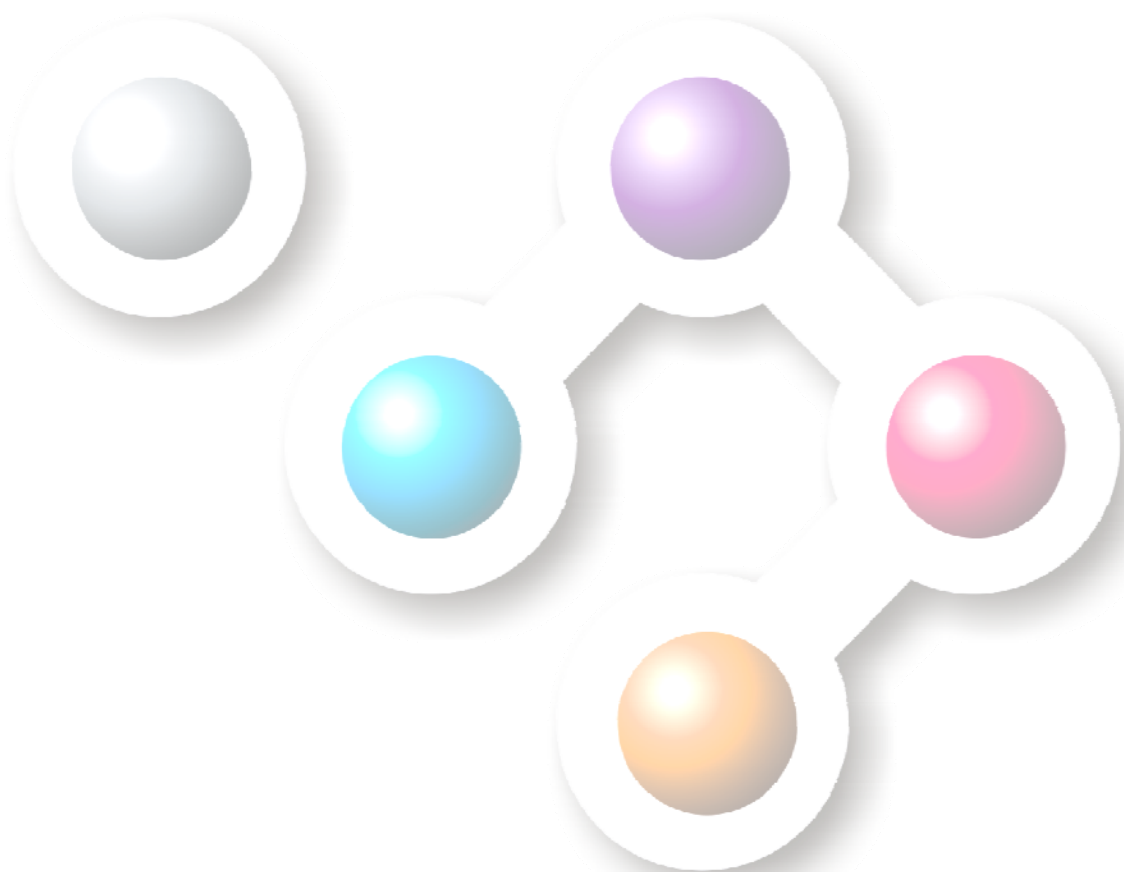


Lycia Development Suite



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



Dynamic Database Interface

Migration Guide for MySQL

Versions 5 and 6

January 2011

Part Number: 016-006-135-011

Querix Dynamic Database Guide for MySQL

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

January 2011 Part Number: 016-006-135-011

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 MySQL, updated for versions 4.3 and retitled 'Database Migration Guide for MySQL'
February 2009:	Updated for v4.4
January 2011:	Updated for v5 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.....	1
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	4
PREREQUISITES FOR CONVERTING TO MYSQL.....	4
CHAPTER 3	5
INFORMIX AND MYSQL COMPARED.....	5
<i>General Comparison</i>	5
<i>Isolation Levels in Informix</i>	6
<i>Isolation Levels in MySQL</i>	6
CHAPTER 4	7
CONNECTING TO MYSQL	7
<i>Concepts</i>	7
<i>Environment Settings</i>	7
<i>Connecting to MySQL through 4GL</i>	7
<i>Connecting to MySQL from Windows</i>	10
Configuring the MySQL Server Instance	10
Creating a database and user	18
Configuring the DSN.....	19
<i>Connecting to MySQL from Unix</i>	22
Basic Configuration.....	22
CHAPTER 5	23
PREPARING FOR THE MYSQL CONVERSION	23
The Conversion Process	23
Migration of the Database Schema	23
Loading Table Data.....	24
Compiling 4GL Code	24
The qxexport Schema Extraction Tool	24
CHAPTER 6	25
INFORMIX COMPATIBILITY	25
<i>Unhandled Problems</i>	26
Valid Column Defaults	26
Matches.....	26
Serial Columns and primary keys	27

DROP INDEX Syntax.....	28
SPL Objects	28
<i>Problems Handled Through Emulation</i>	29
<i>Problems Fully Handled</i>	30
Function / Operator Naming.....	30
DDL Syntax	30
SQL Error Codes	31
APPENDIX A	33
THE 'LYCIA' DYNAMIC SQL TRANSLATOR	33
<i>Introduction</i>	33
<i>Concepts</i>	34
Database Drivers	34
Bind Variables	34
Buffer Variables	36
<i>How Dynamic Translation Works</i>	37
SQL interpretation.....	37
SQL Generation.....	37
Datatype Mappings	38
The qx_\$\$schema table.....	38
Type Promotion	38
Benefits of Type Promotion.....	39
Areas where Type Promotion is used	39
APPENDIX B	41
CONFIGURING OPTIONS	41
APPENDIX C	43
DATATYPE MAPPINGS	43
APPENDIX D	47
COMMON CONVERSION ISSUES WHEN MIGRATING TO A DIFFERENT VENDOR'S DATABASES	47
System Catalogs.....	47
Matches	47
Cursor Names	48
Isolation Levels.....	48
Stored Procedures	48
INDEX.....	49

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 MySQL.

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.

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

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 Lycia Development tools to open up for modern GUI screen interactions and databases such as MySQL.

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 MySQL

In order to use Lycia with MySQL you will need the following installed on your system:

- MySQL database software (found at <http://www.mysql.com/products/>)
- MySQL Connector/ODBC (found at <http://www.mysql.com/products/connector/odbc/index.html>)
- A full installation of the Lycia 4.5 suite
- 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 MySQL Compared

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

General Comparison

Subject	Informix	MySQL
Product Range	Informix Standard Engine Informix Dynamic Server (Enterprise, Workgroup & Express Editions) Informix Online (Extended, Standard & Personal Editions)	
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	

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 MySQL

Read Committed	Does not place or honor table locks whilst reading data, and will read uncommitted data.
Read Uncommitted	Places no locks on data being read, but only reads data that has been committed. This is the default isolation level.
Repeatable Read	Reads data in the snapshot captured during at the beginning of the transaction.
Serializable	Similar to repeatable read, however all select statements will lock in share mode.

CHAPTER 4

Connecting To MySQL

Concepts

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

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		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 MySQL 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 MySQL/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'.

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

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 MySQL. 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.

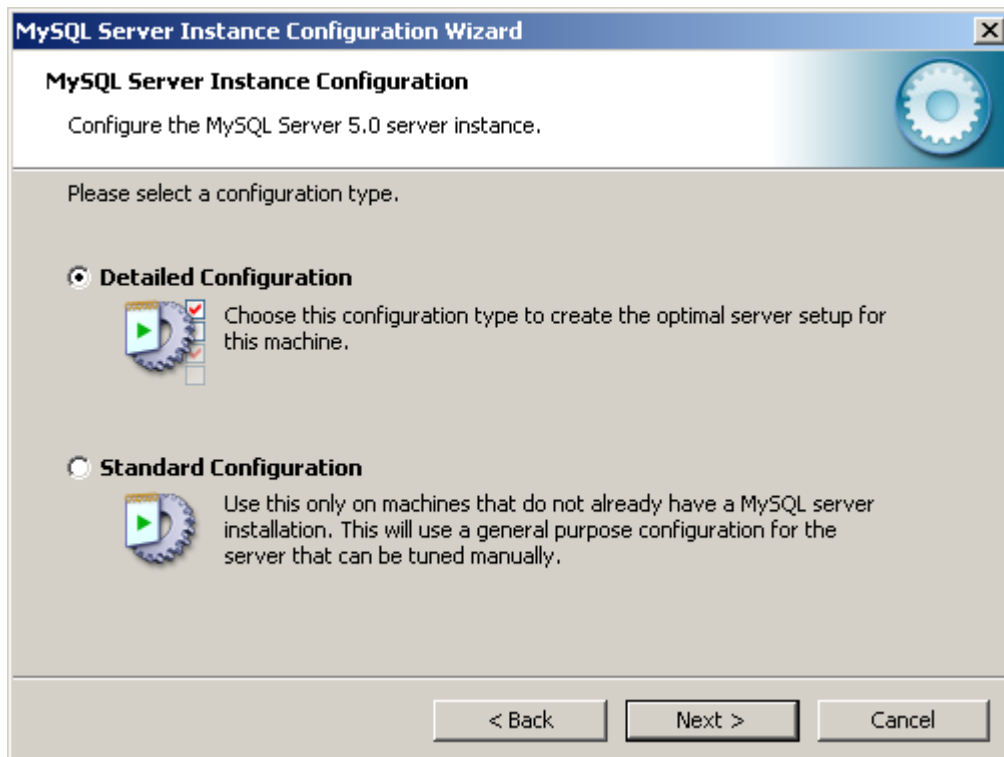
ODBC Connect Option	Purpose
AutoTranslate	When set to true, character data is converted to Unicode through the ANSI code page. The default for this value is true.
DATABASE	The name of the database to connect to. This option is automatically handled by the Lycia runtime, and shouldn't generally be provided in an ODBC connection string. This parameter is overridden by the default database specified within the DSN (if any).
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 MySQL 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 MySQL.
QuotedID	When yes, QUOTED_IDENTIFIERS is set ON for the connection, MySQL expects all quoted literal values to use single quotes, reserving double quotes for delimiting identifiers.
Regional	When yes, the MySQL 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 MySQL 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 MySQL ODBC driver to use Windows Authentication Mode for login validation. The UID and PWD keywords are optional. When no, instructs the MySQL ODBC driver to use a MySQL username and password for login validation. The UID and PWD keywords must be specified.
UID	Name of the MySQL 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 MySQL from Windows

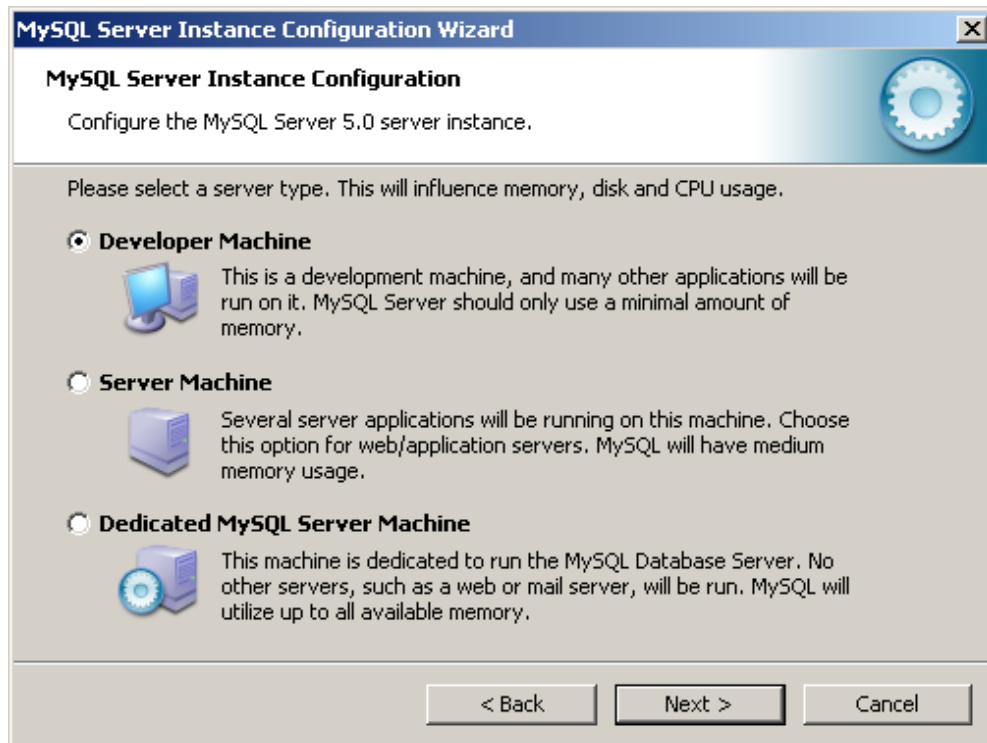
Configuring the MySQL Server Instance

To set up an ODBC connection you will need to download and install MySQL Community Server and MySQL Connector (ODBC). Firstly, install MySQL onto your system and then run the MySQL Server Instance Configuration Wizard. This can either be done straight after the install, or by clicking on Start -> All Programs -> MySQL -> MySQL Server 5.0 -> MySQL Server Instance Config Wizard. The following instructions show a typical configuration:

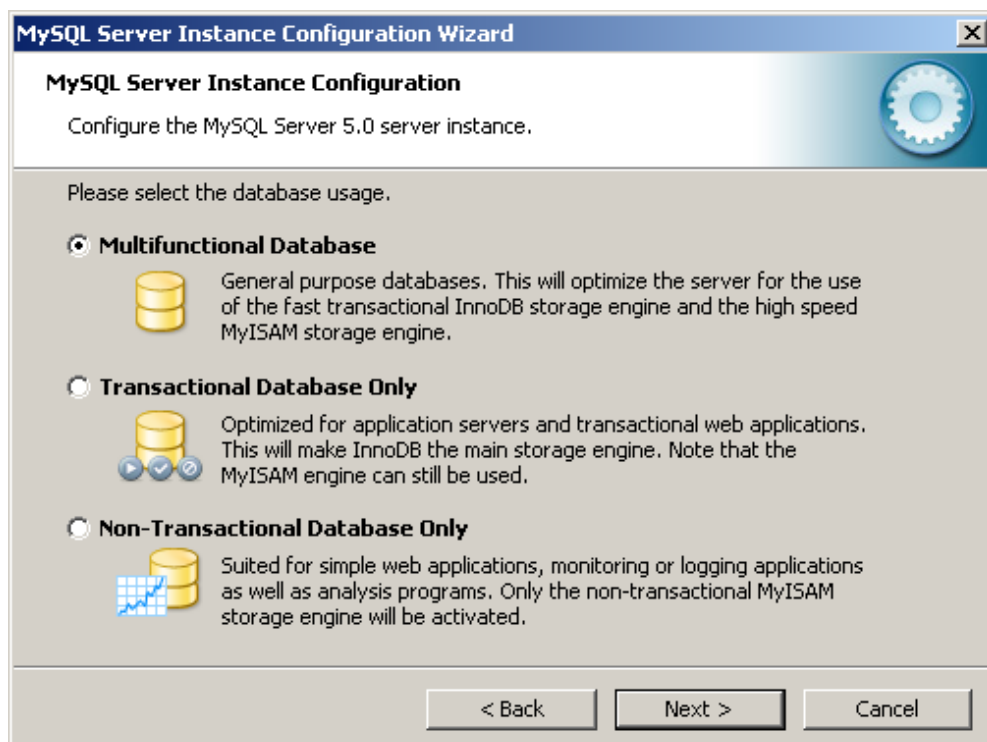
1. Click on Next to get to the following screen, choose Detailed Configuration and click Next



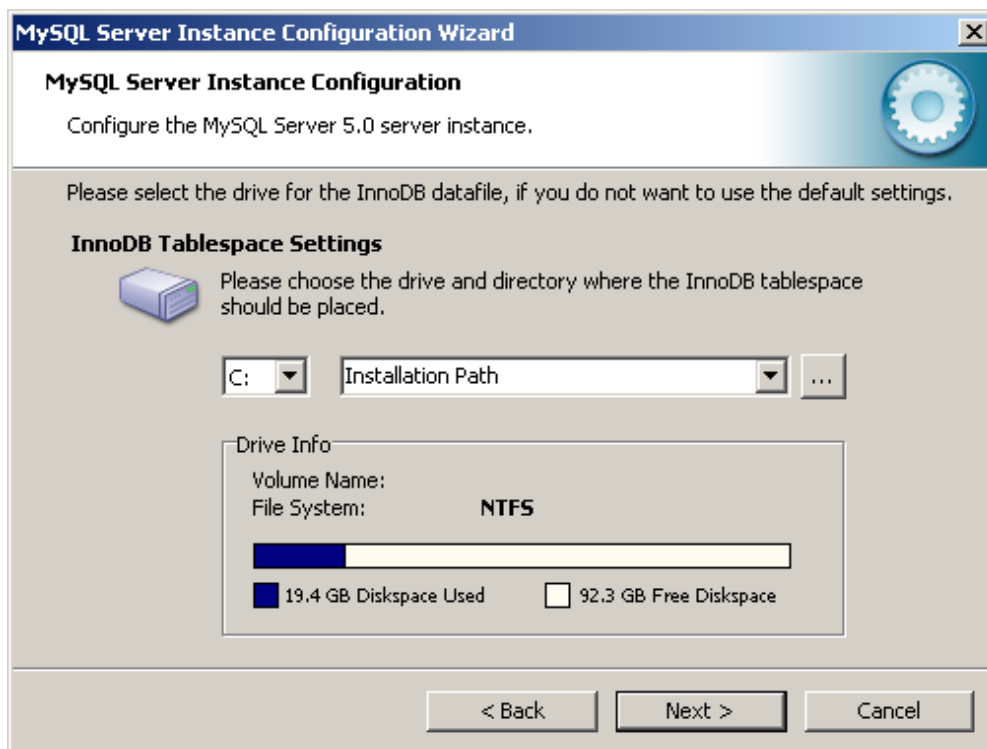
2. For most configurations the server type will be a Developer Machine. Select this and click Next



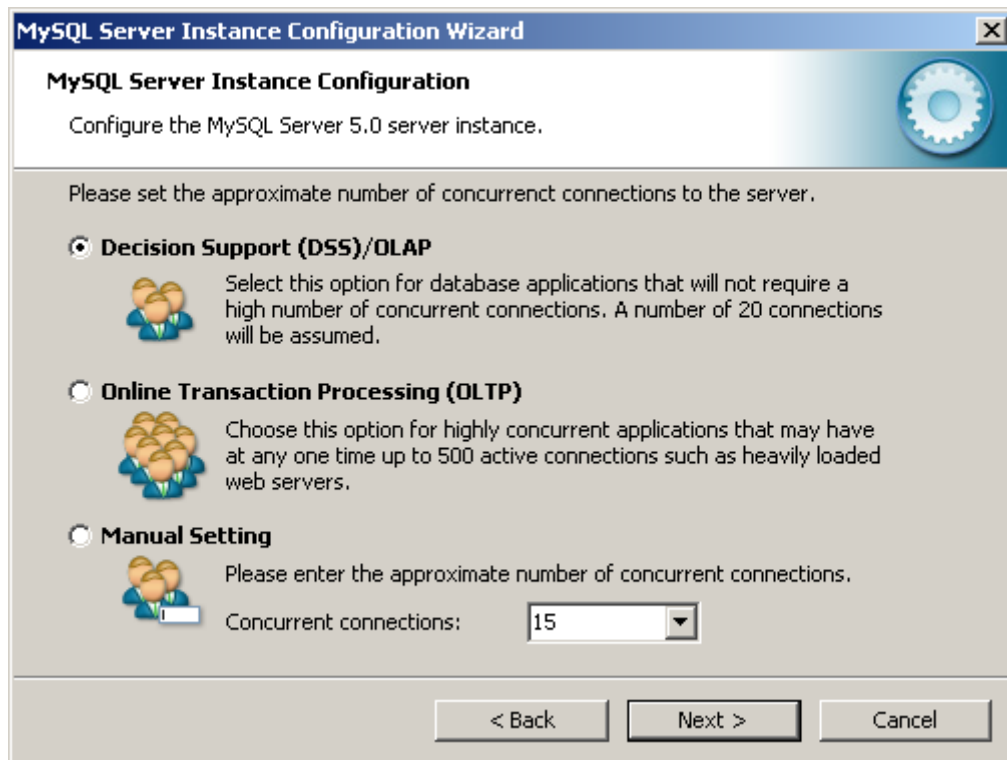
3. The following screen asks you to select the appropriate usage for the database. In this case we are going to go with a Multifunctional Database. Click Next.



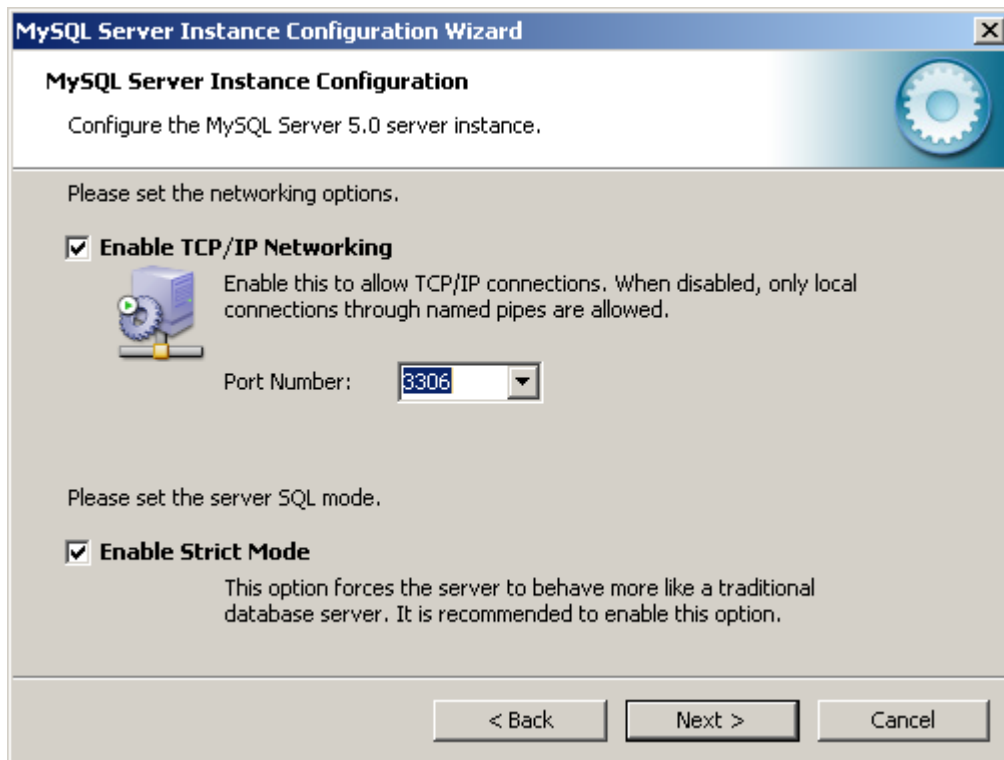
- The following screen asks you to choose the drive and directory for the InnoDB tablespace. Again, we will keep the default. Click Next.



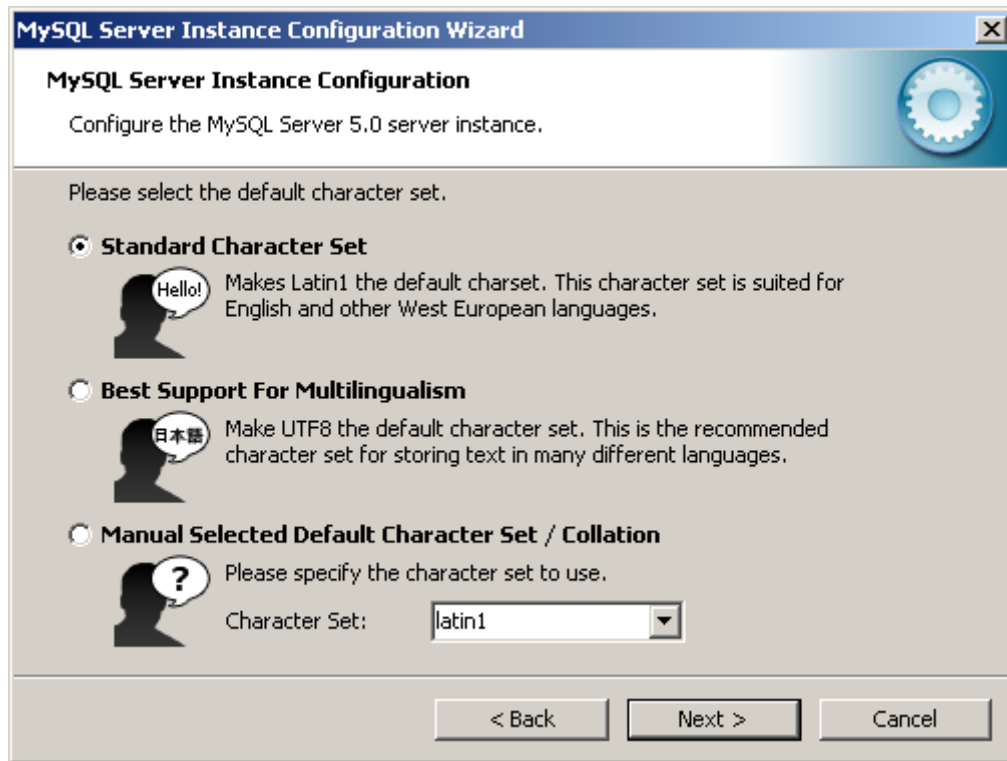
5. Now you will need to decide approximately how many concurrent connection there will be to the server. We will choose Decision Support(DSS)/OLAP which allows up to 20 connections.



- Now choose whether to enable TCP/IP networking and whether to enable strict mode.
We will enable both.



7. Now choose the character set most likely to be required for this server. We will choose the Standard Character Set



- Next, select the required Windows options. We will select Install As Windows Service.



9. The following screen requires you to choose a root password and decide whether to create an anonymous account. Enter your password, confirm the password and click Next.




MySQL Server Instance Configuration Wizard


MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please set the security options.

☒ **Modify Security Settings**

 New root password: Enter the root password.

 Confirm: Retype the password.

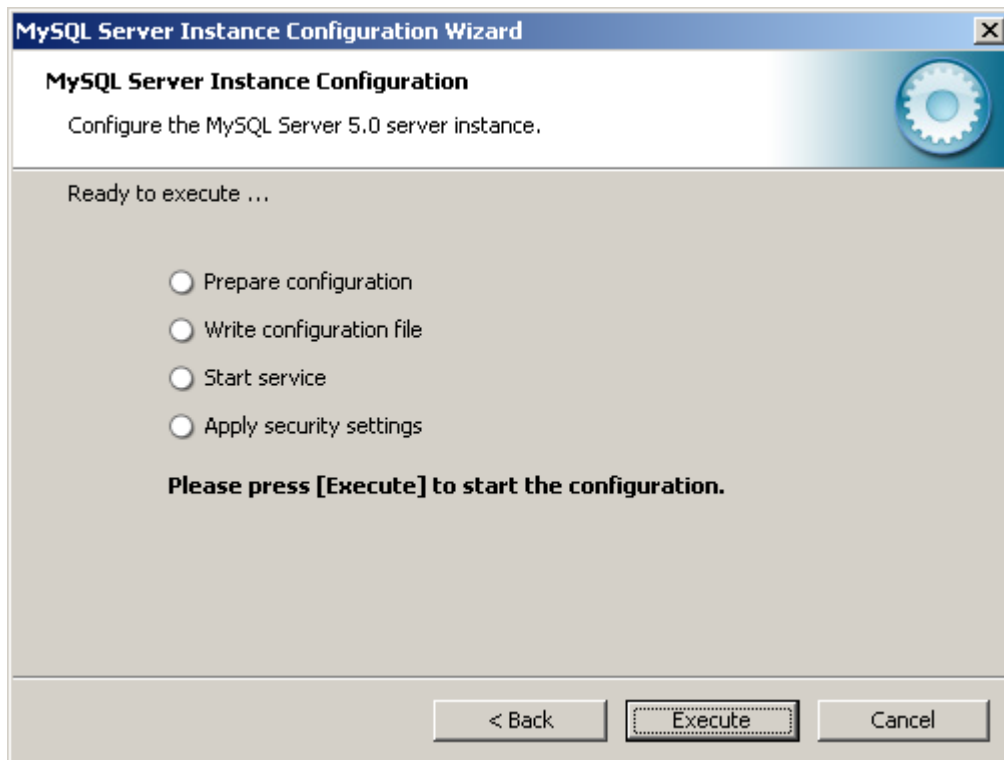
☐ Enable root access from remote machines

☐ **Create An Anonymous Account**

 This option will create an anonymous account on this server. Please note that this can lead to an insecure system.

< Back Next > Cancel

10. Finally, once all the information is entered press Execute on the next screen to create the configuration and Finish once complete.



Creating a database and user

11. Having created the server instance, we now need to create a database and a user. To do this go to the MySQL Command Line Client found at Start -> All Programs -> MySQL -> MySQL Server 5.0 -> MySQL Command Line Client. The first thing you will need to do is to enter the password you entered during the Server Instance Configuration. Once logged on, you will then need to create a database using the following command, where DB_NAME refers to the name of the database you wish to create:

```
create database DB_NAME;
```

12. Secondly, you will need to create a user that will have access to this database. Use the following command, where USERNAME is the name of the user to be created:

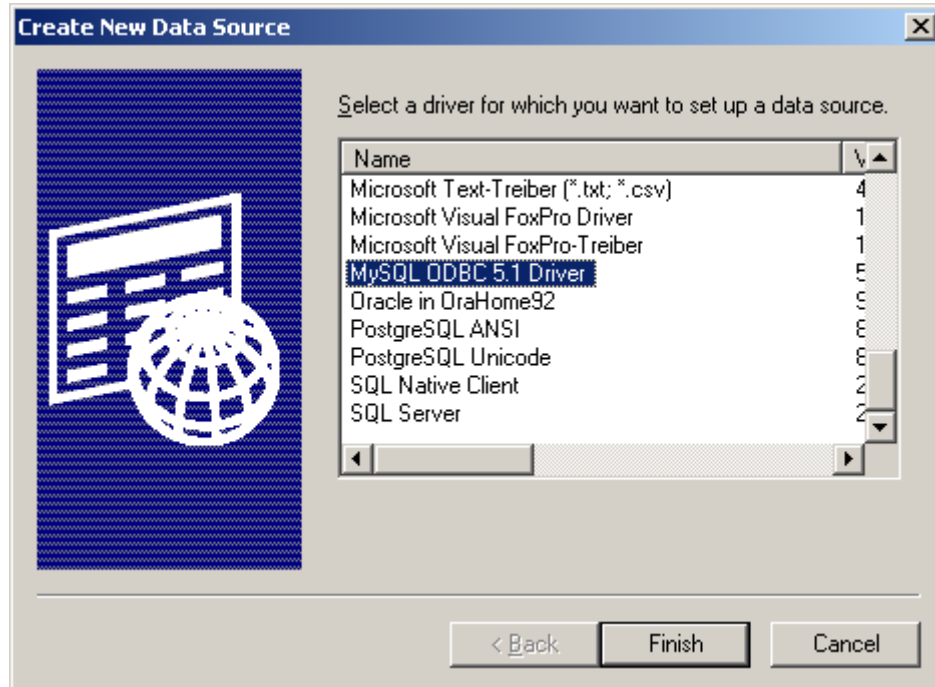
```
create user USERNAME;
```

13. Now, with the database and user created, we need to give access to the database to the user. This is done using the following command, where PASSWORD is the original password entered previously:

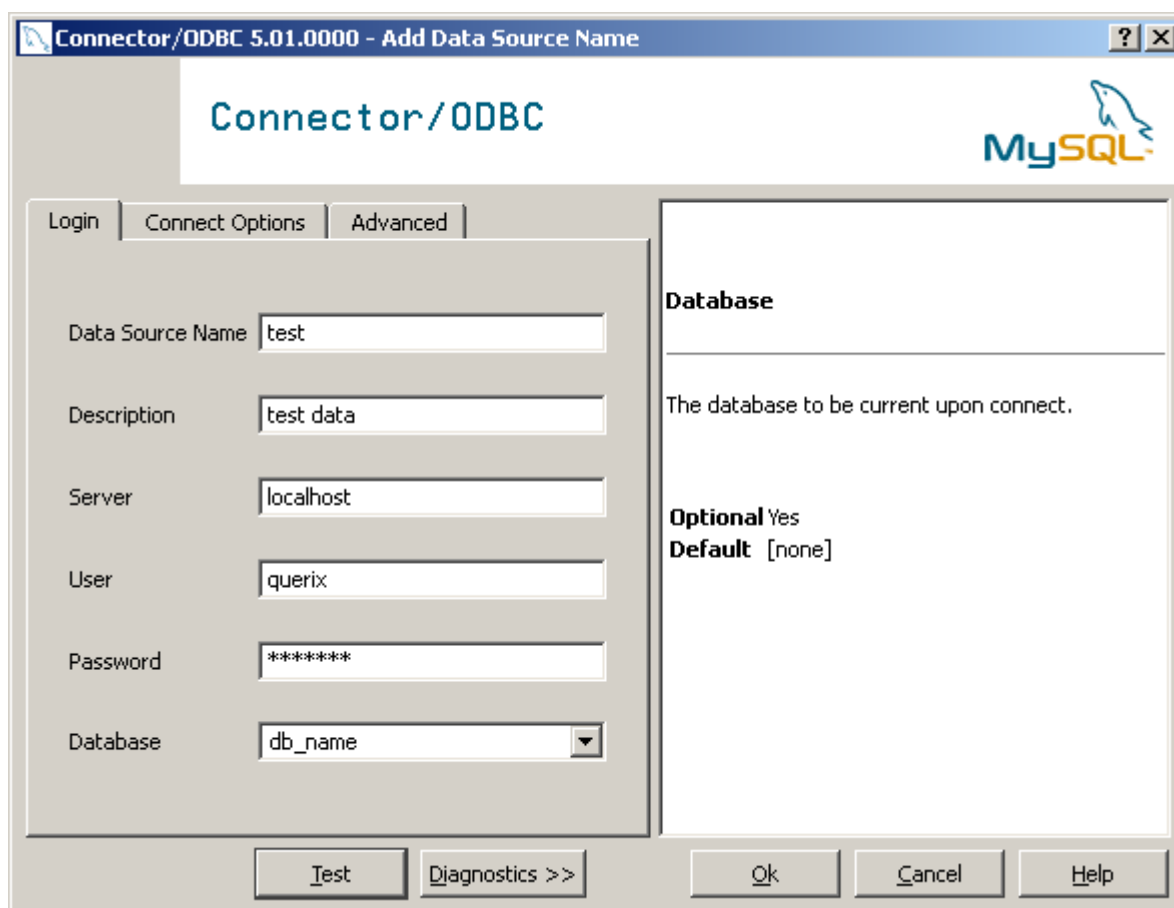
```
grant all on DB_NAME.* to USERNAME identified by PASSWORD
```

Configuring the DSN

14. With the server instance set up and the database and user created, we will now need to configure the DSN. Install the MySQL Connector/ODBC and then go to Control Panel -> Administrative Tools -> Data Sources (ODBC). Then click on the System DSN tab and then Add. In the screen that pops up, select the version of MySQL that you have installed and click Finish.



15. This will bring up the Connector/ODBC screen which allows you to enter details about the ODBC connection.



- Data Source Name
 - This will be the name by which you will refer to the database in the 4GL
- Description
 - This is an optional field that allows to enter a description of the data being stored, for example, "list of student names"
- Server
 - The name of the server on which the data will reside. If this is not the localhost then you will need to enter connection details in the tab Connect Options
- User
 - This is the name of the user that was created in the Command Line Client
- Password
 - The password that was entered during the instance creation

- Database
 - This is the database to be current on connection, use the database that was created using the Command Line Client

Having completed all of the above, you are now able to connect to the database using the 4GL DATABASE statement and the DSN for use in your programs.

Connecting to MySQL 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 = MySQL cms database

[cms]
Description = CMS sample database under MySQL
Driver = /usr/local/lib/mysql/libmysqlclient.so
Database = cms
Servername = localhost
Username =
Password =
Port = 3306
Protocol = 6.4
```

CHAPTER 5

Preparing for the MySQL 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.



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

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 MySQL'.

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 MySQL). 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

Valid Column Defaults

MySQL does not allow function / expression values as column defaults. This means that the following Informix expression values cannot be used as column defaults in MySQL:

- TODAY
- CURRENT
- USER

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.

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

- Unlike LIKE / MATCHES, the RLIKE operator 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	RLIKE Expression
WHERE table1.col MATCHES 'ABCD*'	WHERE table1.col RLIKE '^ABCD.*'
WHERE table1.col MATCHES 'ABCD?'	WHERE table1.col RLIKE '^ABCD.'
WHERE table1.col MATCHES '*ABCD'	WHERE table1.col RLIKE '.*ABCD'
WHERE table1.col MATCHES '?ABCD'	WHERE table1.col RLIKE '^ABCD'
WHERE table1.col MATCHES 'ABCD[0-9]'	WHERE table1.col RLIKE '^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.

Serial Columns and primary keys

In MySQL if you have a table containing a serial column, it must be part of a primary key. This restriction is not in place in Informix. For example, the table below contains a serial column:

```
CREATE TABLE t1(
  a SERIAL,
  b INTEGER,
  ...
)
```

In the above table, t1, there is no primary key clause. Lycia will in this case, for MySQL tables, automatically create a primary key clause on the serial column, as below:

```
CREATE TABLE t1(
```

```
    a SERIAL PRIMARY KEY,  
    b INTEGER,  
    ...  
)
```

A further restriction in MySQL is that you cannot have duplicate values in the serial column, 'a' in the above table.

In the case where a serial column is present and there is also a primary key clause which uses the serial column, the primary key clause will not be altered:

```
CREATE TABLE t2(  
    a SERIAL,  
    b INTEGER,  
    ...  
    PRIMARY KEY(a,b)  
)
```

In the situation below, where a primary key is created, but it doesn't use the serial column, a table creation error will occur. This has to be modified in MySQL to rectify this.

```
CREATE TABLE t3(  
    a SERIAL,  
    b INTEGER,  
    ...  
    PRIMARY KEY(b,c)  
)
```

DROP INDEX Syntax

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

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

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.

DDL Syntax

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

- ALTER TABLE
- Constraint naming definitions

The dynamic SQL translator automatically converts all DDL from the Informix syntax to the Oracle syntax.

Note:

MySQL does not support multiple actions in an ALTER TABLE statement. For example, the Informix statement

```
ALTER TABLE table1 ADD (col1 INTEGER, col2 INTEGER)
```

Will generate a runtime error in MySQL, as the action consists of 2 actions (adding col1 and adding col2). The statement must be rewritten to perform each action atomically, for example:

```
ALTER TABLE table1 ADD (col1 INTEGER)
```

```
ALTER TABLE table1 ADD (col2 INTEGER)
```

SQL Error Codes

4GL applications are largely dependent upon expected error codes from Informix. MySQL 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].

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.

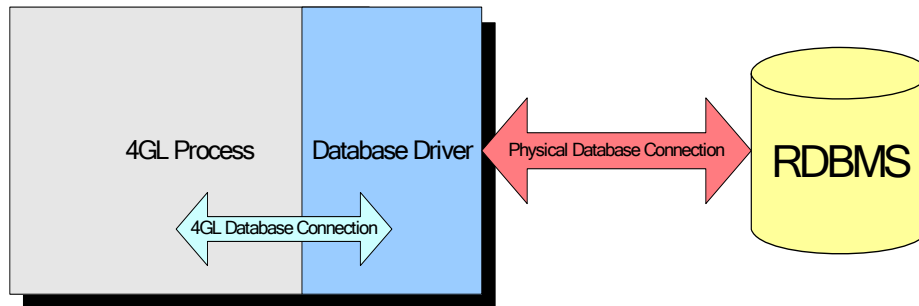
The overall result is that 4GL and ESQL/C programs can be used with a wide range of RDBMSs with the absolute minimum of code modification.



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



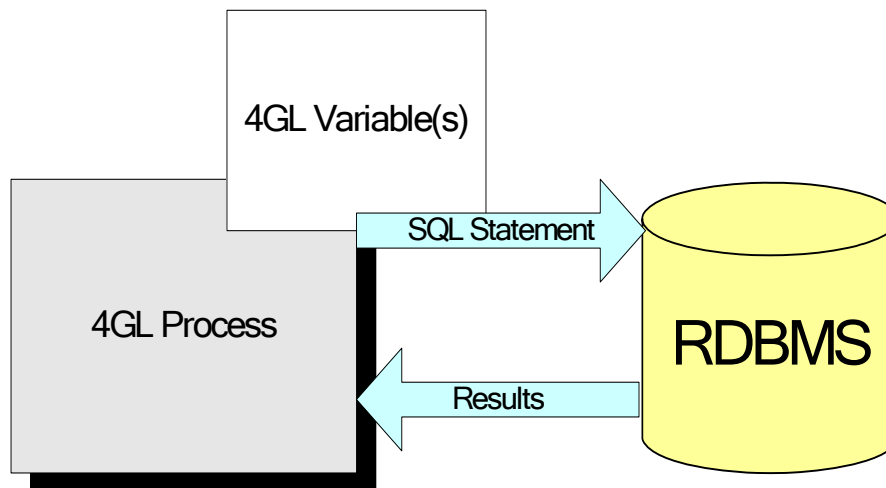
Each database vendor provides a different means of connecting to and communicating with a database. For this reason, it is not possible to use the same communication mechanism with, for example, SQL-Server as you would with Informix.

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
```

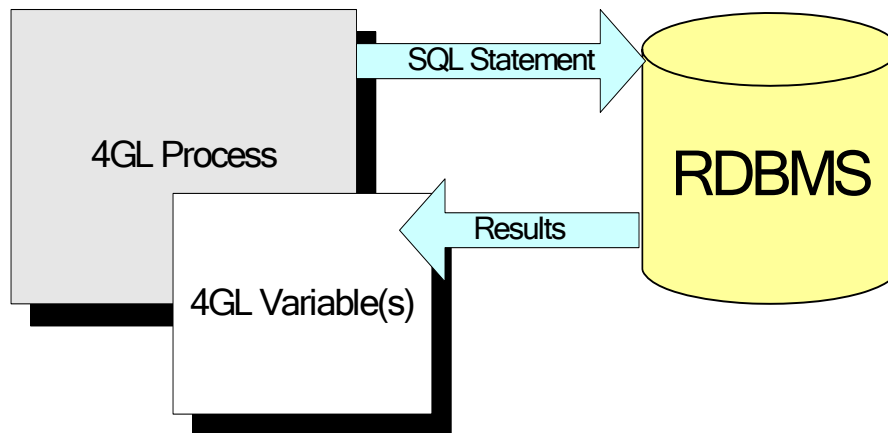
In this example, both the literal value '1' and the 4GL variable 'my_int' are considered bind variables for the SQL statement.

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:

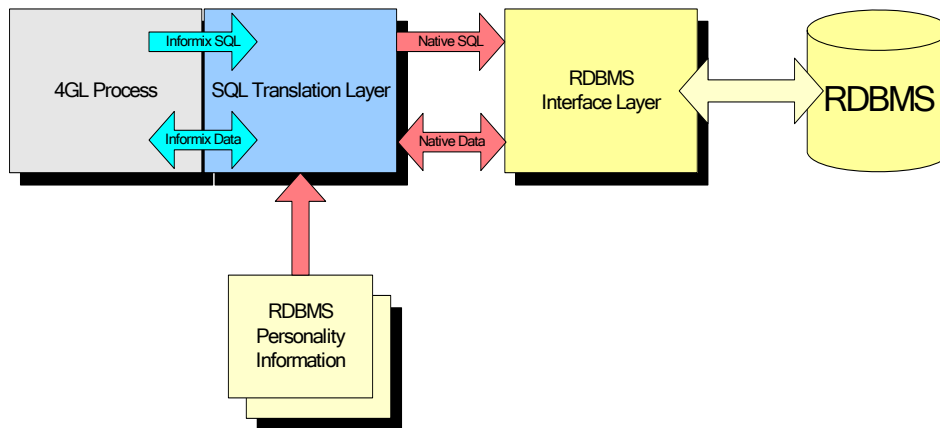
```
DEFINE rec1 RECORD LIKE x1.*  
  
DECLARE c1 CURSOR FOR  
  SELECT *  
  FROM x1  
  
FOREACH c1 INTO rec1.*  
  ...
```

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.



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.

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

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 MySQL 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	MySQL Datatype	Notes
CHAR	CHAR	CHAR	
VARCHAR	VARCHAR	VARCHAR	
LVARCHAR	VARCHAR	VARCHAR	
NCHAR	NCHAR	NATIONAL CHAR	
NVARCHAR	NVARCHAR	NATIONAL VARCHAR	
SMALLINT	SMALLINT	SMALLINT	
INTEGER	INTEGER	INTEGER	
INTEGER8	INTEGER8	BIGINT	
SERIAL	INTEGER	INTEGER	Column is automatically assigned as PRIMARY KEY.
SERIAL8	INTEGER8	BIGINT	Column is automatically assigned as PRIMARY KEY.
FLOAT	FLOAT	DOUBLE	
SMALLFLOAT	SMALLFLOAT	FLOAT	
DOUBLE PRECISION	FLOAT	DOUBLE	
DECIMAL	DECIMAL	DECIMAL	
MONEY	MONEY	DECIMAL	

Informix SQL Datatype	Informix 4GL Datatype	MySQL Datatype	Notes
DATE	DATE	DATE	
DATETIME	DATETIME	DATETIME	MySQL DATETIME types do not support fractional precision.
INTERVAL	INTERVAL	FLOAT	Emulated value
BYTE	BYTE	BLOB	
TEXT	TEXT	TEXT	

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

MySQL	Informix 4GL	Informix SQL	Notes
BIT	CHAR	CHAR	
TINYINT	CHAR(1)	CHAR(1)	
BOOL	SMALLINT	SMALLINT	
MEDIUMINT	INTEGER	INTEGER	
INTEGER	INTEGER	INTEGER	
BIGINT	BIGINT	BIGINT	
FLOAT	SMALLFLOAT	SMALLFLOAT	
DOUBLE	FLOAT	FLOAT	
DECIMAL	DECIMAL	DECIMAL	
DATE	DATE	DATE	
DATETIME	DATETIME	DATETIME	
TIMESTAMP	DATETIME	DATETIME	
TIME	DATETIME	DATETIME	
YEAR	SMALLINT	SMALLINT	
CHAR	CHAR	CHAR	
NATIONAL CHAR	NCHAR	NCHAR	

MySQL	Informix 4GL	Informix SQL	Notes
VARCHAR	VARCHAR	VARCHAR	
NATIONAL VARCHAR	NVARCHAR	NVARCHAR	
BINARY	CHAR	CHAR	
VARBINARY	VARCHAR	VARCHAR	
TINYTEXT	TEXT	TEXT	
TEXT	TEXT	TEXT	
MEDIUMTEXT	TEXT	TEXT	
LONGTEXT	TEXT	TEXT	
TINYBLOB	BYTE	BYTE	
BLOB	BYTE	BYTE	
MEDIUMBLOB	BYTE	BYTE	
LOB	BYTE	BYTE	

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 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 fg1c), 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	8	ODBC_DSN	7
AnsiNPW	8	Preparing	23
APP	8	Prerequisites	4
Authentication	5	Problems Fully Handled.....	30
AutoTranslate	9	Problems Handled Through Emulation.....	29
Bind Variables	34	Product Range	5
Buffer Variables	36	PWD.....	9
Common Conversion	47	QuotedID.....	9
compared	5	qx__\$schema table	38
Comparison.....	5	qxexport	24
Compiling 4GL code	24	QXORA_DB_IS_DSN	7
Configuring Options	41	QXORA_EMULATE_ROWID.....	7
Connecting.....	7, 24	Read Committed (ANSI).....	6
Connecting to SQL Server from Windows.....	10	Read Uncommitted (ANSI).....	6
Conversion.....	23	Regional	9
Cursor Names	48	Repeatable Read.....	6
Cursor Stability	6	SAVEFILE.....	9
DATABASE.....	9	schema object.....	23
Database Drivers.....	34	SERIAL8.....	43
Datatype Mappings	38, 43	Serializable (ANSI).....	6
DATETIME.....	44	SERVER	9
Dirty Read	6	SQL Compliance.....	5
DSN	9	SQL Error Codes.....	31
Dynamic SQL Translator.....	33	SQL Generation	37
Environment Settings.....	7	SQL interpretation.....	37
FILEDSN.....	9	SSTRACE	7
Informix Compatibility	25	StatsLog	9
Informix datatypes	23	StatsLogFile	9
INTEGER8.....	43	Storage	5
INTERVAL	44	Stored Procedures	48
Isolation Levels.....	5, 6, 48	Supported Connection Interfaces	5
LANGUAGE	9	System Catalogs.....	47
Loading table data	24	Translation	37
Matches.....	47	Type Promotion.....	38
Migration of the Database Schema	23	UID	9
MONEY	43	Unhandled Problems	27
NCHAR	43	Unix.....	22
NVARCHAR.....	43	WSID	9
ODBC Connect Option.....	8		