

No solution yet.

Issue: TEMPORARY keyword in CREATE TABLE and DROP TABLE statements

In MySQL, the TEMPORARY keyword in a CREATE TABLE statement is used to create a temporary table.

In MySQL, the TEMPORARY keyword in a DROP TABLE statement is used to drop a temporary table.

MySQL Example:

**A:** create temporary table temp\_table\_a (a int not null);

**B:** create temporary table if not exists atest.temp\_table\_b (b int not null);

**C:** drop temporary table temp\_table\_a;

**D:** drop temporary table if exists atest.temp\_table\_b;

Solution:

Replace the TEMPORARY keyword with a single pound sign before the table name (#). Omit the database name.

SQL Server Example:

**A:** create table #temp\_table\_a (a int not null)

**B:** IF NOT EXISTS (SELECT \* FROM tempdb.sys.objects WHERE object\_id =

 OBJECT\_ID(N'tempdb..#temp\_table\_b') AND type in (N'U'))

BEGIN

create table #temp\_table\_b (b int not null)

END

**C:** drop table #temp\_table\_a;

**D:** IF EXISTS (SELECT \* FROM tempdb.sys.objects WHERE object\_id =

 OBJECT\_ID(N'tempdb..#temp\_table\_b') AND type in (N'U'))

BEGIN

drop table #temp\_table\_b;

END

Issue: Hiding tables

In MySQL, if a temporary table is created with same name as an existing non-temporary table**,** the existing non-temporary table is hidden until the temporary table is dropped.

MySQL Example 1:

create table permanent\_temp (v varchar(4) not null, d datetime not null);

insert permanent\_temp values ('ABCD',now());

select \* from permanent\_temp; -- 'ABCD' '2007-02-08 16:19:40'

create temporary table permanent\_temp (i int not null);

insert permanent\_temp values (1);

select \* from permanent\_temp; -- 1

drop table permanent\_temp; -- drop temporary table

select \* from permanent\_temp; -- 'ABCD' '2007-02-08 16:19:40'

drop table permanent\_temp; -- drop permanent table

MySQL Example 2:

create table permanent\_temp (i int not null);

insert permanent\_temp values (0);

select \* from permanent\_temp; -- 0

create temporary table permanent\_temp (i int not null);

insert permanent\_temp values (1);

select \* from permanent\_temp; -- 1

drop temporary table permanent\_temp; -- drop temporary table

select \* from permanent\_temp; -- 0

drop temporary table permanent\_temp;

-- Error Code : 1051 Unknown table 'permanent\_temp'

drop table permanent\_temp; -- drop permanent table

Solution:

No solution yet.

Issue: Temporary tables in functions

In MySQL, temporary tables can be used in functions. SQL Server does not allow this.

MySQL Example:

create function temp\_func\_sum () returns double

begin

declare s double;

select sum(i) into s from temp\_func;

return s;

end

**session 1:**

create temporary table temp\_func (i int not null);

insert into temp\_func values (1),(2),(3),(4);

select temp\_func\_sum(); -- 10

**session 2:**

create temporary table temp\_func (i numeric(19,9) not null);

insert into temp\_func values (1.1000),(1.0100),(1.0010),(1.0001);

select temp\_func\_sum(); -- 4.1111

**session 3:**

create temporary table temp\_func (i varchar(4) not null);

insert into temp\_func values ('ABCD'),('5A'),('3.14P'),('1')

select temp\_func\_sum(); -- 9.14

Solution:

No solution yet.

###### SCHEMA keyword in DATABASES statements

Issue: SCHEMA keyword in DATABASE statements

In DATABASE statements (ALTER, CREATE, DROP, RENAME) the SCHEMA keyword can be used as synonym of DATABASE keyword.

Solution:

Replace the SCHEMA keyword with the DATABASE keyword.

###### CHARACTER SET and COLLATE clauses in DDL statements

Issue: CHARACTER SET and COLLATE clauses in DDL statements

In MySQL terminology, a *character* set is a set of symbols and encodings and a collation is a set of rules for comparing characters in a character set. The term *collation* in SQL Server combines both these meanings.

MySQL Example:

create database db\_a character set latin1 collate latin1\_swedish\_ci;

Solution:

Convert the MySQL CHARACTER SET and COLLATE clauses to the SQL Server COLLATE clause.

Issue: CONVERT TO CHARACTER SET and [DEFAULT] CHARACTER SET clauses in ALTER TABLE statements

Unlike MySQL, SQL Server does not support collation changes on table level.

Solution:

No solution yet.

###### CREATE INDEX statement

Issue: NULL values in UNIQUE indexes

A MySQL UNIQUE index allows multiple NULL values for columns that can contain NULL.

MySQL Example:

create table tabindex\_b (i int null);

create unique index idx\_tabindex\_b on tabindex\_b (i);

insert tabindex\_b values (1);

insert tabindex\_b values (2);

insert tabindex\_b values (3);

insert tabindex\_b values (1);

-- Duplicate entry '1' for key 1

insert tabindex\_b values (null);

-- 1 row(s) affected

insert tabindex\_b values (null);

-- 1 row(s) affected

Solution:

No solution yet.

Issue: FULLTEXT indexes

MySQL can create FULLTEXT indexes on tables that do not have a unique key index.

MySQL can create more than one FULLTEXT index on a table.

The WITH PARSER option can be used only with FULLTEXT indexes. It associates a parser plugin with the index if full-text indexing and searching operations need special handling.

MySQL Example:

create table tabindex\_full (i int not null, t varchar(2048) null)

engine = myisam;

create unique index unique\_tabindex\_full on tabindex\_full (i);

create fulltext index full\_tabindex\_full on tabindex\_full (t);

Solution:

FULLTEXT indexes can be converted if all of the following conditions are met:

* Only one FULLTEXT index exists for the table.
* The table has a valid index to enforce a full-text search key.
* The FULLTEXT index does not use a parser plugin.

SQL Server Example:

create table tabindex\_full (i int not null, t varchar(2048) null)

create unique index unique\_tabindex\_full on tabindex\_full (i)

create fulltext index on tabindex\_full (t) key index unique\_tabindex\_full

Issue: SPATIAL indexes

SPATIAL indexes are supported only for MyISAM tables and can include only spatial columns that are defined as NOT NULL.

Solution:

No solution yet.

Issue: Index prefix length

For CHAR, VARCHAR, BINARY, and VARBINARY columns, indexes can be created that use only the leading part of column values, by using *col\_name*(*length*) syntax to specify an index prefix length. BLOB and TEXT columns also can be indexed, but a prefix length must be given. Prefix lengths are given in characters for non-binary string types and in bytes for binary string types. That is, index entries consist of the first length characters of each column value for CHAR, VARCHAR, and TEXT columns, and the first length bytes of each column value for BINARY, VARBINARY, and BLOB columns.

MySQL Example:

create table tabindex\_text (t text not null);

create unique index unique\_tabindex\_text on tabindex\_text (t(4));

insert tabindex\_text values ('ABCDEFG');

insert tabindex\_text values ('ABCEFGD');

insert tabindex\_text values ('ABKLMND');

insert tabindex\_text values ('ABCKLMN');

insert tabindex\_text values ('ABCDLMN');

-- Duplicate entry 'ABCD' for key 1

create table tabindex\_blob (b blob null);

create unique index unique\_tabindex\_blob on tabindex\_blob (b(4));

insert tabindex\_blob values (0x01020304050607);

insert tabindex\_blob values (0x01020310111204);

insert tabindex\_blob values (0x01020304101112);

-- Duplicate entry ' ' for key 1

Solution:

Add the computed columns, which emulate the index prefix length functionality, to the table. Create the index on the computed columns.

SQL Server Example:

create table tabindex\_text (t varchar(max) not null,

 t\_comp as convert(varchar(4),t))

create unique index unique\_tabindex\_text on tabindex\_text (t\_comp)

insert tabindex\_text values ('ABCDEFG')

insert tabindex\_text values ('ABCEFGD')

insert tabindex\_text values ('ABKLMND')

insert tabindex\_text values ('ABCKLMN')

insert tabindex\_text values ('ABCDLMN')

-- Cannot insert duplicate key row ...

create table tabindex\_blob (b varbinary(max) null,

 b\_comp as convert(varbinary(4),b));

create unique index unique\_tabindex\_blob on tabindex\_blob (b\_comp);

insert tabindex\_blob values (0x01020304050607);

insert tabindex\_blob values (0x01020310111204);

insert tabindex\_blob values (0x01020304101112);

-- Cannot insert duplicate key row ...

###### CREATE TABLE statement

Issue: CONSTRAINT names

MySQL allows you to omit constraint names.

MySQL allows duplicate constraint names

MySQL Example:

create table tab\_constr (

id int not null, d datetime,

fk int not null,

constraint primary key (id),

constraint unique (d),

constraint foreign key (fk) references tab\_a (i)

);

create table tab\_constr\_dub (

id int not null, d datetime,

constraint key\_tab\_constr\_dub primary key (id),

constraint key\_tab\_constr\_dub unique (d)

);

Solution:

Generate valid and unique constraints names.

SQL Server Example:

create table tab\_constr (

id int not null, d datetime,

fk int not null,

constraint pk\_tab\_constr primary key (id),

constraint uq\_tab\_constr unique (d),

constraint fk\_tab\_constr foreign key (fk) references tab\_a (i)

);

create table tab\_constr\_dub (

id int not null, d datetime,

constraint pk\_tab\_constr\_dub primary key (id),

constraint uq\_tab\_constr\_dub unique (d)

);

Issue: Index definitions

In MySQL, indexes can be defined in the body of a table declaration.

MySQL Example:

create table tab\_index (

i int not null,

n int not null,

d datetime null,

v varchar(2048) not null,

primary key (i),

index idx\_tab\_index (n),

key (d),

fulltext index ft\_tab\_index (v)) engine = myisam;

Solution:

Remove index declarations from table declarations. Convert them into separate CREATE INDEX statements.

Generate valid and unique index names.

Replace the KEY keyword with the INDEX keyword.

SQL Server Example:

create table tab\_index (

i int not null,

n int not null,

d datetime null,

v varchar(2048) not null,

primary key (i))

create index idx\_tab\_index on tab\_index (n)

create index key\_tab\_index on tab\_index (d)

create fulltext index on tab\_index (v) key index pk\_\_tab\_index\_\_72e607db

Issue: FOREIGN KEY constraint indexes

Unlike SQL Server, in MySQL, for FOREIGN KEY constraints the index is created automatically.

Solution:

No solution yet.

Issue: RESTRICT keyword in reference options

MySQL supports the RESTRICT keyword in reference options.

Solution:

Replace the RESTRICT keyword in reference options with the NO ACTION keyword.

Issue: KEY keyword in column definitions

MySQL supports the KEY keyword in a column definition.

Solution:

Replace the KEY keyword in a column definition with the PRIMARY KEY keyword.

Issue: AUTO\_INCREMENT column option and AUTO\_INCREMENT table option

MySQL supports the AUTO\_INCREMENT column option and AUTO\_INCREMENT table option.

MySQL Example:

create table auto\_a (

i int not null auto\_increment primary key,

d datetime null

)

auto\_increment = 1000;

insert auto\_a values (null,now()),(null,now()),(null,now());

select \* from auto\_a;

-- 1000 2007-02-16 17:41:43

-- 1001 2007-02-16 17:41:43

-- 1002 2007-02-16 17:41:43

Solution:

The MySQL AUTO\_INCREMENT column option should be replaced by using the SQL Server IDENTITY column property. The MySQL AUTO\_INCREMENT table option value should be converted to SQL Server as a seed parameter of the IDENTITY property.

SQL Server Example:

create table auto\_a (

i int not null identity(1000,1) primary key,

d datetime null

)

insert auto\_a values (getdate())

insert auto\_a values (getdate())

insert auto\_a values (getdate())

Issue: MERGE tables

MERGE tables (storage engine) are a collection of identical MyISAM tables that can be used as one.

MySQL Example:

create table merge\_a (a int not null) engine=myisam ;

create table merge\_b (b int not null) engine=myisam;

create table merge\_m (m int not null)

engine=merge union=(merge\_a,merge\_b);

insert merge\_a values (1),(2),(3);

insert merge\_b values (4),(5),(6),(7);

select \* from merge\_a; -- 1 2 3

select \* from merge\_b; -- 4 5 6 7

select \* from merge\_m; -- 1 2 3 4 5 6 7

update merge\_m set m=m+10 where m % 2 = 0;

select \* from merge\_a; -- 1 12 3

select \* from merge\_b; -- 14 5 16 7

Solution:

No solution yet.

Issue: CREATE TABLE...SELECT syntax

MySQL allows you to create one table from another by adding a SELECT statement at the end of the CREATE TABLE statement.

MySQL creates new columns for all elements in the SELECT list that have a unique name.

CREATE TABLE...SELECT adds the result data into a new table.

MySQL Example:

create table sel\_a select 1 as id from dual;

show create table sel\_a;

-- create table sel\_a (id bigint not null)

select \* from sel\_a; -- 1

create table sel\_b (id bigint not null) select 2 as id from dual;

show create table sel\_b;

-- create table sel\_b (id bigint not null)

select \* from sel\_b; -- 2

create table sel\_c (id bigint not null) select 'ABC' as val, 1000 as id

from dual;

show create table sel\_c;

-- create table sel\_c (val varchar(3) not null, id bigint not null)

select \* from sel\_c; -- ABC 1000

create table sel\_d (id bigint not null) select 'ABC' as val, 1000

from dual;

show create table sel\_d;

-- create table sel\_d (id bigint not null, val varchar(3) not null,

 `1000` bigint not null)

select \* from sel\_d; -- 0 ABC 1000

Solution:

No solution yet.

Issue: CREATE TABLE...LIKE syntax

MySQL supports LIKE to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table.

MySQL Example:

create table like\_a (

i int not null auto\_increment primary key,

d datetime not null unique,

v varchar(1024) null,

fulltext (v)

) engine = myisam;

create table like\_b like like\_a;

show create table like\_b;

/\*

create table `like\_b` (

 `i` int not null auto\_increment,

 `d` datetime not null,

 `v` varchar(1024) collate latin1\_general\_ci default null,

 primary key (`i`),

 unique key `d` (`d`),

 fulltext key `v` (`v`)

) engine=myisam default charset=latin1 collate=latin1\_general\_ci

\*/

Solution:

No solution yet.

Issue: Foreign key references to other databases

MySQL allows creating foreign key references to other databases.

Solution:

No solution yet.

###### ALTER TABLE statement

Issue: IGNORE keyword

The IGNORE extension controls how ALTER TABLE works if there are duplicates on unique keys in the new table or if warnings occur when strict mode is enabled. If IGNORE is not specified, the copy is aborted and rolled back if duplicate-key errors occur. If IGNORE is specified, only the first row in rows that have duplicates on a unique key is used. The other conflicting rows are deleted.

Solution:

No solution yet.

Issue: FIRST | AFTER keywords

In MySQL, to add a column at a specific position within a table row, the FIRST or AFTER *col\_name* clause is used. The default is to add the column last. You can also use the FIRST and AFTER clauses in CHANGE or MODIFY column operations.

Solution:

No solution yet.

Issue: ALTER clause

ALTER...SET DEFAULT and ALTER...DROP DEFAULT specify a new default value for a column or remove the old default value, respectively.

MySQL Example:

create table alter\_def (i int not null);

insert alter\_def values ();

alter table alter\_def alter i set default 99;

insert alter\_def values ();

alter table alter\_def alter i drop default;

insert alter\_def values ();

select \* from alter\_def; -- 0 99 0

Solution:

Convert ALTER...SET DEFAULT and ALTER...DROP DEFAULT clauses to ADD/DROP DEFAULT constraint clauses.

SQL Server Example:

create table alter\_def (i int not null);

insert alter\_def default values; -- Cannot insert the value NULL into 'i'

alter table alter\_def add constraint i\_def default 99 for i;

insert alter\_def default values;

alter table alter\_def drop constraint i\_def;

insert alter\_def default values; -- Cannot insert the value NULL into 'i'

select \* from alter\_def; -- 99

Issue: CHANGE clause

MySQL supports a CHANGE clause in ALTER TABLE statements.

MySQL Example:

create table change\_a (i int not null);

alter table change\_a change i i varchar(64) null;

show create table change\_a;

alter table change\_a change i d datetime not null;

show create table change\_a;

Solution:

The CHANGE clause is converted to a SQL Server ALTER COLUMN clause. If CHANGE renames a column, use an additional call to **sp\_rename**.

SQL Server Example:

create table change\_a (i int not null);

alter table change\_a alter column i varchar(64) null;

sp\_help 'change\_a';

alter table change\_a alter column i datetime not null;

exec sp\_rename 'change\_a.i', 'd', 'COLUMN';

sp\_help 'change\_a';

Issue: MODIFY clause

MySQL supports the MODIFY clause in ALTER TABLE statements.

MySQL Example:

create table modify\_a (i int not null);

alter table modify\_a modify i varchar(64) null;

show create table modify\_a;

alter table modify\_a modify i datetime not null;

show create table modify\_a;

Solution:

Convert the MODIFY clause to an ALTER COLUMN clause.

SQL Server Example:

create table modify\_a (i int not null);

alter table modify\_a alter column i varchar(64) null;

sp\_help 'modify\_a';

alter table modify\_a alter column i datetime not null;

sp\_help 'modify\_a';

Issue: DROP [COLUMN] *col\_name* clause

In MySQL, ALTER TABLE can contain a DROP clause, which has the same meaning as DROP COLUMN.

Solution:

Add any missing COLUMN keywords.

Issue: DROP PRIMARY KEY clause

MySQL supports the DROP PRIMARY KEY clause in ALTER TABLE statements.

MySQL Example:

create table alter\_c (id int not null, v varchar(64) not null);

alter table alter\_c add constraint pk\_alter\_c primary key (id);

alter table alter\_c drop primary key;

Solution:

The DROP PRIMARY KEY clause should be replaced with a DROP CONSTRAINT *pk\_constraint\_name* clause.

SQL Server Example:

create table alter\_c (id int not null, v varchar(64) not null)

alter table alter\_c add constraint pk\_alter\_c primary key (id)

alter table alter\_c drop constraint pk\_alter\_c

Issue: DROP {INDEX|KEY} *index\_name* clause

MySQL supports DROP INDEX clause in ALTER TABLE statements.

MySQL Example:

create table alter\_i (

i int not null,

n int not null,

primary key (i),

index idx\_tab\_index (n));

alter table alter\_i drop index idx\_tab\_index;

Solution:

DROP INDEX clauses in ALTER TABLE statements should be converted to separate DROP INDEX statements.

SQL Server Example:

create table alter\_i (

i int not null,

n int not null,

primary key (i))

create index idx\_tab\_index on alter\_i (n)

drop index alter\_i.idx\_tab\_index

Issue: DROP FOREIGN KEY clause

MySQL supports the DROP FOREIGN KEY clause in ALTER TABLE statements.

MySQL Example:

create table alter\_f (f\_id int not null, c\_id int not null);

alter table alter\_f add constraint fk\_alter\_f foreign key (c\_id)

 references alter\_c (id);

alter table alter\_f drop foreign key fk\_alter\_f;

Solution:

Replace the DROP FOREIGN KEY clause with a DROP CONSTRAINT *fk\_constraint\_name* clause.

SQL Server Example:

create table alter\_f (f\_id int not null, c\_id int not null)

alter table alter\_f add constraint fk\_alter\_f foreign key (c\_id)

 references alter\_c (id)

alter table alter\_f drop constraint fk\_alter\_f

Issue: DISABLE KEYS and ENABLE KEYS clauses

The ALTER TABLE...DISABLE KEYS clauses tell MySQL to stop updating non-unique indexes for a MyISAM table. You then use ALTER TABLE...ENABLE KEYS to re-create missing indexes. MySQL does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should speed up the operation significantly.

Solution:

No solution yet.

Issue: RENAME clause

MySQL supports a RENAME clause in ALTER TABLE statements.

MySQL Example:

create table rename\_a (i int not null);

insert rename\_a values (1);

select \* from rename\_a; -- 1

alter table rename\_a rename to rename\_b;

select \* from rename\_a; -- Table 'rename\_a' doesn't exist

select \* from rename\_b; -- 1

Solution:

Convert the RENAME clause to a separate **sp\_rename** call.

SQL Server Example:

create table rename\_a (i int not null);

insert rename\_a values (1);

select \* from rename\_a; -- 1

exec sp\_rename 'rename\_a', 'rename\_b';

select \* from rename\_a; -- Invalid object name 'rename\_a'

select \* from rename\_b; -- 1

Issue: ORDER BY clause

In MySQL, the ORDER BY clause in an ALTER TABLE statement enables you to create a new table with the rows in a specific order.

Solution:

No solution yet.

###### RENAME DATABASE statement

Issue: RENAME DATABASE statements

MySQL supports the RENAME DATABASE statement.

Solution:

Convert RENAME DATABASE statements to a **sp\_renamedb** call.

###### RENAME TABLE statement

Issue: RENAME TABLE statements

The MySQL RENAME TABLE statement renames one or more tables or views.

MySQL Example:

create table rename\_a (i int not null);

insert rename\_a values (1);

create table rename\_b (d datetime not null);

insert rename\_b values (now());

rename tables rename\_a to rename\_c, rename\_b to rename\_a,

 rename\_c to rename\_b;

select \* from rename\_a; -- 2007-02-20 15:03:37

select \* from rename\_b; -- 1

select \* from rename\_c; -- Table 'ATest.rename\_c' doesn't exist

create view rename\_view\_a as select 'view\_string' as vs;

rename table rename\_view\_a to rename\_view\_b;

select \* from rename\_view\_a; -- Table 'ATest.rename\_view\_a' doesn't exist

select \* from rename\_view\_b; -- 'view\_string'

Solution:

Convert each RENAME TABLE operation into a separate **sp\_rename** call.

SQL Server Example:

create table rename\_a (i int not null);

insert rename\_a values (1);

create table rename\_b (d datetime not null);

insert rename\_b values (getdate());

exec sp\_rename 'rename\_a', 'rename\_c'

exec sp\_rename 'rename\_b', 'rename\_a'

exec sp\_rename 'rename\_c', 'rename\_b'

select \* from rename\_a; -- 2007-02-20 15:06:48.967

select \* from rename\_b; -- 1

select \* from rename\_c; -- Invalid object name 'rename\_c'

create view rename\_view\_a as select 'view\_string' as vs;

exec sp\_rename 'rename\_view\_a', 'rename\_view\_b';

select \* from rename\_view\_a; -- Invalid object name 'rename\_view\_a'

select \* from rename\_view\_b; -- 'view\_string'

Issue: Moving tables between databases with the RENAME TABLE statement

The MySQL RENAME TABLE statement can be used to move a table from one database to another.

MySQL Example:

create table world.rename\_table (v varchar(8) null);

insert world.rename\_table values ('ABC');

rename table world.rename\_table to sakila.rename\_table;

select \* from world.rename\_table;

-- Table 'world.rename\_table' doesn't exist

select \* from sakila.rename\_table; -- 'ABC'

Solution:

No solution yet.

###### CREATE VIEW, ALTER VIEW, DROP VIEW statements

Issue: Database name prefix at view name

Unlike MySQL, in SQL Server CREATE/ALTER/DROP VIEW does not allow specifying the database name as a prefix to the object name.

MySQL Example:

create view sakila.view\_a as select 'ABCDE' as s;

select \* from sakila.view\_a; -- ABCDE

drop view sakila.view\_a;

Solution:

No solution yet.

Issue: LOCAL keyword in WITH CHECK OPTION clause

In a WITH CHECK OPTION clause for an updatable view, the LOCAL and CASCADED keywords determine the scope of check testing when the view is defined in terms of another view. The LOCAL keyword restricts the CHECK OPTION to only the view that is being defined. The CASCADED keyword causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is CASCADED.

MySQL Example:

create table t1 (a int);

create view v1 as select \* from t1 where a < 2

with check option;

create view v2 as select \* from v1 where a > 0

with local check option;

create view v3 as select \* from v1 where a > 0

with cascaded check option;

insert into v1 values (2); -- CHECK OPTION failed 'ATest.v1'

insert into v2 values (2);

insert into v3 values (2); -- CHECK OPTION failed 'ATest.v3'

select \* from t1; -- 2

Solution:

No solution yet.

Issue: Unnamed columns in view select list

MySQL automatically generates names for unnamed columns in a view's select list.

MySQL Example:

create table table\_name (a int not null, b int not null);

insert table\_name values (1,2);

create view view\_name as

select a, b, a+b, a\*b, now() from table\_name;

select \* from view\_name where `a+b`=3;

Solution:

Generate column names based on the MySQL select list.

SQL Server Example:

create table table\_name (a int not null, b int not null);

insert table\_name values (1,2);

create view view\_name as

select a, b, a+b as [a+b], a\*b as [a\*b], getdate() as [now()]

from table\_name;

select \* from view\_name where [a+b]=3;

###### CREATE EVENT, ALTER EVENT, DROP EVENT statements

Issue: MySQL events

MySQL *events* are tasks that run according to a schedule. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time.

Solution:

No solution yet.

###### CREATE, ALTER, DROP PROCEDURE/FUNCTION statements

Issue: Database name prefix at procedure/function name

Unlike MySQL, in SQL Server CREATE/ALTER/DROP PROCEDURE/FUNCTION do not allow specifying the database name as a prefix to the object name.

MySQL Example:

create function sakila.func\_drop () returns float

begin

declare s float;

set s:=3.14;

return s;

end

drop function sakila.func\_drop;

Solution:

No solution yet.

Issue: SQL SECURITY characteristic

In MySQL, the SQL SECURITY characteristic can be used to specify whether the routine should be executed by using the permissions of the user who creates the routine or the user who invokes it. The default value is DEFINER.

MySQL Example:

**root user:**

create table table\_access (i int not null);

insert table\_access values (1),(2),(3),(4),(5);

create procedure proc\_access ()

sql security definer

begin

select \* from table\_access;

end

grant execute on ATest.\* to abc;

**abc user:**

select \* from ATest.table\_access;

-- SELECT command denied to user 'abc'

call ATest.proc\_access();

-- 1 2 3 4 5

**root user:**

drop procedure proc\_access;

create procedure proc\_access ()

sql security invoker

begin

select \* from table\_access;

end

**abc user:**

select \* from ATest.table\_access;

-- SELECT command denied to user 'abc

call ATest.proc\_access();

-- SELECT command denied to user 'abc'

Solution:

No solution yet.

Issue: Routine parameter names

Unlike SQL Server, MySQL does not require an @ prefix for the names of a routine's variables.

MySQL Example:

create function func\_pi (p int) returns float

begin

declare s float;

set s:=p\*pi();

return s;

end

Solution:

Convert parameter names in routines to SQL Server procedure or function parameter names and use an at sign (@) as the first character.

SQL Server Example:

create function func\_pi (@p int) returns float

begin

declare @s float;

set @s=@p\*pi();

return @s;

end

Issue: INOUT procedure parameters

MySQL supports INOUT specification for procedure parameters.

MySQL Example:

 create procedure proc\_inout (a int, inout b int)

 begin

 set b=b+a;

 end

set @b=0;

call proc\_out(7,@b);

select @b; -- 7

call proc\_out(7,@b);

select @b; -- 14

Solution:

Convert MySQL INOUT procedure parameters to SQL Server OUT procedure parameters.

SQL Server Example:

 create procedure proc\_inout (@a int, @b int out)

 as

 begin

 set @b=@b+@a;

 end

declare @b int

set @b=0;

exec proc\_inout 7, @b out

select @b; -- 7

exec proc\_inout 7, @b out

select @b; -- 14

Issue: OUT procedure parameters

MySQL supports an OUT specification for procedure parameters.

An OUT parameter passes a value from the procedure back to the caller. Its initial value is NULL within the procedure, and its value is visible to the caller when the procedure returns.

MySQL Example:

 create procedure proc\_out (a int, out b int)

 begin

 set b=isnull(b)+a;

 end

set @b=99;

call proc\_out(7,@b);

select @b; -- 8

call proc\_out(7,@b);

select @b; -- 8

Solution:

No solution yet.

Issue: AS keyword before procedure body

MySQL does not use the AS keyword before the procedure body.

MySQL Example:

create procedure proc\_as (a int, b int)

begin

select a+b;

end

Solution:

Add the AS keyword before the procedure body.

SQL Server Example:

create procedure proc\_as (@a int, @b int)

as

begin

select @a+@b;

end

##### Data Manipulation Statements

This section describes differences between the MySQL and SQL Server 2005 Data Manipulation Languages and provides common solutions for typical migration issues.

It also covers SELECT, INSERT, UPDATE, and DELETE statements, discusses the conversion of a number of MySQL specific clauses such as LIMIT, and explains the differences between join syntax.

###### LIMIT clause

Issue: LIMIT clause in SELECT statements

The MySQL SELECT result can be limited by using the LIMIT clause.

If a LIMIT clause has two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return.

If a LIMIT clause has one argument, the value specifies the number of rows to return from the beginning of the result set.

MySQL Example:

select id, data from t1 order by id limit 3, 2;

Solution:

Emulate a LIMIT clause with one argument by using the TOP clause of a SELECT statement.

A LIMIT clause with two arguments can be emulated by using a subquery with the ROW\_NUMBER() function.

SQL Server Example:

select id, data

from

( select id, data, row\_number () over (order by id) - 1 as rn

 from t1 ) rn\_subquery

where rn between 3 and (3+2)-1

order by id

Issue: LIMIT and ORDER BY clauses in a single-table DELETE statement

The LIMIT clause places a limit on the number of rows that can be deleted.

If the DELETE statement includes an ORDER BY clause, the rows are deleted in the order specified in the clause. This is really useful only in conjunction with LIMIT.

MySQL Example:

delete from t3 where data<ascii(id) limit 2;

Solution:

Emulate the LIMIT clause by using the TOP clause in a DELETE statement.

Review the logic of DELETE statements with LIMIT and ORDER BY clauses.

SQL Server Example:

delete top (2) from t3 where data<ascii(id);

Issue: LIMIT and ORDER BY clauses in single-table UPDATE statements

You can use LIMIT *row\_count* to restrict the scope of an UPDATE statement.

If an UPDATE statement includes an ORDER BY clause, rows are updated in the order specified by the clause.

MySQL Example:

create table t\_upd (id int not null primary key, v varchar(8) not null);

insert t\_upd values (1,'A'),(2,'B'),(3,'C');

update t\_upd set id=id+1; -- Error Code : 1062 Duplicate entry '2' for key 1

update t\_upd set id=id+1 order by id desc; -- 3 row(s)affected

update t\_upd set v=concat(v,'+',v) limit 1; -- 1 row(s)affected

Solution:

Emulate the LIMIT clause by using the TOP clause of the UPDATE statement.

The logic of UPDATE statements with LIMIT and ORDER BY clauses should be reviewed.

SQL Server Example:

create table t\_upd (id int not null primary key, v varchar(8) not null)

insert t\_upd values (1,'A')

insert t\_upd values (2,'B')

insert t\_upd values (3,'C')

update t\_upd set id=id+1 -- 3 row(s) affected

update top (1) t\_upd set v=v+'+'+v -- 1 row(s) affected

###### DELETE statement

Issue: Multi-table DELETE

For multiple-table syntax, DELETE deletes from each *tbl\_name* the rows that satisfy the conditions.

MySQL Example:

create table del\_a (id int not null, v varchar(8) not null);

create table del\_b (id int not null, v varchar(8) not null);

insert del\_a values (1,'A'),(2,'B'),(3,'C');

insert del\_b values (1,'C'),(2,'B'),(3,'A');

-- deletes row with id=3 from both tables

delete a, b from del\_a a, del\_b b where a.id=b.id and a.v>b.v;

-- deletes row with id=3 only from table del\_a

delete a from del\_a a, del\_b b where a.id=b.id and a.v>b.v;

Solution:

Multi-table DELETE can be emulated by using separate DELETE statements for each table in aggregate with a table variable (or temporary table) for saving intermediate data.

SQL Server Example:

create table del\_a (id int not null, v varchar(8) not null)

create table del\_b (id int not null, v varchar(8) not null)

insert del\_a select 1,'A' union select 2,'B' union select 3,'C'

insert del\_b select 1,'C' union select 2,'B' union select 3,'A'

-- deletes row with id=3 from both tables

declare @temp table (id int, v varchar(8))

delete a output deleted.id, deleted.v into @temp

from del\_a a, del\_b b where a.id=b.id and a.v>b.v

delete b

from @temp a, del\_b b where a.id=b.id and a.v>b.v

-- deletes row with id=3 only from table del\_a

delete a from del\_a a, del\_b b where a.id=b.id and a.v>b.v

###### UPDATE statement

Issue: Multi-table UPDATE

For multiple-table syntax, UPDATE updates rows in each table named in *table\_references* that satisfy the conditions.

MySQL Example:

create table upd\_a (id int not null, v varchar(32) not null);

create table upd\_b (id int not null, v varchar(32) not null);

insert upd\_a values (1,'A'),(2,'B'),(3,'C');

insert upd\_b values (1,'C'),(2,'B'),(3,'A');

update upd\_b b, upd\_a a

set a.v=concat('Z',b.v,'+',b.v),

 b.v=concat('Z',a.v,'+',a.v)

where a.id=b.id and a.v>=b.v;

Solution:

Multi-table UPDATEs can be emulated by using separate UPDATE statements for each table in aggregate with a table variable (or temporary table) for saving intermediate data.

SQL Server Example:

create table upd\_a (id int not null, v varchar(32) not null)

create table upd\_b (id int not null, v varchar(32) not null)

insert upd\_a select 1,'A' union select 2,'B' union select 3,'C'

insert upd\_b select 1,'C' union select 2,'B' union select 3,'A'

declare @temp table (id int, v\_old varchar(32), v\_new varchar(32))

update b

set b.v='Z'+a.v+'+'+a.v

output deleted.id, deleted.v, inserted.v into @temp

from upd\_b b, upd\_a a

where a.id=b.id and a.v>=b.v;

update a

set a.v='Z'+b.v\_new+'+'+b.v\_new

from @temp b, upd\_a a

where a.id=b.id and a.v>=b.v\_old;

###### INSERT statement

Issue: INSERT statements and VALUES syntax

In MySQL, INSERT statements that use VALUES syntax can insert multiple rows.

MySQL Example:

create table tab\_ins (id int not null, n numeric(19,9) not null);

insert tab\_ins values (10,101.80),(20,120.90),(30,150.70);

Solution:

Create a separate INSERT for each row.

SQL Server Example:

create table tab\_ins (id int not null, n numeric(19,9) not null)

insert tab\_ins values (10,101.80)

insert tab\_ins values (20,120.90)

insert tab\_ins values (30,150.70)

Issue: INSERT...ON DUPLICATE KEY UPDATE syntax

In MySQL, if you specify ON DUPLICATE KEY UPDATE, and a row is inserted that would cause a duplicate value in a UNIQUE index or PRIMARY KEY, an UPDATE of the old row is performed.

Also, you can use the VALUES(*col\_name*) function in the UPDATE clause to refer to column values from the INSERT portion of the INSERT...UPDATE statement.

MySQL Example:

create table ins\_upd

 (a int not null primary key, b int not null, c int not null);

insert ins\_upd values (1,2,3),(2,3,4),(1,20,30)

 on duplicate key update c=values(c);

select \* from ins\_upd; -- 1 2 30, 2 3 4

Solution:

Check for the presence of added keys in the index and execute an INSERT statement for new keys and an UPDATE statement for existing keys.

Replace the VALUES function with its value.

SQL Server Example:

create table ins\_upd

 (a int not null primary key, b int not null, c int not null)

declare c cursor forward\_only static read\_only

 for select 1,2,3 union all select 2,3,4 union all select 1,20,30

declare @a int, @b int, @c int

open c

fetch c into @a, @b, @c

while @@fetch\_status=0

begin

if not exists (select top 1 0 from ins\_upd where a=@a)

 insert ins\_upd values (@a, @b, @c)

 else update ins\_upd set c=@c where a=@a

fetch c into @a, @b, @c

end

close c deallocate c

select \* from ins\_upd; -- 1 2 30, 2 3 4

Issue: INSERT and AUTO\_INCREMENT fields

The MySQL INSERT statement allows inserting into AUTO\_INCREMENT fields.

MySQL Example:

create table table\_autoinc

(id int not null auto\_increment, v varchar(32) null, key (id));

insert into table\_autoinc (v) values ('Value\_1');

insert into table\_autoinc (v) values ('Value\_2');

insert into table\_autoinc (v) values ('Value\_3');

insert into table\_autoinc (id, v) values (40,'Value\_4');

insert into table\_autoinc (id, v) values (50,'Value\_5');

insert into table\_autoinc (id, v) values (40,'Value\_4\_2');

insert into table\_autoinc (id, v) values (40,'Value\_4\_3');

insert into table\_autoinc (v) values ('Value\_6');

insert into table\_autoinc (v) values ('Value\_7');

select \* from table\_autoinc; -- id: 1,2,3,40,50,40,40,51,52

Solution:

This functionality can be emulated by using SET IDENTITY\_INSERT statements.

SQL Server Example:

create table table\_autoinc

(id int not null identity(1, 1), v varchar(32) null)

create index idx\_table\_autoinc on table\_autoinc (id)

insert into table\_autoinc (v) values ('Value\_1');

insert into table\_autoinc (v) values ('Value\_2');

insert into table\_autoinc (v) values ('Value\_3');

set identity\_insert table\_autoinc on

insert into table\_autoinc (id, v) values (40,'Value\_4');

insert into table\_autoinc (id, v) values (50,'Value\_5');

insert into table\_autoinc (id, v) values (40,'Value\_4\_2');

insert into table\_autoinc (id, v) values (40,'Value\_4\_3');

set identity\_insert table\_autoinc off

insert into table\_autoinc (v) values ('Value\_6');

insert into table\_autoinc (v) values ('Value\_7');

select \* from table\_autoinc; -- id: 1,2,3,40,50,40,40,51,52

Issue: Expression in INSERT VALUES syntax

An expression in INSERT VALUES syntax can refer to any column that was set earlier in a value list.

MySQL Example:

create table ins\_expr (a float not null, b float not null);

insert ins\_expr values (sin(4),abs(a));

Solution:

Replace the column reference with its value.

SQL Server Example:

create table ins\_expr (a float not null, b float not null);

insert ins\_expr values (sin(4),abs(sin(4)));

Issue: Inserting explicit and implicit default values

If both the column list and the VALUES list are empty, INSERT creates a row with each column set to its default value: INSERT INTO *tbl\_name* () VALUES();

MySQL Example:

create table ins\_def (a int null, b int not null, c int default 1);

insert ins\_def values (); -- insert ins\_def () values ();

select \* from ins\_def; -- NULL 0 1

Solution:

No solution yet.

###### REPLACE statement

Issue: REPLACE statement

A MySQL REPLACE statement works exactly like an INSERT statement, except that if an old row in the table has the same value as a new row for a PRIMARY KEY or a UNIQUE index, the old row is deleted before the new row is inserted.

MySQL Example:

create table tab\_repl

 (a int not null primary key, b int not null, c int not null);

replace tab\_repl values (1,2,3),(2,3,4),(1,20,30);

select \* from tab\_repl; -- 1 20 30, 2 3 4

Solution:

Check for the presence of added keys in the index and execute an INSERT statement for new keys and an UPDATE statement for existing keys, or delete existing keys before inserting.

SQL Server Example:

create table tab\_repl

 (a int not null primary key, b int not null, c int not null)

declare c cursor forward\_only static read\_only

 for select 1,2,3 union all select 2,3,4 union all select 1,20,30

declare @a int, @b int, @c int

open c

fetch c into @a, @b, @c

while @@fetch\_status=0

begin

 delete from tab\_repl where a=@a;

 insert tab\_repl values (@a, @b, @c)

 fetch c into @a, @b, @c

end

close c deallocate c

select \* from tab\_repl; -- 1 20 30, 2 3 4

###### SELECT statement

Issue: DISTINCTROW keyword

MySQL supports the DISTINCTROW keyword in SELECT statements.

Solution:

Replace the MySQL DISTINCTROW keyword with the SQL Server DISTINCT keyword.

Issue: References in ORDER BY clause

An ORDER BY clause can refer to fields missed in the SELECT list when the DISTINCT clause is used.

MySQL Example:

create table tab\_dist (a int, b int);

insert tab\_dist values (1,30),(2,20),(3,40),(4,20),(5,10);

select distinct b from tab\_dist order by a desc; -- 10 40 20 30

Solution:

This behavior can be emulated using GROUP BY clause.

SQL Server Example:

create table tab\_dist (a int, b int);

insert tab\_dist values (1,30)

insert tab\_dist values (2,20)

insert tab\_dist values (3,40)

insert tab\_dist values (4,20)

insert tab\_dist values (5,10)

select b from tab\_dist group by b order by min(a) desc;

Issue: DUAL table

In MySQL, you can specify DUAL as a dummy table name in situations where no tables are referenced.

MySQL Example:

select curdate();

select curdate() from dual;

select count(\*) from dual; -- 1

Solution:

Usually you can ignore FROM DUAL clauses. However, a SELECT COUNT(\*) FROM DUAL statement should be converted to SELECT 1.

Issue: SELECT...FROM...PROCEDURE syntax

In MySQL, you can define a procedure in Visual C++® that can access and modify the data in a query before it is sent to the client.

Solution:

No solution yet.

###### SELECT…INTO and LOAD DATA INFILE statements

Issue: SELECT…INTO *variable* statement

MySQL supports SELECT…INTO *variable* statements.

MySQL Example:

select max(host), max(user) into @h, @u from mysql.user;

Solution:

Define variable assignment in the field list.

SQL Server Example:

select @h=max(host), @u=max([user]) from mysql.dbo.[user];

Issue: SELECT...INTO OUTFILE and LOAD DATA INFILE statements

MySQL uses SELECT...INTO OUTFILE statements to write data from a table to a file and LOAD DATA INFILE statements to read the file back into a table.

MySQL Example:

select \* into outfile "d:\\in.out\\table\_gh.dat" from gh;

delete from gh;

load data infile "d:/in.out/table\_gh.dat" into table gh;

Solution:

The SQL Server **bcp** utility allows both writing tables to a file and loading the contents of a file into tables.

SQL Server Example:

exec xp\_cmdshell 'bcp "select \* from ATest.dbo.gh" queryout "d:\table\_gh.dat" -c -T'

delete from gh

exec xp\_cmdshell 'bcp ATest.dbo.gh in "d:\table\_gh.dat" -k -E -c -T'

###### GROUP BY, HAVING, and ORDER BY clauses

Issue: Column references in GROUP BY and HAVING clauses

In MySQL, columns from the SELECT list can be referenced in a GROUP BY or HAVING clause by using column aliases or by positions (only in GROUP BY).

MySQL Example:

create table tab\_alias (field\_a int, field\_b int);

insert tab\_alias values (1,1),(1,2),(1,3),(2,1),(2,2);

select field\_a as a, count(\*) from tab\_alias group by a order by a desc;

-- 2 2, 1 3

select field\_b as b, count(\*) from tab\_alias group by 1 order by 1 desc;

-- 3 1, 2 2, 1 2

Solution:

Alias and column position references in GROUP BY and HAVING clauses should be changed with referenced fields.

SQL Server Example:

create table tab\_alias (field\_a int, field\_b int)

insert tab\_alias select 1,1 union all select 1,2 union all select 1,3 union all select 2,1 union all select 2,2

select field\_a as a, count(\*) from tab\_alias group by field\_a order by a desc

select field\_b as b, count(\*) from tab\_alias group by field\_b order by 1 desc

Issue: GROUP BY sorting

In MySQL, if you use GROUP BY, output rows are sorted according to the GROUP BY columns as if you had an ORDER BY for the same columns.

MySQL extends the GROUP BY clause so that you can also specify ASC and DESC after columns named in the clause.

MySQL Example:

select help\_category\_id, count(\*)

 from mysql.help\_topic group by help\_category\_id desc

Solution:

In SQL Server, add the ORDER BY clause for sorting.

 Issue: ORDER BY NULL syntax

To avoid the overhead of sorting that GROUP BY produces in MySQL, the ORDER BY NULL clause is used.

MySQL Example:

select host, count(\*) from mysql.user group by host order by null

Solution:

This syntax can be ignored.

Issue: SELECT and ORDER BY clauses can have fields without aggregation but are not presented in GROUP BY

MySQL extends the use of GROUP BY to allow selecting fields that are not mentioned in the GROUP BY clause.

A similar MySQL extension applies to the HAVING clause. The SQL standard does not allow the HAVING clause to name any column that is not found in the GROUP BY clause if it is not enclosed in an aggregate function. MySQL allows the use of these columns to simplify calculations.

This extension assumes that the non-grouped columns have the same group-wise values. Otherwise, the result is indeterminate.

MySQL Example:

create table customer (custid int, name varchar(32));

insert customer values (1,'Customer\_A');

insert customer values (2,'Customer\_B');

insert customer values (3,'Customer\_C');

create table orde (custid int, payments numeric(19,2));

insert orde values (1,50.80);

insert orde values (1,140.84);

insert orde values (2,32.80);

select orde.custid, customer.name, max(payments)

from orde, customer

where orde.custid = customer.custid

group by orde.custid;

Solution:

SQL Server does not support queries whose fields are present in the SELECT list or ORDER BY clause without aggregation, but are missing in GROUP BY clause. Modify the query by including these fields in the GROUP BY clause.

Issue: HAVING clause without GROUP BY clause

In MySQL you can use a HAVING clause without a GROUP BY clause.

MySQL Example:

create table tab\_hav (class varchar(128), amount int, date datetime);

insert tab\_hav values ('PRINTER',2,'20061215');

insert tab\_hav values ('SCANNER',3,'20070123');

insert tab\_hav values ('FAX',5,'20070918');

insert tab\_hav values ('PRINTER',1,'20070921');

insert tab\_hav values ('PHONE',4,'20070308');

insert tab\_hav values ('SCANNER',2,'20070514');

insert tab\_hav values ('PRINTER',3,'20071011');

select \* from tab\_hav having sum(amount)=20; -- 1 row

select \* from tab\_hav having max(amount)=20; -- 0 row

select \* from tab\_hav having max(amount)=5; -- 1 row

Solution:

To emulate this functionality, convert the HAVING clause to a WHERE clause and use a subquery to calculate table aggregate functions.

###### JOINs

Issue: JOIN...USING syntax

The USING (*column\_list*) clause names a list of columns that must exist in both tables. If tables a and b both contain columns c1, c2, and c3, the following join compares corresponding columns from the two tables: a LEFT JOIN b USING (c1,c2,c3)

MySQL Example:

create table tab\_value

 (key\_a char(8), key\_b char(8), key\_c char(8), value int);

create table tab\_subvalue

 (key\_a char(8), key\_b char(8), key\_c char(8), subvalue int);

insert tab\_value values ('A','A','A',1),('B','D','E',2),('X','Y','Z',3);

insert tab\_subvalue

 values ('A','A','A',100),('A','A','A',120),('B','D','M',200),

 ('X','Y','Z',318),('X','Y','Z',350);

select value, subvalue

from tab\_value v join tab\_subvalue sv using (key\_a,key\_b,key\_c)

order by value, subvalue;

select value, subvalue

from tab\_value v join tab\_subvalue sv using (key\_a,key\_b)

order by value, subvalue;

Solution:

Replace the USING clause with ON and set the condition by all joined fields.

SQL Server Example:

select value, subvalue

from tab\_value v

 join tab\_subvalue sv on v.key\_a=sv.key\_a and v.key\_b=sv.key\_b

 and v.key\_c=sv.key\_c

order by value, subvalue

select value, subvalue

from tab\_value v

 join tab\_subvalue sv on v.key\_a=sv.key\_a and v.key\_b=sv.key\_b

order by value, subvalue

Issue: CROSS JOIN and INNER JOIN

In MySQL, CROSS JOIN is a syntactic equivalent of INNER JOIN (they can replace each other).

MySQL Example:

select value, subvalue

from tab\_value v inner join tab\_subvalue sv

order by value, subvalue;

select value, subvalue

from tab\_value v cross join tab\_subvalue sv on v.key\_a=sv.key\_a

order by value, subvalue;

Solution:

MySQL INNER join can be used without join conditions (ON …). In this case it works as CROSS JOIN.

MySQL CROSS join can be used with join conditions (ON …). In this case it works as INNER JOIN.

SQL Server Example:

select value, subvalue

from tab\_value v cross join tab\_subvalue sv

order by value, subvalue

select value, subvalue

from tab\_value v inner join tab\_subvalue sv on v.key\_a=sv.key\_a

order by value, subvalue

Issue: STRAIGHT\_JOIN

STRAIGHT\_JOIN is identical to JOIN, except that the left table is always read before the right table. This can be used for those (few) cases in which the join optimizer puts the tables in the wrong order.

MySQL Example:

select value, subvalue

from tab\_value v straight\_join tab\_subvalue sv

order by value, subvalue;

select value, subvalue

from tab\_value v straight\_join tab\_subvalue sv on v.key\_a=sv.key\_a

order by value, subvalue;

Solution:

MySQL STRAINT\_JOIN is an optimization issue and can be replaced with an INNER or CROSS join in most cases.

SQL Server Example:

select value, subvalue

from tab\_value v cross join tab\_subvalue sv

order by value, subvalue

select value, subvalue

from tab\_value v inner join tab\_subvalue sv on v.key\_a=sv.key\_a

order by value, subvalue

Issue: NATURAL JOIN

The NATURAL [LEFT] JOIN of two tables is defined to be semantically equivalent to an INNER JOIN or a LEFT JOIN with a USING clause that names all columns that exist in both tables.

MySQL Example:

select value, subvalue

from tab\_value v natural join tab\_subvalue sv

order by value, subvalue;

Solution:

If the joined tables have columns with the same names, convert the NATURAL join to an INNER join by these columns. Otherwise, convert a NATURAL join as a CROSS join.

SQL Server Example:

select value, subvalue

from tab\_value v

 join tab\_subvalue sv on v.key\_a=sv.key\_a and v.key\_b=sv.key\_b

 and v.key\_c=sv.key\_c

order by value, subvalue

###### Subqueries

Issue: Row subqueries

MySQL allows row subqueries. A row subquery is a subquery variant that returns a single row and can thus return more than one column value.

MySQL Example:

**A:** select \* from gh where (id, value) = row(1, 'A');

**B:** select \* from gh

 where (id, value) = (select subid, value from gj where gj.id=gh.id);

Solution:

Rewrite MySQL statements that have row subqueries by using the logical AND operator and EXISTS condition.

SQL Server Example:

**A:** select \* from gh where id = 1 and value = 'A'

**B:** select \* from gh

 where exists (select 1 from gj

 where gj.id=gh.id and gj.subid=gh.id and gj.value=gh.value)

###### Prepared statements

Issue: Server-side prepared statements

MySQL 5.1 supports server-side prepared statements. The scope of a prepared statement is the client session within which it is created.

PREPARE stmt\_name FROM preparable\_stmt

EXECUTE stmt\_name [USING @var\_name [, @var\_name] ...]

{DEALLOCATE | DROP} PREPARE stmt\_name

Example:

 create procedure ProcPrepare ()

 begin

 execute prep\_stmt using @a, @b;

 end

prepare prep\_stmt from 'select sqrt(pow(?,2) + pow(?,2)) as hypotenuse';

set @a = 3, @b = 4;

call procprepare();

deallocate prepare prep\_stmt;

Solution:

No solution yet.

###### DO command

Issue: DO syntax

The MySQL DO command executes expressions but does not return any results. In most respects, DO is shorthand for SELECT *expr*, ..., but has the advantage that it is slightly faster when you do not care about the result.

Example 1:

select @a:=200; -- sets @a and returns 200

select @a; -- returns 200

do @a:=300; -- sets @a

select @a; -- returns 300

Example 2:

create table TableDO (d int not null);

 create function func\_do (par\_d int) returns int

 begin

 delete from TableDO where d=par\_d;

 return row\_count();

 end

insert TableDO values (1), (2), (3);

do func\_do(2);

select d from TableDO; -- 1, 3

do @r:=func\_do(1);

select @r, d from TableDO; -- 1 3

Solution:

No solution yet.

###### HANDLERs

Issue: HANDLER interface

MySQL supports the HANDLER interface for reading table data.

HANDLER tbl\_name OPEN [ AS alias ]

HANDLER tbl\_name READ index\_name { = | >= | <= | < } (value1,value2,...)

 [ WHERE where\_condition ] [LIMIT ... ]

HANDLER tbl\_name READ index\_name { FIRST | NEXT | PREV | LAST }

 [ WHERE where\_condition ] [LIMIT ... ]

HANDLER tbl\_name READ { FIRST | NEXT }

 [ WHERE where\_condition ] [LIMIT ... ]

HANDLER tbl\_name CLOSE

The HANDLER statement provides direct access to table storage engine interfaces. It is available for MyISAM and InnoDB tables.

Example:

HANDLER TableA OPEN;

HANDLER TableA READ FIRST LIMIT 100;

HANDLER TableA CLOSE;

Solution:

No solution yet.

###### MODIFIERs

Issue: Modifiers (LOW\_PRIORITY, DELAYED, HIGH\_PRIORITY, QUICK, IGNORE) in DML statements

These MySQL modifiers enable you to apply something similar to an isolation level to each statement separately and to manage error raising.

Solution:

No solution yet.

##### Transactional and Locking Statements

This section discusses the main differences between MySQL and SQL Server 2005 locking and transaction control statements—starting, committing and rolling back, table locking, working with isolation levels, and AUTOCOMMIT mode.

###### BEGIN TRANSACTION statements

Issue: Different begin transaction syntax

MySQL and SQL Server use different syntax to start a transaction.

Solution:

Replace MySQL START TRANSACTION statements in batches and routines and BEGIN/BEGIN WORK statements in batches by using the SQL Server BEGIN TRANSACTION statement.

Issue: Begin transaction statements implicitly commit current transaction

The MySQL begin transaction statement implicitly commits the current transaction.

Solution:

Add a commit transaction statement with check @@TRANCOUNT state before the begin transaction statement.

Issue: Statements that cause an implicit commit

Each of the following MySQL statements (and any synonyms for them) implicitly end a transaction, as if you had done a COMMIT before executing the statement:

ALTER FUNCTION, ALTER PROCEDURE, ALTER TABLE, BEGIN, CREATE DATABASE, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE TABLE, DROP DATABASE, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP TABLE, LOAD MASTER DATA, LOCK TABLES, LOAD DATA INFILE, RENAME TABLE, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE TABLE, UNLOCK TABLES.

Solution:

Add a commit transaction statement with check @@TRANCOUNT state before statements that cause an implicit commit.

Issue: Statements that cannot be rolled back

Some MySQL statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, and those that create, drop, or alter tables or stored routines.

Solution:

No solution yet.

Issue: Storage engines that do not support transactions

Transactions do not influence operations with tables that are based on storage engines that do not support transactions.

MySQL Example:

create table tran\_x (i int not null) engine = innodb;

create table tran\_y (i int not null) engine = myisam;

start transaction;

insert tran\_x values (7);

insert tran\_y values (7);

rollback;

select \* from tran\_x; --

select \* from tran\_y; -- 7

Solution:

No solution yet.

###### END TRANSACTION statements

Issue: End transaction statement without a begin transaction statement

End transaction statements can be executed without prior begin transaction statements.

Solution:

Add IF (@@TRANCOUNT>0) condition before COMMIT / COMMIT WORK / ROLLBACK / ROLLBACK WORK statements.

Issue: CHAIN clause

The MySQL AND CHAIN clause causes a new transaction to begin as soon as the current one ends. The new transaction has the same isolation level as the just-terminated transaction.

Solution:

Replace the MySQL CHAIN clause with a SQL Sever BEGIN TRANSACTION statement after the END TRANSACTION statement.

Issue: RELEASE clause

The RELEASE clause causes the server to disconnect the current client connection after terminating the current transaction.

Solution:

No solution yet.

###### Named transaction SAVEPOINT statements

Issue: Different savepoint syntax

MySQL and SQL Server have different syntax for savepoint.

MySQL Example:

create table tran\_a (i int not null);

begin;

insert tran\_a values (1);

start transaction;

insert tran\_a values (2);

rollback and chain;

insert tran\_a values (3);

savepoint spoint;

insert tran\_a values (4);

rollback to savepoint spoint;

commit;

select \* from tran\_a; -- 1 3

Solution:

Replace MySQL SAVEPOINT *savepoint\_name* statements with SQL Server SAVE TRANSACTION *savepoint\_name* statements.

Replace MySQL ROLLBACK [WORK] TO SAVEPOINT *savepoint\_name* statements with SQL Server ROLLBACK TRANSACTION *savepoint\_name* statements.

SQL Server Example:

create table tran\_a (i int not null)

begin transaction

insert tran\_a values (1)

if (@@trancount>0) commit

begin transaction

insert tran\_a values (2)

if (@@trancount>0) rollback

begin transaction

insert tran\_a values (3)

save transaction spoint

insert tran\_a values (4)

rollback transaction spoint

if (@@trancount>0) commit

select \* from tran\_a -- 1 3

Issue: RELEASE SAVEPOINT statement

The MySQL RELEASE SAVEPOINT statement removes the named savepoint from the set of savepoints of the current transaction.

Solution:

No solution yet.

###### SET AUTOCOMMIT statements

Issue: SET AUTOCOMMIT statement

MySQL supports the SET AUTOCOMMIT statement.

MySQL Example:

create table tran\_auto (i int not null);

set autocommit = 1;

insert tran\_auto values (10);

insert tran\_auto values (20);

rollback;

set autocommit = 0;

insert tran\_auto values (30);

insert tran\_auto values (40);

rollback;

insert tran\_auto values (50);

commit;

select \* from tran\_auto; -- 10 20 50

Solution:

Replace SET AUTOCOMMIT = 1 with SET IMPLICIT\_TRANSACTIONS OFF.

Replace SET AUTOCOMMIT = 0 with SET IMPLICIT\_TRANSACTIONS ON.

SQL Server Example:

create table tran\_auto (i int not null)

set implicit\_transactions off

insert tran\_auto values (10)

insert tran\_auto values (20)

if (@@trancount>0) rollback

set implicit\_transactions on

insert tran\_auto values (30)

insert tran\_auto values (40)

if (@@trancount>0) rollback;

insert tran\_auto values (50)

if (@@trancount>0) commit

select \* from tran\_auto -- 10 20 50

###### LOCK TABLES and UNLOCK TABLES statements

Issue: LOCK TABLES and UNLOCK TABLES syntax

LOCK TABLES locks base tables (but not views) for the current thread.

UNLOCK TABLES releases any locks held by the current thread.

If a thread obtains a READ lock on a table, that thread (and all other threads) can only read from the table. If a thread obtains a WRITE lock on a table, only the thread holding the lock can write to or read the table. Other threads are blocked from writing to or reading the table until the lock is released.

Solution:

No solution yet.

###### SET TRANSACTION ISOLATION LEVEL statement

Issue: MySQL default transaction isolation level

In MySQL the default transaction isolation level is REPEATABLE READ. In SQL Server the default transaction isolation level is READ COMMITTED.

Solution:

No solution yet.

###### XA transaction statements

Issue: XA transactions

The MySQL XA transactions implementation is based on the X/Open CAE document [Distributed Transaction Processing: The XA Specification](http://www.opengroup.org/public/pubs/catalog/c193.htm).

The XA interface to a MySQL server consists of SQL statements that begin with the XA keyword.

Solution:

No solution yet.

##### Database Administration Statements

This section discusses converting MySQL administrative statements, including account management and table maintenance.

###### Account management statements

Issue: CREATE USER, DROP USER, GRANT, RENAME USER, REVOKE, SET PASSWORD statements

MySQL and SQL Server have different syntax for account management statements.

Solution:

No solution yet.

###### Table maintenance statements

Issue: ANALYZE TABLE, BACKUP TABLE, CHECK TABLE, CHECKSUM TABLE, OPTIMIZE TABLE, REPAIR TABLE, RESTORE TABLE statements

SQL Server does not have the identical statements.

Solution:

No solution yet.

###### SET statement

Issue: Multiple variable assignments in SET statements

In MySQL a SET statement can contain multiple variable assignments, separated by commas. A SQL Server SET statement can contain only one assignment to a variable.

MySQL Example:

 create procedure proc\_set\_var ()

 begin

 declare a, b int;

 set a=10, b=20;

 select a+b;

 end

call proc\_set\_var () -- 30

Solution:

Convert each variable assignment to a separate SET statement.

SQL Server Example:

 create procedure proc\_set\_var

 as

 begin

 declare @a int, @b int

 set @a=10

 set @b=20

 select @a+@b

 end

exec proc\_set\_var -- 30

 Issue: Server system variables (global variables)

The MySQL server maintains many system variables that indicate how it is configured. Each system variable has a default value. System variables can be set at server startup by using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the SET statement, which enables you to modify the operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

System variable values can be set globally at server startup by using options on the command line or in an option file.

Many system variables are dynamic and can be changed while the server runs by using the SET statement. To change a system variable with SET, refer to it as *var\_name*, optionally preceded by a modifier:

* To indicate explicitly that a variable is a global variable, precede its name by GLOBAL or @@global.
* To indicate explicitly that a variable is a session variable, precede its name by SESSION, @@session., or @@.
* LOCAL and @@local. are synonyms for SESSION and @@session.

If no modifier is present, SET changes the session variable.

A SET statement can contain multiple variable assignments, separated by commas.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, set it in an option file.)

To set a SESSION variable to the GLOBAL value or a GLOBAL value to the compiled-in MySQL default value, use the DEFAULT keyword.

**See also:** [SQL Mode](#_SQL_Mode_(SQL_MODE)

Example:

create table table\_inc

(id int not null auto\_increment, unique key (id), v varchar(8) null);

insert table\_inc (v) values ('A');

insert table\_inc (v) values ('B');

insert table\_inc (v) values ('C');

set session auto\_increment\_increment = 7;

insert table\_inc (v) values ('D');

insert table\_inc (v) values ('E');

set session auto\_increment\_increment = default;

insert table\_inc (v) values ('F');

insert table\_inc (v) values ('G');

select id from table\_inc; -- 1 2 3 8 15 16 17

Solution:

No solution yet.

###### SHOW statement

Issue: SHOW syntax

SHOW has many forms that provide information about databases, tables, columns, or status information about the server.

**See also:** [DESCRIBE syntax](#DESCRIBE)

Solution:

No solution yet.

Issue: DESCRIBE syntax

DESCRIBE provides information about the columns in a table. It is a shortcut for SHOW COLUMNS FROM. These statements also display information for views.

Solution:

No solution yet.

###### Other administrative statements

Issue: CACHE INDEX, LOAD INDEX INTO CACHE, FLUSH, RESET, KILL statements

SQL Server does not have the identical statements.

Solution:

No solution yet.

##### Stored Procedures and Functions (Routines)

This section discusses differences between the SQL procedural extension language in MySQL and Microsoft SQL Server. This includes the creation and calling of stored procedures and functions, working with local variables, cursors, and the control of flow statements.

###### CALL statements

Issue: Syntax for calling procedures

MySQL uses the CALL statement to invoke a procedure.

MySQL supports expressions as call parameters.

MySQL Example:

create procedure proc\_case (s varchar(64),

 out s\_low varchar(64), out s\_up varchar(64))

begin

set s\_low:=lower(s), s\_up:=upper(s);

end

call proc\_case (date\_format(now(),'%D %M %Y'),@low,@up);

select @low, @up; -- 23rd october 2007, 23RD OCTOBER 2007

Solution:

Convert MySQL CALL statements to Transact‑SQL EXEC statements. Expressions in call parameters should be calculated with temporary variables before the statements. Include the OUTPUT keyword for output parameters.

SQL Server Example:

create procedure proc\_case (@s varchar(64),

 @s\_low varchar(64) out, @s\_up varchar(64) out) as

begin

select @s\_low=lower(@s), @s\_up=upper(@s)

end

declare @s varchar(64), @low varchar(64), @up varchar(64)

set @s=convert(varchar(64),getdate(),106)

exec proc\_case @s, @low output, @up output

select @low, @up; -- 23 oct 2007, 23 OCT 2007

###### Compound statements block

Issue: Empty compound statements

The empty compound statement (BEGIN END) is legal in MySQL but not in Transact‑SQL.

MySQL Example:

create procedure empty\_block(i int)

begin

select sin(i);

 begin

 end;

select cos(i);

end

Solution:

Ignore these statements.

Issue: Labeled compound statements

MySQL compound statements can be labeled. You cannot use end\_label unless begin\_label is also present. If both are present, they must have the same name.

MySQL Example:

create procedure lab\_comp()

begin

 s: begin

 select 'STEP 1'; -- displayed

 leave s;

 select 'STEP 2'; -- ignored

 end;

 select 'STEP 3'; -- displayed

end

Solution:

Emulate LEAVE behavior in a labeled compound statement by using the Transact‑SQL GOTO statement.

SQL Server Example:

create procedure lab\_comp as

begin

 begin

 select 'STEP 1'; -- displayed

 goto s;

 select 'STEP 2'; -- ignored

 end;

 s: select 'STEP 3'; -- displayed

end

###### Local variables

Issue: Declaring variables of the same type

MySQL allows declaring several variables of one type in a single statement.

MySQL Example:

declare x, y, z int;

Solution:

Declare the type of each variable in SQL Server.

SQL Server Example:

declare @x int, @y int, @z int

Issue: DECLARE with default

In MySQL, the DEFAULT clause provides a default value for the variable. The value can be specified as an expression.

MySQL Example:

declare x, y int default sqrt(225);

declare z int default 123;

Solution:

Initialize the variable after declaring it.

SQL Server Example:

declare @x int, @y int

select @x=sqrt(225), @y=sqrt(225)

declare @z int

select @z=123

Issue: Scope of local variables

In MySQL, the scope of a local variable is within the BEGIN...END block in which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable of the same name.

MySQL Example:

create procedure var\_scope()

begin

declare a, b int;

set a=5, b=7;

select a, b; -- 5 7

 begin

 declare a int;

 set a=9;

 select a, b; -- 9 7

 end;

select a, b; -- 5 7

end

Solution:

No solution yet.

Issue: SQL variable names can be the same as column names

If a SQL statement contains a reference to a column and a declared local variable with the same name, MySQL interprets the reference as the name of a variable.

MySQL Example:

create procedure var\_field()

begin

create temporary table if not exists vf (a int, b int);

insert vf values (1,1),(1,2),(1,3);

select a, b from vf; -- 1 1, 1 2, 1 3

 begin

 declare a int default 7;

 select a, b from vf; -- 7 1, 7 2, 7 3

 end;

end

Solution:

Interpret dual references during conversion as the name of a variable.

SQL Server Example:

create procedure var\_field as

begin

create table #vf (a int, b int)

insert #vf values (1,1)

insert #vf values (1,2)

insert #vf values (1,3)

select a, b from #vf -- 1 1, 1 2, 1 3

 begin

 declare @a int

 set @a=7

 select @a, b from #vf -- 7 1, 7 2, 7 3

 end;

end

###### Conditions and handlers

Issue: MySQL condition handling

MySQL manages conditions by defining handlers.

The DECLARE CONDITION statement specifies conditions that need specific handling.

The DECLARE HANDLER statement specifies handlers. Each handler handles one or more conditions. If one of these conditions occurs, the statement (compound statement) specified in the handler is executed.

Example:

create table TableCondition\_A (c\_a int not null);

create table TableCondition\_B (c\_b int not null);

create procedure ProcCondition (in par\_value int, inout par\_null\_error int)

begin

declare cond\_a condition for sqlstate value '23000';

-- Error: 1048 SQLSTATE: 23000 (ER\_BAD\_NULL\_ERROR)

-- Message: Column '%s' cannot be null

declare continue handler for cond\_a

 begin set par\_null\_error=par\_null\_error+1; end;

set par\_null\_error=0;

insert TableCondition\_A values (par\_value);

insert TableCondition\_B values (par\_value);

end

call ProcCondition (null, @err);

select @err; -- 2

call ProcCondition (100, @err);

select @err; -- 0

Solution:

No solution yet.

###### Cursors

Issue: "No Data" cursor state

MySQL requires a handler for conditions with SQLSTATE value 02000 to detect "No Data" cursor state.

MySQL Example:

create table t1 (id char(16), data int);

create table t2 (i int);

create table t3 (id char(16), data int);

insert t1 values ('A',65),('K',75),('Q',81),('S',83),('W',87);

insert t2 values (10),(100),(20),(200),(30);

create procedure curdemo()

begin

 declare done int default 0;

 declare a char(16);

 declare b, c int;

 declare cur1 cursor for select id, data from t1;

 declare cur2 cursor for select i from t2;

 declare continue handler for sqlstate '02000' set done = 1;

 open cur1;

 open cur2;

 repeat

 fetch cur1 into a, b;

 fetch cur2 into c;

 if not done then

 if b < c then

 insert into t3 values (a,b);

 else

 insert into t3 values (a,c);

 end if;

 end if;

 until done end repeat;

 close cur1;

 close cur2;

end

call curdemo();

select \* from t3; -- A 10, K 75, Q 20, S 83, W 30

Solution:

Use the Transact‑SQL @@FETCH\_STATUS variable to detect the status of the last cursor FETCH statement.

SQL Server Example:

create procedure curdemo as

begin

 declare @done int set @done=0

 declare @a char(16)

 declare @b int, @c int

 declare cur1 cursor forward\_only static read\_only

 for select id, data from t1;

 declare cur2 cursor forward\_only static read\_only

 for select i from t2;

 open cur1;

 open cur2;

 while @done=0

 begin

 fetch cur1 into @a, @b;

 if @@fetch\_status<>0 set @done = 1

 fetch cur2 into @c;

 if @@fetch\_status<>0 set @done = 1

 if @done<>1

 begin

 if @b < @c

 begin insert into t3 values (@a,@b); end

 else

 begin insert into t3 values (@a,@c); end

 end

 end;

 close cur1 deallocate cur1

 close cur2 deallocate cur2

end

###### Flow control constructs

Issue: IF statement

MySQL and SQL Server have different syntax for the IF statement.

MySQL Example:

if (1>2)

 then select 'A'; select 'B';

 elseif (2>3) then select 'C'; select 'D';

 elseif (3>4) then select 'E'; select 'F';

 else select 'G'; select 'H';

end if;

Solution:

The MySQL IF statement can be easily emulated in SQL Server.

SQL Server Example:

if (1>2)

 begin select 'A' select 'B' end

 else if (2>3) begin select 'C' select 'D' end

 else if (3>4) begin select 'E' select 'F' end

 else begin select 'G' select 'H' end

Issue: CASE statement

MySQL and SQL Server have different syntax for the CASE statement.

MySQL Example:

case int\_value

when 1 then select 'A'; select 'AA';

when 2 then select 'B';

when 1 then select 'A1'; select 'A2'; -- ignored

when 3 then select 'C';

else select 'NULL';

end case;

Solution:

CASE statements can be emulated by using SQL Server IF statements.

SQL Server Example:

if @int\_value=1 begin select 'A' select 'AA' end

 else if @int\_value=2 begin select 'B' end

 else if @int\_value=1 begin select 'A1' select 'A2' end

 else if @int\_value=3 begin select 'C' end

 else begin select 'NULL' end

Issue: LOOP and REPEAT statements

SQL Server does not have the identical statements.

MySQL Example:

declare i int;

set i=0;

m: loop

 set i:=i+1;

 if (sin(i) - cos(i) < 0) then leave m; end if;

 end loop;

select i; -- 4

set i=0;

repeat

 set i:=i+1;

until (sin(i) - cos(i) < -1)

end repeat;

select i; -- 5

Solution:

MySQL LOOP and REPEAT statements can be easily emulated by using WHILE statements in SQL Server.

SQL Server Example:

declare @i int;

set @i=0;

while 1=1

begin

 set @i=@i+1;

 if (sin(@i) - cos(@i) < 0) break;

end;

select @i; -- 4

set @i=0;

while 1=1

begin

 set @i=@i+1;

 if (sin(@i) - cos(@i) < -1) break;

end

select @i; -- 5

Issue: LEAVE and ITERATE statements

SQL Server does not have the identical statements.

MySQL Example:

create procedure proc\_goto(s varchar(64), a int, b int)

begin

 m1: loop

 if (a>b) then leave m1; end if;

 set s:=concat(substring(s,1,a-1),

 upper(substring(s,a,1)),substring(s,a+1));

 set a:=a+1;

 if (a>b) then iterate m1; end if;

 set a:=a+1;

 end loop;

 select s;

end

call proc\_goto ('abcdefghijklmnopqrstuvwxyz',5,10)

-- abcdEfGhIjklmnopqrstuvwxyz

Solution:

Emulate this behavior by using Transact‑SQL BREAK and CONTINUE statements.

SQL Server Example:

create procedure proc\_goto (@s varchar(64), @a int, @b int) as

begin

 while 1=1

 begin

 if (@a>@b) break

 set @s=substring(@s,1,@a-1)+

 upper(substring(@s,@a,1))+substring(@s,@a+1,len(@s));

 set @a=@a+1;

 if (@a>@b) continue

 set @a=@a+1;

 end

 select @s

end

exec proc\_goto 'abcdefghijklmnopqrstuvwxyz',5,10

-- abcdEfGhIjklmnopqrstuvwxyz

###### Routines

Issue: DML statements in functions

MySQL functions can contain DML statements. This is not supported in SQL Server.

Example:

create table TableFuncA (a int not null);

create table TableFuncB (b int not null);

create function new\_func\_a (par\_int int) returns int

begin

delete from TableFuncA where a=par\_int;

return row\_count();

end

insert TableFuncA values (10), (20), (20), (30), (30), (30),

(40), (40), (40), (40);

insert TableFuncB values (20), (40), (50);

select new\_func\_a(b)

from TableFuncB; -- 2 4 0

select \* from TableFuncA; -- 10 30 30 30

Solution:

No solution yet.

##### Triggers

This section explains how to convert MySQL triggers to SQL Server 2005 triggers.

Issue: FOR EACH ROW triggers

MySQL supports only FOR EACH ROW triggers, which are not supported in SQL Server.

MySQL Example:

create table t\_data (

id int not null primary key,

v varchar(128) not null, log\_date datetime not null);

create table t\_log (

id int null, action varchar(6) null,

v\_old varchar(128) null, v\_new varchar(128) null,

log\_date\_old datetime null, log\_date\_new datetime null);

create trigger trg\_data\_ins

after insert

on t\_data

for each row

begin

 declare a varchar(6);

 if (new.v!='') then set a:='INSERT'; else set a:='EMPTY'; end if;

 insert t\_log (id,action,v\_old,v\_new,log\_date\_old,log\_date\_new)

 values (new.id,a,null,new.v,null,new.log\_date);

end

insert t\_data

values (1,'A',now()),(2,'B',now()),(3,'',now()),(4,'C',now());

Solution:

FOR EACH ROW trigger, functionality can be emulated by using a SQL Server cursor.

SQL Server Example:

create trigger trg\_data\_ins

on t\_data

after insert

as

begin

declare @id int, @v varchar(128), @log\_date datetime

declare for\_each\_row cursor forward\_only static read\_only

 for select id, v, log\_date from inserted

declare @a varchar(6);

open for\_each\_row

fetch for\_each\_row into @id, @v, @log\_date

while @@fetch\_status = 0

begin

 if (@v!='') set @a='INSERT' else set @a='EMPTY';

 insert t\_log (id,action,v\_old,v\_new,log\_date\_old,log\_date\_new)

 values (@id,@a,null,@v,null,@log\_date);

 fetch for\_each\_row into @id, @v, @log\_date

end

close for\_each\_row

deallocate for\_each\_row

end

Issue: BEFORE triggers

MySQL supports BEFORE triggers. In MySQL triggers, the BEFORE keyword indicates that the trigger is invoked before the execution of the triggering statement.

Inside a trigger, you can refer to columns in the subject table (the table associated with the trigger) by using the aliases OLD and NEW. OLD.*col\_name* refers to a column in an existing row before it is updated or deleted. NEW.*col\_name* refers to the column of a new row to be inserted or an existing row after it is updated.

MySQL Example:

create trigger trg\_data\_upd

before update

on t\_data

for each row

begin

 set new.log\_date:=now();

 if (old.v='') then set new.v:=''; end if;

 insert t\_log (id,action,v\_old,v\_new,log\_date\_old,log\_date\_new)

 values (old.id,'UPDATE',old.v,new.v,old.log\_date,new.log\_date);

end

update t\_data set v=concat(v,'+',v);

Solution:

A BEFORE trigger can be emulated by using a SQL Server INSTEAD OF trigger.

SQL Server Example:

create trigger trg\_data\_upd

on t\_data

instead of update

as

begin

declare @id\_old int, @v\_old varchar(128), @log\_date\_old datetime

declare @id\_new int, @v\_new varchar(128), @log\_date\_new datetime

declare for\_each\_row cursor forward\_only static read\_only

 for select id, v, log\_date from deleted

open for\_each\_row

fetch for\_each\_row into @id\_old, @v\_old, @log\_date\_old

while @@fetch\_status = 0

begin

 select @id\_new=id, @v\_new=v, @log\_date\_new=log\_date

 from inserted where id=@id\_old

 set @log\_date\_new=getdate();

 if (@v\_old='') set @v\_new='';

 insert t\_log (id,action,v\_old,v\_new,log\_date\_old,log\_date\_new)

 values (@id\_old,'UPDATE',@v\_old,@v\_new,@log\_date\_old,@log\_date\_new);

 -- insted of -----------------------------------------

 update t\_data

 set v=@v\_new, log\_date=@log\_date\_new

 where id=@id\_old

 ------------------------------------------------------

 fetch for\_each\_row into @id\_old, @v\_old, @log\_date\_old

end

close for\_each\_row

deallocate for\_each\_row

end

##### SQL Mode (SQL\_MODE system variable)

Issue: Applying and operating in different SQL modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients. Modes define what SQL syntax MySQL supports and the kind of data validation checks it performs.

You change and retrieve the SQL mode in MySQL by using the **sql\_mode** system variable.

Example:

SET sql\_mode = '';

SELECT NOT 1 BETWEEN -5 AND 5; -- 0

SET sql\_mode = 'HIGH\_NOT\_PRECEDENCE';

SELECT NOT 1 BETWEEN -5 AND 5; -- 1 -- (NOT 1) BETWEEN -5 AND 5

Solution:

No solution yet.

#### Migrating MySQL System Functions

This section describes how to map MySQL system functions to equivalent SQL Server 2005 functions and provides solutions for converting MySQL functions.

##### Equivalent Functions

The following MySQL system functions are usable as is in SQL Server code:

ASCII, LEFT, LOWER, LTRIM, REPLACE, REVERSE, RIGHT, RTRIM, SOUNDEX, SPACE, SUBSTRING, UPPER, ABS, ACOS, ASIN, ATAN, ATAN2, CEILING, COS, COT, DEGREES, EXP, FLOOR, LOG, LOG10, PI, POWER, RADIANS, RAND, ROUND, SIGN, SIN, SQRT, TAN, DAY, MONTH, COALESCE, NULLIF, CAST, CONVERT.

##### Nonsupported Functions

The following MySQL functions cannot be easily emulated in SQL Server because of logical and physical organization and security model differences:

BENCHMARK, CHARSET, COERCIBILITY, COLLATION, CRC32, DATE\_ADD with INTERVAL, DATE\_SUB with INTERVAL, GET\_FORMAT, PERIOD\_ADD, PERIOD\_DIFF, SUBTIME, TIMESTAMP, TIMESTAMPADD, TIMESTAMPDIFF, MATCH, EXTRACTVALUE, UPDATEXML, GET\_LOCK, IS\_FREE\_LOCK, MASTER\_POS\_WAIT, RELEASE\_LOCK.

##### Emulated Functions

Issue: Functions that have a variable parameter count

The following functions have a variable parameter count in MySQL:

GREATEST(*value1*, *value2*, …)

LEAST(*value1*, *value2*,...)

INTERVAL(*N*, *N1*, *N2*, *N3*, ...)

CHAR(*N*, ... [USING *charset\_name*])

ELT(*N*, *str1*, *str2*, *str3*,...)

FIELD(*str*, *str1*, *str2*, *str3*, ...)

MAKE\_SET(*bits*, *str1*, *str2*,...)

Solution:

Functions that have a variable parameter count can be emulated by using the Transact‑SQL CASE function. Or, you can try to use the XML data type to pass data into an emulation function, but you must do an additional data conversion to and from XML format.

Issue: IF(*expr1*, *expr2*, *expr3*)

If *expr1* is TRUE (*expr1* <> 0 and *expr1* <> NULL) then IF() returns *expr2*, otherwise it returns *expr3*.

MySQL Example:

if(@a>@b, @a, @b-@a)

Solution:

Emulate this function by using the Transact-SQL CASE function.

SQL Server Example:

case when @a > @b then @a else @b - @a end

Issue: BIN(*N*)

Returns a string representation of the binary value of *N*.

Solution:

Emulate this function in Transact‑SQL by using string functions and bitwise operators.

Issue: BIT\_LENGTH(*str*)

Returns the length of the string *str* in bits.

Solution:

Emulate this function in Transact‑SQL by using the DATALENGTH function.

Issue: CONCAT(*str1*, *str2*, ….). CONCAT\_WS(*separator*, *str1*, *str2*, ...)

Returns the string that results from concatenating the arguments.

MySQL Example:

CONCAT('A','B','C'), CONCAT\_WS('#','A','B','C')

Solution:

Use the SQL Server plus operator (+) for string concatenation.

SQL Server Example:

'A'+'B'+'C', 'A'+'#'+'B'+'#'+'C'

Issue: CONV(*N*, *from\_base*, *to\_base*)

Converts numbers between different number bases.

Solution:

Use Transact‑SQL mathematical functions and bitwise operators to emulate this function.

Issue: EXPORT\_SET(*bits*, on, off [, *separator* [, *number\_of\_bits*]])

Returns a string such that for every bit set in the value bits, you get an on string and for every reset bit, you get an off string.

Solution:

Use Transact‑SQL mathematical functions and bitwise operators to emulate this function.

Issue: FIND\_IN\_SET(*str*, *strlist*)

Returns a value in the range of 1 to *N* if the string *str* is in the string list *strlist* consisting of *N* substrings.

Solution:

Use the Transact‑SQL CHARINDEX function to emulate this function.

Issue: FORMAT(*X*, *D*)

Formats the number *X* to a format like '#,###,###.##', rounded to *D* decimal places, and returns the result as a string.

Solution:

Use the Transact‑SQL ROUND and CONVERT functions to emulate this function.

Issue: HEX(*N\_or\_S*)

If *N\_or\_S* is a number, returns a string representation of the hexadecimal value of *N*, where *N* is a longlong (BIGINT) number. If *N\_or\_S* is a string, returns a hexadecimal string representation of *N\_or\_S* where each character in *N\_or\_S* is converted to two hexadecimal digits. UNHEX(*S*) performs the inverse operation of HEX(*S*).

Solution:

Emulate HEX(*N\_or\_S*) functionality by using Transact‑SQL string functions, convert functions, and bitwise operators.

Issue: INSERT(*str*, *pos*, *len*, *newstr*)

Returns the string *str*, with the substring beginning at position *pos* and *len* characters long replaced by the string newstr.

Solution:

Use the Transact‑SQL REPLACE or SUBSTRING functions to emulate this functionality.

Issue: LOAD\_FILE(*file\_name*)

Reads the file and returns the file contents as a string. SQL Server cannot read data from an external file into a variable.

Solution:

Emulate LOAD\_FILE(*file\_name*) by using bulk load statements or an extended stored procedure.

Issue: NOW()

Returns the current date and time.

MySQL Example:

NOW()

Solution:

Use the similar Transact‑SQL function, GETDATE.

SQL Server Example:

GETDATE()

Issue: REPEAT(*str*, *count*)

Returns a string consisting of the string *str* repeated *count* times.

MySQL Example:

REPEAT('A', 10)

Solution:

Use the similar Transact-SQL function, REPLICATE.

SQL Server Example:

REPLICATE('A', 10)

Issue: ISNULL(*expr*)

If *expr* is NULL, ISNULL() returns 1, otherwise it returns 0.

MySQL Example:

ISNULL(@a)

Solution:

Use the Transact‑SQL CASE function and IS NULL clause to emulate this functionality.

SQL Server Example:

CASE WHEN @a IS NULL THEN 1 ELSE 0 END

Issue: STRCMP(*expr1*, *expr2*)

Compares two strings.

Solution:

Try using Transact‑SQL comparison operators to emulate STRCMP(*expr1*, *expr2*).

Issue: CONVERT\_TZ(*dt*, *from\_tz*, *to\_tz*)

Converts a datetime value *dt* from the time zone given by *from\_tz* to the time zone given by *to\_tz* and returns the resulting value. SQL Server does not have time zone functionality.

Solution:

Time zone functionality can be emulated by using SQL Server CLR or extended stored procedures.

Issue: DATE\_FORMAT(*date*, *format*)

Formats the date value according to the format string. Transact‑SQL does not have a similar function.

Solution:

You can use Transact-SQL date, string, and convert functions to emulate DATE\_FORMAT(*date*, *format*) functionality.

Issue: FROM\_DAYS(*N*)

Given a day number *N*, returns a DATE value.

Solution:

Use the Transact‑SQL CONVERT function to emulate FROM\_DAYS(*N*).

Issue: MAKEDATE(*year*, *dayofyear*)

Returns a date, given *year*, and *dayofyear* values.

Solution:

Use the Transact‑SQL DATEADD function to emulate this function.

Issue: SEC\_TO\_TIME(*seconds*)

Returns the *seconds* argument, converted to hours, minutes, and seconds.

Solution:

Use Transact‑SQL arithmetic operators and convert functions to emulate this function.

Issue: TIME\_TO\_SEC(*time*)

Returns the *time* argument, converted to seconds.

Solution:

Use Transact‑SQL arithmetic operators and string functions to emulate this function.

Issue: TO\_DAYS(*date*)

Given a *date*, returns the day number (the number of days since year 0).

Solution:

Use the Transact‑SQL CONVERT function to emulate this function.

Issue: BIT\_COUNT(*N*)

Returns the number of bits that are set in the argument *N*.

Solution:

Emulate the MySQL BIT\_COUNT(*N*) function in Transact‑SQL by using string functions and bitwise operators.

Issue: Encryption and compression functions

AES\_ENCRYT. AES\_DECRYPT. COMPRESS. UNCOMPRESS. ENCODE. DECODE. DES\_ENCRYPT. DES\_DECRYPT. ENCRYPT. MD5. OLD\_PASSWORD. PASSWORD. SHA. SHA1. UNCOMPRESSED\_LENGTH.

Solution:

Emulate this functionality by using SQL Server security and cryptographic functions.

Issue: LAST\_INSERT\_ID()

Returns the first automatically generated value that was set for an AUTO\_INCREMENT column by the most recent INSERT or UPDATE statement to affect the column.

Solution:

Use the Transact-SQL @@IDENTITY or SCOPE\_IDENTITY functions to emulate LAST\_INSERT\_ID().

Issue: DEFAULT(*column*)

Returns the default value for a table column.

Solution:

Use a system view data to emulate DEFAULT(*column*).

Issue: INET\_ATON(*expr*)

Given the dotted-quad representation of a network address as a string, returns an integer that represents the numeric value of the address. INET\_NTOA(*expr*). Given a numeric network address (4 or 8 byte), returns the dotted-quad representation of the address as a string.

Solution:

Use Transact‑SQL arithmetic operators and string functions to emulate these functions.

Issue: GROUP\_CONCAT(*expr*)

This function returns a string result with the concatenated non-NULL values from a group.

Solution:

This function can be emulated by using Transact‑SQL code as in the following example:

declare @v varchar(max)

set @v=''

select @v=@v+','+isnull(field\_a,'') from table\_1

select substring(@v,2,len(@v))

Issue: INSTR(*str*, *substr*), POSITION(*substr* IN *str*)

Returns the position of the first occurrence of substring *substr* in string *str*. LOCATE(*substr*, *str* [, *pos*]). Returns the position of the first occurrence of substring *substr* in string *str*, starting at position *pos*.

Solution:

Use the CHARINDEX function to emulate this functionality.

#### Conclusion

From this migration guide you learned the differences between MySQL and SQL Server 2005 database platforms, and the steps necessary to convert a MySQL database to SQL Server.

For more information:

* [SQL Server Web site](http://www.microsoft.com/sql/default.mspx)
* [SQL Server TechCenter](http://technet.microsoft.com/en-us/sqlserver/default.aspx)
* [SQL Server Developer Center](http://msdn2.microsoft.com/en-us/sqlserver/default.aspx)

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 screenshots, clear writing, or another reason?
* Are you rating it low due to poor examples, fuzzy screenshots, unclear writing?

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