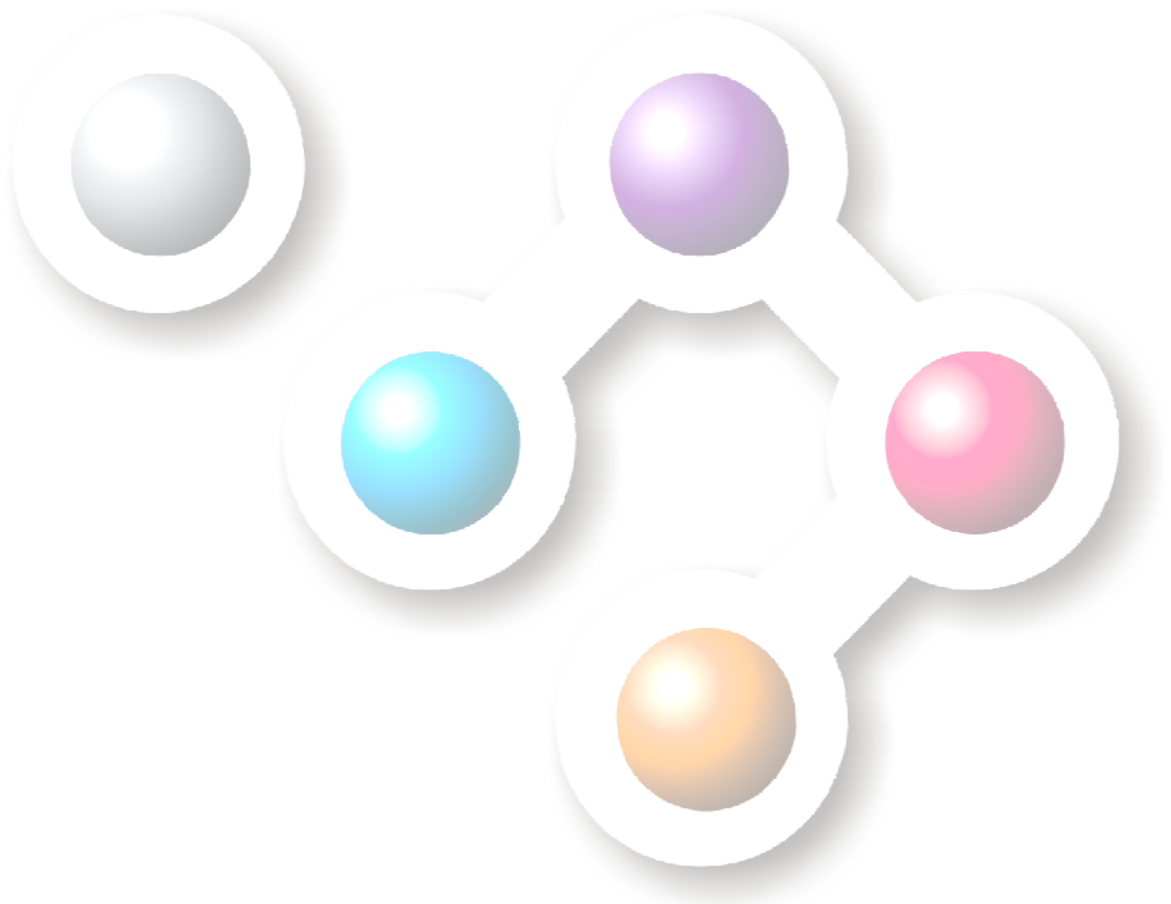


Lycia Development Suite



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



Dynamic Database Interface

Migration Guide for PostgreSQL

Versions 5 and 6

January 2011

Part Number: 014-006-135-011

Querix Dynamic Database Guide for PostgreSQL

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

January 2011. Part Number: 014-006-135-011

Published by:

Querix Ltd, 50 The Avenue, Southampton, 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 PostgreSQL, updated for versions 4.3 and retitled 'Database Migration Guide for PostgreSQL'
November 2008:	Updated for 4.4
January 2011:	Updated for v 5 and v6

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.....	7
ABOUT THIS GUIDE	1
<i>Who should read this Reference Guide?</i>	1
<i>Conventions used in this book</i>	1
Typefaces and Icons	1
CHAPTER 1	3
INTRODUCTION	3
CHAPTER 2	5
PREREQUISITES FOR CONVERTING TO POSTGRESQL	5
CHAPTER 3	7
INFORMIX AND POSTGRESQL COMPARED	7
<i>General Comparison</i>	7
<i>Isolation Levels in Informix</i>	8
<i>Isolation Levels in PostgreSQL</i>	8
CHAPTER 4	9
CONNECTING TO POSTGRESQL.....	9
<i>Concepts</i>	9
<i>Environment Settings</i>	9
<i>Connecting to PostgreSQL through 4GL</i>	9
<i>Connecting to PostgreSQL from Windows</i>	13
Installing PostgreSQL	13
Adding a database	13
DSN Configuration	14
Creating a DSN	14
<i>Connecting to PostgreSQL from UNIX</i>	17
Basic Configuration.....	17
CHAPTER 5	19
PREPARING FOR THE POSTGRESQL CONVERSION	19
The Conversion Process	19
Migration of the Database Schema	19
Loading Table Data.....	20
Compiling 4GL Code	20
The qxexport Schema Extraction Tool	20
CHAPTER 6	21
INFORMIX COMPATIBILITY	21
<i>Unhandled Problems</i>	22
Matches.....	22
CHAR	23

Failed Statements in a Transaction.....	23
Explicit SERIAL Values.....	23
SPL Objects	24
<i>Problems Handled Through Emulation.....</i>	<i>25</i>
ROWID	25
WHERE CURRENT OF	25
<i>Problems Fully Handled.....</i>	<i>26</i>
Function/Operator Naming	26
Reserved Words as Identifiers	26
DDL Syntax	27
Join Syntax.....	27
SQL Error Codes	27
<i>Configurable Behaviour.....</i>	<i>27</i>
APPENDIX A	29
THE 'LYCIA' DYNAMIC SQL TRANSLATOR	29
<i>Introduction.....</i>	<i>29</i>
<i>Concepts</i>	<i>30</i>
Database Drivers	30
Bind Variables	30
Buffer Variables	32
<i>How Dynamic Translation Works.....</i>	<i>33</i>
SQL interpretation.....	33
SQL Generation.....	33
Datatype Mappings	34
The qx_\$\$schema table.....	34
Type Promotion	34
Benefits of Type Promotion.....	35
Areas where Type Promotion is used	35
APPENDIX C	37
DATATYPE MAPPINGS	37
APPENDIX D	41
COMMON CONVERSION ISSUES WHEN MIGRATING TO A DIFFERENT VENDOR'S DATABASES	41
System Catalogs.....	41
Matches	41
Cursor Names	42
Isolation Levels	42
Stored Procedures	42
INDEX	43

About This Guide

Who should read this Reference Guide?

This reference guide is intended to be used as a template for application developers who plan to modify existing applications or develop new applications to work with PostgreSQL.

This document assumes a background in Informix® development environments and Informix databases.


Conventions used in this book

Typefaces and Icons

Constant-width text is used for code examples and fragments, or command line functions.

Constant-width bold is used for emphasis.

Constant-width text is used for code sample variables.

	<p>This icon indicates a suggestion, discusses prerequisites for the action about to be taken, or indicates a warning about the use of available options.</p>
---	---

Significant code fragments are generally reproduced in a monospace font like this:

```
database cms
main
    display "Hello World"
end main
```

Code examples shown in this document may be used within any Querix applications – no special permission is required.

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 PostgreSQL.

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.

CHAPTER 2

Prerequisites for Converting to PostgreSQL

In order to use Lycia with PostgreSQL, you must have the following:

- PostgreSQL version 7.4.x or later installation (see <http://www.postgresql.org/> for downloads and installation instructions)
- 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

CHAPTER 3

Informix and PostgreSQL Compared

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

General Comparison

Subject	Informix	PostgreSQL
Product Range	Informix Standard Engine Informix Dynamic Server (Enterprise, Workgroup & Express Editions) Informix Online (Extended, Standard & Personal Editions)	PostgreSQL
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 Compliance	SQL-92 (entry level) SQL-99 (partial) Proprietary extensions	
Supported Connection Interfaces	Proprietary, ADO, OLEDB, ODBC, JDBC	
User Authentication	Operating System 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 honor 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 PostgreSQL

Read Committed (ANSI)	Places share locks on data being read. Default isolation level.
Read Uncommitted (ANSI)	Synonymous with Read Committed
Repeatable Read (ANSI)	Synonymous with Serializable
Serializable (ANSI)	Prevents external updates/inserts/deletes on read data.

CHAPTER 4

Connecting To PostgreSQL

Concepts

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

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.
QXSS_EMULATE_ROWID	false	When set to true, causes the runtime to emulate rowids in the RDBMS. See chapter 'Informix Compatibility' for more information on ROWID emulation.

Connecting to PostgreSQL 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 PostgreSQL/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
Description	A description for this DSN.
Driver	The psqlodbc driver file.
Trace	Whether ODBC tracing is enabled for this DSN. ODBC tracing can have a hugely negative impact on application performance.
TraceFile	The file to which the ODBC trace should be written.
Database	The database used in this connection.

ODBC Connect Option	Purpose
Servername	The name of the server hosting the PostgreSQL instance.
UserName	The username for this connection.
Password	The password associated with the user.
Port	The port for this connection. Default is 5432.
Protocol	The PostgreSQL protocol version. Default is 7.4.
ReadOnly	Database is read-only.
RowVersioning	Allow the row version column to be visible when cataloging.
ShowSystemTables	Allow system tables to be visible when cataloging the database (disable this option for performance optimization).
ShowOidColumn	Allow the Oid column to be visible when cataloging a table.
FakeOidIndex	
ConnSettings	Additional information to send to the server on connection completion.
SSLmode	Enable SSL encryption over the communication.
Fetch	Fetch max count.
Socket	Socket buffer size.
UnknownSizes	How to handle unknown result set sizes.
Maxvarcharsize	Declares the maximum size of a VARCHAR value. VARCHAR values greater than this length are considered to be of type LONGVARCHAR. Default is 255.
MaxLongvarcharsize	
Debug	
CommLog	Enable logging of the communication with the server.
Optimizer	Enable the generic optimizer in the server.
Ksqs	Keyset query optimization

ODBC Connect Option	Purpose
UseDeclareFetch	Use Declare/Fetch cursors
TextAsLongvarchar	Text types are represented as LONGVARCHAR
UnknownsAsLongvarchar	Unrecognised server/client types are represented as LONGVARCHAR
BoolAsChar	
Parse	
CancelAsFreeStmt	Causes the SQLCancel operation to also act as SQLFreeStmt (windows only).
ExtraSysTablesPrefixes	
LFConversion	Perform CR->CR/LF conversion
UpdatableCursors	
DisallowPremature	
TrueIsMinus1	Controls whether the value TRUE is seen as 1 or -1.
BI	Type used to represent the BIGINT type. Value should be the enumeration of one of the SQL_XXX types as defined in the ODBC driver manager.
ByteAsLongvarbinary	
UseServerSidePrepare	Controls whether statements are prepared by the driver or by the server.
LowerCaseIdentifier	Controls whether identifiers are treated as lower case or mixed case.
XaOpt	

Connecting to PostgreSQL from Windows

Installing PostgreSQL

For the installation of PostgreSQL see <http://www.postgresql.org/>. Here, you will find all the necessary downloads and extensive documentation.

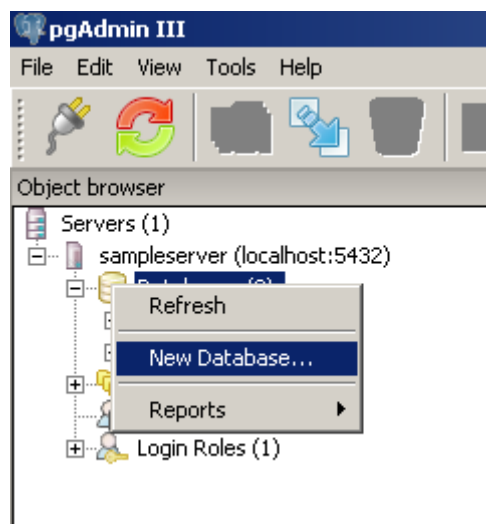
During installation of PostgreSQL, ensure that you note the values entered for Service Name, Account Name, Account Domain and the Account Password (if left blank, a password is generated automatically).

Adding a database

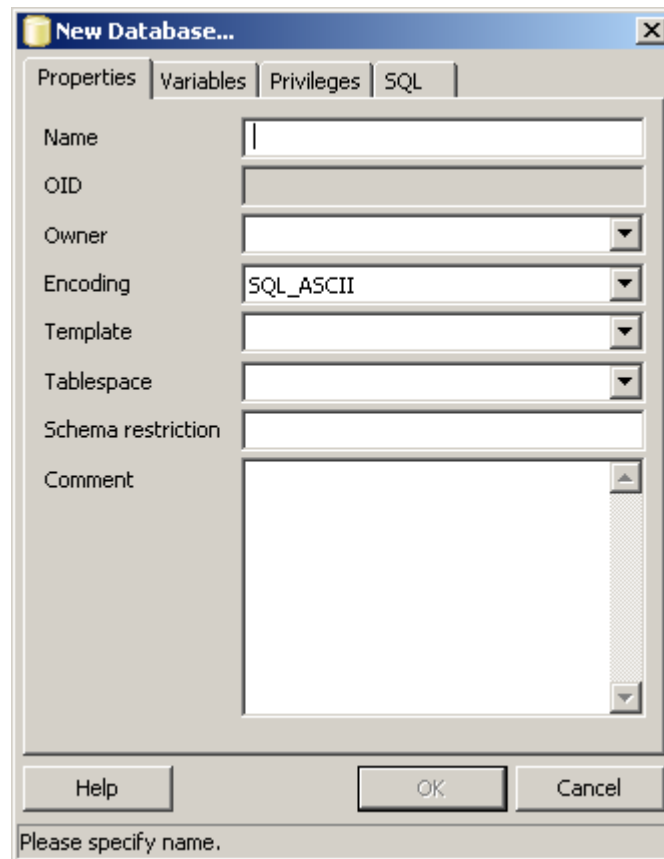
Once installed, go to the pgAdmin tool, found at:

All Programs -> PostgreSQL version_number -> pgAdmin III

There should be the name of the server you specified during installation in the left hand explorer window. Under this server there should also be a Databases group; right-click on the word Databases and select New Database.




In the box that opens, enter the database name and any other details as necessary and press OK.



DSN Configuration

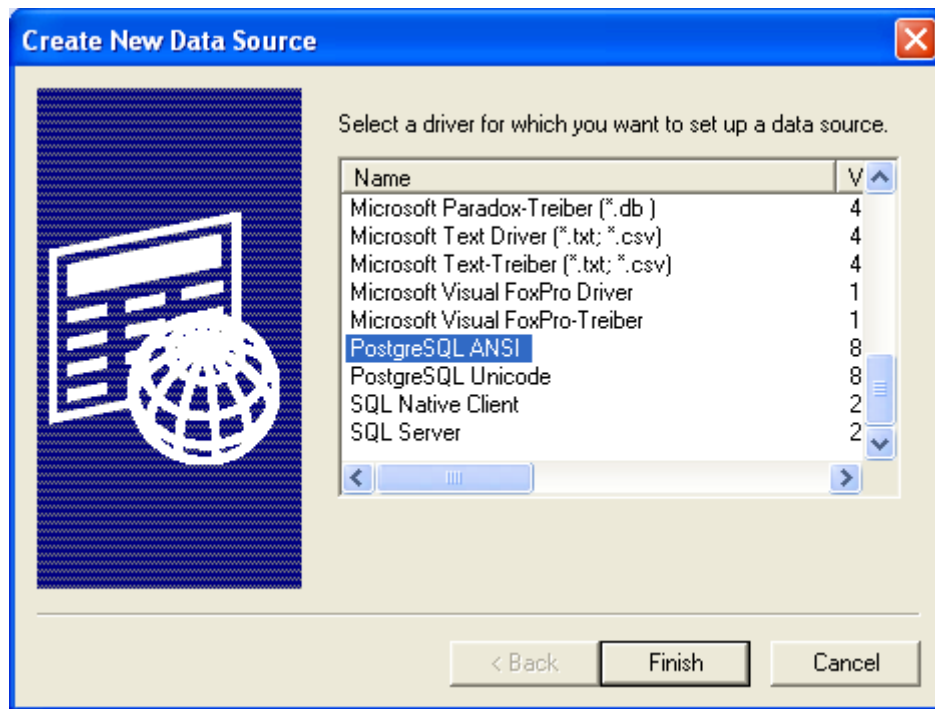
DSN (Data Source Names) are configured locally, through the ODBC data source manager located within 'Control Panel'. The ODBC data source manager can be found under:

Control Panel -> Administrative Tools -> Data Sources (ODBC)

	<p>It is generally recommended to use a System DSN when configuring a connection to PostgreSQL.</p>
---	---

Creating a DSN

Click on the System DSN tab and then click on Add. In the list that follows, scroll down and select the required PostgreSQL version (ANSI is recommended) then click on Finish.



In the screen that follows you will be asked to enter details of the ANSI ODBC Driver:

- **Data Source**

This is the location of the data that is to be used in the connection

- **Description**

Use this field to describe the data being used, e.g., Student Names

- **Database**

This is the name of the database that you intend to work with

- **SSL Mode**

Can be set to one of the following: Disable, Prefer, Allow or Require

- **Server**

If being run on the host machine, this will be localhost, otherwise this is the name of the server on which the database resides

- **Port**

Defaults to 5432 unless configured differently

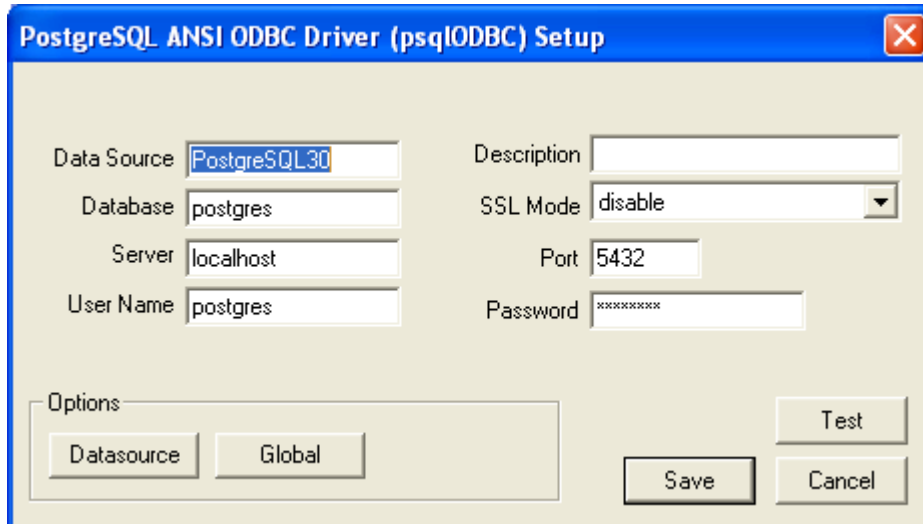
- **User Name**

Enter the user name here that was used when adding the database

- **Password**

If password protected, enter the password here

With this information correct, click Save to save the settings.



With this information saved, we need to set the environment variable in the GUI Servers in LyciaStudio. Go to Window->Properties->4GL->Run/Debug->GUI Servers, select a server and press Environment button. You will see environment variables, if the environment variable ODBC_DSN is not already there, enter ODBC_DSN=< PostgreSQL DSN> into the file.

When the server name is set in the environment variables, any attempt to connect to a database will succeed as the data source is specified in the ODBC driver, which also has the name of the database to connect to if the database specified in the 4GL does not exist.

Connecting to PostgreSQL from UNIX

Whichever ODBC driver manager you are using on UNIX, the DSN configuration is found within the file 'odbc.ini'. The files /etc/odbc.ini and /etc/odbcinst.ini are considered to be the sources for system DSNs, and the file ~/.odbc.ini (note the leading '.') is considered to be a source for user DSNs.

Typically the odbc.ini file search order is considered to be:

- \$ODBCINI
- ~/.odbc.ini
- /home/.odbc.ini
- /etc/odbc.ini

The format of the file is simple, consisting of a header declaring the data sources within the file, then a list of configuration for each data source. For example

```
[ODBC Data Sources]
DSN1 = Description for DSN 1
DSN2 = Description for DSN 2
...

[DSN1]
# configuration for DSN1

[DSN2]
# configuration for DSN2
```

The configuration for the DSN is composed of the ODBC connection options listed earlier in this chapter.

Basic Configuration

```
[ODBC Data Sources]
cms = PostgreSQL cms database

[cms]
Description = CMS sample database under PostgreSQL
Driver = /usr/local/lib/psqlodbc.so
Database = cms
Servername = localhost
Username =
Password =
Port = 5432
Protocol = 6.4
```


CHAPTER 5

Preparing for the PostgreSQL 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 chapter 'Connecting To PostgreSQL'.

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 PostgreSQL). 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 LyciaStudio 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 that are not handled
- Problems that are handled through 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

Unhandled Problems

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.

- 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]*'	No equivalent.
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.

PostgreSQL also provides the '~' operator. This operator provides POSIX regular expression matching facilities. The Lycia Dynamic SQL Translator will automatically supplement the MATCHES operator for the '~' operator, however there are a number of problems to note in the pattern being matched.

- Unlike LIKE / MATCHES, the '~' operations matches the pattern anywhere within the string. For this reason, the '^' character should be used to specify that a pattern should start at the beginning of the string only.

- Unlike MATCHES, the '*' wildcard matches multiple occurrences of the preceding expression. The pattern '.' is equivalent to the use of '*' in MATCHES
- Unlike MATCHES, the '?' wildcard matches 0 or 1 occurrences of the preceding expression. The pattern '.' is equivalent to the use of '?' in MATCHES.

MATCHES Expression	~ Expression
WHERE table1.col MATCHES 'ABCD*'	WHERE table1.col ~ '^ABCD.*'
WHERE table1.col MATCHES 'ABCD?'	WHERE table1.col ~ '^ABCD.'
WHERE table1.col MATCHES '*ABCD'	WHERE table1.col ~ '.*ABCD'
WHERE table1.col MATCHES '?ABCD'	WHERE table1.col ~ '^.*ABCD'
WHERE table1.col MATCHES 'ABCD[0-9]'	WHERE table1.col ~ '^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.

CHAR

The CHAR data type is limited to 8104 characters in PostgreSQL, whereas for Informix it is restricted to 32,267.

Failed Statements in a Transaction

Under some versions of PostgreSQL, a failed statement within a transaction will cause all subsequent SQL statements to fail with an error until the transaction is closed.

Explicit SERIAL Values

When inserting explicit values into a SERIAL column, PostgreSQL will not update the next generated value. For example

```
CREATE TABLE test1 (
  a SERIAL(100),
  ...
)

INSERT INTO test1 VALUES (150, ...)
```

Under these circumstances, the next value generated when inserting '0' into the serial column under Informix would be 151. PostgreSQL, however, will generate 100 as the next serial value.

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.

ROWID

PostgreSQL provides a ROWID-like facility by placing optionally placing OID pseudo-columns on tables. By default, the Lycia Database Interface for PostgreSQL will attempt to use the OID column as a ROWID (automatically renaming all references to ROWIDs within SQL as appropriate).

In addition, a facility to emulate Informix ROWID behaviour is also provided. If this facility is enabled, the OID column(s) will be ignored, and the emulated ROWID used instead.

	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.
---	---

WHERE CURRENT OF

PostgreSQL does not have an equivalent of the Informix WHERE CURRENT OF statement. This behaviour will be emulated automatically using the ROWID mechanism in use (whether it is OID or emulated ROWID).

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 PostgreSQL. This includes such things as:

- ALTER TABLE
- Constraint naming

Join Syntax

PostgreSQL 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.

SQL Error Codes

4GL applications are largely dependant upon expected error codes from Informix. PostgreSQL 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.

Configurable Behaviour

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

dbi.postgresql.rowid = (emulate | oid | none)

This option determines the method for handling ROWIDs. This option defaults to 'oid' (using the PostgreSQL built-in OID features).

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.

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.

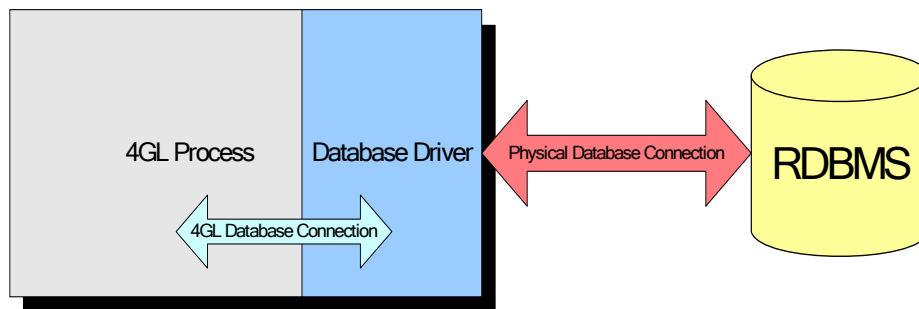
The overall result is that 4GL and ESQL/C programs can be used with a wide range of RDBMSs with the absolute minimum of code modification.



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



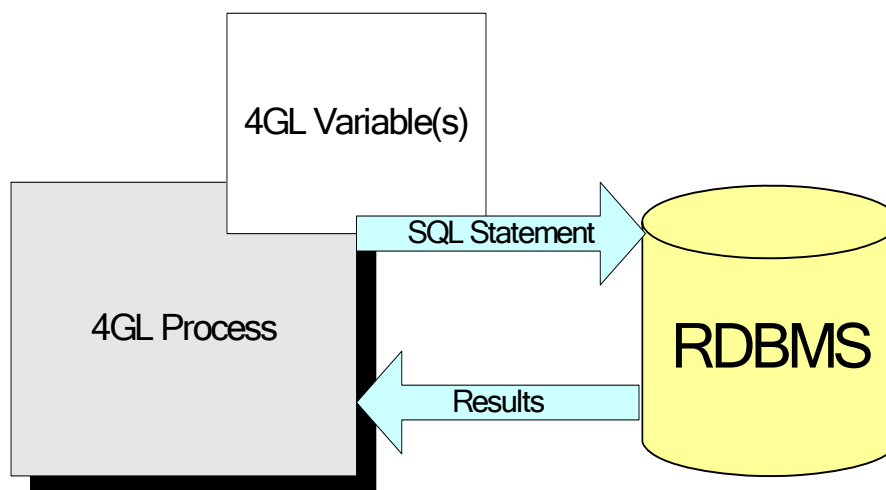
Each database vendor provides a different means of connecting to and communicating with a database. For this reason, it is not possible to use the same communication mechanism with, for example, PostgreSQL as you would with Informix.

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
```

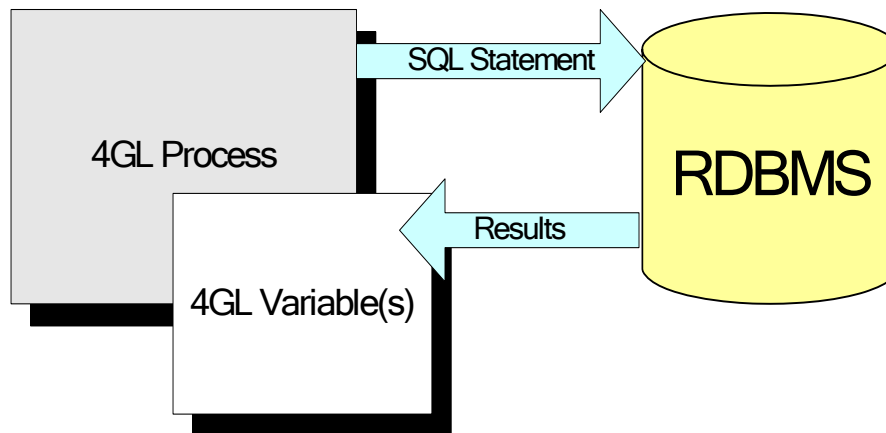
In this example, both the literal value '1' and the 4GL variable 'my_int' are considered bind variables for the SQL statement.

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

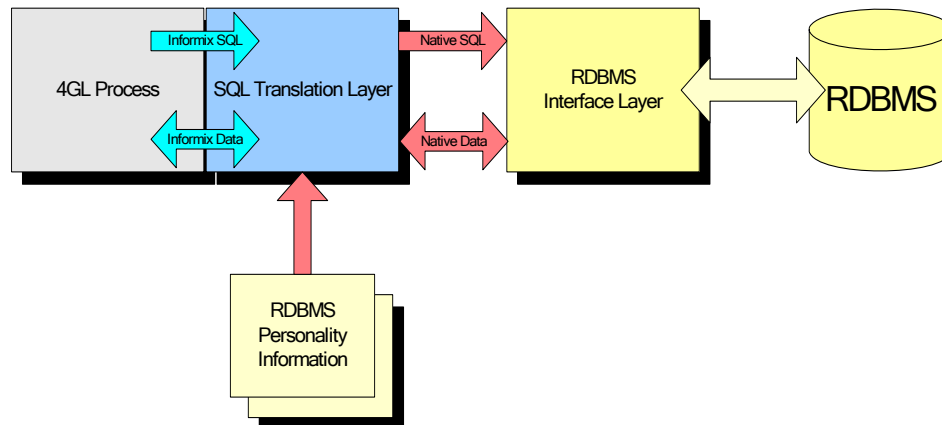
FOREACH c1 INTO rec1.*
  ...
```

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

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 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 PostgreSQL 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	PostgreSQL Datatype	Notes
CHAR	CHAR	CHAR	
VARCHAR	VARCHAR	VARCHAR	
LVARCHAR	VARCHAR	VARCHAR	
NCHAR	NCHAR	CHAR	
NVARCHAR	NVARCHAR	VARCHAR	
SMALLINT	SMALLINT	SMALLINT	
INTEGER	INTEGER	INTEGER	
INTEGER8	INTEGER	BIGINT	
SERIAL	INTEGER	SERIAL	
SERIAL8	INTEGER	BIGSERIAL	
FLOAT	FLOAT	DOUBLE PRECISION	
SMALLFLOAT	SMALLFLOAT	REAL	
DECIMAL	DECIMAL	DECIMAL	
MONEY	MONEY	DECIMAL	
DATE	DATE	DATE	
DATETIME	DATETIME	TIMESTAMP	
INTERVAL	INTERVAL	INTERVAL	
BYTE	BYTE	BYTEA	

Informix SQL Datatype	Informix 4GL Datatype	PostgreSQL Datatype	Notes
TEXT	TEXT	TEXT	

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

PostgreSQL	Informix 4GL	Informix SQL	Notes
BIGINT	BIGINT	BIGINT	
BIGSERIAL	BIGINT	SERIAL8	
BOOLEAN	CHAR	CHAR	
BYTEA	BYTE	BYTE	
CHAR	CHAR	CHAR	
DATE	DATE	DATE	
DECIMAL / NUMERIC	DECIMAL	DECIMAL	
DOUBLE PRECISION	FLOAT	FLOAT	
INTERVAL	INTERVAL	INTERVAL	
MONEY	MONEY	MONEY	
REAL	SMALLFLOAT	SMALLFLOAT	
SERIAL	INTEGER	SERIAL	
TEXT	TEXT	TEXT	
TIME	DATETIME	DATETIME	
TIMESTAMP	DATETIME	DATETIME	
UUID	CHAR	CHAR	

PostgreSQL	Informix 4GL	Informix SQL	Notes
VARCHAR	VARCHAR	VARCHAR	

APPENDIX D

Common Conversion Issues when migrating to a 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 Catalogs

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 is 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

ANSI_NULL_DEFAULT	37	MONEY	39
ANSI_NULLS	37	NCHAR.....	39
ANSI_WARNINGS	37	NVARCHAR	39
Authentication	7	ODBC Connect Option	10
Bind Variables	30	ODBC_DSN	9
Buffer Variables	32	Preparing	19
Collation Sequences.....	38	Prerequisites	5
Common Conversion	43	Problems Fully Handled.....	26
compared	7	Problems Handled Through Emulation.....	25
Comparison.....	7	Product Range	7
Compiling 4GL code	20	qx__\$schema table	34
CONCAT_NULL_YIELDS_NULL	37	qxexport	20
Configuring Options	37	QXORA_DB_IS_DSN	9
Connecting.....	9, 20	QXORA_EMULATE_ROWID	9
Conversion.....	19	Read Committed (ANSI).....	8
Creating a DSN.....	15	Read Uncommitted (ANSI).....	8
Cursor Names	44	Repeatable Read.....	8
Cursor Stability.....	8	Repeatable Read (ANSI).....	8
CURSOR_CLOSE_ON_COMMIT.....	37	ROWID	25
Database Creation Options	37	schema object.....	19
Database Drivers.....	30	SERIAL8.....	39
Datatype Mappings.....	34, 39	Serializable (ANSI).....	8
DATETIME.....	39	SQL Compliance.....	7
Dirty Read	8	SQL Error Codes.....	27
DSN Configuration	14	SQL Generation	33
Dynamic SQL Translator.....	29	SQL interpretation.....	33
Environment Settings.....	9	SSTRACE	9
Informix Compatibility	21	Storage	7
Informix datatypes.....	19	Stored Procedures	44
INTEGER8	39	Supported Connection Interfaces	7
INTERVAL	40	System Catalogs.....	43
Isolation Levels.....	7, 8, 44	Translation	33
Loading table data	20	Type Promotion.....	34
Matches.....	43	Unhandled Problems	22
Migration of the Database Schema	19	Unix.....	17