Caché ISQL Migration Guide

Version 2018.1
2019-09-20

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Caché ISQL Migration Guide
Caché Version 2018.1 2019-09-20

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This book describes how to migrate schemas and stored procedures from Informix ISQL and it will provide you with an understanding of the ISQL implementation in Caché.

The book addresses a number of topics:

- An Overview, which includes configuring ISQL and migrating source code and data.

For a detailed outline, see the Table of Contents.

When using Caché ISQL, you may find the following additional sources useful:

- *The Caché SQL Reference* provides details on individual SQL commands and functions, as well as information on the Caché SQL configuration settings, error codes, data types, and reserved words.
- *Using the Caché SQL Gateway* describes how to use the Caché SQL Gateway, which enables you to treat external tables as if they were native Caché tables.
- *Using Caché ODBC* describes how to use Caché ODBC, which enables you to access Caché tables via ODBC from external applications.
- *Using Java with Caché* includes information on connecting to Caché via the Caché JDBC driver.

For general information, see *Using InterSystems Documentation*.

1

Overview

Caché ISQL is an implementation of Informix ISQL which supports many of its features.

Caché ISQL also contains a few proprietary extensions not found in Informix ISQL.

This document will help you to quickly migrate schemas and stored procedures from Informix databases and it will provide you with an understanding of the ISQL implementation in Caché.

1.1 Getting Started

To migrate existing ISQL applications to Caché ISQL, you need to perform three operations:

- Configure Caché for ISQL
- Migrate ISQL source code
- Migrate ISQL data

1.2 ISQL Configuration Settings

Caché provides four configuration settings to aid in the migration of ISQL to Caché SQL. These configuration settings can be set either by using the Caché Management Portal or by using a direct global reference.

To set ISQL configuration using the Management Portal:

- From the main page of the Management Portal, select **System Administration**, **Configuration**, **SQL and Object Settings**, then **ISQL Compatibility Settings**.

  Here you can turn on or off **Support Delimited Identifiers** (the default is No), **Generate Trace Code** (the default is No), and **Return Result of Stored Procedure Call as Resultset** (the default is Yes). You can also specify a **Reserved Word Prefix** to be appended to an identifier when it conflicts with a Caché SQL reserved word. The purposes of these settings are described below.

  If you set one or more configuration options using the Management Portal, the **Informix SQL Settings** heading will be followed by an asterisk, indicating that changes have been made but not yet saved. You must press the **Save** button for configuration changes to take effect.

- From the main page of the Management Portal, select **System Administration**, **Configuration**, **SQL and Object Settings**, then **General SQL Settings**. From the SQL tab, you can set the **Default SQL Schema Name**. This is the system-wide default schema name (which maps to a package) for all unqualified DDL entities.
- From the main page of the Management Portal, select **System Administration**, **Configuration**, **SQL and Object Settings**, then **User-defined DDL Mappings**. You can use this option to map any needed user-defined data types.

To set ISQL configuration using direct global reference, or to determine the current ISQL configuration settings, you can use the following globals:

- ^%SYS("xsql","informix","DELIMIDENT") = 0 | 1 controls the delimited identifier setting. If 1, the Informix source is parsed with double quotes used as identifier delimiters instead of literal string identifiers.
- ^%SYS("xsql","informix","RESERVEDWORDPREFIX")=*prefix*. If set, then any column names that are reserved words will be resolved by adding *prefix* to the column name.
- ^%SYS("xsql","informix","TRACE") = 0 | 1. If 1, then TRACE code will be generated. Any procedure converted while TRACE is 1 will contain calls to %XSQL.Log for each deferred statement—one for %Prepare and one for the result. At runtime, any procedure that was converted with TRACE = 1 will log messages in a %XSQL.Log file. The log is unique for each process.
- ^%SYS("xsql","informix","FUNCTIONRETURNSSET") = 0 | 1. If 1, then a function that returns multiple values returns them as a result set.

1.2.1 Identifier Conflicts

Caché provides two alternative ways to resolve conflicts between ISQL identifiers (such as table names and column names) and Caché SQL reserved words.

If a reserved word prefix is configured, an identifier name that conflicts with a Caché SQL reserved word is automatically assigned the specified prefix. This prefixing is performed for any identifier that matches against the SQL reserved words list. You can add “reserved words” to this list (for the purpose of resolving ISQL identifiers) using the global variable ^%SYS("xsql","informix",word), where *word* is the word to be reserved, in uppercase letters. This ability to supplement the reserved word list is useful to maintain consistency between database systems that use different reserved word lists.

If a reserved word prefix is *not* configured and delimited identifier support is configured, the converter delimits an identifier that matches against the SQL reserved words list (and any supplements) by enclosing it with quotation marks. A delimited identifier can take any value, including a reserved word.

1.2.2 TRACE

XSQL deferred statements are dynamically prepared at runtime. The resulting implementation is cached to improve the performance of subsequent executions. Cached implementations are purged whenever any of the referenced objects are modified. The Informix ISQL converters supports a TRACE feature for compiled procedures that logs the deferred statement text, argument values, and any errors encountered.

1.2.2.1 Configuring Trace Support

The trace feature for Informix can be activated in three ways:

- Go to the Caché Management Portal: **[System] > [Configuration] > [SQL Settings] > [ISQL Compatibility Settings]**, and turning on the **Generate Trace Code** setting.
- Set **$ZUTIL(115,14,1)**.
- Invoke the ObjectScript macro $$$TraceOn

  .

1.2.2.2 TRACE Statements in Source Code

When Caché converts an ISQL procedure with trace configuration activated, basic support for trace is generated. TRACE statements within the procedure code determine trace behavior. The TRACE ON statement activates the trace feature and all generated traces will be logged. This trace feature is activated by default. The TRACE OFF statement deactivates the trace feature and no further messages will be logged. If the trace feature is activated, TRACE *expression* logs the value of *expression* to the current log.

1.2.2.3 TRACE Logging

Output from the trace is sent to a text file located in the same directory as the CACHE.DAT. This can be overridden by using the **%New()** method to instantiate a %XSQL.Log class object as %xsqlLog with a file name specified. For example: SET %xsqlLog = ##class(%XSQL.Log).%New("c:\mylogs\mytrace.log",1). The second argument to **%New()** is the *initialize* flag. If it is TRUE (1) and the log file already exists, then it is cleared. If *initialize* is FALSE (0) then trace information is appended to the existing log file, and a "log restarted..." message is written to the log. By default, log file names are qualified by the current job number.

If TRACE is active, a message is logged whenever a result set is added to the current context object. The result set is retrieved from the context object and the first ten rows copied to the log.

If TRACE is active, logging is performed for all embedded SQL / deferred SQL code execution in the converted Informix procedures. The SQL statement tracing contains the statement text, any arguments and their values, the SQLCODE value, %ROWCOUNT, and %msg following the run (if SQLCODE < 0), and the elapsed time it took to execute the SQL statement. For CURSOR based SELECTs, a trace is recorded for the query, the OPEN, and the CLOSE of the cursor.

To write a message to the trace log, invoke the ObjectScript macro $$$TraceMessage(text), where *text* is any quoted string.

1.2.3 Legacy Function Call Mode

The **Return Result of Stored Procedure Call as Resultset** configuration setting governs how functions that return multiple values should return those values. The default of "Yes" returns the values as a result set. This setting supports an ISQL legacy function call mode whereby any return values are returned in a single row result set instead of a return value and/or output parameters.

1.3 Migrating Source Code

The initial application migration is simple:

1. *Migrate the DDL/DML Script Files:* The following method imports all DDL/DML script file in a given directory. All files with the extension .sql in the directory will be imported.

```
$SYSTEM.SQL.DDLImportDir(DDLMode, directory, logfile, EOSDelimiter)
```

   DDLMode: Vendor from which the script file originated. This parameter is required. The only permitted value is `Informix`

   directory: The full path name of the directory to import. This parameter is required.

   logfile: Either the full path name of the log file to report errors in, or the number 1. This parameter is optional. The default is DDLImportDir.log in the directory loaded. If this parameter value is 1, a separate log file is generated for each file loaded. The name of each log file will be the same as the file imported, but with the extension .log instead of .sql.

EOSDelimiter: End of statement delimiter. This parameter is optional. Defaults to an appropriate value based on the value of %DDLMode.

For further details, see the *InterSystems Class Reference*.

2. Alternatively, you can invoke the **Informix()** method of the $System.SQL class to import the schema. This method supports CREATE TABLE, ALTER TABLE, CREATE INDEX, CREATE VIEW, SET OPTION, and GRANT statements. For further details, see the *InterSystems Class Reference*.

   If the ISQL source contains CREATE PROC statements, then a class method containing the CREATE PROC source is created. Caché places this class method in either an existing class or in a new class whose name is based on the schema and procedure name. If the procedure already exists, then the existing version is replaced by the new version. If a class matching the class name generated from the schema and procedure already exists, then this class name is used — if it was previously generated by the ISQL utility. If not, then a unique class name is generated, based on the schema and procedure name. The resulting class is compiled once the procedure has been successfully created.

   If logging is requested then the source statements are logged along with the name of the containing class, class method, and the formal arguments generated. Any errors encountered by the process are also reported in the log. If an error is detected during CREATE PROC processing, Caché deletes any new class that was generated for that procedure.

3. *Inspect the log file for errors:* Search by Error #. A summary count of errors and successful imports will appear at the end of the log. In most cases, errors can be worked around or addressed by using information found in this document.

   Cache provides the %XSQL.System.CacheMessageXRef class to cross-reference Informix error codes and messages and the corresponding Caché SQLCODE error codes and messages.

4. *Compile:* When you import DDL, table and view definition compilation is automatically performed. To compile other ISQL source code, it is best to use the command as follows:

   ```
   DO $SYSTEM.OBJ.CompileAll("-l")
   ```

   The lowercase “L” qualifier flag specifies that locking *is not* applied for the duration of the compile. For a full list of flag qualifiers, call `DO $SYSTEM.OBJ.ShowFlags()`.

When compiling ISQL methods, ObjectScript code is generated. There is no system-level support for native ISQL. It is best to maintain the methods in ISQL to retain the familiar look of the original stored procedures.

1.4 Migrating Informix Stored Procedures

Caché provides a converter to convert Informix SPL routines to Cache class methods. The resulting class methods contain ObjectScript code that replicates Informix SPL behavior, including temporary tables, deferred statement resolution (statements are prepared dynamically at runtime; no metadata validation occurs at class compile time), and exception handling. The Caché converter generates class methods whose formal spec consists of the SPL declared formal arguments. Functions that return more than one value have additional formal arguments that correspond to the declared return types. Iterator functions return a result set whose columns correspond to the declared return types.

The original Informix SPL source is retained in the class description for the generated class. The converter retains comments in the source in the converted output.

Optionally, the Informix SPL converter can log all converted routines, errors encountered, and the names of each the class method generated. This log is maintained in %XSQL.Log. A summary of all errors encountered during conversion is also placed at the end of the log.

The ObjectScript version of an SPL routine uses %XSQL deferred statements, private temporary tables, and an error message cross reference.

1.4.1 Temporary Tables

Informix temporary tables (CREATE TEMP TABLE) are implemented as Caché global private temporary tables. These are temporary tables that can only be accessed by the current process, and are defined until either explicitly dropped or the process terminates. They are created using the Caché SQL command CREATE GLOBAL PRIVATE TEMPORARY TABLE.

A GLOBAL PRIVATE TEMPORARY table is the same as a GLOBAL TEMPORARY table except that the table object is maintained in %processcontext instead of %sqlcontext. %processcontext is a ProcedureContext instance that is never new'ed or killed for the lifetime of the process. Temporary table objects referenced by %processcontext will therefore be in scope until dropped or the process ends. Dropping a temporary table may not cause the table to be physically dropped. If any active result sets reference the temporary table then it will remain active until those result set objects are closed (destructed). An attempt to create a temporary table that is still active due to active result sets will result in an error.

1.4.2 Procedure Calls and Iterator Functions

Informix supports four types of SPL routines:

- A procedure, which returns no value (SPL). Invoked from Caché using the DO command.
- A standard function, which return a single value (SPL RETURN). These are invoked from Caché using SET retval=function().
- A function that returns multiple values (SPL RETURN value1[,value2[,...]]). Informix SPL routines do not support output format arguments. Caché converts this to a class method with multiple output formal arguments. It is invoked using the DO command; output values are retrieved from the procedure context object and assigned to the declared INTO variables.
- An iterator function that returns multiple rows, each row containing one or more values (SPL RETURN...WITH RESUME). Caché implements this functionality as a Cache class method with multiple output formal arguments and with no return value. It is invoked using the DO command. It uses a result set class %ResultSet.InformixFunction to first insert successive rows into the result set, then reset the iterator to the first row of the result set to enable retrieval of multiple rows. Rows are retrieved from the first result set returned in the procedure context object. The columns in the result set correspond to the declared return types.

If the returned value(s) is longer than the defined size of the return variable, Caché (like Informix) automatically truncates the returned value(s).

A scalar function cannot return a result set. Caché issues an SQLCODE -432 when a function is called that expects a single return value but actually returns multiple rows. This corresponds to Informix error -686.

There is one significant difference in the flow with iterator functions. The Informix statement FOREACH EXECUTE calls a procedure that returns a result set. That result set can reference temporary objects that are only valid in the scope of the called procedure context. In the original Informix, a FOREACH EXECUTE resumes execution at the point of the previous RETURN WITH RESUME statement. The converted ObjectScript method executes the iterator function to completion and returns the results in a result set that is then iterated over by the FOREACH loop. The actual column values are retrieved directly from the result set without the called procedure context. The EXIT FOREACH closes the cursor and releases the lock on the last retrieved row.

1.4.3 Formal Arguments

Caché supports two settings for formal arguments: DEFAULTFORMAL and DEFAULTFORMALTYPE. DEFAULTFORMAL is an optional setting that contains the name of a dummy formal argument. DEFAULTFORMALTYPE is an optional setting containing the type of the DEFAULTFORMAL argument.

If DEFAULTFORMAL is set, then a formal argument with that name is generated for any procedure or function that does not declare any formal arguments. If DEFAULTFORMALTYPE is defined then it is the Informix type declaration used for the generated formal argument. If DEFAULTFORMALTYPE is not defined, it defaults to INTEGER.

1.5 Migrating the Data

In the Management Portal select **[System] > [SQL]**, then select the Data Migration Wizard.

1.5.1 Informix ROWID

The procedure conversion utility translates ISQL column references of ROWID to %ID. In Informix, the ROWID column is a hidden column in nonfragmented tables and in fragmented tables that were created with the WITH ROWIDS clause. The ROWID column is unique for each row, but it is not necessarily sequential. In Caché SQL it corresponds to %ID.

1.5.2 NULL and Empty String Handling

In Informix, the string data types CHAR and VARCHAR treat the empty string ("") as identical to a single blank space (" "). For all numeric, time and date data types, empty string ("") is treated as NULL.

When concatenating strings, concatenating NULL to *string* results in NULL. Concatenating an empty string to *string* has no effect on *string*.

1.5.3 String Trailing Blanks

In Caché strings trailing blanks are automatically truncated. This is true for both CHAR and VARCHAR data types. As this differs from standard Informix behavior, this may be an important consideration when migrating data.

1.5.4 DATETIME Data Type

Caché supports the Informix DATETIME data type by mapping it to the %Library.InformixTimestamp datatype class. Informix does not require the entire date or time to be present; it uses qualifiers to specify the range of components of the date and time. Caché supports syntax for specifying the largest and smallest qualifiers. The YEAR TO DAY qualifier maps as a DATE data type. The MINUTE TO SECOND qualifier maps as a TIME data type. All others qualifiers map as %Library.InformixTimeStamp. For example, YEAR TO SECOND and HOUR TO SECOND.

Caché supports DATETIME arithmetic. It supports addition and subtraction operations for DATE, TIME, TIMESTAMP, and INTERVAL.

1.5.5 SERIAL Data Type

Caché supports the Informix SERIAL data type by mapping it to the %Library.Counter datatype. When a field is defined as data type %Counter, during SQL INSERT Caché increases the counter value by 1 (one) if no value, a 0 (zero), or a non-numeric value is inserted for the field; and assigns the resulting counter value to the field. If a positive integer value is explicitly inserted into the field, insert the value, and, if necessary, advance the counter node so that the counter node value is not less than the specified field value.

There is no automatic creation of a UNIQUE constraint for fields defined with the %Counter type. If you want to ensure the value is UNIQUE, you must create your own UNIQUE index on the property.

Fields defined as %Counter fields may not be updated unless the old value of the field is 0 or NULL. This might be the case if you add a %Counter field to an existing table that already has data and you want to go back and populate the %Counter

field for the existing rows. Otherwise, attempting to update a %Counter field generates an SQLCODE -105 error and a %msg text.

The Caché SQL %Counter data type provides functionality similar to the Informix SERIAL data type. Informix limits you to a single SERIAL field per table, but Caché has no restriction on the number of %Counter fields defined per table.

1.6 ISQL Language Implementation

1.6.1 ISQL Comments and Annotations

ISQL supports the following comment types:

- `-- comment` (two hyphens) single line comment prefix.
- `{ comment }` multiple line comment.
- `/* comment */` multiple line comment.

Multiple line comments can be nested.

ISQL supports the following single-line annotation formats:

- `--!! annotation`
- `--## annotation`

If an annotation is embedded in a SELECT statement, this annotation is appended to the FROM clause after conversion to Caché SQL. For example:

```
FOREACH
    SELECT a,b,c
    INTO v_a, v_b, v_c
    FROM
       --!!%NOFLATTEN %NOTOPOPT
       my_table as f
    WHERE f.a IN ( "a") AND f.b = 88 AND f.c ="ZZ"
    ORDER BY a, c DESC, b DESC
```

Annotations can only be used in a SELECT statement.

1.6.2 MATCHES Operator

ISQL supports the MATCHES operator for pattern matching. It supports the * wildcard character, which can be used to represent 0 or more characters.

1.6.3 DEFINE var LIKE Command

The **DEFINE var LIKE column** statement defines the data type of a local variable based on the data type of a column. This command is supported if the column reference can be resolved at conversion time.

1.6.4 LET Command

The **LET** command is used to assign a value to a variable. It can be used to assign more than one variable, as follows:

```
LET a,b = c,d;
```

1.6.5 Supported Functions

The Informix compiler supports the following Informix functions:

- DATE
- CURRENT returns the current date in internal storage format (the date portion of the $HOROLOG value).
- LOWER
- REPLACE
- SUBSTR
- SUBSTRING
- TRIM (partial support)
- UPPER