

LabVIEW Database Connectivity Toolkit

The LabVIEW Database Connectivity Toolkit allows you to quickly connect to local and remote databases and perform many common database operations without having to know structured query language (SQL) programming. This toolkit greatly simplifies the process of making fast and repeated calls to popular databases, such as Microsoft Access, SQL Server, and Oracle. If you need advanced database functionality and flexibility, the Database Connectivity Toolkit also offers complete SQL capabilities.

There are numerous database formats available from a variety of software vendors. In many cases, you may need to save data to multiple different types of databases. The Database Connectivity Toolkit provides one consistent API for numerous databases to save you the time of learning different APIs. The toolkit can connect to any database with an ADO-compliant OLE DB provider or ODBC driver, including popular databases such as:

- Microsoft Access
- Microsoft SQL Server
- Oracle
- Visual FoxPro
- dBase
- Paradox

This tutorial is designed to briefly introduce you to some of the functionality included in the LabVIEW Database Connectivity Toolkit. Topics include connecting to a database, inserting data in to and selecting data from a database, and executing SQL queries. Feel free to explore other parts of the LabVIEW environment as you follow along – LabVIEW has numerous features to offer.

Database Connectivity Introduction	2
Database Terminology	2
The Database Programming Model	3
Connecting to a Database	7
Inserting Data in to a Database	13
Selecting Data from a Database.....	16
Executing SQL Queries	19
Summary	20

Database Connectivity Introduction

Database Terminology

SQL (Structured Query Language) is a set of character string commands that is a widely supported standard for database access. SQL statements allow you to use a common set of commands to access different databases. SQL statements can be used to store data into a database, query a database for records that match certain criteria, and many other database operations.

At its most basic level, the Database Connectivity Toolkit uses SQL statements to access, modify, and view information in databases. These statements are constructed in the toolkit's VIs and then executed through a Microsoft technology, OLE DB (Object Linking and Embedding Database), that provides a connection between LabVIEW and the database.

Inside LabVIEW, property and invoke Nodes make calls to the Microsoft ActiveX Data Objects (ADO). ADO will then connect to the database through either a Data Source Name or Universal Data Link.

OLE DB is comprised of a set of Microsoft Component Object Model (COM) interfaces that support various DBMS. For instance, OLE DB allows interaction with traditional DBMS such as Microsoft Access and other data storage systems such as Microsoft Excel. OLE DB is a C++ API that allows access to databases through a C++ compiler. OLE DB uses a provider (driver) to talk with the different DBMS.

With OLE DB you can communicate with any DBMS that supplies an ODBC driver or OLE DB provider. OLE DB uses the OLE DB Provider for ODBC as a conversion layer between OLE DB and ODBC if an ODBC driver is used to communicate with a database. However, there are native OLE DB Providers for different DBMS such as SQL Server, Jet (Microsoft Access), and Oracle. These native OLE DB providers are more efficient than the OLE DB Provider for ODBC because they remove the need to convert from OLE DB to ODBC and then to the DBMS.

OLE DB replaced ODBC as the underlying communication technology that allows the Database Connectivity Toolkit to interact with different databases. To change the database that the toolkit is working with, simply change the OLE DB Provider that is selected to communicate between LabVIEW and the DBMS.

The Database Programming Model

The Database Connectivity Toolkit has three “major” palettes of VIs, all located in the **All Functions » Database** palette.

Database VIs

The High Level VIs in the Database Connectivity Toolkit allow access to the database without the use of any SQL statements. There are VIs to open and close connections to the database, insert and delete tables, insert and select records, and convert variants. All data stored in a database is passed using the Variant data type in LabVIEW.

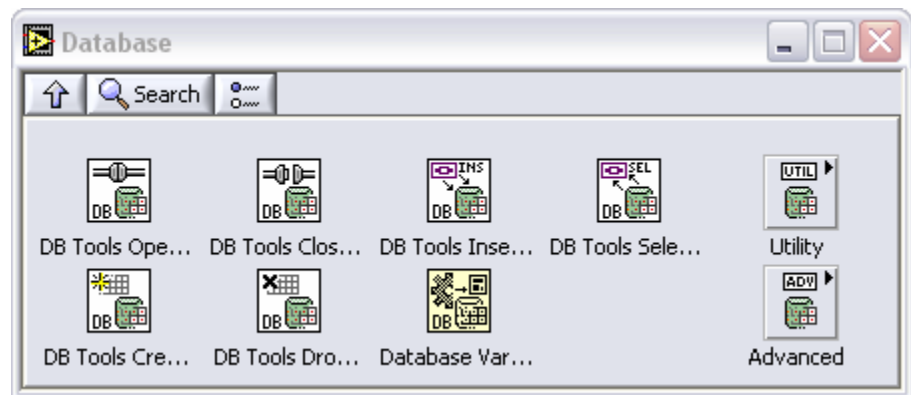


Figure 1.1 – High Level Database Operations

Advanced VIs

The Advanced Palette gives you more flexibility in searching, modifying, and viewing data from a database. The Advanced VIs allow you to execute SQL queries into databases. They also allow you to select and navigate through recordsets in the database. Furthermore, these VIs expose some of the more advanced features of the Database Connectivity Toolkit such as parameterized queries.

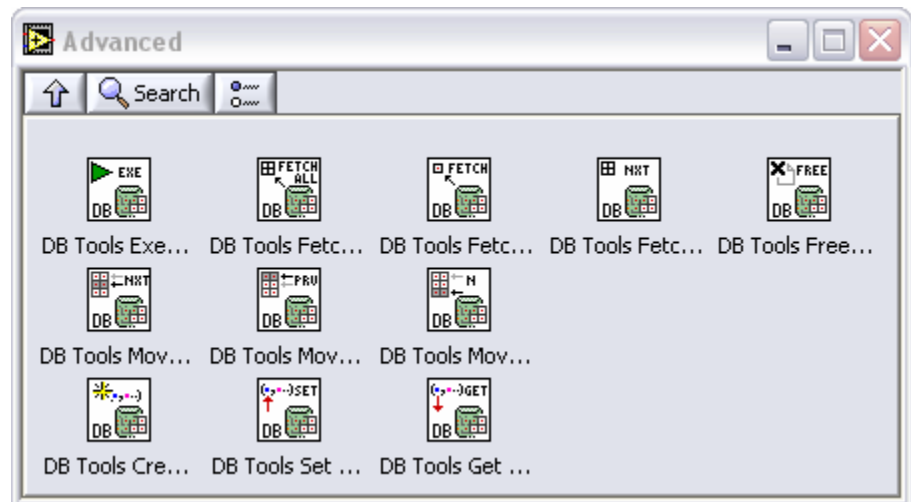


Figure 1.2 – Advanced Database Operations

Utility VIs

The Utility VIs are used to get and set connection and general information about the database. Using the Utility VIs you can view the names of the tables in a database, and the fields (columns) in a table. You can also view and set different connection parameters for the database such as connection string and connection timeout. In the Utility VIs, there are also VIs to store recordsets to file and load them back in to LabVIEW from a file.

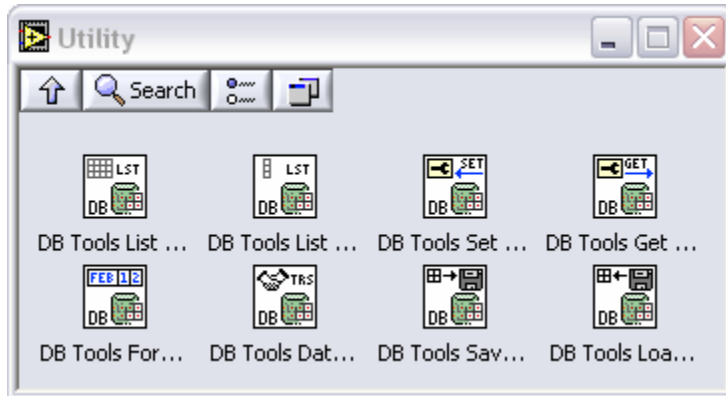


Figure 1.3 – Utility Database Operations

The Database Programming Model

Interacting with a database in LabVIEW is, at its most abstract layer, a three step process:

1. Establish a connection with a database. This connection is through a System DSN, File DSN, or UDL.
2. Perform operations on the database such as inserting records, updating records, and querying records.
3. Close the connection to the database and check for errors.

This basic programming model is shown in Figure 1.4. Each of the three steps are discussed briefly below. Additional information, along with hands-on exercises, is provided in the next few sections.

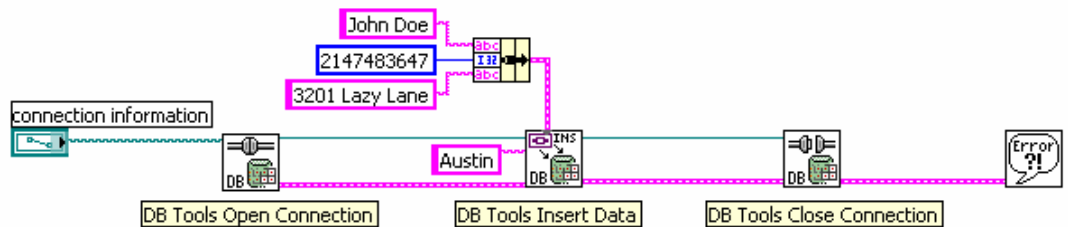


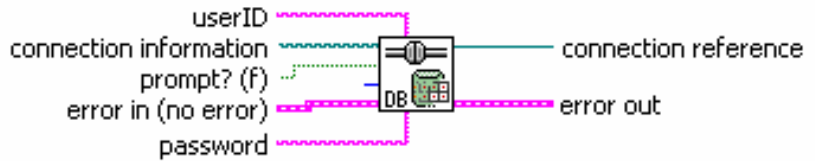
Figure 1.4 – Basic Database Programming Model

Step 1: Connect to Database

The first step in doing database operations with the Database Connectivity Toolkit is to connect to the database. The **DB Tools Open Connection** VI is used to open connections to the database. This VI determines the connection from the **connection information** terminal. This polymorphic

VI will accept a path to a File DSN or UDL, or the name of System or User DSNs. If a connection has not yet been established to a database, one can be created by setting the optional **prompt** terminal to **TRUE**.

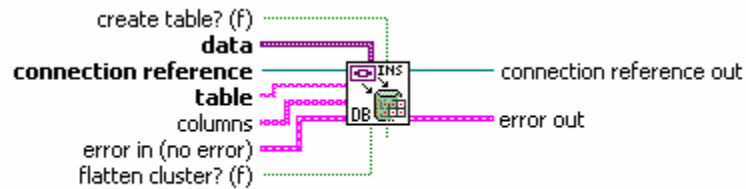
The **connection reference** output terminal is actually an ActiveX ADO refnum that must be used on subsequent database operations.



DB Tools Open Connection.vi

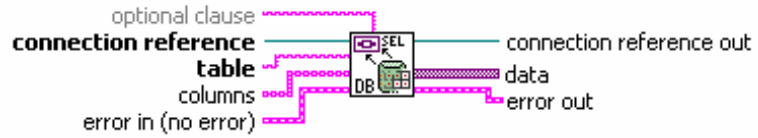
Step 2: Insert or Select Data

After the connection to the database is established, you can insert data into, or select data from, the database in a number of methods. The highest level inserts are done with the **DB Tools Insert Data** VI. To use this VI, a connection reference must be wired in. Also, the table used for storing the data must be specified. You can select to create the table if it does not already exist. You can also specify the field(s) the data is to be inserted into. The data itself is supplied in the **data** terminal. Each record to insert is bundled together as a cluster. An array of clusters should be created to insert more than one record into the database.



DB Tools Insert Data.vi

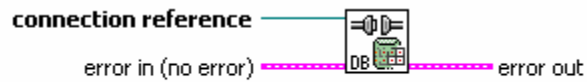
If you want to select data from the database, use the **DB Tools Select Data** VI to select records. To use this VI, you first need to wire in a **connection reference** and the name of the **table** that you want to extract the information from. You can also specify specific **columns** to extract. The **DB Tools Select Data** VI will return all of the fields from all of the records that are in the columns supplied and meet the optional clause. The **optional clause** is an SQL statement. If no SQL statement is wired into the **optional clause** it will extract the data from all of the records. If you want to extract all of the fields of all the records, then simply leave the **optional clause** and **columns** terminals empty. The data will be returned through the **data** terminal. It will be returned as a two-dimensional array of variants. Each row of the array is a different record and each column is a different field in the record. Use the **Database Variant To Data** VI to convert from the variant to the correct data type.



DB Tools Select Data.vi

Step 3: Close Connection

After you have completed all of the operations on the database, close the connection to the database. This operation is done with the **DB Tools Close Connection VI**. Once a connection to a database has been closed, no more operations can be done on that database from within LabVIEW until a new connection is established with a **DB Tools Open Connection VI**.



DB Tools Close Connection.vi

2

Connecting to a Database

The first step in doing database operations with the Database Connectivity Toolkit is to connect to the database. There are two major types of connections that can be created for LabVIEW to communicate with a database - Data Source Names (DSNs) and Universal Data Links (UDLs). DSNs are used to facilitate ODBC communication, and UDLs are used for OLE DB connections between the DBMS and LabVIEW.

ODBC DSN (Data Source Name)

A Data Source Name uses ODBC for communication between an application and the database. When creating a DSN you specify information such as the name of the data source (database), ODBC driver used for connection, and security information. DSNs are created and configured using the **ODBC Data Source Administrator (Control Panel » Administrative Tools » Data Sources(ODBC))**.

There are three types of DSN you can create:

- **System DSNs** are contained in the system registry and apply to all system users.
- **User DSNs** are stored in the registry and allow a single user to access a database.
- **File DSNs** are files (*.dsn) that allow anyone with access to the file to access the database.

UDL (Universal Data Link)

Universal Data Links use ADO for communication between an application and a database. UDLs contain information about the OLE DB provider that is used for communication, server information, user ID and password, and database. The default OLE DB provider for UDLs is the OLE DB provider for ODBC, but the native providers for the DBMS are more efficient.

You can create a UDL in three different ways:

1. From within LabVIEW by going to **Tools » Create Data Link**.
2. Using the DB Open Create Connection VI (**All Functions » Database**) which can provide a prompt to create a UDL.
3. From outside LabVIEW by right-clicking on the Desktop or in Windows Explorer and selecting **New » Microsoft Data Link**.

In the next exercise you will first create a blank Microsoft Access database, and then create two different connections to that database - a system DSN and a UDL.

You can complete the following exercise in approximately 10 minutes.

Part 1. Create a Database

In this section you will create a simple database that will be used for the rest of the parts of the exercise.

1. Open Microsoft Access (**Start » Programs » Microsoft Access**).
2. Create a new **Blank Database** from the **New File** menu (Figure 2.1), name it **Test Results**, and store it on the Desktop.

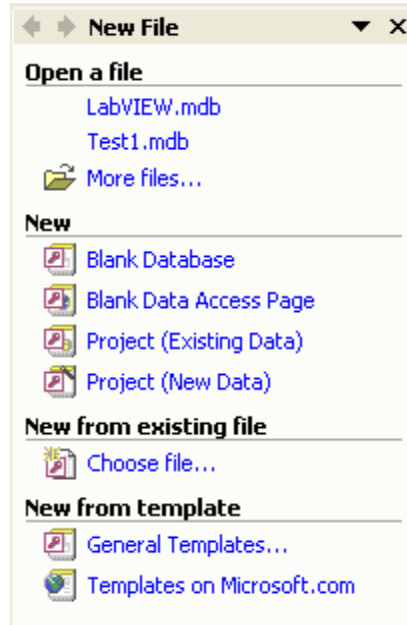


Figure 2.1 - New File Menu in Microsoft Access

3. Save the database and close Microsoft Access. You will add tables and data to the database using the Database Connectivity Toolkit in Exercise 2.

Part 2. Create a System DSN

In this section you will create a System DSN that will connect to the Test Results MS Access database.

1. Launch the ODBC Data Source Administrator (**Control Panel » Administrative Tools » Data Sources (ODBC)**).
2. Select the **System DSN** tab, shown in Figure 2.2. From this tab you can create and edit System DSNs on the system.

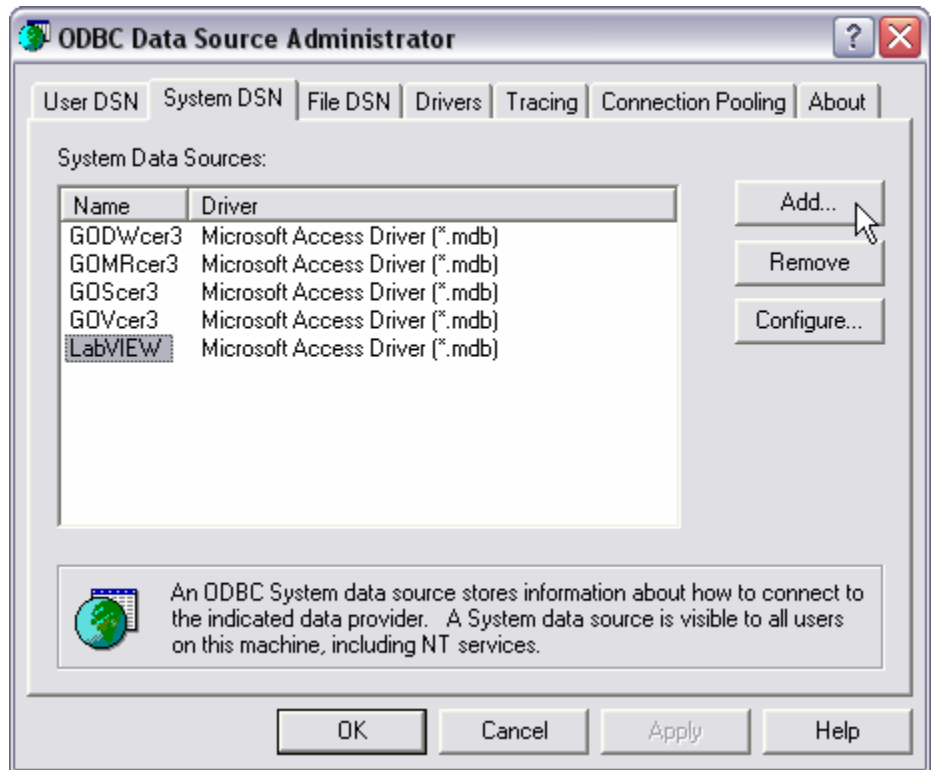


Figure 2.2 - ODBC Data Source Administrator

3. Select the **Add** Button to create a new System DSN.
4. The **Create New Data Source** window (Figure 2.3) allows you to select the ODBC driver that you will use to communicate with the database. Since your database is using Microsoft Access, select the **Microsoft Access Driver (*.mdb)**.

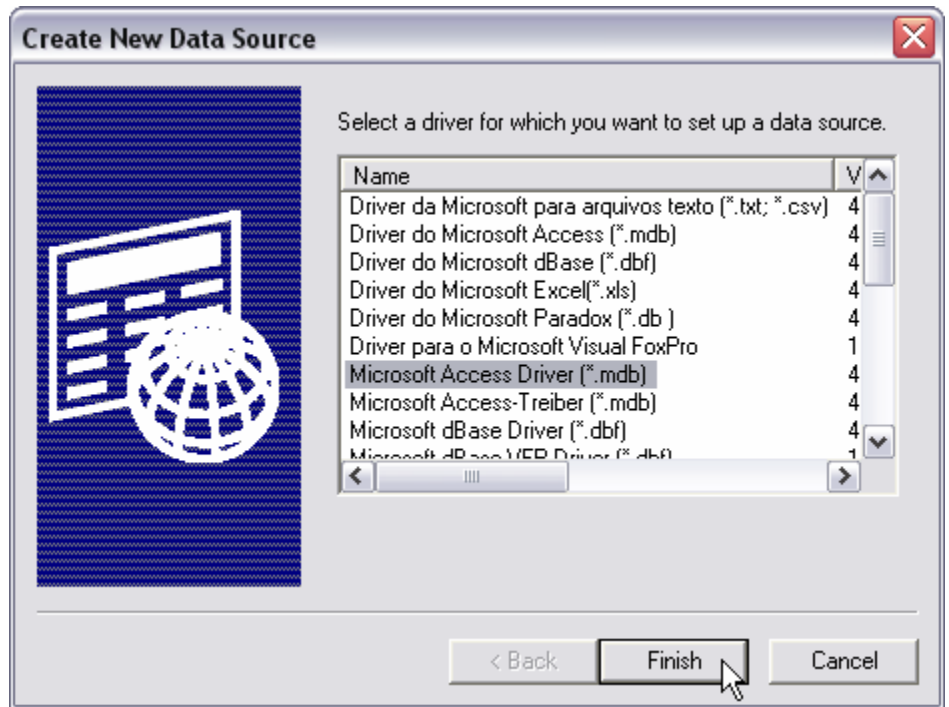


Figure 2.3 - Create New Data Source Window

5. The **ODBC Microsoft Access Setup** window is used to configure the DSN. Change the Data Source Name (DSN) to **TestResults** and write a short description about the DSN. Next, press **Select...** and navigate to the **Test Results.mdb** database on the Desktop that you created earlier. When you have completed these actions, the window should resemble Figure 2.4

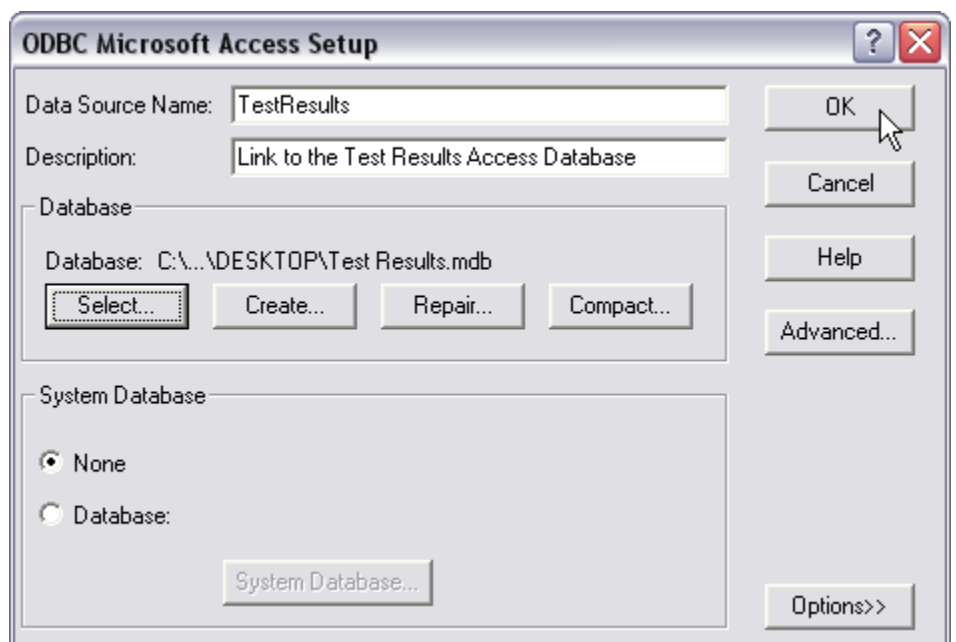


Figure 2.4 - ODBC Microsoft Access Setup Window

6. Select **OK** to finish creating the DSN.

Part 3. Create a UDL

In this step you will create the second type of database connection, a Universal Data Link. You can create a UDL in many different ways. In this exercise you will create a UDL from within LabVIEW.

1. Launch LabVIEW either from **Start »Programs»National Instruments»LabVIEW** or by double-clicking on the LabVIEW short-cut on the Desktop.
2. From a blank VI, select **Tools » Create Data Link** to launch the **Data Link Properties** window.
3. On the **Provider** tab, select the provider for the DBMS that you are communicating with. By default this is the Microsoft OLE DB Provider for ODBC Drivers. In this case, select the **Microsoft Jet 4.0 OLE DB Provider** to communicate with the Microsoft Access database. It is more efficient to use the OLE DB provider for the DBMS if one is provided instead of the OLE DB Provider for ODBC Drivers. After you have selected the provider, click the **Next** button to move to the next step.

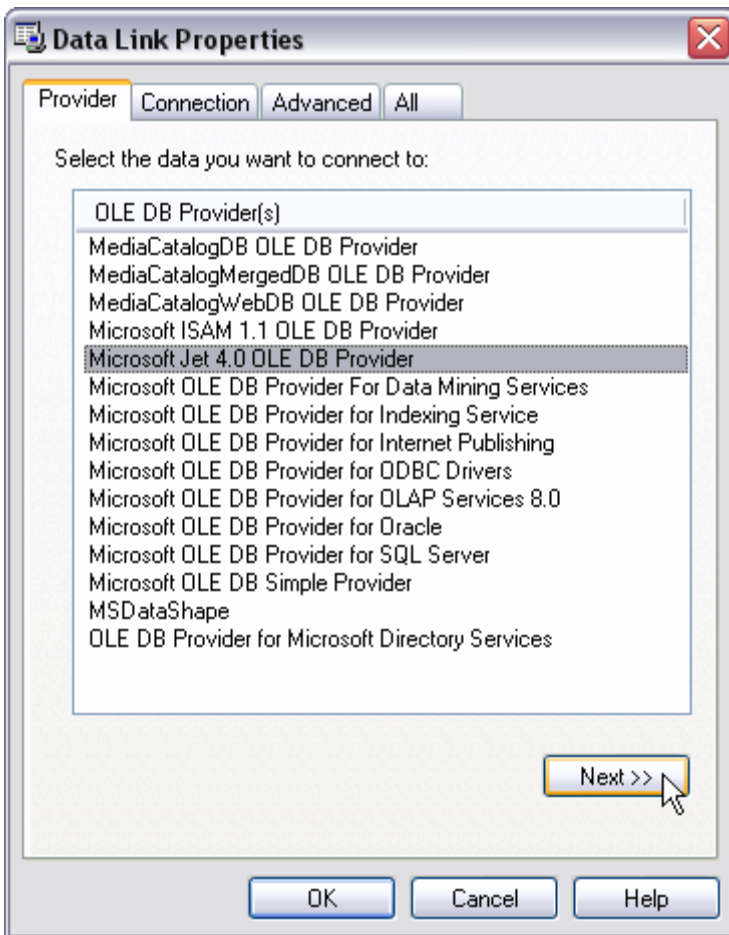


Figure 2.5 - Data Link Properties Provider Page

4. On the **Connection** tab, browse to and select the **Test Results.mdb** database you created on the desktop in Step 1. Click the **Test Connection** button to test the UDL connection.

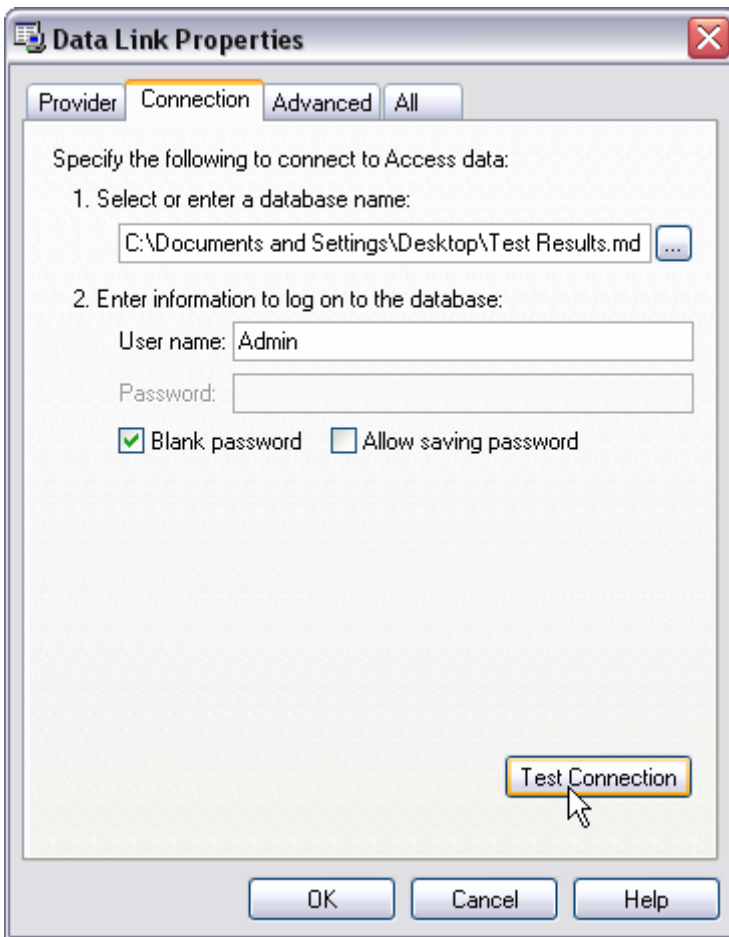


Figure 2.6 - Data Link Properties Connection Page

5. Click the **OK** button to complete the setup of the UDL.
6. Save the UDL as **TestResultsUDL.udl** in the **C:\Program Files\National Instruments\LabVIEW 8.0\Database\data links** folder.

In this example, you created an empty database in Microsoft Access. You then created two different connections to this database – a system DSN and a UDL.

End of Exercise 2-1

3

Inserting Data in to a Database

Once the connection to the database is established, you can then use the Database Connectivity VIs in LabVIEW to insert data in to the database.

In the next exercise you will use LabVIEW to create a table in a database and insert data into the table. Use the UDL created in Exercise 2-1 to communicate with the database.

You can complete the following exercise in approximately 10 minutes.

1. Open Exercise 3-1 from the Exercises\Database Connectivity Toolkit\Exercise 3-1 folder on the Desktop. The front panel and part of the block diagram has already been created for you.
2. Follow the steps below to build the remainder of the block diagram shown in Figure 3.1.

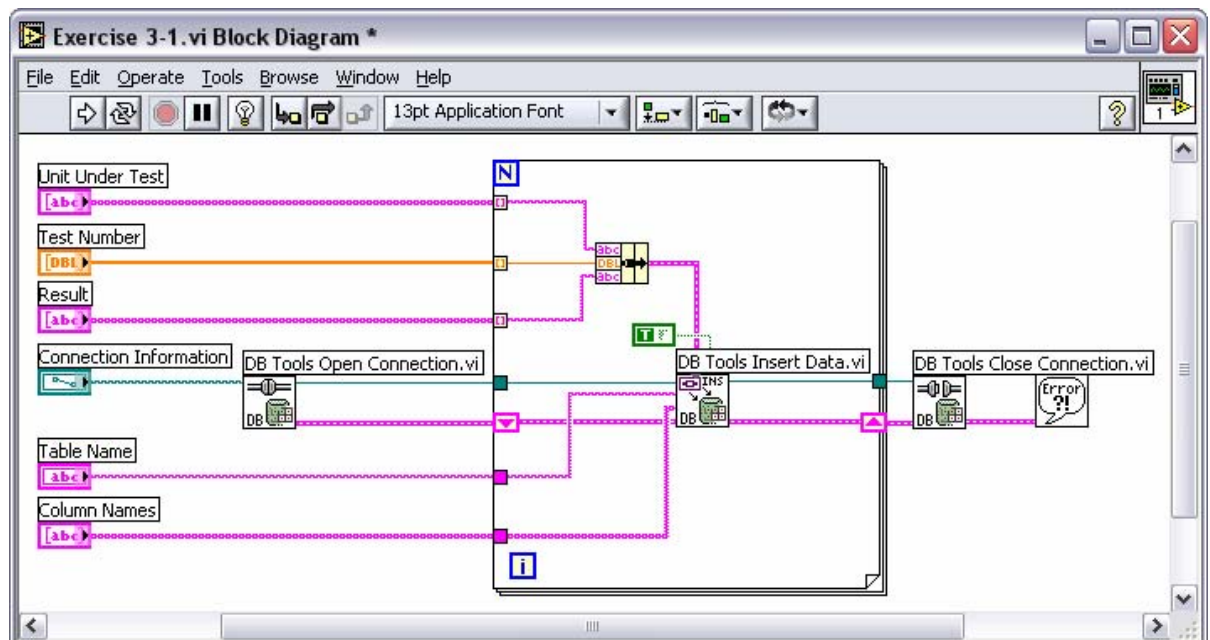


Figure 3.1 - Exercise 3-1 Block Diagram



- a. Place a **DB Tools Open Connection VI (All Functions » Database)** outside of the For Loop and wire the **Connection Information** File Path control to the **connection information**



- terminal.** Wire the **error** output to the shift register on the For Loop.
- b. Place a **DB Tools Insert Data VI (All Functions » Database)** inside the For Loop, and make the following connections:
 - i. Wire the **connection reference** from the **DB Tools Create Connection VI** to the **connection reference** input terminal.
 - ii. Wire the **output cluster** from the Bundle function into the **data** input.
 - iii. Wire the boolean constant in the **True** state to the **create table** input. This ensures that the table is created in the database if it does not already exist.
 - iv. Wire the **Table Name** string control to the **table** terminal. This is the name of the table that you are going to create and insert the data into.
 - v. Wire the **Column Names** array of String controls into the **columns** terminal. These names are the names of the columns you will add into the table and then insert the data into.
 - vi. Wire the **error out** to the shift register on the For Loop.
 - c. Place a **DB Tools Close Connection VI (All Functions » Database)** outside of the For Loop and wire the **connection reference** from the **DB Tools Insert Data VI** output terminal to the **connection reference** input terminal. Be sure to disable auto-indexing if it is enabled. Wire the error from the Shift Register to the **error in** input terminal.
 - d. Place a **Simple Error Handler VI (All Functions » Timing and Dialog)** at the end to display errors that may occur during the program. Wire the **error out** from the **DB Tools Close Connection VI** into the **error in** terminal of the **Simple Error Handler**.
3. Run the VI once. This will create the **ManufacturingTests** table in the **Test Results** database. It will also add six records into the table corresponding to the tests run on the UUTs.
 4. Save the VI to the Desktop as **Insert Test Results.vi**.
 5. View the updated Test Results Database in Microsoft Access.
 - a. Open Microsoft Access and select **File » Open** to open the **Test Results** database located on the Desktop.
 - b. Double-click on the **ManufacturingTests** table in the **Test Results** database to open the table and view the stored data. The table should appear similar to Figure 3.2.

UnitUnderTest	TestNumber	Result
A11C231	1	Failure - Segment 13
A11C231	2	Passed
A11C232	1	Passed
A11C232	2	Failure - Segment 2
A11C233	1	Failure - Segment 1
A11C233	2	Passed
*		

Record: 1 of 6

Figure 3.2 - ManufacturingTests : Table Window in Microsoft Access

In this exercise, the Database Connectivity Toolkit VIs allowed you to quickly and easily create a table in a Microsoft Access database and insert data in to that table, all from the LabVIEW environment.

End of Exercise 3-1

4

Selecting Data from a Database

Another common database operation is selecting data from a database.

In the next exercise you will use the high-level **Select Data From Database VI** to select data from a database.

You can complete the following exercise in approximately 20 minutes.

1. Open Exercise 4-1 from the Exercises\Database Connectivity Toolkit\Exercise 4-1 folder on the Desktop. The front panel and part of the block diagram has already been created for you.
2. Follow the steps below to build the remainder of the block diagram shown in Figure 4.1.

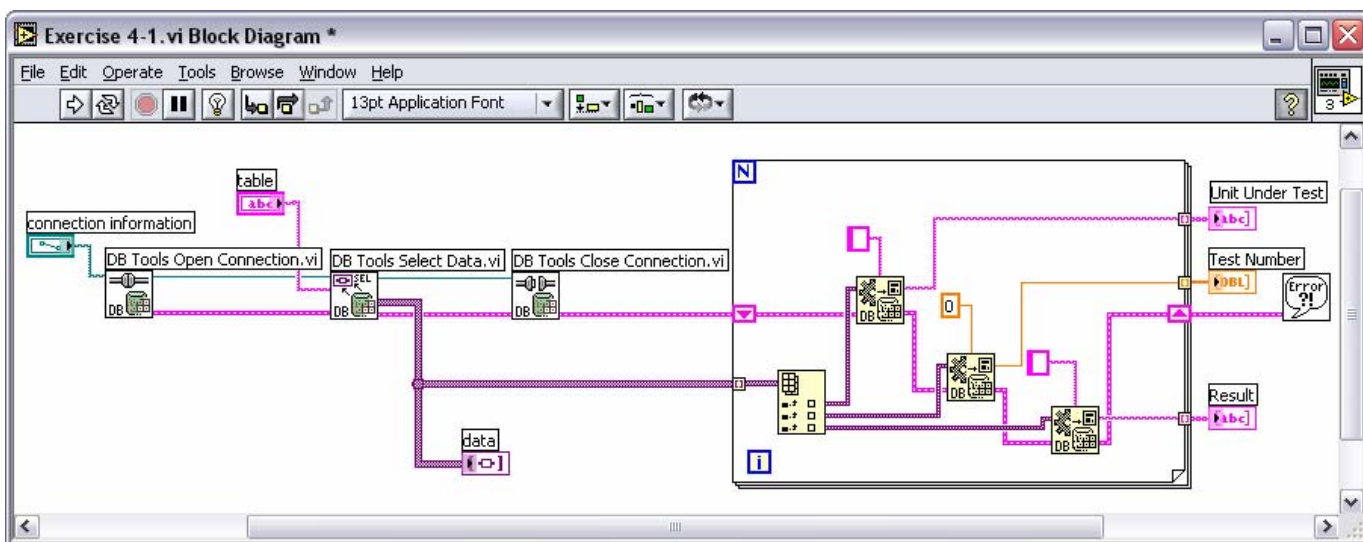


Figure 4.1 - Exercise 4-1 Block Diagram



- a. Place a **DB Tools Open Connection VI (All Functions » Database)** outside of the For Loop. Wire the **connection information** terminal to the **connection information** input. This control specifies the path to the UDL used to establish the connection to the database.
- b. Place a **DB Tools Select Data VI (All Functions » Database)** on the block diagram and complete the following wiring:
 - i. Wire the **table** string control to the **table** input terminal. This specifies which table you will select the data from.
 - ii. Connect the **connection information** and **error out** output terminals from the **DB Tools Create Connection VI** to the **connection information** and **error in** terminals of the **DB Tools Select Data VI**.
 - iii. Wire the **data** output terminal to the **data** variant array. Create

a second wire branch and wire the **data** to the For Loop for extraction in the loop.



- c. Place a **DB Tools Close Connection (All Functions » Database)** VI outside of the For Loop and wire the **connection reference** and **error out** from the **DB Tools Select Data** VI output terminals to the **connection reference** and **error in** input terminals of the **DB Tools Close Connection** VI. Also wire the **error out** to the shift register on the For Loop.



- d. Inside the **For Loop**, wire the **data** variant array to the **array** input of the **Index Array**.



- e. Place three **Database Variant To Data** functions (**All Functions » Database**) inside the For Loop. Wire the **error** clusters together to ensure data flow and to check for errors.

- i. Wire the first **element** of the **Index Array** to the **Database Variant** terminal of the first **Database Variant To Data** function. Wire a string constant to the **type** terminal. This will ensure that the variant data type is converted to a string.
- ii. Wire the second **element** of the **Index Array** to the **Database Variant** terminal of the second **Database Variant To Data** function. Wire the numeric constant to the **type** terminal to specify the variant data type as a numeric double.
- iii. Wire the third **element** of the **Index Array** to the **Database Variant** terminal. Wire the remaining string constant to the **type** terminal.

- f. Wire the **error out** from the final **Database Variant to Data** function to the Shift Register. Wire the **data** output terminals of each function to the edge of the For Loop. Ensure that each has **Auto-Indexing** enabled, and then wire the data to their corresponding indicators - **Unit Under Test**, **Test Number**, and **Result** respectively.



- g. Place a **Simple Error Handler** VI at the end to display errors that may occur during the program. Wire the **output** from the Shift Register to the **error in** terminal of the **Simple Error Handler**.

3. Run the VI. The VI will extract all of the data from the database and display it in the appropriate indicators, as Figure 4.2 illustrates.

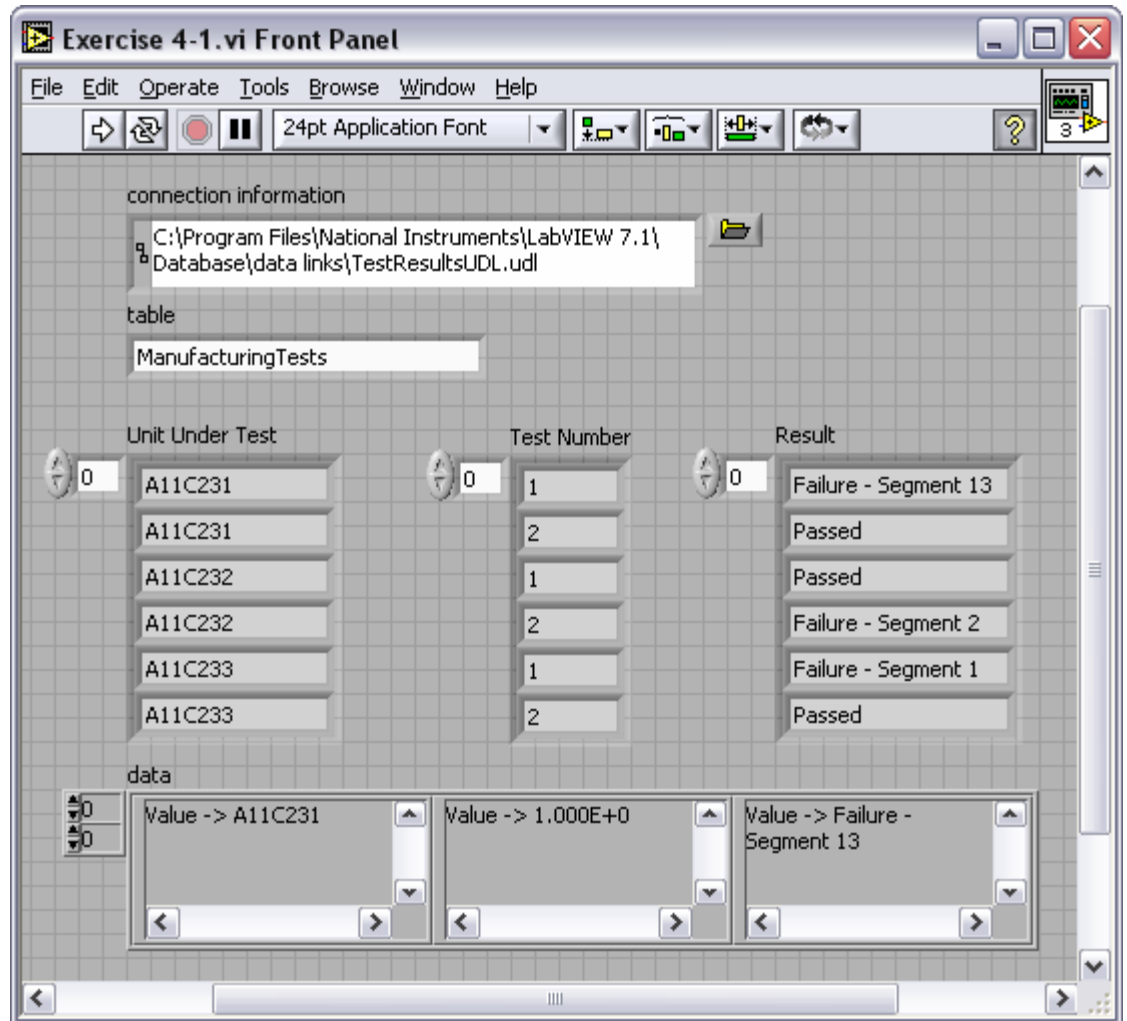


Figure 4.2 - Exercise 4-1 Front Panel After Running VI

4. Save the VI to the Desktop as **Select Test Results.vi**.

In this exercise, you selected and displayed data from an Access database using the Database Connectivity Toolkit VIs.

End of Exercise 4-1

5

Executing SQL Queries

To execute an SQL statement and navigate through the returned data, you can use the Advanced VIs in the Database Connectivity Toolkit. To execute a query, use the **DB Tools Execute Query VI (All Functions » Database » Advanced)**. This VI accepts an **SQL query** string, which is an SQL query, and it returns a **recordset reference**. The recordset reference is a recordset corresponding to all of the records that matched the query parameters.

Once you have a reference to the matching data, you can fetch data from the recordset in a variety of forms. You can extract all of the data at one time using the **DB Tools Fetch Recordset Data VI**, or you can navigate through individual records or fields using the other Advanced VIs.

After the data has been fetched from the recordset, close the reference to the recordset using the **DB Tools Free Object VI**.

In the SQL Query example in Figure 5.1, all of the fields for records in the **testdata** table that have **true** in their **col4** field are fetched from the database.

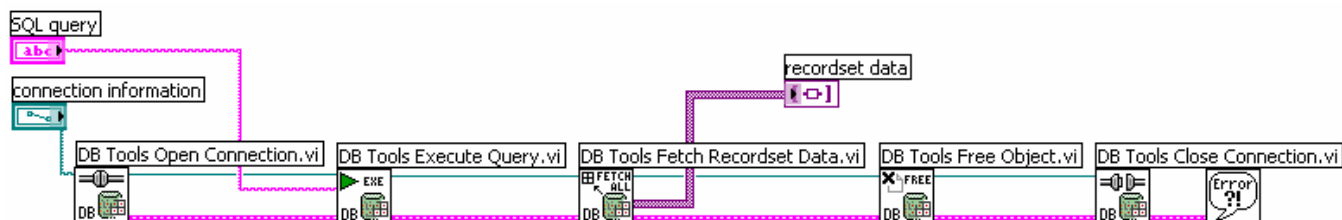


Figure 5.1 – Example of Executing an SQL Query

6

Summary

Congratulations! During this short tutorial you have successfully accomplished the following tasks using some of the functionality included in the LabVIEW Database Connectivity Toolkit:

- Created two different connections to a database – a system DSN and a UDL
- Created a table and inserted data in to a database using the Database Connectivity Toolkit VIs
- Selected and retrieved data from a database using the Database Connectivity Toolkit VIs
- Learned more about advanced functionality available with the Database Connectivity Toolkit, such as executing SQL queries

Now that you are familiar with some of the functionality included in the Database Connectivity Toolkit, please feel free to continue exploring the numerous other features both LabVIEW and this toolkit have to offer.