NATIONAL INSTRUMENTS™
LabVIEW™

Database Connectivity Toolset User Manual

NATIONAL INSTRUMENTS™

May 2001 Edition
Part Number 321525C-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ottawa) 613 233 5949, Canada (Québec) 514 694 8521,
Canada (Toronto) 905 785 0085, China (Shanghai) 021 6555 7838, China (ShenZhen) 0755 3904939,
Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24, Germany 089 741 31 30,
Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Malaysia 603 9596711, Mexico 5 280 7625, Netherlands 0348 433466,
New Zealand 09 914 0488, Norway 32 27 73 00, Poland 0 22 528 94 06, Portugal 351 1 726 9011,
Singapore 2265886, Spain 91 640 0085, Sweden 08 587 895 00, Switzerland 056 200 51 51,
Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

Conventions ix
Related Documentation x

Chapter 1
Introduction

Overview 1-1
Installing the LabVIEW Database Connectivity Toolset 1-2
Upgrading from Previous Versions 1-3
The Connection Reference Data Type Has Changed 1-5

Chapter 2
Getting Started with the Database Connectivity Toolset

Database Concepts 2-1
Background of the Database Connectivity Toolset 2-3
ODBC Standard 2-3
Structured Query Language (SQL) 2-4
SQL Dialects 2-6
Universal Data Access, OLE DB, and ADO 2-6
OLE DB Standard 2-7
OLE DB Provider for ODBC 2-8
OLE DB Provider for SQL Server 2-9
OLE DB Provider for Jet 2-10
OLE DB Provider for Oracle 2-11
ActiveX Data Objects (ADO) 2-12

Chapter 3
Using the Database Connectivity Toolset

Connecting to a Database 3-1
DSNs and Data Source Types 3-1
ODBC Administrator 3-2
Examples of Using DSNs 3-4
UDLs 3-7
Configuring a UDL 3-9
Example Using a UDL 3-11
High-Level Database VIs 3-11
Writing Data to a Database 3-12

Reading Data from a Database ... 3-13
Reading Specific Data from a Table ... 3-16
Creating and Deleting Tables ... 3-17
Supported Data Types ... 3-18
Working with Date/Time ... 3-19
Handling NULL Values ... 3-20
Currency and Boolean Data Types ... 3-22
Using the Database Connectivity Toolset Examples ... 3-22
Using the Examples with Other Databases ... 3-22
Using the Examples without a Database ... 3-23

Chapter 4
Database Connectivity Toolset Utilities

Getting Table and Column Information ... 4-1
Getting and Setting Database Properties ... 4-2
ADO Reference Classes ... 4-3
Specific Properties ... 4-4
Formatting Date and Time ... 4-4
Performing Database Transactions ... 4-5
Locking Transactions and Setting Isolation Levels ... 4-6
Writing and Reading Data Files ... 4-8

Chapter 5
Advanced Database Operations

Executing SQL Statements and Fetching Data ... 5-1
Navigating through Database Records ... 5-3
Using Cursors ... 5-3
Cursor Types ... 5-4
Moving Through Recordsets ... 5-6
Using Parameterized Statements ... 5-8
Using Stored Procedures ... 5-10
Creating Stored Procedures ... 5-11
Running Stored Procedures without Parameters ... 5-12
Running Stored Procedures with Parameters ... 5-13

Chapter 6
Building Applications

Using the Database Connectivity Toolset Build Script 6-1
Installing MDAC 6-2
MDAC 2.6 6-4
Using Non-English Versions of Windows 6-4
Using Data Links and DSNs 6-5

Appendix A
SQL Quick-Reference

Appendix B
References

Appendix C
Supported Data Types

Appendix D
Technical Support Resources

Glossary

Index

About This Manual

The *LabVIEW Database Connectivity Toolset User Manual* describes the virtual instruments (VIs) used to communicate and pass data between LabVIEW and either a local or a remote database management system (DBMS). You should be familiar with the operation of LabVIEW, your computer, and your computer operating system.

Conventions

The following conventions appear in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- *LabVIEW Database Connectivity Toolset Help*, available by selecting **Help»Database Toolset VIs and Examples**
- *LabVIEW Help*, available by selecting **Help»Contents and Index**

1

Introduction

This chapter describes the installation procedure, installed components, and the main features of the LabVIEW Database Connectivity Toolset.

Overview

The LabVIEW Database Connectivity Toolset is an add-on package for accessing databases. The toolkit contains a set of high-level functions for performing the most common database tasks and advanced functions for customized tasks.

The following list describes the main features of the LabVIEW Database Connectivity Toolset:

- Works with any provider that adheres to the Microsoft ActiveX Data Object (ADO) standard.
- Works with any database driver that complies with ODBC or OLE DB.
- Maintains a high level of portability. In many cases, you can port your application to another database by changing the connection string you pass to the DB Tools Open Connection VI.
- Converts database column values from native data types to standard LabVIEW Database Connectivity Toolset data types, further enhancing portability.
- The default ADO ODBC provider permits the use of SQL statements with all supported database systems, even non-SQL systems.
- Includes VIs to retrieve the name and data type of a column returned by a SELECT statement.
- Creates tables and selects, inserts, updates, and deletes records without using SQL statements.

Because of the wide range of databases the LabVIEW Database Connectivity Toolset works with, some portability issues remain. Consider the following issues when choosing your database system:

- Some database systems, particularly the flat-file databases such as dBase, do not support floating–point numbers. In cases where floating–point numbers are not supported, the toolkit converts floating–point numbers to the nearest equivalent, usually binary coded decimal (BCD), before storing them in the database. Very large or very small floating–point numbers can easily pass the upper or lower limits of the precision available for a BCD value.
- The Microsoft ODBC driver for Oracle and the Microsoft OLE DB Provider for Oracle do not support BLOB (binary) data types. You can not use Oracle with the Database Connectivity Toolset for binary data with these drivers. Instead, use the OLE DB Provider (and ODBC driver) provided by Oracle from their web site at: `http://technet/oracle.com/tech/nt/ole_db/`
- Restrictions on column names vary among database systems. For maximum portability, limit column names to ten uppercase characters without spaces. You might be able access longer column (field) names or names that contain spaces by enclosing the name in double quotes.
- Some database systems do not support date, time, or date and time data types.

Installing the LabVIEW Database Connectivity Toolset

Before you install the LabVIEW Database Connectivity Toolset, uninstall previous versions of the SQL Toolkit.

The LabVIEW Database Connectivity Toolset is shipped on a CD. If you need to install it on a system without a CD drive, either mount it as a volume over a network or create a set of floppies. Install the toolset by inserting the LabVIEW Database Connectivity Toolset CD into the CD drive and running `setup.exe` using one of the following methods:

- Select **Start»Run** from the Windows taskbar. In the dialog box that appears, enter *x*`:\setup\database.exe`, where *x* denotes the proper letter designation of the drive. Click the **OK** button and follow the instructions that appear on your screen.
- Launch Windows Explorer. Click the icon of the drive that contains the installation disk. Double-click `setup.exe` in the list of files on the installation disk.

When the installation dialog box appears on the screen, you can change the default directories for the toolkit. The LabVIEW Database Connectivity Toolset installation program installs the following components:

- Database Connectivity Toolset—Installs the Database Connectivity Toolset components and examples.
- SQL Compatibility VIs—Installs a set of VI libraries compatible with the SQL Toolkit. Install this component only if you have existing applications that use the SQL Toolkit.
- MDAC—Installs the Microsoft Data Access Components version 2.5. This includes ActiveX Data Objects (ADO) software and drivers, ODBC drivers, and OLE DB drivers.

Note If you are using a non-English version of Windows, do no install the MDAC component. The typical installation of the Database Connectivity Toolset installs the English version of MDAC. You should do a custom install and not include the MDAC component. You get the localized version of MDAC and instructions for your OS from the Microsoft web site at `http://www.microsoft.com/data/`

In addition, the Database Connectivity Toolset installer modifies or creates an entry in the registry under the `HKEY_LOCAL_MACHINE»National Instruments»LabVIEW»addons` key.

Upgrading from Previous Versions

This section provides an overview of how to upgrade from previous versions of the SQL Toolkit.

The new LabVIEW Database Connectivity Toolset uses Microsoft ADO (ActiveX Data Objects) technology for Universal Data Access. ADO is a thin ActiveX wrapper to OLE DB, an updated version of the ODBC standard. Because ADO is an ActiveX interface, you can automate it from LabVIEW using the Invoke and Property Nodes. Therefore, you can use the LabVIEW Database Connectivity Toolset to communicate between LabVIEW and any database that has an OLE DB or ODBC driver.

A library of compatibility VIs is included with the Database Connectivity Toolset so users of the SQL Toolkit can convert their applications to the ADO technology. These compatibility VIs install into the `LabVIEW\vi.lib\addons\_SQL` directory in the same directory and VI library structure used by the SQL Toolkit. VIs with the same names and connectors are included for most of the SQL Toolkit. However, ADO

does not provide all the functionality of the SQL Toolkit. The following VIs were unable to be reproduced in the compatibility VIs using ADO:

- Get Column Information
- Get Database Information
- Get DSN Information
- Get Procedure Col Information
- Get References
- Get Table Information
- Get Type Information
- Get Column Alias
- Get Column Attributes
- Get Column Expression
- Get Column Width
- Request Column Information
- Request Database Information
- Request DSN Information
- Request Procedure Col Information
- Request Table Information
- Request Type Information
- Clear SQL Parameter
- Close Fetch Log File
- Get Statement Options
- Set Statement Options
- Get Supported Isolation Levels

Although these VIs do not exist in the SQL compatibility VIs, some of the new Database Connectivity Toolset VIs can perform similar operations.

Note As with any compatibility VI provided for LabVIEW, you should try to convert applications that use them to use the new technology. Subsequent releases of the Database Connectivity Toolset might not contain the SQL compatibility VIs.

The Connection Reference Data Type Has Changed

The connection reference data type for the SQL Toolkit for G is a word (I16) integer and the connection reference for the Database Connectivity Toolset is an ActiveX ADO refnum. The entire set of SQL compatibility VIs now uses the new refnum. You are told to uninstall the SQL Toolkit before installing the new Database Connectivity Toolset because mixing the old and new toolkit VIs results in broken connection reference wires because of this change in data types.

If you have existing subVIs you built that use the old I16 connection references as inputs/outputs, these wires will be broken when you use the new SQL compatibility VIs. The SQL compatibility VIs do not include a converter between the two reference types because the old references are 16-bit integers and the new ActiveX refnums are 32-bit integers. A converter would be type casting the 32-bit integers to 16-bit integers to pass the references between subVIs. The converter loses half the information of the original refnum and while all subVIs would wire to each other correctly, the ActiveX refnums would be corrupt and result in runtime errors. Therefore, the connection references are left as incompatible data types and cause compiler errors (broken wires) to better alert users to the conflict. You can right-click on the refnum inputs for the SQL compatibility VIs and select **Create»Control/Indicator** to get the correct refnum types for your VIs.

2

Getting Started with the Database Connectivity Toolset

This chapter introduces the basic concepts of database interactions using the LabVIEW Database Connectivity Toolset. It also describes the Structured Query Language (SQL), the Open Database Connectivity (ODBC) standard, and ActiveX Data Objects (ADO).

The LabVIEW Database Connectivity Toolset accesses several popular file-based databases and high-performance Relational Database Management system (RDBMS) software packages that run on a variety of computers and operating systems. Through specific application programs, users can log on to, create, store, modify, retrieve, sort, and manage data in local or networked databases. Commands expressed in the Structured Query Language (SQL) perform these functions.

Database Concepts

A database consists of an organized collection of data. Most modern Database Management Systems (DBMS) store data in tables. The tables are organized into records, also known as rows, and fields, also known as columns. Every table in a database must have a unique name. Similarly, every field within a table must have a unique name.

The database tables have many uses. Table 2-1 is an example table that you could use with a simple test executive program to record test sequence results. It contains columns for the unit under test number, the test name, the test result, and two measurements. The data in the table is not inherently ordered. Ordering, grouping, and other manipulations of the data occur when you use a SELECT statement to retrieve the data from the table. A row can have empty columns, which means that the row contains NULL values. NULL values for databases are not exactly the same as NULL values in the C programming language. Refer to Chapter 3, *Handling NULL Values*, for more information about how LabVIEW handles NULL values.

Note The NULL values in a table row are not the same as NULL values in the C programming language. This manual refers to NULL values in tables as SQL NULL values, to distinguish them from standard NULL values.

Table 2-1. Sample Test Sequence Results

UUT_NUM	TEST_NAME	RESULT	MEAS1	MEAS2
20860B456	TEST1	PASS	0.5	0.6
20860B456	TEST2	PASS	1.2	—
20860B123	TEST1	FAIL	–0.1	0.7
20860B789	TEST1	PASS	0.6	0.6
20860B789	TEST2	PASS	1.3	—

Non-relational databases allow you to store all the information in one large structure like Table 2-1. This method is sometimes inefficient, because all of the information is in one table, and searching for a specific piece of data can be difficult and time-consuming. Relational databases have information stored in multiple structures, or tables, where each table can be smaller and contain a specific subset of information. Figure 2-1 shows two database tables that are related to each other through the lot field.

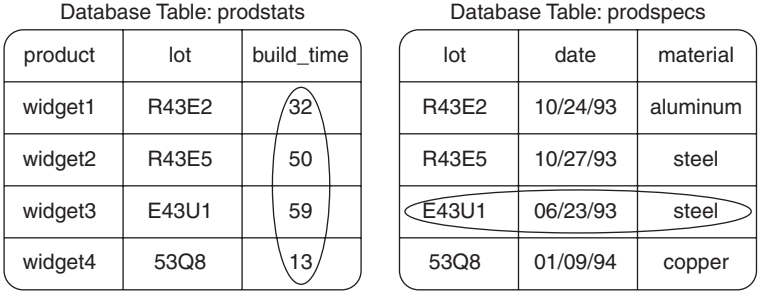

Figure 2-1. Relational Database Table Concept

Each column in a table has a data type, such as CHARACTER (fixed and variable length), NUMBER, DECIMAL, INTEGER, FLOAT, REAL, DOUBLE PRECISION, DATE, LONG, and RAW. The available data types vary depending on the DBMS. The LabVIEW Database Connectivity Toolset uses a set of common data types. The toolset automatically maps these data types into the appropriate type in the underlying database. By using the common data types, the toolkit program can access a variety of databases with little or no modification. Refer to Chapter 3, *Using the Database Connectivity Toolset*, for more information about these supported data types.

Background of the Database Connectivity Toolset

The LabVIEW Database Connectivity Toolset is a re-architecture of the SQL Toolkit. The SQL Toolkit is built from a code interface node (CIN) that calls a series of Dynamic Link Libraries (DLLs) provided by a third party vendor known as INTERSOLV (now called MERANT). These DLLs made system calls into Microsoft's application programming interface (API) for database access called ODBC.

ODBC Standard

The SQL Access Group, including representatives of Microsoft, Tandem, Oracle, Informix, and Digital Equipment Corporations, developed the Open Database Connectivity (ODBC) standard as a uniform method for applications to access databases. ODBC 1.0 released in September 1992. The standard consists of a multilevel API definition, a driver packaging standard, an SQL implementation based on ANSI SQL, and a means for defining and maintaining Data Source Names (DSN). A DSN is a quick way to refer to a specific database. You specify a DSN with a unique name and by the ODBC driver that communicates with the physical database, local or remote. You must define a DSN for each database to which an application program connects.

The SQL Toolkit currently supports ODBC 2.0. Because the toolkit and the drivers that come with it comply with the ODBC standard, you can port LabVIEW database applications to other supported databases with minimal

changes. The SQL Toolkit VIs call the Microsoft API for ODBC. ODBC then communicates with a database's specific driver that translates the call to the database's low level language, as shown in Figure 2-2.

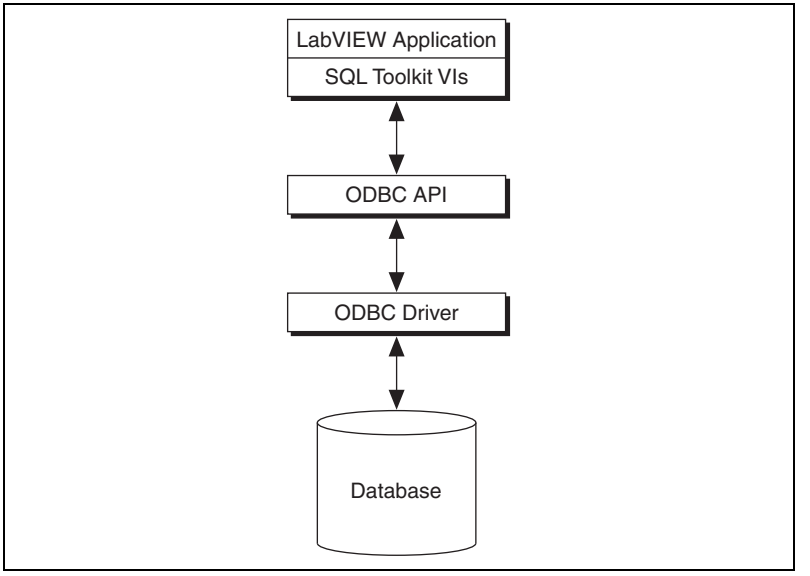

Figure 2-2. Communication Path between LabVIEW and a Database Using the SQL Toolkit for G

The SQL Toolkit is compatible with any database providing an ODBC driver that translates the ODBC calls to the native database language. You need to know SQL to use the SQL Toolkit.

Structured Query Language (SQL)

The Structured Query Language consists of a set of character string commands and is a widely supported standard for database access. You can use the SQL commands to describe, store, retrieve, and manipulate the rows and columns in database tables. IBM developed the language, and it became publicly available in the late-1970s. Since then, the American National Standards Institute (ANSI), the International Standards Organization (ISO), and the Federal Information Processing Standards (FIPS) have adopted SQL and most major commercial relational database

products support it to some degree. It is a non-procedural language for processing sets of records in database tables. The following are three pertinent classes of SQL statements:

- Data Definition/Control Language (DDL/DCL) statements define and control the structure of the database. They also define and grant access privileges to database users. Use the statements to create, define, and alter databases and tables.
- Data Manipulation Language (DML) statements operate on the data contents of database tables. You use these statements to insert rows of data into a table, update rows of data in a table, delete rows from a table, and conduct database transactions.
- Queries are SQL SELECT statements that specify which tables and rows are retrieved from the database.

Table 2-2 describes some frequently used SQL commands.

Table 2-2. Common SQL Commands

Command	Function
CREATE TABLE	Creates a database table and specifies the name and data type for each column therein. The result is a named table in the database. CREATE TABLE is a DDL command. DROP TABLE is the complementary DDL command.
INSERT	Adds a new data row to the table, allowing values to be specified for each column. INSERT is a DML command.
SELECT	Initiates a search for all rows in a table whose column data satisfy specified combinations of conditions. The result is an active set of rows that satisfy the search conditions. SELECT is a query command.
UPDATE	Initiates a search as in SELECT, then changes the contents of specific column data in each row in the resulting active set. UPDATE is a DML command.
DELETE	Initiates a search as in SELECT, then removes the resulting active set from the table. DELETE is a DML command.

A typical SQL statement might read as follows:

```
SELECT product, lot FROM prodstats WHERE lot=R43E2
```

The interpretation of this statement is to retrieve the columns product and lot from the prodstats table, including only those rows where the value contained in the lot column equals R43E2. This query creates a one-row, two-column active set as shown in Table 2-3.

Table 2-3. Example Query Results

widget	R43E2

SQL Dialects

Several database publishers use their own SQL dialects, in their products. These dialects usually consist of extensions to the standard SQL commands that perform higher-level or database-specific functions. Conversely, certain accessible databases do not directly support standard SQL functions. Differences are most noticeable when using the CREATE TABLE and ALTER TABLE commands. Most of the databases use their own particular column type keywords. Many of the databases have different syntax for date-and-time formats. ODBC-compliant SQL Toolkit software helps to minimize the effects of these SQL variants. Refer to Appendix A, *SQL Quick-Reference*, for information about particular databases.

Universal Data Access, OLE DB, and ADO

The ODBC standards design was to access only relational databases. Microsoft realized this as a limitation and developed a platform called Universal Data Access (UDA) where applications can exchange relational or non-relational data across intranets or the Internet, essentially connecting any type of data with any type of application. OLE DB is the Microsoft system-level programming interface to diverse sources of data. ADO is the application-level programming interface.

The Microsoft Data Access Components (MDAC) are the practical implementation of Microsoft's UDA strategy. The LabVIEW Database Connectivity Toolset includes MDAC 2.5 as part of its installation. MDAC 2.5 includes the ODBC, OLE DB, and ADO components. MDAC also installs several data providers you can use to open a connection to a specific data source such as an MS Access database.

OLE DB Standard

OLE DB specifies a set of Microsoft Component Object Model (COM) interfaces that support various database management system services. These interfaces enable you to create software components that comprise the UDA platform. OLE DB is a C++ API that allows for lower-level database access from a C++ compiler. The following are three general types of COM components for OLE DB:

- OLE DB Data Providers—Data source-specific software layers that are responsible for accessing and exposing data.
- OLE DB Consumers—Data-centric applications, components, or tools that use data through the OLE DB interfaces. Using networking terms, OLE DB consumers are the clients, and the OLE DB data provider is the server.
- OLE DB Service Providers—Optional components that implement standard services to extend the functionality of data providers. Examples of these services include cursor engines, query processors, and data conversion engines.

As mentioned previously, the LabVIEW Database Connectivity Toolset installs MDAC 2.5 if it is not already installed on your machine. Windows 2000 and Windows ME contain MDAC as part of the operating system. MDAC includes several OLE DB providers for various data sources. All data access in the LabVIEW Database Connectivity Toolset occurs through an OLE DB provider. If you do not specify a provider, the toolkit automatically uses the default ODBC provider described in the following section. Microsoft provides some relational data providers as part of the MDAC installation.

OLE DB Provider for ODBC

The OLE DB provider for ODBC was developed and released with MDAC 1.0 in 1996. Because ODBC is the best way to communicate between an application and any relational database, OLE DB provider for ODBC acts as a conversion layer between OLE DB interfaces and ODBC. The hierarchy of data interface layers between ADO and a database using the OLE DB provider for ODBC appears in Figure 2-3.

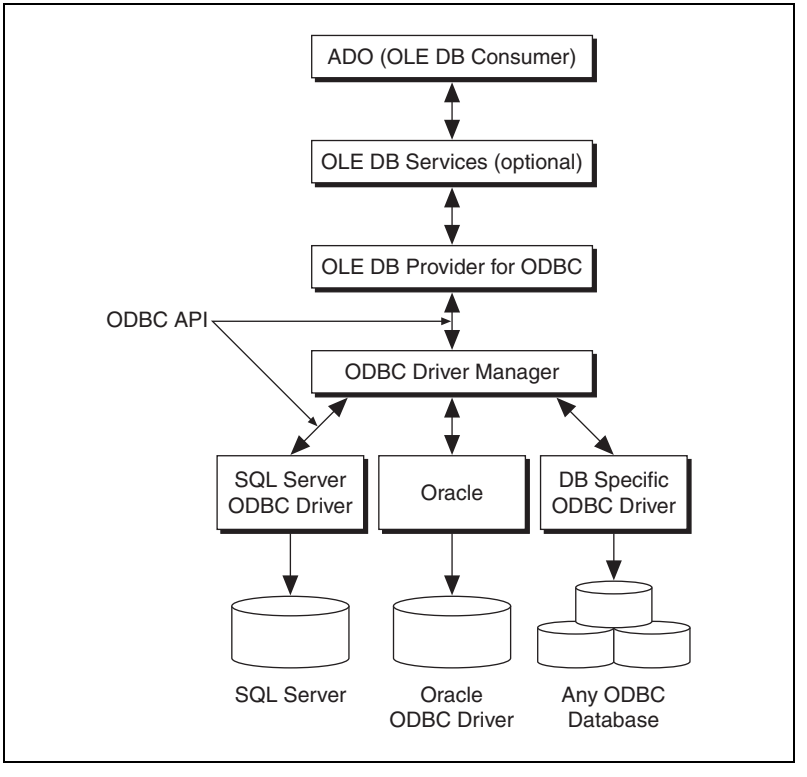

Figure 2-3. Communication Path between ADO and a Database Using the OLE DB Provider for ODBC

With MDAC 2.0, released in 1998, Microsoft provided an OLE DB provider for SQL Server, Jet, and Oracle database systems. Native providers are much faster than using the OLE DB Provider for ODBC, because they eliminate the need for both the OLE DB to ODBC conversion process and for the ODBC driver and Driver Manager layers. For this reason, you should always use the native OLE DB data provider for the data source you are accessing if it is available.

OLE DB Provider for SQL Server

The OLE DB provider for SQL Server, shown in Figure 2-4, exposes data stored in Microsoft SQL Server 6.5 or later databases.

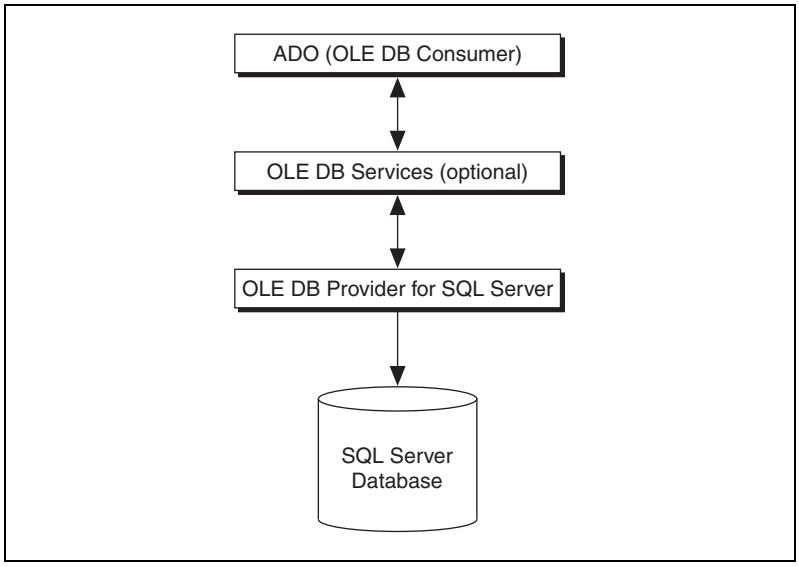

Figure 2-4. Communication Path between ADO and an SQL Server Database Using the Native OLE DB Provider

OLE DB Provider for Jet

The OLE DB Provider for Jet uses the Microsoft Jet database engine to expose data stored in Microsoft Access databases (`.mdb`) and numerous Indexed Sequential Access Method (ISAM) databases, including Paradox, dBase, Btrieve, Excel, and FoxPro. The Jet database engine is included with Microsoft Access and is the underlying DBMS of Microsoft Access. Visual Basic for Applications is the host language for the Jet DBMS.

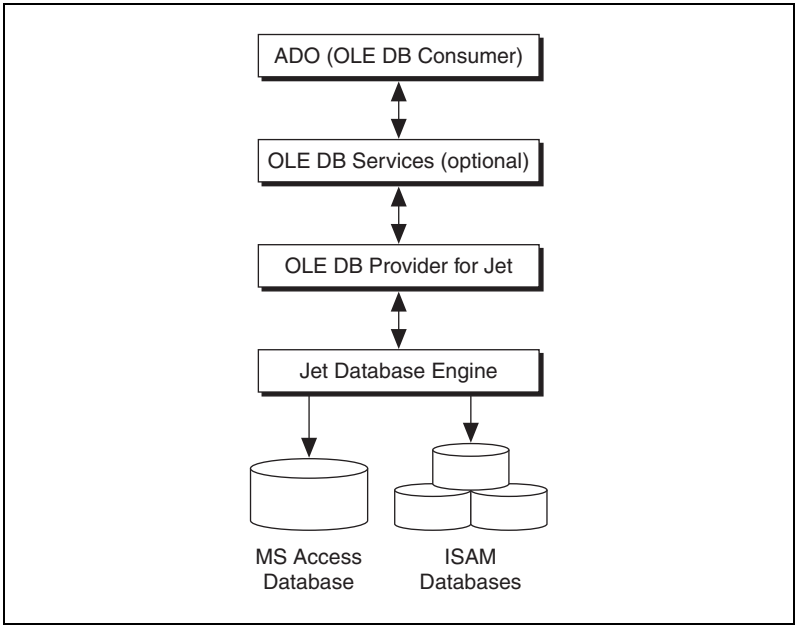

Figure 2-5. Communication Path between ADO and an Access Database Using the Native OLE DB Provider

Jet 1.1 shipped with an interface for using the database engine programmatically. The Jet interface was Data Access Objects (DAO) 1.1. DAO is a COM component that gives custom applications the power and flexibility of the Jet database engine in a simple object model. DAO is also language-independent, meaning that any programming language or toolset that supported OLE Automation can use DAO and the Jet database engine.

Visual Basic 3.0 also shipped with the Jet data control, which allowed applications to be bound quickly to databases through the Jet database engine. Today the Jet database engine supports many more features and DAO exposes all Jet functionality. DAO has become one of the most widely used and accepted data access technologies by all types of developers. This is partly due to the fact that DAO builds on the success of ODBC.

Despite the availability of the OLE DB Provider for Jet and comparable benchmarks, many developers still use DAO because some of the functionality of DAO, such as data definition and security, is still not available in the OLE DB Provider for Jet.

Note Although DAO and ADO are both APIs for communicating with and manipulating data in databases, they are separate and different. DAO is specifically used with the Jet database engine, but ADO is part of Microsoft's UDA strategy for sharing data between any applications and over the Internet. Refer to the *ActiveX Data Objects (ADO)* section later in this chapter for more information about ADO.

OLE DB Provider for Oracle

The OLE DB provider for Oracle exposes OLE DB interfaces for retrieving and manipulating data stored in Oracle 7.3.3 or later databases. The OLE DB provider for Oracle is implemented as a layer on top of the Oracle Native API, the Oracle Call Interface (OCI). Oracle also provides an OLE DB provider that is slightly newer than the Microsoft version. If you experience any problems, use Oracle's provider available on their web site at `http://technet.oracle.com/software/tech/ nt/ole_db/`

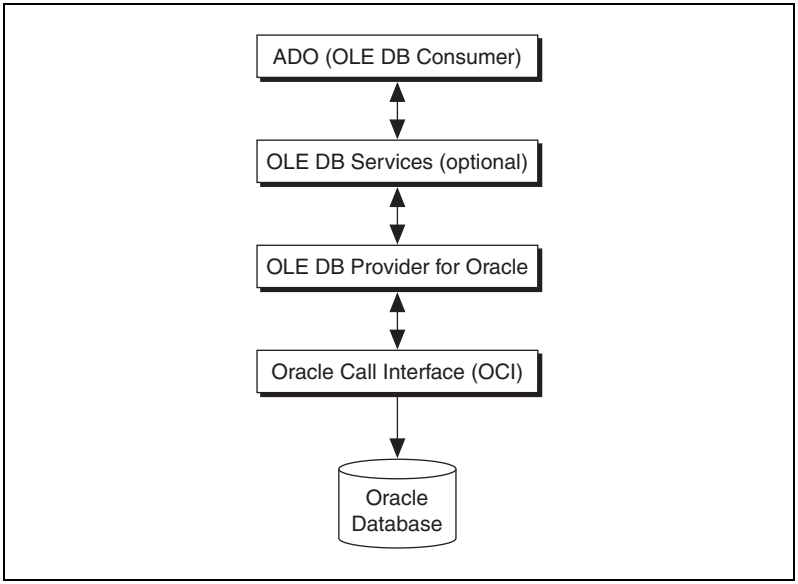

Figure 2-6. Communication Path between ADO and an Oracle Database Using the Native OLE DB Provider

Microsoft also provides a number of OLE DB data providers for non-relational data sources including the following:

- OLE DB provider for AS/400
- OLE DB provider for Index Server
- OLE DB provider for Internet Publishing
- OLE DB provider for Active Directory
- OLE DB provider for Microsoft Exchange
- OLE DB provider for OLAP (Online Analytical Processing)

Two third party vendors, Merant, `http://www.merant.com`, and ISG Navigator, `http://www.isgnavigator.com`, also supply OLE DB providers.

If you need access to a data source that does not provide an OLE DB data provider and does not support ODBC, you can create custom OLE DB data providers that can expose any data source. For example, you can develop custom OLE DB data providers for data sources such as the following:

- Personal address book
- Windows registry
- Scheduled tasks
- Shared memory

An advantage of UDA is the ability to develop custom OLE DB data providers because it enables standardized access to data sources beyond Microsoft products and the popular relational database systems. Refer to Appendix B, *References*, for more information about writing your own OLE DB providers.

ActiveX Data Objects (ADO)

As mentioned previously, OLE DB is a C++ system-level programming interface, and ADO is the application-level programming interface to diverse sources of data. ADO is an ActiveX wrapper to OLE DB so that any programming language or tool that supports COM can use the OLE DB technology through ADO. The LabVIEW Database Connectivity Toolset consists of ADO calls made through Invoke and Property Nodes.

You do not need to know the details about ADO or the other standards mentioned in this chapter in order to use the LabVIEW Database Connectivity Toolset. However, knowledge about the history and architecture of ADO helps you use these tools more efficiently. The object

model for ADO is made up of three main COM objects, Connection, Command, and Recordset, as shown in Figure 2-7. According to the ADO standard, each of these top-level objects can exist independently without the others. However, the Database Connectivity Toolset has a hierarchical structure where the Connection object is necessary in order to use a Command or Recordset object.

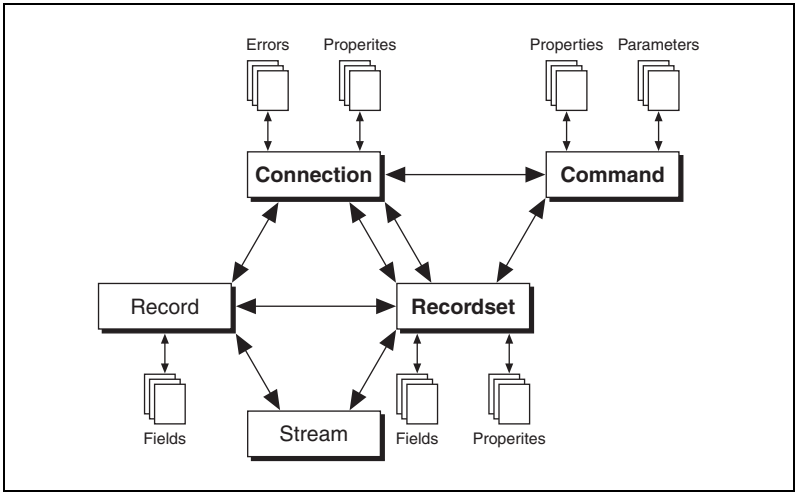

Figure 2-7. ADO Object Hierarchy

Table 2-4 describes each of the components of the ADO object model.

Table 2-4. ADO Object Model Components

Component	Description
Connection	This object represents an open connection to an OLE DB data source. It contains methods for setting timeouts and maintaining information about the connection.
Command	The two major uses for a command object are to execute statements against an OLE DB connection and to retrieve a recordset based on an SQL query or stored procedure.
Recordset	This object represents a set of records and is used to manipulate data in a data source. You also can control cursors and the locking types for recordsets.

Table 2-4. ADO Object Model Components (Continued)

Component	Description
Record	This object represents a single row in a recordset. The Record, Stream, and Recordset objects work together to help you navigate through data.
Stream	This object represents binary data, usually stored in Unicode.
Property	This object is the building block of the other ADO objects. The properties collection contains only the properties added to the object by the data provider and does not contain the intrinsic properties of the object.
Error	This object represents a single error.
Parameter	This object represents a single parameter for a Command object. Generally, parameters are used with any type of parameterized commands where an action is defined once but can have results changed depending on the variable values.
Field	This object represents a single column of data in a recordset. The fields collection is the default property of the Recordset object, so you do not often see its name in the code.

The LabVIEW Database Connectivity Toolset is based upon version 2.5 of ADO. This version of ADO, part of the MDAC 2.5 package, automatically installs with the toolkit on platforms where an older version or no version of MDAC exists.

Note ADO is a Windows-only technology. However, you can run the database servers on other platforms as long as an OLE DB provider or an ODBC driver exists for the database. For example, if you use an Oracle database on a UNIX machine connected to a network, you can use the LabVIEW Database Connectivity Toolset VIs on a Windows machine on that network to access the database through the OLE DB provider for Oracle.

3

Using the Database Connectivity Toolset

This chapter describes how to use the Database Connectivity Toolset to communicate with databases. The first section describes the different methods of connecting to a database. The next section describes the high-level VIs you can use to create tables, delete tables, insert data into tables, or select data from tables in a database. Next is a description of the data types supported by the Database Connectivity Toolset. The last section describes how you can use the Database Connectivity Toolset examples, even when you do not have a database installed.

Connecting to a Database

Before you can access data in a table or execute SQL statements, you must establish a connection to a database. The LabVIEW Database Connectivity Toolset supports multiple simultaneous connections to a single database or to multiple databases. Use the DB Tools Open Connection VI to establish the connection to a database.

Connecting to a database is where most errors occur because each database management system (DBMS) uses different parameters for the connection and different levels of security. The different standards use different methods of connecting to databases. For example, ODBC uses Data Source Name (DSN) for the connection and ADO uses Universal Data Links (UDL) for the connection. The DB Tools Open Connection VI supports all the various methods for connecting to a database as described in this section.

DSNs and Data Source Types

A DSN is not only the name of the data source, or database, you are connecting to, it also contains information about the ODBC driver and other connection attributes including paths, security information, and read-only status of the database. Two main types of DSNs exist, machine data sources and file data sources. Machine data sources are in the system registry and apply to all users of the computer system or to a single user,

system DSNs and user DSNs respectively. A file DSN is a text file with the extension `*.dsn` and is accessible to anyone with access to that file. File DSNs are not restricted to a single user or computer system. All DSNs are created and configured using the ODBC Administrator.

ODBC Administrator

You use the ODBC Administrator icon in your Control Panel to register and configure drivers to make them available as data sources for your applications. Your system saves all changes you make within the ODBC Administrator. When you double-click the ODBC Administrator icon in the Control Panel, **Administrative Tools»Data Sources (ODBC)** in Windows NT/2000, the ODBC Data Source Administrator dialog box appears as shown in Figure 3-1.

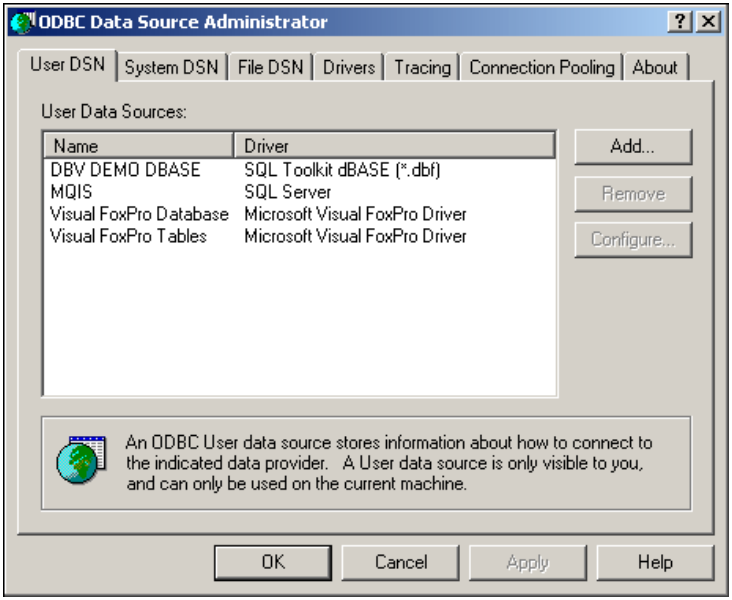

Figure 3-1. Data Sources Dialog Box

The ODBC Data Source Administrator dialog box lists all the registered ODBC data sources. You use the tabs to specify the type of DSN, User, System, or File. You then can use the **Add** or **Configure** buttons to display a driver-specific dialog box where you can configure a new or an existing

data source. The system then saves the configuration for the data source in the registry or to a file. When you create a new DSN, the Create New Data Source dialog box contains a list of all the ODBC drivers for your system, as shown in Figure 3-2.

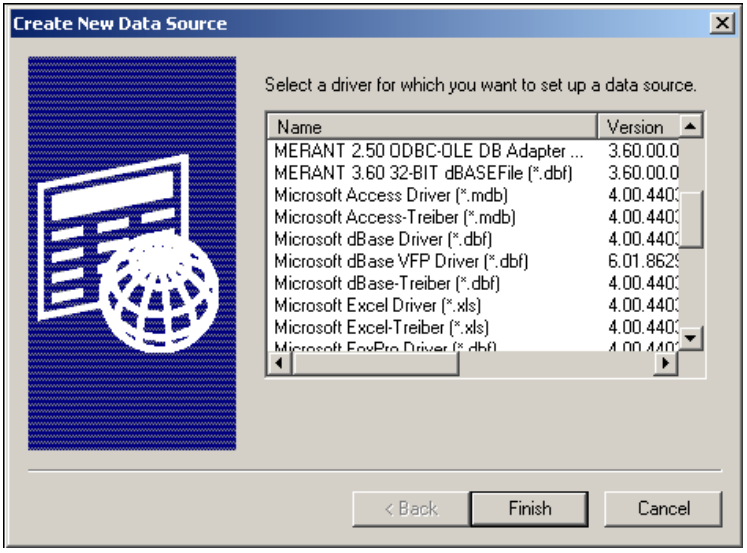

Figure 3-2. Available ODBC Drivers

The LabVIEW Database Connectivity Toolset does not provide custom ODBC drivers but installs MDAC. Microsoft provides several ODBC drivers with MDAC. Database system vendors and third-party developers also offer large selections of ODBC drivers. The LabVIEW Database Connectivity Toolset complies with the ODBC standard, so you can use it with any ODBC-compliant database drivers. Refer to your vendor documentation for information about registering your specific database drivers with the ODBC Administrator.

After you select a particular driver, a second dialog box appears that contains specific settings for that driver. ODBC drivers for databases such as SQL Server and Oracle contain settings and additional dialog boxes for configuring items such as server information, user identification, and

passwords. Figure 3-3 shows the ODBC Access Driver Setup dialog box for the system DSN named LabVIEW that is used for the Database Connectivity Toolset examples.

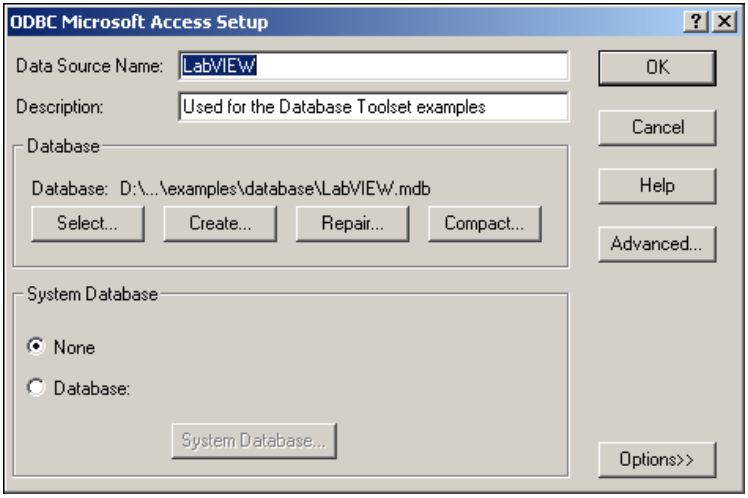

Figure 3-3. ODBC Access Driver Setup Dialog Box

Examples of Using DSNs

This section shows several examples of using the DB Tools Open Connection VI to connect to a various databases.

Figure 3-4 shows how you use a string to specify a System DSN, or a User DSN, called MS Access to open a connection to a specific database.

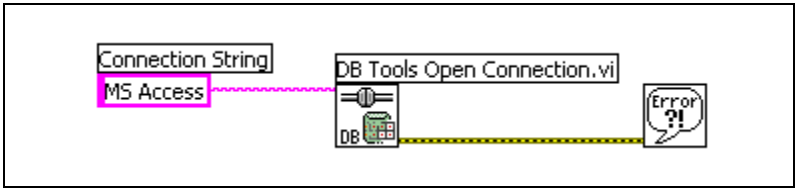

Figure 3-4. Connecting to an Access Database Using a System DSN

Figure 3-5 shows how you use a path to specify a File DSN named `access.dsn` to open a connection to the database. Notice that the connection information input is polymorphic and accepts either a string or path for the DSN.

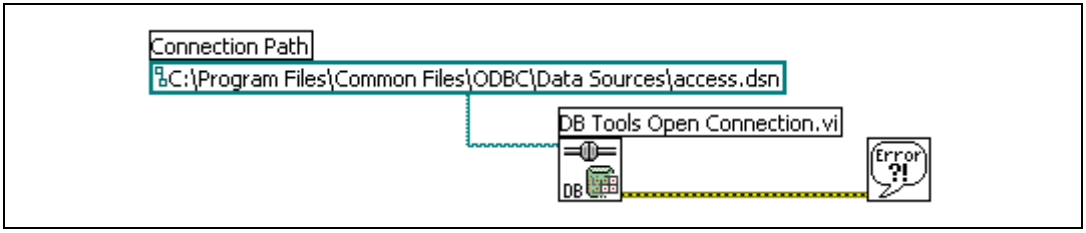

Figure 3-5. Connecting to an Access Database Using a File DSN

Figure 3-6 shows a connection made to an Oracle database using a System DSN. Notice that the user ID and password values are used. Some DBMS require these parameters to be set in order to connect to a database. You should be familiar with your DBMS and how to specify the connection parameters.

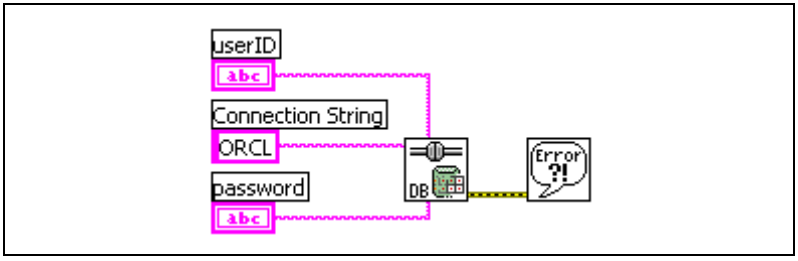

Figure 3-6. Connecting to an Oracle Database Using a System DSN

As shown in the previous examples, connecting to a database using the DB Tools Open Connection VI requires only a string or path value specifying the DSN along with optional user ID and password strings depending upon the DBMS. Therefore, the majority of problems in defining a connection occur when you create the DSN. Some ODBC drivers have an option to test the connection. Test the connection between the DSN and the database before you try to do anything with the Database Connectivity Toolset.

Figure 3-7 shows where you can test the connection using the Test Data Source button at the end of the process for creating a DSN for a SQL Server 7 database.

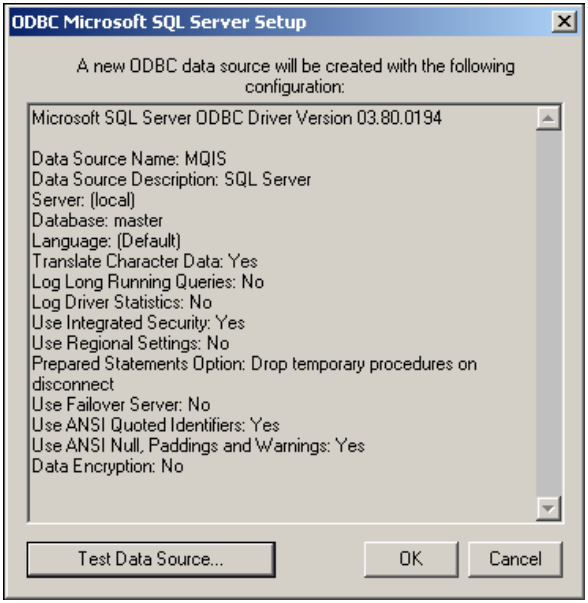

Figure 3-7. Testing a Database Connection using SQL Server

Creating a DSN is the way to connect to a database using ODBC. ADO and OLE DB use a method called a Universal Data Link (UDL).

UDLs

A UDL is similar to a DSN because it also describes more than just the data source. A UDL contains information about what OLE DB provider is used (the default is the Microsoft OLE DB provider for ODBC drivers), server information, user ID and password (if required), default database, and other related information. You can create a UDL in one of the following three ways:

- Use the **prompt?** input from the DB Tools Open Connection VI as shown in Figure 3-8.

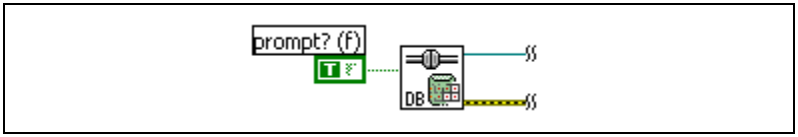

Figure 3-8. Using the Prompt to Create a UDL

  The prompt? input opens the **Data Link Properties** dialog box when the DB Tools Open Connection VI runs. You can select the appropriate options to make the database connection.

- Select **Tools»Create Data Link** in LabVIEW to open the **Data Link Properties** dialog box and save a UDL file to the `LabVIEW/database/data links` directory.

Note The Database Connectivity Toolset installer creates a directory called data links inside the `LabVIEW/database` directory. Save all UDL files and File DSNs to this directory so you can easily find them.

- Right-click in the Windows environment, the desktop or in any open directory, and select **New»Microsoft Data Link** from the shortcut menu shown in Figure 3-9. You must have the file extensions displaying in the **Tools»Folder Options»View** tab for this method to work.

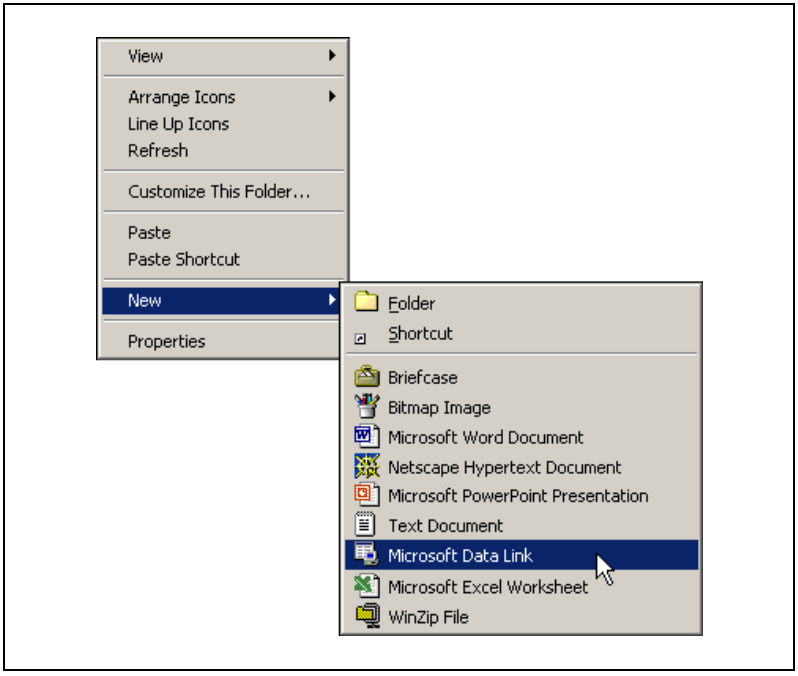

Figure 3-9. Using Windows to Create a UDL

The UDL is saved as a file (`*.udl`) in the location where you originally right-clicked. You can double-click the UDL file to open the **Data Link Properties** dialog box to configure the UDL settings.

Configuring a UDL

Any method of creating a UDL invokes the **Data Link Properties** dialog box. Select a data provider from the **Provider** tab. Refer to the *Universal Data Access, OLE DB, and ADO* section of Chapter 2, *Getting Started with the Database Connectivity Toolset*, to determine which provider to use with your database. Figure 3-10 shows the default provider of Microsoft OLE DB provider for ODBC drivers.

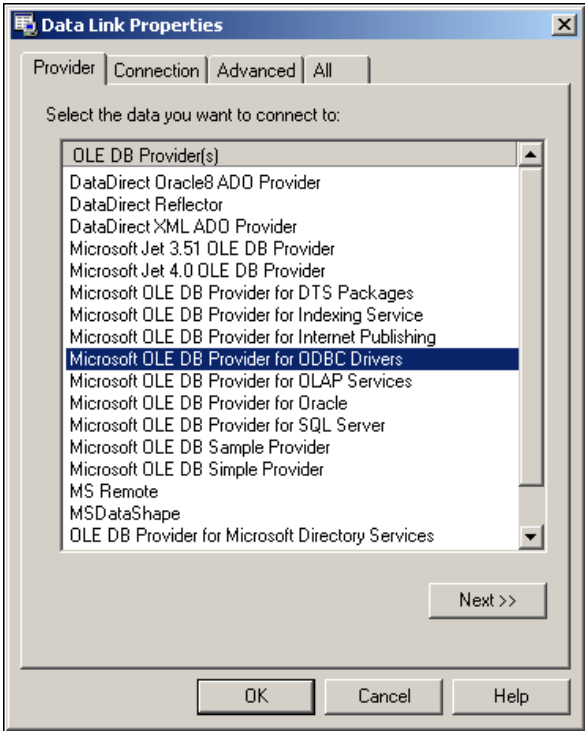

Figure 3-10. Selecting the Provider for a UDL

After you select a data provider from the list on the **Provider** tab, you then can configure the specifics of the database connection on the **Connection** tab. The options shown on the **Connection** tab are different depending upon which provider you choose. For example, the **Connection** tab for the ODBC provider contains a selection for a DSN or connection string along with user name and password information. Figure 3-11 shows the **Connection** tab options for using the Jet 4.0 provider for Microsoft Access.

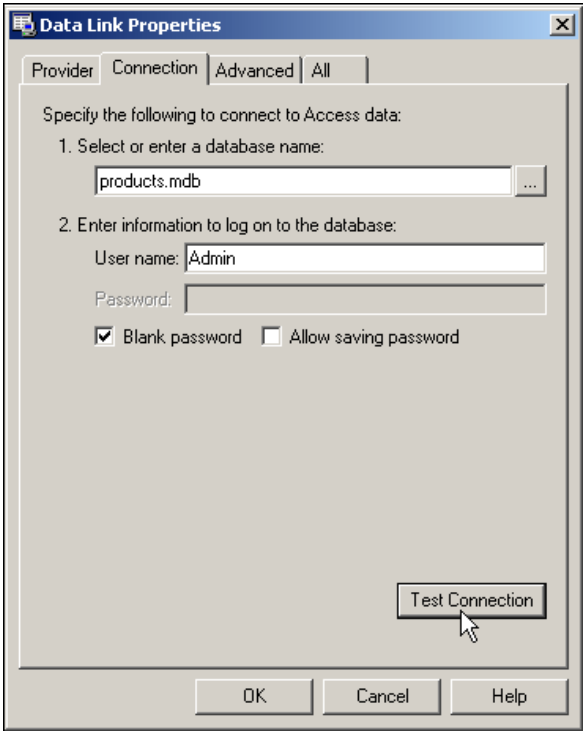

Figure 3-11. Configuring the Connection for a UDL

Click the **Test Connection** button to test the database connection after you configure the various properties. Make sure the connection test passes before you click the **OK** button to exit.

Example Using a UDL

Use a path control or constant to specify the path to the UDL file unless you wire a TRUE to the **prompt?** input on the DB Tools Open Connection VI. Figure 3-12 shows a path constant specifying the UDL for an access database.

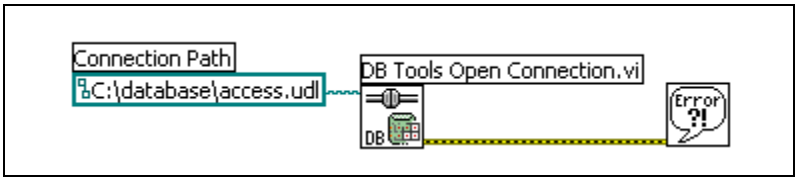

Figure 3-12. Connecting to an Access Database Using a UDL

Although you might have created the DSN or UDL correctly, you still might not be able to connect to a specific database because of situations beyond your control. For example, the requested server might be down, the network might be down, all of the server connections might be full and no other users can connect, the maximum number of user licenses might have been reached, you do not have permission to access the specified database, the specified DSN does not exist (either you are on a different machine or the specified DSN was deleted), or the selected data provider is the wrong one for the database. If you are get errors from the DB Tools Open Connection VI, you can open the UDL file manually and click the **Test Connection** button to verify that you have the correct settings and that you have access to the database. If the Test Connection fails, you cannot connect to that database with the LabVIEW Database Connectivity Toolset. Contact your Database Administrator (DBA) for help.

High-Level Database VIs

This section describes how you can use the high-level Database Connectivity Toolset VIs and function to write data to or read data from databases and to create and delete tables.

Writing Data to a Database

Writing data to a database with the Database Connectivity Toolset is similar to writing data to a file. You open a connection, insert the data, and you close the connection when you are finished. Figures 3-13 and 3-14 show the panel and diagram of a VI that writes test information into a database table. The connection information is a path to the UDL called `test.udl` and the table name is `testdata`.

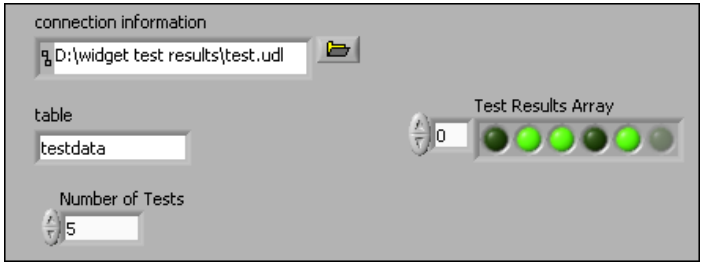

Figure 3-13. Front Panel that Writes Data to a Database Table

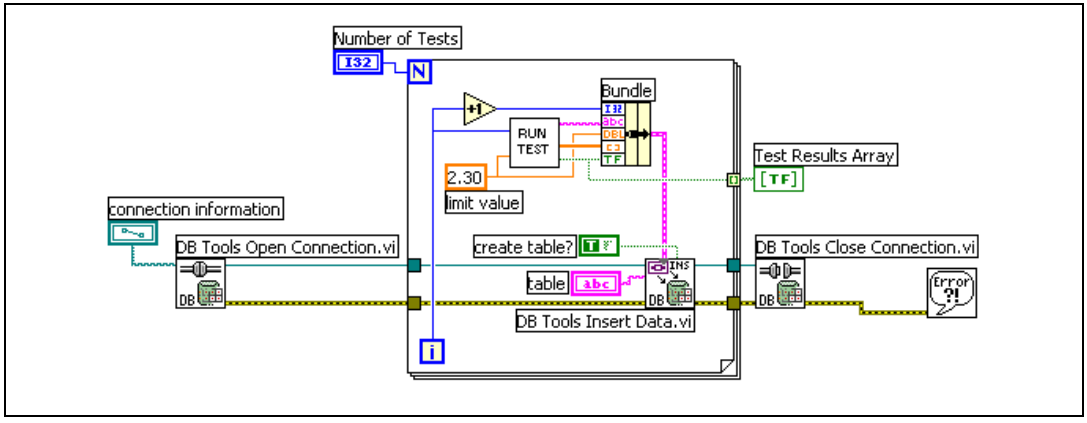

Figure 3-14. Block Diagram that Writes Data to a Database Table

Figure 3-17 uses three Database Connectivity Toolset VIs, DB Tools Open Connection, DB Tools Insert Data, and DB Tools Close Connection. The **create table?** input to the DB Tools Insert Data VI is set to TRUE to create the specified table if it does not already exist. If this table does exist, then the data is appended to the existing table. The DB Tools Insert Data VI accepts any type for the data input. If the input type is a cluster, each cluster element is placed into a different field. The LabVIEW data types are converted to the appropriate database data types. Refer to Appendix C,

Supported Data Types, for more information about data types. Figure 3-15 shows the `testdata` table as it appears in Microsoft Access. Note that the front panel and block diagram previously shown do not specify the type of database to use. That configuration occurs when the `test.udl` is created.

	col0	col1	col2	col3	col4
	1	1/25/2001 10:56:42 AM	2.3	Long binary data	false
	2	1/25/2001 10:56:42 AM	2.3	Long binary data	true
	3	1/25/2001 10:56:43 AM	2.3	Long binary data	true
	4	1/25/2001 10:56:43 AM	2.3	Long binary data	false
	5	1/25/2001 10:56:43 AM	2.3	Long binary data	true
▶					

Figure 3-15. Database Table Displayed in Access

Notice that the column names are not specified in the VI, so the table uses default column names. The **columns** input on the DB Tools Insert Data VI is an array of strings where each element is the name of a field.

Reading Data from a Database

You can read data from a database table using a similar method to writing data. You open a connection to the database, select the data from a table, and then close the connection. Figures 3-16 and 3-17 show how you can read the data back from the `testdata` table used in the previous example.

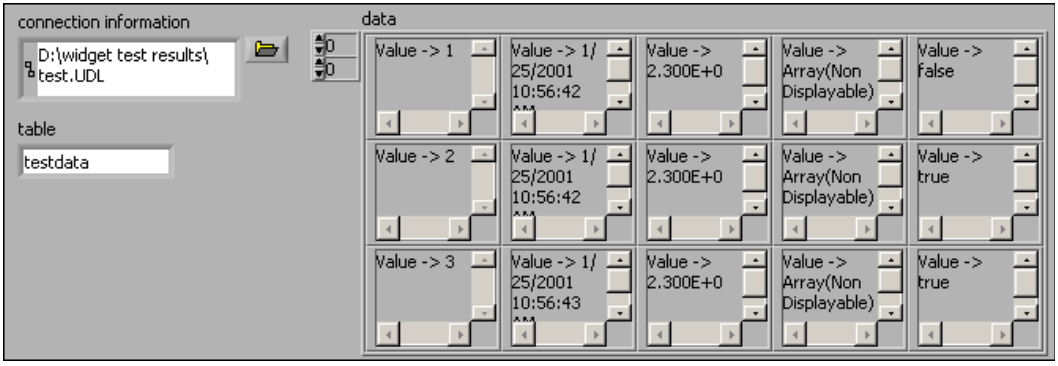

Figure 3-16. Front Panel that Reads Data from a Database Table

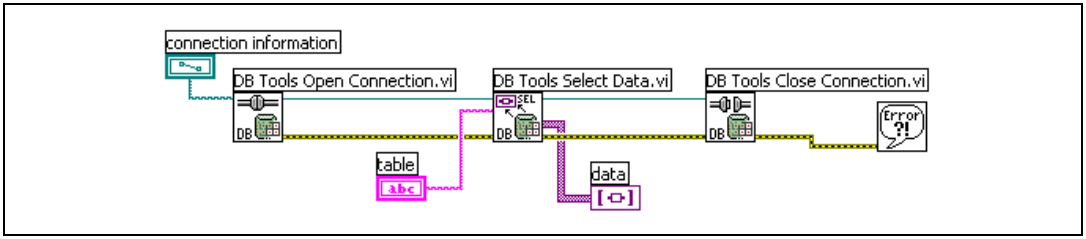

Figure 3-17. VI Block Diagram that Reads Data from a Database Table

Notice in Figure 3-16 and Figure 3-17 that the database data returns as a two dimensional array of variants. ADO is based on ActiveX that defines variants as its data types. Variants work well in languages such as Visual Basic that are not strongly typed. Because LabVIEW is strongly typed, you must use the Database Variant To Data function located on the **Functions»Database** palette to convert the variant data to a LabVIEW data type before you can display the data in standard indicators such as graphs, charts, and LEDs.

Figures 3-18 and 3-19 show the front panel and block diagram for a VI that reads all the data from a database table and then converts the data to appropriate data types in LabVIEW. Notice that the raw data array (fourth column) that is not displayed properly in either Access or the variant is now displayed in a waveform graph.

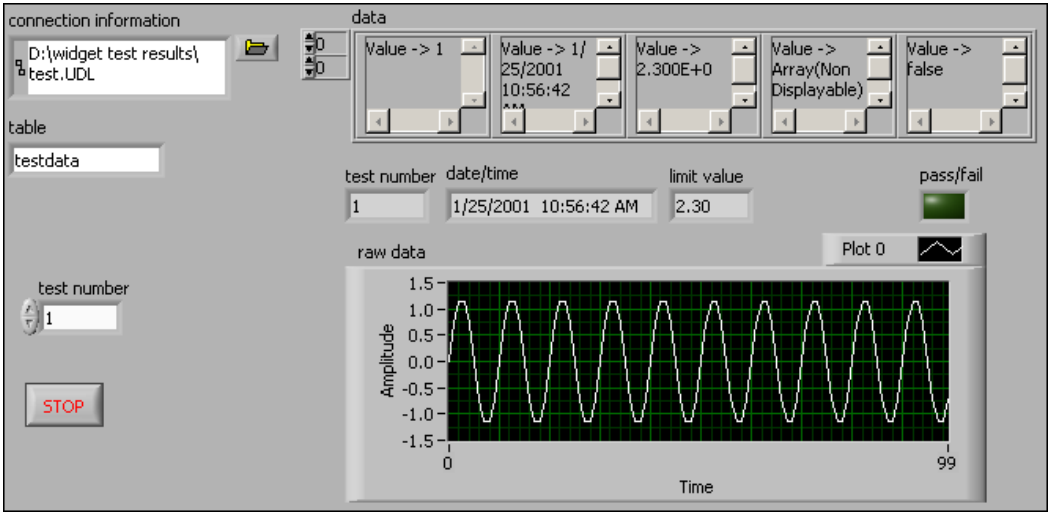

Figure 3-18. Front Panel that Reads and Converts Data from a Database Table

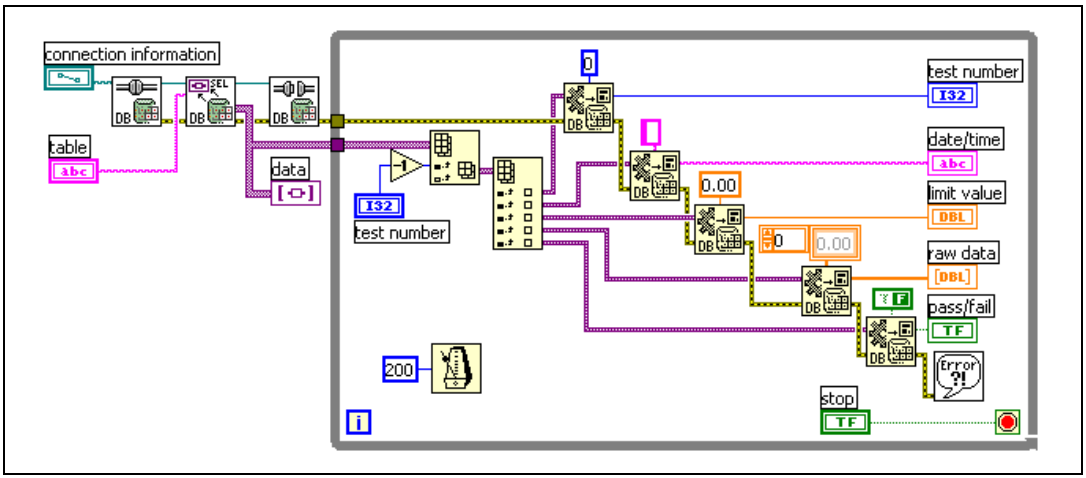

Figure 3-19. Block Diagram that Reads and Converts Data from a Database Table

You can read data from more than one table in a database using the **table** input to the DB Tools Select Data VI. Figure 3-20 shows how you can use a comma-delimited string to specify multiple table names. The data array includes all rows and columns from both tables in the order they appear in the **table** string.

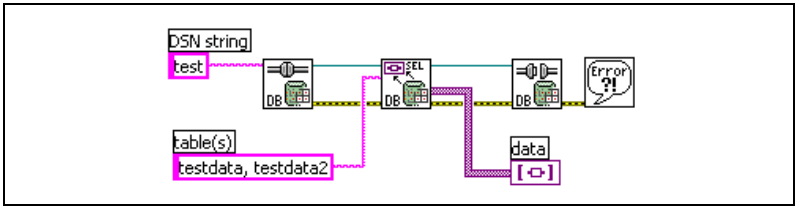

Figure 3-20. Specifying Multiple Database Tables for Reading Data

Reading Specific Data from a Table

If you are reading data from a large table or set of tables, it might take several seconds to return all the data. There is no limit other than your computer resources, memory and speed, to the size of the database table you can read. Read only the necessary fields or perform an SQL query to limit the amount of information to read into LabVIEW at one time. Figure 3-21 shows how you can limit the returned data by using the **columns** string array to specify which columns to read.

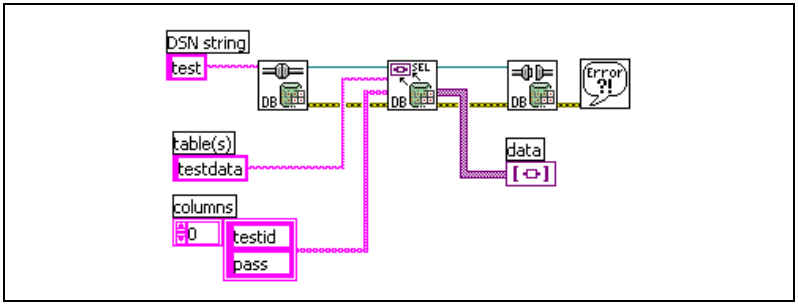

Figure 3-21. Specifying Column Names for Reading Data

In Figure 3-21, only the `testid` and `pass` fields are returned from a table named `testdata`. You can limit the returned data further by specifying conditions using the **optional clause** string. Figure 3-22 shows how you can limit the results from the previous example by returning the `testid` and `testdate` fields for the records where the `pass` field equals `TRUE`.

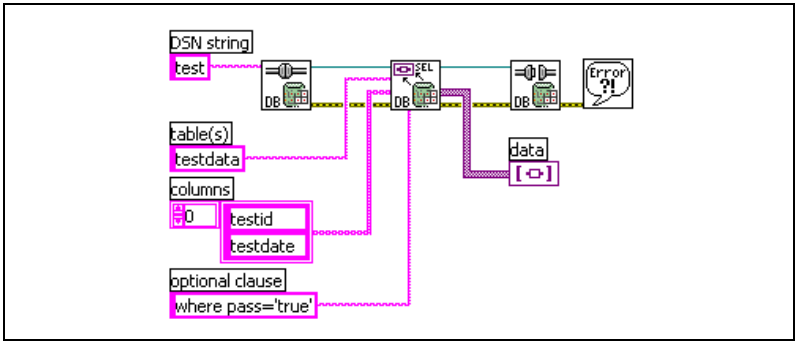

Figure 3-22. Specifying Conditions for Reading Data

The statement `where pass='true'` is part of an SQL query. Refer to Appendix A, *SQL Quick-Reference*, or one of the SQL references listed in Appendix B, *References*, for more information about how to create an SQL query.

Note If you receive an error while using the DB Tools Select Data VI, either a specified field in the **columns** string array does not exist in the table or that column name contains characters such as space, `-`, `\`, `/`, or `?`. Do not use these characters when naming tables in a database. However, if an existing database contains such characters, enclosing the column name in double quotes often solves the problem.

Creating and Deleting Tables

Use the DB Tools Create Table and DB Tools Drop Table VIs to create or delete tables in a database. Use the DB Tools Open Connection VI to connect to a database and then use the DB Tools Create Table VI or DB Tools Drop Table VI to perform the desired operation. Use the DB Tools Close Connection VI to end communication with the database. Figure 3-23 shows how you can create a new table and Figure 3-24 shows how to delete a table.

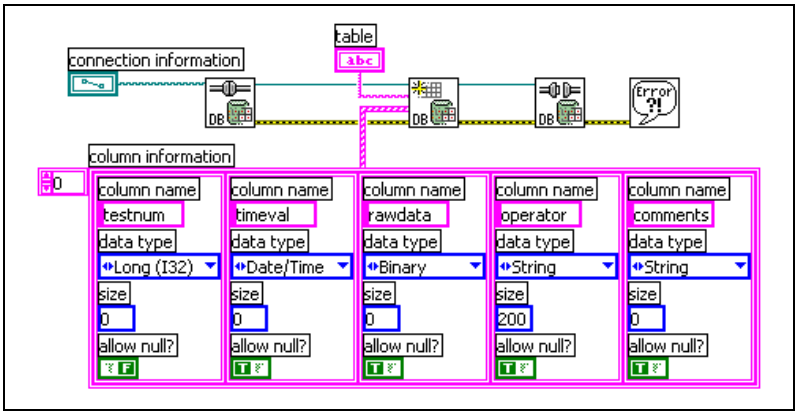

Figure 3-23. Block Diagram that Creates a Database Table

The DB Tools Create Table is a subVI in the DB Tools Insert Data VI to create the table if specified and it does not already exist. However, use the DB Tools Create Table at the highest level if you want more control over the database fields such as specifying column names, data types, and whether to allow NULL values. The size field only affects the string data type. If you use the default size of 0 for a string, then the maximum size for a string is defined by the specified provider.

Figure 3-24 shows how you use the DB Tools Drop Table VI to delete a table from a database.

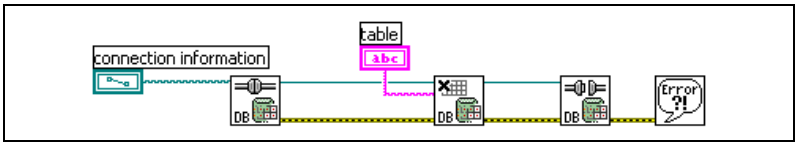

Figure 3-24. Block Diagram that Deletes a Database Table

Supported Data Types

LabVIEW, ADO, and each DBMS support a different set of data types. Therefore, the Database Connectivity Toolset maps the various LabVIEW data types to data types supported by some of the common DBMS. Table 3-1 shows which native database data types the Database Connectivity Toolset supports.

Table 3-1. Database Connectivity Toolset Data Types

Database Toolset Data Type	SQL Data Type
String	Char, VarChar
Long	Integer
Single	Real
Double	Double Precision
Date/Time	Date/Time

All LabVIEW 6.0 data types are supported but not necessarily in their native form. For example, bytes (U8 and I8) and words (U16 and I16) can be treated as longs (I32). The binary data type encompasses any piece of LabVIEW data that cannot be represented natively in the database such as waveform, cluster, or array data. Refer to Appendix C, *Supported Data Types*, for more information about data types.

Reference numbers (refnums) are the only LabVIEW data types not supported by the Database Connectivity Toolset because refnums are ephemeral constructs whose values are meaningless after usage. If you do want to save a refnum to a database table, you must first type cast the refnum to an integer and then write the integer to the table. The exceptions being DAQ, IVI, and VISA Channel refnums that are inserted as strings

into the database. These refnums are converted back into DAQ, IVI, or VISA Channel refnums using the Database Variant To Data function.

Working with Date/Time

Date/time is an important data type for databases. However, date and time in LabVIEW are represented as strings, and there is no way to differentiate a string type from date/time type. You must format date and time differently to insert date/time information into a particular database. You must use the DB Tools Format Datetime Str VI located on the **Functions»Database»Utilities** palette whenever you insert date/time strings into a database. The DB Tools Format Datetime Str VI formats the string into the correct format for SQL. It places a header at the beginning of the string that is later decoded in other VIs to determine that the string is a date/time string. The main problem with date/time is that no uniformity exists and each database supports a different format. In other words, when you select date/time values from a database, they might come back in a different form depending on the DBMS.

Handling NULL Values

Databases have NULL fields that are empty fields containing no data. LabVIEW has no concept of NULL, so NULLs are treated as default data. This means that a NULL is whatever the default value is such as an empty string, a zero-value numeric, or a FALSE boolean. You cannot easily differentiate between a `0.00` value in a numeric from one that is NULL. Figures 3-25 and 3-26 show how NULL values are represented in different formats.

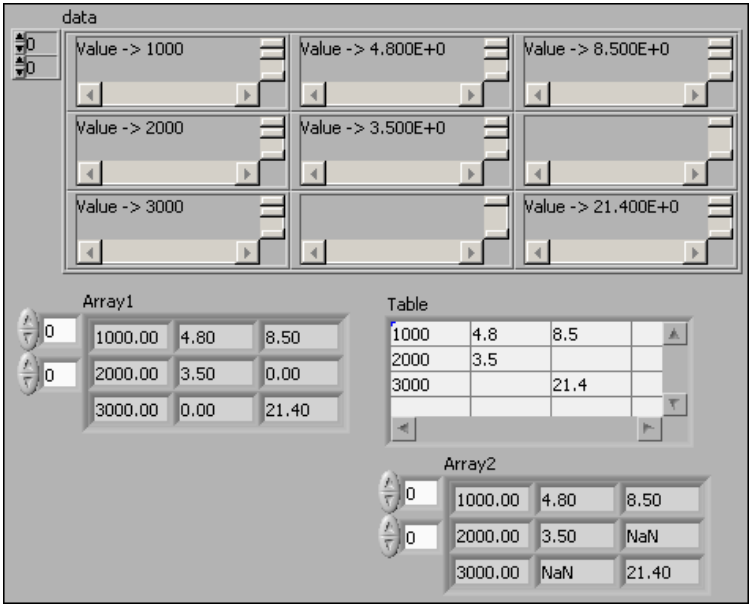

Figure 3-25. Front Panel Showing How NULLs Are Handled

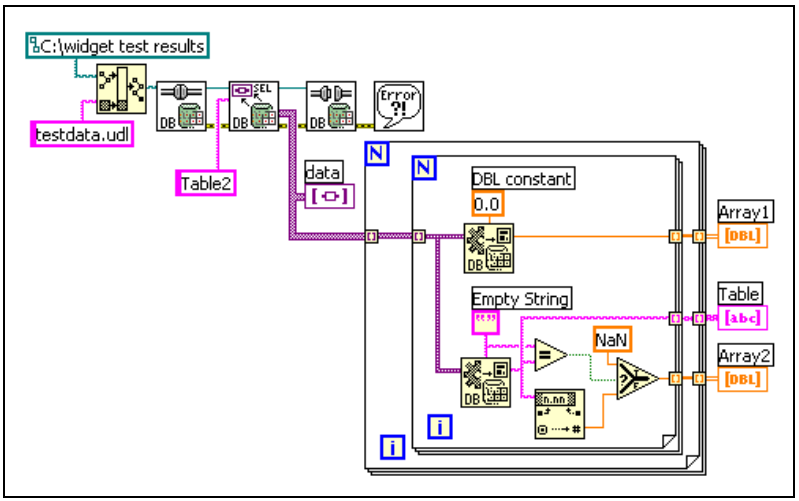

Figure 3-26. Block Diagram Showing How NULLs Are Handled

When you convert the NULL values in the variant array into numeric values, the NULLs become `0.00` values. However, when you convert the variant array into strings, the NULLs become empty strings. You convert the strings to numbers and, when the string is empty, insert a NaN (not a number) value.

Currency and Boolean Data Types

Currency and Boolean (Yes/No in Microsoft Access) are common data types and are not directly supported by the LabVIEW Database Connectivity Toolset because these data types are not available in other DBMS such as Oracle. However, you can write data to and read data from these field types with the Database Connectivity Toolset VIs using strings. Figure 3-27 shows how you can convert the Boolean and currency data to strings and write the information to the appropriate fields in an Access table.

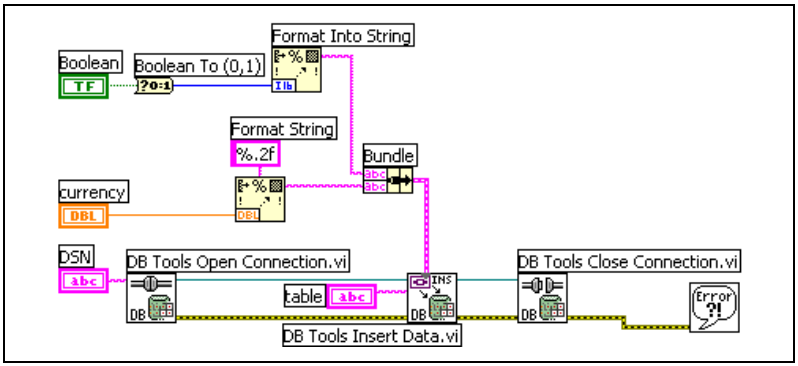

Figure 3-27. Writing Boolean and Currency Data

When you read Boolean data from a table, the data is returned as `TRUE` or `FALSE` strings. Currency data is read from a table as a number without the `$` currency symbol. Avoid using data types that are not supported by the Database Connectivity Toolset.

Using the Database Connectivity Toolset Examples

The Database Connectivity Toolset includes several examples. These examples use a UDL called `LabVIEW.udl` to link to a Microsoft Access database named `LabVIEW.mdb`. Refer to the *Database Connectivity Toolset Help* for more information about the examples.

Using the Examples with Other Databases

You can use the Database Connectivity Toolset example VIs by modifying the `LabVIEW.udl` file. Double-click this UDL file in the `LabVIEW/Examples` directory to open the Data Link Properties window. You then can select a different provider and set the connection properties for your DBMS. The default values for some of the example VIs assume

the presence of a particular table in the database to read data from or add data to. You need to modify the example to fit the table names, column names, and data types required.

Using the Examples without a Database

You do not need to have MS Access or any other database installed to use the Database Connectivity Toolset examples. If you run the examples with their default values, the data is read from or written to the `LabVIEW.mdb` file even if you do not have Access installed. If you want to create a new database file to write data to and read data from, you can copy and rename the `LabVIEW.mdb` file and use the DB Tools Drop Table VI to remove the existing tables. You then can use the DB Tools Create Table VI to create new tables specific to your application.

4

Database Connectivity Toolset Utilities

This chapter describes how to use the VIs located on the **Functions» Database»Utility** palette of the Database Connectivity Toolset for various common operations. The Utility VIs are used for a variety of operations. This chapter only covers some of those uses. Refer to the *Database Connectivity Toolset Help* for more information about using these VIs.

Getting Table and Column Information

Sometimes you must work with databases created by other users or groups, and you are not familiar with the structure of the database. You can use the DB Tools List Tables VI to determine what tables exist in a particular database. The DB Tools List Columns VI returns an array of column or field names in a table and also returns information about the data type and size of each field. Figures 4-1 and 4-2 show how you can use these Utility VIs to get information about a database.

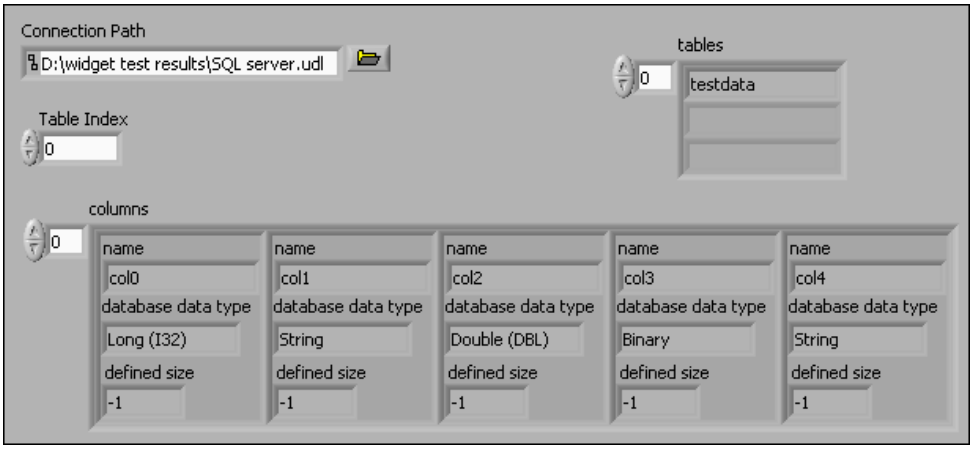

Figure 4-1. Front Panel Showing How to Get Database Information

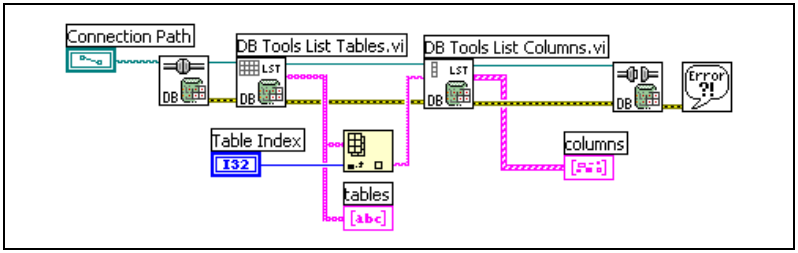

Figure 4-2. Block Diagram Showing How to Get Database Information

If you want to get information about all the fields in all the tables, you can place the DB Tools List Columns VI into a For Loop instead of using the Index Array function. Refer to Chapter 3, *Using the Database Connectivity Toolset*, and Appendix C, *Supported Data Types*, for more information about how the Database Connectivity Toolset maps the data types from DBMS to LabVIEW data types.

Getting and Setting Database Properties

You can read or set various database properties using the DB Tools Get Properties and DB Tools Set Properties VIs. Both of these VIs are polymorphic and can accept different types of reference. The exact properties you can set or read are based on what type of reference you wire to the reference input. Refer to Chapter 2, *Getting Started with the Database Connectivity Toolset*, for more information about the Connection, Command, and Recordset object classes for ADO. The Database Connectivity Toolset supports these three object classes and has a fourth called Command-Recordset to handle the close relationship between the ADO Recordset and Command objects.

ADO Reference Classes

Table 4-1 describes the purpose of each object class as it relates to the Database Connectivity Toolset.

Table 4-1. Database Connectivity Toolset Object Classes

Object Class	Description
Connection	Use this class to define the database connection parameters, such as the OLE DB provider used, the connection string used, and the default database used. After you create a Connection reference, delete it with the DB Tools Free Object VI. Refer to Chapter 5, *Advanced Database Operations*, for more information about Connection references.
Command	Use this class to execute commands and capture parameters returned from stored procedures. Create a Command reference by first creating a Connection reference and then calling the DB Tools Create Parameterized Query VI. You can get or set properties related to the command or the parameters associated with the command. After you create a Command reference, delete it with the DB Tools Free Object VI. Refer to Chapter 5, *Advanced Database Operations*, for more information about Connection references.
Recordset	Use this class to manipulate data. Create a Recordset reference by creating a Connection reference and then calling DB Tools Execute Query VI. You can get or set properties related to the column information, the number of records available, the beginning or end of file markers, and the type of cursor used. After you create a Recordset reference, delete it with the DB Tools Free Object VI. Refer to Chapter 5, *Advanced Database Operations*, for more information.
Command-Recordset	Use this class for situations where commands and recordsets are used together, such as SQL queries. Create a Command-Recordset reference by first creating Connection and Command references and then calling DB Tools Execute Query VI. You can get or set all the properties available to the Command and Recordset references. After you create a Command-Recordset reference, delete it with the DB Tools Free Object VI. Refer to Chapter 5, *Advanced Database Operations*, for more information about Connection references.

Be careful what references you wire from one VI to the next. You might obtain unexpected results when you wire a different type of reference than expected to the input of a VI. Refer to the *Database Connectivity Toolset Help* for more information about what reference types are used for VI inputs and outputs.

Specific Properties

The list of available properties changes not only with the reference type but also with the data provider. Each OLE DB data provider supports different properties for each of the ADO class types. Also, OLE DB developers are not required to implement properties in the same way. If you use a particular property with one database, that same property might not work the same for another database. Some properties are read-only and you cannot set them. Refer to the *Database Connectivity Toolset Help* for more information about specific property values.

Formatting Date and Time

No standard format exists for date and time although most databases support a specific data type for date/time. The DB Tools Format Datetime Str VI formats a LabVIEW date/time string so that the other Database Connectivity Toolset VIs recognize it as a separate data type and inserts that data type into the database properly. Figure 4-3 shows the DB Tools Format Datetime Str VI used to send a time stamp to the second field of the `testdata` table.

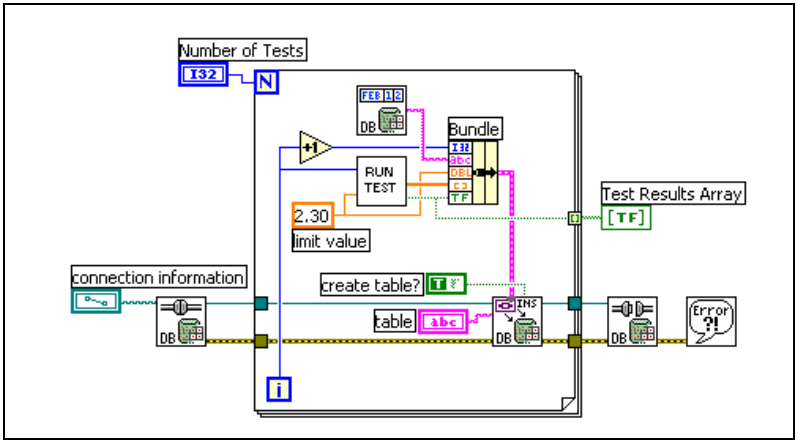

Figure 4-3. Writing Date and Time to a Database

The first version of this VI, shown in Figure 3-14, *Block Diagram that Writes Data to a Database Table*, writes the date and time to the database as a text string because LabVIEW recognizes date and time in this format. The Datasheet View in Microsoft Access shows the same information as shown in Figure 3-15, *Database Table Displayed in Access*. However, the Design View for the table in Figure 3-15 shows Field `col1` as a text data

type. Figure 4-4 shows the data type for the table created by the diagram in Figure 4-3 as Date/Time.

testdata2 : Table

Field Name	Data Type	Description
col0	Number	
col1	Date/Time	
col2	Number	
col3	OLE Object	
col4	Text	

Figure 4-4. Table Showing How to Get Database Information

You read this data back into LabVIEW exactly the same way as before. You treat the date/time data as a string.

Performing Database Transactions

Protecting the integrity of a database is often difficult. Multiple users can have access to a single database at the same time and each of them can change the data. You can use the DB Tools Database Transaction VI as shown in Figure 4-5 to specify when to actually perform, or commit, a database operation and when to return to the previous state of, or rollback, the database operation.

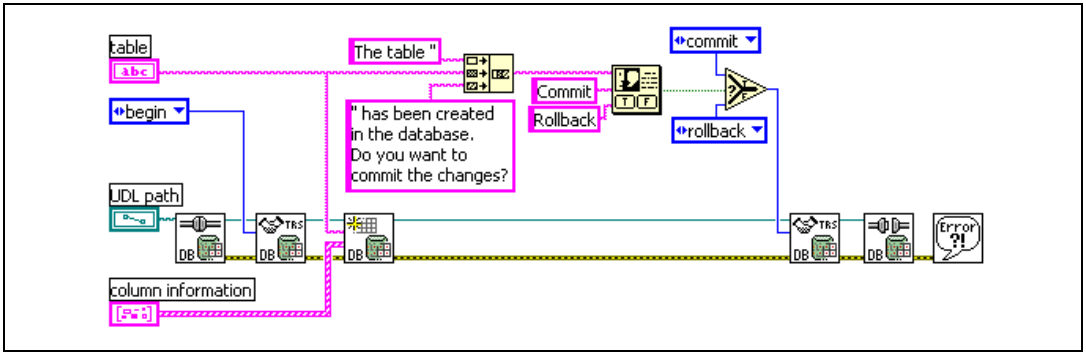

Figure 4-5. Block Diagram of the Transaction Example

The simple example shown in Figure 4-5 demonstrates how you can open a database connection, start a transaction, create a table, prompt the user to either commit or rollback the transaction, and close the database connection. If the user selects to commit the changes, the table is created as specified by the **column information** cluster. If the user selects to rollback the changes, the table is not created. You can use a similar method to protect the data in your database tables. You can group operations that belong

together into a single transaction and commit the transaction when you are finished or rollback the transaction if an error occurs. You also can use locking to determine who has access to a database during a transaction.

Locking Transactions and Setting Isolation Levels

Locking is an important activity in multi-user database systems where different users have access to the same data at the same time. Without data locking, more than one user can modify the same record at the same time, possibly causing data inconsistencies. Locking provides a way to have concurrent database access while minimizing the various problems it can cause.

Isolation levels represent different locking strategies. The higher the isolation level, the more complex the locking strategy is and the better it is at preventing data inconsistencies. The following data inconsistencies are examples of what different isolation levels try to prevent:

- Dirty Reads—User 1 modifies data while user 2 uses that same data before user 1 can commit the changes. User 2, therefore, uses incorrect data.
- Non-Repeatable Reads—User 1 reads records while user 2 modifies records. User 1 rereads the records and finds that a record has changed or been deleted.
- Phantom Reads—User 1 reads records while user 2 adds records. User 1 rereads the records and finds additional records.

The DB Tools Database Transaction VI, as shown in Figure 4-6, contains an optional input for setting the isolation level used for the transaction. You need to set this value only if other transactions might be pending at the same time.

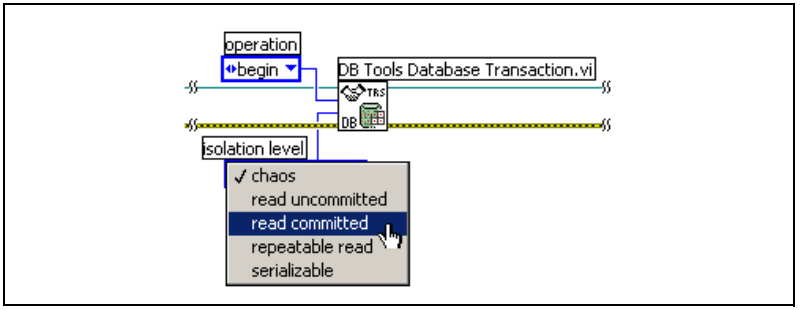

Figure 4-6. Block Diagram of the Transaction Example

At the lowest level of isolation, all the problems mentioned previously can occur. At the highest level of isolation, none of these problems can occur. Different databases support the following different isolation levels:

- Chaos (lowest level)—Transactions are not safe from each other. One transaction might overwrite another.
- Read Uncommitted—Locks are obtained on modifications only and held to the end of the transaction. Reading does not involve any locking. Dirty reads, non-repeatable reads, and phantom reads are all possible.
- Read Committed—Locks are obtained on reading and modification, but locks are released after reading and held until the end of the transaction for modifications. This transaction cannot see changes made by other transactions until they are committed. Dirty reads are not possible, but non-repeatable reads and phantom reads are possible.
- Repeatable Read—Locks are obtained on reading and modifications. Locks are held until the end of the transaction for both reading and modifying records. Locks on non-modified access are released after reading. You do not see any changes in records without re-querying the database. Dirty reads and non-repeatable reads are not possible, but phantom reads are possible.
- Serializable (highest level)—All read or modified data is locked until the end of the transaction. The transaction occurs in complete isolation. Dirty reads, non-repeatable reads, and phantom reads are not possible.

When you choose a higher isolation level, you improve the locking strategy, but you will have less user concurrency. The presented data might not be the most current data.

Writing and Reading Data Files

You can use the File I/O functions and VIs to write the database data to a file just as you do with any other data in LabVIEW. You also can use the DB Tools Save Recordset To File VI and DB Tools Load Recordset From File VI to write the database data to file. Rowset Persistence describes writing database records to file. The data is sent to the file and also the structure and properties of the database recordset. Figure 4-7 shows how you can write recordset data to a file.

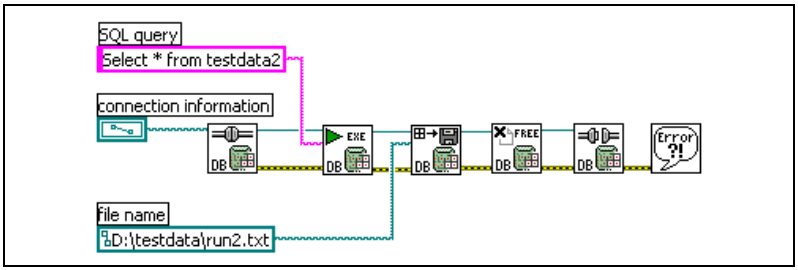

Figure 4-7. Writing Data to File

The DB Tools Save Recordset To File VI requires a recordset reference input, so you must use the DB Tools Execute Query VI to generate a recordset reference. This VI executes an SQL query. If you put a table name into the SQL query input to the DB Tools Execute Query VI, it includes all the information in that table. The recordset reference references the results, passes them to the DB Tools Save Recordset To File VI, and writes them to file. The DB Tools Free Object VI releases the recordset reference and the connection to the database is closed.

The recordset data saves to file in one of the following two formats:

- Advanced Data TableGram (ADTG) format is a proprietary Microsoft binary format. ADTG has the advantage of being a compact binary format that results in much smaller files written faster than if you use XML format.
- Extensible Markup Language (XML) is a text-based industry standard specification for describing data. It is similar to HyperText Markup Language (HTML) except XML can use an unlimited set of tags to describe any type or structure of data. Figure 4-8 shows the resulting XML data file written by the VI in Figure 4-7.

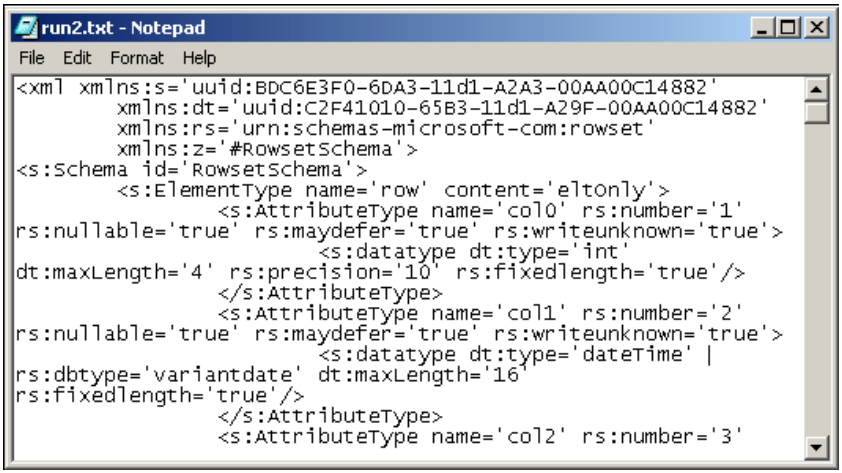

Figure 4-8. Persisted Data in XML Format

You can see the database structure and data attributes described in the XML formatted file. Persisting data to files in the XML and ADTG formats is a feature of ADO and OLE DB and not a special feature of the Database Connectivity Toolset.

You can read data back into LabVIEW from one of these files using the DB Tools Load Recordset From File VI as shown in Figure 4-9.

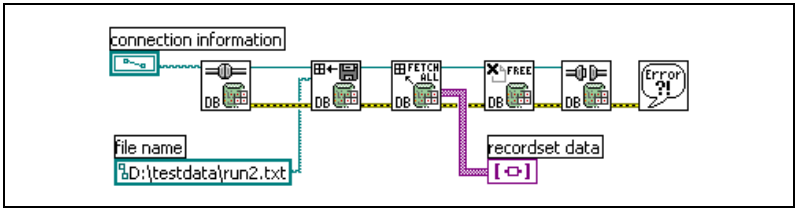

Figure 4-9. Persisting Data from File

The DB Tools Load Recordset From File VI returns a recordset reference when it opens the specified file. You then can use any VI from the Database Connectivity Toolset that accepts a recordset reference to perform operations on that data. For the example shown is Figure 4-9, the DB Tools Fetch Recordset Data VI returns the recordset data as a two-dimensional array of variants. The DB Tools Free Object VI releases the recordset reference and closes the database connection.

5

Advanced Database Operations

This chapter describes how to use the Advanced VIs in the LabVIEW Database Connectivity Toolset for advanced database operations. The chapter describes how to execute SQL statements and fetching data using the Database Connectivity Toolset Advanced VIs.

Use the Advanced VIs for more detailed database operations. The Advanced VIs and the Utility VIs are the building blocks for the VIs on the **Functions»Database** palette. You use the Advanced VIs when you need more control over what is sent to or read from the database. You might need to use SQL when you use these lower-level VIs. Refer to Appendix A, *SQL Quick-Reference,* for more information about constructing SQL command strings.

Executing SQL Statements and Fetching Data

SQL is the language used with relational databases. Common operations include creating and deleting tables, inserting data into databases, querying databases for particular recordsets, and manipulating data in tables. Refer to Appendix B, *References*, for more information about SQL. This section describes how you can use SQL statements with the Database Connectivity Toolset and how you can fetch the data resulting from an SQL query.

Use the DB Tools Execute Query VI to send an SQL string to a database as shown in Figure 5-1. You can then use any of the Fetch VIs, DB Tools Fetch Recordset Data, DB Tools Fetch Element Data, and DB Tools Fetch Next Recordset, to return the results of a query. The SQL string does not have to specify a query only. You can put any SQL statement into the SQL string. You do not need to specify any string if you are passing a Command reference to the DB Tools Execute Query VI. The DB Tools Create Parameterized Query VI creates a Command reference. When the DB Tools Execute Query VI receives a Command reference input, the VI executes the previously created SQL query.

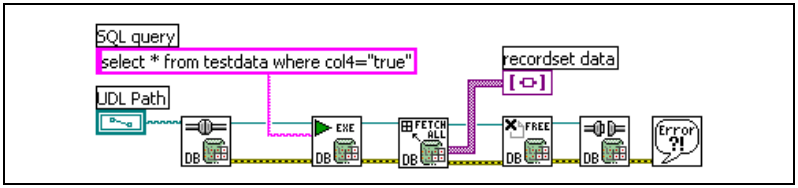

Figure 5-1. Fetching All the Query Results

The SQL query string shown in Figure 5-1 asks for all the records in the `testdata` table where the fifth field contains a TRUE value. The DB Tools Fetch Recordset Data VI returns a two dimensional array of variants where all the tests passed. Refer to Chapter 3, *Using the Database Connectivity Toolset*, for more information about converting the variant array to LabVIEW data types. The DB Tools Execute Query VI creates a Recordset reference, so you must use the DB Tools Free Object VI to release the Recordset reference value.

Note A record is a single row of data and a recordset is a collection of records, or multiple rows, from a database table.

The DB Tools Fetch Recordset Data VI returns all the records from a query as shown in Figure 5-1. If you know this query will return a large amount of data or you want to retrieve information from one record, you can use the DB Tools Fetch Element Data as shown in Figure 5-2.

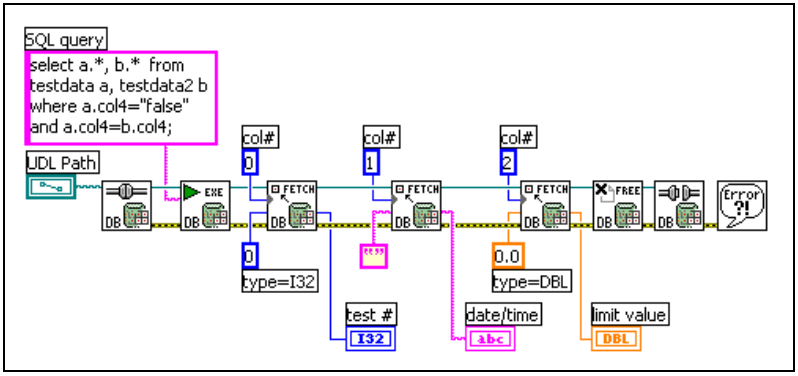

Figure 5-2. Fetching the Query Results from One Record

The DB Tools Fetch Element Data VI returns a value from a single field. You can specify the field, or column, either by a numerical value as shown in Figure 5-2 or by a string specifying the column name. You must also specify the data type as the DB Tools Fetch Element Data VI uses the

Database Variant To Data function. Notice the SQL query used in Figure 5-2. This is an example of an SQL inner join operation. An inner join is when you combine the fields of several tables through a common value or expression. The records in the `testdata` and `testdata2` tables are combined for all the tests where both tables contain failed tests.

The DB Tools Fetch Recordset Data VI returns all the records that satisfy the SQL query and the DB Tools Fetch Element Data VI returns an element from the first record that satisfies the query. Use the VIs described in the *Navigating through Database Records* section later in this chapter to navigate through the resulting records. Also, some SQL queries such as stored procedures return multiple recordsets. Use the DB Tools Fetch Next Recordset VI to read each recordset.

Note Not all databases support queries that return multiple recordsets. The inner join statement in Figure 5-2, returns a single recordset that happens to contain the results from multiple tables. Therefore, the recordset might contain several records, or rows, where the columns from multiple tables have been joined.

Navigating through Database Records

Operations in a relational database act on a complete set of rows. The recordset returned by an SQL SELECT statement consists of all the rows that satisfy the conditions of the statement. Applications, especially interactive and online applications, cannot always work effectively with the entire recordset as a unit. Use cursors to allow applications that cannot work with the entire recordset as a unit to work with one row at a time.

Note The Database Connectivity Toolset does not require you to know about cursors in order to use them. However, the following information can help advanced users who wish to have more control over their applications.

Using Cursors

A cursor is a placeholder that points to a specific record in a recordset. A cursor keeps track of the position in the recordset and allows you to perform multiple operations row by row against a recordset, with or without returning to the original table. Every cursor uses temporary resources to hold its data. These resources can be memory, a disk paging file, temporary disk files, or even temporary storage in the database. Cursors can reside in one of the following two locations:

- Client-side cursor—The temporary storage resources are located on the client computer. In addition, the client receives the entire database

recordset across the network. Client-side cursors lead to very quick database operations because everything happens locally on the client machine. However, when you work with large databases, a client-side cursor can be extremely expensive in time and memory use because the client machine must receive all the data from the server. Also, only the static cursor type is supported by client-side cursors. Refer to the *Cursor Types* section later in this chapter for more information about cursor types.

- Server-side cursor—The temporary storage resources are located on the database server machine. The server-side cursor returns only the requested data over the network. Server-side cursors provide better performance than the client-side cursor when you work with large databases or in situations where excessive network traffic is a problem. You have a choice of four different cursor types when you use a server-side cursor.

The Database Connectivity Toolset uses only server-side cursors because they offer more flexibility, and because they provide better performance for large amounts of data.

Cursor Types

Although you can only use a server-side cursor with the Database Connectivity Toolset, you do have a choice of server-side cursor types. The type of cursor used by your application to navigate the recordset affects the ability to move forward and backward through the rows in a recordset, sometimes called scrollability. The ability to move forward and backward through a recordset adds to the time and resources necessary to use the cursor. Use the simplest cursor that provides the required data access and only change the cursor type if you absolutely need the added functionality. Set the cursor type using the DB Tools Execute Query VI by creating a cursor type constant, and then selecting from the choices, as shown in Figure 5-3.

Note Not all data providers and/or databases support all the cursor types shown in Figure 5-3. For example, the Jet 4.0 OLE DB Provider for Microsoft Access does not support dynamic cursors, so if you request a dynamic cursor, the provider gives you a static cursor that is not correctly implemented.

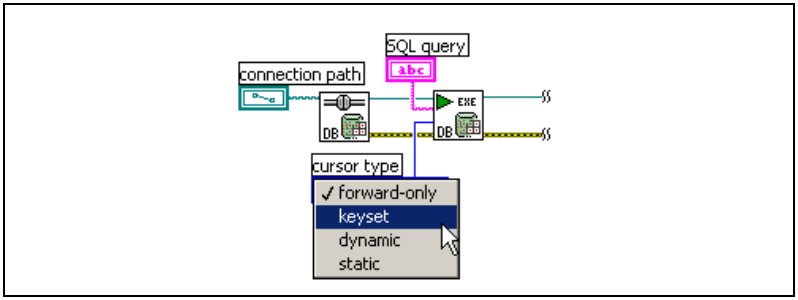

Figure 5-3. Possible Cursor Types

Your cursor choice depends on whether you need to change or simply view the data. If you just need to scroll through a set of results but not change data, use a forward-only or static cursor. If you have a large result set and need to select just a few rows, use a keyset cursor. If you want to synchronize a result set with recent adds, changes, and deletes by all concurrent users, use a dynamic cursor. The following cursor types are available:

- Forward-only—This cursor is the default and permits only forward movement through the recordset. Any changes made to the database by other users during navigation will not be seen. Forward-only cursors are dynamic because detection of changes occurs as the current row is processed. This is a high-performance cursor that uses the least resources.
- Keyset—This cursor allows forward and backward navigation. You can see records added by other users, but records deleted by others will not be removed from view.
- Dynamic—This cursor allows forward and backward navigation. You can see all changes made to the database, locally and by other users. Use the dynamic cursor if your application must detect all concurrent updates made by other users.
- Static—This cursor allows forward and backward navigation with no ability to see any changes made by other users during navigation. The static cursor always displays the result set as it was when the cursor was first opened. Use the static cursor if your application does not need to detect data changes and requires scrolling.

Moving Through Recordsets

Use the DB Tools Move To Next Record, DB Tools Move To Previous Record, and DB Tools Move To Record N VIs to navigate through the results of a database query. Figures 5-4 and 5-5 show how you can scroll forward and backward through a recordset using a static cursor.

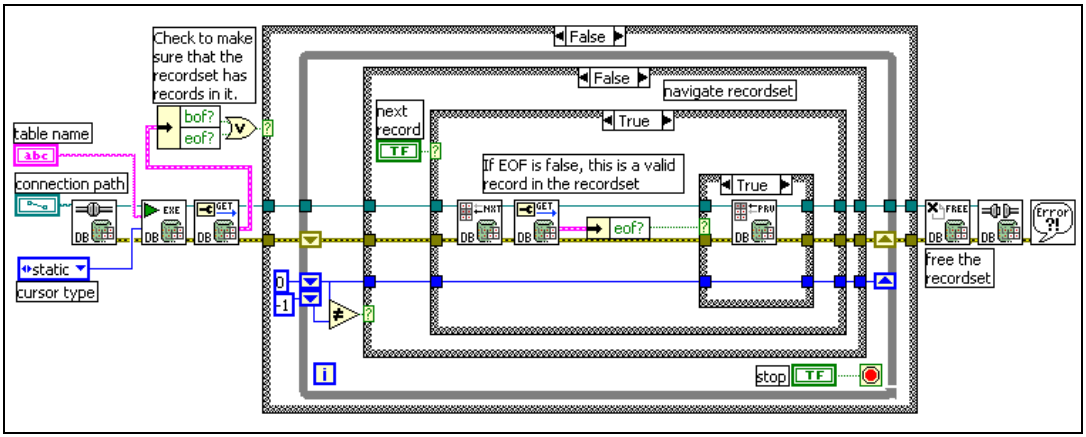

Figure 5-4. Navigating to the Next Record in a Recordset

First, the connection to a database is opened and the DB Tools Execute Query is used to open a table and specify a static cursor. Usually, you use the DB Tools Execute Query to send an SQL statement to a database. However, if you send the table name to this VI, it returns a recordset reference to all the records in that table. The DB Tools Get Properties VI returns the beginning of file (BOF) and the end of file (EOF) markers to make sure that there are records in the table. There are two buttons on the panel labeled Next Record and Previous Record. Clicking Next Record, calls the DB Tools Move To Next Record VI, the EOF property is read, and trying to move past the last record in the table, calls the DB Tools Move To Previous Record VI. The cursor is now pointed at the last record in the table.

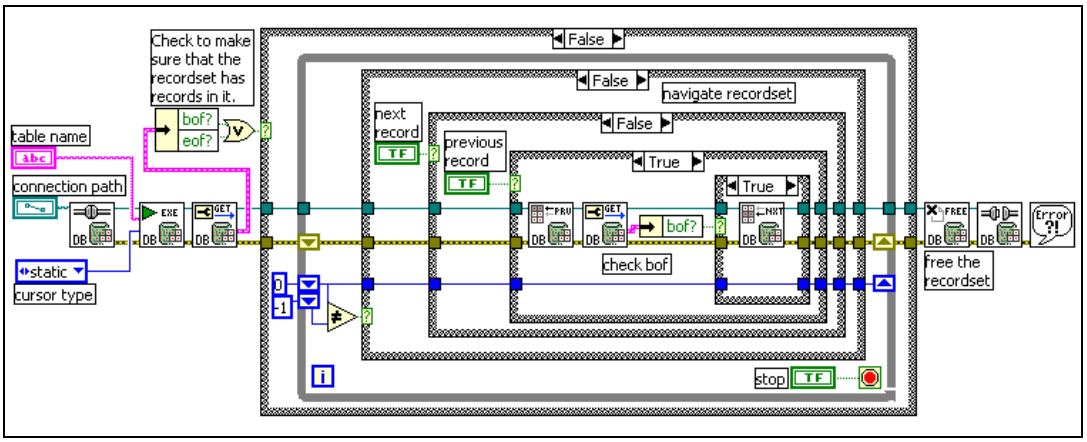

Figure 5-5. Navigating to the Previous Record in a Recordset

Figure 5-5 shows what happens when you click the **Previous Record** button. The DB Tools Move To Previous Record VI is called, the BOF property is read, and if you have tried to move before the first record in the table, the DB Tools Move To Next Record VI is called. This leaves the cursor pointed at the first record in the table. Notice that you need to use the static cursor type in these examples because you are scrolling forward and backward in the recordset.

Figure 5-6 shows how you use the DB Tools Move To Record N VI to display the information from any record in a table when you know the record number. The first record in the recordset has a value of zero.

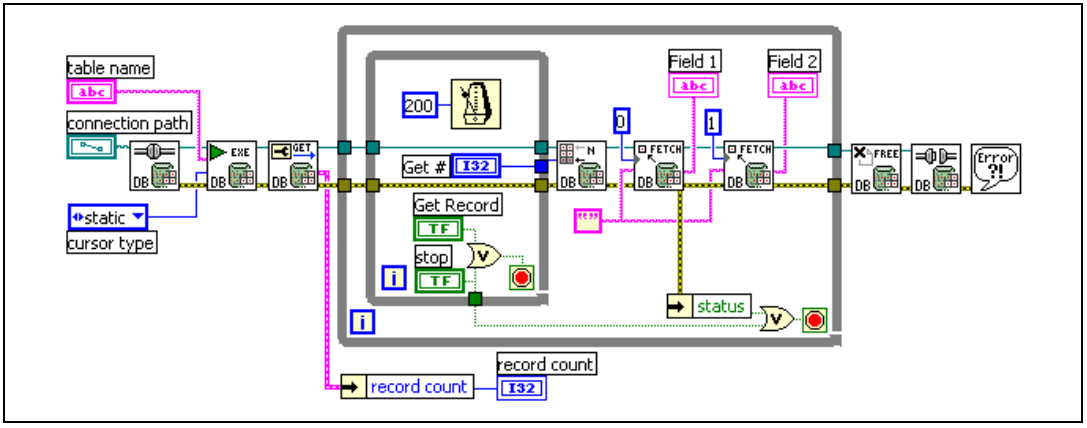

Figure 5-6. Navigating to the nth Record in a Recordset

The DB Tools Get Properties VI used in Figure 5-6 returns the number of records in the table and displays that value in the panel. You can also use this value to make sure the user does not enter a value for **Get #** that is not a valid record number. The VI in Figure 5-6 ends with an error message if you ask for a record number that does not exist. The static cursor used in Figure 5-6 allows scrollability through the entire recordset.

Using Parameterized Statements

You can use parameters in statements when using the Database Connectivity Toolset. Parameterized statements allow you to specify the SQL statement once, but vary the parameters, such as the matching criteria of a WHERE clause, over time. Prepare a parameterized statement using the DB Tools Create Parameterized Query VI as shown in Figure 5-7. You specify the SQL statement with parameters. You create the parameters input either by creating a control on the panel or by creating a constant on the block diagram. The parameters input is an array of clusters where each array value represents a column or field in the database table. Each parameter cluster contains the following four values:

- The name of the parameter—This is a string that you can leave empty if the parameter is not named.
- The parameter type—This is an enumerated type with the following values: string, long, single, double, date/time, or binary.
- The parameter direction—This is an enumerated type with the following four values: input, output, input/output, or return value.
- The initial value—You can leave this value as an empty variant and set the value later with the DB Tools Set Parameter Value VI as shown in Figure 5-7.

The example shown in Figure 5-7 shows the SQL statement, `insert into testresults (channel, dateval, binval) values (?, ?, ?)`. This means that a table named `testresults` already exists with three fields (columns) named `channel`, `dateval`, and `binval`. The question marks represent parameters set later in the data acquisition loop. The **parameters** array also specifies the names of the parameters and their data types. These data types must match the data types as defined by the `testresults` database table. The DB Tools Create Parameterized Query VI returns an error if the number of parameters described in the SQL statement do not match the number of array elements in the **parameters** input, if the column names in the SQL statement do not match the column names in the `testresults` database table, or if the data types defined in

the **parameters** array cluster do not match the data types in the `testresults` database table.

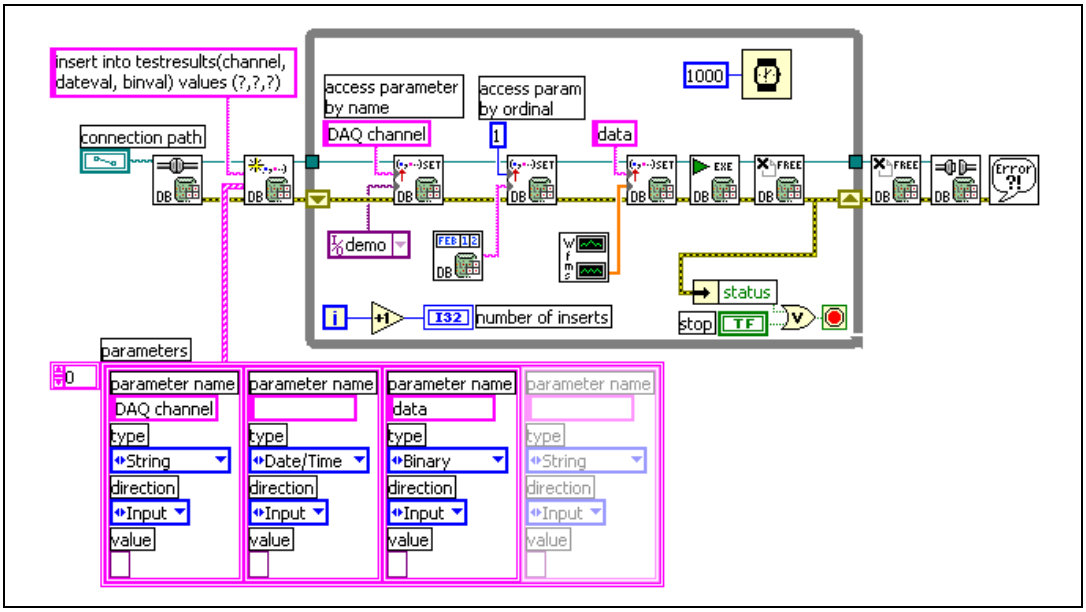

Figure 5-7. Writing Parameterized Data to a Table

Figure 5-7 shows the DB Tools Open Connection and the DB Tools Create Parameterized Query VIs used before the data acquisition loop begins. Inside the acquisition loop, the DB Tools Set Parameter Value VI writes the data to the database. Notice that you can specify the parameter index input as either a string specifying the parameter name or as a number representing the index into the parameters array. After setting all the parameter values, the DB Tools Execute Query VI executes the statement. The DB Tools Free Object VI and the DB Tools Close Connection VI release the various reference values and close the database connection.

The example in Figure 5-7 can be viewed in a different way by taking the ADO object reference types into account. The DB Tools Open Connection VI creates a Connection reference. The DB Tools Create Parameterized Query VI uses the Connection reference and creates a Command reference. The Command reference passes through the DB Tools Set Parameter Value VI and the DB Tools Execute Query VI changes it to a Command-Recordset reference. The first DB Tools Free Object VI takes the Command-Recordset reference and returns a Command reference. The second DB Tools Free Object VI takes the Command reference and returns a Connection reference. Last, the DB Tools Close Connection VI releases

the Connection reference. Each time an ADO object reference opens, you must call the DB Tools Free Object to close it.

You can use parameters in any kind of SQL statement and not just INSERT statements as shown in the example in Figure 5-7. However, not all databases or data providers support parameterized statements. Refer to your ADO, data provider, or database documentation for more information about what features are supported. Parameterized SQL queries are an advanced topic in most books about SQL programming. Parameters used with stored procedures are discussed in the next section.

Using Stored Procedures

A stored procedure is a precompiled collection of SQL statements and optional control-of-flow statements, similar to a macro. Each database and data provider supports stored procedures differently. For example, you can create a stored procedure using the Jet 4.0 provider, but Access does not support them through its usual user interface. A stored procedure created in one DBMS might not work with another. This section describes how to create and run stored procedures, with and without parameters, using the Database Connectivity Toolset.

Although stored procedures are an advanced topic, they offer the following benefits to your database applications:

- Performance—Stored procedures are usually more efficient and faster than using regular SQL queries because the SQL statements are parsed for syntactical accuracy and precompiled by the DBMS when the stored procedure is created. Also, combining a large number of SQL statements with conditional logic and parameters into a stored procedure allows the procedures to perform the queries, make decisions, and return the results without extra trips to the database server.
- Maintainability—You isolate the lower-level database structure from the LabVIEW application by using stored procedures. As long as the table names, column names, parameter names and types do not change from what is stated in the stored procedure, you do not need to modify the procedure when changes are made to the database schema. Stored procedures are also a way to support modular SQL programming because after you create a procedure, you and other users can reuse that procedure without knowing the details of the tables involved.
- Security—When creating tables in a database, the DBA is able to set EXECUTE permissions on stored procedures without granting

SELECT, INSERT, UPDATE, and DELETE permissions to users. Therefore, the data in these tables is protected from users who are not using the stored procedures.

Creating Stored Procedures

You usually create stored procedures in the DBMS environment. Some DBMSs, such as SQL Server contain a library of system stored procedures, with names starting with `sp_`, that perform common administrative tasks with databases. Refer to the documentation for your DBMS for the exact syntax to use when creating a stored procedure. You also can use the LabVIEW Database Connectivity Toolset to create stored procedures as shown in Figure 5-8.

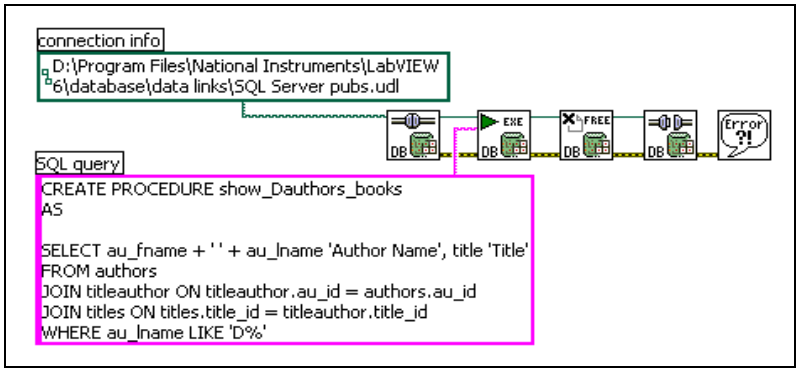

Figure 5-8. Creating a Stored Procedure

Figure 5-8 uses the same VIs as performing a typical SQL query, DB Tools Open Connection, DB Tools Execute Query, DB Tools Free Object, and DB Tools Close Connection. The syntax of the SQL query string is the only difference. The SQL query string is a stored procedure that calls the show_Dauthors_books query. The query string specifies the use of three tables in the `pubs` database. The procedure joins the `authors`, `titles`, and `titleauthor` tables to create a list of all the authors whose last name begins with D, and the books they have published. The procedure also arranges the result, where the first column combines the first and last names and the second column contains the book title. When the VI runs, it creates a stored procedure that does not use parameters. The *Running Stored Procedures without Parameters* section describes how you can then call the stored procedure using another VI.

Running Stored Procedures without Parameters

You can run a stored procedure by simply inserting the name of the procedure as an SQL query. Figures 5-9 and 5-10 show how you can call the stored procedure created in Figure 5-8 using the DB Tools Execute Query VI.

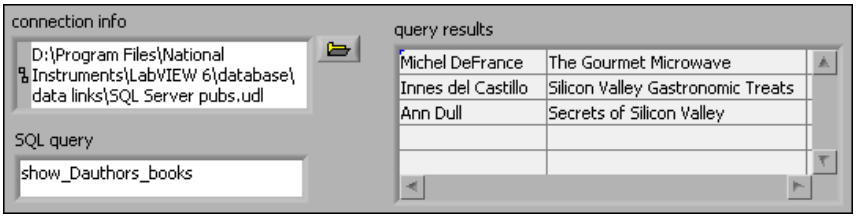

Figure 5-9. Front Panel Running a Stored Procedure

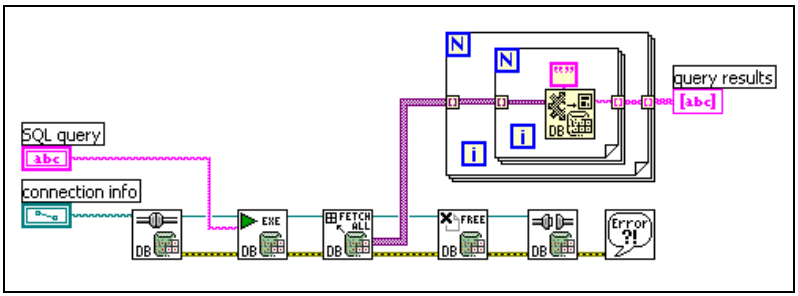

Figure 5-10. Block Diagram Running a Stored Procedure

Figure 5-10 shows the DB Tools Execute Query VI sending the name of the stored procedure. The DB Tools Fetch Recordset Data returns the results of the stored procedure in a two dimensional array of variants. The nested For Loops convert the variants to strings so the results can be displayed in a LabVIEW table indicator.

Running Stored Procedures with Parameters

Stored procedures can use variables internally as well as pass parameters into and out of the procedure. Using the Database Connectivity Toolset, there are two ways available to use parameters with stored procedures. In the first method, you build SQL query strings that contain the name of the stored procedure with the values embedded at the appropriate places in the query. For example, assume you want to use the following stored procedure:

```
CREATE PROCEDURE AddPart
        @part_name char(40),
        @part_qty int,
        @part_price money,
        @part_descr varchar(255) = NULL
AS
    INSERT parts (name, qty, price, description)
    VALUES (@part_name, @part_qty, @part_price,
    @part_descr)
```

This stored procedure adds a record containing the part name, quantity, unit price, and description to the table named parts. Figure 5-11 shows the block diagram that calls this `AddPart` procedure.

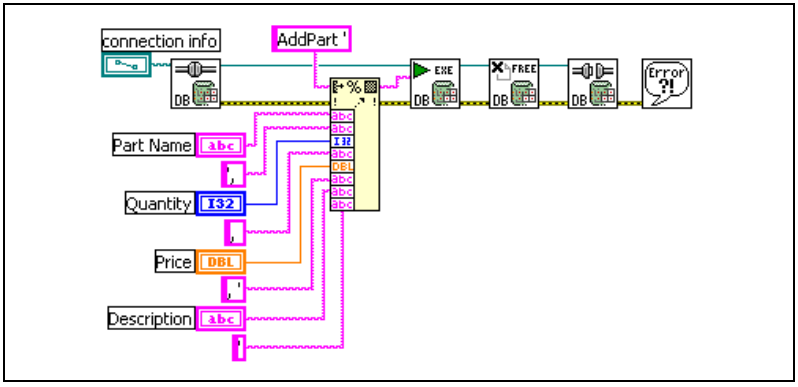

Figure 5-11. Using Parameters with a Stored Procedure

Figure 5-11 shows how you construct the SQL query string using the Format Into String function. The resulting query for this stored procedure is `AddPart 'widget', 24, 0.99, 'misc parts'`. The DB Tools Execute Query VI sends this query to the database just as you would send any other SQL Query. You must know the parameters and data types used

in a stored procedure in order to properly call it from the Database Connectivity Toolset.

The second way to use parameters with stored procedures is to use the DB Tools Create Parameterized Query, DB Tools Set Parameters, and DB Tools Get Parameters VIs. Figure 5-12 shows how you can use the previous VIs to run the same stored procedure shown in Figure 5-11.

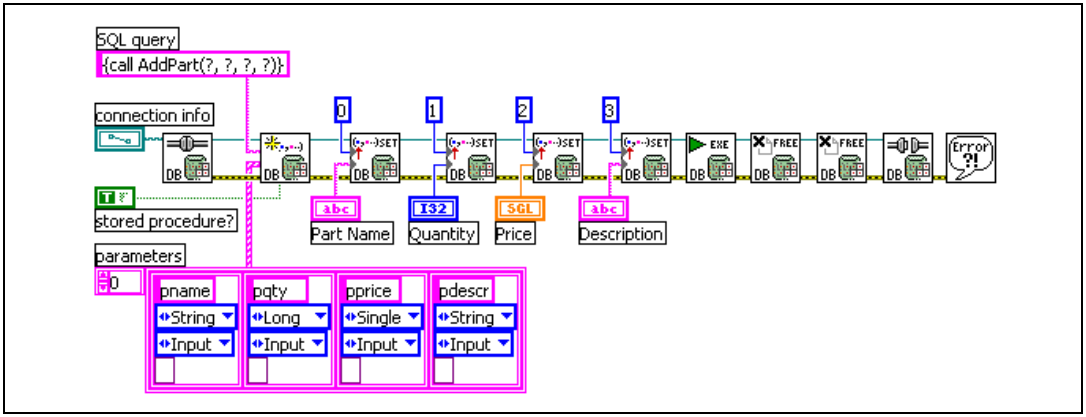

Figure 5-12. Using a Parameterized Stored Procedure

The format for the SQL query is slightly different than in the previous example. The query string shown in Figure 5-12 uses the ODBC method for calling a stored procedure where the previous example used the Transact SQL (T-SQL) method used by SQL Server. The parameters in the SQL query string are defined by the question marks and the parameters array constant needs to contain as many elements as there are question marks in the query. You specify the parameter name, data type, direction (input, output, input/output, or return value), and the parameter value in each parameter array element. You also set the **stored procedure?** input to the DB Tools Create Parameterized Query VI to TRUE.

If you did not explicitly state values in the parameters array, use the DB Tools Set Parameter Value VI as shown in Figure 5-12. The DB Tools Set Parameter Value VI is polymorphic and can accept any LabVIEW data type for the value input. The example above shows string, integer, and single values wired to the DB Tools Set Parameter Value VI. You can specify the parameter index as either a numeral as shown above or you can put the parameter name as defined in the parameters array. After all the parameters are defined, the DB Tools Execute Query VI runs the parameterized query. The recordset and command references are freed with the DB Tools Free Object VIs and the database connection is closed.

The stored procedure examples shown in this section are specifically written for SQL Server. Oracle uses PL/SQL to create stored procedures. Although the syntax for PL/SQL is different, you still can create and run stored procedures for Oracle using the Database Connectivity Toolset. Refer to the National Instruments support services for more information about the stored procedures in Oracle and other topics related to the Database Connectivity Toolset for LabVIEW.

6

Building Applications

This chapter describes how you can build applications, executables or shared libraries, that use the LabVIEW Database Connectivity Toolset. The LabVIEW Database Connectivity Toolset requires some additional options that are not part of a standard application build routine such as MDAC and any DSN or UDL files.

Using the Database Connectivity Toolset Build Script

An application build script is included with the Database Connectivity Toolset to help you create applications from programs you build using the Database Connectivity Toolset VIs. Select **Tools»Build Application or Shared Library (DLL)** to open Application Builder. Click the **Load** button to open the **Open** window as shown in Figure 6-1. Open `database application template.bld` in the `LabVIEW/database` directory.

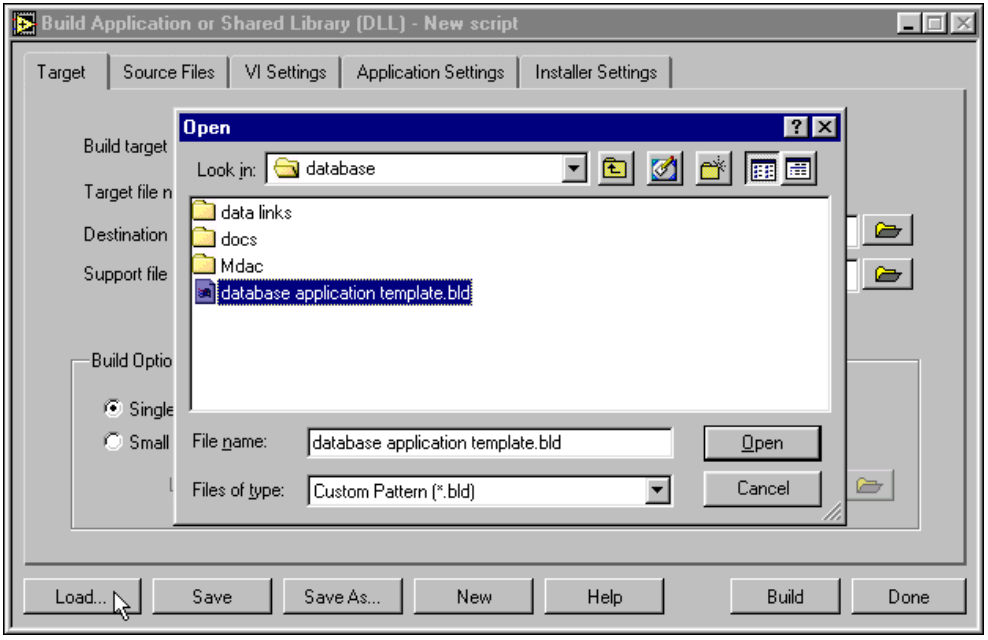

Figure 6-1. Loading a Build Script

After you load the build script, you can go through the usual process of selecting the target for the application (`.exe` or `.dll`), the directory structure, and the build options. Click the **Source Files** tab to add your top-level VI, shown in Figure 6-2. Notice that the project includes several files that are used to install the Microsoft Data Access Components (MDAC).

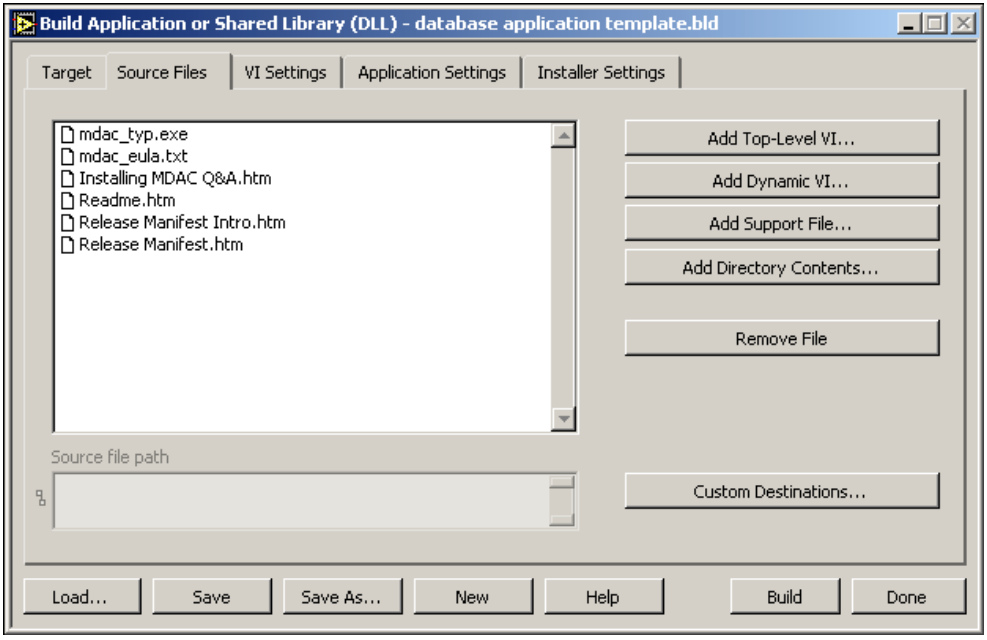

Figure 6-2. Including MDAC Source Files with Your Application

Note Include any necessary file DSNs or UDLs here that are associated with your application in the Source Files tab.

Installing MDAC

You can have your application install MDAC 2.5 (ADO, OLE DB, and ODBC) on the target machine by including the following supporting files, shown in Figure 6-2.

- `mdac_typ.exe`—This is the MDAC 2.5 installer.
- `mdac_eula.txt`—This is the End User License Agreement for MDAC. Read this document before building your application because it contains important terms and conditions.

- `Installing MDAC Q&A.htm`—This document contains the answers to frequently asked questions about installing/uninstalling MDAC and other related issues. This document is not required for building an application, but it contains useful information your users might need if anything goes wrong with the MDAC installation on the target computer.
- `Readme.htm`—This document is the readme for the MDAC 2.5 release. It contains known issues and current information about MDAC 2.5 and related technologies.
- `Release Manifest Intro.htm`—This document is the introduction for the release notes of MDAC. It gives a listing of the contents of the Release Manifest.
- `Release Manifest.htm`—This document describes the features, the list of installed files, and a complete list of known issues for MDAC 2.5.

You need to install the `mdac_typ.exe` file with the source files and MDAC with some command line arguments as shown in Figure 6-3. Select the **Installer Settings** tab in the Application Builder. Click the **Advanced** button to open the **Advanced Installer Settings** dialog box.

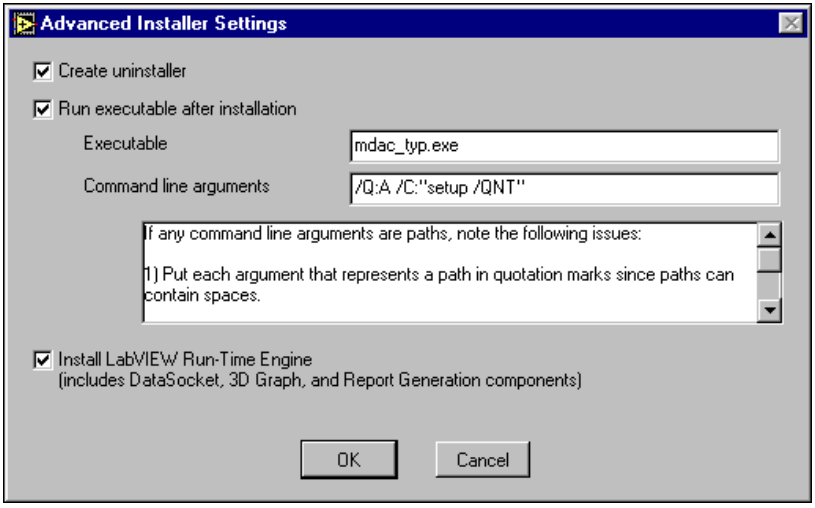

Figure 6-3. Running the MDAC Installer with Options

The /Q:A /C:"setup /QNT" command line prompt runs the MDAC setup in unattended mode, no visible user interface, and is not automatically rebooted. Refer to the Microsoft web site at http://msdn.microsoft.com/library/psdk/dasdk/mdac2lyb.htm for more options for the MDAC installation.

MDAC 2.6

The current version of MDAC at the time of the Database Connectivity Toolset release is version 2.6. This version is fully compatible with the Database Connectivity Toolset. However, the Database Connectivity Toolset ships with MDAC 2.5 because it does not yet support any of the new features of MDAC 2.6 and more localized versions of MDAC 2.5 are available.

Using Non-English Versions of Windows

The English version of MDAC 2.5 is included with the Database Connectivity Toolset and with the build script. However, do not include this mdac_typ.exe file with your built application if you use a non-English version of Windows. Microsoft provides localized versions of MDAC 2.5 for many different languages. Download the appropriate file(s) from the Microsoft web site at http://www.microsoft.com/data/download.htm#25info

Using Data Links and DSNs

You need to include the UDL and DSN files for the database connection with your application. User and System DSNs are only applicable to a particular user or computer and you must create them manually on the target machine using the ODBC Administrator. Figure 6-4 shows how you can include a data link with your application on the **Source Files** page of the Application Builder.

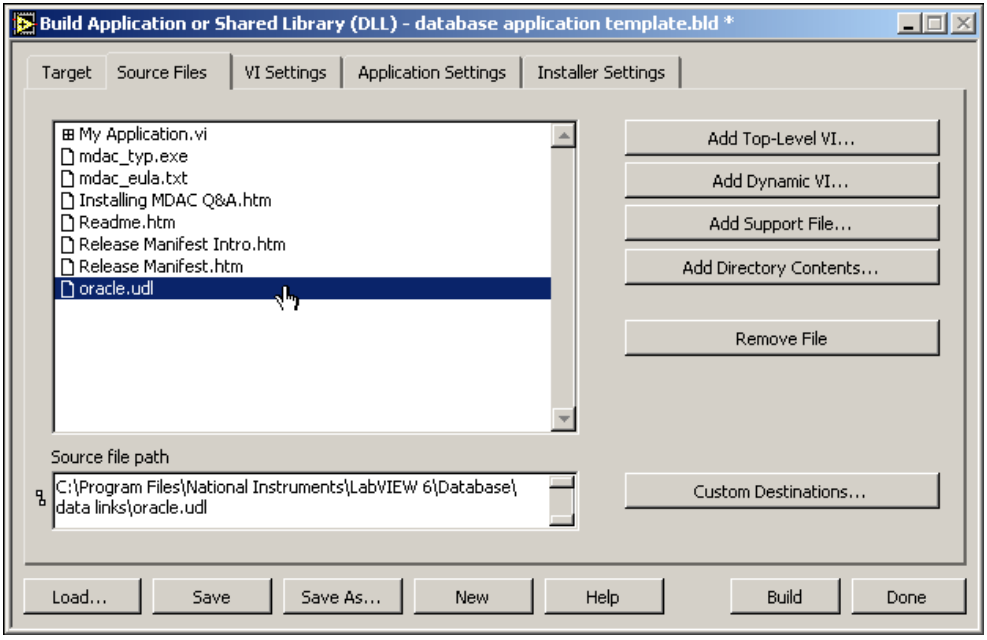

Figure 6-4. Including Data Links with Your Application

You cannot create data links by right-clicking in a Windows Explorer window and selecting **New»Microsoft Data Link**. The installer for the Database Connectivity Toolset creates data links by registering UDL files in the Windows registry. If you want this capability, use the Register UDL In Explorer example VI, shown in Figure 6-5.

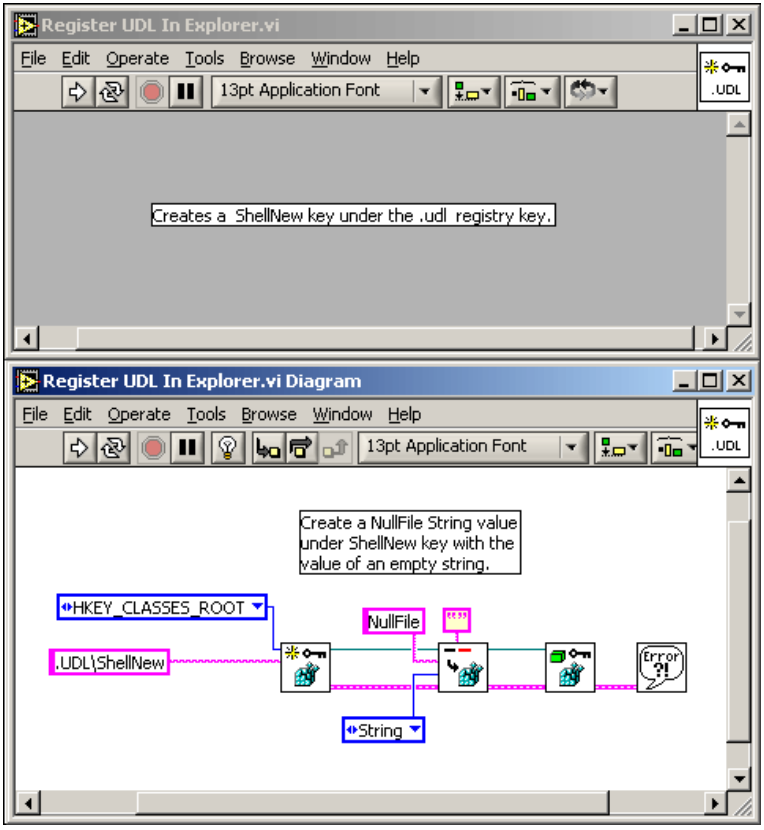

Figure 6-5. Registering a UDL in a VI

The Register UDL In Explorer example VI creates a registry key called `UDL\ShellNew` in the `HKEY_CLASSES_ROOT` entry that has an empty string value named `NullFile`. You can run this VI once after you install your application on a target machine. You must reboot the computer for this registry change to take effect.

Note You must have the Windows Folder Options set to show file extensions in order to use the **New»Microsoft Data Link** capability. Another option would be creating a new file, giving it a `.udl` extension, and bypassing the message about file associates. The icon should change to the UDL icon and you now can configure it using the **Data Link Properties** window.

A

SQL Quick-Reference

This appendix lists and briefly explains the more common SQL commands, operators, and functions available for databases. Refer to the the database vendor documentation for information about specific SQL implementations. Refer to Appendix B, *References*, for information about general-purpose and tutorial texts on SQL.

This appendix includes a series of tables that describe SQL commands, objects, clauses, and operators. Words that appear as all capital letters are SQL keywords. Items having parentheses ( ) require the parentheses in the SQL statement. Items enclosed by square brackets [ ] are optional. Items enclosed by curly brackets { } refer to items in other tables in this appendix. The vertical bar | means or.

SQL Commands

Table A-1 lists SQL commands you can use with the SQL Toolkit VIs.

Table A-1. SQL Commands

SQL Command	Syntax	Description and Examples
`CREATE TABLE`	`CREATE TABLE table_defn (column_defn, column_defn, ...)`	CREATE is used to generate and define new database tables. `CREATE TABLE tab1 (col1 NUMBER (6,2), col2 CHAR(12) NOT NULL, col3 DATE)`
`DELETE`	`DELETE FROM table_defn [WHERE where_clause]`	DELETE removes rows from a database table. A WHERE clause is used to select specific rows to delete. `DELETE FROM tab1 WHERE col1 >= 12345`
`DROP TABLE`	`DROP TABLE table_defn`	DROP is used to remove database tables. `DROP TABLE tab1`

Table A-1. SQL Commands (Continued)

SQL Command	Syntax	Description and Examples
INSERT INTO	INSERT INTO table_defn [options] [(col_name, col_name,...)] VALUES (expr, expr,...)	INSERT creates a new record in a database table and places data values into its columns. Column data values are specified in a VALUES clause. Options are specific to individual databases. INSERT INTO tab1 (col1, col2, col3) VALUES (1, 'abcd', {2/21/93})
SELECT	SELECT [DISTINCT] {* | col_expr, col_expr,...} FROM {from_clause} [WHERE {where_clause}] [GROUP BY {group_clause,...}] [HAVING {having_clause,...}] [ORDER BY {order_clause,...}] [FOR UPDATE OF {col_expr,...}]	SELECT is used to query specified columns FROM database tables. A WHERE clause is used to restrict the selection; and ORDER BY and GROUP BY clauses are used to organize the resulting data. SELECT col1, col2, col3 FROM tab1 WHERE col1 >= (col3 * col2) ORDER BY col3 ASC
UPDATE	UPDATE table_defn [options] SET col_name = expr, ... [WHERE where_clause]	UPDATE is used to SET columns in existing rows to new values. A WHERE clause is used to restrict which rows to update. Options are database specific. UPDATE tab1 SET col1 = (col1 * 1.5) WHERE col1 <1000

SQL Object Definitions

Table A-2 lists and defines SQL objects, which are the building blocks for all SQL statements.

Table A-2. SQL Objects

Object Abbreviation	Object Name	Description and Examples
table_defn	Table Definition	Describes a table on which to perform an operation. It might be simply a table name, or might include a full path specification (file-based databases only). modtest C:\qcdata\modtest
col_name	Column Name	Used to refer to columns in tables. Column name restrictions are imposed by some databases. Salary
col_expr	Column Expression	Used to specify a single column name or a complex combination of column names, operators, and functions. SerNo (highlimit-lowlimit) AVG(Salary)
sort_expr	Sort Expression	any column expression
data_type	Data Type	Specifies a column's data type. CHAR(30)
constraint	Constraint	Specifies a constraint on the contents of a column. NOT NULL
column_defn	Column Definition	Used in CREATE TABLE to describe a column to create in a new table. It consists of col_name, data_type, and (optional) constraint. SerNo CHAR(30) NOT NULL
char_expr	Character Expression	Any expression that results in a character datatype.
date_expr	Date Expression	Any expression that results in a date datatype.

Table A-2. SQL Objects (Continued)

Object Abbreviation	Object Name	Description and Examples
`number_expr`	Number Expression	Any expression that results in a number datatype.
`logical_expr`	Logical Expression	Any expression that results in a logical datatype.
`expr`	Expression	Any expression containing objects, operators, and/or functions. Character strings must be enclosed in single quotes. Date strings must be enclosed in curly brackets.

SQL Clauses

Table A-3 lists and defines SQL clauses, which are used to fully specify SQL commands.

Table A-3. SQL Clauses

Clause Name and Syntax	Applicable Commands	Description and Examples

FROM
table_defn [options]
[table_alias]
	SELECT
	DELETE
		Describes a table to perform an operation on. It might be just a table name or might include a full path specification (file-based databases only). Options are database specific.
		`SELECT FROM modtest`
		Table alias is used to specify a column name prefix to be used in subsequent clauses.
		`SELECT FROM C:\qcdata\modtes\MT`

WHERE expr1 comparison_oper expr2 [logical_oper expr3 comparison_oper expr4]...	SELECT DELETE UPDATE	Specifies conditions that are applied to each row in the table to determine an active set of rows. expr1 and expr2 are any valid expressions. comparison_oper is any comparison operator. Logical operators can be used to connect multiple conditions. `SELECT * FROM C:\qcdata\modtes\MT WHERE (MT.fld1 = 3 AND MT.fld2 <= 365)` **Note:** Use of table alias MT.
GROUP BY col_expr{, col_expr,...}	SELECT	Specifies one or more column expressions to use to group active set rows. `SELECT * FROM C:\qcdata\modtest\MT WHERE (MT.fld1 = 3 AND MT.fld2 <= 365) GROUP BY MT.fld4`

Table A-3. SQL Clauses (Continued)

Clause Name and Syntax	Applicable Commands	Description and Examples
HAVING expr1 comparison_oper expr2	SELECT used with GROUP BY	Specifies conditions to apply to active set row groups. GROUP BY must be specified first. SELECT * FROM C:\qcdata\modtest\MT WHERE MT.fld1 = 3 AND MT.fld2 <= 365 GROUP BY MT.fld4 HAVING AVG(MT.fld5) >= 2344.56
ORDER BY {sort_expr [DESC or ASC]}...	SELECT	Use to specify row order in the active set of rows and/or groups. DESC is descending order. ASC is ascending order. SELECT * FROM C:\qcdata\modtest\MT WHERE (MT.fld1 = 3 AND MT.fld2 <= 365) GROUP BY MT.fld4 HAVING AVG(MT.fld5) >= 2344.56 ORDER BY MT.fld2 DESC
FOR UPDATE OF col_name1[,col_name2,...]	SELECT	Use to lock columns in selected rows for updates or deletion. SELECT * FROM C:\qcdata\modtest\MT WHERE (MT.fld1 = 3 AND MT.fld2 <= 365) FOR UPDATE OF MT.fld1, MT.fld3

SQL Operators

Table A-4 lists SQL expressions and operators.

Table A-4. SQL Operators

Operator Class	Operators	Description	Example
Constants		numeric constant	1234, 1234.5678
	' ' " "	character constant	'abcd', "abcd"
	{}	date – time constant	{4/17/61}, {14:32:56}
	.T. .F.	logical constant	.T., .F.
Numeric	( )	operator precedence	(A + B) * (C – D)
	+ –	sign	– A
	* /	multiply/divide	A * B, A / B
	+ –	add/subtract	A + B, A – B
	** ^	exponentiation	A**B, A^B
Character	+	concatenate, keep trailing blanks	'txt a '+'txt b' (gives 'txt a txt b')
	–	concatenate, move trailing blanks to end	'txt a '–'txt b' (gives 'txt atxt b ')
Comparison (true / false)	=	equal	WHERE a = b
	<>	not equal	WHERE a <> b
	>=	greater than or equal	WHERE a >= b
	<=	less than or equal	WHERE a <= b
	IN	contained in the set ()	WHERE a IN ('apple','orange') WHERE a IN (SELECT...)
	NOT IN	not contained in the set ()	WHERE a NOT IN ('peach','pear')

Table A-4. SQL Operators (Continued)

Operator Class	Operators	Description	Example
Comparison (true / false) (continued)	ANY, ALL	compare with a list of rows	WHERE a = ANY (SELECT ...)
	BETWEEN	within a range of values	WHERE c BETWEEN a AND e
	EXISTS	existence of at least one row	WHERE EXISTS (SELECT ...)
	[NOT] LIKE	character pattern match	WHERE a LIKE 'tar%'
	[NOT] NULL	empty (no value)	WHERE a NOT NULL
Date	+ –	add/subtract	testdate+5 (result is a new date) testdate – {1/30/18} (result is a number of days)
Logical	( )	precedence	WHERE (a AND b) OR (c AND d)
	NOT	complement of operand	WHERE NOT (a IN (SELECT ...)
	AND	true if both operands are true	WHERE a = 1 AND b <= 1000
	OR	true if either operand is true	WHERE a = 1 OR b <= 1000
Set	UNION	set of all rows from all individual distinct queries	SELECT ... UNION SELECT...
Other	*	all columns	SELECT * FROM tab1

SQL Functions

Table A-5 lists SQL functions.

Table A-5. SQL Functions

Function	Description
ROUND(number_expr1,number_expr2)	number_expr1 rounded to number_expr2 decimal places
CHR(number_expr)	character having ASCII value number_expr
LOWER(char_expr)	force all to lower case in char_expr
LTRIM(char_expr)	remove leading blanks from char_expr
RTRIM(char_expr)	remove trailing blanks from char_expr
TRIM(char_expr)	remove trailing blanks from char_expr
SUBSTR(char_expr, number_expr1,number_expr2)	substring of char starting at character number number_expr1 of length number_expr2
UPPER (char_expr)	force all letters in char_expr to upper case
LEFT(char_expr)	leftmost character in char_expr
RIGHT(char_expr)	rightmost character in char_expr
SPACE(number_expr)	generate a string of number_expr blanks
IIF(logical_expr,True_Value, False_Value)	returns True_Value if logical_expr is true, returns False_Value if logical_expr is false
STR(number_expr,width[,precision])	converts number_expr to a character string of width and optional precision fractional digits
STRVAL(expr)	converts any expr to a character string
TIME()	returns current time of day as character string
LEN(char_expr)	number of characters in char_expr
AVG(numeric_column_name)	average of all non-NULL values in numeric_column_name
COUNT(*)	number of all rows in a table
MAX(column_expr)	maximum value of column_expr

Table A-5. SQL Functions (Continued)

Function	Description
MAX(number_expr1,number_expr2)	larger of number_expr1 and number_expr2
MIN(column_expr)	minimum value of column_expr
MIN(number_expr1,number_expr2)	smaller of number_expr1 and number_expr2
SUM(column_expr)	sum of values in column_expr
DTOC(date_expr, fmt_value[, 'separator_char'])	convert date_expr from date to character string using fmt template and (optional) separator character (default is '/'). fmt_values are as follows: 0 => MM/DD/YY 1 => DD/MM/YY 2 => YY/MM/DD 10 => MM/DD/YYYY 11 => DD/MM/YYYY 12 => YYYY/MM/DD
DTOS(date_expr)	convert from date_expr to character string using format YYYYMMDD
USERNAME()	returns name of current user as character string (not supported by all databases)
MOD(number_expr1,number_expr2)	divides number_expr1 by number_expr2 and returns remainder
MONTH(date_expr)	returns month part of date_expr as a number
DAY(date_expr)	returns day part of date_expr as a number
YEAR(date_expr)	returns year part of date_expr as a number
POWER(number_expr1, number_expr2)	raises number_expr1 to number_expr2 power
INT(number_expr)	returns integer part of number_expr
NUMVAL(char_expr)	converts char_expr to a number (if valid expr)
VAL(char_expr)	converts char_expr to a number

Table A-5. SQL Functions (Continued)

Function	Description
`DATE()`	returns current date in date format
`TODAY()`	returns current date in date format
`DATEVAL(char_expr)`	converts char_expr to date format
`CTOD(char_expr, fmt)`	converts char_expr to date format using fmt template 0 => MM/DD/YY 1 => DD/MM/YY 2 => YY/MM/DD

B

References

This appendix presents a categorized reference list for further reading on the ADO standard, databases and database design, the ODBC standard, the Structured Query Language, and related topics. Refer to the Microsoft Universal Data Access web site at `http://www.microsoft.com/data/` for more information about the different standards and to access the latest and localized versions of MDAC and ADO.

ADO Standard and Applications

Professional ADO 2.5 Programming
David Sussman, et al ISBN 1-861002-75-0

Visual Basic Developer's Guide to ADO
Mike Gunderloy ISBN 0-7821-2556-5

ADO Examples and Best Practices
William R. Vaughn ISBN 1-893115-16-X

Effective ADO
Northwest Training Systems ISBN 0-9673353-1-0

Use ADO to Maximize Stored Procedures
Dan Fox Nov 1, 1999
SQL Server Magazine at `http://www.sqlmag.com/articles/`

Databases and Database Application Design

Access Database Design & Programming
Steven Roman ISBN 1-56592-626-9

Client Server SQL Applications
S. Khoshafian, et al ISBN 1-55860-147-3

DATABASE - A Primer
C. Date ISBN 0-201-11358-9

PC Magazine Guide to Client/Server Databases
J. Salemi ISBN 1-56276-070-X

Microsoft SQL Server 7.0
William Robinson ISBN 0-672-31663-3

Oracle8i for Dummies
Carol McCullough-Dieter ISBN 0-7645-0570-X

Visual Basic Oracle8 Programmer's Reference
Dov Trietsch ISBN 1-861-00178-9

Open Data Base Connectivity (ODBC) Standard

Microsoft ODBC 2.0 Programmer's Reference and SDK Guide
Microsoft Press ISBN 1-55615-658-8

SQL Language Instructional Publications

A Visual Intro to SQL
J. Trimble, et al ISBN 0-471-61684-2

The Practical SQL Handbook
J. Bowman et al ISBN 0-201-62623-3

A Guide to the SQL Standard
C. Date, et al ISBN 0-201-55822-X

Instant SQL Programming
Joe Celko ISBN 1-874416-50-8

SQL Fundamentals
John Patrick ISBN 0-13-096016-0

SQL for Dummies
Allen G. Taylor ISBN 0-7645-0415-0

C

Supported Data Types

This appendix contains several tables showing how the data types used by LabVIEW, SQL, ADO, Microsoft Access, Oracle, and SQL Server map to the Database Connectivity Toolset and to each other.

Table C-1 lists LabVIEW data types and the data types they correspond to in the Database Toolset.

Table C-1. LabVIEW and the Database Toolset Data Types

LabVIEW Data Type	Database Toolset Data Type
Number	Number
String/Path	Varchar
Array	Binary
Cluster	Binary
Boolean	Varchar
Enum	Number
Variant	Binary or not supported
Picture Control	Binary
WDT	Binary
Refnum	Not supported
I/O Channel	Varchar or binary
Complex Numbers	Not supported

Table C-2. SQL-92 Data Types

Category	SQL Data Type	Description
Exact Numerics	INTEGER	Precision depends on the specific SQL implementation; database developer cannot specify the precision.
	SMALLINT	Can be the same as or smaller representation as Integer.
	NUMERIC	You can specify both the precision and the scale. For example, Numeric (10,2) gives a maximum value of 99,999,999.99.
	DECIMAL	Similar to Numeric, but can hold larger values than defined if the specific implementation (OS) permits.
Approximate Numerics	—	For values so large that exactness isn't necessary, but a close approximation.
	REAL	Single-precision floating point number determined by the OS implementation of a SGL.
	DOUBLE PRECISION	Double-precision floating-point number determined by the OS implementation of a DBL.
	FLOAT	Allows you to specify precision such as FLOAT (5)
Character Strings	CHAR (x)	Fixed character data such as CHAR (16). Extra is filled with spaces.
	VARCHAR (x)	Varying character data. Does not pad with spaces.
Bit Strings	BIT (x)	Similar to CHAR. Defaults to 1 bit.
	BIT VARYING (x)	Similar to VARCHAR.

Table C-2. SQL-92 Data Types (Continued)

Category	SQL Data Type	Description
Datetimes	DATE	Length of 10 positions in the form: YYYY-MM-DD
	TIME (p)	Has the form: HH:MM:SS.SSS… specified by p
	TIMESTAMP	Length of 19 as 10 for date, a space, and 8 time
	TIME WITH TIME ZONE	Same as TIME with an offset (–12:59 to +13:00) to GMT.
	TIMESTAMP WITH TIME ZONE	Same as TIMESTAMP with an offset to GMT.
Intervals	year-month	Difference between two datetime values given in years and months.
	day-time	Difference between two datetime values given in days, hours, minutes, and seconds.
NULL	—	Undefined field.

Table C-3 lists SQL Server data types, their descriptions, and their size and range.

Table C-3. SQL Server Data Types

SQL Data Type	Description	Size and Range
binary (n)	Miscellaneous binary information, *n* bytes long	Fixed length of *n* bytes
bit	A single true or false value	At least one byte; up to eight bit columns can be combined into a byte by SQL Server
char (n)	Up to *n* letters, numbers, or punctuation characters	Fixed length of *n* bytes
(small) datetime	A date, time, or both	smalldatetime: 4 bytes, from January 1, 1900 to June 6, 2079, accurate to one min. datetime: 8 bytes, January 1, 1753 to December 31, 9999, accurate to +/– 0.002 seconds
decimal (p [,s])	An arbitrary precision floating point number allowing p significant digits with up to *s* to the right of the decimal	*p* significant digits with up to *s* digits to the right of the decimal point; requires 5 bytes if $p < 10$, 7 for *p* of 10 to 19, and 13 for *p* of 20 to 28
image	Large binary data	16 bytes; stores up to 2 GB of data outside the table
(small or tiny) int	Whole numbers	tinyint: one byte, 0–255 smallint: two bytes, –32768 to 32767 int: four bytes, +/– about 2 billion
(small) money	Monetary amounts	smallmoney: four bytes, +/– 200,000 money: eight bytes, +/– about 900 trillion
numeric (p [,s])	General number type (same as decimal)	Generally equivalent to decimal
real (n)	General floating point type with *n* bits of precision	where *n* is bits of precision; for *n* up to 24 (7 significant digits), 4 bytes; 8 bytes for $n > 24$ (15 significant digits)

Table C-3. SQL Server Data Types (Continued)

SQL Data Type	Description	Size and Range
text	Large text objects	16 bytes; stores up to 2GB of data outside the table
timestamp	Field automatically filled in with a time-based, unique counter value when a row is inserted or updated	8 bytes
varbinary (n)	Variable length binary field, up to *n* bytes	variable size up to 8000 bytes, plus four bytes of overhead; *n* is maximum bytes allowed
varchar (n)	Variable length text field, up to *n* characters	variable size up to 8000 characters; *n* is maximum bytes allowed
ntext	Unicode version of text	—
nchar	Unicode version of char	—
nvarchar	Unicode version of varchar	—
NULL	special non-value	—

Table C-4. ADO Data Types

AdArray (Does not apply to ADOX.)	0x2000	A flag value, always combined with another data type constant, that indicates an array of that other data type.
adEmpty	0	Specifies no value (DBTYPE_EMPTY).
AdSmallInt	2	Indicates a two-byte signed integer (DBTYPE_I2).
adInteger	3	Indicates a four-byte signed integer (DBTYPE_I4).
adSingle	4	Indicates a single-precision floating-point value (DBTYPE_R4).
adDouble	5	Indicates a double-precision floating-point value (DBTYPE_R8).
adCurrency	6	Indicates a currency value (DBTYPE_CY). Currency is a fixed-point number with four digits to the right of the decimal point. It is stored in an eight-byte signed integer scaled by 10,000.
adDate	7	Indicates a date value (DBTYPE_DATE). A date is stored as a double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day.
adBSTR	8	Indicates a null-terminated character string (Unicode) (DBTYPE_BSTR).
adIDispatch	9	Indicates a pointer to an **IDispatch** interface on a COM object (DBTYPE_IDISPATCH). **Note:** This data type is currently not supported by ADO. Usage might cause unpredictable results.
adError	10	Indicates a 32-bit error code (DBTYPE_ERROR).
adBoolean	11	Indicates a Boolean value (DBTYPE_BOOL).

Table C-4. ADO Data Types (Continued)

adVariant	12	Indicates an Automation **Variant** (DBTYPE_VARIANT). **Note:** This data type is currently not supported by ADO. Usage might cause unpredictable results.
adIUnknown	13	Indicates a pointer to an **IUnknown** interface on a COM object (DBTYPE_IUNKNOWN). **Note:** This data type is currently not supported by ADO. Usage might cause unpredictable results.
adDecimal	14	Indicates an exact numeric value with a fixed precision and scale (DBTYPE_DECIMAL).
adTinyInt	16	Indicates a one-byte signed integer (DBTYPE_I1).
adUnsignedTinyInt	17	Indicates a one-byte unsigned integer (DBTYPE_UI1).
adUnsignedSmallInt	18	Indicates a two-byte unsigned integer (DBTYPE_UI2).
adUnsignedInt	19	Indicates a four-byte unsigned integer (DBTYPE_UI4).
adBigInt	20	Indicates an eight-byte signed integer (DBTYPE_I8).
adUnsignedBigInt	21	Indicates an eight-byte unsigned integer (DBTYPE_UI8).
adFileTime	64	Indicates a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (DBTYPE_FILETIME).
adGUID	72	Indicates a globally unique identifier (GUID) (DBTYPE_GUID).
adBinary	128	Indicates a binary value (DBTYPE_BYTES).
adChar	129	Indicates a string value (DBTYPE_STR).
adWChar	130	Indicates a null-terminated Unicode character string (DBTYPE_WSTR).

Table C-4. ADO Data Types (Continued)

adNumeric	131	Indicates an exact numeric value with a fixed precision and scale (DBTYPE_NUMERIC).
adUserDefined	132	Indicates a user-defined variable (DBTYPE_UDT).
adDBDate	133	Indicates a date value (yyyymmdd) (DBTYPE_DBDATE).
adDBTime	134	Indicates a time value (hhmmss) (DBTYPE_DBTIME).
adDBTimeStamp	135	Indicates a date/time stamp (yyyymmddhhmmss plus a fraction in billionths) (DBTYPE_DBTIMESTAMP).
adChapter	136	Indicates a four-byte chapter value that identifies rows in a child rowset (DBTYPE_HCHAPTER).
adPropVariant	138	Indicates an Automation PROPVARIANT (DBTYPE_PROP_VARIANT).
adVarNumeric	139	Indicates a numeric value (**Parameter** object only).
adVarChar	200	Indicates a string value (**Parameter** object only).
adLongVarChar	201	Indicates a long string value (**Parameter** object only).
adVarWChar	202	Indicates a null-terminated Unicode character string (**Parameter** object only).
adLongVarWChar	203	Indicates a long null-terminated Unicode string value (**Parameter** object only).
adVarBinary	204	Indicates a binary value (**Parameter** object only).
adLongVarBinary	205	Indicates a long binary value (**Parameter** object only).

Table C-5. Oracle Data Types Mapped to ADO

Oracle Data Type	ADO Data Type	Fixed Width?
BLOB	128	False
LONG RAW	128	False
BFILE	128	False
RAW	128	False
CLOB	129	False
LONG	129	False
CHAR	129	True
DECIMAL	131	True
DOUBLE PRECISION	5	True
DATE	135	True
VARCHAR2	129	False
NUMBER	131	False

Table C-6. SQL Server Data Types Mapped to ADO

SQL Server Data Type	ADO Data Type	Fixed Width?
uniqueidentifier	72	True
ntext	130	False
nvarchar	130	False
sysname	130	False
nchar	130	True
bit	11	True
tinyint	17	True
tinyint identity	17	True
image	128	False

Table C-6. SQL Server Data Types Mapped to ADO (Continued)

SQL Server Data Type	ADO Data Type	Fixed Width?
varbinary	128	False
binary	128	True
timestamp	128	True
text	129	False
char	129	True
numeric	131	True
numeric() identity	131	True
decimal	131	True
money	6	True
smallmoney	6	True
decimal() identity	131	True
int	3	True
int identity	3	True
smallint	2	True
smallint identity	2	True
float	5	True
real	4	True
datetime	135	True
smalldatetime	135	True
varchar	129	False

Table C-7. Access Data Types Mapped to ADO

Access Data Type	ADO Data Type	Fixed Width?
GUID	72	True
LONGCHAR	130	False
VARCHAR	130	False
CHAR	130	True
BIT	11	True
BYTE	17	True
LONGBINARY	128	False
VARBINARY	128	False
BINARY	128	True
CURRENCY	6	True
INTEGER	3	True
COUNTER	3	True
SMALLINT	2	True
REAL	4	True
DOUBLE	5	True
DATETIME	135	True

Table C-8. Supported SQL Data Types by Database and ADO

SQL Data Types	ORACLE8	SQL Server 7.0	Access 2000
Integer	Yes adNumeric(131)	Yes adInteger(3)	Yes adInteger(3)
Float	Yes adDouble(5)	Yes adDouble(5)	Yes adDouble(5)
Single	Not supported	Not supported	Yes adSingle(4)

Table C-8. Supported SQL Data Types by Database and ADO (Continued)

SQL Data Types	ORACLE8	SQL Server 7.0	Access 2000
Real	Yes adDouble(5)	Yes adSingle(4)	Yes adSingle(4)
Double Precision	Yes adDouble(5)	Yes adDouble(5)	Not Supported— but Double works
Smallint	Yes adNumeric(131)	Yes adSmallint(2)	Yes adSmallint(2)
Datetime	Not supported	Yes adDBTimeStamp(135)	Yes adDBTimeStamp(135)
Date	Yes adDBTimeStamp(135)	Not supported	Yes adDBTimeStamp(135)
Binary(n)	Not supported	Yes adBinary(128) 8000 bytes max	Yes adBinary(128) 510 bytes max
Varbinary(n)	Not supported	Yes adVarBinary(204) 8000 bytes max	Yes adVarBinary(204) 510 bytes max
Char(n)	Yes adChar(129) 2000 bytes max	Yes adChar(129) 8000 bytes max	Yes adWChar(130) 255 UNICODE char
Varchar(n)	Yes adVarChar(200) 2000 bytes max	Yes adVarChar(200) 8000 bytes max	Yes adWVarChar(202) 255 UNICODE char
Varchar2(n)	Yes adVarChar(200) 4000 bytes max	—	—

D

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `ni.com`.

NI Developer Zone

The NI Developer Zone at `ni.com/zone` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `ni.com`.

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `ni.com`. Branch office Web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Glossary

A

ActiveX	Microsoft's Object Linking and Embedding (OLE) technology that allows components from one application to be embedded in another application. For example, placing an Excel table into a Word document. Microsoft first used the term OLE to refer to its COM-based architecture, then later dropped that designation in favor of ActiveX. Since both OLE and ActiveX are based on COM, the term COM is also used. As a result, any combination of the words COM, OLE and ActiveX followed by the words control, object and component may mean the same thing.
ActiveX Data Objects (ADO)	Microsoft API that is designed as the Microsoft standard for data access. It is a COM object and is available for all Microsoft programming languages and applications. ADO is the method used by the Database Connectivity Toolkit to communicate with databases.
Advanced Data TableGram (ADTG)	Proprietary Microsoft binary file format and one of the two file formats that are supported specifically by the Database Connectivity Toolset. ADTG has the advantage of being a compact binary format that results in much smaller files written faster than if you use XML format.
American National Standards Institute (ANSI)	Standards organization that is involved with adopting SQL as a standard and partly responsible for the fact that most major commercial relational database products support SQL to some degree.
Application Programming Interface (API)	Language and message format that an application program uses to communicate with the operating system or some other system or control program such as a DBMS or communications protocol. APIs are implemented by writing function calls in the program, which provide the linkage to the required subroutine for execution. Thus, an API implies that some program module is available in the computer to perform the operation or that it must be linked into the existing program to perform the tasks.

B

Beginning Of File (BOF)	Marker that points just before the first record in a database table.
Binary coded decimal (BCD)	Storage of numbers in which each decimal digit is converted into binary and is stored in a single character or byte. For example, a 12-digit number would take 12 bytes.
Binary Large Objects (BLOB)	Database field that holds any digitized information including text, images, audio and video. BLOB is a data type used by Oracle to store binary data. LabVIEW data types such as arrays, waveforms, and clusters are stored as BLOBs while using the Database Connectivity Toolset.
build script	File that is generated and used by the LabVIEW Application Builder that describes how to create an executable or shared library (DLL) from a VI. The name of the build script used by the Database Connectivity Toolset is database application `template.bld`, and it is in the database directory in your LabVIEW directory.

C

client-side cursor	Type of cursor that requires all data to reside on the client's, or user's, machine. A client-side cursor is slow and memory intensive for large databases.
Code Interface Node (CIN)	One method of incorporating C code into the LabVIEW environment. It consists of compiling a code resource. Many restrictions and rules apply when creating or using a CIN, so you should refer to the LabVIEW manuals.
command	One of the main ADO objects. The two major uses for a command object are to execute statements against an OLE DB connection and to retrieve a recordset based on an SQL query or stored procedure.
commit	Term used with database transactions. When you commit a transaction, it carries through all the operations within the transaction and saves the changes to the database.

compatibility VIs	Library of VIs that is provided with the Database Connectivity Toolset so users of the SQL Toolkit for G can convert their applications to the new methodology. These compatibility VIs are installed into the `LabVIEW\vi.lib\addons\_SQL` directory and have the same names and connectors as the VIs in the SQL Toolkit.
connection	One of the main ADO objects that represents an open connection to an OLE DB data source. A connection object contains methods for setting timeouts and maintaining information about the connection.
constraints	Used by SQL to define rules determining what values the table entries can have.
cursor	Placeholder that points to a specific record in a recordset.

D

Data Access Objects (DAO)	Pogramming interface for data access using Microsoft Access. DAO/Jet provides access to the Jet database, and DAO/ODBCDirect provides an interface to ODBC databases.
Data Control Language (DCL)	Component of SQL that protects databases from harm such as locking a database so users cannot modify data in the database in the middle of a transaction.
Data Definition Language (DDL)	Component of SQL that creates, modifies, or deletes database structures, or tables.
Data Manipulation Language (DML)	Component of SQL that operates on data within the database.
data source	A database or any other application or object that contains data you wish to access with the Database Connectivity Toolset.
Data Source Name (DSN)	Way to refer to a specific database. You specify a DSN with a unique name and by the ODBC driver that communicates with the physical database (local or remote). You create a DSN using the ODBC Administrator in Windows.

data types	Categories of data that reside in a data source. Data types greatly vary between different data sources and the Database Connectivity Toolset converts data between types in order to be compatible with LabVIEW, Oracle, Access, SQL Server, SQL, and other DBMS. Examples of data types include integers, floats, string, binary, BLOB, date/time, array, and variant.
database	Set of related files that is created and managed by a database management system (DBMS). Today, DBMSs can manage any form of data including text, images, sound and video. Database and file structures are always determined by the software.
Database Administrator (DBA)	Person(s) responsible for maintaining databases within a company and has all access privileges.
database management system (DBMS)	System that stores information in and retrieves information from databases. Examples of a DBMS are Oracle and SQL Server.
drop	Another term for delete. For example, when you drop a table, you are deleting that table from the database as well as destroying the structure of the table.
Dynamic Link Library (DLL)	Executable program module used in Windows that performs some function. DLLs are called by a running application and are loaded to provide additional functionality. LabVIEW has the capability to call and create DLLs.

E

End Of File (EOF)	Marker that points after the last record in a database table.
Extensible Markup Language (XML)	Text-based industry standard specification for describing data and one of the two file formats that are supported specifically by the Database Connectivity Toolset. XML is similar to HTML (HyperText Markup Language) except XML can use an unlimited set of tags to describe any type or structure of data.

F

Federal Information Processing Standard (FIPS)	Standards organization involved with adopting SQL as a standard and partly responsible for the fact that most major commercial relational database products support SQL to some degree.
fetching	Another name for retrieving data from a database table.
field	Another name for a column in a database table. A Field is also an ADO object that represents a single column of data in a recordset. The fields collection is the default property of the Recordset object, so you will not often see its name in the code.
File DSN	Type of DSN where the connection information resides in a common text file and can be used on any machine or any user.

I

Indexed Sequential Access Method (ISAM) databases	DBMS where data is stored sequentially, while maintaining an index of key fields to all the records in the file for direct access. Examples of ISAM databases include Paradox, dBase, Btrieve, Excel, and FoxPro. ISAM databases use the Microsoft Jet Engine as their API.
Inner join	Specific type of SQL query that combines the results of several tables into one recordset.
INSERT	Specific SQL query that is used to send data to a database table.
INTERSOLV	Third party vendor that provides ODBC and OLE DB drivers for many DBSMs. The Database Connectivity Toolset does not contain any Merant drivers but they have been tested and work with the new VIs. Merant's web site is at `http://www.merant.com`.
Invoke Node	LabVIEW function on the **Functions»Communication»ActiveX** palette that allows you to call ActiveX methods, or functions.
International Standards Organization (ISO)	Standards organization involved with adopting SQL as a standard and partly responsible for the fact that most major commercial relational database products support SQL to some degree.

ISG Navigator	Third party vendor that provides ODBC and OLE DB drivers for many DBSMs. ISG Navigator's web site is at `http://www.isgnavigator.com`.
isolation levels	Represent different locking strategies. The higher the isolation level, the more complex the locking strategy is and the better it is at preventing data inconsistencies. ADO, and the Database Connectivity Toolset, support five isolation levels—Chaos (lowest level), Read Uncommitted, Read Committed, Repeatable Read, and Serializable (highest level).

L

locking	Method of protecting data that is used in multi-user database systems where different users have access to the same data at the same time.

M

MDB files	Microsoft Access databases.
Microsoft Component Object Model (COM)	Component software architecture from Microsoft, which defines a structure for building program routines (objects) that can be called and executed in a Windows environment. COM provides the interfaces between objects, and Distributed COM (DCOM) allows them to run remotely. COM objects can be small or large, can be written in several programming languages, and can perform any kind of processing.
Microsoft Data Access Components (MDAC)	Practical implementation of Microsoft's UDA strategy. The LabVIEW Database Connectivity Toolset version 1.0 includes MDAC 2.5 as part of its installation. MDAC 2.5 includes the ODBC, OLE DB, and ADO components. MDAC also installs several data providers you can use to open a connection to a specific data source such as an Access database.
Microsoft Jet database engine	Underlying DBMS of Microsoft Access databases (`.mdb`) and numerous Indexed Sequential Access Method (ISAM) databases including Paradox, dBase, Btrieve, Excel, and FoxPro. Visual Basic for Applications is the host language for the Jet DBMS.

N

non-relational database	Method of storing data where all of the information is stored in one large structure.
NULL value	Empty field containing no data in databases. This does not imply that it contains default data. It contains no data. LabVIEW has no concept of NULL, so NULLs are treated as default data for a particular data type such as an empty string, a zero-value numeric, a FALSE Boolean, or whatever the default value is for that data type.

O

ODBC Administrator	Windows Control Panel utility used to specify database connections. You use the ODBC Administrator to configure and create DSNs.
OLE DB	Common term for an OLE DataBase which is a COM programming interface for data access from Microsoft. It functions in a similar manner as ODBC, but for every type of data source not just SQL databases. Applications can also use OLE DB to access ODBC databases. OLE DB is a C++ API that allows for lower-level database access from a C++ compiler.
OLE DB Consumers	One of the three general types of COM components for OLE DB. Consumers are data-centric applications, components, or tools that use data through the OLE DB interfaces. Using networking terms, OLE DB consumers are the clients and the OLE DB data provider is the server.
OLE DB Data Providers	One of the three general types of COM components for OLE DB. Data providers are source specific software layers responsible for accessing and exposing data. Using networking terms, OLE DB consumers are the clients and the OLE DB data provider is the server.
OLE DB Service Providers	One of the three general types of COM components for OLE DB. Service Providers are optional components that implement standard services to extend the functionality of data providers. Examples of these services include cursor engines, query processors, and data conversion engines.
Online Analytical Processing (OLAP)	Decision support software that allows you to analyze information that has been summarized into multidimensional views and hierarchies. For example, OLAP tools are used to perform trend analysis on sales and financial information.

Open Database Connectivity (ODBC)	Database API from Microsoft that provides a common language for Windows applications to access databases on a network. ODBC is made up of the function calls programmers write into their applications and the ODBC drivers themselves. For client/server database systems such as Oracle and SQL Server, the ODBC driver provides links to their database engines to access the database. For desktop database systems such as Access, dBASE and FoxPro, the ODBC drivers actually manipulate the data. ODBC supports SQL and non-SQL databases. Although the application always uses SQL to communicate with ODBC, ODBC will communicate with non-SQL databases in its native language.
Oracle Call Interface (OCI)	Oracle's Native API for communicating between an application and the Oracle DBMS.

P

Parameter	Variable within an SQL query. A Parameter is also an ADO object that represents a single parameter for a Command object. Generally, parameters are used with any type of parameterized commands where an action is defined once but can have results changed depending on the variable values.
persistence	Another term used to describe writing database records to file. Not only is the data sent to the file, but also the structure and properties of the database recordset.
polymorphic	Term used in LabVIEW to mean a VI or function can accept different data types for a single input. For example, the DB Tools Open Connection is polymorphic because it can accept either a path or a string for the connection information input.
primary key	Field or combination of fields in a database table that uniquely identifies each record in the table.
property	Attribute or quality of an object. Property is also the ADO object which is the building block of the other ADO objects. The properties collection contains only the properties added to the object by the data provider and does not contain the intrinsic properties of the object.
property node	LabVIEW function on the **Functions»Communication»ActiveX** palette that allows you to set or read the properties of an ActiveX object.

Q

query	Another name for an SQL statement or command. All SQL strings are called queries whether they ask for data or not.

R

record	Another name for a single row or entry in a database table. A record is also an ADO object that works together with the Stream and Recordset objects to help you navigate through data.
recordset	Collection of records. Recordset is also the name of one of the main ADO objects and it represents a set of records and is used to manipulate data in a data source. You also can control cursors and the locking types for recordsets.
references	As used by the Database Connectivity Toolset, refernces are input and output parameters that specify connections to ADO objects. The four references used in the Database Connectivity Toolset are connection, recordset, command, and command-recordset references.
refnums	LabVIEW data type that refer to a specific connection. The Database Connectivity Toolset uses ActiveX ADO refnums as components of the references.
relational database	Method for storing data where the information is stored in multiple structures called tables. These structures are related to each other by one or more columns, or fields.
Relational Database Management system (RDBMS)	DBMS that uses relational databases.
rollback	Term used with database transactions. When you rollback a transaction, it returns the database to the state it was in before the transaction started.

S

schema	Overall organization for multiple tables in a database. A schema is also called a conceptual view or a complete logical view.
scrollability	Ability to move forwards and backwards within a recordset.

SELECT	Specific SQL query that is used to retrieve data from a database table.
server-side cursor	Type of cursor that only downloads to the client machine the specific record the cursor points to and keeps the remaining recordset data on the server.
SQL Access Group	Includes representatives of Microsoft, Tandem, Oracle, Informix, and Digital Equipment Corporations—developed the ODBC standard in September 1992 as a uniform method for applications to access databases. The standard consists of a multilevel API definition, a driver packaging standard, an SQL implementation based on ANSI SQL, and a means for defining and maintaining DSNs.
Stream	ADO object that represents binary data, usually stored in Unicode. The Stream object works together with the Record and Recordset objects to help you navigate through data.
Structured Query Language (SQL)	Language used to interrogate and process data in a relational database. Originally developed by IBM for its mainframes, all database systems designed for client/sever environments support SQL. SQL commands can be used to interactively work with a database or can be embedded within a programming language to interface to a database. SQL is a declarative data description and manipulation language rather than a procedural programming language. However, some programming extensions to SQL have turned it into a full-blown database programming language.
stored procedure	Precompiled set of SQL statements that perform a particular task, similar to a function or macro. Stored procedures are an advanced SQL technique that provides performance improvements and a way to share code.
System DSN	Type of DSN that is stored in the Windows registry key `HKEY_LOCAL_MACHINE/SOFTWARE/ODBC/ODBC.INI` and is available to all users of that machine.

T

table	Structure used to store data in a database. Tables consist of fields (columns) and records (rows).
transaction	Indivisible unit of work done with database tables. You can enclose many operations within a transaction so that all the operations are performed or none of them are.

U

Universal Data Access (UDA) — Current technology and platform defined by Microsoft where applications can exchange relational or non-relational data across intranets or the Internet—essentially connecting any type of data with any type of application. OLE DB is the Microsoft system-level programming interface of UDA and ADO is the API.

Universal Data Link (UDL) — File that defines a data source connection as defined by UDA. A UDL contains information about what OLE DB provider is used (the default is the Microsoft OLE DB provider for ODBC drivers), server information, user ID and password (if required), default database, and other related information.

User DSN — is a type of DSN that is stored in the Windows registry key `HKEY_CURRENT_USER/SOFTWARE/ODBC/ODBC.INI` and is available to a particular user of that machine.

V

variant — Data type used by ADO to return data from database tables. Variants work well in languages such as Visual Basic, which are not strongly typed. Because LabVIEW is strongly typed, you need to use the Database Variant To Data function located on the **Functions»Database** palette to convert the variant data to a LabVIEW data type before you can display the data in standard indicators such as graphs, charts, LEDs, etc.

X

XML — is the acronym for Extensible Markup Language.

Index

A

Access data types mapped to ADO (table), C-11
ActiveX Data Object standard. *See* ADO standard.
ADO object reference types (example), 5-9
ADO standard, 2-12 to 2-14
    ADO reference classes (table), 4-3
    components of ADO object model (table), 2-13 to 2-14
    data types
        Access data types mapped to ADO (table), C-11
        ADO data types (table), C-6 to C-8
        Oracle data types mapped to ADO (table), C-9
        SQL data types mapped to ADO (table), C-9
        SQL server data types mapped to ADO (table), C-10
        supported SQL data types by database and ADO (table), C-11 to C-12
    object hierarchy (figure), 2-13
    version 2.5 as basis for LabVIEW Database Connectivity Toolset, 2-14
ADTG (Advanced Data TableGram) format, 4-8
advanced database VIs, 5-1 to 5-15
    executing SQL statements and fetching data, 5-1 to 5-3
    navigating through database records, 5-3 to 5-5
        cursor types, 5-4 to 5-5
        moving through recordsets, 5-6 to 5-8
        using cursors, 5-3 to 5-4
    parameterized statements, 5-8 to 5-10
    stored procedures, 5-10 to 5-15
        creating, 5-11
        running with parameters, 5-13 to 5-15
        running without parameters, 5-12
application development, 6-1 to 6-7
    Application Builder, 6-1 to 6-2
    build script, 6-1 to 6-2
    data links and DSNs, 6-5 to 6-7
    MDAC
        installing, 6-2 to 6-4
        MDAC 2.6, 6-4
        using non-English versions of Windows, 6-4

B

BLOB (binary) data type, 1-2
Boolean data types, 3-22
building applications, 6-1 to 6-7
    Application Builder, 6-1 to 6-2
    application design references, B-1
    build script, 6-1 to 6-2
    data links and DSNs, 6-5 to 6-7
    MDAC
        installing, 6-2 to 6-4
        MDAC 2.6, 6-4
        using non-English versions of Windows, 6-4

C

Chaos isolation level, 4-7
clauses, SQL (table), A-5 to A-6
client-side cursor, 5-3 to 5-4
column information, retrieving, 4-1 to 4-2
COM interfaces. *See* Component Object Model (COM).
Command component, ADO object model (table), 2-13
Command reference class, ADO, 4-3

Command-Recordset reference class, ADO, 4-3
commands, SQL
common commands (table), 2-5
quick reference (table), A-1 to A-2
Component Object Model (COM)
OLE DB Consumers, 2-7
OLE DB Data Providers, 2-7
OLE DB Service Providers, 2-7
configuration
ODBC Administrator, 3-2 to 3-4
UDLs, 3-9 to 3-10
connecting to database, 3-1 to 3-11
DSNs
data source types, 3-1 to 3-2
examples of using, 3-4 to 3-6
ODBC administrator, 3-2 to 3-4
UDLs, 3-7 to 3-11
configuring, 3-9 to 3-10
creating, 3-7 to 3-8
example of using, 3-11
Connection component, ADO object model (table), 2-13
Connection reference class, ADO, 4-3
Connection tab, Data Link Properties dialog box, 3-10
conventions used in manual, *x*
CREATE TABLE command (table), 2-5, A-1
creating tables, 3-17 to 3-18
currency data types, 3-22
cursors
client-side cursor, 5-3 to 5-4
dynamic, 5-5
forward-only, 5-5
keyset, 5-5
server-side cursor, 5-4
static, 5-5
types of cursors, 5-4 to 5-5
using cursors, 5-3 to 5-4
customer education, D-1

D

Data Definition/Control Language (DDL/DCL) statements, 2-5
data links. *See* UDLs (Universal Data Links).
Data Links Properties dialog box (figure), 3-9
Data Manipulation Language (DML) statements, 2-5
Data Source Names (DSNs). *See* DSNs (Data Source Names).
data types
Access data types mapped to ADO (table), C-11
ADO (table), C-6 to C-8
automatic mapping of data types, 2-3
Boolean, 3-22
columns in tables, 2-3
currency, 3-22
date/time, 3-19
LabVIEW and DB Toolset data types (table), C-1
NULL values, 3-20 to 3-21
Oracle data types mapped to ADO (table), C-9
SQL
mapped to ADO (table), C-9
server types (table), C-4 to C-5
server types mapped to ADO (table), C-10
SQL-92 data types (table), C-2 to C-3
supported data types by database and ADO (table), C-11 to C-12
supported data types, 3-18 to 3-22, C-1 to C-12
database connections. *See* connecting to database.

database properties
ADO reference classes (table), 4-3
getting and setting, 4-2 to 4-4
specific properties, 4-4
utility VIs for getting and setting, 4-2 to 4-4
database transactions, 4-5 to 4-7
block diagram of transaction example (figure), 4-5
locking transactions, 4-6
setting isolation levels, 4-6 to 4-7
Database Variant To Data function, 5-3
databases
application design references, B-1
basic concepts, 2-1 to 2-3
reference list, B-1
date and time
date/time data types, 3-19
formatting, 4-4 to 4-5
DB Tools Close Connection VI
creating and deleting tables, 3-17
creating stored procedures, 5-11
using parameterized statements, 5-8
writing data, 3-12
DB Tools Create Parameterized Query VI
executing SQL statements, 5-1
running stored procedures with parameters, 5-13
using parameterized statements, 5-8, 5-9
DB Tools Create Table VI, 3-17, 3-23
DB Tools Database Transaction VI, 4-6
DB Tools Drop Table VI, 3-17, 3-18, 3-23
DB Tools Execute Query VI
creating stored procedures, 5-11
executing SQL statements, 5-1, 5-2
generating refnums, 4-8
opening tables and specifying cursor, 5-6
running stored procedures, 5-12
with parameters, 5-13, 5-14
using parameterized statements, 5-8
DB Tools Fetch Element Data VI, 5-1 to 5-3
DB Tools Fetch Next Recordset VI, 5-1
DB Tools Fetch Recordset Data VI
fetching data, 5-1, 5-2
reading and writing data files, 4-8
running stored procedures, 5-12
DB Tools Format Datetime Str VI, 3-19, 4-4
DB Tools Free Object VI
creating stored procedures, 5-11
fetching data, 5-2
reading and writing data files, 4-8
running stored procedures with parameters, 5-13
using parameterized statements, 5-8, 5-10
DB Tools Get Parameters VI, 5-14
DB Tools Get Properties VI, 4-2, 5-6, 5-8
DB Tools Insert Data VI, 3-12, 3-17
DB Tools List Columns VI, 4-1 to 4-2
DB Tools Load Recordset From File VI, 4-8, 4-9
DB Tools Move To Next Record VI, 5-6 to 5-8
DB Tools Move To Previous Record VI, 5-6 to 5-8
DB Tools Move To Record N VI, 5-6 to 5-8
DB Tools Open Connection VI
creating and deleting tables, 3-17
creating stored procedures, 5-11
examples
DSNs, 3-4 to 3-5
UDLs, 3-7, 3-11
using parameterized statements, 5-8
writing data to database, 3-12
DB Tools Save Recordset To File VI, 4-8
DB Tools Select Data VI, 3-15, 3-17
DB Tools Set Parameter Value VI, 5-9, 5-14
DB Tools Set Properties VI, 4-2
DDL/DCL (Data Definition/Control Language) statements, 2-5
DELETE command (table), 2-5, A-1

deleting tables, 3-17 to 3-18
dirty reads, preventing, 4-6
DML (Data Manipulation Language) statements, 2-5
documentation
  conventions used in manual, *ix*
  related documentation, *x*
DSNs (Data Source Names)
  building applications, 6-5 to 6-7
  data source types, 3-1 to 3-2
  examples of using, 3-4 to 3-6
  testing connections, 3-5 to 3-6
dynamic cursor, 5-5

E

Error component, ADO object model (table), 2-14
examples
  DSNs, 3-4 to 3-6
  LabVIEW Database Connectivity Toolset
    using with other databases, 3-22 to 3-23
    using without a database, 3-23
  UDLs, 3-11
Extensible Markup Language (XML), 4-8

F

fetching data, 5-1 to 5-3
Field component, ADO object model (table), 2-14
floating point numbers, 1-2
Format Into String function, 5-13
formatting date and time, 4-4 to 4-5
forward-only cursor, 5-5
functions, SQL (table), A-9 to A-11

H

high-level database VIs, 3-11 to 3-18
  creating and deleting tables, 3-17 to 3-18
  reading data from database, 3-13 to 3-15
  reading specific data from table, 3-16 to 3-17
  writing data to database, 3-12 to 3-13

I

INSERT command (table), 2-5, A-1
installing
  LabVIEW Database Connectivity Toolset, 1-2 to 1-3
  MDAC, 6-2 to 6-4
isolation levels
  preventing data inconsistencies, 4-6
  setting, 4-6 to 4-7

K

keyset cursor, 5-5

L

LabVIEW Database Connectivity Toolset
  ADO version 2.5 as basis of, 2-14
  background information, 2-3 to 2-6
  examples
    using with other databases, 3-22 to 3-23
    using without a database, 3-23
  installing, 1-2 to 1-3
  overview, 1-1 to 1-2
  upgrading from previous versions, 1-3 to 1-4
locking transactions, 4-6

M

manual. *See* documentation.
MDAC (Microsoft Data Access Components)
  installing, 6-2 to 6-4
  MDAC 2.6, 6-4
  overview, 2-6
  using non-English versions of Windows, 6-4
Microsoft ActiveX Data Object standard. *See* ADO standard.
Microsoft Component Object Model (COM). *See* Component Object Model (COM).
Microsoft Data Access Components. *See* MDAC (Microsoft Data Access Components).

N

navigating through database records, 5-3 to 5-5
  cursor types, 5-4 to 5-5
  moving through recordsets, 5-6 to 5-8
  using cursors, 5-3 to 5-4
NI Developer Zone, D-1
non-repeatable reads, preventing, 4-6
NULL values, handling, 3-20 to 3-21

O

object definitions, SQL (table), A-3 to A-4
ODBC Administrator, 3-2 to 3-4
  available ODBC drivers (figure), 3-3
  ODBC Access Driver Setup dialog box (figure), 3-4
  ODBC Data Source Administrator dialog box (figure), 3-2 to 3-4
ODBC standard
  background information, 2-3 to 2-4
  references, B-2
OLE DB providers
  Jet, 2-10 to 2-11
  ODBC, 2-8
  Oracle, 2-11 to 2-12
  SQL Server, 2-9
OLE DB standard
  OLE DB Consumers, 2-7
  OLE DB Data Providers, 2-7
  OLE DB Service Providers, 2-7
  overview, 2-7
Open Database Connectivity standard. *See* ODBC standard.
operators, SQL (table), A-7 to A-8
Oracle databases
  BLOB data types not supported by OLE DB Provider, 1-2
  OLE DB provider for Oracle, 2-11 to 2-12
  Oracle data types mapped to ADO (table), C-9
  PL/SQL, 5-15

P

Parameter component, ADO object model (table), 2-14
parameterized statements, 5-8 to 5-10
phantom reads, preventing, 4-6
PL/SQL (Oracle databases), 5-15
properties. *See* database properties.
Property component, ADO object model (table), 2-14
Provider tab, Data Link Properties dialog box, 3-9 to 3-10

R

Read Committed isolation level, 4-7
Read Uncommitted isolation level, 4-7
reading
    data files, 4-8 to 4-9
    data from database, 3-13 to 3-15
    specific data from database, 3-16 to 3-17
record, definition (note), 5-2
Record component, ADO object model (table), 2-14
Recordset component, ADO object model (table), 2-13
Recordset reference class, ADO, 4-3
reference numbers (refnums)
    generating with DB Tools Execute Query VI, 4-8
    not supported, 3-18 to 3-19
references
    ADO standard and applications, B-1
    databases and database application design, B-1
    ODBC standard, B-2
    SQL language instructional publications, B-2
Register UDL in Explorer example VI, 6-6
relational databases, 2-2
Repeatable Read isolation level, 4-7

S

SELECT command (table), 2-5, A-2
Serializable isolation level, 4-7
server-side cursor, 5-4
SQL (Structured Query Language)
    background information, 2-4 to 2-6
    classes of SQL statements, 2-5
    clauses (table), A-5 to A-6
    commands
        common SQL commands (table), 2-5
        quick reference (table), A-1 to A-2
    data types
        mapped to ADO (table), C-9
        server types (table), C-4 to C-5
        server types mapped to ADO (table), C-10
        SQL-92 data types (table), C-2 to C-3
        supported data types by database and ADO (table), C-11 to C-12
    executing statements with advanced database VIs, 5-1 to 5-3
    functions (table), A-9 to A-11
    instructional publications, B-2
    object definitions (table), A-3 to A-4
    operators (table), A-7 to A-8
    supported by standards organizations, 2-4 to 2-5
    typical statement (example), 2-6
    variants of, 2-6
SQL Toolkit, 2-3 to 2-4
static cursor, 5-5
stored procedures, 5-10 to 5-15
    creating, 5-11
    definition, 5-10
    maintainability benefits, 5-10 to 5-11
    performance benefits, 5-10
    running
        with parameters, 5-13 to 5-15
        without parameters, 5-12
    security benefits, 5-11
Stream component, ADO object model (table), 2-14
Structured Query Language. *See* SQL (Structured Query Language).
system integration, by National Instruments, D-1

T

tables
basic concepts, 2-1
creating, 3-17 to 3-18
data types, 2-3
deleting, 3-17 to 3-18
getting table and column information, 4-1 to 4-2
NULL values in table rows (note), 2-2
relational databases, 2-2
sample test sequence results (table), 2-2
technical support resources, D-1 to D-2
Test Data Source button (figure), 3-6
time
date/time data types, 3-19
formatting date and time, 4-4 to 4-5
transactions. *See* database transactions.

U

UDLs (Universal Data Links), 3-7 to 3-11
building applications, 6-5 to 6-7
configuring, 3-9 to 3-10
creating, 3-7 to 3-8
example of using, 3-11
registering in VI, 6-6
Universal Data Access (UDA), 2-6
UPDATE command (table), 2-5, A-2
upgrading from previous versions, 1-3 to 1-4
utility VIs, 4-1 to 4-9
formatting date and time, 4-4 to 4-5
getting and setting database properties, 4-2 to 4-4
getting table and column information, 4-1 to 4-2
performing database transactions, 4-5 to 4-7
writing and reading data files, 4-8 to 4-9

V

variant forms of SQL, 2-6
VIs
advanced database VIs, 5-1 to 5-15
executing SQL statements and fetching data, 5-1 to 5-3
navigating through database records, 5-3 to 5-5
parameterized statements, 5-8 to 5-10
stored procedures, 5-10 to 5-15
high-level database VIs, 3-11 to 3-18
creating and deleting tables, 3-17 to 3-18
reading data from database, 3-13 to 3-15
reading specific data from table, 3-16 to 3-17
writing data to database, 3-12 to 3-13
utility VIs, 4-1 to 4-9
formatting date and time, 4-4 to 4-5
getting and setting database properties, 4-2 to 4-4
getting table and column information, 4-1 to 4-2
performing database transactions, 4-5 to 4-7
writing and reading data files, 4-8 to 4-9

W

Web support from National Instruments, D-1
Worldwide technical support, D-2
writing
data files, 4-8 to 4-9
data to database, 3-12 to 3-13

X

XML (Extensible Markup Language), 4-8