SQL Toolkit for G Reference Manual

February 1997 Edition
Part Number 321525A-01

NOTE: National Instruments Corporation has purchased DatabaseVIEW™ from Ellipsis Products, Incorporated. Contact National Instruments Corporation for all sales and support information.

Internet Support

`support@natinst.com`
E-mail: `info@natinst.com`
FTP Site: `ftp.natinst.com`
Web Address: `http://www.natinst.com`

Bulletin Board Support

BBS United States: (512) 794-5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support

(512) 418-1111

Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678

International Offices

Australia 02 9874 4100, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 09 527 2321, France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
Israel 03 5734815, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886,
Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway   Austin, TX 78730-5039   Tel: (512) 794-0100

Important Information

Warranty

Copyright

Trademarks

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

Contents

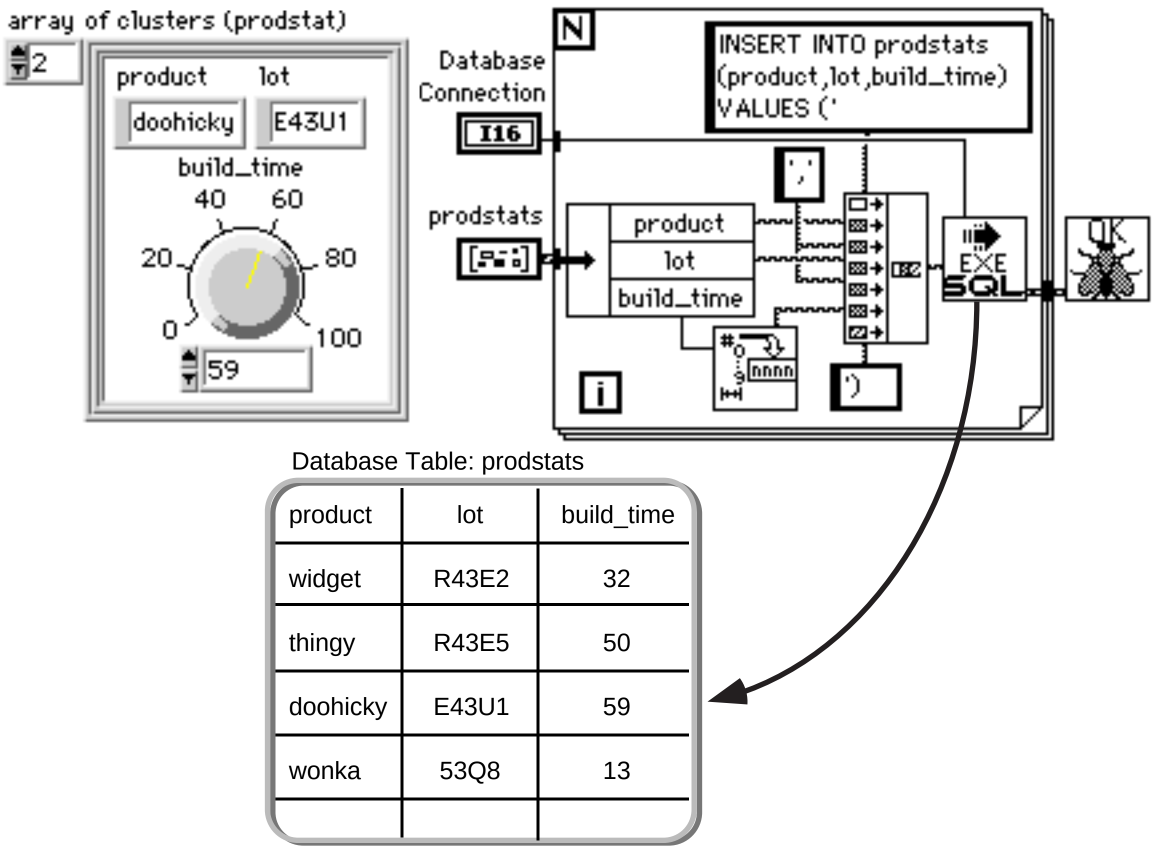
List of Figures

List of Tables

Welcome and Introduction

Welcome to the world of scientific and technical databases!

DatabaseVIEW for Windows is Ellipsis' premier Windows product in the Medium family of software that connects the LabVIEW graphical programming environment to popular database systems. The DatabaseVIEW for Windows virtual instrument (VI) library is an Open Database Connectivity (ODBC) compliant programmatic bridge, or, application programming interface (API) from LabVIEW to text files, spreadsheets, and several leading database systems listed in Table 1, pg. xvi. It provides LabVIEW developers with full Dynamic Structured Query Language (Dynamic SQL) access to all database functions and services directly from the LabVIEW block diagram as shown in Figure 1.



product	lot	build_time
widget	R43E2	32
thingy	R43E5	50
doohicky	E43U1	59
wonka	53Q8	13

Figure 1. DatabaseVIEW for Windows Environment

Table 1. Databases Supported by DatabaseVIEW for Windows

Btrieve	Interbase	PROGRESS
dBASE II - V, FoxBase FoxPro1, 2.5, & Compatibles	MDI Gateway (DB2, Teradata, SQL/400, SQL/DS)	SQLBase
EXCEL & EXCEL5 files	MS SQL Server	Sybase 10
HP ALLBASE/SQL	NetWare SQL 2.11, 3.x	ASCII Text
HP IMAGE/SQL	ORACLE 6 & 7	XDB
INFORMIX 4 & 5	OS/2 DBM and DB2/2	
INGRES 6.4/03 & 6.4/04	Paradox & Paradox 5	

DatabaseVIEW for Windows "bridges" the data acquisition and analysis domain of LabVIEW to the data storage, retrieval, and management services of databases; thereby creating a complete technical information system. These systems may be applied to a variety of problems, both local and distributed in nature, in the areas of:

- Supervisory Control & Data Acquisition (SCADA) Systems
- Laboratory Automation Systems
- Automated Test Equipment (ATE) Systems
- Laboratory Information Management Systems (LIMS)
- Workgroup Data Management
- Computer Integrated Manufacturing (CIM) and Test Systems
- Statistical Process Control (SPC) Systems
- Quality Control Test Systems
- Database / Enterprise Integration

DatabaseVIEW for Windows (DBVIEW) may be used to access a database that is local to the computer running LabVIEW (stand-alone operation); or a server database located on other computers via network connections (remote or client-server operation). Client/server database operation is available only with certain database systems, as noted in the DatabaseVIEW Driver Help (see *Database Driver Help*, pg. 6 - 8).

Database access is achieved through a modular library of VIs that ‘encapsulate’ common database service functions into an application programming interface. Figure 2 illustrates how simple database programming can be with DBVIEW VIs.

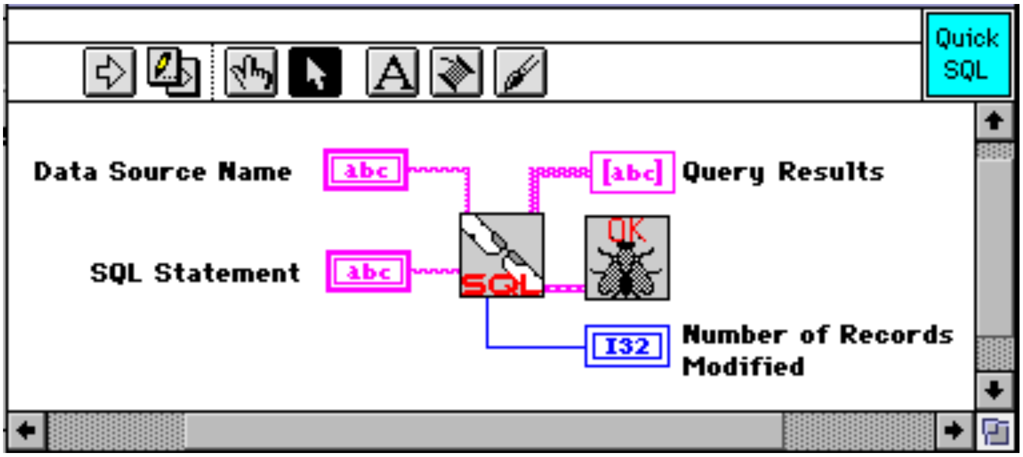


Figure 2. Simple Database Programming with DatabaseVIEW for Windows

The DatabaseVIEW for Windows library is organized into multiple functional levels or layers (see Figure 3, pg. xviii). *Access VIs* afford direct database access with a minimum of block diagram programming, and are designed to insulate the LabVIEW developer from the particulars of database operations. The *Interface VIs* provide detailed access to lower-level database functions. General-purpose *Utilities* and on-line help files are also provided to assist in database application development, set-up, and operation. The utilities are not intended for direct use in database application programs, but rather are intended to serve as development aids for ODBC administration and on-line help access.

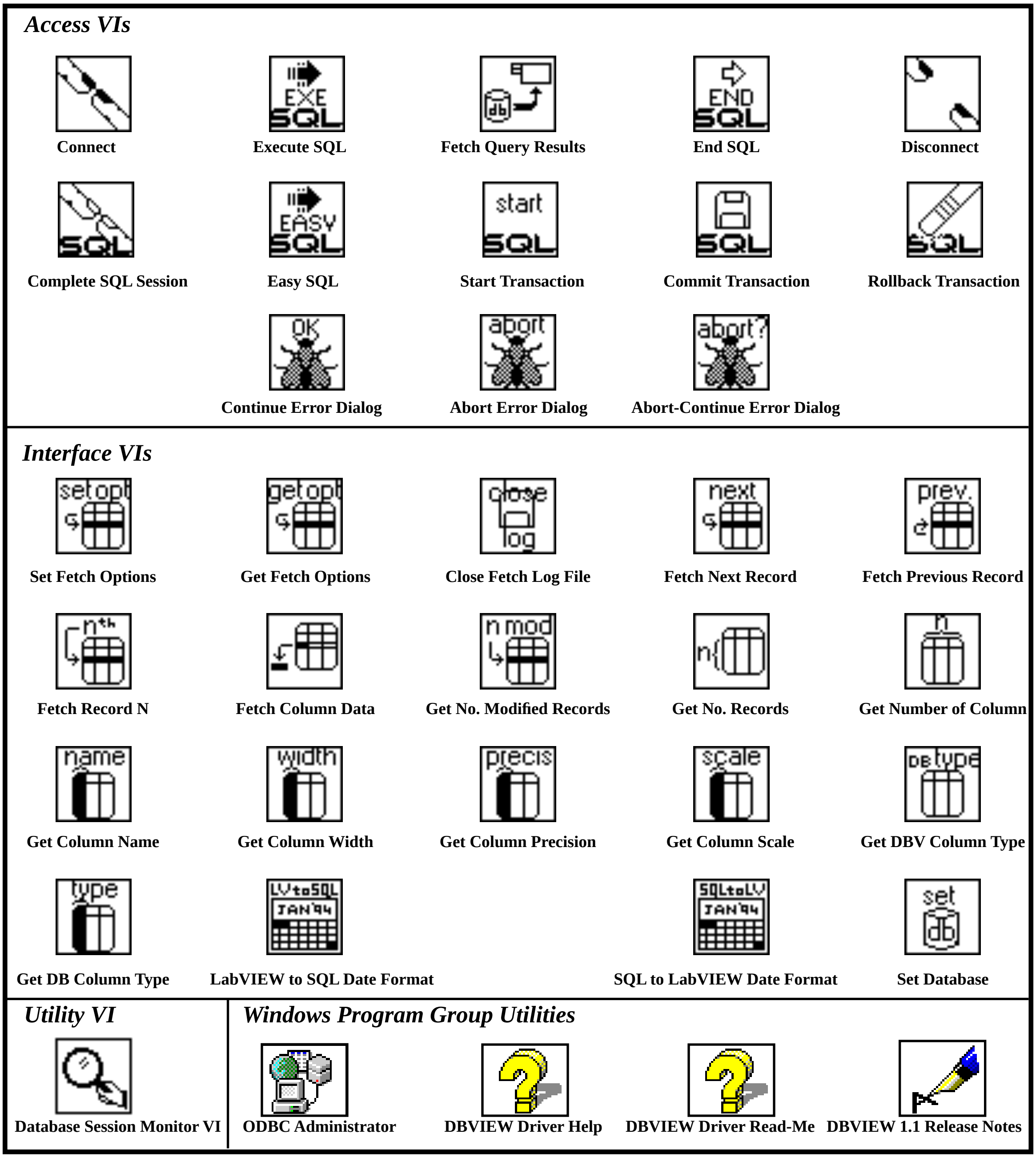


Figure 3. DatabaseVIEW for Windows Functional Organization

About This Manual

This *DatabaseVIEW for Windows User Reference* describes how to use DatabaseVIEW VIs to access databases from the LabVIEW environment. It has been written from a LabVIEW application developer's perspective. The manual presents database and LabVIEW information only as it pertains to building application VIs that use DatabaseVIEW for Windows VIs in their block diagrams. A fundamental conceptual and operational knowledge of both LabVIEW and the chosen databases was assumed in writing this user manual. A reference listing is provided in Chapter 1 for users requiring further detailed information about databases in general and LabVIEW.

Experience has shown that a basic "good job" can be achieved implementing LabVIEW database applications using just the information contained in this manual. However, the Ellipsis technical staff strongly recommends acquiring the full developer software package for the chosen databases, whenever possible, to achieve a truly optimized database implementation.

How This Manual is Organized

Chapter 1 - Installation & Quick Start

Chapter 1 provides direct DatabaseVIEW for Windows installation and operating instructions. For users already familiar with basic LabVIEW and database operation, this chapter contains enough information to begin developing application VIs immediately. Users wishing a more detailed explanation of database operation are referred to Chapter 3, *Using DatabaseVIEW VIs*, pg. 3 - 1 where this information is reviewed in depth. A reference listing is also provided for more detailed information on LabVIEW and database software.

Chapter 2 - Environment & Applicability

Chapter 2 introduces the DatabaseVIEW operating environment. Brief descriptions of LabVIEW, the databases, the Open Database Connectivity (ODBC) standard, the Structured Query Language, and network communications are given. Discussion is presented on the conceptual link between LabVIEW and a relational data model. Finally, the suitability of incorporating database functionality into various classes of LabVIEW applications is explored.

Chapter 3 - Using DatabaseVIEW VIs

Chapter 3 describes how to use the DatabaseVIEW VIs to achieve high-level SQL access to databases with a minimum of programming. It begins with a general discussion of database operations from a programmer's perspective; and includes descriptions of how to implement session and transaction operations using the DBVIEW VIs.

Chapter 4 - Access VI Reference

Chapter 4 provides a detailed description for each of the DatabaseVIEW Access VIs, which are used to perform high-level database operations. There are three categories of VIs: Session Operation, Transaction Operation, and Error Handling. Each of the Access VIs is built entirely from the low-level Interface VIs described in Chapter 5.

Chapter 5 - Interface VI Reference

Chapter 5 provides a detailed description for each of the Interface VIs, which are the elemental building blocks for the Access VIs. Interface VIs provide more complete low-level access to database services.

Chapter 6 - Development Utilities

Chapter 6 describes general-purpose utilities, which are used as development aids. They provide access to ODBC administration, database session monitoring, and on-line help facilities.

Chapter 7 - Application Examples

Chapter 7 presents several example applications that incorporate DatabaseVIEW VIs. The first example, Quick SQL, illustrates the simplest of database programming. The second, a factory quality control test station application, presents a generic top-down structured design process for developing LabVIEW applications that incorporate database functionality. The remaining examples are provided for user study and reuse.

Appendices

The appendices provide additional information on SQL commands, a glossary of terms, and an index to the VI libraries.

Customer Communication

Ellipsis Products actively strives for quality growth in all its products, documentation, and services through a concerted program of customer feedback. We welcome and encourage all of your written or oral comments and suggestions.

If you purchased DatabaseVIEW for Windows from National Instruments, then you should first contact NI with your technical support questions. The applications engineering staff at NI has been thoroughly trained in all aspects of DatabaseVIEW installation and operation. You may reach NI at:

Internet:	lvsupport@natinst.com
Telephone:	(512) 795-8248
Fax:	(512) 794-5678

If you purchased DatabaseVIEW directly from Ellipsis Products, then you should contact us directly for technical support at:

Internet:	ellipsis@world.std.com or A5683643656@attpls.net
AT&T PLS:	A5683643656
Compuserve:	76600,1012
Telephone:	(617) 236-0141 from 9:00 AM to 5:00 PM Eastern USA time.
Fax:	(617) 236-0141 or (617) 659-7004
Mail:	Ellipsis Products, Inc. 741 Grove Street Norwell, MA 02061

Chapter 1 Installation & Quick Start

This chapter provides direct installation and operating instructions for DatabaseVIEW for Windows. For users already familiar with basic LabVIEW and SQL database operation, this chapter contains enough information to begin developing application VIs immediately. Users wishing a more detailed explanation of DBVIEW database operation are referred to Chapter 3 where this information is reviewed in depth. A reference listing is also provided for more detailed information on LabVIEW and database software.

Operating Environment Requirements

This section describes the hardware and software requirements for DatabaseVIEW VI library operation. Establishing absolute system hardware and software requirements can be a challenging task given the wealth of databases that DatabaseVIEW for Windows supports and the great variety of possible computer system configurations. This task is further complicated when using databases that are accessed via a network. For these reasons, this chapter specifies only the operating environment requirements for a basic DatabaseVIEW for Windows installation. The user should consult *Database Driver Help*, pg. 6 - 8 and *Database Driver Read-Me*, pg. 6 - 8 for details on the selected databases. A rule of thumb for quick estimation is: **the computer must be sufficiently equipped to run LabVIEW *and* access the database at the same time**. Careful attention must be paid to disk storage space, RAM (conventional, expanded, and extended), network driver software, network hardware, etc. These factors are particularly important when developing applications that process very large queries, as well as those that make use of the random fetch options (see *Fetch Option VIs*, pg. 5 - 1).

- LabVIEW 3 for Windows requires:
    - an 80386 or 80486 (recommended) CPU with floating point capability
    - Microsoft Windows 3.1 or later running in 386 enhanced mode
    - 8Mb minimum of RAM
    - 33 Mb of disk space for a full installation

- DatabaseVIEW for Windows 1.0 requires:
    - a minimum of 9.2 Mb of disk space for a full installation[1].

Experience with DBVIEW and LabVIEW 3 for Windows has shown that a minimum of 8 Mb of RAM is required to perform useful work.

Installation

Installing DatabaseVIEW Software

DatabaseVIEW for Windows is distributed on 4 DOS formatted high-density floppy disks which contain the following groups of VI libraries (.llb) and support files:

- DBVIEW Access and Interface VI Libraries
- DBVIEW Application Examples and Utilities
- DBVIEW ODBC Drivers and Support Files

Backup copies of these disks should be made as soon as possible to prevent information loss.

The Access and Interface VI libraries contain all VIs necessary to build complete database application VIs. The Application Examples and Utilities group contains several instructive programming examples, as well as additional utilities that are useful for general database development and operation. The ODBC Drivers and Support Files consist of several Windows dynamic link library (.dll) files, known as *database drivers*, that are necessary to access the various supported databases[2]. Each group of files is installed directly from the floppy disks.

1. Note that a “full” installation would include the ODBCdrivers for ALL supported databases, however, it is unlikely that any single installation would require this. A typical situation would likely include only a few specific databases, and would typically require around 3.5 Mb of disk space.

2. Note that some databases require additional dynamic link library and other files that must be purchased from the database publishers directly. Consult *Database Driver Help*, pg. 6 - 8 for specifics about a particular database.

DBVIEW files are automatically installed onto your computer by a conventional Windows SETUP program. SETUP will prompt for the specific location to install the DBVIEW files on the hard disk. The VI library .LLB files may be installed in any directory that you prefer, while the ODBC database drivers are installed in the current \WINDOWS\SYSTEM directory. SETUP also automatically installs or updates the necessary ODBC administration and on-line help files in the \WINDOWS and \WINDOWS\SYSTEM directories, and creates a Data Source Name (DSN) for the DBVIEW Examples databases. (ODBC and DSNs are described in *The ODBC Standard*, pg. 2 - 4, and *ODBC Administrator*, pg. 6 - 3.)

Starting the Installation Process

The following instructions offer a simple guideline for application developers who wish to have the DBVIEW VIs appear under the LabVIEW Functions menu when editing block diagrams. **You must quit <u>all</u> Windows applications and control panels prior to running SETUP. Only the Windows Program Manager should be visible on your screen.** This is to ensure that critical Windows files are not in use prior to installer execution. You should also jot down the full path of your LabVIEW directory prior to starting SETUP.

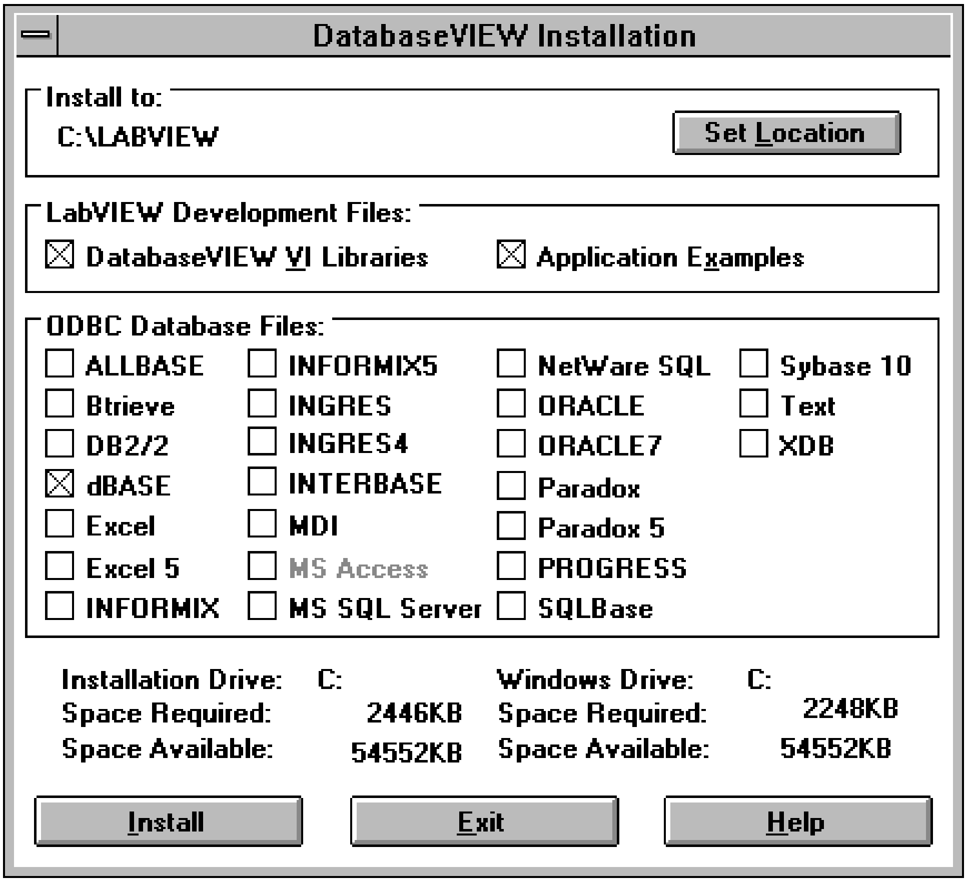
1. Insert the floppy disk labeled "Disk 1 - Setup" into your floppy drive.
2. From the Program Manager **File** menu, choose **Run**.
3. Type **A:Setup** into the dialog box, then press <enter>.

   (Be sure to substitute your floppy drive letter for **A** in the dialog box.)

The SETUP program will begin to execute. A series of dialog boxes will then be presented to guide you through the installation. Note that you may exit the installation program at any time by clicking on the **Exit** pushbutton control, which is present in all dialogs. A **Help** pushbutton control is also available on several of the dialogs to provide additional information.

Selecting the Components to Install

SETUP presents a selection dialog so that you may choose which DBVIEW components to install. Figure 4 illustrates this dialog. There are three important selections to make: the target disk location to install the VI libraries, which development files to install, and which ODBC database drivers to install. **Note that the DatabaseVIEW VI Libraries and at least one ODBC Database Driver are the absolute minimum required to achieve database access.**



DatabaseVIEW Installation

Install to: C:\LABVIEW [Set Location]

LabVIEW Development Files:

☒ DatabaseVIEW VI Libraries ☒ Application Examples

ODBC Database Files:

☐ ALLBASE	☐ INFORMIX5	☐ NetWare SQL	☐ Sybase 10
☐ Btrieve	☐ INGRES	☐ ORACLE	☐ Text
☐ DB2/2	☐ INGRES4	☐ ORACLE7	☐ XDB
☒ dBASE	☐ INTERBASE	☐ Paradox	
☐ Excel	☐ MDI	☐ Paradox 5	
☐ Excel 5	☐ MS Access	☐ PROGRESS	
☐ INFORMIX	☐ MS SQL Server	☐ SQLBase	

Installation Drive:	C:	Windows Drive:	C:
Space Required:	2446KB	Space Required:	2248KB
Space Available:	54552KB	Space Available:	54552KB

[Install] [Exit] [Help]

Figure 4. SETUP Selection Dialog

Install to: Target Disk Location

The default target disk location is C:\LABVIEW, a typical location for the LabVIEW home directory. If necessary, you should replace this with your LabVIEW home directory path by clicking on the **Set Location** pushbutton control, which will present a dialog for you to specify the full directory path. You may specify any valid path you wish, however choosing the LabVIEW home directory will ensure that the DBVIEW VIs are visible from the Functions menu while editing block diagrams. Click on **OK** to accept the path and return to the selection dialog.

LabVIEW Development Files

The default selection includes both the DBVIEW VI libraries and the application examples. Click on the checkbox controls to select the items you wish to install. Note that the DBVIEW VI libraries and the dBASE ODBC driver must be installed for the application examples to operate.

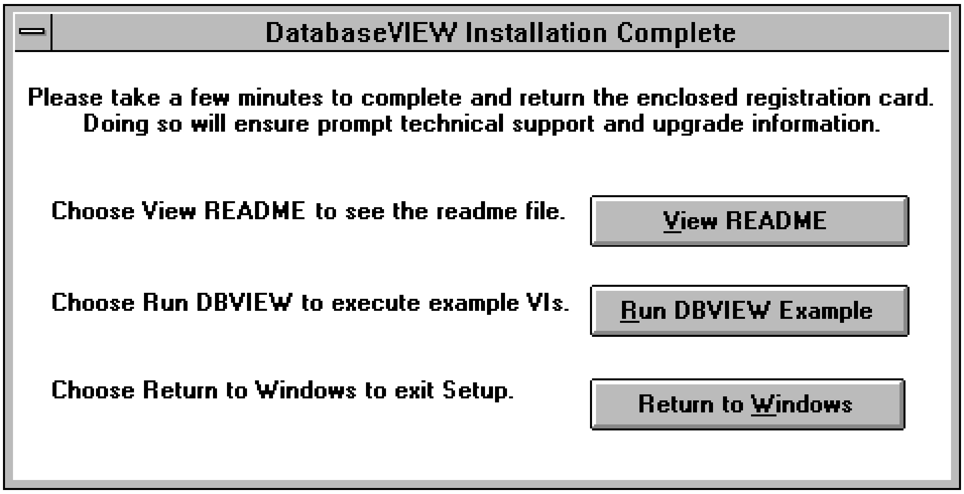
ODBC Database Drivers

Click on the checkbox controls to select the drivers you wish to install. To conserve disk space, select only the database drivers that you will need for your application. The default selection includes the dBASE driver because it is needed for the application examples to work. Thus, dBASE will remain checked as long as examples are checked above.

Continuing the Installation

When you’ve completed your selections, click on the **Install** pushbutton control to begin installing the selected files. SETUP will prompt you to insert the necessary installation disks. If the install path you’ve specified is not the LabVIEW home directory, SETUP will prompt you to specify the actual LabVIEW path via a dialog. The LabVIEW path is necessary to properly install the DBVIEW program group and items in the Windows Program Manager. The LabVIEW path is also necessary to properly add the DBVIEW installation path to the VI search path in LABVIEW.INI.

When SETUP has completed the installation, it will present an options dialog as shown in Figure 5. Three options are offered: view the README file to learn more about this release of DBVIEW, run the application examples (if installed) using LabVIEW, or return to Windows. (One or more of these may be dimmed depending on the items you chose to install.) Make your selection by clicking on the corresponding pushbutton control. SETUP will perform the selected action, and then automatically exit.



Please take a few minutes to complete and return the enclosed registration card. Doing so will ensure prompt technical support and upgrade information.

Choose View README to see the readme file.

Choose Run DBVIEW to execute example VIs.

Choose Return to Windows to exit Setup.

Figure 5. SETUP Completion Options

Running the DatabaseVIEW Demo example is a good way to confirm that the installation was completely successful. It also provides an excellent starting point for exploring database access from LabVIEW.

If you choose to run the application examples, LabVIEW will be launched, and a VI select dialog will be presented as shown in Figure 6. Select an example VI to run, and then click **OK**.

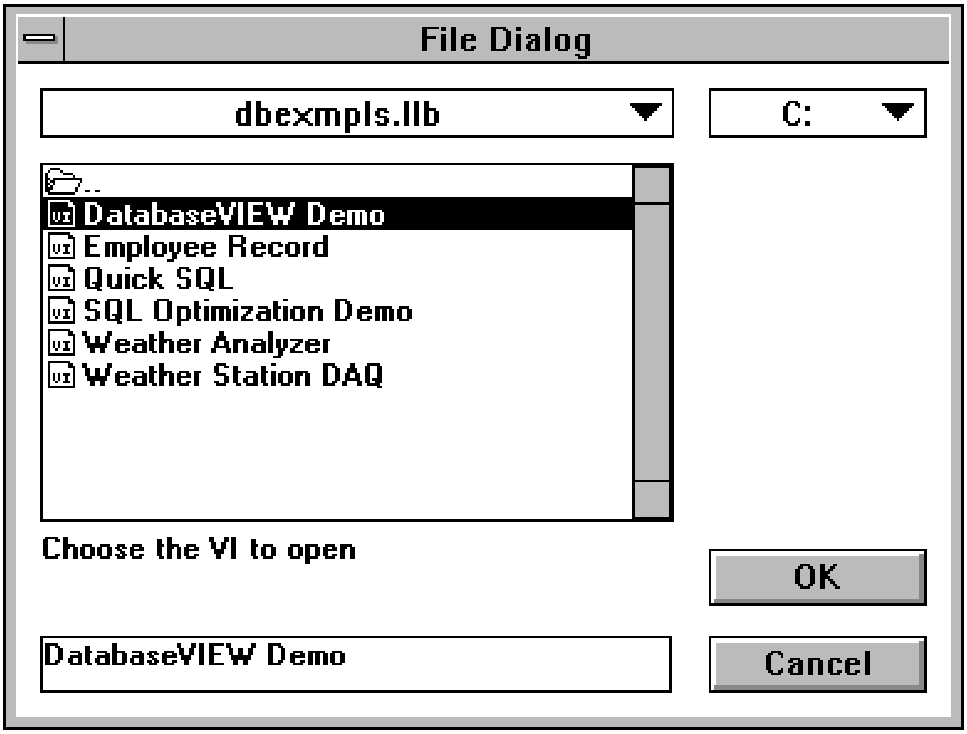
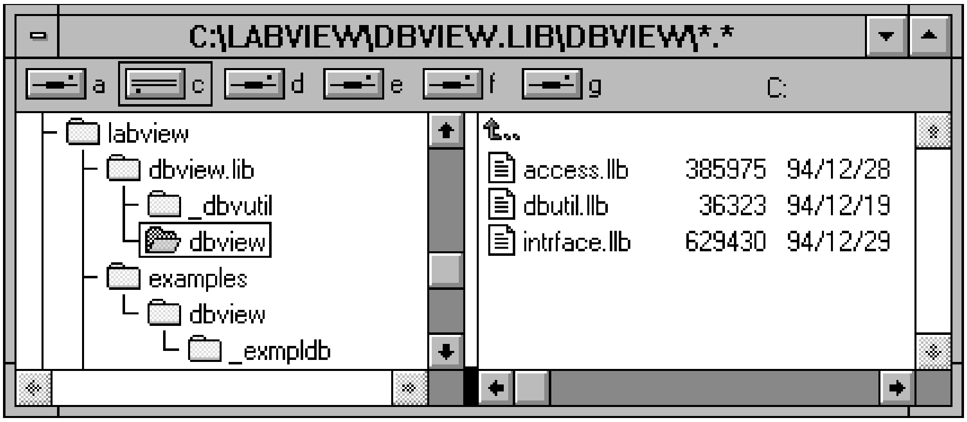


Figure 6. Running the DBVIEW Examples

DBVIEW installation is now complete. Figure 7, pg. 1 - 8 shows how a typical installation might look from the Windows File Manager. Note that the application example files have been installed in the LabVIEW EXAMPLES directory. SETUP has also created a Program Group in the Windows Program Manager, as shown in Figure 8, pg. 1 - 8. The program item DatabaseVIEW Read Me may be double-clicked to learn more about this release of DBVIEW. The remaining program items are described in detail in Chapter 6 and Chapter 7.



```
access.llb    385975  94/12/28
dbutil.llb     36323  94/12/19
intrface.llb  629430  94/12/29
```

Figure 7. Installed DBVIEW Files

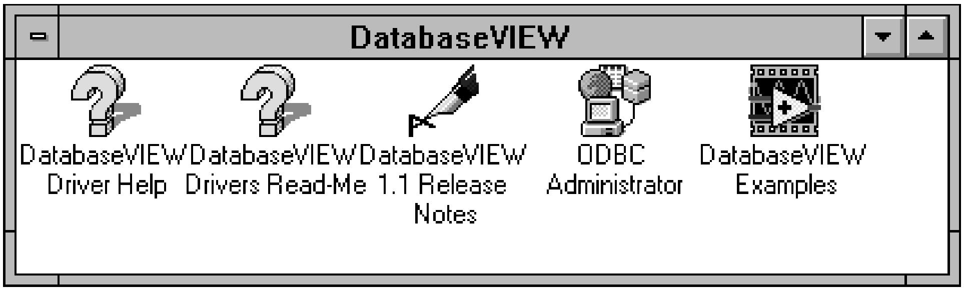


Figure 8. DBVIEW Program Group

The DBVIEW VIs are now accessible from the LabVIEW **Functions** menu as shown in Figure 9.

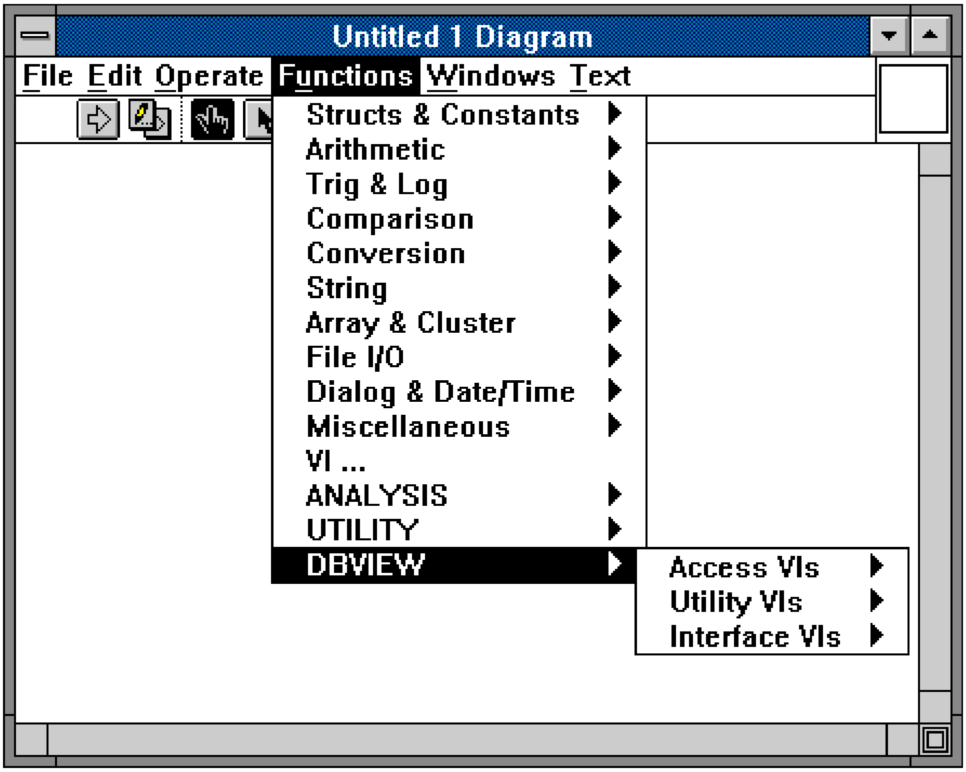


Figure 9. LabVIEW Functions Menu with DBVIEW Installed

Installing Additional Database Drivers

After the initial installation, SETUP may be used at any time to install additional database drivers. This includes drivers provided with the DBVIEW distribution, as well as new drivers that Ellipsis may make available from time to time. For the former, simply perform the same procedure described above.

DatabaseVIEW for Windows complies with the ODBC standard, and should therefore function properly with most ODBC compatible drivers, regardless of their source. ODBC drivers from other vendors should be installed using the publisher’s instructions, or by using the *ODBC Administrator*, pg. 6 - 3.

Database Installation

Many of the databases that DBVIEW supports do not require any additional software to operate, while others require some components from their respective vendors. The user should follow the vendors’ instructions, where applicable, for installing the selected databases and necessary network hardware and software. There are several possible database access configurations, including local file-based; local database engine; remote file-based; and remote client/server based. These configurations are described further in Chapter 2.

Wherever practical, a database specific application (ideally one published by the database manufacturer) should be used to confirm that the database and its associated communications software are operating correctly. This method of testing is essential for remote databases that must be accessed over a network.

Once the database has been sucessfully installed and tested, the user should consult the on-line help system, in particular *Database Driver Help*, pg. 6 - 8 for additional set-up instructions. This is especially important for those database systems that require data dictionary preparation.

Quick Start

The following sections provide a condensed description of how LabIEW applications use DBVIEW VIs and SQL to interact with databases. Further discussion on these topics is given in Chapter 2, *Environment & Applicability*, pg. 2 - 1, Chapter 3, *Using Database-VIEW VIs*, pg. 3 - 1, Appendix A, *SQL Quick-Reference*, pg. A - 1, and in *Database Driver Help*, pg. 6 - 8.

The ODBC Standard

The Open Database Connectivity standard, developed by Microsoft Corp., provides a common method for application access to databases. It consists of a multi-level application programming interface definition, a driver packaging standard, a common SQL implementation based on ANSI SQL, and a means for defining and maintaining Data Source Names (DSN). A Data Source Name is essentially a ‘shorthand’ means to refer to a specific database. It is specified by a unique name, and the ODBC driver that is used to communicate with the physical database (local or remote). A DSN must be defined for each database an application program needs to connect to. DSNs are defined using the *ODBC Administrator*, described on page 6-3.

The Structured Query Language

The Structured Query Language, or SQL, is a set of commands that all DBVIEW-based application programs use to describe, insert, and retrieve data in accessible databases. SQL is a broadly subscribed international standard for database access, and is supported to some degree in most major commercial relational database products. It is a non-procedural language that allows the user to refer to and process sets of records in database tables, which consist of rows (records) and columns (fields). Some frequently used SQL commands are:

- **CREATE TABLE** - creates a new database table and specifies the name and data type for each column therein.
- **INSERT** - adds a new data row to the table, allowing values to be specified for each column.

- **SELECT** - searches for and retrieves all rows in a table whose column data satisfy specified conditions.
- **UPDATE** - searches for rows satisfying specified conditions, and then changes the contents of specific columns in those rows.
- **DELETE** - searches for rows satisfying specified conditions, and then deletes those rows from the table.

An example SQL statement might read as follows (refer to the database table in Figure 1, pg. xv, SQL keywords are bold):

SELECT product, lot **FROM** prodstats **WHERE** lot='R43E2'

This statement is interpreted as "retrieve the contents of columns 'product' and 'lot' from the 'prodstats' table, including only those rows where the value contained in the 'lot' column equals 'R43E2'".

See Appendix A, *SQL Quick-Reference*, pg. A - 1, and the 'SQL for Flat File Drivers' topic in DatabaseVIEW Driver Help for a more complete description of the SQL language and its components.

The Typical SQL Database Session

All database operations are performed through a *SQL session*. A typical SQL session consists of the following sequence of operations (Figure 10):

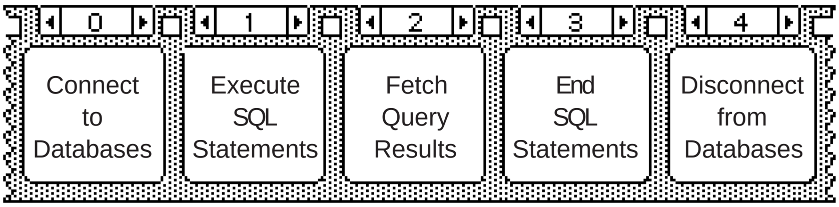


Figure 10. The SQL Session Sequence

Note that the SQL session is depicted as a LabVIEW sequence structure only to emphasize the required operation precedence.

0. Connect to Databases - establish a logical *connections* to the selected databases by specifying an ODBC Data Source Names (DSN). A SQL session is considered active once a successful connection has been established.

1. Execute SQL Statements - activate *SQL references* and transmit the SQL statements to connected databases to be executed. Memory permitting, any number of SQL statements may be executed once connections have been established.

2. Fetch Query Results - retrieve any active SQL SELECT query results into LabVIEW. Data may be retrieved all at once or piecemeal depending on the application's needs.

3. End SQL Statements - clean up and release memory and database resources when active SQL statements are no longer needed. SQL references are rendered inactive, and query results may no longer be fetched.

4. Disconnect from Databases - disconnect from the selected databases. A SQL session is no longer active once it has been disconnected.

Within a session, database manipulation activities are often logically grouped into *transactions*. DBVIEW (in conjunction with certain selected databases) provides a transaction management mechanism to temporarily buffer groups of database changes. An application may *start* a *transaction* to begin a transaction sequence, *commit* a transaction to make any changes permanent; or, in the event of exceptions, *rollback* a transaction to discard the changes.

DatabaseVIEW VI Library Structure

DBVIEW VIs are used to perform all operations in a SQL session. They are collected into two functional levels: high-level Access VIs designed for ease of programming; and low-level Interface VIs that provide facilities for detailed database operation and performance optimization.

DatabaseVIEW Access VIs

These VIs provide high-level access to SQL database functionality. Their use requires very little detailed knowledge of database programming interfaces. They allow the developer to rapidly add database access to application VIs with a minimum of LabVIEW programming. The following sections briefly introduce the DBVIEW Access VIs. Detailed descriptions are provided in Chapter 4, *Access VI Reference*, pg. 4 - 1.

Connect

Connect (pg. 4 - 2) establishes and activates a connection to a selected database identified by its Data Source Name, thereby beginning a database session.

Complete SQL Session

Complete SQL Session (pg. 4 - 5) performs all steps of the database session sequence. It establishes an active connection to the specified database, executes the SQL statement, retrieves SELECTed data (if any), and then disconnects from the database.

Easy SQL

Easy SQL (pg. 4 - 7) performs a complete SQL operation on a previously connected database. The SQL statement is first executed, and then any resulting query data is retrieved directly into the LabVIEW block diagram.

Execute SQL

Execute SQL (pg. 4 - 9) activates a SQL reference and executes the SQL statement on the selected database. Query data must be retrieved separately. This VI is particularly useful for optimizing application performance.

Fetch Query Results

Fetch Query Results (pg. 4 - 10) retrieves SQL query data from a specified active SQL reference into the LabVIEW block diagram. A SQL statement must have executed successfully before this VI can be run.

End SQL

End SQL (pg. 4 - 12) is used to terminate a SQL reference and clean-up memory and file resources after a SQL operation has been completed.

Disconnect

Disconnect (pg. 4 - 13) terminates the specified database session, thereby disconnecting from the database.

Start Transaction

Start Transaction is used to begin a transaction series on the selected database connection. Database changes performed by subsequent SQL operations are made only temporarily, and may be removed if desired.

Commit Transaction

Commit Transaction (pg. 4 - 15) causes the database to permanently enter all changes that have been made to its contents during the current transaction.

Rollback Transaction

Rollback Transaction (pg. 4 - 16) instructs the database to permanently undo all changes that have been made to the database during the current transaction.

Abort Error Dialog

Abort Error Dialog (pg. 4 - 17) presents an abort dialog box that describes a database error and then terminates the database session and LabVIEW execution.

Abort-Continue Error Dialog

Abort-Continue Error Dialog (pg. 4 - 19) presents a selection dialog box that describes a database error and then allows the operator to either continue or terminate the database session and LabVIEW execution.

Continue Error Dialog

Continue Error Dialog (pg. 4 - 21) presents a continue dialog box that describes a database error and then continues VI execution.

DBVIEW Interface VIs

These VIs provide the lowest level access to database operation. Details on the VIs may be found in Chapter 5, *Interface VI Reference*, pg. 5 - 1.

Set Fetch Options

Get Fetch Options

Close Fetch Log File

Fetch Next Record

Fetch Previous Record

Fetch Record N

Fetch Column Data

Get Number of Modified Records

Get Number of Records

Get Number of Columns

Get Column Name

Get DatabaseVIEW Column Type

Get Database Column Type

Get Column Width

Get Column Precision

Get Column Scale

Set Database

LabVIEW to SQL Date Format

SQL Date to LabVIEW Format

Utility VIs

The DBVIEW Utility VIs provide general-purpose tools for database application development and operation. Details on these VIs may be found in *Utility VIs*, pg. 6 - 1.

Database Session Monitor VI

Database Session Monitor VI (pg. 6 - 1) provides a convenient method for monitoring the status of active database connections and SQL statements during application development and integration testing.

Administration and On-Line Help Utilities

Administration and On-Line Help Utilities provide additional application development and operational support. These utilities are accessed from the Windows Program Manager via program items in the DatabaseVIEW program group. Details on these utilities may be found in *Administration and On-Line Help Utilities*, pg. 6 - 3.

ODBC Administrator

ODBC Administrator (pg. 6 - 3) is a Microsoft utility that is used to define and maintain ODBC components, including data source names (DSNs) and drivers.

Database Driver Help

Database Driver Help (pg. 6 - 8) is a Windows Help file that provides essential setup, configuration, and performance information for each of the ODBC drivers and their corresponding databases.

Database Driver Read-Me

Database Driver Read-Me (pg. 6 - 8) is a Windows Help file that provides ‘late-breaking’ supplemental information for select ODBC drivers and databases.

Reference Reading List

This section presents a categorized reference list for further reading on LabVIEW; the ODBC databases; database design, use and administration; systems analysis and data modeling; Structured Query Language; and structured development processes.

The DatabaseVIEW development team also strongly recommends, based on extensive experience, that the user have on hand any and all available documentation about the selected database systems.

INTERSOLV/Q+E ODBC Driver Documentation

- DatabaseVIEW Driver Help (a DatabaseVIEW Program Item)
- DatabaseVIEW Driver Readme (a DatabaseVIEW Program Item)

LabVIEW Documentation

• LabVIEW for Windows Users Manual	P/N 320534-01
• LabVIEW for Windows Function Reference Manual	P/N 320535-01
• LabVIEW for Windows Tutorial	P/N 320593-01
• LabVIEW for Windows Master Index	P/N 320597-01
• Other LabVIEW documentation from National Instruments Corp.	

Other LabVIEW Sources

- LabVIEW Graphical Programming
  G. Johnson — ISBN 0-07-032719-X
- LabVIEW Technical Resource Quarterly Newsletter
  L. Gruggett, J. Parker eds. — Phone: 214.827.9931

Databases

- DATABASE - A Primer
  C. Date — ISBN 0-201-11358-9
- PC Magazine Guide to Client/Server Databases
  J. Salemi — ISBN 1-56276-070-X

SQL Language Instructionals

- 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

Database Application Design

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

Open Data Base Connectivity (ODBC) Standard

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

Chapter 2 Environment & Applicability

This chapter introduces the DatabaseVIEW for Windows operating environment. Brief descriptions of LabVIEW, the supported databases, the Open Database Connectivity (ODBC) standard, the Structured Query Language, and network communications are given. Discussion is presented on the conceptual link between LabVIEW and a database data model. Finally, the suitability of incorporating database functionality into various classes of LabVIEW applications is explored.

Introduction to the DatabaseVIEW Operating Environment

The DatabaseVIEW for Windows VI Library provides an ODBC compatible programmatic bridge between the LabVIEW virtual instrument and several popular database systems. It provides LabVIEW developers with full SQL access to database functions and services from the LabVIEW block diagram. The following paragraphs provide basic descriptions of these different software systems.

LabVIEW for Windows

LabVIEW for Windows is a graphical software development and runtime environment that is particularly suited for scientific and engineering applications. Programs developed and executed in LabVIEW are based on the metaphor of *virtual instruments* (see Figure 11, pg. 2 - 2). LabVIEW virtual instruments are software constructions that consist of an on-screen graphical *front panel* with animated manipulatable controls like knobs, switches, and buttons; an executable *block diagram* that graphically expresses the instrument's functionality; and an *icon/connector* that defines data flow paths to/from other VIs. The block diagram is built from icons representing front panel I/O, computational functions, and other virtual instruments; all connected by wires that direct the flow of data among the components.

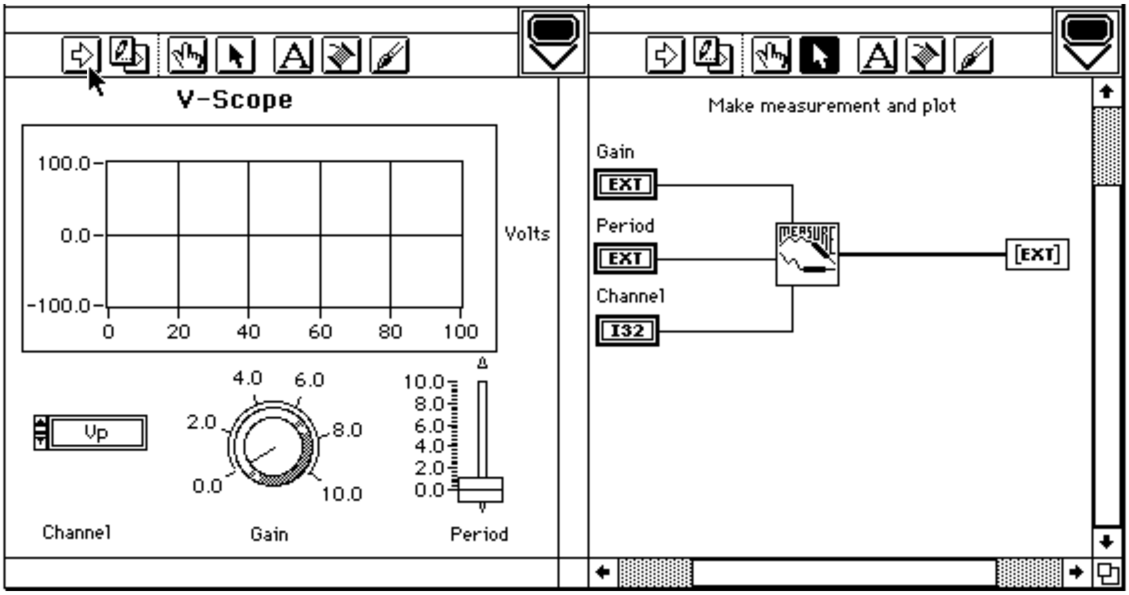


Figure 11. LabVIEW Virtual Instrument

For a complete description of the LabVIEW system refer to the LabVIEW for Windows Users Manual, P/N 320534-01.

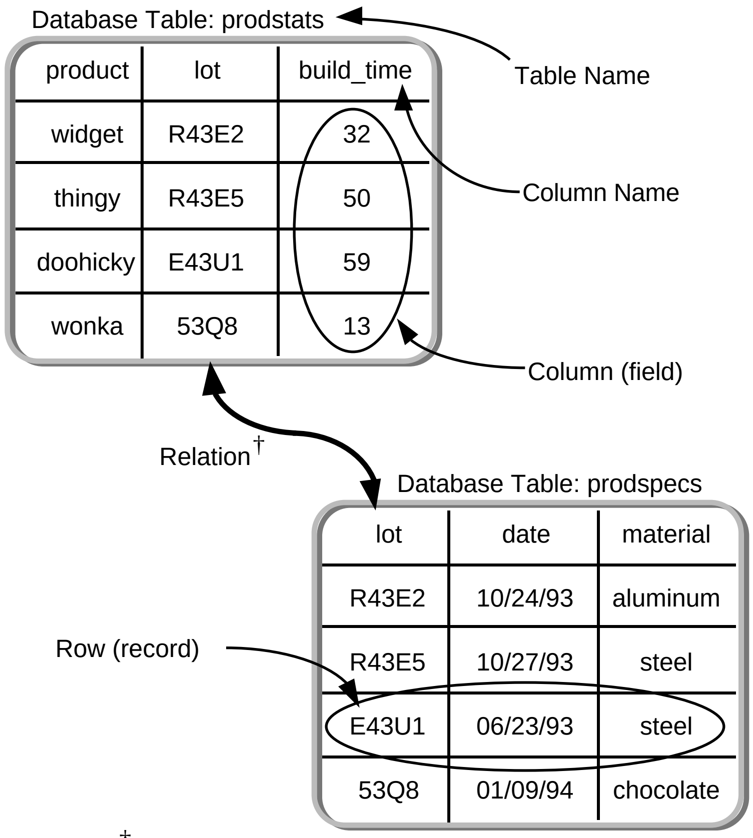
The DatabaseVIEW Databases

DatabaseVIEW for Windows provides access to several popular file-based databases and high-performance Relational Database Management System (RDBMS) software packages that run on a great variety of computers and operating systems. Through specific application programs users can logon to, create, store, modify, retrieve, sort, and otherwise manage data in databases located on the same or networked computers. These functions are performed via commands expressed in the Structured Query Language. A detailed explanation of each supported database is provided in DatabaseVIEW Driver Help and DatabaseVIEW Driver Readme utilities (See *Administration and On-Line Help Utilities*, pg. 6-3).

The DBVIEW VI library specifically allows LabVIEW developers to build application programs that access databases using SQL commands.

Database Models

A database “model” is simply a logical way to view how data is organized and stored in computers. Several of the more popular contemporary database models are based on the notion of databases that contain data structures called tables. Tables consist of rows (or records), which are made up of columns (or fields), as illustrated in Figure 12. Each table in a database has a unique name, and each column within a table has a unique name and data type. Each database supports a variety of column datatypes, commonly including: CHARACTER (fixed and variable length), NUMBER, DECIMAL, INTEGER, FLOAT, REAL, DOUBLE PRECISION, DATE, LONG, and RAW.



Database Table: prodstats

product	lot	build_time
widget	R43E2	32
thingy	R43E5	50
doohicky	E43U1	59
wonka	53Q8	13

Database Table: prodspecs

lot	date	material
R43E2	10/24/93	aluminum
R43E5	10/27/93	steel
E43U1	06/23/93	steel
53Q8	01/09/94	chocolate

[†]Note: not all database systems support relations.

Figure 12. Database Table Concept

DBVIEW supports most underlying database column types directly. Others are supported indirectly by automatically mapping the underlying type into a common set of datatypes. In most cases, this allows a single LabVIEW application to access multiple databases without modification.

Table 2. DBVIEW Data Types

Type Code	Data Type Description
1	fixed length character string
2	variable length character string
3	Binary Coded Decimal (BCD) number
4	4-byte long integer
5	2-byte integer
6	4-byte single precision floating-point
7	8-byte double precision floating-point
8	26-byte date-time character string of the form YYYY-MM-DD HH:MM:SS.SSSSSS

All SQL commands, and therefore column values for INSERTs or UPDATEs to the databases, are represented as LabVIEW character strings. Likewise, all SELECT query data is returned from the databases as LabVIEW strings. Column data may be converted to/from the appropriate LabVIEW data types using LabVIEW's built-in string conversion functions (see the *LabVIEW for Windows Functions Reference Manual*) as well as DBVIEW conversion VIs (*Datatype Conversion VIs*, pg. 5 - 23).

The ODBC Standard

The Open Database Connectivity standard, developed by Microsoft Corp., provides a common method for application access to databases. The standard provides a uniform means of database access that is portable across databases, applications, and platforms. It consists of a multi-level application programming interface definition, a driver packaging standard, a common SQL implementation based on ANSI SQL, and a means for defining and maintaining Data Source Names (DSN). A Data Source Name is essentially a 'shorthand' means to refer to a specific database. It is specified by a unique name, and the ODBC driver that is used to communicate with the physical database (local or remote). A DSN must be defined for each database an application program needs to connect to. DSNs are defined using the *ODBC Administrator*, described on page 6-3.

DatabaseVIEW for Windows VIs and the supplied database drivers comply with the ODBC standard. LabVIEW database applications written with DBVIEW VIs are easily ported across vitrually all of the supported databases with minimal modification.

Database Communication: Local or Remote Connections

A database may be located on the computer running LabVIEW; on a remote computer accessed via network or modem; or a combination of both. A database may be *file-based*, where DatabaseVIEW directly accesses the database disk files to manipulate data; or *enginebased*, where DatabaseVIEW communicates with an executing database program to manipulate data. All database operations must begin by establishing a *connection* to the desired database(s); which consists of selecting a database, a communications link, and, where necessary, logging on with a username and password. The following figures illustrate several possible local and remote database configurations, including the various layers of software needed to implement them (often called a "communication stack").

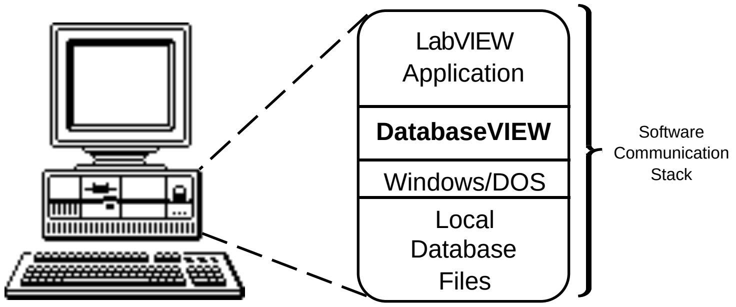


Figure 13. Local File-Based Database Connection

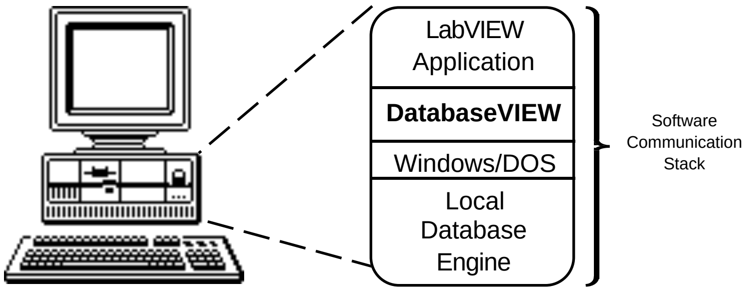


Figure 14. Local Engine-Based Database Connection

Network Communications

There are several network communications packages that allow application programs to communicate with remote database files (file-based) and server (engine-based) databases, over a broad range of local and wide-area network protocols. With the appropriate communications software installed, application programs can access local or remote databases transparently, without any additional programming. The end user simply selects a defined Data Source Name for the desired database into one of the database connection VIs described in Chapter 4. The communications software then transmits the request through the proper network protocol and physical channel.

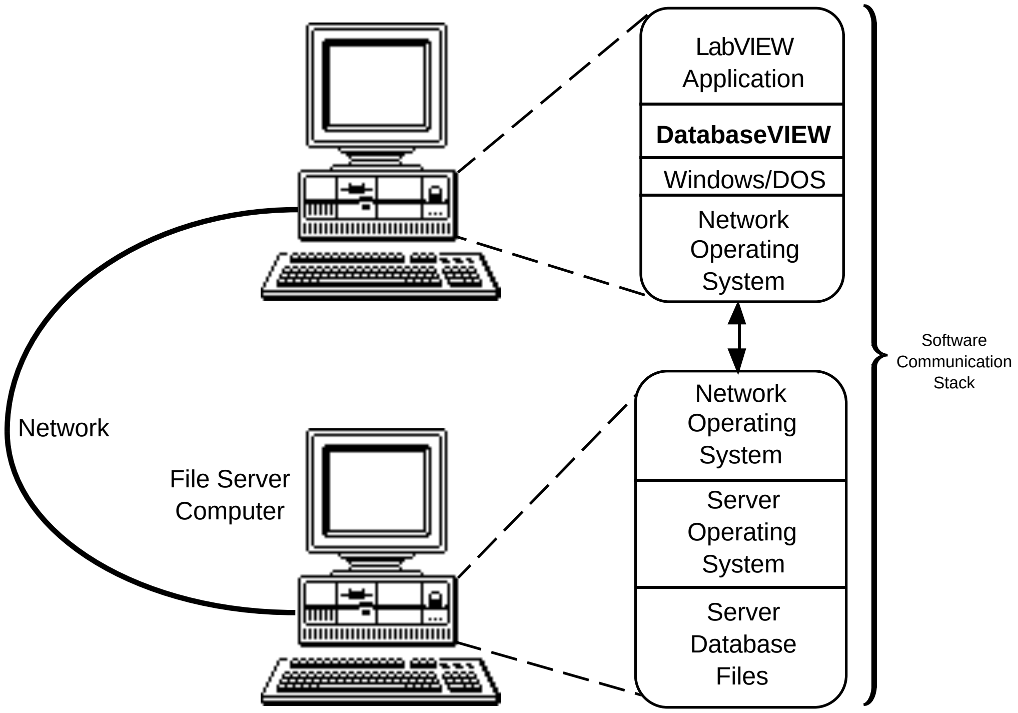


Figure 15. Remote File-Based Database Connection

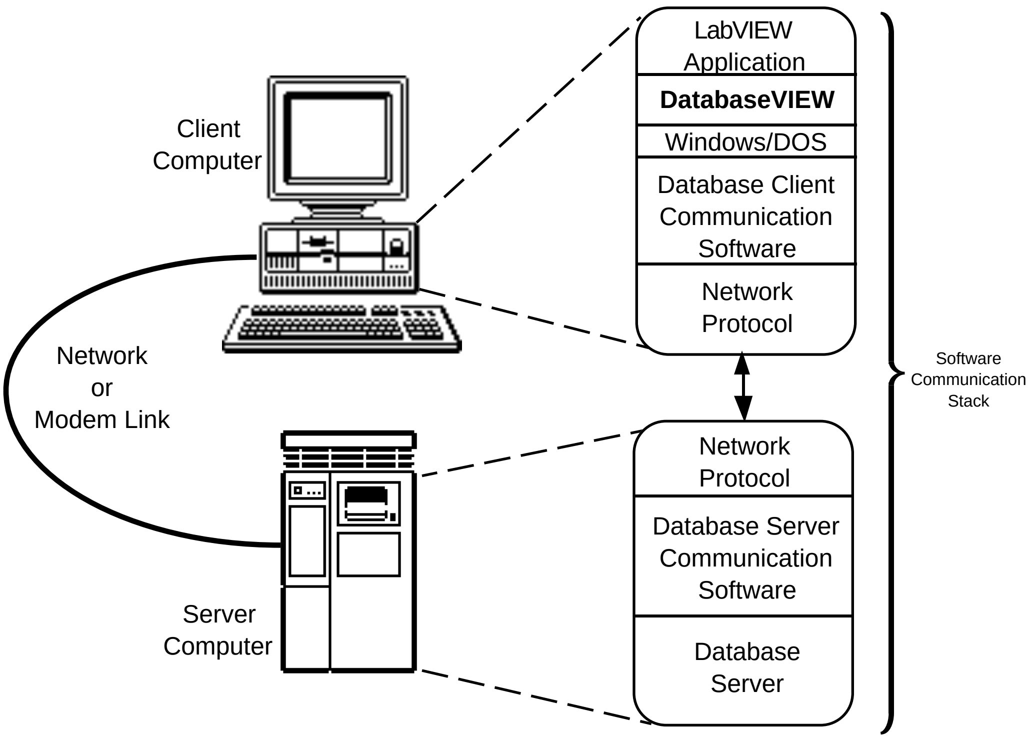


Figure 16. Remote Client-Server Database Connection

The Structured Query Language

The Structured Query Language, or SQL, is a set of character-string commands that all DBVIEW-based application VIs must use to describe, store, retrieve, and manipulate data in accessible databases. The language was initially developed by IBM and commercialized in the late-1970s. SQL has since been adopted in ANSI, ISO, and FIPS standards and is supported to some degree in most major commercial relational database products. It is a non-procedural language that allows the user to refer to and process sets of records in database tables. There are three pertinent classes of SQL statements:

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

Some frequently used SQL commands are:

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.

An example SQL statement (based on the database tables in Figure 12, pg. 2 - 3) might read as follows (SQL keywords are bold):

SELECT product, lot **FROM** prodstats **WHERE** lot=’R43E2’

This statement is interpreted as “retrieve the columns ‘product’ and ‘lot’ from the ‘prodstats’ table, including only those rows where the value contained in the ‘lot’ column equals ‘R43E2’”. The results of this query would be a one row, two column active set as follows:

Table 3. Example Query Results

widget	R43E2

SQL Variants

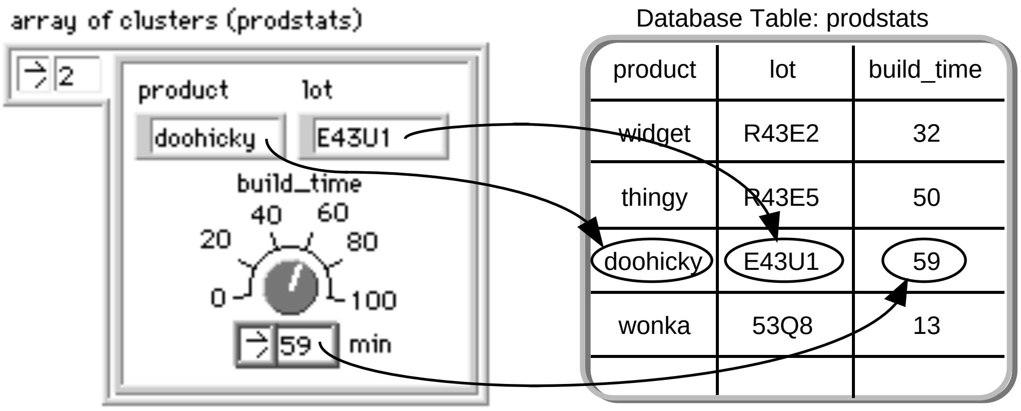
Standards notwithstanding, several database publishers employ their own SQL variants or “dialects” in their products. These variants usually consist of extensions to the standard SQL that perform higher-level or database specific functions. Conversely, certain of the accessible databases do not directly support standard SQL functions. Differences are most prevalent when using CREATE TABLE and ALTER TABLE. Most of the databases use their own particular column type keywords. Also, many of the databases have different syntax for date/time formats. ODBC-compliant DBVIEW software helps to minimize the effects of these SQL variants. Consult Appendix A, *SQL Quick-Reference*, pg. A - 1, and the ‘SQL for Flat-File Drivers’ topic in the DatabaseVIEW Driver Help program item for details on particular databases.

DatabaseVIEW for Windows VIs - Putting it all together

There are two essential facets to integrating software environments: data integration and operational integration. The following sections describe the data and operational basis for LabVIEW - database integration.

Data Integration - The Conceptual Fit

As noted above, the conventional (relational) database model is based on the notion of tables made up of rows and columns. From a LabVIEW perspective, a row in a database table corresponds directly to a *cluster*, and columns in the table corresponds to *elements* within the cluster. Therefore a database table corresponds directly to a one-dimensional array of LabVIEW clusters, as shown in Figure 17, pg. 2 - 10.



product	lot	build_time
widget	R43E2	32
thingy	R43E5	50
doohicky	E43U1	59
wonka	53Q8	13

Figure 17. Conceptual Mapping: LabVIEW Array of Clusters to Database Tables

Functional Integration - The Operational Fit

The cluster metaphor extends beyond data definition and organization, and into data access and manipulation. SQL data operations on rows in a database table correspond directly to LabVIEW operations on clusters in a one-dimensional array of clusters. Table 4, pg. 2-11 illustrates the functional mapping between the more frequently used SQL statements described earlier and the equivalent LabVIEW operations. Note not all of these SQL statements are supported by all accessible databases.

Table 4. SQL to LabVIEW Functional Correspondence

SQL Command	LabVIEW Operation
CREATE TABLE	**edit VI front panel** to define a named 1D array of clusters with specific named cluster element types
INSERT	add cluster elements to a 1D array of clusters using **Bundler** and **Array Builder**
SELECT	use **Search1D array** of clusters in a loop to find all clusters whose elements satisfy multiple inter-related conditions
UPDATE	use **Search1D Array** as in SELECT, then use **Bundler** and **Replace Array Element** in a loop to to change specific element values
DELETE	use **Search1D Array** as in SELECT, then use **Array Subset** and **Array Builder** in a loop to eliminate each unwanted cluster

Note that the last three of these equivalent LabVIEW functions would require that the entire one-dimensional array of clusters reside in memory so that the search/modify operations could be performed on each cluster that satisfies the conditions. This would be impossible on large datasets that could not fit into memory all at once.

Applicability - To Database or Not to Database...

In general, each LabVIEW application must satisfy a unique set of data management requirements. Therefore, each must be evaluated to determine the applicability of a database. Specifically, the case must be made that a database provides data management facilities that are either unavailable or impractical within the scope of a LabVIEW development effort. Alternately, a case can be made that some applications would derive no benefit, and may in fact incur a performance penalty through a database implementation.

LabVIEW Data Management

LabVIEW itself provides limited facilities for data persistence: front panel logging, file data logging, and ordinary disk file storage. Each of these methods will preserve the subject data and may therefore be appropriate for some application tasks. Truly comprehensive data management, sharing, and retrieval, however, can require significant block diagram programming. In particular, data access control, searching, and sorting functions are not provided directly on arrays of clusters in LabVIEW. Also, the LabVIEW data files, whether log files or the VIs themselves, can often grow tremendously large, and thereby become difficult to manage. Finally, the original datalog type specifier is required to retrieve the logged data, which means that the data is difficult to share and is only accessible to other LabVIEW applications.

The Database Advantage

Databases were developed specifically to provide access control, and rapid search, retrieval, and sorting of stored data. The following paragraphs describe several common features that some databases provide for improving data management and retrieval.

- Indexing - Many, but not all, databases employ a logical device called an *index* to accelerate the search process. Indices are used to help speed queries that search for particular values in “key” (often used) columns. The database accomplishes this by maintaining a separate map of the values in the key field. The maps can be searched very quickly to identify which rows in the main table satisfy the query conditions.

- Data Dictionary - Many, but not all, databases employ a logical device called a *data dictionary* to track the structure of database tables. Descriptions of the columns and rows in database tables are automatically maintained in the data dictionary and are often available through ordinary queries.

- Concurrent Access - Many, but not all, networked databases support many concurrent client connections, thereby allowing several locations to work virtually simultaneously on the same data tables.

- Transactions - Data may be added to, altered in, and deleted from database tables, all within the context of *transactions*. Transactions ensure that database contents are not left in an inconsistent state if/when failures occur.

- Access Control - Some database systems employ extensive security measures to provide access control, buffering, error checking, and recovery capabilities.

- Relational Joins - Many, but not all, relational databases permit data searches to be applied across multiple related database tables simultaneously.

- Open Access - There are literally thousands of existing non-LabVIEW applications that can access data stored in many popular database tables, thereby allowing open access to important information.

Time Series vs. Summary Data

Many LabVIEW applications create time series data, i.e. measurement samples of physical phenomena taken at regularly spaced intervals. The information in these data are often extracted through temporal analysis, thus the data is informative only when used collectively in the (time ascending) order it was sampled. Experience has shown that the substantial benefits of using a conventional database are derived from the random access search, retrieval, and sort capabilites of the database engine. Therefore, applications that have significant search, retrieval, and sorting demands stand the most to gain from a database implementation. Because time series data has an implicit sort order associated with the sampling process, little benefit would be gained from storing high-rate time series data in a database.

A database will, however, benefit such applications when used to store summary information about multiple time series taken during different test sessions as well as the file location of the original raw (time sample) data.

Chapter 3 Using DatabaseVIEW VIs

This chapter describes how to use the DatabaseVIEW for Windows VIs to achieve high-level SQL access to databases with a minimum of programming. It begins with a general discussion of database operations from a programmer's perspective, and includes descriptions of how to implement session and transaction operations using the DBVIEW VIs.

General Database Operation

It is essential to have a basic understanding of database operations so the DBVIEW VIs can be applied effectively. A detailed understanding of database operations becomes particularly important when attempting to optimize application and database performance. The following paragraphs discuss basic operations that all database users must be familiar with.

The reader is referred to the selected database vendor documentation, where available, and the DatabaseVIEW Driver Help for additional database-vendor-specific information.

The Database Session

All operations on a database, whether local or remote, occur within the context of a *session*, depicted in Figure 18. (Please note that the session is depicted here as a sequence structure only to emphasize the required operation precedence). DBVIEW maintains all session information in the logical data structures illustrated in Figure 19, pg. 3 - 2 and described in the following sections.

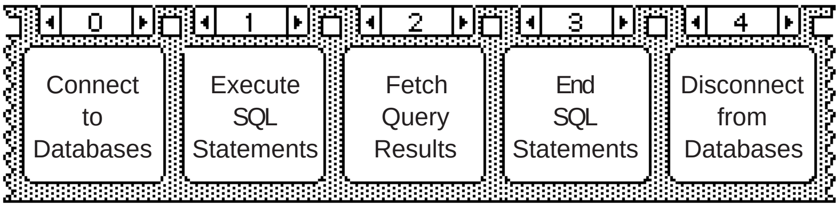


Figure 18. The Database Session Sequence

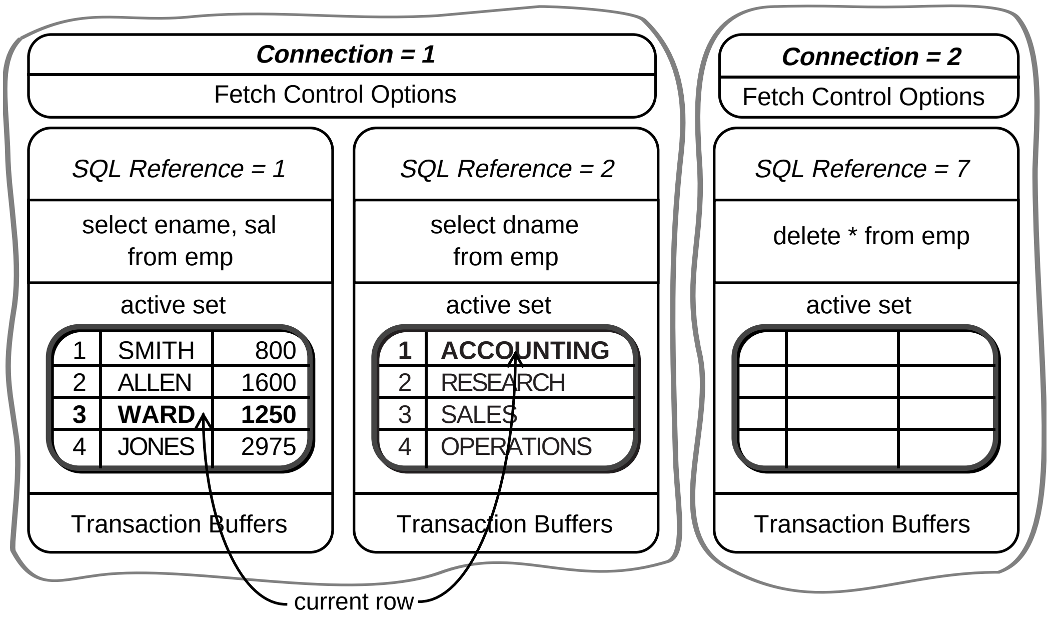


Figure 19. DatabaseVIEW Logical Data Structures

Establishing a Connection to a Database

A database session is initiated by establishing a *connection* to the desired database using the Access VI **Connect** (pg. 4 - 2). A connection is simply a logical channel to a database that has been registered as an ODBC *Data Source*. Data sources are identified by a user-specified Data Source Name (DSN), which fully specify the database to connect and logon to, and, when connecting to a remote database, the network channel and protocol to communicate through. Data sources are defined and maintained by using the ODBC Administrator program item, described in *ODBC Administrator*, pg. 6 - 3.

For example, the dBASE data source name used in the DBVIEW Example VIs consists simply of “DBV DEMO DBASE”. Alternately, DBVIEW applications may present the user with one or more dialog windows to prompt for the desired data source name and any non-default connection parameters.

Once an active connection is established, **Connect** automatically generates a unique identifier at its *Connection Reference* output terminal. *Connection Reference* must be wired as input to all subsequent VIs on the block diagram that wish to operate on this data source. The connection will remain active until terminated by **Disconnect** (see below).

The *Connection Reference* operates in much the same manner as the **taskID** in LabVIEW data acquisition VIs.

Multiple connections to the same or different databases may be active concurrently, when supported by the selected databases.

Executing SQL Statements

Once a connection has been established, SQL statements can be executed to store, manipulate, query, and otherwise control data in database tables. This is accomplished by executing the Access VI **Execute SQL** (pg. 4 - 9). DatabaseVIEW handles SQL statements in two distinct ways:

- DBVIEW SQL - when using non-SQL databases, including all file-based databases, DBVIEW itself parses and executes SQL statements. In these cases, the user is restricted to using only DBVIEW supported SQL statements.

- Database SQL - for most engine-based databases, DBVIEW transmits the SQL statements to the underlying database's SQL interpreter for parsing and execution. In these cases, the user may use whatever SQL the underlying database supports.

SQL statement execution actually consists of two parts:

- Parsing - DBVIEW (or the underlying database) checks the SQL statement for correct syntax; confirms that the referenced database objects (databases, tables, and columns) actually exist; and ensures that the current user has been granted access to the objects.
- Execution - DBVIEW (or the underlying database) executes the statement and thereby performs the specified data action.

DBVIEW VIs group the SQL parse and execute parts into one unified operation.

Once a SQL statement has been executed, **Execute SQL** automatically generates a unique identifier at its *SQL Reference* output terminal. *SQL Reference* must be wired as input to all subsequent VIs on the block diagram that wish to operate on this SQL statement. The SQL statement will remain active until terminated by **End SQL**(see below).

> *SQL Reference* (like *Connection Reference)* operates in much the same manner as the **taskID** or file reference number in LabVIEW data acquisition VIs.

Multiple SQL references (executed SQL statements) may be active concurrently on one or more database connections (where supported by the underlying database) however, a SQL reference cannot be shared between connections.

While many SQL databases impose different requirements on SQL syntax and execution, these differences are minimized by DBVIEW's ODBC-compliant design. Consult DatabaseVIEW Driver Help for details on SQL for the selected database.

Fetch SELECT Results

As noted earlier, the result of executing a SQL SELECT statement is an *active set* of rows that satisfy the conditions specified in the WHERE clause. These rows must then be retrieved from DBVIEW for use in the LabVIEW block diagram. Rows are fetched one-at-a-time from a SQL reference. The most recently fetched row is called the *current row* (see Figure 19, pg. 3 - 2).

Rows are typically fetched sequentially from the active set, thereby preserving the order specified in the optional ORDER BY clause. However, the rows may be fetched randomly by setting the appropriate *fetch control options* as described in *Fetch Option VIs*, pg. 5 - 1. Rows and columns may be fetched individually using the VIs described in *Record and Column Fetch VIs*, pg. 5 - 6. Alternatively, the entire active set may be retrieved all at once (subject to available memory) into LabVIEW by using the Access VI **Fetch Query Results** (pg. 4 - 10).

All column values are returned as LabVIEW strings, which may then be converted into the desired datatypes using LabVIEW's built-in string functions. Additionally, string - date conversions may be performed using the **SQL Date to LabVIEW Date Format** (pg. 5 -

23) and **LabVIEW Date to SQL Date Format** (pg. 5 - 24) VIs. Fetch operations must be referred to previously executed SQL operations via the *SQL Reference* control.

End SQL Statements

When all operations on a SQL reference have been completed, the reference may either be reused by executing another SQL statement, or disposed using **End SQL** (pg. 4 - 12). **End SQL** deactivates the SQL reference, frees-up all data structures associated with it, and disposes any fetched data in the active set. This is particularly important to efficient memory use.

Disconnect From the Database

A database session is concluded when the application program disconnects from a particular database. This is accomplished by using the Access VI **Disconnect** (pg. 4 - 13).

Complete SQL Session - the “One-Step” SQL VI

DBVIEW provides a single all-in-one Access VI, **Complete SQL Session** (pg. 4 - 5), that performs all of the operations in a database session. The VI connects to the selected database; executes a SQL statement; fetches SELECTed rows and columns (if any); ends the SQL statement; and disconnects from the database. This VI is especially suited to testing prototype SQL statements (see *Example 1 - Quick SQL*, pg. 7 - 9). Figure 20, pg. 3 - 6 shows a simple but complete query block diagram using **Complete SQL Session**.

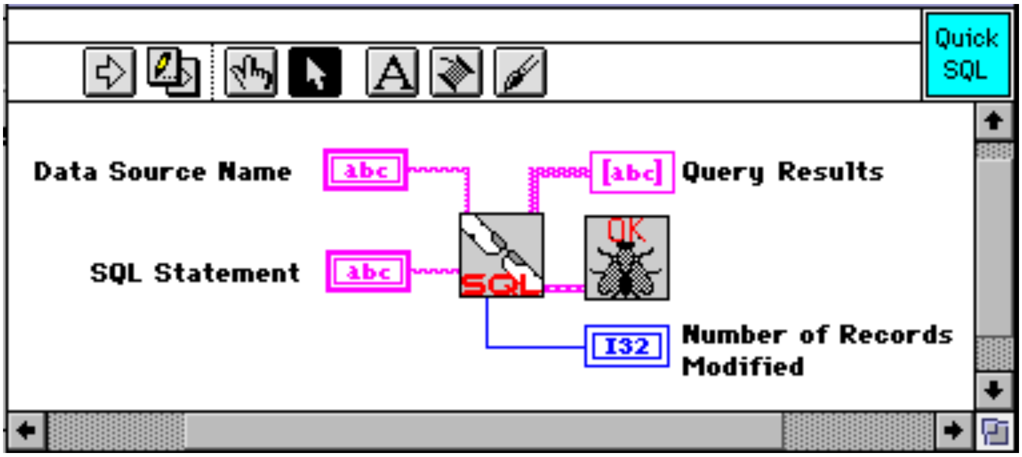
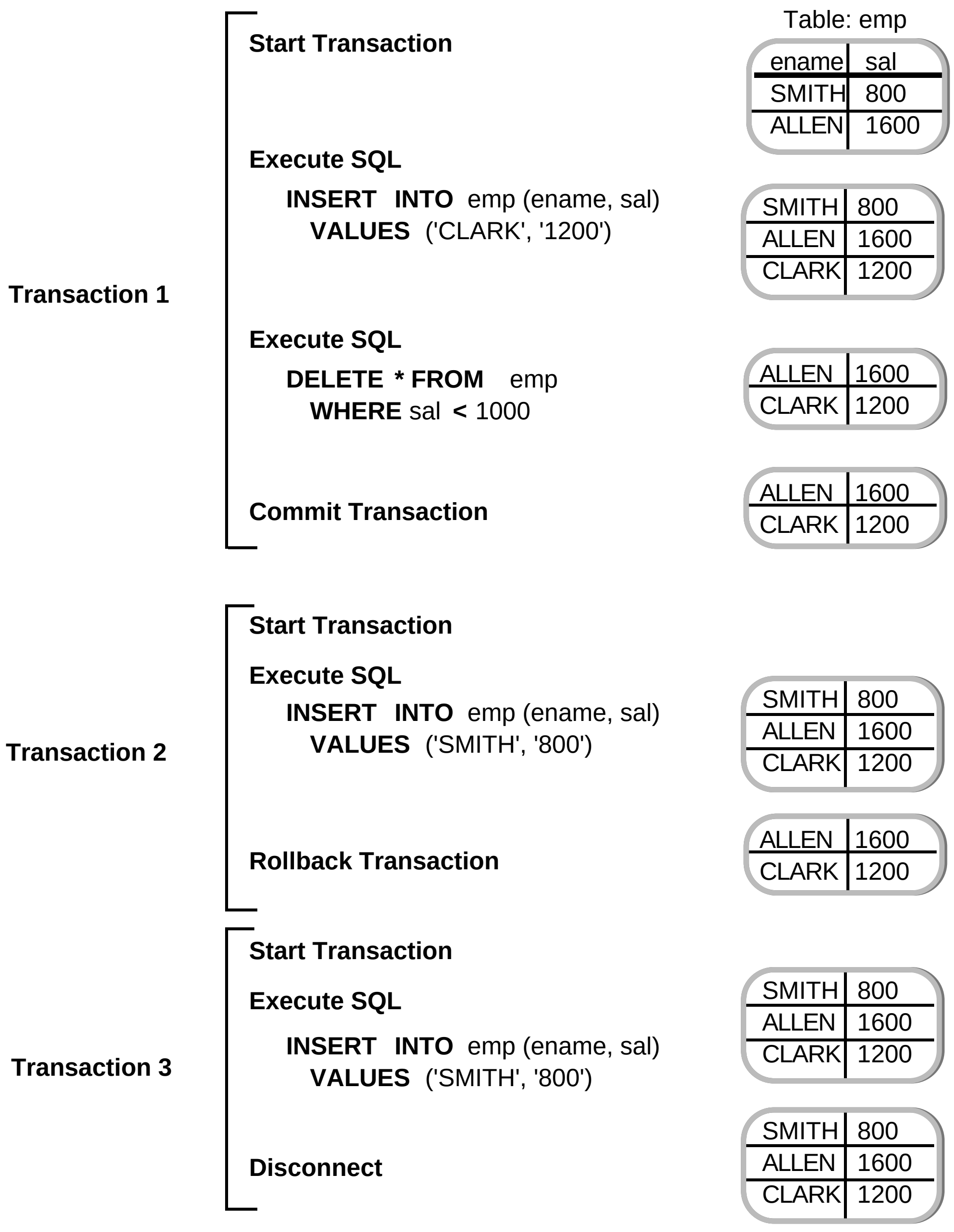


Figure 20. Example Block Diagram Using Complete SQL Session

Database Transactions

A database transaction is a related sequence of data manipulation language (DML) SQL statements that must be treated as an indivisible unit to maintain data consistency. *Transaction Operation VIs*, pg. 4 - 14 are provided that allow LabVIEW applications to apply groups of row changes to the database collectively. This is especially useful where there is a possibility of a session being interrupted when related data in multiple tables is in an inconsistent state. Each database supports transactions in a slightly different way. Consult Database Driver Help (pg. 6 - 8) for details.

A transaction (Figure 21, pg. 3 - 7) consists of all DML statements that have executed since the last transaction was completed. A transaction begins immediately upon executing **Start Transaction** (pg. 4 - 14). Transactions are completed by executing either **Commit Transaction** (pg. 4 - 15) or **Rollback Transaction** (pg. 4 - 16). When **Commit Transaction** is executed, all DML operations in the current transaction are made permanent to the database; while **Rollback Transaction** discards all DML operations in the current transaction. (Note: some, but not all, databases support “auto-commit on disconnect”, illustrated in “Transaction 3” of Figure 21).



```
INSERT  INTO  emp (ename, sal)
  VALUES  ('CLARK', '1200')
```

```
DELETE  * FROM   emp
  WHERE sal  < 1000
```

```
INSERT  INTO  emp (ename, sal)
  VALUES  ('SMITH', '800')
```

```
INSERT  INTO  emp (ename, sal)
  VALUES  ('SMITH', '800')
```

Table: emp

ename	sal
SMITH	800
ALLEN	1600

SMITH	800
ALLEN	1600
CLARK	1200

ALLEN	1600
CLARK	1200

ALLEN	1600
CLARK	1200

SMITH	800
ALLEN	1600
CLARK	1200

ALLEN	1600
CLARK	1200

SMITH	800
ALLEN	1600
CLARK	1200

SMITH	800
ALLEN	1600
CLARK	1200

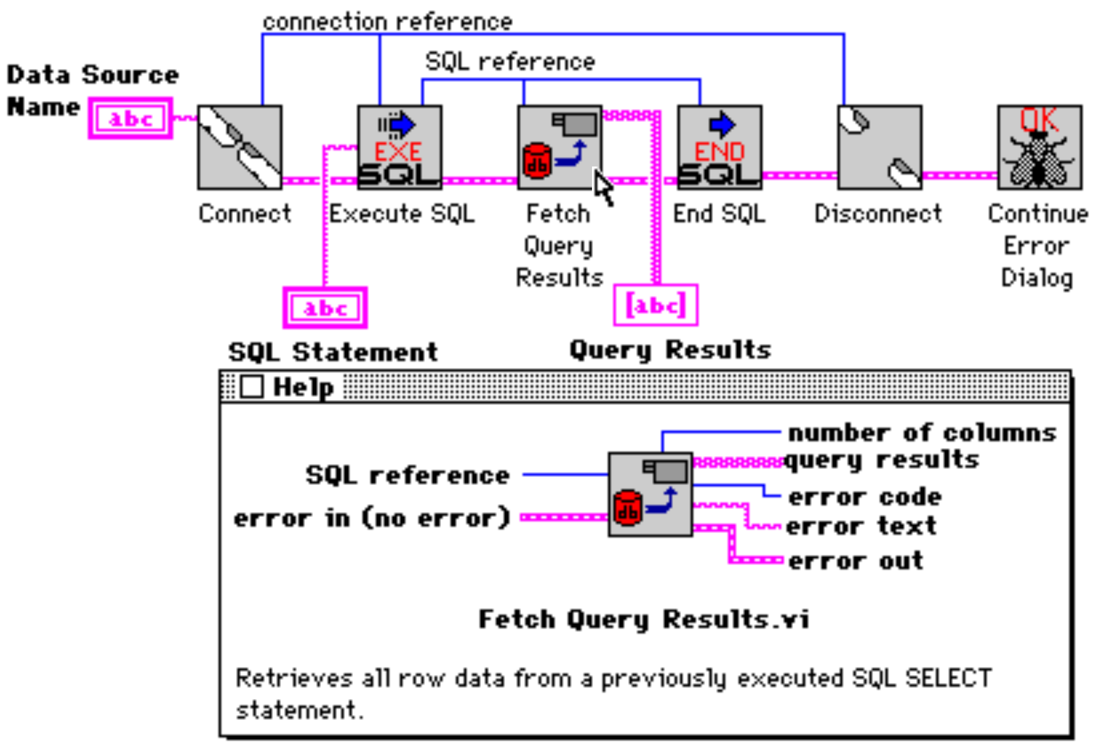
Figure 21. Database Transactions

Sequencing DatabaseVIEW VIs

Operation sequence is very important when implementing a database session: a connection must be active before a SQL statement can be executed; a SQL statement must be executed before data can be fetched; etc. This required sequencing could be accomplished easily using LabVIEW sequence structure. However, sequence structures are inherently “non-dataflow”, and experience has proven that they can be cumbersome to work with. A more efficient method is to implement a session sequence using dataflow dependencies among the VIs.

DBVIEW VIs have been designed and optimized to simplify dataflow programming on the block diagram. All DBVIEW VIs that are sequence dependent have been implemented with explicit sequencing controls and indicators, namely *error in* and *error out*. These cluster controls/indicators may be wired in a modified daisy-chain fashion as shown in Figure 22, pg. 3 - 9. As each VI completes, it produces *error out*, which propagates to the *error in* terminal of the subsequent VI and enables it to execute. This sequencing technique, while optional, greatly simplifies block diagram programming and improves flow visibility.

In addition to sequencing, DBVIEW VIs that are daisy-chained perform error checking and conditional execution that is directly compatible with LabVIEW’s DAQ VIs. This will be explained further in *Database Error Handling*, pg. 3 - 10.



Retrieves all row data from a previously executed SQL SELECT statement.

Figure 22. Sequencing DBVIEW VIs Using *error in* and *error out*

DBVIEW On-Line Help

DatabaseVIEW for Windows provides extensive on-line help information to assist the user in building database application VIs. The on-line help system provides specific details for each of the supported databases, including: system operating requirements, configuring data sources, using connection parameters and dialog windows, managing data dictionary information, column datatypes, and error codes. Detailed information is also provided on SQL language construction syntax for the Flat-File Driver, which is used for all file-based databases. DatabaseVIEW Driver Help uses the standard Windows help system, and is conveniently accessed by using the DatabaseVIEW Driver Help and DatabaseVIEW Driver Readme program items in the Windows program group.

Database Error Handling

DBVIEW VIs return numeric error codes and their textual interpretations indicating the results of all database operations. The error messages are either generated by DBVIEW itself or returned, unaltered, from the underlying database. In general, successful completion of an operation is indicated by error code = 0. Error results may be reported to the user through the *Error Handling VIs*, pg. 4 - 17.

As noted earlier, DBVIEW VIs provide *error in* and *error out* terminal clusters (Figure 23) that are directly compatible with LabVIEW's DAQ VIs. The conditional execution logic, based on the *status* terminal boolean, is as follows:

> If *status* is TRUE, then an error has occurred in a prior VI. This VI simply passes *error in* contents to *error out*, and DOES NOT PERFORM ANY DATABASE FUNCTIONS.
>
> If *status* is FALSE, then no prior error is indicated. This VI PERFORMS ITS NORMAL DATABASE FUNCTION, and reports execution results into *error out* .

The *code* terminal in the *error in* cluster is the numeric result code from a prior VI. *text* contains the name of the VI that reported the error.

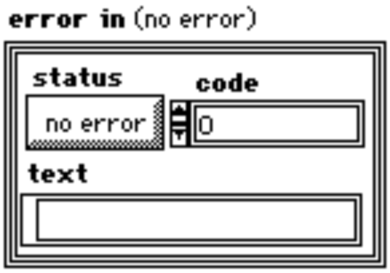


Figure 23. *Error In* Cluster Control

For LabVIEW DAQ VIs, the *code* value is often used to 'look-up' the corresponding text that describes the error condition. Due to the diverse multi-vendor nature of ODBC environments, however, it is impossible to know apriori all error code values and the corresponding text messages. Indeed, the same error code value will almost certainly result in different interpretation from vendor to vendor. Fortunately, the ODBC standard requires that compliant drivers ALWAYS return error text at the time of execution. DBVIEW VIs, in turn, always append this error text to the *text* value in the error out cluster at execution time, thereby enhancing the description of the error condition. This extention to National Instruments's error reporting mechanism remains fully compatible with the DAQ VIs, so that DBVIEW VIs may be wired directly into DAQ daisy-chains as shown in Figure 24.

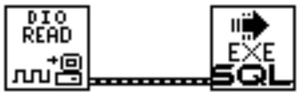


Figure 24. DAQ and DBVIEW Common Error Handling Mechanism

ADVANCED TOPIC: Optimizing Database Sessions

Substantial opportunities exist to optimize database performance through proper design practice. In particular, session performance can be improved by careful selection of combinations of Session VIs. The goal is to minimize message transfers and streamline query operations between the application VI and the target database. Message transfers occur whenever a session or transaction VI is executed, and when query operations occur during certain SQL statements. To reduce database processing and communication overhead, there are several rules that can be applied, including the following:

- Develop efficient SQL statements

  There are often several different SQL statements that yield the same results, however, some SQL statements require substantially less block diagram programming and/or database processing time than others. *Example 5 - SQL Optimization Demo*, pg. 7 - 21 illustrates the benefits of optimized SQL.

- Use Indexes

  Several of the supported databases may be instructed to maintain indexes, or pointers, to values in selected columns that dramatically speed up certain searches on the selected columns. Index capabilities for each of the supported databases are described in DatabaseVIEW Driver Help.

It should be noted that more thorough application performance optimization will require the use of low-level VIs described in Chapter 5, *Interface VI Reference*, pg. 5 - 1.

For a complete discussion on performance optimization for a particular database, refer to that database's documentation and to the DatabaseVIEW Driver Help.

Chapter 4 Access VI Reference

This chapter provides a detailed description for the Access VIs, which are used to perform high-level database operations. There are three categories of VIs: Session Operation, Transaction Operation, and Error Handling. Many of the Access VIs are built from the low-level Interface VIs described in Chapter 5, *Interface VI Reference*, pg. 5 - 1.

These VIs may be found in the ACCESS.LLB library, typically installed in the \LAB-VIEW\DBVIEW.LIB\DBVIEW directory.

Some of the VIs have been set-up in LabVIEW to open their front panels upon being called; much in the fashion of an operating system dialog window that prompts the user and waits for an interactive response. These are designated by the word "Dialog" in the VI name.

Session Operation VIs

Session Operation VIs perform one or more parts of the database session. They are used to connect to and execute SQL statements against selected databases. Some of the VIs may be used individually, while others must be used in combination to achieve database activity. Each VI description is annotated with a session symbol that indicates at-a-glance which parts of the database session are performed therein (see Figure 25).

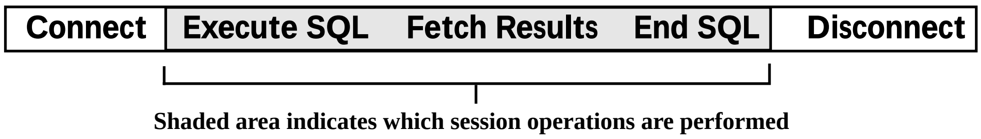
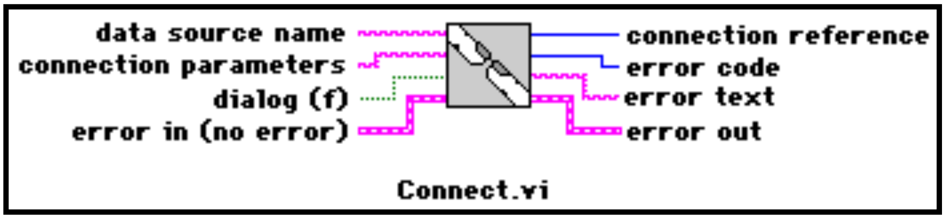

Connect
Execute SQL
Fetch Results
End SQL
Disconnect
Shaded area indicates which session operations are performed

Figure 25. Session Symbol

Connect

Connect establishes an active connection to the selected database.



Data Source Name specifies the name of a defined ODBC Data Source to connect to.

Connection Parameters specifies additional and/or override database-specific parameters to apply to the connection.

Dialog (F) displays a standard dialog window and allows the user to select a particular data source to connect to from a list of all currently defined data sources.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Connection Reference returns a unique identifier for the specified connection that must be used in subsequent operations.

Error Code is the result code from the operation.

Error Text is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Connect attempts to establish and activate a connection to the selected data source, and if successful, begins a database session. Named data sources are defined and maintained through the ODBC Administrator program item (see *ODBC Administrator*, pg. 6 - 3).

The Connection Parameters control is used to specify additional connection parameter values not specified in the data source definition and/or to override (replace) specified parameters for the current session. Allowed connection parameters are database specific (see DatabaseVIEW Driver Help for details on your selected database). For example, the DBVIEW examples data source, DBV DEMO DBASE, has the dBASE-specific parameter 'Database Directory' defined as C:\LABVIEW\EXAMPLES\DBVIEW_EXMPLDB, the default directory where the example database tables are installed. To override this, set the Connection Parameters control to:

DB=*c:\some\other\path\name*

This new database directory path overrides the data source definition for the current session only.

The Dialog control is used to allow the application user to select the desired data source and other (optional) connection parameters interactively. When a Data Source Name is not specified and Dialog is set to True, a data source selection dialog window is presented to the user as shown in Figure 26. The user clicks on and highlights the desired Data Source Name, and then clicks on OK.

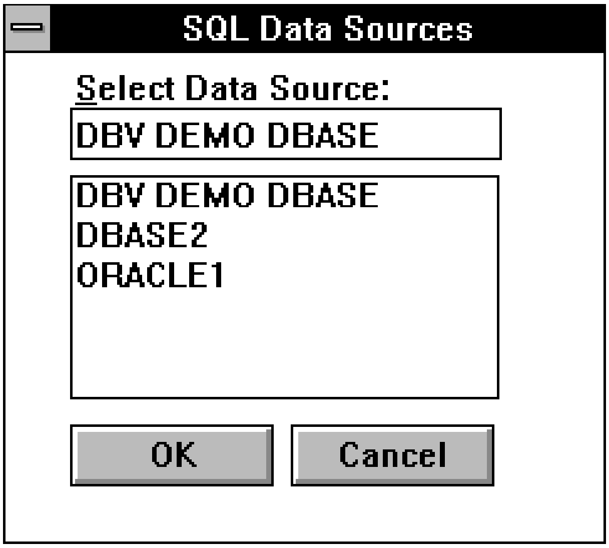
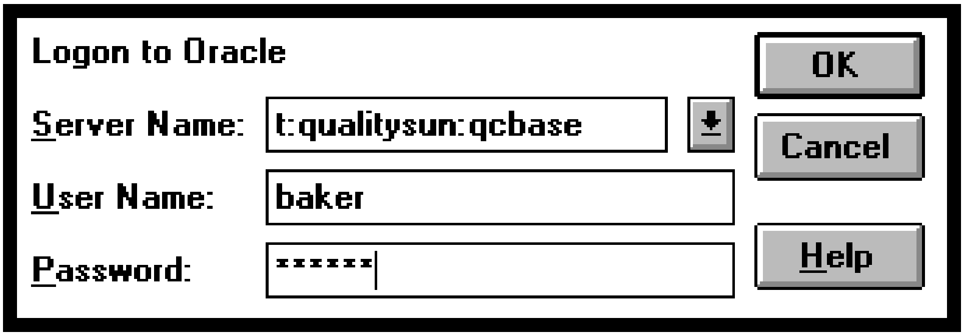


Figure 26. Data Source Selection Dialog Window

Some databases, for example ORACLE, allow multiple connection/logon options to be associated with a single named data source. When applicable, a database-specific selection dialog window is subsequently presented automatically as shown in Figure 27. In this case, the dialog window allows the user to specify his User Name and Password, and to select a specific ORACLE server, as shown in Figure 27.

Another method is to specify a defined Data Source Name and set Dialog to True. This will cause the database-specific dialog window to be displayed directly, bypassing the data source selection window.

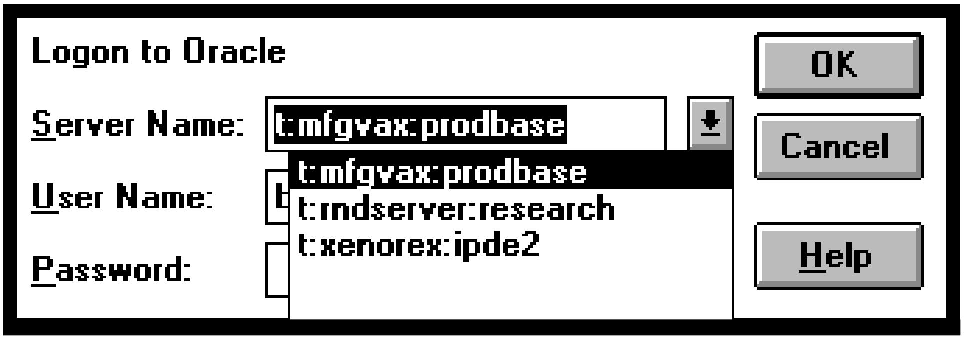

Logon to Oracle

Server Name: t:qualitysun:qcbase

User Name: baker

Password: ******

Figure 27. ORACLE-Specific Logon Dialog Window


Logon to Oracle

Server Name: t:mfgvax:prodbase

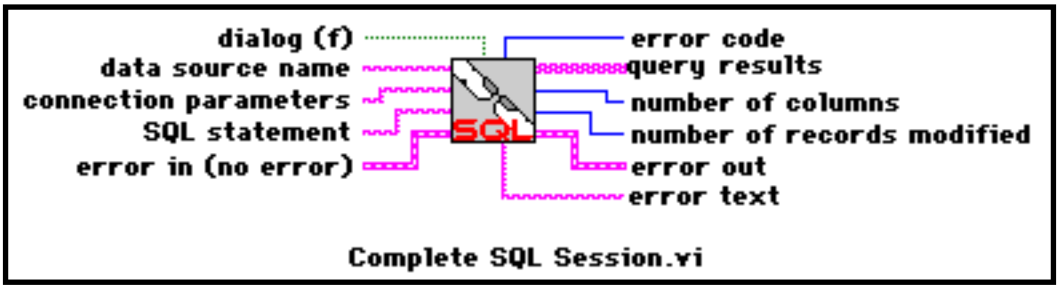
t:mfgvax:prodbase
t:rndserver:research
t:xenorex:ipde2

Figure 28. Selecting an ORACLE Server

Connect	Execute SQL	Fetch Results	End SQL	Disconnect

Complete SQL Session

Complete SQL Session performs a complete database session.



Data Source Name specifies the name of a defined ODBC Data Source to connect to.

Connection Parameters specifies additional and/or override database-specific parameters to apply to the connection.

SQL Statement specifies the desired SQL operation.

Dialog (F) displays a standard dialog window and allows the user to select a particular data source to connect to from a list of all currently defined data sources.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Query Results returns the results of a SQL SELECT statement in a two-dimensional array of strings. Other SQL statements return no data.

Number of Columns returns the number of columns that were returned by the referenced SQL SELECT operation.

No. Modified Records returns the number of records, if any, that were modified by the referenced SQL operation.

Error Code is the result code from the operation.

Error Text is a textual interpretation of the error code, where available.

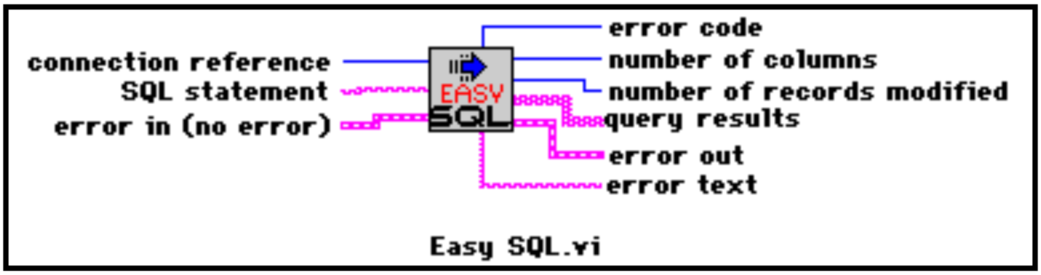
Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Complete SQL Session performs all steps of a database session. It establishes an active connection to the specified database, begins a transaction, executes the SQL statement, retrieves SELECTed data (if any), concludes the transaction, and then disconnects from the database. The transaction is always rolled-back if an error is encountered on SQL statement execution, and always committed if rows are modified and no SQL error is encountered. The SQL statement must be supported by the selected database (see DatabaseVIEW Driver Help).

Connect	**Execute SQL**	**Fetch Results**	**End SQL**	**Disconnect**

Easy SQL

Easy SQL performs a SQL operation on a connected database, and returns any SELECT query results.



I16 **Connection Reference** specifies an existing database connection number to use to perform the SQL operation.

abc **SQL Statement** specifies the desired SQL operation.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

[abc] **Query Results** returns the results of a SQL SELECT statement in a two-dimensional array of strings. Other SQL statements return no data.

I32 **Number of Columns** returns the number of columns that were returned by the referenced SQL SELECT operation.

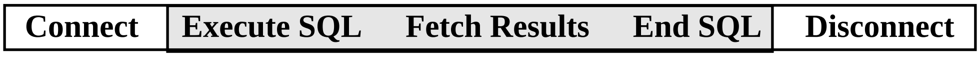
I32 **Number of records modified** returns the number of records, if any, that were modified by the referenced SQL operation.

I32 **Error Code** is the result code from the operation.

abc **Error Text** is a textual interpretation of the error code, where available.

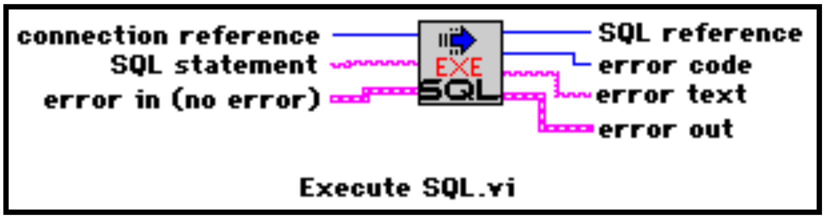
Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Easy SQL performs all steps of a database session except connecting and disconnecting. It requires that an active connection be established with a database, either local or remote, prior to execution. Easy SQL causes the specified SQL statement to be executed on the referenced database connection. The SQL statement must be supported by the selected database (see DatabaseVIEW Driver Help). If a SQL SELECT statement is executed, the VI attempts to return all resulting data (subject to available memory) into a two-dimensional array of LabVIEW strings for further processing.



Execute SQL

Execute SQL executes a SQL statement on the selected database.



I16 **Connection Reference** specifies an existing database connection number to use to perform the SQL operation.

abc **SQL Statement** specifies the desired SQL operation.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I16 **SQL Reference** returns a unique identifier for the specified SQL statement that must be used in subsequent operations.

I32 **Error Code** is the result code from the operation.

abc **Error Text** is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

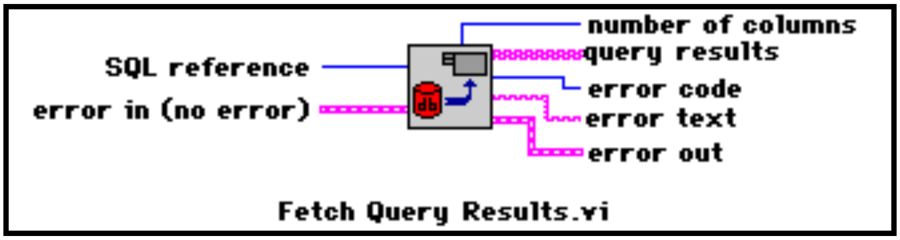
This VI performs only the execute segment of the SQL operation. It does not return any data. A database connection must be active prior to its execution. Once Execute SQL has completed, any data rows resulting from a SQL SELECT statement must be fetched using, for example, **Fetch Query Results** (pg. 4 - 10).

Execute SQL is best suited for performing SQL operations that return no data, or performing multiple SQL operations with changing SQL statements, e.g. within a LabVIEW loop.

Connect | **Execute SQL** | **Fetch Results** | **End SQL** | **Disconnect**

Fetch Query Results

Fetch Query Results retrieves all row data from the active set of a previously executed SQL SELECT statement.



I16 **SQL Reference** specifies the previously executed SQL SELECT operation number to fetch query results from.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

[abc] **Query Results** returns the entire results of a SQL SELECT statement in a two-dimensional array of strings, given sufficient memory.

I32 **Number of Columns** returns the number of columns that were returned by the referenced SQL operation.

I32 **Error Code** is the result code from the operation.

abc **Error Text** is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

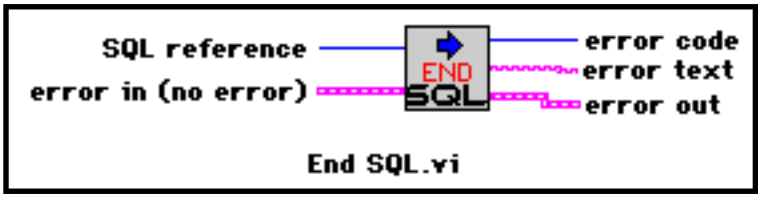
Fetch Query Data attempts to retrieve <u>all data rows</u> resulting from the active set of a SQL SELECT, subject to available memory. A SQL statement must have been successfullly executed prior to executing this VI. Row data is moved from the referenced SQL operation into a two-dimensional array of LabVIEW strings. To fetch data rows and columns individually use the VIs described in *Record and Column Fetch VIs*, pg. 5 - 6.

Once query data has been fetched, the columns may be extracted and converted from string to other data types using LabVIEW's built-in string conversion function. Consult the *LabVIEW Function Reference Manual* for details. Conversion from/to the DBVIEW date type may be performed using **SQL Date to LabVIEW Date Format** (pg. 5 - 23) and **LabVIEW Date to SQL Date Format** (pg. 5 - 24).

Connect	**Execute SQL**	**Fetch Results**	**End SQL**	**Disconnect**

End SQL

End SQL terminates the specified SQL reference execution.



I16 **SQL Reference** specifies a previously executed SQL operation number. Set to -1 to terminate all currently active SQL operations.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I32 **Error Code** is the result code from the operation.

abc **Error Text** is the textual interpretation of the error code, where available, from the operation.

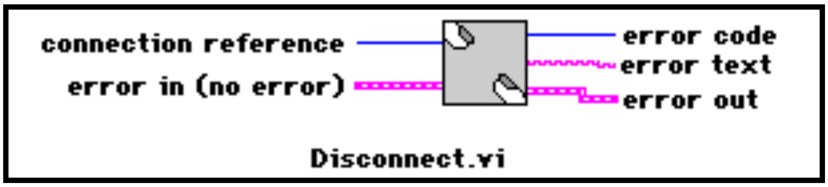
Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

End SQL is used to deactivate the SQL reference and clean-up after a SQL operation has been completed, including all row and column fetches. All local DBVIEW and remote database system resources are released, and query results, if any, are disposed. It is particularly important to execute End SQL when random fetch options have been enabled so as to properly close and delete any temporary log files.

Connect	Execute SQL	Fetch Results	End SQL	Disconnect

Disconnect

Disconnect terminates the referenced database connection.



Connection Reference specifies an active database connection number to disconnect. Set to -1 to disconnect all currently active connections.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Error Code is the result code from the operation.

Error Text is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Disconnect terminates and inactivates the specified database connection and releases all associated resources, thereby ending the database session. Any other active database sessions remain connected. This VI supports a “disconnect all” option that allows a single call to Disconnect to deactivate all active database connections. This function is invoked by executing Disconnect with the Connection Reference control set to -1. This feature is particularly useful during application development to clean up all connections in case of a failure.

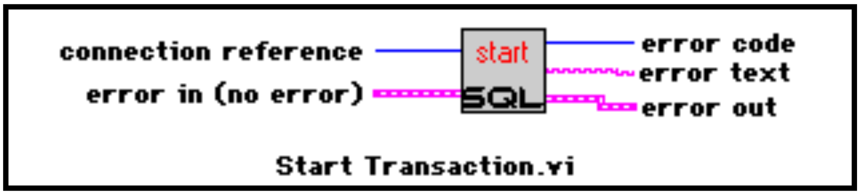
Connect **Execute SQL** **Fetch Results** **End SQL** **Disconnect**

Transaction Operation VIs

All transactions Operation VIs may be executed anytime during an active session. Each database system conducts transaction operations differently. Refer to DatabaseVIEW Driver Help for details on each database system.

Start Transaction

Start Transaction begins a database transaction series on the specified database connection.



Connection Reference specifies an active database connection number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Error Code is the result code from the operation.

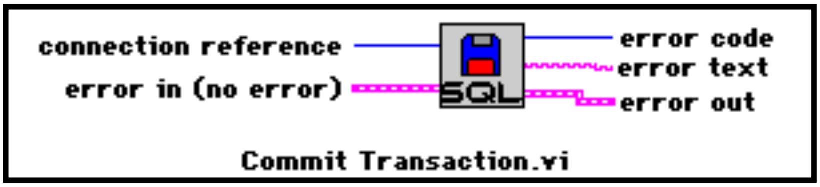
Error Text is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Start Transaction is used to start a transaction series. When a transaction is started, changes to database contents by subsequent SQL operation such as INSERT, UPDATE, and DELETE are recorded to a temporary buffer. The changes may be made permanent to the database by executing **Commit Transaction** (pg. 4 - 15). Alternatively, the changes may be discarded by executing **Rollback Transaction** (pg. 4 - 16). Transactions are only supported by certain database systems.

Commit Transaction

Commit Transaction causes the database to permanently enter all changes that have been made to the contents during the current transaction.



Connection Reference specifies an active database connection number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Error Code is the result code from the operation.

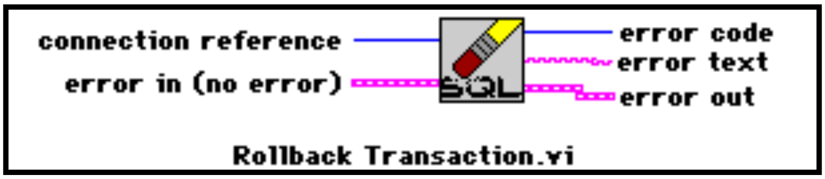
Error Text is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Commit Transaction is used to conclude a transaction and save changes made by SQL statements such as INSERT, ALTER, and DELETE. All changes made since **Start Transaction** (pg. 4 - 14) was executed are made permanent to the database contents. Transactions are only supported by certain database systems.

Rollback Transaction

Rollback Transaction causes the database to permanently discard all changes that have been made during the current transaction.



Connection Reference specifies an active database connection number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Error Code is the result code from the operation.

Error Text is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

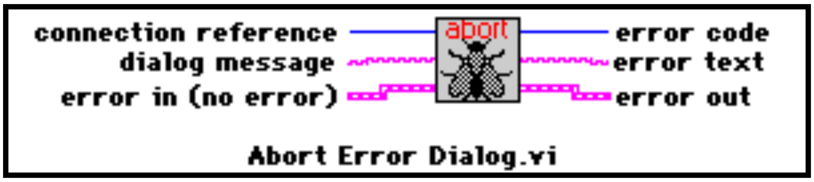
Rollback Transaction is used to conclude a transaction and undo all changes made by SQL statements such as INSERT, UPDATE, and DELETE. Only changes made since executing the most recent **Start Transaction** (pg. 4 - 14) are discarded. Database contents revert to their state immediately before the transaction began, and all intervening changes are ignored. Transactions are only supported by certain database systems.

Error Handling VIs

Error Handling VIs may be executed anytime a session is active. They provide a convenient means to inform an operator that an exception condition has occurred, and allow an interactive response where appropriate. Care must be exercised in using these VIs to abort LabVIEW execution, especially when multiple databases connections are active.

Abort Error Dialog

Abort Error Dialog presents an abort dialog box that describes a database error and then terminates the database connection and VI execution.



I16 **Connection Reference** specifies an active database connection number.

abc **Dialog Message** inputs a string to display in the abort dialog box.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I32 **Error Code** is the result code from the operation.

abc **Error Text** is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Abort Error Dialog is used to provide interpretation and visual indication to an operator when an error is returned from a database operation. When called, the VI checks for an error detection using Error In. If an error has occurred, it displays a dialog window containing the error text (unbundled from the Error In cluster), the Dialog Message, and a single pushbutton labelled “Abort”. After the operator clicks on the Abort button, the VI disconnects from the database specified in Connection Reference, terminates the database session, and then terminates LabVIEW execution.

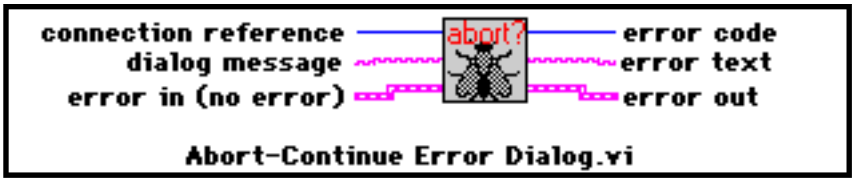
If no error is detected at Error In, then VI execution continues normally.

Set Connection Reference to -1 to disconnect and terminate all active database sessions.

The Dialog Message control is useful for indicating the location of an error within the application VI, e.g. "...while connecting", "...while executing SQL", “...while fetching data”, etc.

Abort-Continue Error Dialog

Abort-Continue Error Dialog presents a selection dialog box that describes a database error and then allows the operator to either continue or terminate the database connection and VI execution.



I16 **Connection Reference** specifies an active database connection number.

abc **Dialog Message** inputs a string to display in the abort dialog box.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I32 **Error Code** is the result code from the operation.

abc **Error Text** is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

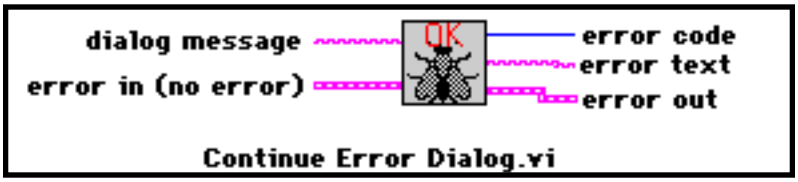
Abort-Continue Error Dialog is used to provide interpretation and visual indication to an operator when an error is returned from a database operation. When called, the VI checks for an error detection using Error In. If an error has occurred, it displays a dialog window containing the error text (unbundled from the Error In cluster), the Dialog Message, and two pushbuttons: "Abort" and "Continue". If the operator clicks on the Abort button, the VI disconnects from the database specified in Connection Reference, terminates the database session, and then terminates LabVIEW execution. If the operator clicks on the Continue pushbutton, VI execution continues normally.

Set Connection Reference to -1 to disconnect and terminate all active database sessions.

The Dialog Message control is useful for indicating the location of an error within the application VI, e.g. "...while connecting", "...while executing SQL", “...while fetching data”, etc.

Continue Error Dialog

Continue Error Dialog presents a dialog box that describes a database error and then continues VI execution.



Dialog Message inputs a string to display in the abort dialog box.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Error Code is the result code from the operation.

Error Text is a textual interpretation of the error code, where available.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Continue Error Dialog is used to provide interpretation and visual indication to an operator when an error is returned from a database operation. When called, the VI checks for an error detection using Error In. If an error has occurred, it displays a dialog window containing the error text (unbundled from the Error In cluster), the Dialog Message, and a "Continue" pushbutton. When the operator clicks on the Continue button, VI execution continues.

The Dialog Message control is useful for indicating the location of an error within the application VI, e.g. "...while connecting", "...while executing SQL", “...while fetching data”, etc.

Chapter 5 Interface VI Reference

This chapter provides a detailed description for each of the Interface VIs, which are the elemental building blocks for the Access VIs. Interface VIs provide detailed low-level access to database services. They are divided into five groups: Fetch Option VIs, Record and Column Fetch VIs, SQL Reference VIs, Datatype Conversion VIs, and Miscellaneous VIs.

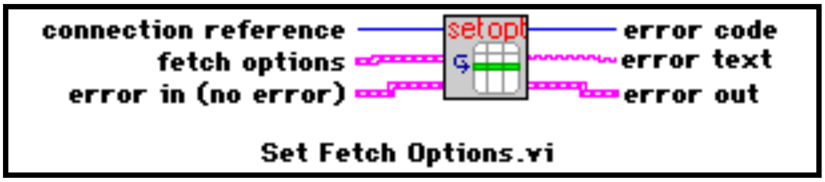
These VIs may be found in the INTRFACE.LLB library, typically installed in the \LABVIEW\DBVIEW.LIB\DBVIEW directory.

Fetch Option VIs

Fetch options allow the user to specify which DatabaseVIEW for Windows VIs may be used to fetch data from SQL SELECT operations. Many database packages permit only sequential row fetches from an active set of query results; some permit random row access; and some dispose the active set of rows automatically upon completion of a transaction. These VIs permit random row access and persistent active sets across all databases, regardless of a database's underlying capability. See DatabaseVIEW Driver Help for details on the selected database.

Set Fetch Options

Set Fetch Options specifies how the fetch VIs may be used to retrieve records from the active set returned by SQL SELECT operations.



Connection Reference specifies an active connection to set the fetch options for.

Fetch Options specifies the fetch options to use for subsequent SQL operations on the connection:

Forward Fetching (default)

Forward/Random/Previous Fetching

Log File When Needed(default)

Log File Always

Disable Fetch after End Of Transaction (EOT)

Truncate Fetch at EOT

Retain Fetch after EOT

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Error Code is the result code from the operation.

Error Text is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Set Fetch Options selects which DatabaseVIEW for Windows VIs may be used to retrieve query data from a SQL SELECT operation. This VI may be called anytime after an active connection has been established. Options are set on each connection reference individually. The Fetch Options cluster elements determine three related aspects of how DBVIEW manages the active set: fetching, log files, and active set persistence after a transaction is completed.

When Forward Fetching is True, **Fetch Next Record** (pg. 5 - 6) may be used.When **Forward/Random/Previous Fetching** is True, **Fetch Next Record** (pg. 5 - 6), **Fetch Previous Record** (pg. 5 - 8), **Fetch Record N** (pg. 5 - 9), and **Get Number of Records** (pg. 5 - 14) may be used.

Depending on the capabilities of the underlying database, DBVIEW may need to write the active set to local disk files to support random fetching. When first executed after a SQL SELECT operation, **Fetch Previous Record**, **Fetch Record N**, or **Get Number of Records** will transfer the entire query result to the temporary files. Subsequent fetch executions will retrive rows from those temp files into LabVIEW. When Log File When Needed is True, log files are used only if required. When Log File Always is True, the active set is always logged to disk (this feature is useful for debugging). Log files are written to the current DOS TEMP directory. The TEMP directory must be specified in the AUTOEXEC.BAT file, or on the command line as

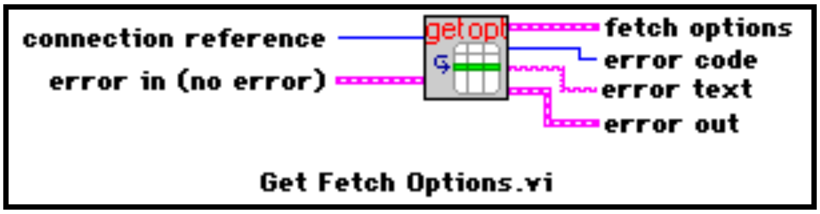
SET TEMP=*your_temp_directory*

where *your_temp_directory* may be any valid existing DOS directory, e.g. C:\TEMP. Care should be taken that there is sufficient disk space available, especially for large queries. Additionally, a sufficient number of open DOS files (the default is 20) must be permitted. If the DOS open files limit is exceeded, use **Close Fetch Log File** (pg. 5 - 5) to close any temporary files not currently in use. The contents of files closed in this way are preserved, and the files will be reopened on subsequent fetches to those SQL references as long as the references remain active.

In order to effectively manage RAM and disk space usage, DBVIEW supports three options for managing the active set upon concluding a transaction (see *Transaction Operation VIs*, pg. 4 - 14). When Disable Fetch After EOT (End Of Transaction) is True, the entire active set is disposed when the current transaction is ended. When Truncate Fetch on EOT is True, all unfetched records in the active set are disposed - previously fetched rows are retained for further use. When Retain Fetch After EOT is True, the entire active set is retained for further use after the current transaction is ended.

Get Fetch Options

Get Fetch Options determines the fetch options currently in effect for the specified connection reference.



I16 **Connection Reference** specifies an active connection to get the fetch option settings for.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Fetch Options specifies the fetch options currently in use for SQL operations on the connection (see **Set Fetch Options** (pg. 5 - 1)).

I32 **Error Code** is the result code from the operation.

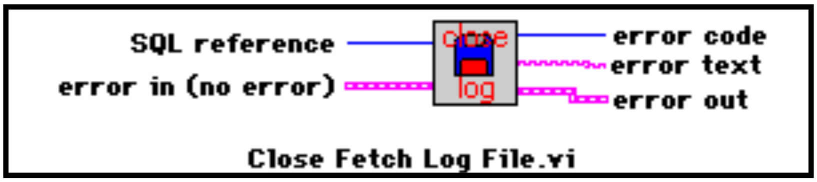
abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Get Fetch Options retrieves the current fetch option settings for the specified connection reference, as set by a previous **Set Fetch Options** (pg. 5 - 1) operation.

Close Fetch Log File

Close Fetch Log File closes any open temporary fetch log files for the specified SQL reference.



SQL Reference specifies a previously executed SQL SELECT operation number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Error Code is the result code from the operation.

Error Text is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Close Fetch Log File closes all temporary fetch log files associated with the specified SQL reference. The contents of the log files are preserved, and the files will be reopened automatically when the next fetch operation is performed on that SQL reference.

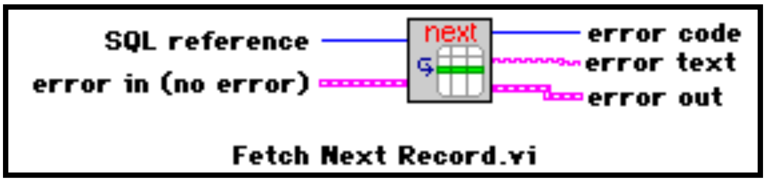
Record and Column Fetch VIs

These VIs retrieve data rows and columns individually from active, previously executed SQL SELECT operations onto the LabVIEW block diagram where the results may be processed. To fetch the entire results of a SELECT (memory permitting) see **Fetch Query Results** (pg. 4 - 10). Records are usually fetched sequentially, thus preserving the order (optionally) specified in the ORDER BY clause of the SELECT statement. Many databases permit only sequential record fetches from query results, while others permit random record access. See DatabaseVIEW Driver Help for details on the selected database. Refer to *Fetch Option VIs*, pg. 5 - 1 for details on when and how to use the random record fetch VIs.

Each time a record is fetched, regardless of the method, it becomes the current record. Once a record has been made current, its column contents may then be fetched into the LabVIEW diagram using **Fetch Column Data** (pg. 5 - 10).

Fetch Next Record

Fetch Next Record retrieves the next record from the results of a SQL SELECT operation.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I32 **Error Code** is the result code from the operation.

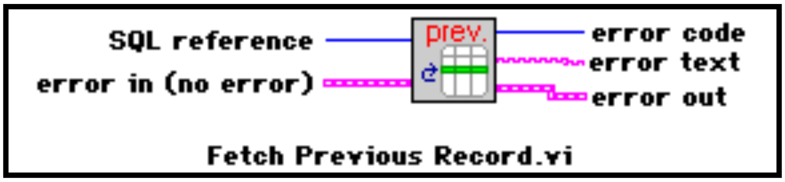
abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Fetch Next Record moves the next record from the database system (or temporary log files if enabled) into local memory, and makes that record the current record. When called for the first time after executing the SQL SELECT reference, this VI will return the first record from the database. Use **Fetch Column Data** (pg. 5 - 10) to retrieve the columns from the current record into LabVIEW.

Fetch Previous Record

Fetch Previous Record retrieves the previous record from the results of a SQL SELECT operation.



SQL Reference specifies a previously executed SQL SELECT operation number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Error Code is the result code from the operation.

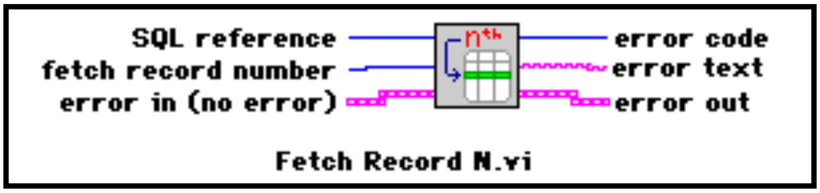
Error Text is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Fetch Previous Record moves the previous record from the database system (or temporary log files) into local memory, and makes that record the current record. **Set Fetch Options** (pg. 5 - 1) must be used to enable this function. Use **Fetch Column Data** (pg. 5 - 10) to retrieve the columns from the record into LabVIEW.

Fetch Record N

Fetch Record N retrieves the Nth record from the results of a SQL SELECT operation.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

I32 **Fetch Record Number** specifies the number of the record to fetch from the SQL SELECT results .

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I32 **Error Code** is the result code from the operation.

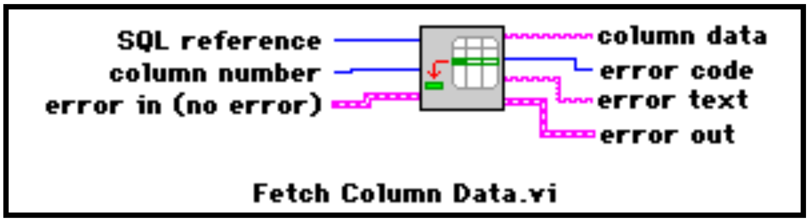
abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Fetch Record N moves the Nth record from the database system (or temporary log files) into local memory, and makes that record the current record. **Set Fetch Options** (pg. 5 - 1) must be used to enable this function. Use **Fetch Column Data** (pg. 5 - 10) to retrieve the columns from the record into LabVIEW.

Fetch Column Data

Fetch Column Data retrieves the contents of a specific column from the current record.



SQL Reference specifies a previously executed SQL SELECT operation number.

Column Number specifies the number of the column to fetch from the current record.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

Column Data returns the results of a SQL SELECT statement in a two-dimensional array of strings. Other SQL statements return no data.

Error Code is the result code from the operation.

Error Text is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

Fetch Column Data returns the data contents of the specified column from the current record into a LabVIEW string. One of the *Record and Column Fetch VIs*, pg. 5 - 6 VIs must have been executed prior to using this VI. Column contents must be retrieved in order, e.g. column 1, 2, 3. A column's contents may not be retrieved twice. Columns may be skipped, but once a subsequent column has been retrieved, no earlier columns may be retrieved. For example: if columns 1, 2, and 4 are fetched in order, then column 3 may not be fetched.

Examine the block diagram for **Fetch Query Results** (pg. 4 - 10) for an example of how to fetch column data.

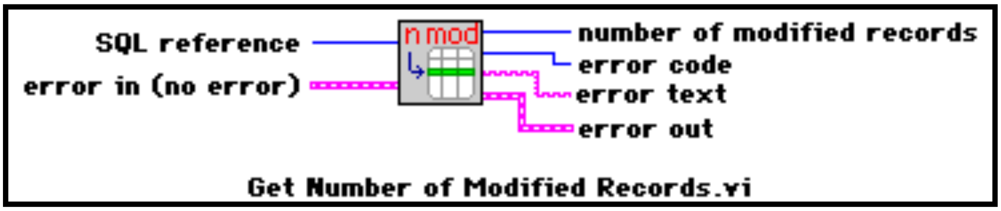
Once a column value has been fetched, its datatype may be converted from string to other data types using LabVIEW’s built-in string conversion function. Consult the *LabVIEW Function Reference Manual* for details. Conversion from/to the SQL date type may be performed using **SQL Date to LabVIEW Date Format** (pg. 5 - 23) and **LabVIEW Date to SQL Date Format** (pg. 5 - 24).

SQL Reference VIs

This group of VIs provide access to the circumstances and operational results of a SQL reference. Specifically, these VIs may be used to determine the column structure of the SQL reference; including column name, data type, width, precision, and scale. This is often referred to as data dictionary information. Additional VIs are provided to determine the results of SQL execution, including the number of rows returned from a SELECT query; the number of rows affected by an INSERT, UPDATE, or DELETE statement; and the underlying database system datatypes.

Get Number of Modified Records

Get Number of Modified Records returns the number of database records that have been modified by SQL INSERT, UPDATE, and DELETE operations.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I32 **Number of Modified Records** returns the number of records that were modified by the referenced SQL operation.

I32 **Error Code** is the result code from the operation.

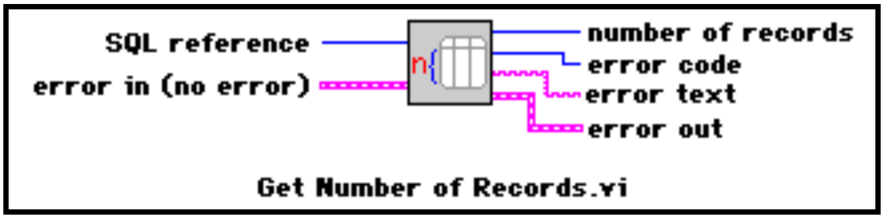
abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

This VI is used to determine the actual number of database records that were modified by the previously executed SQL operation. The Number of Modified Records indicator will always be 0 if the SQL operation was a SELECT.

Get Number of Records

Get Number of Records determines the number of records satisfying the referenced SQL SELECT operation.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I32 **Error Code** is the result code from the operation.

I32 **Number of Records** returns the number of records that were chosen by the referenced SQL SELECT operation.

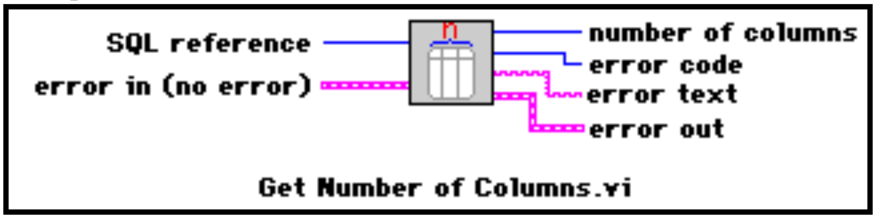
abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

This VI is used to determine the actual number of records that were found when a SQL SELECT operation was executed. Get Number of Records may only be used if random fetching has been previously enabled by **Set Fetch Options** (pg. 5 - 1). Note that when this VI is executed, DBVIEW actually retrieves all selected records from the database system into temporary log files. Care should be taken that sufficient files and disk space are available to accommodate all retrieved data.

Get Number of Columns

Get Number of Columns determines the number of columns returned by the referenced SQL SELECT operation



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I32 **Number of Columns** returns the number of columns that were returned by the referenced SQL operation.

I32 **Error Code** is the result code from the operation.

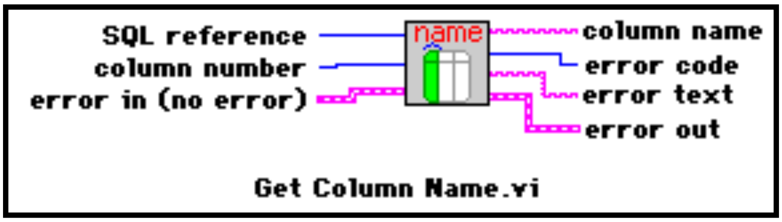
abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

This VI is used to determine the actual number of columns that were retrieved when a SQL SELECT operation was executed.

Get Column Name

Get Column Name determines the name of a specified column returned by the referenced SQL SELECT operation.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

I16 **Column Number** specifies the column number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

abc **Column Name** returns the actual name of the specified column.

I32 **Error Code** is the result code from the operation.

abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

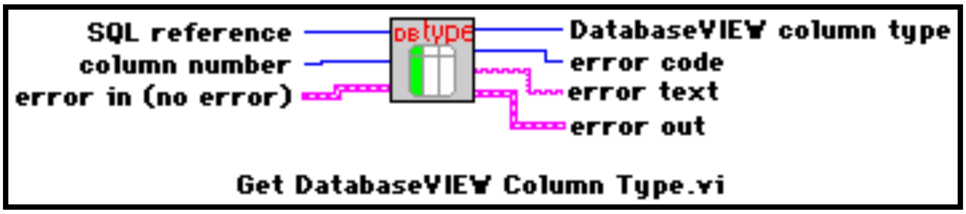
This VI is used to determine the actual names of columns that were retrieved when a SQL SELECT operation was executed. If a retrieved column is the result of a SQL aggregation function on one or more columns, e.g.

SELECT ((begin_balance) - (end_balance)) **FROM** accnt,

then the returned name will be an empty string.

Get DatabaseVIEW Column Type

Get DatabaseVIEW Column Type determines the DBVIEW data type of a specified column returned by the referenced SQL operation.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

I16 **Column Number** specifies the column number to get the type for.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I16 **DatabaseVIEW Column Type** returns the DBVIEW data type code of the specified column.

Table 5. DBVIEW Data Type Codes

Type Code	Data Type Description
1	fixed length character string
2	variable length character string
3	Binary Coded Decimal (BCD) number
4	4-byte long integer
5	2-byte integer
6	4-byte single precision floating-point
7	8-byte double precision floating-point
8	26-byte date-time string of the form YYYY-MM-DD HH:MM:SS.SSSSSS

I32 **Error Code** is the result code from the operation.

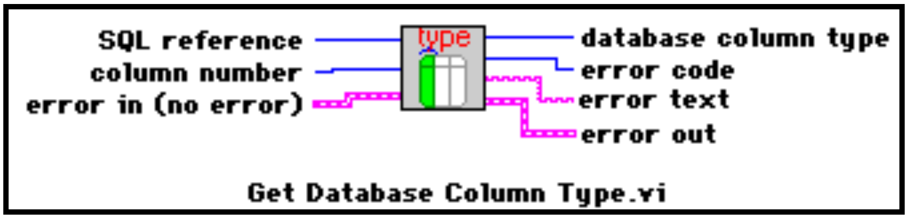
Error Text is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

This VI is used to determine the DBVIEW data type of a column retrieved when a SQL SELECT operation was executed.

Get Database Column Type

Get Database Column Type determines the database data type of a specified column returned by the referenced SQL SELECT operation.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

I16 **Column Number** specifies the column number to get the type for.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I16 **Database Column Type** returns the underlying database data type code of the specified column.

I32 **Error Code** is the result code from the operation.

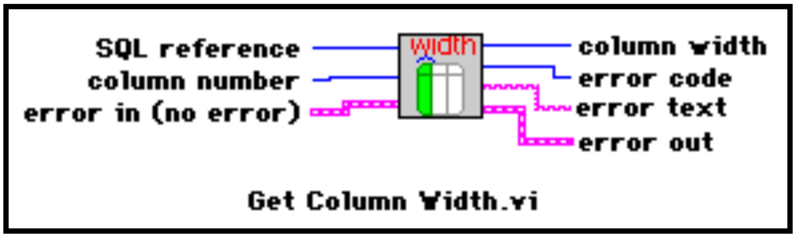
abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

This VI is used to determine the underlying database data type of a column retrieved when a SQL SELECT operation was executed. Each database system supports different data types and corresponding type codes. Consult DatabaseVIEW Driver Help for a definition of the type codes for the selected database.

Get Column Width

Get Column Width determines the data width of a specified column returned by the referenced SQL SELECT operation.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

I16 **Column Number** specifies the column number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I16 **Column Width** returns the maximum number of bytes that may be contained in the specified column.

I32 **Error Code** is the result code from the operation.

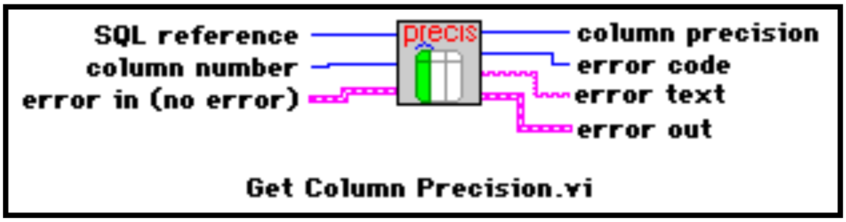
abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

This VI is used to determine the maximum data width of a column retrieved when a SQL SELECT operation has executed. Note that this is not the width of a particular returned column value, but rather the largest data width that the selected column may contain in the database. The actual column value width will always be less than or equal to Column Width.

Get Column Precision

Get Column Precision determines the decimal precision of a specified numeric column returned by the referenced SQL SELECT operation.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

I16 **Column Number** specifies the column number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I16 **Column Precision** returns maximum number of decimal digits that the selected column retains for its contents.

I32 **Error Code** is the result code from the operation.

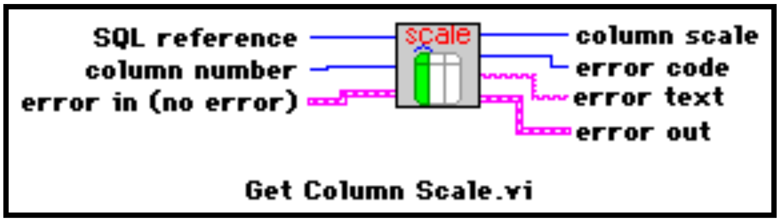
abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

This VI is used to determine the maximum number of decimal digits in a numeric column retrieved when a SQL SELECT operation was executed. This includes all digits on either side of the decimal point.

Get Column Scale

Get Column Scale determines the decimal scale of a specified numeric column returned by the referenced SQL SELECT operation.



I16 **SQL Reference** specifies a previously executed SQL SELECT operation number.

I16 **Column Number** specifies the column number.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I16 **Column Scale** returns maximum number of fractional decimal digits that the selected column retains for its contents.

I32 **Error Code** is the result code from the operation.

abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

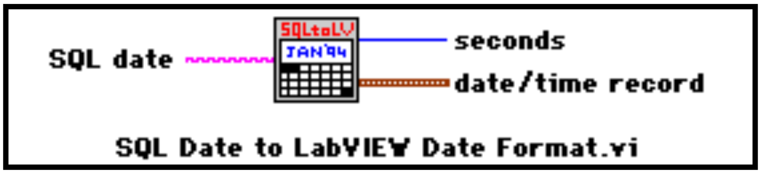
This VI is used to determine the maximum number of fractional decimal digits, i.e. to the right of the decimal point, in a numeric column retrieved when a SQL SELECT operation was executed.

Datatype Conversion VIs

This group of VIs perform conversions between DBVIEW date-time strings and LabVIEW's date-time cluster representation.

SQL Date to LabVIEW Date Format

SQL Date to LabVIEW Format converts from SQL date string to the LabVIEW's date-time representation.



SQL Date specifies a SQL date string, e.g. a column value of date type that has been returned from a SQL SELECT operation.

Seconds returns the number of seconds that have elapsed since 12:00am., Friday, January 1, 1904.

Date/Time Record is a cluster that contains several elements representing the LabVIEW date (see the *LabVIEW Function Reference Manual*).

This VI is used to convert from a SQL date string to the LabVIEW representations for date-time. It may be used to convert the date column values from a SQL SELECT operation into LabVIEW representation so that date arithmetic can be performed (seconds), or so that the individual parts of the date can be used (date-time rec). Note that LabVIEW supports date-time resolution to the nearest second, therefore the fractional part of the SQL date string will always be ignored when converting. SQL Date must have the form:

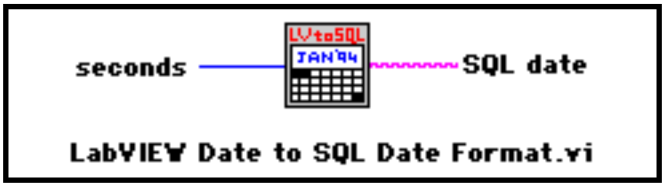
YYYY/MM/DD HH:MM:SS.SSSSSS

To ensure that date-time column values are returned from queries in this form, use the SQL function DTOC (page A - 10) in your SQL statement, with fmt_value set to 12, e.g.

```
SELECT DTOC(test_date,12),test_value,operator FROM testtable
```

LabVIEW Date to SQL Date Format

LabVIEW to SQL Date Format converts from LabVIEW's date-time representation to a SQL date string.



I32 **Seconds** specifies a date-time as the number of seconds (elapsed since 12:00am., Friday, January 1, 1904) to convert to a SQL date string. If **seconds** is not wired, the current date-time is converted.

abc **SQL Date String** returns the equivalent SQL date string.

This VI is used to convert from the LabVIEW date-time representation in seconds to the equivalent SQL date string for use in INSERT or UPDATE statements. The resulting SQL date string takes the form:

[dt'YYYY-MM-DD HH:MM:SS.SSSSSS']

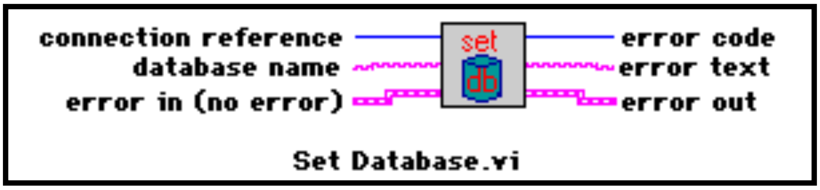
Note that LabVIEW supports date-time resolution to the nearest second, therefore the fractional part of the SQL date string will always be set to zero when converting.

Miscellaneous VIs

This group of VIs provides additional functions for database operations.

Set Database

Set Database specifies the default database to use for subsequent operations on the referenced connection.



I16 **Connection Reference** is used to designate a specific, currently active database connection.

abc **Database Name** is used by several of the supported databases to select a particular default database instance to use on the server computer.

Error In (no error) describes any error conditions that occurred prior to executing this VI. See *Database Error Handling*, pg. 3 - 10.

I32 **Error Code** is the result code from the operation.

abc **Error Text** is a textual interpretation of the error code, where available from the operation.

Error Out describes any error conditions that occurred while executing this VI. See *Database Error Handling*, pg. 3 - 10.

This VI is used to control the default database to use for operations on the specified connection reference. This function is only supported by certain database systems. Consult DatabaseVIEW Driver Help for details on the selected database system.

_DatabaseVIEW

This VI is the lowest-level foundation driver for all DBVIEW VIs. It consists of a single LabVIEW Code Interface Node (CIN), which contains the C language code that calls the appropriate Dynamic Link Libraries (DLLs) for the selected databases. Ellipsis Product’s intent is that ALL DBVIEW VIs be *open* so that the user may “peek-under-the-hood” and adapt and optimize the library to individual application needs. **The _DatabaseVIEW VI, however, cannot be adapted - and should not be used directly on any application block diagrams.**

Chapter 6 Development Utilities

This chapter describes the general-purpose DatabaseVIEW for Windows Utilities. These are supplied as tools for the application developer.

Utility VIs may be found in the DBUTIL.LLB library, which is typically installed in the \LABVIEW\DBVIEW.LIB\DBVIEW directory. Other utilities are locatedin the DatabaseVIEW program group.

Utility VIs

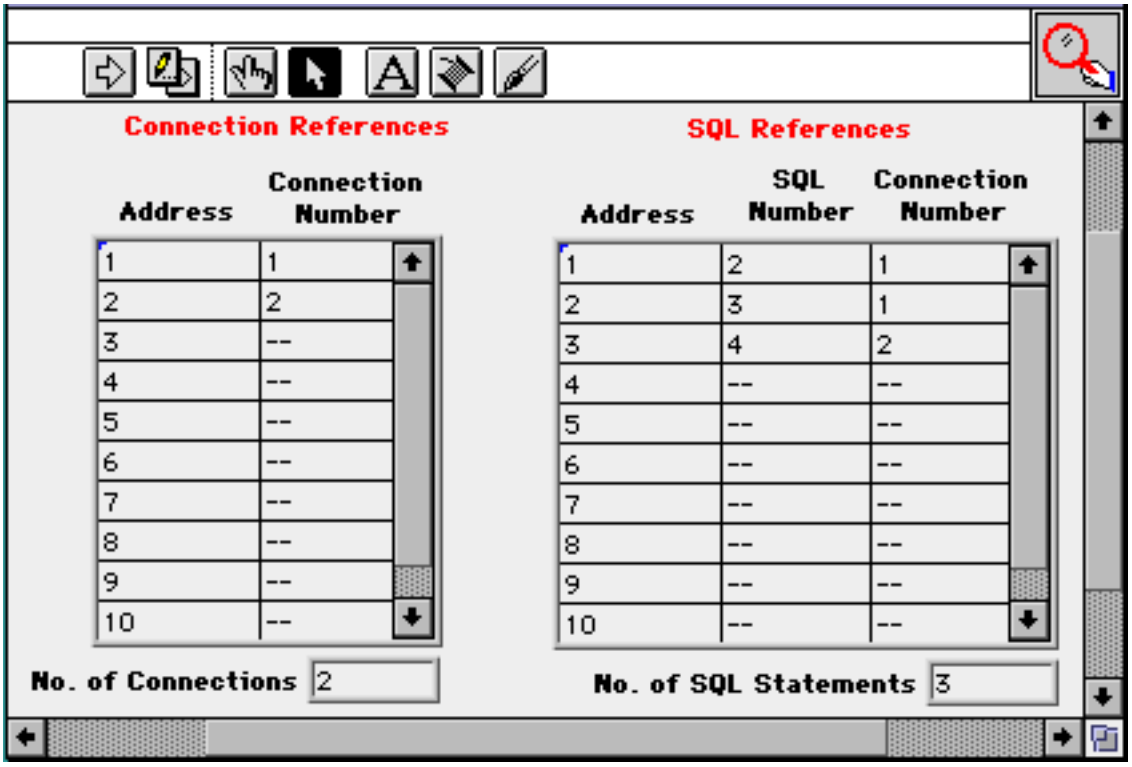
Database Session Monitor VI

The Database Session Monitor VI provides a convenient tool to monitor active database connections and SQL statements. It accomplishes this by probing DatabaseVIEW's internal database connection and SQL reference structures. It is particularly useful when unit testing and integrating your application programs. Figure 29, pg. 6- 2 shows the Database Session Monitor front panel. Note that this VI is typically executed as a stand-alone program, and would almost never be incorporated into another block diagram.

Database Session Monitor may be opened and run just as any LabVIEW VI. It may be run while your development application is running, during a break/step point, or after an error condition occurs. Each time it is run, Database Session Monitor probes DatabaseVIEW internals and displays all active connections references in the Connection References table indicator, and all active SQL statements in the SQL References table indicator.

Active connections and SQL statements are displayed using the same reference numbers that are automatically generated and used on your block diagrams (see **Connect** (pg. 4 - 2) and **Execute SQL** (pg. 4 - 9)). In addition, SQL references are displayed along with their corresponding connections. For example, in Figure 29, there are two active connections, reference numbered 1 and 2. Connection 1 has two active SQL statements reference numbered 2 and 3; while connection 2 has one active SQL statement reference numbered 4 - for a total of three active SQL references.

You should note that if you run Monitor while your application is running, it will not interfere with you application's database operation. It will, however, consume some memory and CPU resources.



Connection References

Address	Connection Number
1	1
2	2
3	--
4	--
5	--
6	--
7	--
8	--
9	--
10	--

No. of Connections 2

SQL References

Address	SQL Number	Connection Number
1	2	1
2	3	1
3	4	2
4	--	--
5	--	--
6	--	--
7	--	--
8	--	--
9	--	--
10	--	--

No. of SQL Statements 3

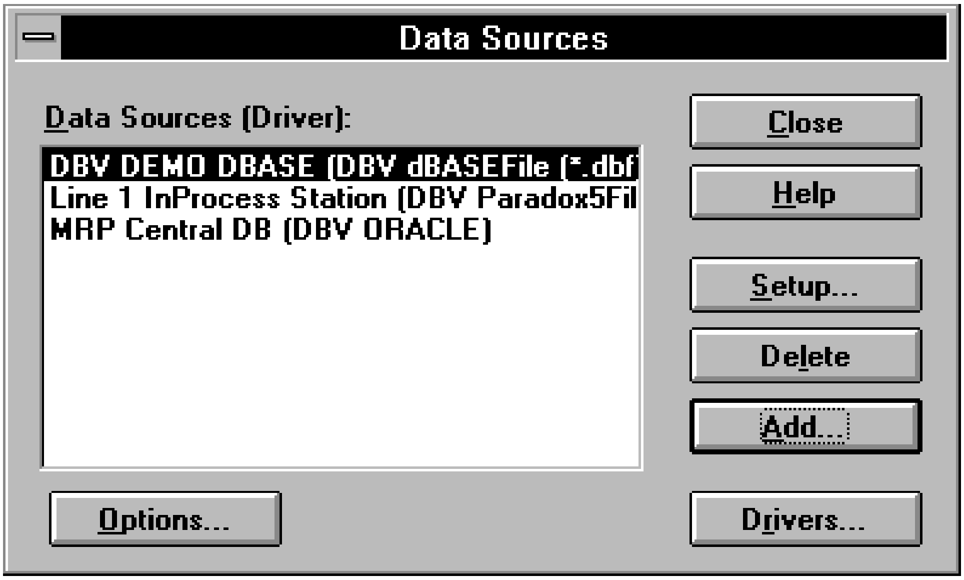
Figure 29. Database Session Monitor VI Front Panel

Administration and On-Line Help Utilities

These utilities are accessed from the DatabaseVIEW program group in the Windows Pro-grram Manager. They provide ODBC administration functions and on-line help for the supported database drivers.

ODBC Administrator

Double clicking on the **ODBC Administrator** program item icon in the DatabaseVIEW program group launches a Windows *Control Panel* program called ODBC Administrator. This program is used to add new ODBC Data Sources, setup and modify existing data sources, delete data sources, and add new ODBC database drivers. Figure 30 shows the control panel window. On-line help information is provided in all ODBC administration windows by clicking on the **Help** buttons.

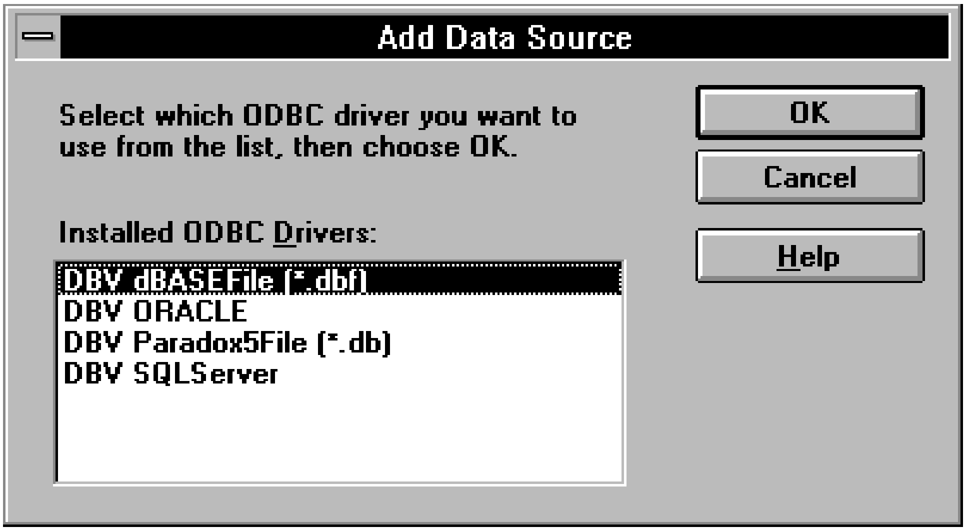


DBV DEMO DBASE (DBV dBASEFile (*.dbf
Line 1 InProcess Station (DBV Paradox5Fil
MRP Central DB (DBV ORACLE)

Figure 30. ODBC Administrator Data Sources Window

Adding a New Data Source

Click on the **Add**... button to create and define a new data source. The Add Data Source dialog window will be displayed as shown in Figure 31, pg. 6- 4. All currently installed ODBC database drivers are displayed in the selection list. Click on and highlight the driver for the desired database, then push the **OK** button. A database specific driver setup dialog window will be displayed to define the data source. Figure 32, pg. 6- 5 shows the dBASE driver setup window. Enter the desired data source name and any other pertinent configuration parameters. Push the **Help** button to view database-specific information on the chosen driver, including descriptions of all parameters. Push **OK** to create the dataset. The control panel places the specified information in the ODBC.INI file in the WINDOWS directory, and the new data source now appears in the data sources windows.

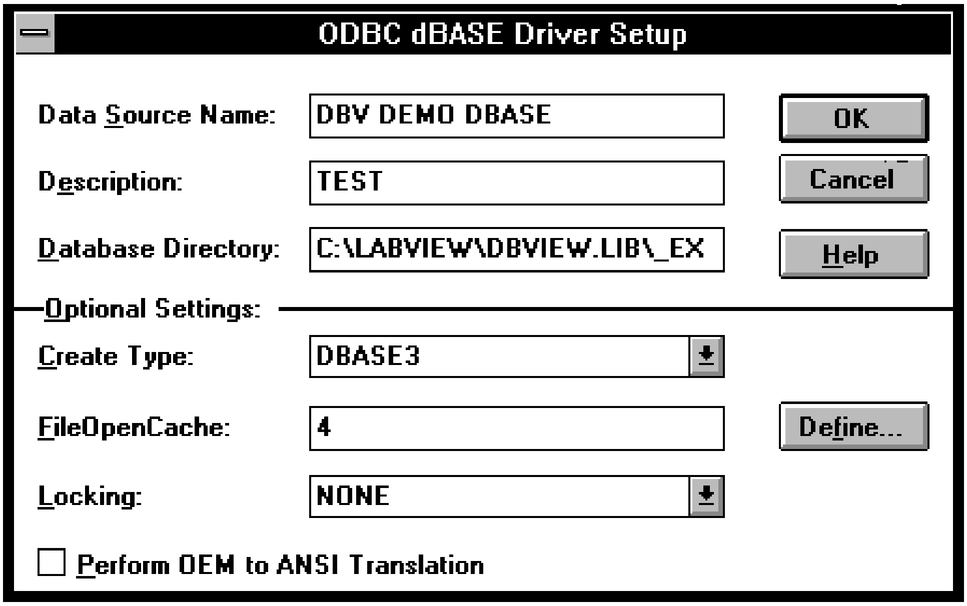


Select which ODBC driver you want to use from the list, then choose OK.

Installed ODBC Drivers:

DBV dBASEFile (*.dbf)
DBV ORACLE
DBV Paradox5File (*.db)
DBV SQLServer

Figure 31. Add Data Source Window


ODBC dBASE Driver Setup

Data Source Name:	DBV DEMO DBASE
Description:	TEST
Database Directory:	C:\LABVIEW\DBVIEW.LIB_EX

Optional Settings:

Create Type:	DBASE3
FileOpenCache:	4
Locking:	NONE

☐ Perform OEM to ANSI Translation

Figure 32. ODBC dBASE Driver Setup Window

Modifying an Existing Data Source

In the Data Sources window, click on and highlight the data source to be modified, and then click on **Setup...** A database-specific driver setup window will be displayed containing the current data source parameters. Modify the parameters and click on **OK** to modify the data source.

Deleting a Data Source

In the Data Sources window, click on and highlight the data source to be deleted, and then click on **Delete**. The data source will be deleted.

Managing ODBC Database Drivers

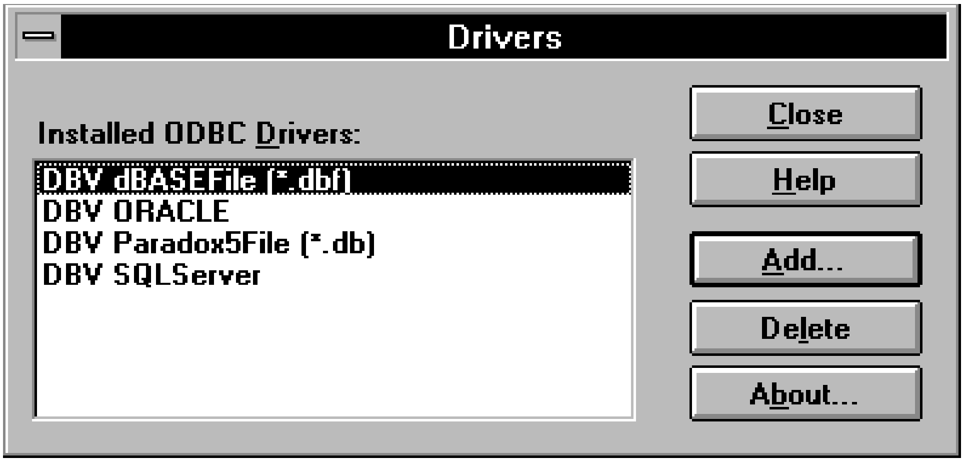
The ODBC Administrator control panel may be used to install and manage ODBC drivers. To install drivers supplied by Ellipsis Products, you must use the DatabaseVIEW for Windows SETUP installation program (see *Installing Additional Database Drivers*, pg. 1 - 9). For ODBC compliant drivers from any other source, follow the publisher's instructions, or use the following procedure.

Adding a New Driver

In the Data Sources window, push the **Drivers...** button to manage ODBC drivers. A Drivers window is displayed as shown in *ODBC Drivers Window*, pg. 6 - 7. Push the **Add...** button to add a new driver. An Add Driver window is displayed as shown in Figure 34, pg. 6- 7. Type the full file path of the ODBC.INF file supplied with the new driver, and then push **OK**. The new driver will be added, and will appear in the Installed ODBC Drivers list in the Drivers and Add Data Source windows. The driver information is installed into the ODBCINST.INI file in the WINDOWS directory. Or, push **Browse...** to search for the ODBC.INF file for the new driver.

Deleting a Driver

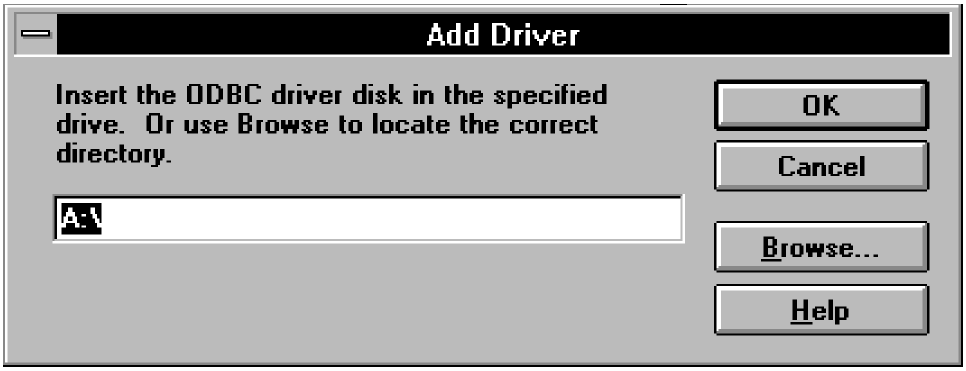
In the Drivers window, click on and highlight the driver to be deleted, and then click on **Delete**. The driver will be deleted. Note that this will also delete all data sources that currently refer to the deleted driver.



Installed ODBC Drivers:

DBV dBASEFile (*.dbf)
DBV ORACLE
DBV Paradox5File (*.db)
DBV SQLServer

Figure 33. ODBC Drivers Window



Insert the ODBC driver disk in the specified drive. Or use Browse to locate the correct directory.

A:\

Figure 34. Add Driver Window

Database Driver Help

Double-clicking on the **DatabaseVIEW Driver Help** program item icon in the Database-VIEW program group launches a standard Windows Help file titled *INTERSOLV DataDirect ODBC Drivers*. This help file provides extensive information on each of the supported databases. The information typically includes:

- Supported Database Configurations
- System Requirements
- Configuring Data Sources
- Connecting to a Data Source Using a Logon Dialog Box
- Connecting to a Data Source Using a Connection String
- Column Data Types
- Isolation and Lock Levels Supported
- ODBC Conformance Level
- Number of Connections and SQL Statements Supported

Additional information is provided for the file-based databases regarding error messages, indexes, join behavior, and data dictionary features, where supported by the database package. The 'SQL for Flat-File Drivers' topic is a particularly helpful reference for SQL statement construction for the file-based databases. In general, this SQL help is applicable to server-based databases as well.

Database Driver Read-Me

Clicking on the **DatabaseVIEW Driver Read-Me** program item icon in the Database-VIEW program group launches a standard Windows Help file titled *INTERSOLV DataDirect ODBC Pack Readme*. This help file provides supplemental information for several of the supported databases, including valuable installation and configuration details not found in the driver help file. All of the main topics are applicable to DatabaseVIEW with the exception of ‘Installing the DataDirect ODBC Pack’, which you may ignore.

There are two topics of particular interest: Driver Options and Solving Connection Problems with DLL Manager.

The Driver Options topic describes several ‘WorkAround’ parameters that may be useful in developing your database application. It refers to modifying the ODBC.INI file to apply the WorkAround attribute-value statements, e.g. WorkArounds=1. You must be very careful when editing ODBC.INI so as not to corrupt your data source definitions. ODBC.INI is located in the Windows directory, typically C:\WINDOWS.

The Solving Connection Problems with DLL Manager topic describes a utility program, similar to Database Session Monitor VI (see above), that allows you to monitor active Windows Dynamic Link Libraries (DLLs). This is particularly useful when attempting to connect to a new database for the first time. The utility program is called DLLMGR.EXE, and is located under your DatabaseVIEW installation directory, typically

C:\LABVIEW\DBVIEW.LIB_DBVUTIL\DLLMGR.EXE

You may launch this application from the Windows File Manager by double-clicking on its icon.

Chapter 7 Application Examples

This chapter presents several example applications that incorporate DatabaseVIEW VIs. The first, a factory quality control test station application, presents a generic top-down structured design process for developing LabVIEW applications that incorporate database functionality. The next example, Quick SQL, illustrates a DBVIEW database application at its simplest. The remaining examples are provided for user study and reuse.

These VIs may be found in the DBEXMPLS.LLB library, which is typically located in the LABVIEW\EXAMPLES\DBVIEW directory.

A Generic Process for Designing a DBVIEW-based VI

As with any complex software application, a methodical development practice must be followed to achieve a quality application to meet a given need. The following paragraphs describe several steps that comprise a typical structured application development process, and how each step addresses the database component of the design. The intent is not to teach structured development, but rather to identify the additional planning, design, testing, integration, and maintenance considerations that a database imposes on the software development cycle.

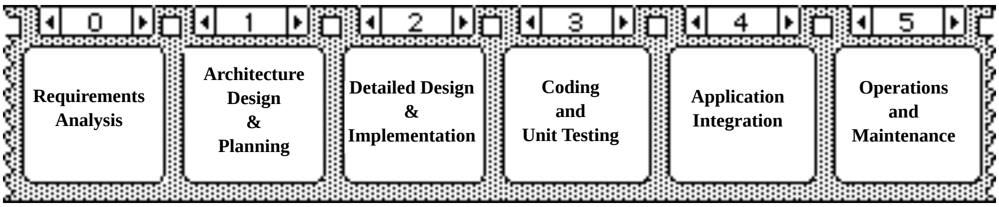

0
Requirements Analysis
1
Architecture Design & Planning
2
Detailed Design & Implementation
3
Coding and Unit Testing
4
Application Integration
5
Operations and Maintenance

Figure 35. Generic Development Process

A simple factory quality control test example[1] is used to illustrate each step in the development process: a company manufactures computer modems, and the QC

1. Note that this tutorial example has been greatly simplified to illustrate the basics of data management analysis and design. Consult the *Reference Reading List*, pg. 1 - 19 for more comprehensive information on structured design and database planning.

department must develop a LabVIEW based system to test each modem at several points during the the assembly process.

.

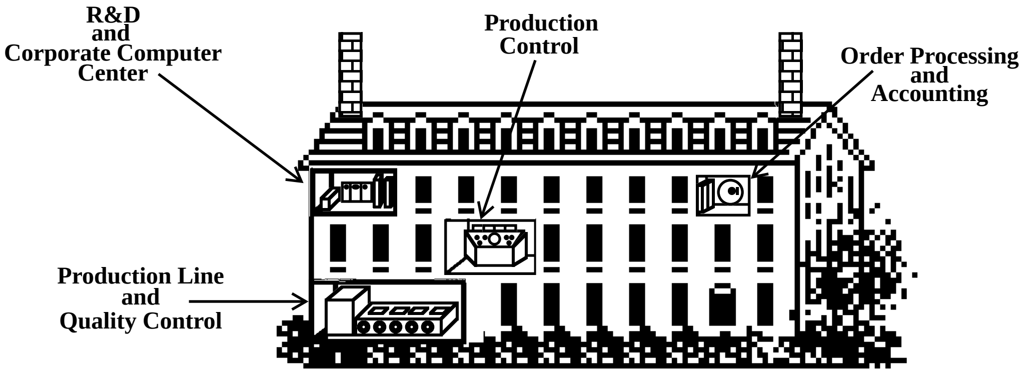

R&D and Corporate Computer Center
Production Control
Order Processing and Accounting
Production Line and Quality Control

Figure 36. Modem Factory

Application Requirements Analysis

Application requirements must be clearly negotiated and agreed upon in advance with the user community. Data acquisition, reduction, display, and recording must all be addressed. In particular, the community must agree on the requirements for application data collection, storage, management, and retrieval; especially across many executions.

The QC department has determined that three parametric tests are needed to test a modem. The parametric tests must be run on the production floor, and the results recorded by unit serial number along with the test time and test operator. The raw time-series test data must be preserved for special studies by the R&D department. Analysis must be performed in the QC department on the most recent 6 months of parametric test data to detect manufacturing quality trends. Production yield results in terms of passed/failed units must also be made available to the production control department. The order processing and accounting department wants to monitor production as well.

Application Architecture Design and Planning

Once settled, application requirements must be mapped into a system architecture. This consists of functional definition and decomposition of the system into hardware, communications, software, and data management components.

In particular, a system data model or *schema* must be defined that completely represents all information to be preserved in the database and on disk. Essentially, a data model is a related collection of database tables containing typed data columns.

Also, estimates must be made of overall system and database usage before actually selecting the components. Appropriately sized database server and client computers, networks, disks, and database partitions are then specified from the estimates.

The QC test system architecture is defined as shown in Figure 37. The corporate server computer will be used as the database host. The test system data model will be a simple single table structure as illustrated in Figure 38, pg. 7 - 4.

Modem production control estimates that 1000 modems per day will be processed 5 days per week. This means a total of 130000 (1000 modems X 5 days X 26 weeks) rows must be on line (in the table) at any time, from which table, tablespace, database, and disk sizes can be calculated. Administration plans for database backup, recovery, and data archiving are also made.

Since the network link to the existing database on the corporate server will take some time to install, a single-user copy of a compatible database is installed on the QC test computer for local development

.

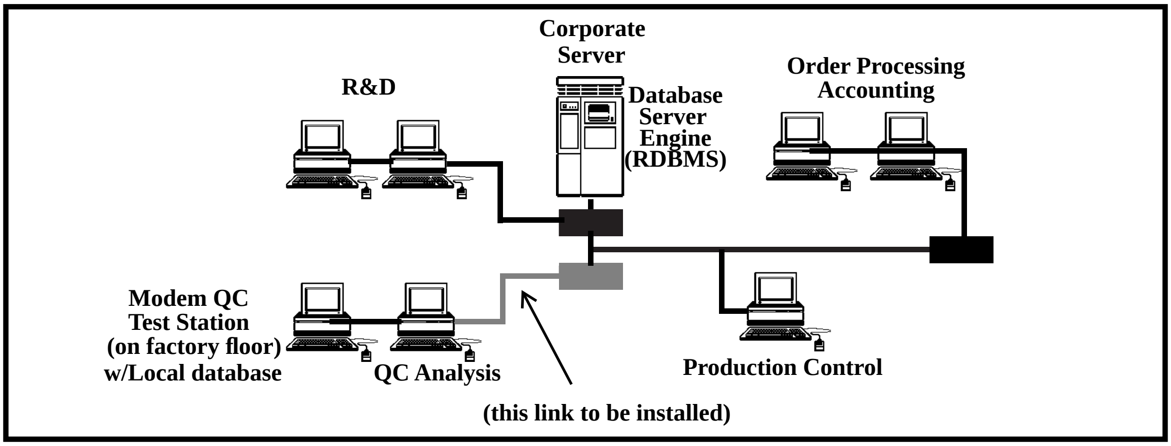
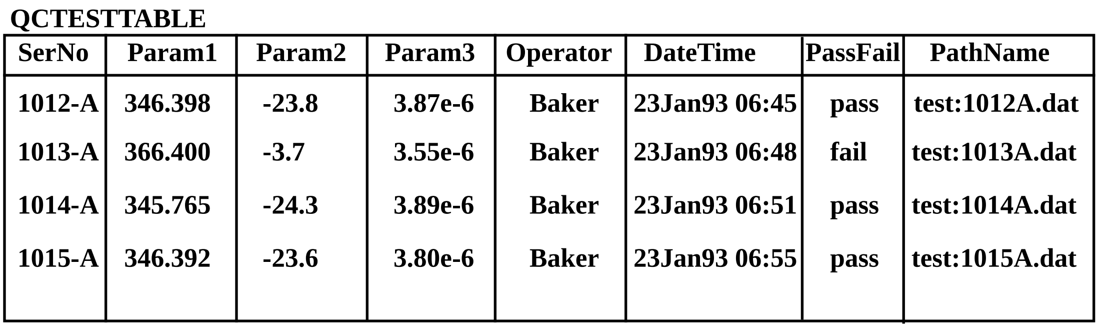

Corporate Server
Database Server Engine (RDBMS)
R&D
Order Processing Accounting
Modem QC Test Station (on factory floor) w/Local database
QC Analysis
Production Control
(this link to be installed)

Figure 37. Modem Factory Architecture



QCTESTTABLE

SerNo	Param1	Param2	Param3	Operator	DateTime	PassFail	PathName
1012-A	346.398	-23.8	3.87e-6	Baker	23Jan93 06:45	pass	test:1012A.dat
1013-A	366.400	-3.7	3.55e-6	Baker	23Jan93 06:48	fail	test:1013A.dat
1014-A	345.765	-24.3	3.89e-6	Baker	23Jan93 06:51	pass	test:1014A.dat
1015-A	346.392	-23.6	3.80e-6	Baker	23Jan93 06:55	pass	test:1015A.dat

Figure 38. Modem Factory Data Model

Detailed Application Design and Implementation

Detailed test process development proceeds concurrently with database design and planning. The application software must be decomposed *top-down* into subVIs that separate data acquisition and analysis operations from database access functions. This allows development and unit testing to proceed concurrently within the development team.

The database table design continues with column type definition where specific database column types are assigned to best represent the selected LabVIEW test data types. Prototype tables and SQL statements are then created and tested with sample data.

Four separate top level VIs are designed, each with several supporting subVIs as shown in Figure 39.

.

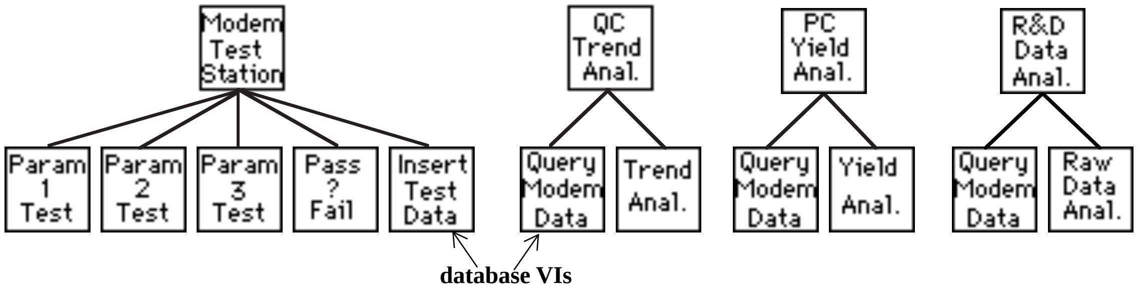


Figure 39. Modem Application Hierarchies

<u>Modem Test Station VI</u> - capture, reduce, pass/fail, and insert test data into database.

<u>QC Trend Analysis VI</u> - query, linear fit, and plot test data over selected time spans.

<u>Production Control Yield Analysis VI</u> - query pass/fail results and plot yield percentage over selected time spans.

<u>R&D Data Analysis VI</u>- query the path names and retrieve and plot the raw test station data for select test runs.

First, an ODBC data source is defined, called “QC Test Database”, using the ODBC Administrator function of the *ODBC Administrator,* pg. 6 - 3. A prototype QC database table is constructed on the local database installation using suitable database tools[1]. Prototypes of the various SQL statements necessary to write/read QC test data to/from the database are also developed. The database is then tested using sample QC data to confirm the design.

1. Suitable tools for database design and prototyping are often included with the database system software. Alternately, the user may quickly develop handy tool VIs using DBVIEW.

Application Coding and Testing

The subVIs are coded bottom-up, i.e. starting with the lowest subVIs and working up the hierarchy. Just as the application specific VIs are tested alone against their respective environments (hardware, software, etc.); the database subVIs are tested alone against the database. At this point communication channels for remote database access, if any, are established.

Two modem test database subVIs are coded and tested:

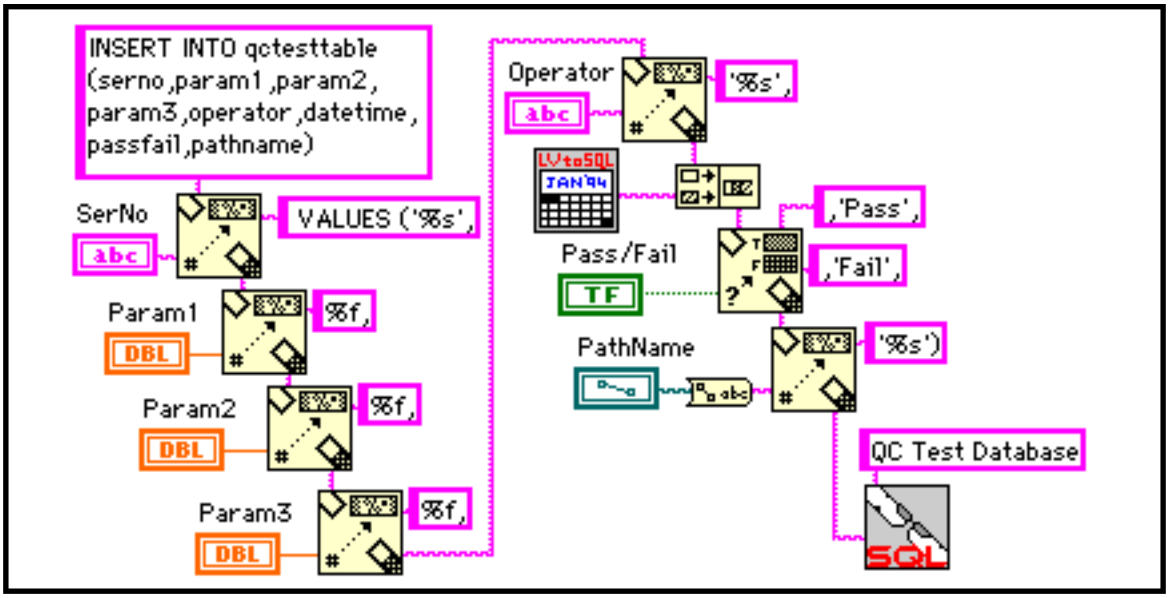


Figure 40. Insert QC Test Data VI Block Diagram

The Insert QC Test Data subVI (see Figure 40) accepts the results of the QC tests, the test date/time, the operator ID, and the pass/fail determination. From these, the VI builds the necessary SQL string and then calls Complete SQL Session (pg. 4 - 5) to establish the database session, insert the test results data, and conclude the session.

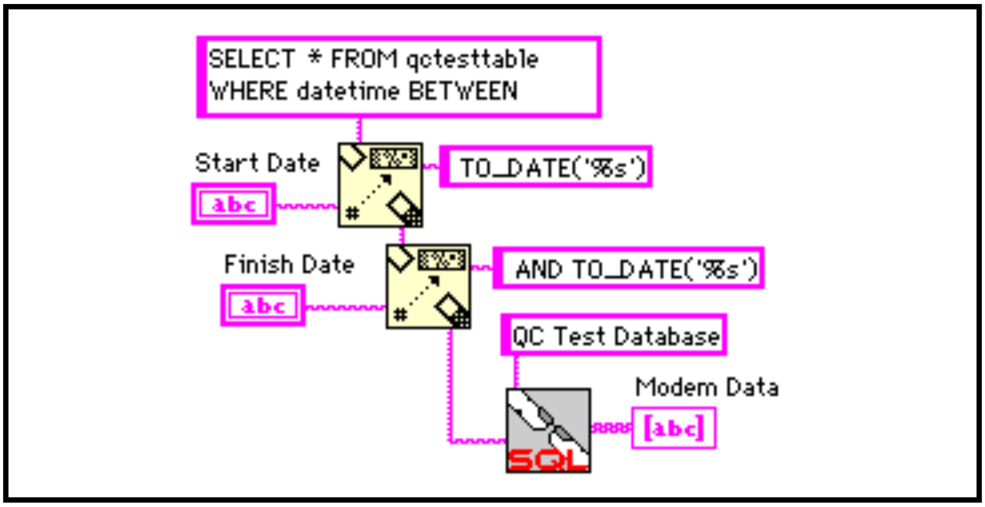


Figure 41. Query Modem Data VI Block Diagram

The Query Modem Data subVI (see Figure 41) retrieves QC test results from the database over a range of production dates. It accepts start and finish dates; builds a SQL statement to retrieve the data; then calls DBVIEW VIs to perform the database query. The results are presented for subsequent analysis and display.

Database VIs are unit tested on the local database with sample modem QC data. The contents of the tables are examined regularly at this stage to confirm correct data handling. Testing is ultimately performed on the corporate server database when it becomes available. This is so that correct connection strings can be determined for the production ODBC data source, and so that database performance optimization can be performed under realistic network conditions.

Application Integration

When unit testing is complete, application specific and database VIs are integrated and tested together with well known data to confirm correct end-to-end operation. After testing is complete, typically the database is "flushed", or cleared of all test data. The application is then ready for real production operation.

The modem test VI is integrated with the Insert QC Test Data subVI and tested on the real test station. The analysis VIs are similarly integrated with the Query Modem Data subVI and run on their target computers. The integrated VIs are tested with a well characterized set of production modems to confirm correct results and to benchmark performance over the network.

Operation and Maintenance

Initially, the integrated application must be carefully and continuously monitored to assess overall system and database performance under actual loading. Configuration control and backup strategies must also be implemented, as well as any necessary periodic maintenance and archiving plans for old or stale data. A user training and trouble reporting system should be instituted to provide ongoing support to the user community. Finally, a long term growth program, if any, should be initiated.

The Modem QC Test database hardware was sized to maintain 6 months of production data on line, and after running the production system for some time, the 130000 rows are surpassed; causing system performance to degrade. An "Archive and Delete Old Data" VI must be developed that archives data older than 6 months to tape, and deletes those rows from the database. This VI must be incorporated into periodic maintenance procedures.

After operational experience, it is learned that the corporate server goes off-line daily for backup, while the production line is still running. A new scheme is devised to detect this condition and use the local database on the test computer to buffer test results until the server has reactivated.

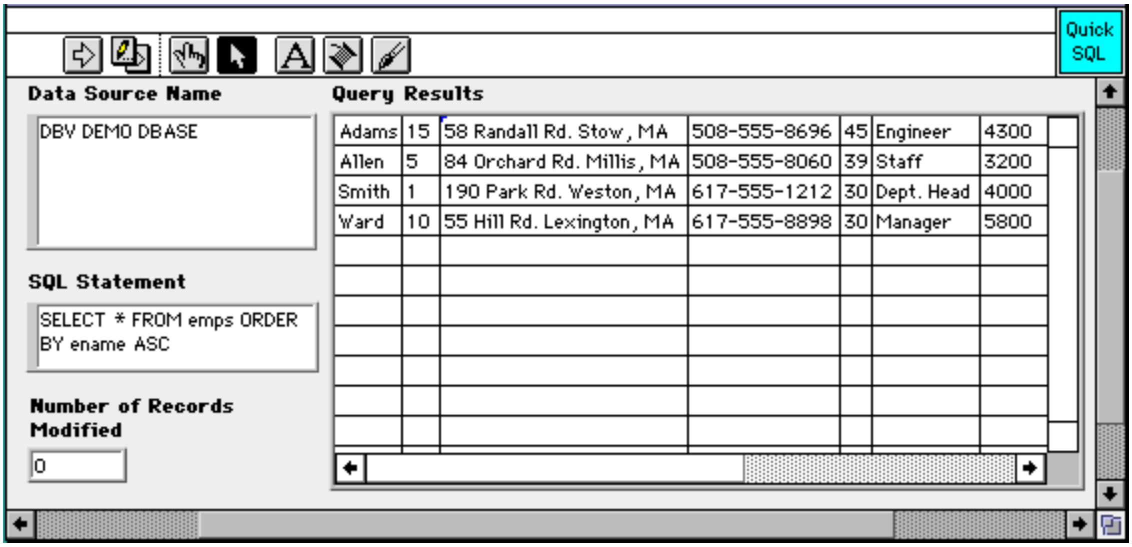
Application Examples

The following sections provide several working examples of how the DBVIEW library is used to develop typical LabVIEW application VIs. The examples illustrate the basic steps and VI calls necessary to enter data into the database and retrieve data from the database. They are constructed from both Access and Interface VIs, and are intended to show the user how the VIs are combined to achieve database operation. The VIs themselves are (optionally) installed as part of the examples group (see page 1-2). The user is encouraged to execute and explore these VIs, and use and adapt them wherever possible to your own applcations.

Note: All DBVIEW example VIs employ dBASE tables. The VIs variously create new tables and/or use existing sample tables that have been provided with the DBVIEW distribution.

Example 1 - Quick SQL

This example illustrates DBVIEW database programming at its simplest. This VI will execute any valid SQL statement, and display any SELECT results. The Quick SQL front panel is shown in Figure 42.



Data Source Name

DBV DEMO DBASE

SQL Statement

SELECT * FROM emps ORDER BY ename ASC

Number of Records Modified

0

Query Results

Adams	15	58 Randall Rd. Stow, MA	508-555-8696	45	Engineer	4300
Allen	5	84 Orchard Rd. Millis, MA	508-555-8060	39	Staff	3200
Smith	1	190 Park Rd. Weston, MA	617-555-1212	30	Dept. Head	4000
Ward	10	55 Hill Rd. Lexington, MA	617-555-8898	30	Manager	5800

Figure 42. Quick SQL Front Panel

The Quick SQL block diagram, shown in Figure 43, employs a single Access VI, **Complete SQL Session** (pg. 4 - 5), which connects to dBASE; executes the SQL statement; retrieves the query results; disposes of the SQL reference; and disconnects from dBASE to terminate the session. The query results are displayed in the Query Results table indicator, while the number of modified rows (if any) and DBVIEW or database errors are displayed in their respective string indicators.

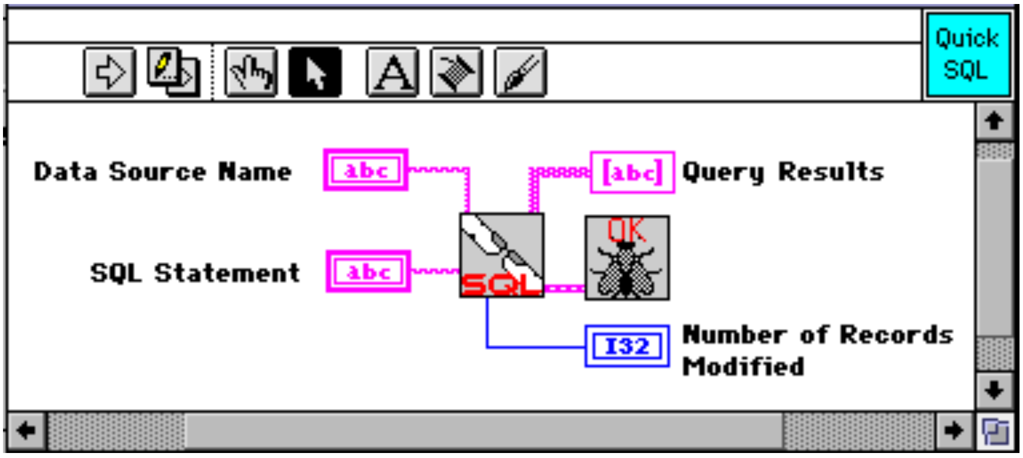


Figure 43. Quick SQL Block Diagram Elements

This VI may be executed against the sample dBASE tables provided with DBVIEW.

Example 2 - Demonstration

This second example consists of a generic end-to-end demonstration of how a database application interacts with a database. The VI displays pop-up subVI panels to explain each stage of the database operations as follows:

1. Connect to dBASE.

2. Create a table in the database to store the demo data.

3. Loop to populate the table with sample data.

4. Query the database for specific data and plot it.
5. Query the database again with different search criteria and plot it.
6. (optionally) Drop the table from the database on request.
7. Disconnect from the database.

In particular the **Demo - Populate Demo Data** subVI illustrates one method for looping operations. This VI creates a table named MVDemo in the database.

- Click on the LabVIEW run arrow to start the demo.

Example 3 - Employee Records using Dynamic SQL

Employee Records presents a typical human resources database that contains records about employees in a small engineering company. The primary purpose of this example is to illustrate how to build SQL statements “on-the-fly” based on user inputs; a technique known as dynamic SQL. The application VI works with a dBASE table, “emps”, provided with the DBVIEW software. It allows the user to insert new employees, query and examine existing employees, and delete employees. Each of these functions is performed through independent SQL statements that are built using simple LabVIEW string functions.

- Click on the LabVIEW run arrow to start the example.
- Click on the **Show SQL** control to view the SQL statements that are created dynamically during subsequent activity. These are displayed in dialog-box windows.

To insert a new employee into the database:

- Click on the **Insert** control.

A pop-up dialog VI is presented to prompt the user for information about the employee to insert:

- Enter the employee information into the string controls provided.
- Click on **OK** to insert the new employee.

To query and examine existing employees in the database:

- Click on the **Query** control.

A pop-up dialog VI is presented for the users to specify which employees to select and display:

- Enter the employee search criteria in the string controls provided.
- Click on **OK** to perform the query.
- Click on << and >> to cycle backward and forward through the selected employees.

To delete an existing employees from the database:

- Click on the **Delete** control.

A pop-up dialog VI is presented to prompt the users for which employees to select and delete:

- Enter the employee search criteria in the string controls provided.
- Click on **OK** to delete the selected employees.

Example 4A -Weather Station DAQ

The Weather Station DAQ is an example of a typical LabVIEW science application. It simulates a weather station data acquisition system that collects data such as temperature, wind speed, barometric pressure, humidity and precipitation. The data is stored to disk hourly after each reading, and at the end of the day statistics are calculated on the past day's data and logged to the database. Statistics are calculated such as the daily high and low temperature, highest wind speed, daily precipitation, average humidity, average barometric pressure, and any snowfall amounts. For simplicity this example only stores the daily summary data to the database (refer to Figure 44).

- Click on the LabVIEW run arrow to start the demo.

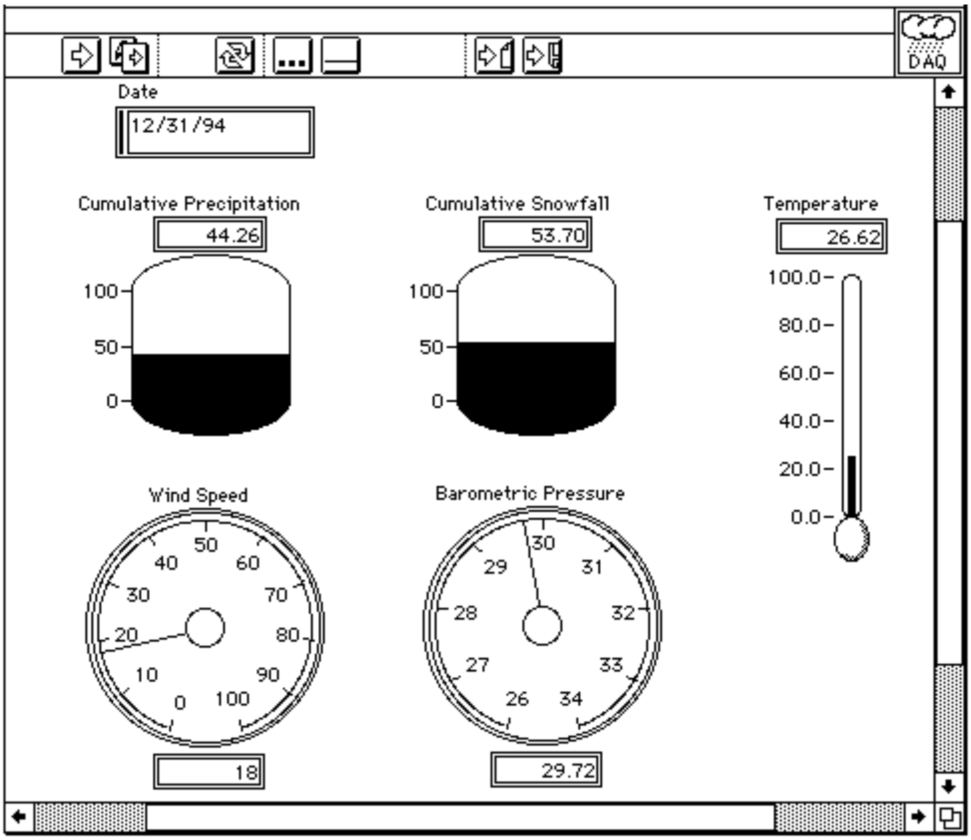


Figure 44. Weather Station DAQ Panel

The application will first create a table called Climate to hold the weather data. It then simulates 365 days worth of weather, logging each to the new table as it goes. Indicators on the front panel will display the progress of the simulation.

Example 4B - Weather Analyzer

This example is provided to show a typical database query application. It is built primarily from the Access VIs to illustrate their simplicity. In particular, it illustrates different user interface techniques that may be employed so that end users may perform database queries without learning SQL.

Note: Weather Analyzer uses the Climate table created by the Weather Station DAQ example. Weather Station must be run successfully before running Weather Analyzer.

The Weather Analyzer illustrates two different approaches to developing a user interface for computing and viewing common weather statistics (e.g. high temperature, low temperature, snow, etc.). The first approach employs specific 'canned' queries, which permit the user to quickly examine weather parameters in standard views with a minimum of input, e.g. only a date range**.** The second approach involves "ad-hoc" querying, where the user may specify multiple selection criteria for special weather data studies. In general, VIs providing ad-hoc querying capabilities require more complex block diagrams to implement friendly ("SQL-free") user interfaces.

- Press the Run arrow to start the application.

The Weather Analyzer panel displays four pushbutton choices to view the weather data. **Temperature Data**, **Wind Data** and **Precipitation Data** are canned queries that graph selected parameters for the months specified, while **General Query** allows the user to perform ad-hoc queries.

- Click on any pushbutton to select a query, or **Quit**.

Temperature Data

Upon opening, this subVI performs a canned query for the yearly high, low and average temperature and displays the result. The operator can then (see Figure 45):

- Operate the ring control to select **High Temperature**, **Low Temperature**, or both.
- Enter dates of interest into **Start Date** and **End Date** controls.

• Click on the **Plot** button to query and plot the results.

If no time period is specified the entire year is graphed.

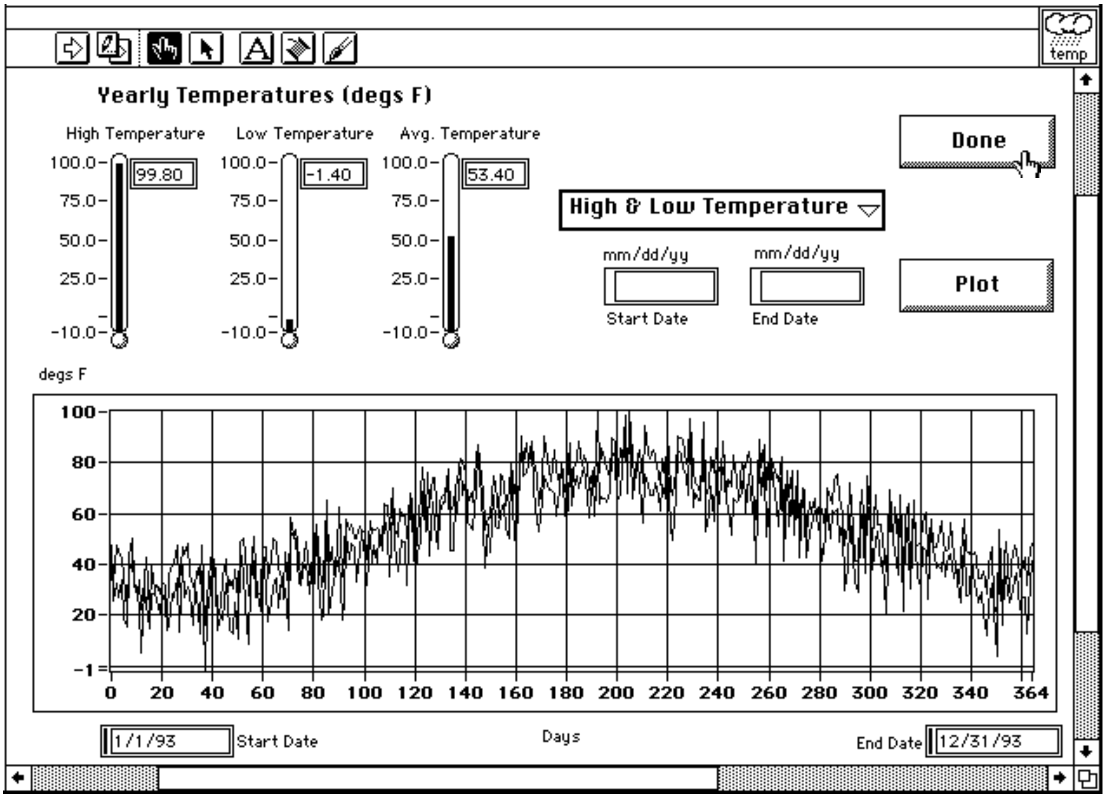


Figure 45. Temperature Data Panel

Wind Data

Upon opening, this subVI performs a canned query for the average wind speed and displays the result. The operator can then (see Figure 46):

- Enter dates of interest into **Start Date** and **End Date** controls.

- Click on the **Plot** button to query and plot the results.

If no time period is specified the entire year is graphed.

- Click on **Done** returns control to the main menu.

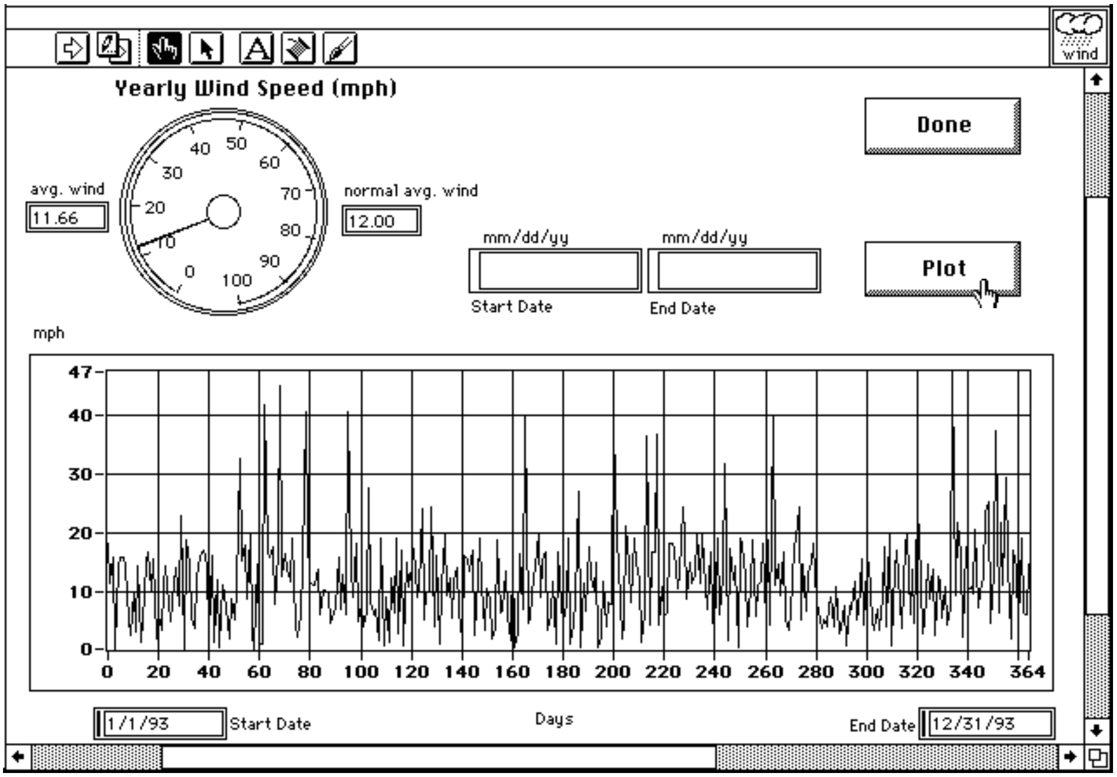


Figure 46. Wind Speed Data Panel

Precipitation Data

Upon opening, this subVI performs a canned query for the average and total yearly rain and snow fall, and displays the result. The operator can then (see Figure 47):

- Operate **Plot Monthly Totals** to select **Snowfall**, **Rainfall**, or **Total Precipitation**.
- Click on the **Plot** button to query and plot the monthly totals in histogram form.
- Click on **Done** returns control to the main menu.

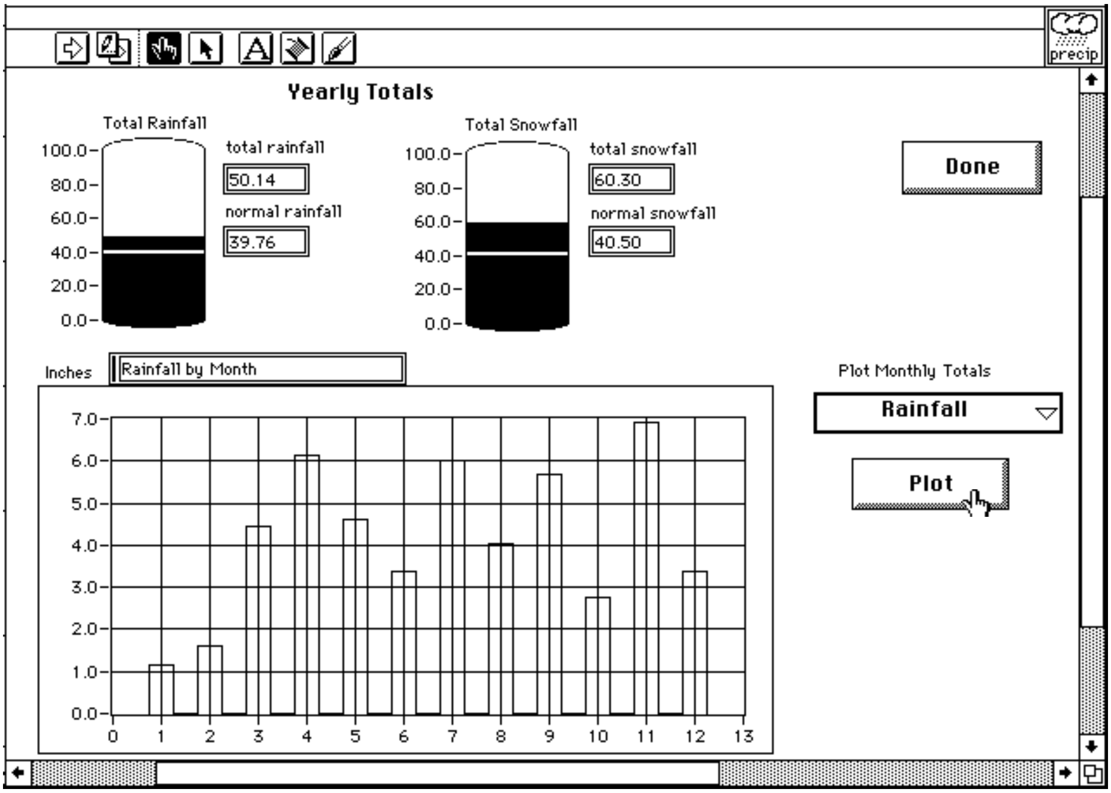
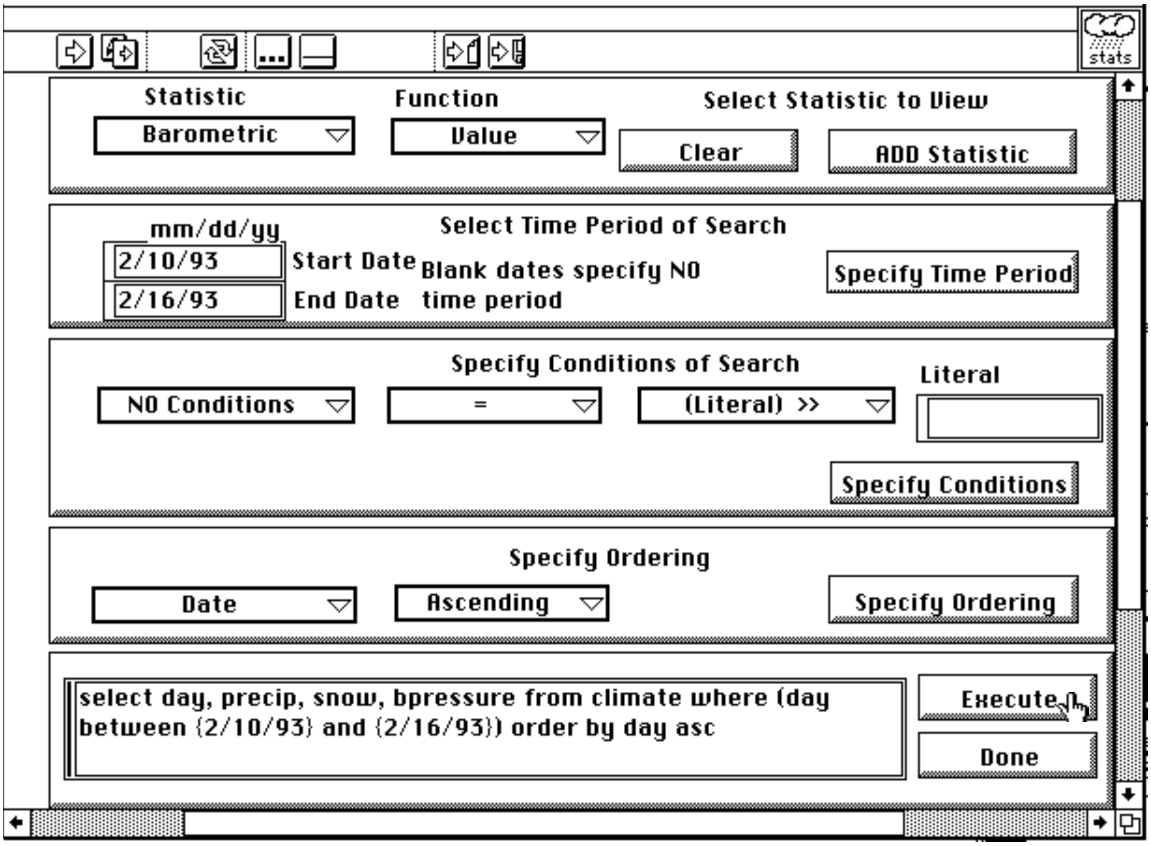


Figure 47. Precipitation Data Panel

General Query

General Query allows the user to interactively build ad-hoc weather data queries. As the request is being built, the SQL equivalent is displayed in the string indicator near the bottom of the panel. There are five sections to this screen (see Figure 48), **Select Statistic to**

View, **Select Time Period of Search**, **Specify Conditions of Search**, **Specify Ordering**, and the **Execute** request section. Each of these are described below.



2/10/93 Start Date
2/16/93 End Date
Blank dates specify NO time period

NO Conditions | = | (Literal) >>

Date | Ascending

```
select day, precip, snow, bpressure from climate where (day
between {2/10/93} and {2/16/93}) order by day asc
```

Figure 48. Weather General Query Panel

Select Statistic to View

The user may select one or more weather parameters to retrieve by repeating the following:

- Operate the **Statistics** ring control to select a parameter to query.

- Operate the **Function** ring control to select the parameter value or a function to perform on the selected parameter.

- Click on the **ADD Statistic** button to enter the choices.

- or -

- Click on the **Clear** button to begin again.

Select Time Period of Search

The user next specifies the date range for the query:

- Enter dates into the **Start Date** and **End Date** string controls (e.g. 06/21/93).

- Click on the **Specify Time Period** button to accept the dates.

If no date is entered, the time period condition is removed from the query statement when **Specify Time Period** is clicked.

Specify Conditions of Search

In this section, the user further restricts the selection by specifying filter conditions, which compare specific weather parameters against other weather parameters or constant values. This section consists of three ring controls which are used to construct the conditional portion of the query statement.

- Operate the left-most ring control to select a weather parameter to apply the condition to, or **NO Conditions** if none are desired.
- Operate the center ring control to select a comparison operator.
- Operate the right-most ring control to select the object of the comparisor.
- Type a value into the **Literal** control if **(Literal) >>** was chosen above.
- Click on the **Specify Conditions** button to add the conditional to the query statement.

Specify Ordering

This section allows the user to specify a sort order for the query results.

- Operate the left-most ring control to choose a weather statistic to sort on, or **No Ordering** if none is desired.
- Operate the next ring control to select **Ascending** or **Descending** order.
- Click on the **Specify Ordering** button to add this to the query.

The ad-hoc query is now complete.

- Click on the **Execute** button to initiate the query.

- or -

- Click on the **Done** button to return to the main menu.

Example Scenario:

The following scenario illustrate how the General Query panel would be used to fulfill the following request:

‘Display the date and amount of every snowfall for the year and order the results by the amount.’

Select Statistics to View:

- Select **Day**, **(value of)**, and press the **ADD Statistic** button.
- Select **Snow**, **(value of)**, and press the **ADD Statistic** button.

Specify Time Period of Search: none.

Specify Conditions of Search:

- Select **Snow**, **>** (greater than), **(Literal) >>**, and enter **0** into the **Literal** control.

Specify Ordering:

- Select **Snow**, **Descending** and then click on the **Specify Ordering** button.

The Following SQL statement will be built:

SELECT day, snow FROM climate WHERE (snow > '0') ORDER BY snow desc

Example 5 - SQL Optimization Demo

In this example, a single data query has been implemented in three different ways to illustrate the substantial benefits of optimized SQL statements. Explore this example's annotated block diagram to learn how a little extra time invested in a concise SQL statement can save substantial block diagram complexity and programming time.

Appendix A SQL Quick-Reference

This appendix lists and briefly explains the more common SQL commands, operators, and functions available for DBVIEW databases. These commands, in particular, are applicable to the file-based databases such as Btrieve, dBASE, Paradox, etc. The intention is not to provide a comprehensive SQL description, but rather to make users aware of the various SQL elements at their disposal. Refer to the ‘SQL for Flat-File Drivers’ topic in the DatabaseVIEW Driver Help for SQL syntax specific to the flat-file drives supplied with this product. Consult the database vendor documentation for details on specific SQL implementations for other databases. General-purpose and tutorial texts on SQL are listed in the *Reference Reading List*, pg. 1 - 19.

The appendix is organized as 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

The following table lists SQL commands that may be used with the DBVIEW VIs.

Table 6. 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

Table 6. SQL Commands

SQL Command	**Syntax**	**Description and Examples**
DROP TABLE	DROP TABLE table_defn	DROP is used to remove database tables. DROP TABLE tab1
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

The following table lists and defines SQL objects, which are the building blocks for all SQL statements.

Table 7. SQL Objects

Object Abbr.	Object Name	Description and Examples
table_defn	Table Definition	Describes a table to perform an operation on. It may be just a table name, or may 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. See DBVIEW On-Line Help for database-specific column data types. 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 7. SQL Objects

Object Abbr.	**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

The following table lists and defines SQL clauses, which are used to fully specify SQL commands.

Table 8. SQL Clauses

Clause Name & Syntax	Applicable Commands	Description and Examples

FROM table_defn [options] [table_alias]	SELECT
	DELETE
		Describes a table to perform an operation on. It may be just a table name, or may 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 may 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

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

Table 8. SQL Clauses

Clause Name & Syntax	**Applicable Commands**	**Description and Examples**
ORDER BY {sort_expr [DESC or ASC]}...	SELECT	Used 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	Used to lock columns in selected rows for updates or deletion. (See *Database Driver Help*, pg. 6 - 8) SELECT * FROM C:\qcdata\modtest\MT WHERE (MT.fld1 = 3 AND MT.fld2 <= 365) FOR UPDATE OF MT.fld1, MT.fld3

SQL Operators

The following table lists SQL expressions and operators.

Table 9. 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	=	equal	WHERE a = b
(true / false)	<>	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’)
	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

Table 9. SQL Operators

Operator Class	**Operators**	**Description**	**Example**
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
	COUNT(*)	count of all rows	SELECT COUNT(*) from tab1
	DISTINCT	only non-duplicate rows	SELECT DISTINCT * FROM ...

SQL Functions

The following table lists SQL functions.

Table 10. 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)	leftmost 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
MAX(number_expr1,number_expr2)	larger of number_expr1 andnumber_expr2
MIN(column_expr)	minimum value of column_expr

Table 10. SQL Functions

Function	**Description**
MIN(number_expr1,number_expr2)	smaller of number_expr1 andnumber_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
DATE()	returns current date in date format
TODAY()	returns current date in date format

Table 10. SQL Functions

Function	**Description**
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

Appendix B Glossary

Access VIs - DatabaseVIEW VIs that provide the simplest, most direct access to databases with a minimum of programming.

Active Set - the results of a SQL SELECT statement consisting of all table rows that match the selection criteria. Only rows and columns from the active set may be fetched into a program.

ANSI - American National Standards Institute

API - see Application Programming Interface

Application Programming Interface - allows a software program to programmatically invoke services from another software program.

Automated Test Equipment (ATE) Systems - a class of systems that employ combinations of general and special purpose computers to automate testing of some article or process.

Client-Server Operation - a mode of computer software interaction whereby one software program, the 'server', responds to requests from one or more 'client' programs; often over a communications network.

Cluster - a LabVIEW control/indicator consisting of a collection of data element controls/indicators.

Commit - make the effects of the SQL statements in a transaction permanently recorded in the database.

Computer Integrated Manufacturing (CIM) and Test Systems - a class of systems that use general and special purpose computers to integrated and automate the design, manufacturing, and test processes.

Connection - the distinct data structures that are established when a logical channel is established from an application program to a database.

Data Dictionary - descriptions of the columns and rows in database tables that are accessible programmatically.

Data Source Name (DSN) - a name associated with a particular defined database that is used when connecting to the database.

Database / Enterprise Integration - a class of problems that focusses on integrating and sharing information in databases across all enterprise organizations as a means to improve productivity.

Database Drivers - ODBC compliant dynamic link library files that provide programmatic access to specific database functions.

Dynamic Link Library (.dll) Files - distinct software modules developed with standard calling interfaces that are loaded and executed at run-time.

Dynamic SQL - Use of Structured Query Language whereby the SQL statement is built programmatically in response to user input rather than apriori.

Encapsulate - to implement a standard program call interface for a software module. Medium accomplishes this through the LabVIEW Code Interface Node mechanism.

Engine-based - databases that have an explicit software program, the "server" that services database requests from other application programs, the "clients".

File-based - databases that do not have an explicit software engine, but instead are accessed directly through their disk file.

FIPS - Federal Information Processing Standards

IBM - International Business Machines Corporation.

Index - a system of pointers maintained by some databases to optimize query performance. They are applied to specified key columns in database tables.

Interface VIs - DatabaseVIEW VIs that provide detailed access to database functionality.

ISO - International Standards Organization

Laboratory Automation Systems - a class of systems that employ general purpose computers to control and monitor the flow of test samples through an automated laboratory analysis process.

Laboratory Information Management Systems (LIMS) - a class of system that controls and automates the capture and flow of information within a Laboratory Automation System.

Network Protocol - signalling rules that form the basis for communication between computer processes over a network.

Open Database Connectivity (ODBC) - an API and driver standard for accessing databases; developed by Microsoft Corp.

Quality Control Test Systems - a class of systems that often employ automated test equipment to perform in-process and final quality testing in a manufacturing process.

Relational Data Model - an information analysis and modeling technique, based in predicate logic, that allows information (entities) in several database tables to be related or "joined" through specific "key" columns.

Relational Database Management System (RDBMS) - typically high-performance server-based database systems that implement inter-table relations functions, or, joins.

Rollback - the act of permanently undoing a transaction from the database. All changes made to the database during the transaction are discarded.

SETUP - the DatabaseVIEW for Windows software installation program.

SQL - see Structured Query Language

SQL reference - the distinct data structures that are created when a SQL statement is executed on a database.

SQL session - a related series of operations conducted on a database. A session begins when a connection has been established to a database, and concludes when the connection has been terminated.

Statistical Process Control (SPC) Systems - a class of process control systems that employ computer automated process measurement and statistical analysis to continually optimize process performance.

Structured Development Process - an application development process that employs a structured approach to application analysis, design, and systhesis.

Structured Query Language - an ANSI standard language for expressing database actions.

Supervisory Control & Data Acquisition (SCADA) Systems - a class of systems that employ one or more general purpose computers to control related collections of programmable logic or other embedded controllers, and to collect performance related measuremen

Systems Analysis and Data Modeling - an application development activity that focusses on identifying, analyzing, and specifying the content and form of the application's database.

Top-Down - a design approach whereby the highest-level system characteristics are designed first, followed by successively more detailed design decomposition.

Transaction - a related sequence of Data Manipulation Language SQL statements that must be performed collectively to maintain data consistency.

Utilities - DatabaseVIEW VIs that provide general purpose tools for application development, ODBC administration, and On-Line Help.

VI - see Virtual Instrument

Virtual Instrument - a software program that is built and executed within the LabVIEW graphical software system. It consists of a front panel, block diagram, and connector pane/icon.

Workgroup Data Management - a general class of problems focussing on automating information capture, flow, and sharing among teams of collaborators.

Appendix C
Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a Fax-on-Demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at (512) 418-1111.

E-Mail Support (currently U.S. only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

	Telephone	Fax
Australia	02 9874 4100	02 9874 4455
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 527 2321	09 502 2930
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 5734815	03 5734816
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name ____

Company ____

Address ____

____

Fax (___)__________________ Phone (___) ______________________________

Computer brand ______________ Model ______________ Processor_______________

Operating system (include version number) ________________________________

Clock speed ______MHz RAM _____MB Display adapter ______________________

Mouse ___yes ___no Other adapters installed ______________________________

Hard disk capacity _____MB Brand ______________________________________

Instruments used ____

____

National Instruments hardware product model __________ Revision _______________

Configuration ____

National Instruments software product ___________________________Version __________

Configuration ____

The problem is: ____

____

____

____

____

List any error messages: ____

____

____

The following steps reproduce the problem:___________________________________

____

____

____

____

SQL Toolkit Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

DAQ hardware ______________________________

Interrupt level of hardware ______________________________

DMA channels of hardware ______________________________

Base I/O address of hardware ______________________________

Programming choice ______________________________

LabVIEW or BridgeVIEW version ______________________________

Other boards in system ______________________________

Base I/O address of other boards ______________________________

DMA channels of other boards ______________________________

Interrupt level of other boards ______________________________

Other Products

Computer make and model ______________________________

Microprocessor ______________________________

Clock frequency or speed ______________________________

Type of video board installed ______________________________

Operating system version ______________________________

Operating system mode ______________________________

Programming language ______________________________

Programming language version ______________________________

Other boards in system ______________________________

Base I/O address of other boards ______________________________

DMA channels of other boards ______________________________

Interrupt level of other boards ______________________________

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *SQL Toolkit Reference Manual*

Edition Date: February 1997

Part Number: 321525A-01

Please comment on the completeness, clarity, and organization of the manual.

____

____

____

____

____

____

____

If you find errors in the manual, please record the page numbers and describe the errors.

____

____

____

____

____

____

____

Thank you for your help.

Name ____

Title ____

Company ____

Address ____

____

Phone (___)____________________________ Fax (___) ____________________________

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Index to VIs