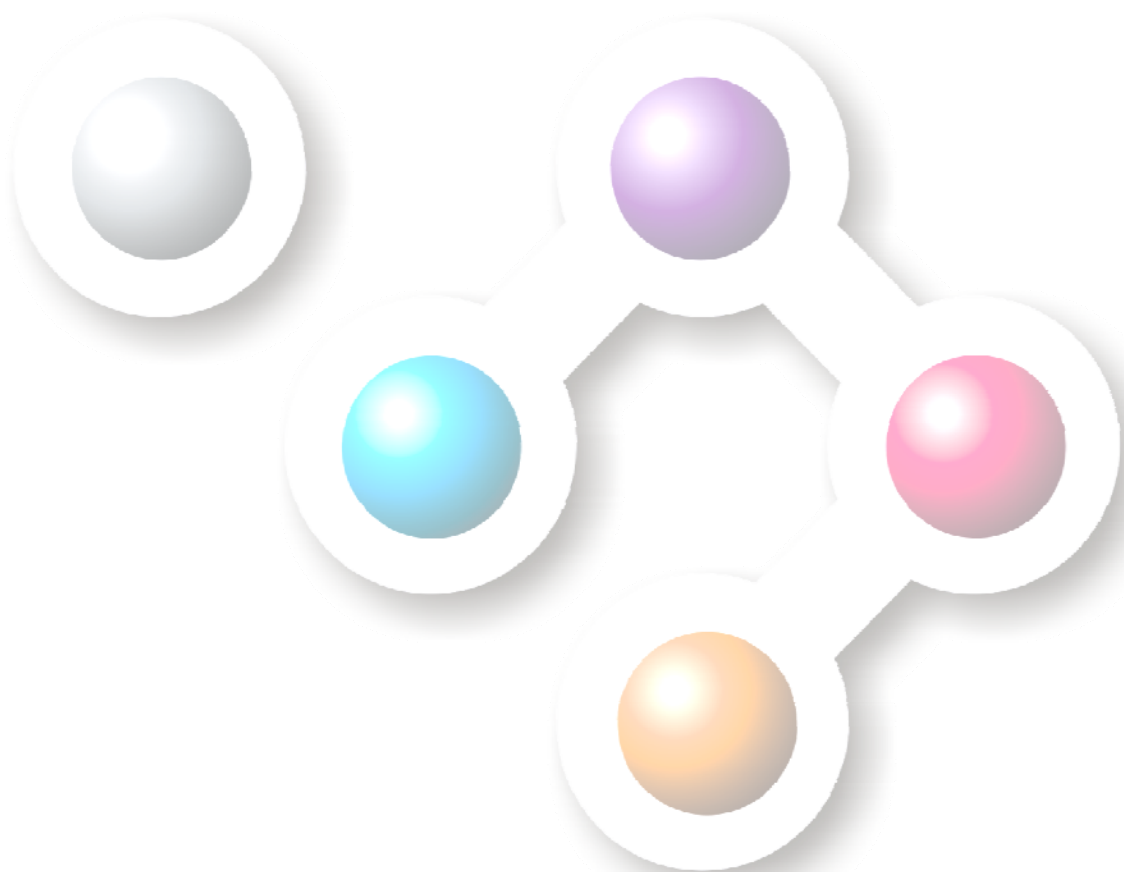


Lycia Development Suite



**Lycia Database Migration Guide for SQL Server
Versions 5 and 6 – January 2011**



Dynamic Database Interface

Migration Guide for SQL Server

Versions 5 and 6

January 2011

Part Number: 013-006-135-011

Querix Dynamic Database Guide for SQL Server

Copyright 2004-2011 Querix (UK) Limited. All rights reserved.

January 2011. Part Number: 013-006-135-011

Published by:

Querix (UK) Limited. 50 The Avenue, Southampton, Hampshire,
SO17 1XQ, UK

Publication history:

May 2004:	First edition titled 'Multiple Database Guide'
September 2004:	Second edition, including database
June 2005:	Updated for 'Hydra4GL' version 4.2
July 2005:	Retitled 'Hydra4GL' and Your Database
January 2008:	Dedicated for SQL Server, updated for versions 4.3 and retitled 'Database Migration Guide for SQL Server'
November 2008:	Updated for V4.3 release
February 2009:	Updated for V4.4 release
January 2011:	Updated for v5 and v6 releases

Last Updated:

January 2011

Documentation written by:

Sean Sunderland/Gemma Davis

Notices:

The information contained within this document is subject to change without notice. If you find any problems in the documentation please submit your comments to documentation@querix.com. No part of this document may be reproduced or transmitted, in any form or by any means, electronic or mechanical, for any purpose without the express permission of Querix (UK) Ltd.

Other products or company names used within this document are for identification purposes only, and may be trademarks of their respective owners.

Contents

CONTENTS.....	I
CHAPTER 1	3
INTRODUCTION	3
CHAPTER 2	5
PREREQUISITES FOR CONVERTING TO SQL SERVER.....	5
CHAPTER 3	7
INFORMIX AND SQL SERVER COMPARED	7
<i>General Comparison</i>	7
<i>Isolation Levels in Informix</i>	8
<i>Isolation Levels in SQL Server</i>	8
CHAPTER 4	9
CONNECTING TO MICROSOFT SQL SERVER.....	9
<i>Concepts</i>	9
<i>Environment Settings</i>	9
<i>Connecting to SQL Server through 4GL</i>	9
<i>Connecting to SQL Server from Windows</i>	12
Installing SQL Server	12
Adding a database	12
DSN Configuration	14
Creating a DSN	15
Authentication	16
<i>Connecting to SQL Server from Unix</i>	19
Configuring FreeTDS	19
freetds.conf	19
odbc.ini.....	19
CHAPTER 5	20
PREPARING FOR THE SQL SERVER CONVERSION	20
The Conversion Process	20
Migration of the Database Schema	20
Loading Table Data.....	21
Compiling 4GL Code	21
The qxexport Schema Extraction Tool	21
CHAPTER 6	23
INFORMIX COMPATIBILITY	23
<i>Unhandled Problems</i>	24
MATCHES in SQL	24
HOLD Cursors	24
DATETIME Precision.....	24
CREATE DATABASE.....	25

DROP INDEX Syntax.....	25
SPL Objects	25
<i>Problems Handled Through Emulation</i>	<i>26</i>
Fetching Floating DECIMAL Types	26
INTERVAL Data Types	26
ROWID	26
Tables Comprised solely of a SERIAL column.....	27
<i>Problems Fully Handled.....</i>	<i>28</i>
Function/Operator Naming	28
Reserved Words as Identifiers	28
DDL Syntax	29
Join Syntax.....	29
GROUP BY Syntax.....	29
Weak Typing.....	30
SQL Error Codes	30
Configurable Behaviour.....	31
APPENDIX A	35
THE 'LYCIA' DYNAMIC SQL TRANSLATOR	35
Introduction.....	35
Concepts	36
Database Drivers	36
Bind Variables	36
Buffer Variables	38
How Dynamic Translation Works.....	39
SQL interpretation.....	39
SQL Generation.....	39
Datatype Mappings	40
The qx_ \$schema table.....	40
Type Promotion	40
Benefits of Type Promotion.....	41
Areas where Type Promotion is used	41
APPENDIX B	43
CONFIGURING OPTIONS	43
Database Creation Options	43
Collation Sequences.....	44
APPENDIX C	45
DATATYPE MAPPINGS	45
APPENDIX D	49
COMMON CONVERSION ISSUES WHEN MIGRATING TO DIFFERENT VENDOR'S DATABASES.....	49
System Catalogues	49
Matches	49
Cursor Names	50
Isolation Levels	50
Stored Procedures	50
INDEX.....	51

CHAPTER 1

Introduction

Informix development tools, including Informix 4GL, Informix ESQL/C and Informix NewEra, were developed to interact solely with an Informix RDBMS. For this reason, applications developed using these tools are dependent upon the behaviour and personality of an Informix RDBMS.

Querix development tools, such as Lycia offer full compatibility with Informix development tools, whilst also offering the ability to operate seamlessly with a non-Informix RDBMS.

Any existing Informix application sources can be re-used and developed further using the 'Hydrda4GL' Development tools to open up for modern GUI screen interactions and databases such as 'Microsoft SQL Server'®.

Using Querix's unique dynamic SQL translation capabilities, the application vendor can still continue to develop using Informix style SQL without any re-training or code re-writes whilst being able to exploit Lycia's extended capabilities.

The result is that identical Informix application sources can operate against both Informix and non-Informix RDBMSs without compromising the investment in the original application sources.

Informix® 4GL, -ESQL-C and -NewEra® are development languages from Informix (now owned by IBM®) and for that reason, they could only interact with Informix databases. Querix™ development tools, such as Lycia™, offer full compatibility with Informix development tools but without the limitations of character mode screens and Informix RDBMSs.

This guide is organised into the steps taken to migrate your application. It should aid the Informix application developers to smoothly and easily migrate and deploy their application to utilise the MS SQL Server RDBMS. There are three key problem classes that the developer needs to take care of

- MATCHES operator in SQL

The Informix MATCHES operator is a non-ANSI extension to SQL that allows relational comparisons based on regular expressions. There is no functionally equivalent to this operator in Microsoft SQL Server.

- DATETIME precision

The Informix DATETIME datatype allows a precision up to 1/10000 seconds (FRACTION(5)). Microsoft SQL Server's TIMESTAMP datatype has a maximum precision of approximately 1/300 seconds. The precision of a DATETIME value will be rounded to 0.000, 0.003 or 0.007 seconds.

- Table locking

Unlike Informix, HOLD Cursors declared as FOR UPDATE will not release table locks on a transaction commit.

These issues are further explained in the chapter 'Informix Compatibility'.

Querix's Lycia offers SQL Server database migration with minimum effort and guarantees that the required source code changes will stay below 1% due to its unique dynamic SQL translation technology.

CHAPTER 2

Prerequisites for Converting to SQL Server

In order to use 'Lycia' with SQL Server, you must have the following:

- Microsoft SQL Server 2000 or later installation (For remote connection, some platforms will require the SQL Server connectivity tools to be installed.)
- A full installation of the 'Lycia' 4.5 application
- An understanding of 4GL programming
- Full access to the application's source code
- A working knowledge of the application

No Informix tools are needed



Microsoft has identified a potential issue when running many processes while using SQL Server on NT Servers. This issue is discussed in the Microsoft Knowledge Base article 824422.

<http://support.microsoft.com/default.aspx?scid=kb:en-us:824422>.

CHAPTER 3

Informix and SQL Server Compared

This chapter is intended to provide a brief overview of the similarities and differences between Informix (OnLine, Dynamic Server and Standard Engine) and Microsoft SQL Server.

General Comparison


Subject	Informix	SQL Server
Product Range	Informix Standard Engine Informix Dynamic Server (Enterprise, Workgroup & Express Editions) Informix Online (Extended, Standard & Personal Editions)	Enterprise Edition Standard Edition Workgroup Edition Developer Edition Express Edition
Storage	Informix Standard Engine uses a directory structure for physical data storage. Informix OnLine and Informix Dynamic Server both use a physical data file for the data storage of a server instance. Multiple databases can reside in one data file.	SQL Server separates physical storage over 2 or more physical data files (per database): - Primary Data File - Secondary Data File (optional) - Log File
SQL Compliance	SQL-92 (entry level) SQL-99 (partial) Proprietary extensions	SQL-92 (entry level) SQL-99 (partial) Proprietary extensions
Supported Connection Interfaces	Proprietary, ADO, OLEDB, ODBC, JDBC	ADO, OLEDB, ODBC, JDBC
User Authentication	Operating System Managed	Operating System & RDBMS Managed
Supported Isolation Levels	Dirty Read Committed Read Cursor Stability Repeatable Read	Read Committed Read Uncommitted Repeatable Read Serializable

Isolation Levels in Informix

Dirty Read	Does not place or honour table locks whilst reading data, and will read uncommitted data.
Committed Read (ANSI)	Places no locks on data being read, but only reads data that has been committed. This is the default isolation level.
Cursor Stability	Whilst reading data, the current row is share-locked. A share locked row cannot be exclusively locked.
Repeatable Read	Acquires a share lock on all rows being read for the duration of a transaction. A share locked row cannot be exclusively locked.

Isolation Levels in SQL Server

Read Committed (ANSI)	Places share locks on data being read. Default isolation level.
Read Uncommitted (ANSI)	Does not place or honour table locks whilst reading data, and will read uncommitted data.
Repeatable Read (ANSI)	All data is locked, preventing external updates on read data.
Serializable (ANSI)	Prevents external updates/inserts/deletes on read data.

	If the database was not created with logging, the DIRTY READ isolation level is the only isolation level available.
---	---

CHAPTER 4

Connecting To Microsoft SQL Server

Concepts

SQL Server connections are maintained through ODBC. While ODBC is a secondary connection method for most RDBMS vendors, it is the principle connection method for SQL Server.

ODBC connections use a DSN to 'describe' the connection parameters. The connection to the database is then made by simply selecting the relevant DSN.

Environment Settings

Variable	Default Value	Function
ODBC_DSN / SQLSERVER		Specifies the base ODBC connection string for the database connection.
QXSS_DB_IS_DSN	false	Specifies that only the string specified in the DATABASE statement should be used to establish the connection. With this set, the runtime will not evaluate the ODBC_DSN environment variable. Defaults to false.
SSTRACE	null	Specifies a file into which to write an ODBC log. Care should be taken in using this variable as ODBC tracing has a significant impact on application performance.

Connecting to SQL Server through 4GL

ODBC connections provide much more flexibility when connecting to a database than traditional Informix database connections. The Dynamic SQL Translator exposes this functionality to the user, whilst still allowing for the traditional method of specifying connections in 4GL/ESQL-C.

The 4GL DATABASE statement may be used in the traditional method, or can be used to specify an ODBC specific connection string when working with SQL Server/ODBC Connections. For example, consider the following ways of specifying a database in the DATABASE statement:

```
DATABASE 'xxx'
```


When executing this statement, if the environment variable ODBC_DSN is set, then the value given to this variable will be the DSN and 'xxx' will be the database name. If, however, ODBC_DSN is not set, then 'xxx' will be the DSN.

```
DATABASE 'xxx@yyy'
```

In this scenario, both the DSN and the database name are explicitly set with 'xxx' being the database name and 'yyy' the DSN.

```
DATABASE 'DSN=xxx; Uid=yyy; pwd=zzz'
```

Using this method of declaring the database will again explicitly set certain variables in the ODBC connection string, in this case the DSN has been set to 'xxx', the user ID (Uid) is 'yyy' and the password (pwd) is 'zzz'.

	<p>The ODBC_DSN environment variable needs to be set in both the system environment and the inet.env file found in \$LYCIADIR\etc</p>
---	---

The following table lists all the relevant ODBC connection options. These may be used within the ODBC_DSN environment variable OR the DATABASE/CONNECT statements.

ODBC Connect Option	Purpose
Address	Network address of the server running an instance of SQL Server. Address is usually the network name of the server, but can be other names such as a pipe, or a TCP/IP port and socket address.
AnsiNPW	Enables ANSI null comparison, data padding and warning behaviour when set to 'Yes'. Default is 'No'.
APP	Name of the application. If not specified, this parameter will be automatically set by the Lycia runtime.
AutoTranslate	When set to true, character data is converted to Unicode through the ANSI code page. The default for this value is true.
DATABASE	The name of the database to connect to. This option is automatically handled by the Lycia runtime, and shouldn't generally be provided in an ODBC connection string. This parameter is overridden by the default database specified within the DSN (if any).

ODBC Connect Option	Purpose
DSN	Name of an existing ODBC user or system data source.
FILEDSN	Name of an existing ODBC file data source.
LANGUAGE	Specifies the language for messages given by the server.
PWD	The password for the SQL Server account used in the connection. This parameter is not required if the connection uses Windows Authentication.
SAVEFILE	Name of an ODBC data source file into which the attributes of the current connection are saved if the connection is successful.
SERVER	Name of the host running SQL Server.
QuotedID	When yes, QUOTED_IDENTIFIERS is set ON for the connection, SQL Server expects all quoted literal values to use single quotes, reserving double quotes for delimiting identifiers.
Regional	When yes, the SQL Server ODBC driver uses client settings when converting currency, date, and time data to character data.
StatsLogFile	Specifies a file used to record performance statistics of the SQL Server ODBC driver.
StatsLog_On	Specifies whether or not performance data should be recorded to the file specified by the StatsLogFile option.
Trusted_Connection	When yes, instructs the SQL Server ODBC driver to use Windows Authentication Mode for login validation. The UID and PWD keywords are optional. When no, instructs the SQL Server ODBC driver to use a SQL Server username and password for login validation. The UID and PWD keywords must be specified.
UID	Name of the SQL Server account to be used in this connection (This parameter is not required if Windows Authentication is specified in the DSN).
WSID	Workstation ID. Typically, this is the network name of the computer on which the application resides (optional). If specified, this value is stored in the master.dbo.sysprocesses column hostname and is returned by sp_who and the Transact-SQL HOST_NAME function.

Connecting to SQL Server from Windows

Installing SQL Server

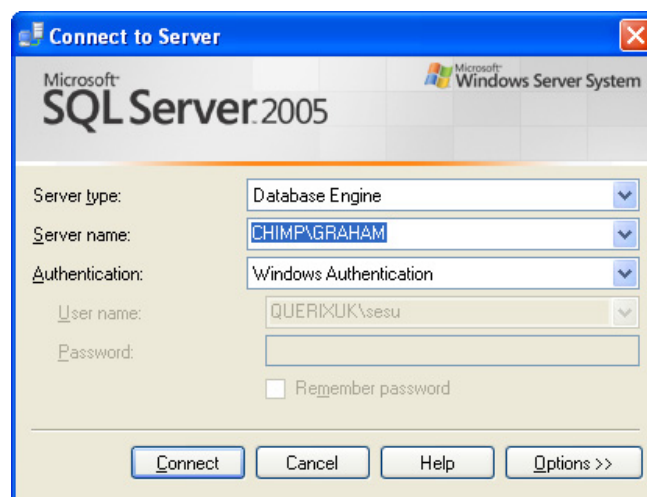
Install as instructed in the SQL Server documentation, at the stage where it asks for an instance name, either choose default or name your own. Make a note of the chosen name.

Adding a database

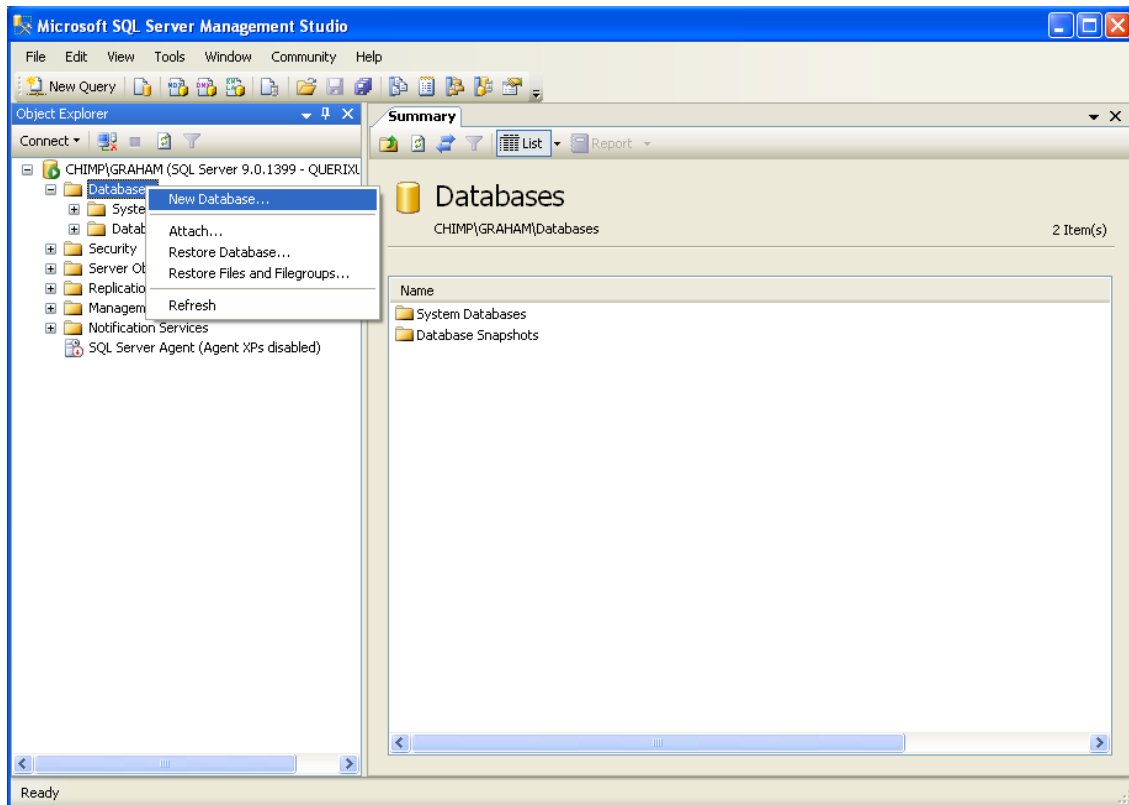
1. Go to the SQL Server Management Studio, found at:

All Programs -> Microsoft SQL Server version_number -> SQL Server Management Studio

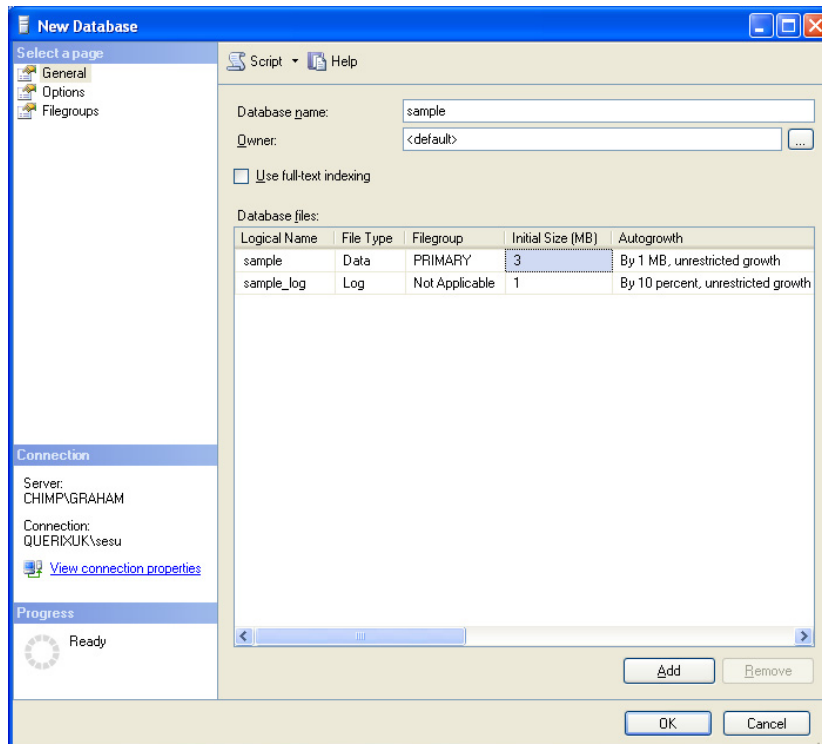
You will be asked to select a Server Type, Server Name and Authentication method. Select the relevant options, with Server Name being the name entered during installation. The other settings should also mirror those chosen during installation. Click on Connect to attempt a connection to the specified server.



2. On the following screen you will see an object explorer on the left-hand side of the screen. Right-click on Databases and select New Database.



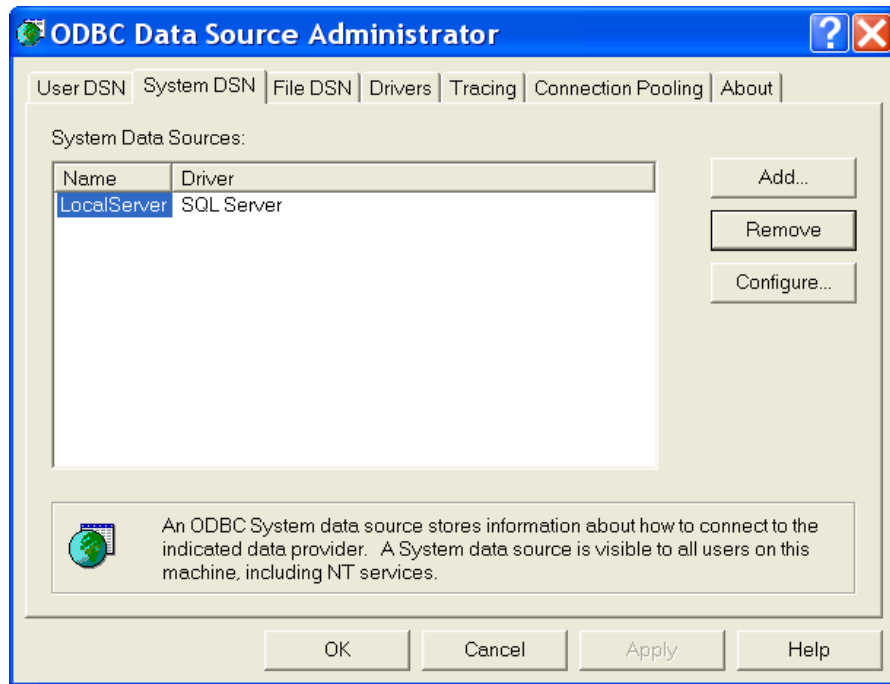
- Next, enter a Database name and press OK. The new database will be created.



DSN Configuration

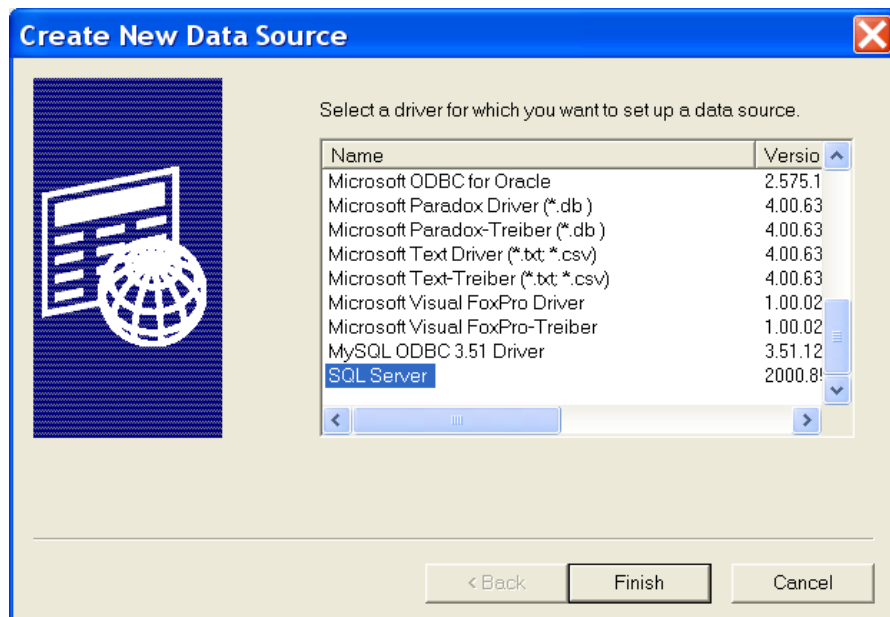
DSN (Data Source Names) are configured locally, through the ODBC data source manager located within 'Control Panel'. The ODBC data source manager can be found under 'Control Panel' – 'Administrative Tools'.

	It is essential to use a System DSN when configuring a connection to SQL Server.
--	--

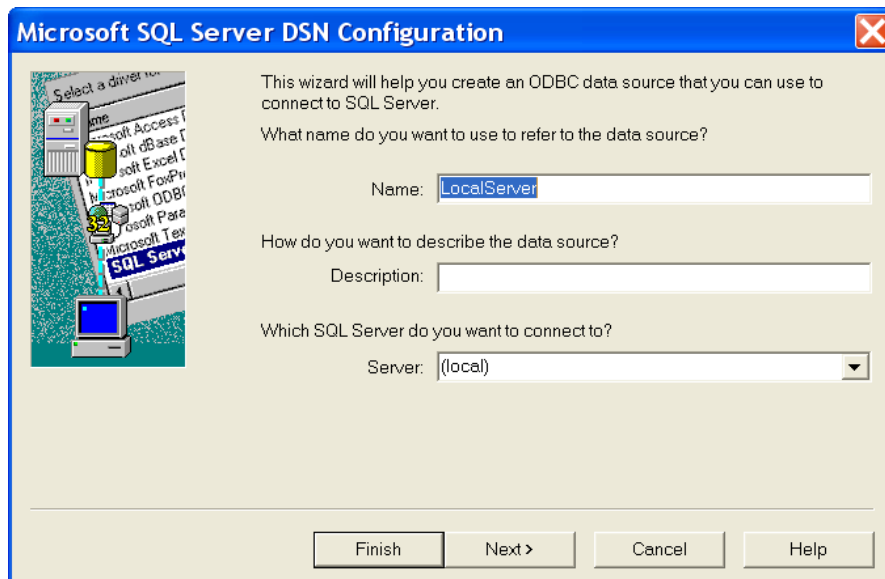


Creating a DSN

Select the 'Add' option. From this option, you will be asked to select the driver for the DSN. Select 'SQL Server'.



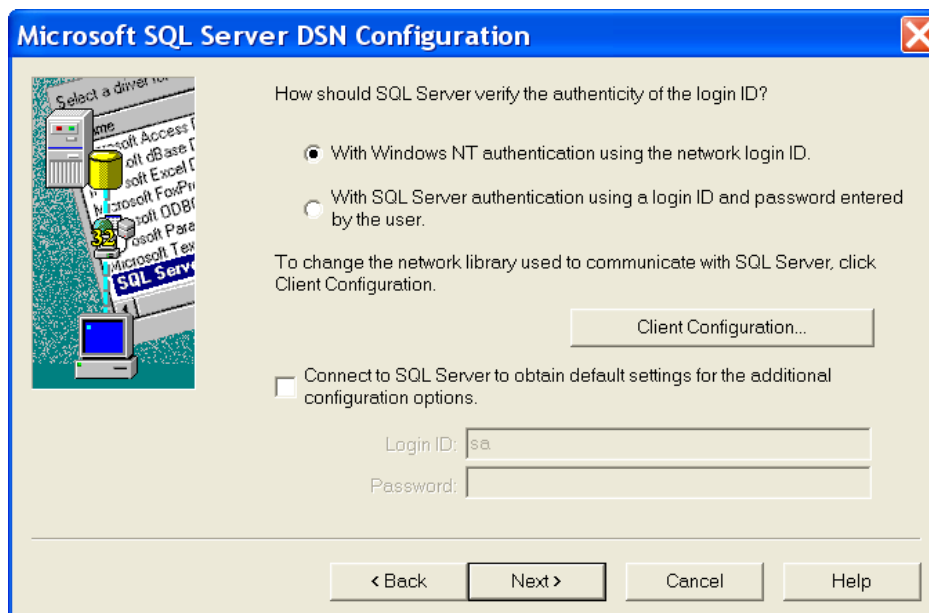
The next step is to specify the base details for the DSN. This includes the name (by which you will refer to the DSN) and the server to which you will be connecting.



Authentication

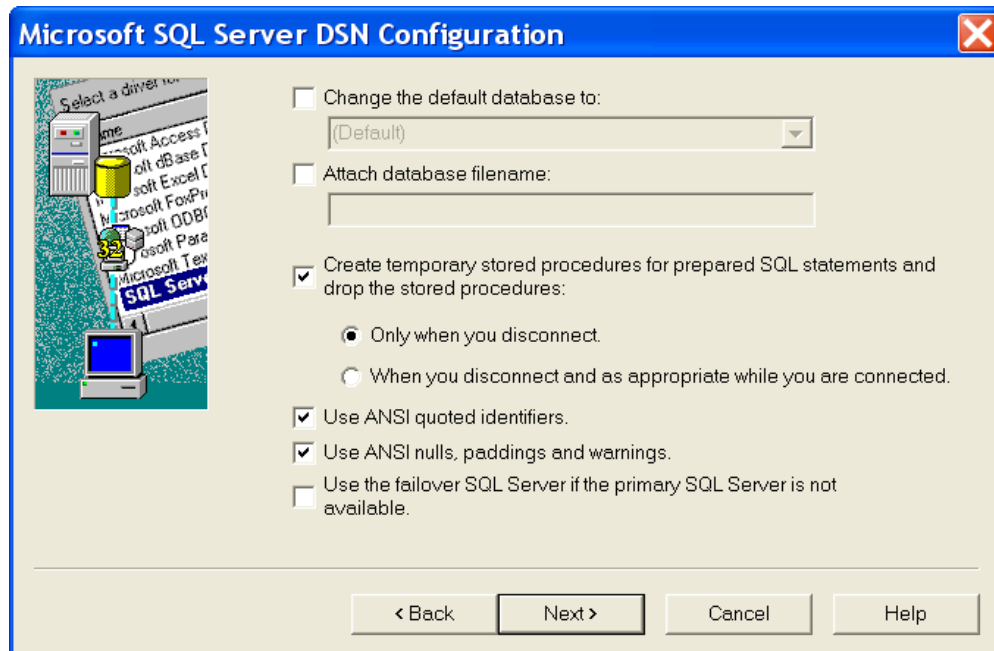
The second page of the DSN configuration specifies the authentication options for the SQL Server connection. This allows you to choose between Windows NT authentication (the connection is performed under the username of the 4GL process), or SQL Server authentication (the connection is performed as a specified username which must exist within the SQL Server database).

It is important that the authentication mode you select is supported by the SQL Server instance.





If SQL Server authentication is specified, you will be required to enter the password within the ODBC connection (either within the DATABASE statement, or the ODBC_DSN environment variable).

Specify the Connection Options



When configuring the DSN, it is strongly recommended to not specify the default database connection. Leave this option unchecked.

Create a New Data Source to SQL Server 



☐ Change the language of SQL Server system messages to:

English

☐ Use strong encryption for data

☒ Perform translation for character data

☐ Use regional settings when outputting currency, numbers, dates and times.

☐ Save long running queries to the log file:

C:\DOCUME~1\DOMINI~1\LOCALS~1\Temp\QUE Browse...

Long query time (milliseconds): 30000

☐ Log ODBC driver statistics to the log file:

C:\DOCUME~1\DOMINI~1\LOCALS~1\Temp\STA Browse...

< Back Finish Cancel Help

Connecting to SQL Server from Unix

While Microsoft does not provide SQL Server connectivity tools from Unix, there are third party products available that will provide the ability to connect to SQL Server from Unix platforms.

To establish a connection from Unix, you will need a copy of the FreeTDS ODBC driver for Unix (available for download from <http://www.freetds.org/>). The FreeTDS driver is available under the terms of the GNU LGPL License.

Configuring FreeTDS

There are two aspects to configuring a connection using FreeTDS. The first is the base configuration of FreeTDS, and the second is the configuration of the ODBC data source.

freetds.conf

FreeTDS is configured within the configuration file `freetds.conf`. This file specifies the available SQL Server connections (which can then be referenced in the ODBC configuration). The simplest example of a configuration file consists of just:

```
[global]
    tds version = 8.0
[example_server1]
    host = host1.sampledomain.com
    port = 1433
```

This allows FreeTDS to establish connections to the SQL Server instance hosted on 'server1'. This will, by default, use TDS version 7.0. By default SQL Server operates on port 1433.

odbc.ini

Like all other Unix/ODBC connections, the `odbc.ini` file contains the configuration information for the DSNs used in the database connection. In the following example, we create a DSN named 'example_dsn' which references the server 'example_server1' in the `freetds.conf` file.

```
[ODBC Data Source]
    example_dsn = FreeTDS ODBC example

[example_dsn]
    Driver      = /usr/local/freetds/lib/libtdsodbc.so
    Description = FreeTDS ODBC connection example
    Trace       = No
    Servername   = example_server1
    Database     = stores
    [Default]
    Driver      = /usr/local/freetds/lib/libtdsodbc.so
```

CHAPTER 5

Preparing for the SQL Server Conversion

When converting an application to work with an additional RDBMS, a number of key steps need to be taken to ensure a successful result. It is strongly recommended that Querix tools be used for all stages of this process, in order to ensure that the application works consistently against each RDBMS with minimal code modification.

This chapter outlines the base process for converting an application to work with an additional RDBMS.

The Conversion Process

The migration process can be thought of as a four-stage process:

- Migration of the Database Schema
- Table loading
- SQL Syntax issues within 4GL code
- Runtime testing


The overall aim of the process should be to adapt an existing application to work with a new RDBMS in addition to still functioning as expected with the original RDBMS.

Migration of the Database Schema

'Lycia' will handle schema object conversions for you by means of a 4GL/ESQL-C application. Unlike the native migration tools for your target RDBMS, 'Lycia' will allow you to retain basic Informix datatypes, which are not necessarily handled natively by the RDBMS. These datatypes can include:

- SERIAL
- MONEY
- DATETIME
- INTERVAL

Also included may be various behavioural facets associated with the datatypes in question. Please refer to the RDBMS vendor specific information in chapter 3 for information on datatype support.

	<p>Using Lycia tools to perform the database object creation will maximise compatibility of the target RDBMS with Informix behaviour, and minimise later conversion problems.</p>
---	---

The simplest method of assisted database object creation is to convert your Informix database schema into a 4GL source. This can be achieved using the Querix utility 'qxexport'.

To invoke this utility from the command line, you simply need to perform:

```
bash$ qxexport <database_name>
```

This will create a directory named <database_name>_qxexport. Within this directory you will find a file named '<database_name>.4gl' and a subdirectory containing unload data for the database (unless the qxexport utility was invoked with the -s flag).

Having created the 4GL module, it is simply a matter of compiling this code and running it against the target RDBMS in order to create the database objects. For more information on connecting a 4GL application to your vendor's RDBMS, please see appendix 'Connecting to Microsoft SQL Server'.

It is worth noting that if your target RDBMS does not support the use of reserved words as identifiers, you may encounter a number of table creation errors at this stage (see chapter 'Reserved Words as Identifiers' for information on the quoted identifiers database creation option in SQL Server). If your RDBMS does not support the use of reserved words as identifiers, this can only be addressed by renaming the conflicting column(s) - bearing in mind that these changes should be propagated into table references within the 4GL/ESQL-C code

Loading Table Data

Loading of the table data should again be handled by 4GL - this ensures that the data is stored and converted correctly where appropriate. The schema extraction tool (qxexport) will automatically unload table data, and generate LOAD statements in the generated 4GL file for schema creation.

The 4GL LOAD statement can be resource intensive. For this reason, the overall schema creation/loading process may take time depending on the volume of data to be loaded.

Compiling 4GL Code

Once the schema objects have been created, you should now be able to proceed with the compilation of 4GL code against the target RDBMS.

The qxexport Schema Extraction Tool

Querix provides a command line tool named 'qxexport' for the extraction of an Informix database schema and table data.

The tool generates a Lycia project, including a 4GL program (and table unload data) which can be compiled and run to automatically create and populate tables in the target RDBMS.

CHAPTER 6

Informix Compatibility

This chapter discusses scenarios within an Informix application which cannot be handled automatically by the Dynamic SQL Translator. It is broken into 3 sections marking the severity of the problem class. These sections cover the following:

- Problems which are not handled
- Problems that are handled through the emulation of Informix behaviour
- Problems that are automatically handled, and are of no overall concern to the developer when working through the Dynamic SQL Translator



Microsoft has identified a potential issue when running many processes while using SQL Server on NT Servers. This issue is discussed in the Microsoft Knowledge Base article 824422.

<http://support.microsoft.com/default.aspx?scid=kb;en-us;824422>.

Unhandled Problems

MATCHES in SQL

The MATCHES keyword within Informix SQL is a non-ANSI extension to SQL supported only by Informix. The majority of RDBMSs do not have a direct equivalent to this operator, and as such, this should be addressed in advance.

The nearest equivalent (for most uses of MATCHES) is the LIKE operator. The SQL Server LIKE operator is able to handle the same types of expression as the Informix MATCHES operator, however a number of changes need to be applied in the source SQL.

- The MATCHES operator should be replaced with the LIKE operator
- The MATCHES wildcard character '*' should be replaced with the LIKE wildcard character '%'.
The MATCHES wildcard character '?' should be replaced with the LIKE wildcard character '_'.
Care should be taken to escape any literal '%' or '_' in the expression being matched. '*' and '?' will no longer need to be escaped.

MATCHES Expression	LIKE Expression
WHERE table1.col MATCHES 'ABCD*'	WHERE table1.col LIKE 'ABCD%'
WHERE table1.col MATCHES 'ABCD?'	WHERE table1.col LIKE 'ABCD_'
WHERE table1.col MATCHES 'ABCD%*'	WHERE table1.col LIKE 'ABCD\%*'
WHERE table1.col MATCHES 'ABCD_\?'	WHERE table1.col LIKE 'ABCD_?'
WHERE table1.col MATCHES 'ABCD[0-9]'	WHERE table1.col LIKE 'ABCD[0-9]'
WHERE table1.col MATCHES table2.col	Care should be taken to check the contents of the column being matched before making decisions regarding the conversion of this statement.

HOLD Cursors

A 'HOLD' cursor based on a 'SELECT ... FOR UPDATE' statement will retain table locks, even after a transaction commit.

DATETIME Precision

The SQL Server DATETIME datatype has a precision of approximately 1/300 seconds (DATETIME values will be rounded to 0.000, 0.003 or 0.007 seconds). The Informix DATETIME datatype has a default precision of 1/1000 seconds and a maximum precision of 1/10000 seconds.

For this reason, DATETIME values of precision greater than FRACTION(3) cannot be supported in SQL Server.

CREATE DATABASE

The Dynamic SQL Translator will not allow CREATE/DROP DATABASE statements to reach the RDBMS, as they are not portable. Creating and Dropping databases must be handled through the SQL Server tools.

DROP INDEX Syntax

The Transact-SQL 'DROP INDEX' DDL statement requires the table name from which the index is being dropped to be specified, for example:

```
DROP INDEX <index name>.<table name>
```

Since the Informix DROP INDEX statement doesn't provide the associate table, there is no deterministic way of automatically converting this statement.

SPL Objects

Objects written using SPL (such as stored procedures, triggers and functions) cannot be automatically converted to the stored procedure language used by the target RDBMS. For this reason, all such objects must be recreated by hand.

Problems Handled Through Emulation

The following issues are automatically handled via emulation of Informix behaviour. In most cases the emulation will be transparent to the developer/user. The relative problems are detailed with each item.

Fetching Floating DECIMAL Types

When fetching a floating point decimal from SQL Server (for example, a DECIMAL(16)), the Dynamic SQL Translator interface has to perform the fetch into an intermediate CHAR(33) datatype. As a result, the decimal format expected by the process (as specified by either DBMONEY or DBFORMAT) must match the format in use by the SQL Server instance.

All operations with fixed point decimals (for example, DECIMAL(10,3)) are unaffected by this.

INTERVAL Data Types

SQL Server does not support any form of interval datatype. For this reason, Lycia provides full interval type emulation within the SQL Server database. This can only be enabled by using Lycia to execute any DDL statements referencing interval datatypes.

ROWID


SQL Server has no equivalent of ROWID; therefore ROWID emulation is provided.

The implementation method is table dependent. For tables with an existing SERIAL column, this column is returned as the ROWID value. For columns without a SERIAL column, a new hidden column ROWID is added to the table.

Setting the environment variable QXORA_EMULATE_ROWID to a value of 1 will enable the ROWID emulation feature. If ROWID emulation is not required the QXORA_EMULATE_ROWID environment variable should be unset.

When performing an INSERT of a literal value into a serial column, the ROWID for the insert is not correctly returned in sqlca.sqlerrd[6].

ROWID emulation is available for SQL Server.

	The ROWID emulation is unable to re-use ROWID values in an integer context. For this reason, it is feasible that an application with a high insert/delete rate may exceed the 2^{32} (approx. 4 billion) value limit. Use of this feature, therefore, should be treated with due consideration.
---	---

Tables Comprised solely of a SERIAL column

The Dynamic SQL Translator uses the SQL Server identity type to provide Informix SERIAL behaviour. SQL Server does not allow tables consisting of just a single identity value. For this reason, the translator will create a second column called qx__\$dummy. This column will not be acknowledged by the Dynamic SQL Translator when processing information about a table.

For example, the table

```
CREATE TABLE test1 (  
    column1 SERIAL  
)
```

Will be created as

```
CREATE TABLE test1 (  
    column1 integer identity,  
    qx__$dummy integer  
)
```

Problems Fully Handled

The following differences are automatically handled, and will present no problems for the developer/user. This is not an exhaustive list and covers only the major differences.

Function/Operator Naming

The Dynamic SQL Translator will convert all standard Informix functions/operators to their Oracle equivalents where an appropriate match exists. The following translations are performed automatically:

- YEAR()
- MONTH()
- DAY()
- DATE()
- MDY()
- WEEKDAY()
- LENGTH()
- TODAY
- CURRENT
- DATETIME / INTERVAL literals
- Character subscripts/substrings

Note that the expressions passed through to functions may also be modified internally depending on the context within the SQL expression.

Reserved Words as Identifiers

The Dynamic SQL Translator will automatically quote any keywords in an identifier context.

For example, the following statement:

```
SELECT *  
FROM x1  
-- The column 'percent' exists in the table x1  
WHERE percent = 50
```

To prevent an SQL error, the translator would reproduce this statement as:

```
SELECT *
FROM x1
-- The column 'percent' exists in the table x1
WHERE "percent" = 50
```

DDL Syntax

There are numerous differences in the core DDL syntax between Informix and SQL Server. This includes such things as:

- ALTER TABLE
- Constraint naming

SQL Server has a number of constraints on the usage of the ALTER TABLE statement. All of the actions in the statement must be heterogeneous, for example:

```
ALTER TABLE table1 ADD column1 INTEGER, ADD column2 INTEGER
```

```
ALTER TABLE table1 ADD (column1 INTEGER, column2 INTEGER)
```

The above statements will translate correctly, since both actions add a column. However, the statement

```
ALTER TABLE table1 ADD column1 INTEGER, DROP column2
```

Cannot be translated, since the SQL Server ALTER TABLE statement will not allow the two different actions to be performed simultaneously. This must be rewritten as distinct statements

```
ALTER TABLE table1 ADD column1 INTEGER
```

```
ALTER TABLE table1 DROP column2
```

Join Syntax

SQL Server does not support the Informix Join syntax, instead using the ANSI join syntax for SQL. The Dynamic SQL Translator automatically handles the conversion for all join expressions.

GROUP BY Syntax

SQL Server does not support ordinals in a GROUP BY expression. The Dynamic SQL Translator automatically converts GROUP BY expressions to a form supported by SQL Server.

For example, the following statement:

```
SELECT 'X', column1, SUM (column2)
FROM table1
GROUP BY 1, 2
```

Is automatically converted to the supported form:

```
SELECT 'X', column1, SUM (column2)
FROM table1
GROUP BY column1
```

Note that the literal 'X' is no longer included in the GROUP BY expression. Other syntactical differences handled in GROUP BY expressions include:

- References to correlated sub-queries
- References to literal values
- Expression aliases

Weak Typing

The Dynamic SQL Translator handles type conversion automatically, before any expressions/values are sent to the RDBMS. This means that weak typing behaviour is fully supported through the Dynamic SQL Translator. This solves many potential problems, including (and not limited to):

- Datetime/Date literals
- Interval Literals
- Numeric Literals
- Numeric Values in a date context
- Datetime/ Date arithmetic
- Typing of Function Arguments for known functions.

SQL Error Codes

4GL applications are largely dependent upon expected error codes from Informix. SQL Server reports errors using the standard SQLSTATE identifier.

For this reason, the Dynamic SQL Translator will convert SQLSTATE values to the Informix equivalent error, and populate the sqlca.sqlcode field accordingly.

Should the RDBMS return an error code which does not correspond to a known Informix error code, the database interface will return sqlca.sqlcode with a value of -999, and the native (ISAM) error stored in sqlca.sqlerrd[2].

Configurable Behaviour

The following options can be set in the fglprofile to configure the behaviour of the Oracle database interface.

`dbi.sserver.rowid = (emulate | none)`

This option determines whether or not ROWID emulation is enabled. This option defaults to 'native' (emulation disabled).

APPENDIX A

The 'Lycia' Dynamic SQL Translator

This chapter discusses the dynamic SQL translation engine, and the work that it performs. This chapter is intended to provide an overview of the capability of the translator.

Although the principles of operation of the translator are common across all RDBMS vendors, the functionality of the translator is sensitive to capabilities and personality of the target RDBMS.

In this chapter we will be considering the general operation of the translator. Microsoft SQL Server specific behaviour is discussed in greater detail in chapter 'Informix Compatibility' of this document.

Introduction

The basic function of the Dynamic SQL Translator (DST) is to make a non-Informix database appear as close as possible to an Informix database for the purposes of 4GL and ESQL-C functionality.

The translator takes the users' Informix SQL statements as input, and dynamically translates them to the format required by the target RDBMS.

In addition to this, the DST also emulates the behaviour of Informix datatypes, such as SERIAL, MONEY, etc., even where no direct equivalent exists within the target RDBMS.

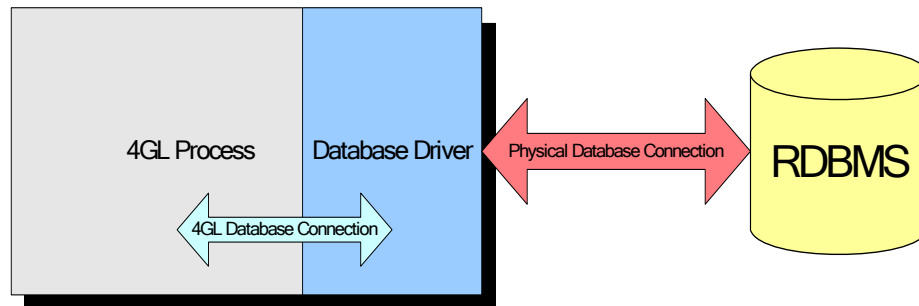
The overall result is that 4GL and ESQL/C programs can be used with a wide range of RDBMSs with the absolute minimum of code modification.



The concepts described in this chapter apply only to non-Informix databases. The Dynamic SQL Translator is not used for Informix connections.

Concepts

Database Drivers



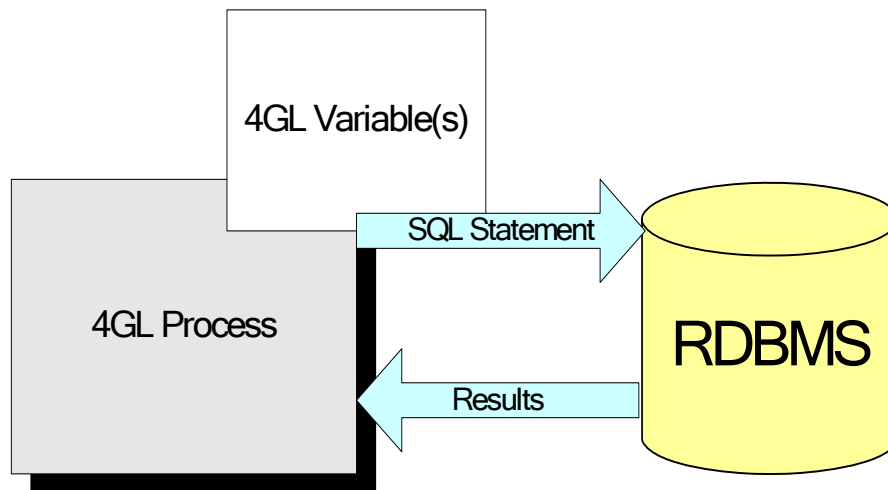
Each database vendor provides a different means of connecting to and communicating with a database. For this reason, it is not possible to use the same communication mechanism with, for example, SQL-Server as you would with Informix.

Each database, therefore, requires a different layer for communication. 'Lycia' provides such a layer for each database vendor supported, and these layers are referred to as 'Database Drivers'.

Multiple connections to the same database type will all use the same driver.

Bind Variables

A bind variable is a 4GL or ESQL-C variable which is passed as a parameter to a SQL statement.



For example, consider the following 4GL statement:

```
DEFINE my_int INTEGER
LET my_int = 1
```

```
SELECT *  
  FROM my_table  
  WHERE my_table.column1 = my_int
```

In this example, the 4GL variable 'my_int' is passed as a parameter (or input value) for the SQL select statement. Similarly, the values that are passed to a prepared statement to substitute the placeholder markers ('?') are also bind variables. For example:

```
DEFINE my_int INTEGER  
...  
PREPARE p1 FROM  
  "SELECT * FROM x1 WHERE x1.a = ? AND x1.b = ?"  
EXECUTE p1 USING 1, my_int
```

In this example, both the literal value '1' and the 4GL variable 'my_int' are considered bind variables for the SQL statement.

SQL statements that can use bind variables are:

DELETE

EXECUTE PROCEDURE

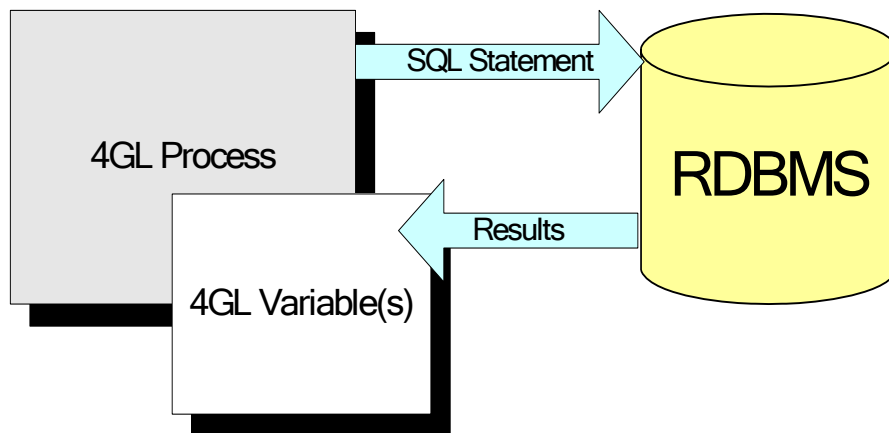
INSERT

SELECT

UPDATE

Buffer Variables

We use the term 'buffer variables' to describe the variables into which an SQL statement returns data. Such variables will typically be found in the 'INTO' clause of a SQL statement.



For example, consider the following 4GL fragment:

```
DEFINE rec1 RECORD LIKE x1.*

DECLARE c1 CURSOR FOR
  SELECT *
  FROM x1

FOREACH c1 INTO rec1.*
  ...
```

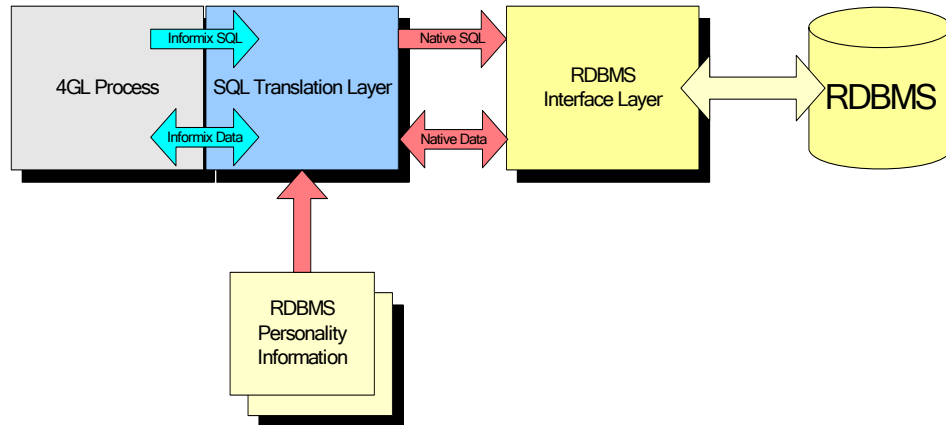
In this example, each element of the record 'rec1' is a buffer variable, as they are modified with each execution of the FOREACH loop.

SQL Statements that can use buffer variables are:

EXECUTE PROCEDURE

SELECT

How Dynamic Translation Works




The translation engine has a number of layers of functionality, and these can be categorised as follows:

SQL Interpretation
SQL generation
Type promotion

SQL interpretation

The first function of the translation layer is to read and understand the SQL statement passed to it. During this process, the translator will evaluate the entities present within an SQL statement.

Once interpreted, the statement is categorised to allow optimal processing of the SQL statement.

	<p>If the translation engine does not fully recognise the SQL statement, it will not attempt to process it. Instead, it will pass the statement to the RDBMS unmodified.</p>
---	--

SQL Generation

The final function of the Dynamic SQL Translator is to recreate the SQL statement in a form that is understood by the target RDBMS. There are numerous aspects to this, but the key areas are:

- **Keywords:** Many SQL keywords are vendor specific
- **Grammatical differences in SQL:** Each RDBMS exhibits minor differences in the grammatical structure of SQL
- **Functions and Operators:** The names and syntax of functions and operators varies between RDBMS vendors.

- Datatypes: Some Informix datatypes are not supported by other RDBMS vendors.

Datatype Mappings

In addition to type promotion in SQL (this will be discussed in the next section), the Dynamic SQL Translator is sensitive to differences in the data types supported by different RDBMS vendors. For this reason, the dynamic SQL translator automatically handles the translation of Informix types to vendor specific types, in such a way that these differences are transparent to a 4GL or ESQL/C process.

The qx__\$schema table

Because interpretation has to be performed for some datatypes, the translator will maintain metadata about such columns in the table qx__\$schema. The qx__\$schema table contains such information as the original (or intended) Informix datatype, and any supplementary information (such as qualifiers for DATETIME types).

Type Promotion

Unlike Informix, a number of RDBMSs are strongly typed. This means that they do not permit comparisons of differing datatypes. For example, where Informix may allow the user to directly compare a CHAR representing a date to a DATE value, SQL-Server will not.

The SQL translator scans the SQL statement for areas where differing datatypes are being used, and attempts to convert the types where possible.

As an example, consider the following scenario:

```
SELECT *
FROM x1
WHERE x1.date_column = "01/01/2001"
```

In this example, the DST will first look at the relational operator '=':

In evaluating this operator, the translator realises that potential problems exist due to a CHAR value being compared to a DATE column. Problems that potentially exist include:

- A strongly typed RDBMS will not allow such a comparison without an explicit cast operation.
- Localisation of the database server may prevent the literal date value from being correctly interpreted.
- The RDBMS may not be able to efficiently cache semantically equivalent statements that differ only by literal value.

Since it is clear that the column cannot be altered, it is necessary to process the literal value instead.

In this case, the translator will first attempt to convert the literal value to a bind value of type CHAR. In this case, the SQL will now be seen as:

```
SELECT *
FROM x1
```

```
WHERE x1.date_column = ?
```

Finally, due to the context of the CHAR bind, the CHAR bind will be converted to a bind of type DATE, preventing the need for any explicit casting operation.

The result of this process is an SQL statement that will be accepted by a strongly typed database, and also provides more efficient SQL than if the vendor's conversion functions (such as CAST or CONVERT) had been used.

Benefits of Type Promotion

There are a number of beneficial side-effects of the type promotion process. Firstly, the database is able to cache SQL statements better if literal values aren't used. For example, consider the following statements:

```
SELECT * FROM x1 WHERE x1.a = 1
SELECT * FROM x1 WHERE x1.a = 2
SELECT * FROM x1 WHERE x1.a = 3
```

The RDBMS will attempt to cache each of these statements. This caching is made more effective by the SQL translator, as in each of these cases, the only statement passed to the RDBMS is:

```
SELECT * FROM x1 WHERE x1.a = ?
```

As a result, the database is able to cache SQL statements much more efficiently.

Areas where Type Promotion is used

Type promotion will be attempted for all parts of SQL where datatypes won't necessarily match. This includes:

- Parameters (bind variables) to INSERT statements
- Parameters (bind variables) to UPDATE statements
- WHERE clauses
- Output parameters (buffer variables) of SELECT statements

Where bind/buffer variables are type promoted, the promotion is applied to a duplicate variable, with the original 4GL variable remaining unaltered.

APPENDIX B

Configuring Options

This section discusses some of the database level settings available in Microsoft SQL Server. All of these settings can be specified during database creation, and where specified, may also be set within a session.

For each of these options, potential impact on 4GL applications is listed.

Database Creation Options

The following options may be specified when creating a database within Microsoft SQL Server. Some of these options may also be set during a session using the session 'SET' statement.

Database option	SET option	Description	Default
ANSI_NULL_DEFAULT	ANSI_NULL_DFLT_ON ANSI_NULL_DFLT_OFF	Affects the nullable option of a column if a column is created without an explicit 'NOT NULL' clause. When set to OFF (or the session setting ANSI_NULL_DFLT_OFF), columns are created implicitly as having a NOT NULL constraint.	OFF
ANSI_NULLS	ANSI_NULLS	Affects the behaviour of relational operators (such as '=' and '< >') when comparing NULL values. Set to ON for ANSI behaviour.	OFF
ANSI_WARNINGS	ANSI_WARNINGS	Affects the generation of ANSI warnings in aggregation over NULL values, and divide by zero operations.	OFF
CONCAT_NULL_YIELDS_NULL	CONCAT_NULL_YIELDS_NULL	Affects the results of concatenation operations involving NULL values. When set to true, ANSI concatenation behaviour is enabled.	OFF
CURSOR_CLOSE_ON_COMMIT	CURSOR_CLOSE_ON_COMMIT	Causes cursors to automatically be closed on a commit. The runtime will automatically enforce this behaviour.	OFF

Database option	SET option	Description	Default
QUOTED_IDENTIFIER	QUOTED_IDENTIFIER	When true, allows identifiers to be quoted with double quotation marks. Literals must be quoted using single quotation marks. The runtime automatically handles the conversion of literals to use single quotes.	OFF
RECURSIVE_TRIGGERS	None	Allows a trigger operation to invoke another trigger.	FALSE

Collation Sequences

Microsoft SQL Server allows the creation of databases with case insensitive collation sequences. If your application is dependent upon case sensitivity within relational operations, for instance, then it is necessary to create the database with a case-sensitive collation sequence.

For example, the following SQL statement:

```
SELECT 1
WHERE 'ABCD' = 'abcd'
```

Will return data in a case insensitive collation sequence, and yield no data in a case sensitive collation sequence.

Collation sequences are named in the form Category_CS_xxx for case-sensitive collation sequences, and Category_CI_xxx for case insensitive collation sequences.

APPENDIX C

Datatype Mappings

The following table lists the Informix SQL/4GL datatypes, and the type mapping adopted by the Dynamic SQL Translator. All Informix datatypes are supported under SQL Server through the Dynamic SQL Translator. If any behavioural emulation is required, it is stated within the notes for that datatype.

Informix SQL Datatype	Informix 4GL Datatype	SQL Server Datatype	Notes
CHAR	CHAR	CHAR	
VARCHAR	VARCHAR	VARCHAR	The Informix VARCHAR type is restricted to 255 characters
LVARCHAR	VARCHAR	VARCHAR	
NCHAR	NCHAR	NCHAR	SQL Server nchar is fixed 16-bit encoding, whereas Informix nchar is based on variable width encodings.
NVARCHAR	NVARCHAR	NVARCHAR	SQL Server nchar is fixed 16-bit encoding, whereas Informix nchar is based on variable width encodings.
SMALLINT	SMALLINT	SMALLINT	
INTEGER	INTEGER	INT	
INTEGER8	INT8	BIGINT	
SERIAL	INTEGER	INTEGER IDENTITY	
SERIAL8	INT8	BIGINT IDENTITY	
FLOAT	FLOAT	REAL	
SMALLFLOAT	SMALLFLOAT	FLOAT	

Informix SQL Datatype	Informix 4GL Datatype	SQL Server Datatype	Notes
DOUBLE PRECISION	FLOAT	REAL	
DECIMAL	DECIMAL	DECIMAL	
MONEY	MONEY	DECIMAL*	While SQL Server has a money datatype, it does not cover the range of values available to an Informix money datatype. Informix's money type has equivalent range to a decimal type.
DATE	DATE	DATETIME*	
DATETIME	DATETIME	DATETIME*	Informix datetime type has maximum precision of fraction(5)
INTERVAL	INTERVAL	FLOAT*	Emulated type
BYTE	BYTE	IMAGE	
TEXT	TEXT	TEXT	

The following table lists the mappings adopted by the Dynamic SQL Translator when encountering a datatype within the SQL Server database.

SQL Server	Informix 4GL	Informix SQL	Notes
BIGINT	INT/INT8	INT8	
INT	INT	INT	
SMALLINT	SMALLINT	SMALLINT	
TINYINT	SMALLINT	SMALLINT	
BIT	SMALLINT	SMALLINT	
DECIMAL	DECIMAL	DECIMAL	
MONEY	MONEY	MONEY	
SMALLMONEY	MONEY	MONEY	
FLOAT	SMALLFLOAT	SMALLFLOAT	
REAL	FLOAT	FLOAT	
DATETIME	DATETIME YEAR TO FRACTION(3)	DATETIME YEAR TO FRACTION(3)	Default mapping is assumed to be datetime, unless the qx__\$schema table states otherwise.
SMALLDATETIME	DATETIME YEAR TO SECOND	DATETIME YEAR TO SECOND	Default mapping is assumed to be datetime, unless the qx__\$schema table states otherwise.
CHAR	CHAR	CHAR	

SQL Server	Informix 4GL	Informix SQL	Notes
VARCHAR	VARCHAR	LVARCHAR	The Informix varchar type has a 255 character limit. The SQL Server varchar type has an 8000 character limit.
NCHAR	NCHAR	NCHAR	
NVARCHAR	NVARCHAR	NVARCHAR	
TEXT	TEXT	TEXT	
NTEXT	TEXT	TEXT	The SQL Server ntext type stores Unicode data. The Informix text type stores fixed 8-bit encoded data.
BINARY	BYTE	BYTE	
VARBINARY	BYTE	BYTE	
IMAGE	BYTE	BYTE	
UNIQUEIDENTIFIER	CHAR(36)	CHAR(36)	Fixed format globally unique identifier. Uses the format "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx", where 'x' represents a hexadecimal digit.

APPENDIX D

Common Conversion Issues when migrating to different Vendor's Databases

Prior to recompiling the 4GL, a number of operational issues need to be addressed in the code. This is due to a number of fundamental operational differences between Informix and the target RDBMS. It is important to address these issues in advance, as the consequences of one issue can have cascading effects.

System Catalogues

The Informix system catalogues are generally not directly available in the target RDBMSs. Dependent upon the RDBMS in question, system catalogues can be either emulated as a series of views, or references to the catalogue in question automatically converted to the native equivalent. In the majority of cases, it is best not to be dependent upon the system catalogues behaving as you would expect under Informix.


'Lycia' provides a set of API functions for accessing the native system catalogues. It is recommended that these functions be used in place of direct queries against the table, for two key reasons:

- The true Informix type and characteristics of the column is returned.
- All work concerning the variances in catalogues are automatically handled.

Matches

The MATCHES keyword within Informix SQL is a non-ANSI extension to SQL supported only by Informix. The majority of RDBMSs do not have a direct equivalent to this operator, and as such, this should be addressed in advance.

The nearest equivalent (for most uses of MATCHES) is the LIKE operator. Unlike the MATCHES operator, LIKE cannot handle regular expressions, only wildcards.

	<p>The MATCHES operator only poses a problem within SQL statements. The operator will pose no problem within general 4GL program logic.</p>
---	---

Owing to the problems associated with the MATCHES operator, wildcard symbols ('*' & '?') entered during a CONSTRUCT statement will automatically converted to LIKE equivalents. Under such circumstances a LIKE operator is generated in place of MATCHES in the generated SQL clauses.

Cursor Names

Due to common restrictions within various RDBMS systems, ALL cursor names are subject to an 18-character length limit. By default, the 4GL compiler will hash all cursor names into an 18 character string. If you disable cursor hashing within the 4GL compiler (by passing the flag '-CH' to fglc), then any cursor whose name is over 18 characters in length will need to be renamed.

Isolation Levels

Isolation levels are RDBMS specific models for handling concurrent transactions. These operations are handled at the server level, and as such, no guarantee can be provided for identical behaviour in similarly named isolation levels between differing RDBMS vendors.

Most RDBMSs will only allow Isolation Levels to be switched between transactions.

Stored Procedures

Stored procedures must be recoded according to the facilities available in the target RDBMS. Whilst we can provide general guidelines for procedure coding, there are no formal procedures for the conversion of SPL.

Index

Address	10	Loading table data	21
ANSI_NULL_DEFAULT	43	Matches	49
ANSI_NULLS	43	MATCHES in SQL	24
ANSI_WARNINGS	43	Migration of the Database Schema	20
AnsiNPW	10	MONEY	46
APP	10	NCHAR	45, 48
Authentication	7, 16	NTEXT	48
AutoTranslate	10	NVARCHAR	45, 48
BIGINT	47	ODBC Connect Option	10
Bind Variables	36	ODBS_DSN	9
Buffer Variables	38	Preparing	20
CHAR	47	Prerequisites	5
Collation Sequences	44	Problems Fully Handled	28
Common Conversion	49	Problems Handled Through Emulation	26
compared	7	Product Range	7
Comparison	7	PWD	11
Compiling 4GL code	21	QuotedID	11
CONCAT_NULL_YIELDS_NULL	43	qx__\$schema table	40
Configuring FreeTDS	19	qxexport	20, 21
Configuring Options	43	QXORA_DB_IS_DSN	9
Connecting	9	Read Committed (ANSI)	8
Conversion	20	Read Uncommitted (ANSI)	8
CREATE DATABASE	25	Regional	11
Creating a DSN	15	Repeatable Read	8
Cursor Names	50	Repeatable Read (ANSI)	8
Cursor Stability	8	Reserved Words as Identifiers	28
CURSOR_CLOSE_ON_COMMIT	43	ROWID	26
DATABASE	10	SAVEFILE	11
Database Creation Options	43	schema object	20
Database Drivers	36	SERIAL column	27
Datatype Mappings	40, 45	SERIAL8	45
DATETIME	46, 47	Serializable (ANSI)	8
DDL Syntax	29	SERVER	11
Dirty Read	8	SMALLDATETIME	47
DSN	11, 15	SQL Compliance	7
DSN Configuration	14	SQL Error Codes	30
Dynamic SQL Translator	35	SQL Generation	39
Environment Settings	9	SQL interpretation	39
Fetching Floating DECIMAL Types	26	SSTRACE	9
FILEDSN	11	StatsLog	11
GROUP BY Syntax	29	StatsLogFile	11
HOLD Cursors	24	Storage	7
Informix Compatibility	23	Stored Procedures	50
Informix datatypes	20	Supported Connection Interfaces	7
INTEGER8	45	System Catalogs	49
INTERVAL	46	TEXT	48
Interval Data Types	26	Translation	39
Isolation Levels	7, 8, 50	Type Promotion	40
Join Syntax	29	UID	11
LANGUAGE	11	Unhandled Problems	24

UNIQUEIDENTIFIER.....	48
Unix.....	19
VARCHAR	48

Weak Typing	30
WSID	11