

SQL Toolkit for G Reference Manual

September 1997 Edition
Part Number 321525B-01



Internet Support

E-mail: support@natinst.com
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



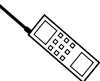
Fax-on-Demand Support

(512) 418-1111



Telephone Support (U.S.)

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



International Offices

Australia 03 9879 5166, 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 09 725 725 11, 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, United Kingdom 01635 523545

National Instruments Corporate Headquarters

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

Important Information

Warranty

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

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

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

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

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

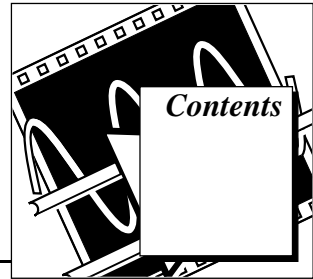
Trademarks

National Instruments™, natinst.com™, LabVIEW®, and BridgeVIEW™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.



About This Manual

Organization of This Manual	xiii
Conventions Used in This Manual	xiv
How to Use the SQL Toolkit Documentation Set	xv
Related Documentation.....	xv
Customer Communication	xvi

Chapter 1

Introduction to the SQL Toolkit

Welcome to the SQL Toolkit for G	1-1
Introduction to the SQL Toolkit Operating Environment	1-2
The G Programming Language	1-2
SQL Toolkit.....	1-4
Database Models	1-4
ODBC Standard.....	1-6
Database Communication: Local and Remote Connections	1-7
Network Communications.....	1-8
Structured Query Language (SQL)	1-9
SQL Variants	1-11
SQL Toolkit VIs	1-11
Data Integration: The Conceptual Fit.....	1-11
Functional Integration: The Operational Fit	1-12
Advantage of Databases	1-13
Using SQL Toolkit Online Help.....	1-14

Chapter 2

Using SQL Toolkit VIs

SQL Made Easy	2-1
Complete SQL Session.....	2-1
Handling SQL Errors.....	2-2
Cluster to SQL VIs	2-3
Using the Cluster to Insert SQL Template VI.....	2-3
Using the Cluster to Select SQL Template VI.....	2-4
Using the Results to Cluster Template VI	2-5

Using the Database Browser for Testing.....	2-6
Optimizing Your SQL Sessions	2-7
Understanding the Components of an SQL Session	2-7
Maintain a Connection to a Database	2-8
Eliminate Unnecessary Data Retrieval	2-9
Speed Up Execution Using Dynamic SQL	2-10
Retrieve Data Directly into G Data Types	2-12
Use Efficient SQL Operations and Databases	2-14
Other SQL Operations.....	2-14
Transaction Management.....	2-14
Database and SQL Session Information	2-15

Chapter 3

High-Level SQL VIs

High-Level SQL VI Descriptions.....	3-1
Connect.....	3-1
Complete SQL Session.....	3-2
Easy SQL.....	3-3
Disconnect.....	3-4

Chapter 4

Error Handling VIs and Cluster-to-SQL Template VIs

Error Handling VIs Overview	4-1
Error Handling VI Descriptions	4-1
Abort Error Dialog	4-1
Abort-Continue Error Dialog	4-2
Continue Error Dialog.....	4-2
SQL-to-Cluster Template VIs Overview.....	4-3
SQL-to-Cluster Template VI Descriptions.....	4-3
Cluster to INSERT SQL.....	4-4
Cluster to SELECT SQL	4-4
Results to Cluster	4-5

Chapter 5

Transaction Operation VIs

Transaction VI Descriptions.....	5-1
Commit Transaction.....	5-1
Rollback Transaction.....	5-2
Start Transaction	5-2

Transaction Locking VI Descriptions	5-3
Get ISO Level	5-4
Get Supported ISO Levels	5-5
Set ISO Level.....	5-5

Chapter 6

Advanced SQL VIs

Statement Execution VI Descriptions	6-2
Execute SQL.....	6-2
Set SQL (<i>Windows 3.1</i>) and Append SQL (<i>Windows 3.1</i>) ...	6-3
End SQL	6-3
Dynamic SQL VI Descriptions	6-4
Prepare SQL	6-5
Clear SQL Parameter.....	6-5
Get Number of Parameters	6-6
Execute Prepared SQL.....	6-6
Set SQL Integer,	6-6
Set SQL Long,	6-6
Set SQL Float,	6-6
Set SQL Double,	6-6
Set SQL Binary,.....	6-6
Set SQL Character,	6-6
Set SQL Date,	6-6
Set SQL Date-Time, and	6-6
Set SQL Time	6-6
Set SQL Decimal Parameter.....	6-7
Set SQL Parameter to Null	6-8
Fetch Data Parsing VI Descriptions.....	6-9
Fetch Query Results.....	6-9
Fetch Next Record,	6-10
Fetch Previous Record, and	6-10
Fetch Record N.....	6-10
Get Number of Columns and.....	6-11
Get Number of Records	6-11
Get Number of Modified Records	6-11
Close Fetch Log File.....	6-12
Handle Null Value	6-12
Fetch Character Column Data,	6-12
Fetch Double Column Data,	6-12
Fetch Float Column Data,	6-12
Fetch Integer Column Data,.....	6-12
Fetch Long Column Data, and.....	6-12

Fetch Column Data.....	6-12
Fetch Decimal Column Data	6-13
Column Information VI Descriptions	6-14
Get Column Name,.....	6-15
Get Column Alias,.....	6-15
Get Column Expression,	6-15
Get Column Scale,.....	6-15
Get Column Width, and	6-15
Get Column Precision	6-15
Get Column Attributes	6-16
Get SQL Toolkit Column Type and.....	6-18
Get Database Column Type	6-18
Request,	6-19
Request Database Information,	6-19
Request DSN Information,	6-19
Request Procedure Col Information,	6-19
Request Type Information, and.....	6-19
Request Table Information.....	6-19
Date Conversion VI Descriptions	6-19
G Date to ODBC Date,.....	6-19
G Date to SQL Date, and	6-19
SQL Date to G Date	6-19

Chapter 7 Configuration VIs

Configuration VI Descriptions	7-1
Set Login Timeout and.....	7-1
Get Login Timeout.....	7-1
Set Query Timeout and	7-2
Get Query Timeout.....	7-2
Set Maximum Rows and	7-2
Get Maximum Rows	7-2
Set Statement Options	7-2
Get Statement Options	7-3
Set Database	7-4
Set Fetch Options and	7-4
Get Fetch Options.....	7-4

Chapter 8

SQL Information VIs

SQL Information VI Descriptions.....	8-1
Get Column Information	8-1
Get Database Information.....	8-2
Get DSN Information	8-2
Get Procedure Col Information	8-2
Get References.....	8-3
Get Table Information	8-3
Get Type Information	8-4

Chapter 9

Application Examples

SQL Toolkit Examples	9-1
SQL Toolkit Demo	9-1
Quick SQL.....	9-2
Sample Transaction	9-3
Employee Record	9-4
Weather Examples.....	9-5
Insert Examples	9-9
Query Examples	9-13
Database Browser Utility.....	9-15

Appendix A

SQL Quick-Reference

SQL Commands.....	A-2
SQL Object Definitions	A-4
SQL Clauses	A-6
SQL Operators	A-8
SQL Functions	A-11

Appendix B

Customer Communication

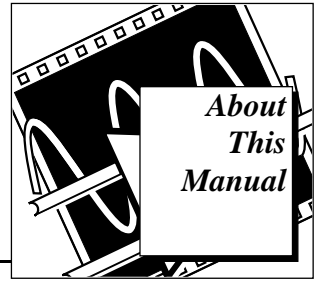
Index

Figures

Figure 1-1.	Virtual Instrument	1-3
Figure 1-2.	Database Table Concept	1-5
Figure 1-3.	Local File-Based Database Connection	1-7
Figure 1-4.	Local Engine-Based Database Connection	1-7
Figure 1-5.	Remote File-Based Database Connection.....	1-8
Figure 1-6.	Remote Client-Server Database Connection	1-9
Figure 1-7.	Conceptual Mapping: Array of Clusters to Database Table	1-11
Figure 2-1.	Using the Complete SQL Session VI to Query a Database	2-1
Figure 2-2.	Modified Cluster to Insert SQL Template	2-4
Figure 2-3.	Modified Cluster to Select SQL Template VI	2-5
Figure 2-4.	Modified Results to Template Cluster VI.....	2-6
Figure 2-5.	The Database Session Sequence	2-7
Figure 2-6.	Complete SQL Session VI Diagram	2-8
Figure 2-7.	Easy SQL VI Diagram	2-9
Figure 2-8.	Inserting Multiple Records	2-10
Figure 2-9.	Improving Performance with Dynamic SQL	2-11
Figure 2-10.	Main Portion of Fetch Query Results	2-12
Figure 2-11.	Modified Fetch Query Results	2-13
Figure 9-1.	Quick SQL Panel	9-2
Figure 9-2.	Quick SQL Block Diagram.....	9-3
Figure 9-3.	Sample Transaction Demo Diagram	9-4
Figure 9-4.	Weather Station DAQ Panel	9-5
Figure 9-5.	Weather Station DAQ Diagram	9-6
Figure 9-6.	Canned Query Example Panel	9-7
Figure 9-7.	Weather Query Generator Panel	9-8
Figure 9-8.	Insert Using ANSI SQL Demo Diagram	9-9
Figure 9-9.	Example Dynamic SQL INSERT Diagram	9-10
Figure 9-10.	Example of Cluster to SQL INSERT Diagram.....	9-11
Figure 9-11.	Insert Using Clusters and Dynamic SQL Demo Diagram	9-12
Figure 9-12.	Example ANSI SQL SELECT to 2D Array of Strings Diagram.....	9-13
Figure 9-13.	SQL Query Results into Standard Data Types Demo.....	9-14
Figure 9-14.	Cluster Query to Cluster Results Demo.....	9-15

Tables

Table 1-1.	SQL Toolkit Data Types	1-5
Table 1-2.	Common SQL Commands	1-10
Table 1-3.	Example Query Results	1-10
Table 1-4.	SQL-to-G Functional Correspondence.....	1-12
Table 6-1.	Data Column Type Codes and Descriptions	6-18
Table A-1.	SQL Commands	A-2
Table A-2.	SQL Objects	A-4
Table A-3.	SQL Clauses	A-6
Table A-4.	SQL Operators	A-8
Table A-5.	SQL Functions	A-11



The *SQL Toolkit for G Reference Manual* describes the features, functions, and operation of the SQL Toolkit. This manual also describes how to use SQL Toolkit VIs to access databases from the G programming environment. It applies to all platform versions of the SQL Toolkit.

The *SQL Toolkit for G Reference Manual* is intended for use by G application developers. The manual presents database and G programming information only as it pertains to building application VIs that use SQL Toolkit VIs in their block diagrams. To use this manual effectively, you should be familiar with the functions and operations of both the G programming language and your chosen databases. The National Instruments technical staff strongly recommends that you acquire the full developer software package for the chosen databases.

Organization of This Manual

The *SQL Toolkit for G Reference Manual* is organized as follows:

- Chapter 1, *Introduction to the SQL Toolkit*, describes the SQL Toolkit, and contains system configuration and basic information that explains how to start using the SQL Toolkit to connect the G programming language to databases.
- Chapter 2, *Using SQL Toolkit VIs*, describes how you can use the SQL Toolkit to communicate with databases.
- Chapter 3, *High-Level SQL VIs*, provides a detailed description of the High-Level SQL VIs, which perform high-level database operations.
- Chapter 4, *Error Handling VIs and Cluster-to-SQL Template VIs*, provides detailed descriptions of the Error Handling VIs and the Cluster-to-SQL templates, which perform high-level database operations.

- Chapter 5, *Transaction Operation VIs*, describes the Transaction Operation VIs, which can be executed at any time during an active session.
- Chapter 6, *Advanced SQL VIs*, introduces the Advanced SQL VIs, which provide detailed low-level access to database services and which, in many cases, form the elemental building blocks of the other SQL Toolkit VIs.
- Chapter 7, *Configuration VIs*, presents the VIs that allow you to configure SQL Toolkit applications.
- Chapter 8, *SQL Information VIs*, provides descriptions of the SQL Information VIs.
- Chapter 9, *Application Examples*, presents example applications that incorporate SQL Toolkit VIs.
- Appendix A, *SQL Quick-Reference*, lists and briefly explains the more common SQL commands, operators, and functions available for SQL Toolkit databases.
- Appendix B, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

bold	Bold text denotes a parameter, menu name, palette name, menu item, return value, function panel item, or dialog box button or option.
<i>italic</i>	Italic text denotes mathematical variables, emphasis, a cross reference, or an introduction to a key concept.
<i>bold italic</i>	Bold italic text denotes an activity objective, note, caution, or warning.
monospace	Text in this font denotes text or characters that you should literally enter from the keyboard. Sections of code, programming examples, and syntax examples also appear in this font. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, variables, filenames, and extensions, and for statements and comments taken from program code.
<>	Angle brackets enclose the name of a key on the keyboard—for example, <PageDown>.

- A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Control-Alt-Delete>.
- <Control> Key names are capitalized.
- » The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options»Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Fonts** option from the last dialog box.
- paths Paths in this manual are denoted using backslashes (\) to separate drive names, directories, and files, as in `C:\dir1name\dir2name\filename`.

How to Use the SQL Toolkit Documentation Set

We suggest you begin by reading Chapter 1, *Introduction to the SQL Toolkit*, which gives you a hands-on introduction to the SQL Toolkit.

When you are familiar with the material in Chapter 1, you can begin to use the *SQL Toolkit for G Reference Manual*.

Related Documentation

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

INTERSOLV ODBC Driver Documentation

- SQL Toolkit Driver Help (available on Windows 95 and Windows NT only)
- SQL Toolkit Driver Readme (available on Windows 95 and Windows NT only)

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 Instructional Publications

- *A Visual Intro to SQL*
J. Trimble, et al ISBN 0-471-61684-2
- *The Practical SQL Handbook*
J. Bowman et al ISBN 0-201-62623-3
- *A Guide to the SQL Standard*
C. Date, et al ISBN 0-201-55822-X
- *Instant SQL Programming*
Joe Celko ISBN 1-874416-50-8

Database Application Design

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

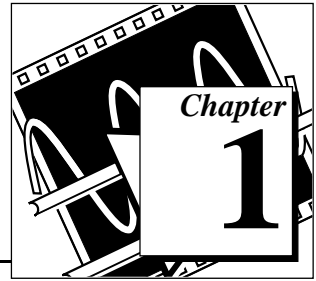
Open Data Base Connectivity (ODBC) Standard

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

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix B, *Customer Communication*, at the end of this manual.

Introduction to the SQL Toolkit



This chapter describes the SQL (Structured Query Language) Toolkit. It also contains system configuration and basic information that explains how to start using the SQL Toolkit to connect the G programming language to databases.

SQL is a language for defining and manipulating data in a relational database. In accordance with the relational model of data, the database is a set of tables where relationships are represented by the values in the tables. You retrieve data by specifying a result table, which can be derived from one or more base tables.

This chapter also includes the following:

- system configuration
- brief descriptions of G, the supported databases, the Open Database Connectivity (ODBC) standard, the Structured Query Language, and network communications
- an overview of the conceptual link between G and a database data model an analysis the suitability of incorporating database functionality into various classes of G applications

Welcome to the SQL Toolkit for G

The SQL Toolkit connects the G graphical programming language to popular database systems. It includes the SQL Toolkit virtual instrument (VI) library, which provides the following features:

- Acts as an Open Database Connectivity (ODBC) Level 2.0-compliant programmatic bridge.
- Acts as an application programming interface (API) from G to text files, spreadsheets, and many leading database systems.
- Provides G developers with full Dynamic Structured Query Language (Dynamic SQL) access to all database functions and services.

The SQL Toolkit bridges the data acquisition and analysis domain of G to the data storage, retrieval, and management services of databases. This creates a complete technical information system. You can apply these systems to a variety of problems, both local and distributed in nature, in the following areas:

- 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

Introduction to the SQL Toolkit Operating Environment

The SQL Toolkit VI Library provides an ODBC-compatible programmatic bridge between the G virtual instrument and several database systems. It provides G developers with full SQL access to database functions and services from the G block diagram.

The following sections briefly describe the various components of the SQL Toolkit Operating Environment.

The G Programming Language

G is a graphical software development and run-time environment that is designed for scientific and engineering applications, but can be used for other applications as well. Programs developed and executed in G are based on *virtual instruments* (VIs) (see Figure 1-1). Each VI consists of an on-screen graphical front panel with animated controls, such as knobs, switches, and buttons; an executable block diagram that graphically expresses the functionality of the instrument; and an icon/connector that defines data flow paths to or from other VIs.

You build block diagrams from icons representing front panel I/O, computational functions, and other VIs. You connect icons to other icons with wires that direct the flow of data among the components.

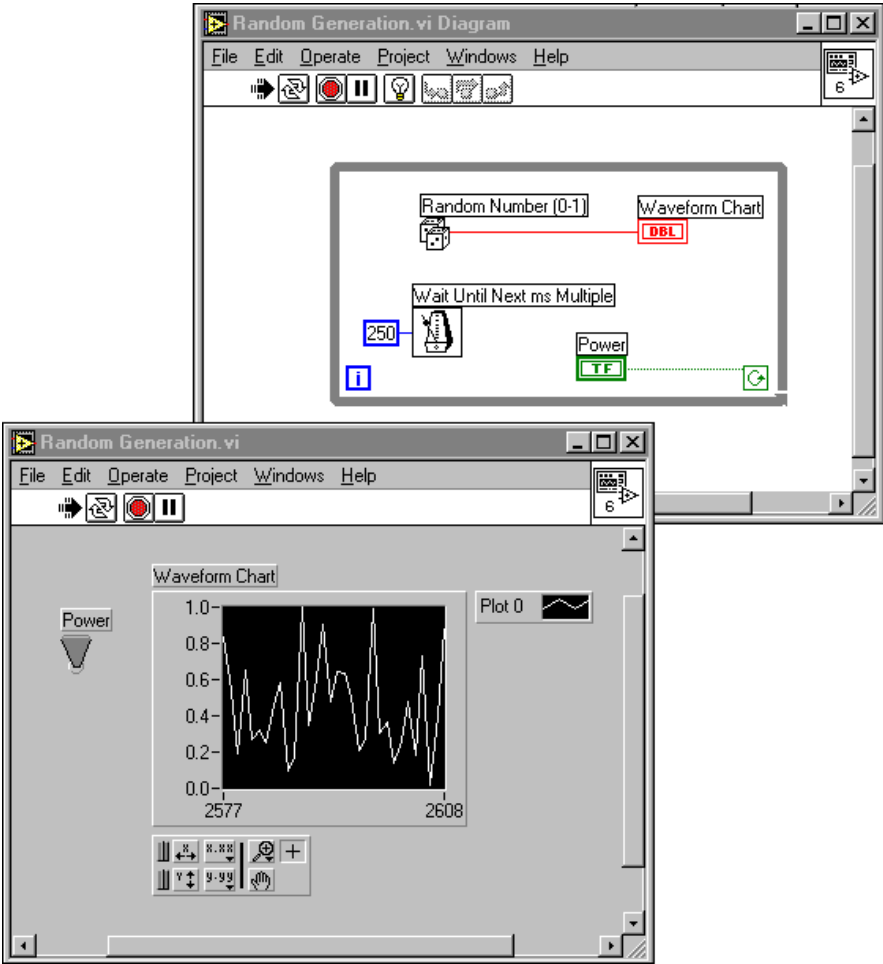


Figure 1-1. Virtual Instrument

For a complete description of the G programming language, refer to the *G Programming Reference Manual*, the *LabVIEW User Manual*, or the *BridgeVIEW User Manual*.

SQL Toolkit

The SQL Toolkit accesses 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 log on to, create, store, modify, retrieve, sort, and manage data in databases located on the same or networked computers. Commands expressed in the Structured Query Language (SQL) perform these functions. Chapter 9, *Application Examples*, provides details on each supported database.

With the SQL Toolkit, G developers can build application programs that access databases using SQL commands.

Database Models

A database model is a logical way to view how data is organized and stored in computers. Popular contemporary database models are based on some data structures called *tables*. Tables consist of rows (or records) and columns (or fields), as illustrated in Figure 1-2. 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 data types, such as CHARACTER (fixed and variable length),

NUMBER, DECIMAL, INTEGER, FLOAT, REAL, DOUBLE PRECISION, DATE, LONG, and RAW.

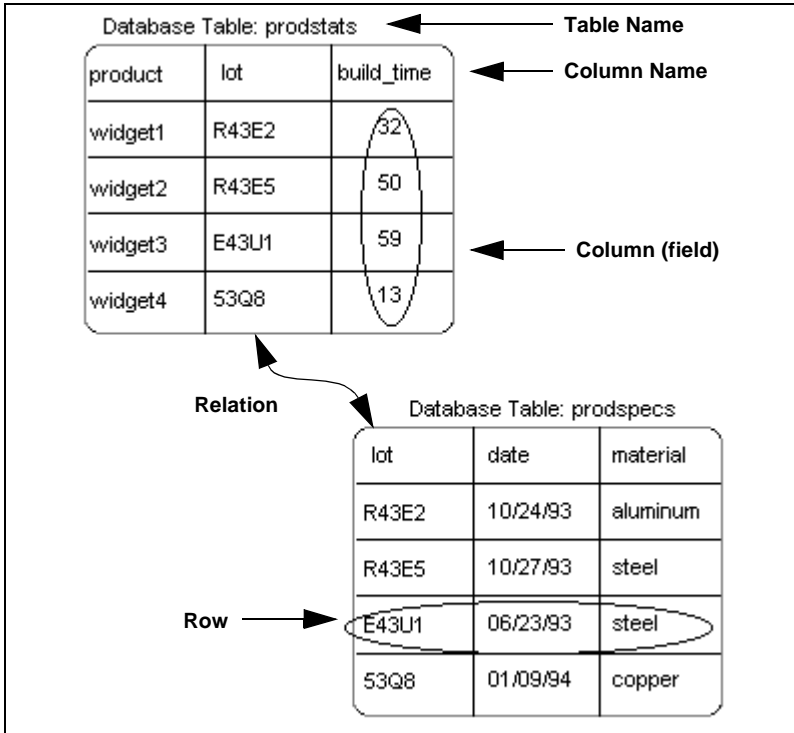


Figure 1-2. Database Table Concept

Most underlying database column types are compatible directly with the SQL Toolkit. To make others that are not directly compatible, map the underlying type into a common set of data types. After doing so, a single G application can access multiple databases without modification, in most cases.

Table 1-1. SQL Toolkit Data Types

Type Code	Data Type Description
1	fixed length character string
2	variable length character string
3	Binary Coded Decimal (BCD) number

Table 1-1. SQL Toolkit Data Types (Continued)

Type Code	Data Type Description
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

Most SQL commands, and therefore column values for `INSERTs` or `UPDATEs` to the databases, are represented as G character strings. Likewise, the databases return most `SELECT` query data as G data strings. To convert column data to or from the appropriate G data types use the built-in string conversion functions (described in the *G Programming Reference Manual*, the *BridgeVIEW User Manual*, and the *LabVIEW User Manual*) and SQL Toolkit Configuration VIs (see Chapter 7, *Configuration VIs*). Alternatively, you can use Dynamic SQL to directly insert and retrieve data using native G datatypes. For more information, refer to the *Dynamic SQL VI Descriptions* and *Column Information VI Descriptions* sections in Chapter 6, *Advanced SQL VIs*.

ODBC Standard

The Open Database Connectivity (ODBC) standard, developed by Microsoft Corporation, provides a common method for applications to access databases. With this standard, you have a uniform way to access databases from a variety of other databases, applications, and platforms. The standard consists of a multilevel application programming interface definition, a driver packaging standard, a common SQL implementation, and a means for defining and maintaining *Data Source Names* (DSN). A DSN is a quick way to refer to a specific database. You specify a DSN with a unique name and by the ODBC driver that communicates with the physical database (local or remote). You must define a DSN for each database to which an application program connects.

The SQL Toolkit and supplied database drivers comply with the ODBC 2.0 standard. Database applications written with the SQL Toolkit supports most databases with little modification.

Database Communication: Local and Remote Connections

A company might locate a database on the computer running G, on a remote computer that is accessed via a network or modem, or on a combination of both. It may be *file-based*, where the SQL Toolkit directly accesses the database disk files to manipulate data or *engine-based*, where the SQL Toolkit communicates with an executing database program to manipulate data, as shown in Figures 1-3 and 1-4.

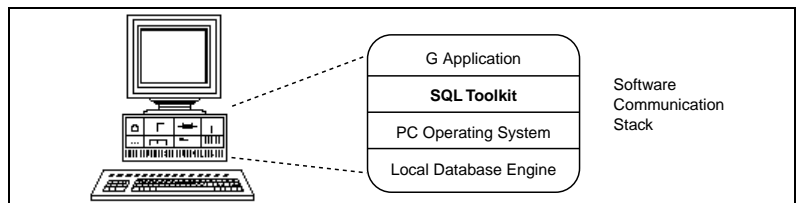


Figure 1-3. Local File-Based Database Connection

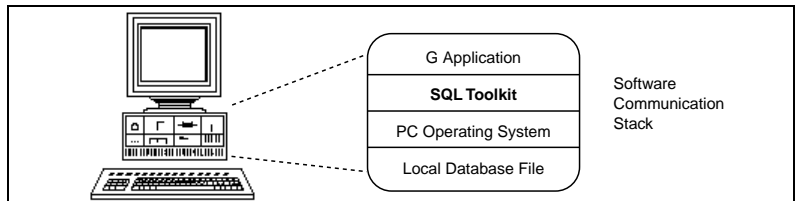


Figure 1-4. Local Engine-Based Database Connection

To begin all database operations, you must establish a *connection* to the desired database(s), which consists of selecting a database and a communications link, and logging on with a user name and password, if necessary. Figures 1-3 and 1-4 above illustrate two possible local and remote database configurations, including the various layers of software needed to implement them (often called a *communication stack*).

Network Communications

With the available network communications packages, application programs can communicate with remote (file-based) and server (engine-based) databases, over a broad range of local and wide-area networks. With the appropriate communications software installed, application programs can access local or remote databases transparently with no additional programming. The user selects a defined designated server network for the desired database into one of the Database Connection VIs. The communications software then transmits the request through the proper network protocol and physical channel.

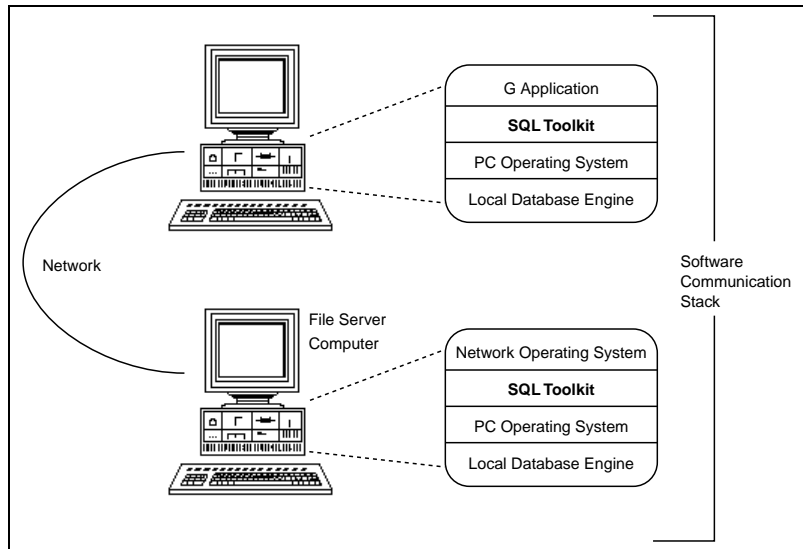


Figure 1-5. Remote File-Based Database Connection

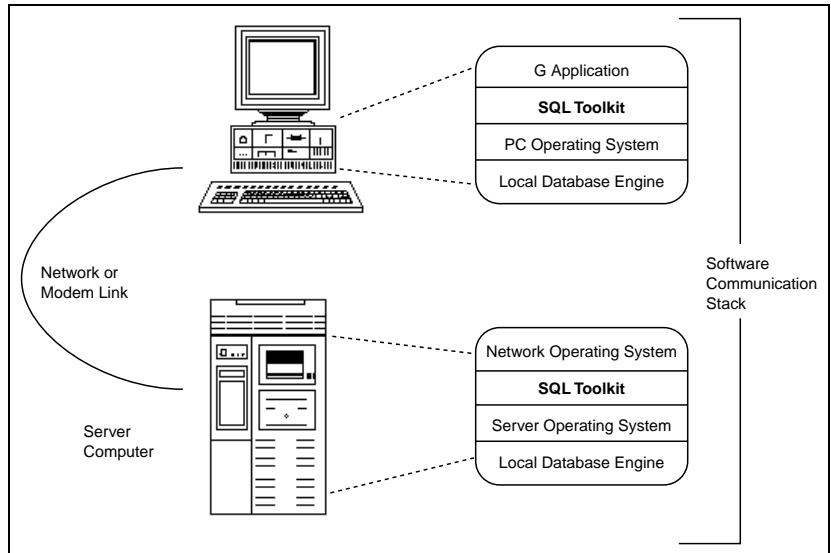


Figure 1-6. Remote Client-Server Database Connection

Structured Query Language (SQL)

Structured Query Language (SQL) is a set of character string commands that all SQL Toolkit Application VIs use to describe, store, retrieve, and manipulate data in accessible databases. IBM developed the language, which became publicly available in the late-1970s. Since then, the American National Standards Institute (ANSI), the International Standards Organization (ISO), and the Federal Information Processing Standard (FIPS) standards have adopted SQL and most major commercial relational database products support it to some degree. It is a non-procedural for processing sets of records in database tables. Below are three pertinent classes of SQL statements:

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

The following table describes some frequently used SQL commands:

Table 1-2. Common SQL Commands

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

A typical SQL statement might read as follows:

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

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.” This query creates a one-row, two-column active set as follows.

Table 1-3. Example Query Results

widget	R43E2
--------	-------

SQL Variants

Standards notwithstanding, several database publishers use 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 accessible databases do not directly support standard SQL functions. Differences are most prevalent when using the CREATE TABLE and ALTER TABLE commands. Most of the databases use their own particular column type keywords. Also, many of the databases have different syntax for date-and-time formats. ODBC-compliant SQL Toolkit software helps to minimize the effects of these SQL variants. Refer to Appendix A, *SQL Quick-Reference*, for details on particular databases.

SQL Toolkit VIs

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

Data Integration: The Conceptual Fit

The conventional (relational) database model is based on tables made up of rows and columns. From a G perspective, a row in a database table corresponds directly to a cluster, and columns in the table correspond to elements within the cluster. Therefore, a database table corresponds directly to a one-dimensional array of clusters, as the following diagram shows.

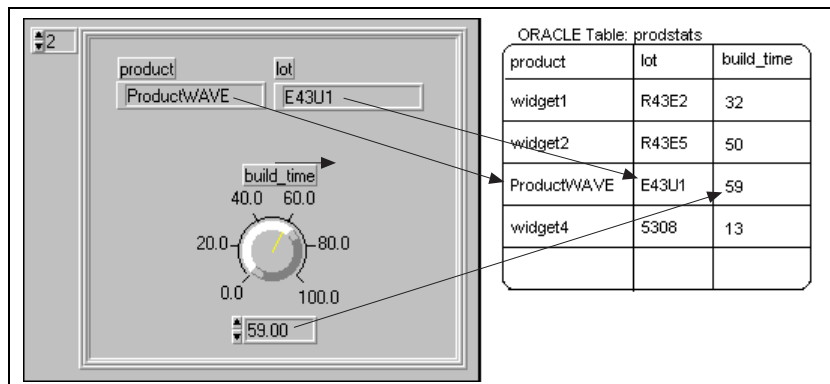


Figure 1-7. Conceptual Mapping: Array of Clusters to Database Table

Functional Integration: The Operational Fit

The cluster metaphor extends beyond data definition and organization into data access and manipulation. SQL data operations on rows in a database table correspond directly to G operations on clusters in a one-dimensional array of clusters. Table 1-4 illustrates the functional mapping between the more frequently used SQL statements described earlier and the equivalent G operations.



Note: *These SQL statements are not supported by all databases.*

Table 1-4. SQL-to-G 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 interrelated conditions
UPDATE	Use Search1D Array as in SELECT, then use Bundler and Replace Array Element in a loop 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: *The last three of these equivalent G functions require that the entire one-dimensional array of clusters reside in memory so that you can perform search/modify operations on each cluster that satisfies the conditions. This is impossible on large data sets that do not fit into memory all at once. Instead, in G, you would manipulate files. Most databases can handle large data sets and make this manipulation simple.*

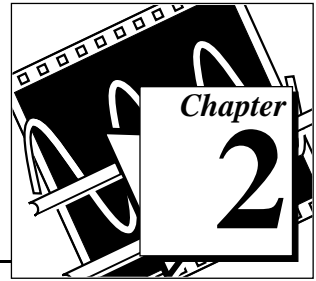
Advantage of Databases

Databases were developed for access control and to rapidly search, retrieve, and sort stored data. Several common features that most databases provide to improve data management and retrieval include the following:

- **Indexing**—Many databases employ a logical device called an index to accelerate the search process. Indexes speed the queries that search for particular values in key, or often used, columns. To do this, the database maintains a separate map of the values in the key field. The maps are searched very quickly to identify which rows in the main table satisfy the query conditions.
- **Data Dictionary**—Many databases employ a logical device, called a *data dictionary*, which tracks the structure of database tables. The data dictionary maintains descriptions of the columns and rows in database tables, and these descriptions are often available through ordinary queries.
- **Concurrent Access**—Many networked databases support many concurrent client connections, which allow several locations to work almost simultaneously on the same data tables.
- **Transactions**—Data can 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 failures occur.
- **Access Control**—Some database systems employ extensive security measures to provide access control, buffering, error checking, and recovery capabilities.
- **Relational Joins**—Many relational databases permit data searches across multiple related database tables simultaneously.
- **Open Access**—Thousands of existing non-G applications can access data stored in many different popular database tables.

Using SQL Toolkit Online Help

The SQL Toolkit provides extensive online help information to help you build database application VIs. You access the online help for SQL Toolkit through the **Help** menu on the block diagram.



Using SQL Toolkit VIs

This chapter describes how you can use the SQL Toolkit to communicate with databases. The first sections describe a set of high-level VIs you can use that make it easy to create, execute, and test SQL statements. The next sections demonstrate how to optimize your SQL operations using lower level SQL VIs, and introduce you to some of the more efficient database commands and configurations.

SQL Made Easy

This section provides information on the high-level VIs available to help you create, execute, and test SQL statements easily.

Complete SQL Session

The SQL Toolkit includes a single all-in-one VI, the Complete SQL Session VI, that you can use to communicate with a database. You access this VI by selecting **Functions»SQL**, which accesses the **SQL** palette. This VI connects to a database, executes an SQL statement, retrieves the results, and closes the connection. If performance is not an issue for you, then this VI is all that you need to communicate with a database.

Figure 2-1 below shows how you could use this VI to query a database using an SQL `SELECT` statement. After calling the Complete SQL Session VI, it checks for errors with the Error Handler VI, which is described in the [Handling SQL Errors](#) section.

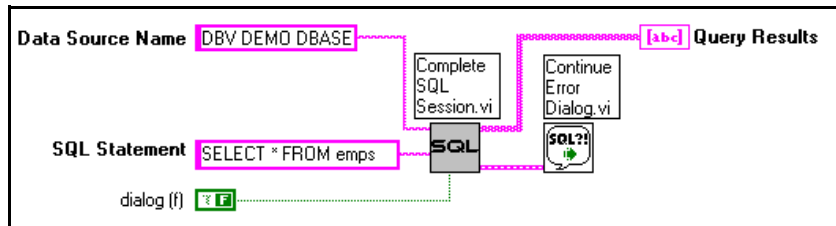


Figure 2-1. Using the Complete SQL Session VI to Query a Database

When you communicate with a database, you specify a Data Source Name (DSN). DSNs specify the database to which you connect. If you connect to a remote database, DSNs specify the network channel and protocol through which to communicate. You use the ODBC Administrator program to define and maintain data sources.

SQL Toolkit includes an example of a database in `Examples\SQL_exmpldb` where the data source name is `DBV Demo DBASE`. The SQL Toolkit installer created the database's DSN and associated it with the example database. If you wanted to connect to a different database, you either could specify the data source name as you configured it in the ODBC Administrator program, or wire up `True` to the **dialog?** input of the Complete SQL Session VI. If you do the latter, dialogs prompt you to select a data source and possibly other connection parameters.

Handling SQL Errors

All SQL Toolkit VIs have **error** inputs and outputs containing error clusters. As with other I/O VIs that have error I/O, if an error is wired up as an input, a VI does nothing. The Disconnect and End SQL VIs are the only exceptions; they attempt to close a connection or end an SQL statement regardless of whether an error is set on the **error** input. If an error passes as an input, it passes unchanged to the output so that you can propagate the error easily to a more convenient time when you can handle or report it.

Because of the multivendor nature of ODBC environments, you cannot know all error code values and their corresponding meaning before hand. Consequently, the General Error Handler VI does not recognize the errors that the SQL VIs might return. The **error string** returned as part of the error cluster should give informative text about each error that occurred. So, even though the General Error Handler does not recognize SQL error codes, **error strings** help alleviate the problem.

You can use the General Error Handler to report errors, or you can use one of the special error handler VIs included in the SQL Toolkit. The Error Handling palette contains VIs that you can use to report an error and give the user the option of aborting or continuing the program. If the user chooses to abort the program, these VIs optionally can take care of closing an outstanding connection. For more complex applications where you might have multiple connections open or other outstanding I/O, you may prefer to use your own error reporting so that you can take

care of closing down whatever additional connections or processes your application controls.

See Chapter 4, *Error Handling VIs and Cluster-to-SQL Template VIs*, later in this manual, for more information on these VIs.

Cluster to SQL VIs

When you execute an SQL statement, you specify that statement as an SQL string. If you are unfamiliar with SQL, Appendix A, *SQL Quick-Reference*, has brief descriptions of common SQL commands. With the high-level SQL VIs, the results are returned as a two-dimensional array of strings, with each row representing a record. Consequently, using the SQL Toolkit VIs can involve formatting and parsing a lot of strings frequently, and converting a lot of G data into SQL commands and converting the results into standard G data types.

The SQL Toolkit has a set of templates designed to simplify working with SQL. Two of these templates make it easy to construct simple `SELECT` and `INSERT` operations. The third template makes it easy to convert the two-dimensional array you might get from a `SELECT` statement into an array of clusters. After you drop one of these templates on a diagram, you should open the VI, save it with a new name, and customize the front panel controls to reflect the type of data on which you want to operate.

See Chapter 4, *Error Handling VIs and Cluster-to-SQL Template VIs*, for more information on the SQL to Cluster Template VIs.

Using the Cluster to Insert SQL Template VI

The Cluster to SQL Insert Template VI is a template you can use to dynamically create an SQL `INSERT` statement based upon a cluster. The template has a cluster on its front panel that you modify to reflect the structure of a record in a table of the database you want to modify. As mentioned earlier, make sure to save a copy of the VI in your own hierarchy rather than modifying the original VI. The following example shows the front panel of the VI modified to insert records continuing a numeric 'serialNumber' and a string 'results' into a table 'testResults'.

The VI has been run, so the string at the right shows the resulting SQL statement.

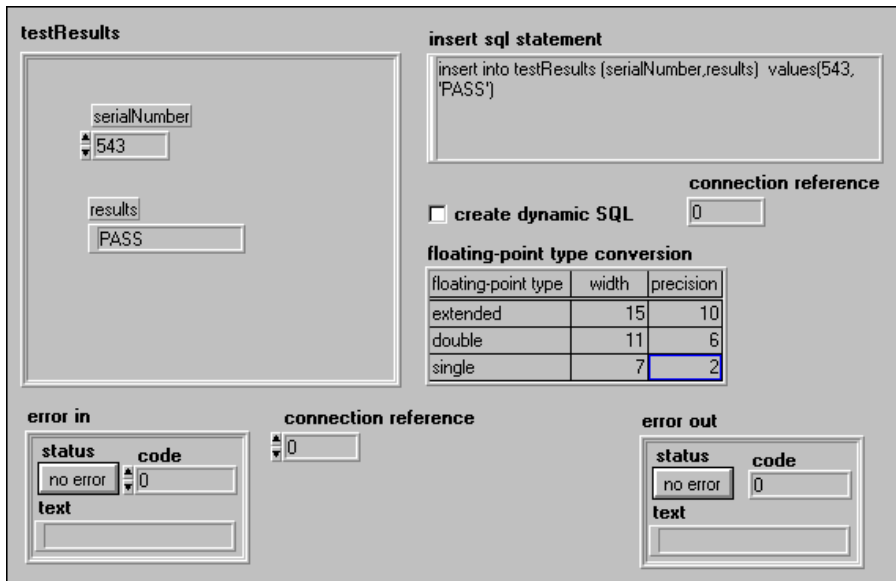


Figure 2-2. Modified Cluster to Insert SQL Template

Notice that the cluster name is used for the name of the table, and the control names within the cluster are used for the field names. It is important for the names to be consistent with the underlying structure of the database with which you want to communicate.

Using the Cluster to Select SQL Template VI

The Cluster to SQL Select Template VI dynamically creates an SQL `SELECT` statement based on a cluster definition. The template has a cluster on its front panel that you modify to reflect the fields of the database that you want to retrieve. As mentioned earlier, make sure to save a copy of the VI in your own hierarchy rather than modifying the original VI. Figure 2-3 shows the front panel of the VI modified to create a `SELECT` statement. You could use this to retrieve the numeric field 'serialNumber' and the string field 'results' from a table

'testResults'. The VI has run, so the string at the right of the figure shows the resulting SQL statement.

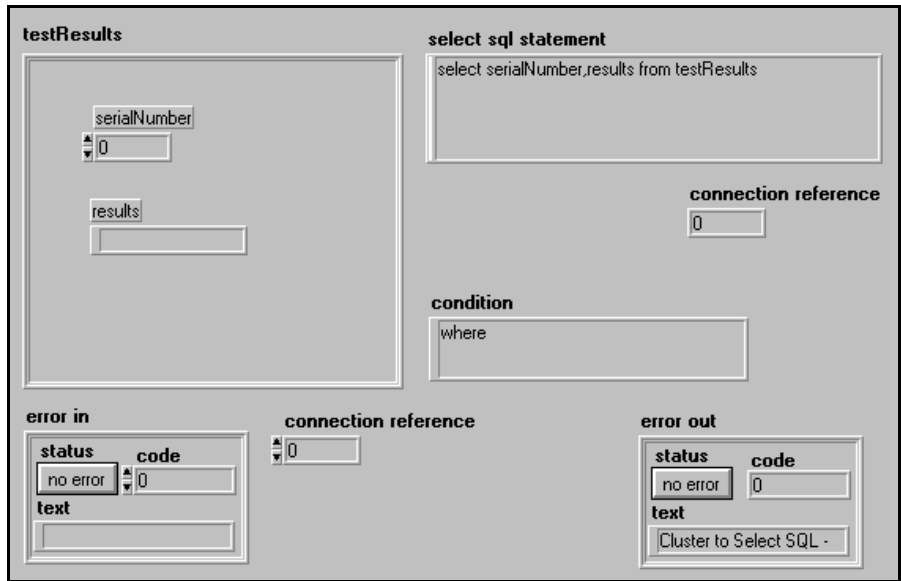


Figure 2-3. Modified Cluster to Select SQL Template VI

Notice that the cluster name is used for the name of the table, and the control names within the cluster are used for the field names. It is important to make the names consistent with the underlying structure of the database with which you want to communicate.

With a `SELECT`, you can specify just the fields that you want to retrieve. Also, this VI allows you to specify a `WHERE` clause, so that only records that match the conditions specified by the `WHERE` would be returned. If you do not specify a `WHERE` clause in an SQL `SELECT` statement, all records are returned.

Using the Results to Cluster Template VI

The Results to Cluster Template VI converts a two-dimensional array into an array of clusters, where each cluster corresponds to a row in the original array. This is convenient for converting the results of an SQL `SELECT` from a two-dimensional array of strings into more convenient G data types.

The template has a cluster on its front panel that you modify to reflect the fields in the two-dimensional array's columns. As mentioned earlier, make sure to save a copy of the VI in your own hierarchy rather than modifying the original VI. Figure 2-4 shows the front panel of the VI modified to parse a two-dimensional array of strings into a cluster containing a numeric field 'serialNumber' and a string field 'results'. The VI has run, so the array at the right shows part of the parsed results.

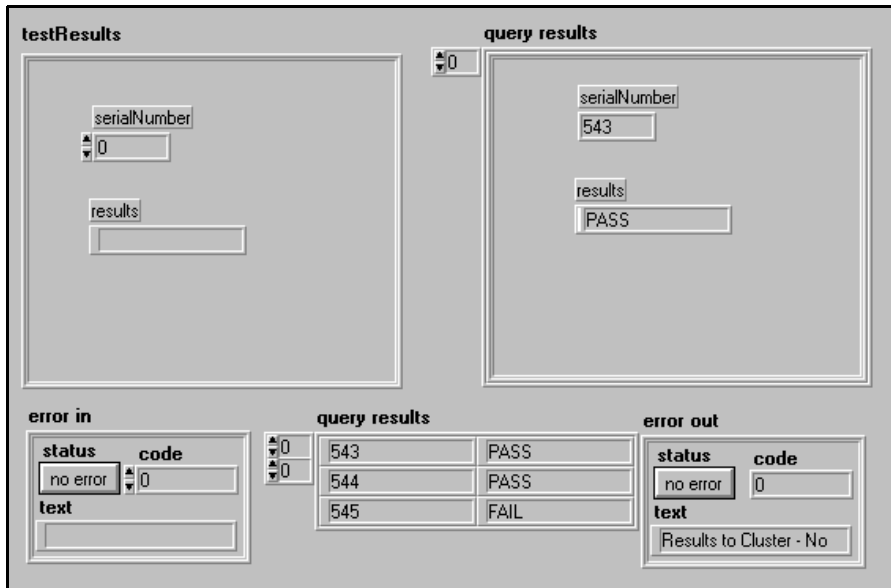


Figure 2-4. Modified Results to Template Cluster VI

Using the Database Browser for Testing

The SQL Toolkit includes a database browser dialog that makes it easy to interactively examine and interact with databases. You access the dialog by selecting the **Project»Database Browser** menu option. You can use this dialog to perform common actions including:

- Connect to databases
- View the structure of databases
- Interactively build SQL `SELECT` statements

- Interactively execute arbitrary SQL statements
- Disconnect from databases; in addition to using this to close connections that you opened from within the database, you can use it to check that your program is closing all connections and SQL statements correctly

See Chapter 9, *Application Examples*, later in this manual for more details on using this dialog.

Optimizing Your SQL Sessions

The Complete SQL Session VI makes it easy to communicate with databases, but it is not the best way if you are concerned about performance. By using lower level SQL VIs, you can eliminate unnecessary steps in SQL operations.

Understanding the Components of an SQL Session

All operations on a database, whether local or remote, occur within the context of a session, as shown in Figure 2-5. The SQL Toolkit maintains all session information in the logical data structures as shown in Figure 2-5 and described in the following sections.



Note:

The session depicted in Figure 2-5 is a sequence structure and included only to emphasize the required operation precedence.

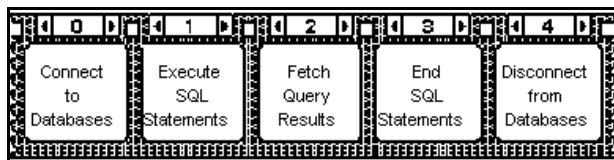


Figure 2-5. The Database Session Sequence

The Complete SQL Session VI performs all of these operations in a single VI. It connects to the selected database, executes the SQL statement, fetches the resulting rows and columns (if any), ends the SQL statement, and disconnects from the database.

Maintain a Connection to a Database

In most applications, you will probably need to execute several SQL statements against a single database. For example, you might `INSERT` several rows or execute several `SELECT` statements. The Complete SQL Session VI handles this inefficiently since it establishes a new connection for each statement. The simplest optimization you can make in the model described in the previous section is to establish the connection yourself, perform as many SQL statements as you need to, and then close the connection when you are finished.

Figure 2-6 displays the diagram of the Complete SQL Session VI.

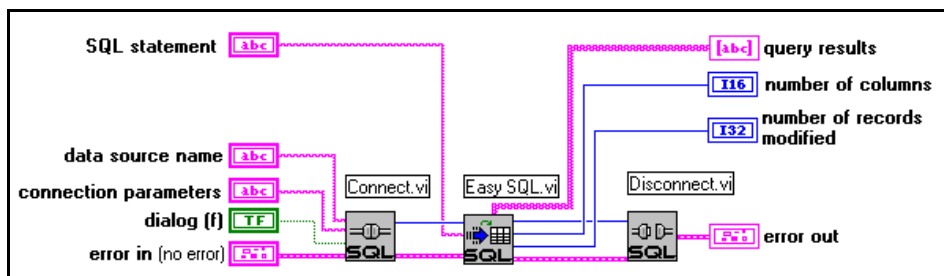


Figure 2-6. Complete SQL Session VI Diagram

This diagram uses the Connect VI to establish a connection, then the Easy SQL VI to execute a statement and retrieve the results, and finally the Disconnect VI to close the connection. Once you establish a connection, you can perform as many SQL statements on that connection as you want. The Connect VI returns a Connection Reference that uniquely identifies that connection. You must wire this reference to all VIs that you want to operate on that data source. The Easy SQL VI makes it simple to execute a statement on a specified connection.

The SQL Toolkit also supports multiple connections to the same database or different databases concurrently, although some database packages might support a single connection at a time only. See the online help for the database with which you intend to communicate to determine if it supports multiple connections.

Eliminate Unnecessary Data Retrieval

Both the Complete SQL Session and the Easy SQL VIs combine the execution of a statement and the retrieval of the results into a single VI. Many SQL statements, such as the `INSERT` statement, do not return any results. By using lower level SQL VIs to execute your statement, you can skip the retrieval process.

Figure 2-7 shows the diagram of the Easy SQL VI.

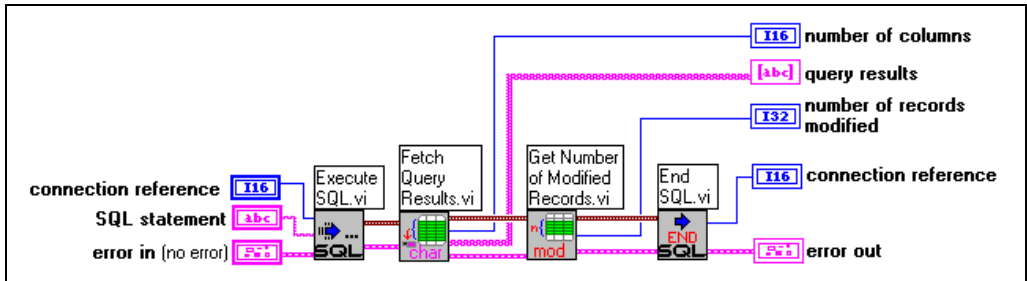


Figure 2-7. Easy SQL VI Diagram

This diagram consists of four steps.

First, it executes the specified statement using the Execute SQL VI. The Execute SQL VI creates a reference number that uniquely identifies the statement that it sets in progress. The VI returns a cluster that contains both the connection reference and the statement reference. You can wire this cluster to all subsequent VIs that operate on the results of an executed statement.

Next, in this diagram, the reference is passed to the Fetch Query Results VI, which returns any resulting data as a two-dimensional array.

Then, since Easy SQL is a general purpose VI, it also calls Get Number of Modified Records, which returns the number of records in the database that were changed as a result of this SQL statement.

Finally, when you finish working on the results of an executed statement, you need to call the End SQL VI, which then returns the original connection reference so that you can easily connect it to other VIs such as the Disconnect VI.

In the case of an INSERT SQL statement, you do not need to call either Fetch Query Results or Get Number of Modified Records. The following diagram shows a simple example of how you might insert multiple records into a database using these lower level SQL VIs.

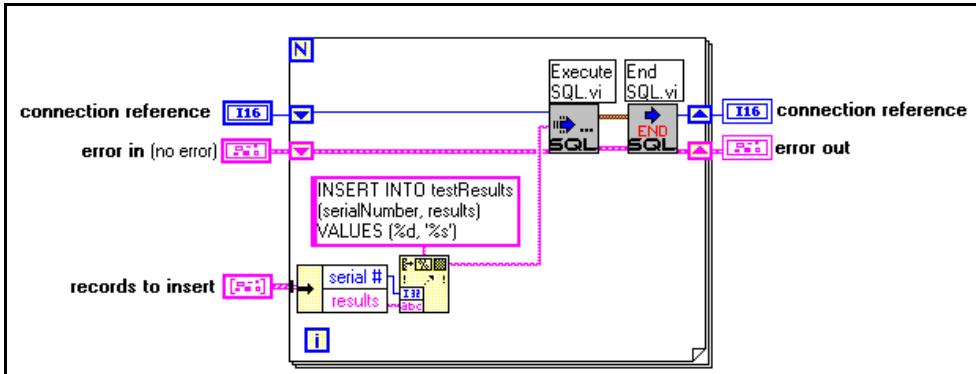


Figure 2-8. Inserting Multiple Records

Speed Up Execution Using Dynamic SQL

When executing an SQL statement, a large part of the time of execution comes from the parsing of the instruction and the arguments. When inserting multiple records into a database, you can spend a lot of time parsing very similar SQL INSERT statements. Dynamic SQL is an alternative to standard SQL that allows you to eliminate redundant parsing, and is appropriate when you execute multiple SQL statements with only the data in the statement varying.

In the previous example, the following SQL INSERT statement was passed to the Execute SQL VI:

```
INSERT INTO testResults (serialNumber, results) VALUES
(6423, 'Pass')
```

To use dynamic SQL, you would pass the following SQL INSERT statement to the Prepare SQL VI (**Functions»SQL»Advanced SQL»Dynamic SQL**):

```
INSERT INTO testResults (serialNumber, results) VALUES
(?, ?)
```

Then, when you are ready to actually insert a record, you fill in the parameters marked with a question mark by calling one of the Set SQL

xxx Parameter VIs, where xxx is the datatype of the value you want to use (for example, you use Set SQL Long Parameter to replace a question mark with a long integer value).

After you fill in the parameters, you call the Execute Prepared SQL VI to execute the statement. If you want to do another INSERT with different values, you can clear each of the parameters using Clear SQL Parameter, fill in the values using Set SQL xxx Parameter, and then execute the statement using Execute Prepared SQL. Clear SQL Parameter is not necessary, but some databases might require you to clear parameter values before updating them with new values.

After you execute a statement using Execute Prepared SQL, you can retrieve any results just as you would with a standard SQL statement, using Fetch Query Results, or one of lower level retrieval VIs that are described in the *Retrieve Data Directly into G Data Types* section. When you are finished using the dynamic SQL statement, call End SQL to clear the SQL statement reference.

Notice that you can use Dynamic SQL statements in other statement types, such as in a SELECT WHERE clause, in which conditional data constantly changes. Dynamic SQL is best for any statement you must execute more than once with only the data varying.

Figure 2-9 shows how you could improve the performance of the previous example using dynamic SQL:

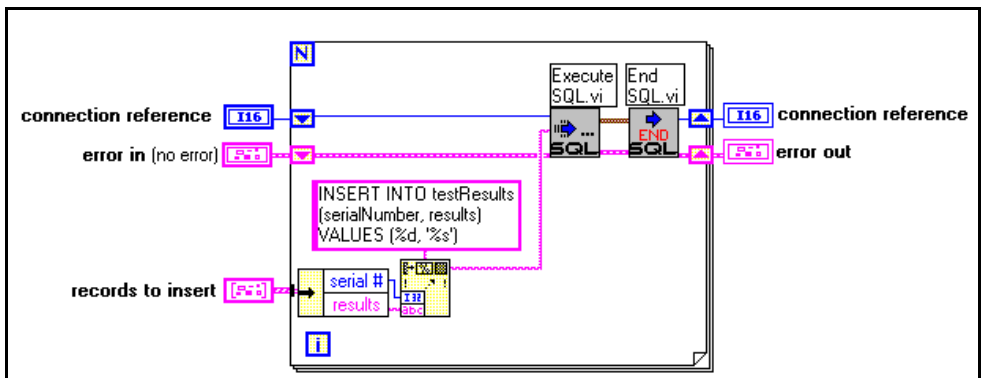


Figure 2-9. Improving Performance with Dynamic SQL

Retrieve Data Directly into G Data Types

The Fetch Query Results VI that is part of the Complete SQL Session and Easy SQL VIs returns data as a two-dimensional array of strings. This method is the most versatile because it retrieves results no matter how many columns or records need to be retrieved, and regardless of the data types. For example, by issuing a `SELECT * FROM tableName`, you can retrieve all records in a database, regardless of the database format.

While Fetch Query Results is easy to use and versatile, it also must convert all the data to a two dimensional array of strings. For large amounts of data, this can be inefficient in terms of memory and performance. Also, in many cases you want the data in standard G data types, so you have to do an additional conversion after the query to convert the results. The Results to Cluster Template VI described earlier in this chapter makes this conversion easy, but it is still inefficient.

The SQL Toolkit has lower level data retrieval VIs that you can use to retrieve data directly into G data types. If you want to use this method, the easiest way to do it is to start with the Fetch Query Results VI, save it with a new name, and modify it. The main portion of the Fetch Query Results are shown in Figure 2-10 below.

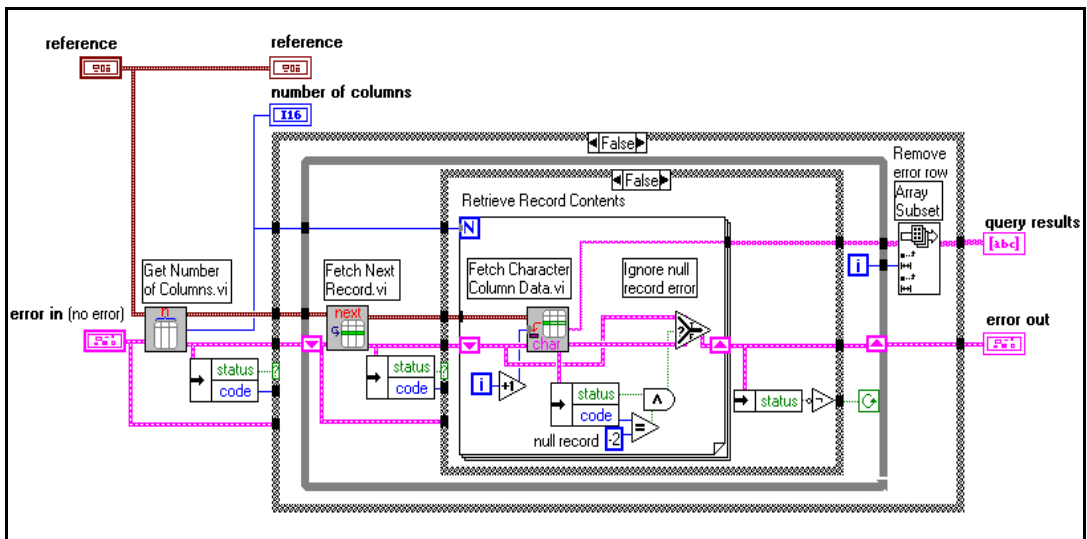


Figure 2-10. Main Portion of Fetch Query Results

This diagram first determines the number of columns that the resulting data contains. It then calls Get Next Record VI to prepare for a record's data, and then loops through the columns retrieving the data using Fetch Column Data. Fetch Column Data retrieves a record regardless of the data type.

To retrieve the results directly into the standard data types, make a copy of the Fetch Query Results VI. Then replace the calls to Fetch Column Data with calls to Fetch *xxx* Column Data, where *xxx* is the datatype of the column that you are retrieving (for example, Fetch Long Column Data to retrieve a long integer value). To use this method, you must know the number of columns and the data types of the columns being returned by the SELECT statement. Figure 2-11 shows how you might modify this VI to retrieve the results of a query that returns records consisting of a long integer and a string.

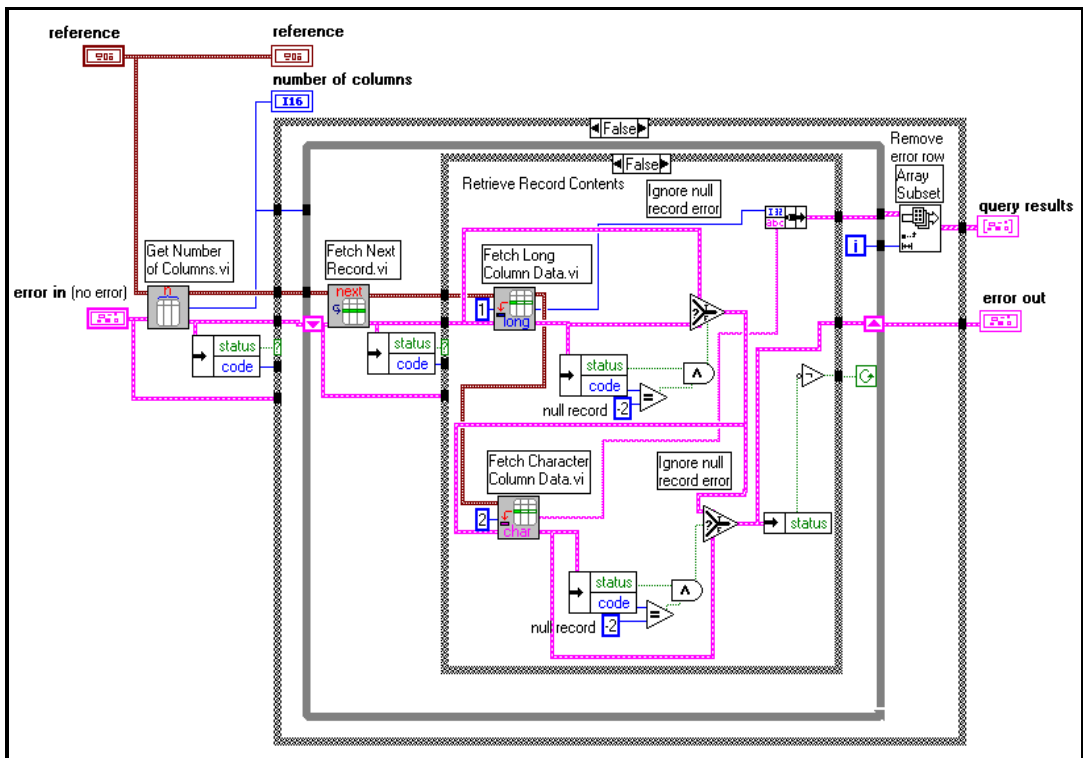


Figure 2-11. Modified Fetch Query Results

Use Efficient SQL Operations and Databases

The optimizations described so far have concentrated on eliminating unnecessary steps by using lower level SQL VIs. Another approach is to eliminate some of this work by using more efficient SQL statements.

For example, it is inefficient to use a simple SQL `SELECT` statement to retrieve all the data in a database if you can use a `WHERE` clause to retrieve only the data you want. The

Examples\SQL\general.llb\SQL Optimization Demo shows how this technique can help you produce more efficient VIs. However, to use this technique effectively you need a good understanding of SQL. The [Related Documentation](#) section in the [About this Manual](#) chapter of this manual lists several references that are useful if you are interested in learning more about SQL.

Another technique is to use *indexing*, a feature supported by many databases. Indexing allows you to mark specific columns so you can search them more efficiently. See the documentation for the database you are using to determine whether you can use indexing in your application.

Other SQL Operations

This section describes some of the other SQL operations you can perform.

Transaction Management

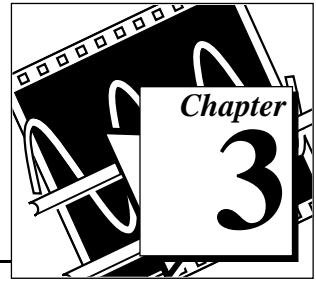
A *database transaction* is a related sequence of data manipulation language (DML) SQL statements that you must treat as an indivisible unit to maintain data consistency. With Transaction Operation VIs, G applications can apply groups of row changes to the database collectively. This is useful if there is a possibility of interrupting a session when related data in multiple tables are in an inconsistent state. Each database supports transactions in a slightly different way.

A transaction consists of all executed DML statements since the last transaction started. A transaction begins immediately upon executing the Start Transaction VI. Executing either the Commit Transaction VI or the Rollback Transaction VI completes the transactions. When you execute Commit Transaction, all DML operations in the current transaction are made permanent to the database; while Rollback Transaction discards all DML operations in the current transaction.

Database and SQL Session Information

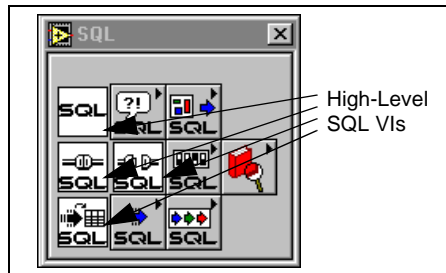
The SQL Toolkit contains a number of VIs that you can use to determine the structure of databases and of the results of a query. These VIs can be useful if you want to build VIs that can manipulate unfamiliar databases. For example, if you wanted to build a general tool for importing data from a database, you could use these VIs to determine the names of tables in databases and the structure of those tables. These VIs are used by the Database Browser dialog to allow you to browse arbitrary databases.

These VIs are divided into two sets. The first, accessible from the **SQL Information** palette (select **Functions»SQL»SQL Information**) make it easy to determine information about the databases associated with a specified connection. The second, available on the **Column Information** palette (select **Functions»SQL»Advanced SQL»Column Information**) make it easy to determine information about the columns that were returned as the result of an SQL statement.



High-Level SQL VIs

This chapter provides a detailed description of the High-Level SQL VIs, which perform high-level database operations. You access these VIs by selecting **Functions»SQL**. The **SQL** palette appears.

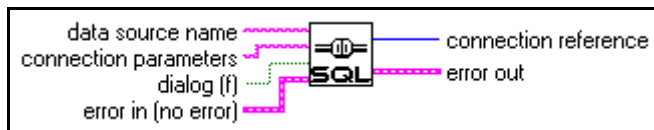


High-Level SQL VI Descriptions

The following High-Level SQL VIs are available. You can use some of the VIs individually, but you must use others in combination for database activity.

Connect

Establishes an active connection to the selected database.



[abc]

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

[abc]

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 from a list of all currently defined data sources.



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

This VI 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 Database Driver.

Use the **connection parameters** control 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 the *SQL Toolkit Driver Online Help* for details on your selected database). For example, the SQL Toolkit examples data source, DBV DEMO DBASE, has the dBASE-specific parameter Database Directory defined as

C:\LABVIEW\EXAMPLES\SQL_EXMPLDB, which is 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.

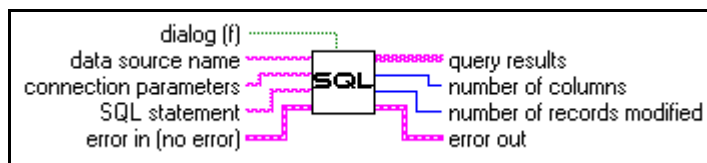
You use the **dialog** control to select a data source and other (optional) connection parameters interactively. When you do not specify a Data Source Name and Dialog is set to True, a data source selection dialog box appears. Click on and highlight the desired Data Source Name and then click **OK**.

With some databases (such as Access), you can associate multiple connection/logon options with a single named data source. When applicable, a database-specific selection dialog window subsequently appears. In this case, you type your **User Name** and **Password** within the dialog window and select a specific Access server.

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

Complete SQL Session

Performs a complete database session.



[abc]

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

[abc]

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

[abc]

SQL statement specifies the desired SQL operation.

TF

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

[abc]

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

I16

number of columns returns the number of columns the referenced SQL SELECT operation returned.

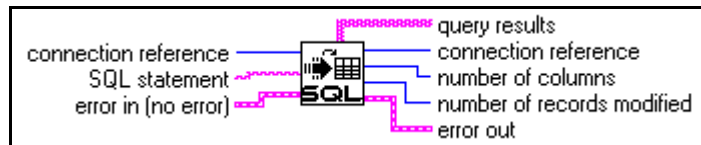
I32

number of records modified returns the number of records, if any, the referenced SQL operation modified.

This VI performs all steps of a database session. It establishes an active connection to the specified database, begins a transaction, executes the SQL statement, retrieves data generated by the `SELECT` command (if any), concludes the transaction, and then disconnects from the database. The transaction rolls back if it encounters an error on SQL statement execution, and commits if rows are modified and it encounters no SQL error. The selected database must support the SQL statement. (See the *SQL Toolkit Driver Online Help* for more information.)

Easy SQL

Performs an SQL operation on a connected database and returns any `SELECT` query results.

**I16**

connection reference specifies an existing database connection number that performs the SQL operation.

abc

SQL statement specifies the desired SQL operation.

I16

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

I16

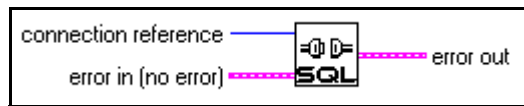
number of columns returns the number of columns that were returned by the referenced SQL SELECT operation.

I32

number of records modified returns the number of records, if any, that by the referenced SQL operation modified.

Disconnect

This performs all steps of a database session except connecting and disconnecting. It requires that you establish an active connection with a database, either local or remote, prior to execution. Easy SQL executes the specified SQL statement to be executed on the referenced database connection. The selected database must support the SQL statement (see *SQL Toolkit Driver Help*). If an SQL SELECT statement is executed, the VI attempts to return all resulting data (subject to available memory) into a Disconnect and terminates the referenced database connection.

**I16**

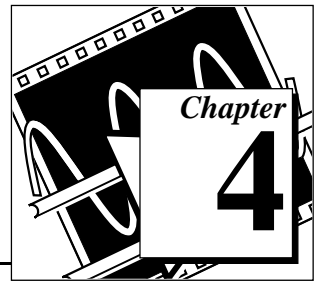
connection reference specifies an active database connection number to disconnect. Set to -1 to disconnect all currently active connections.

TF

disconnect always(F) when set to True disconnects even when an error is passed into the VI.

This VI deactivates 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, so a single call to Disconnect deactivates all active database connections. Executing Disconnect with the Connection Reference control set to -1 invokes this function. You may find this feature particularly useful during application development to clean up all connections in case of a failure.

Error Handling VIs and Cluster-to-SQL Template VIs



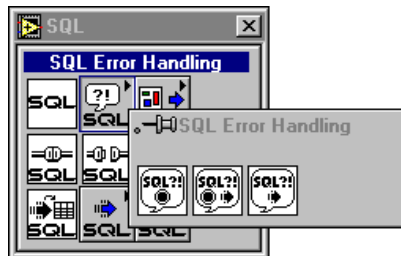
This chapter provides detailed descriptions of the Error Handling VIs and the Cluster-to-SQL templates, which perform high-level database operations.

Error Handling VIs Overview

You can execute Error Handling VIs any time that a session is active. They can inform you when an exception condition occurs, and an interactive response, if appropriate, can run. You must exercise caution when using these VIs to abort execution, especially when multiple databases connections are active.

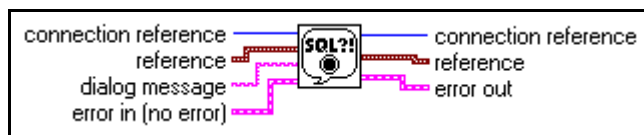
Error Handling VI Descriptions

You can access these VIs by selecting **Functions»SQL»SQL Error Handling**. Doing so accesses the **SQL Error Handling** palette.



Abort Error Dialog

Presents an abort dialog box that describes a database error and then terminates the database connection and VI execution.





connection reference specifies an active database connection number.



dialog message inputs a string to display in the abort dialog box.

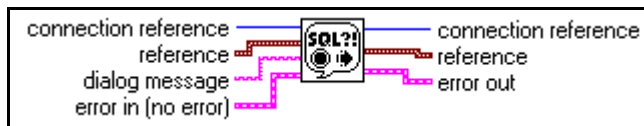
You use the **Abort Error** dialog box to interpret and identify an error returned from a database operation. When called, the VI checks for an error detection using **error in**. If an error occurs, it displays a dialog window containing the error text (unbundled from the **error in** cluster), the **dialog message**, and a single button labeled **Abort**. After you click on the **Abort** button, the VI disconnects from the database specified in **connection reference**, terminates the database session, and then terminates execution. Set connection reference to -1 to disconnect and terminate all active database sessions.

If no error is detected at **error in**, VI execution continues normally.

The **dialog message** control can indicate the location of an error within the application VI, e.g. "...while connecting", "...while executing SQL", "...while fetching data", etc.

Abort-Continue Error Dialog

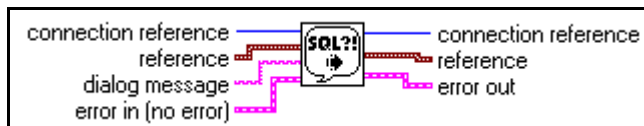
Presents a selection dialog box that describes a database error. You can continue or terminate the database connection and VI execution.



Use the **Abort-Continue Error** dialog box to interpret and identify an error that 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 box with the error text (unbundled from the **error in** cluster), the **dialog message**, and two buttons: **Abort** and **Continue**. If you click on **Abort**, the VI disconnects from the database specified in connection reference, terminates the database session, and terminates execution. If you click on **Continue**, VI execution continues normally.

Continue Error Dialog

Presents a dialog box that describes a database error and then continues VI execution.



This VI interprets and identifies an error that 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** button. When you click on **Continue**, VI execution continues.

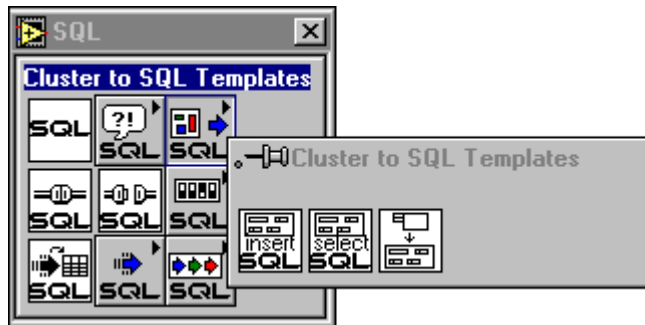
SQL-to-Cluster Template VIs Overview

Cluster VIs generate SQL statements dynamically from G clusters and convert results into G clusters. By supplying an input cluster, where the cluster name is the table name and the control names are column names, the VIs generate `INSERT` or `SELECT` SQL statements.

The data types of the controls must match as closely as possible the data types contained in the database for the corresponding columns. You can use the following control types: DBL, SGL, I32, I16, I8, U32, U16, U8, and string. You can use DBL, SGL, I32, I16, U32, and U16 for any numeric data type. You can use I8 and U8 for Boolean data types. Strings are used for character types, binary types, and date data types. If a character string conforms to the 26-byte ODBC date-time format, the conversion is read as a date type. Otherwise, it assumes a character type. The control order is the order that the data appears in the SQL statement. Because the three VIs are intended as templates, it is a good idea to copy and modify these VIs for your application.

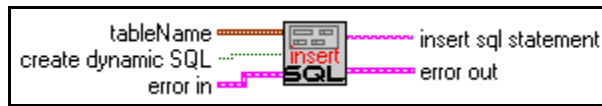
SQL-to-Cluster Template VI Descriptions

You access these VIs by selecting **Functions»SQL»SQL to Cluster Templates**.



Cluster to INSERT SQL

Dynamically creates an `INSERT` SQL statement specified by the **tableName** cluster input.



First, copy this VI, then add the desired controls to the cluster. You must name each control to correspond with a column name in the database. Rename the cluster name to the database table name. Lastly, copy this cluster throughout your application where needed.



tableName specifies a table definition. Add named and typed controls to the cluster. Each control represents a column to insert data into. Change **tableName** to the actual database table name.



create dynamic SQL specifies which type of SQL statement to create. If set to `True`, this control creates a dynamic SQL statement with statement parameters. If set to `False`, this control creates an ANSI Standard SQL statement.



insert sql statement is the `SQL INSERT` statement that is returned.

Cluster to SELECT SQL

Dynamically creates a `SELECT` SQL statement specified by the **tableName** cluster input.



First, copy the VI. Then add the controls you want to the cluster. You must give each control a name that corresponds to the column name in the database. Rename the cluster to the database **tableName**. Finally, copy this cluster throughout your application where needed.



tableName specifies a table definition. Add named and typed controls to the cluster. Each control represents a column to query from the database. Change **tableName** to the actual database table name.



condition specifies any conditionals to be applied to the query.

For example:

where (a=1)

where ((a>0) and (a<10))

where (b='text')



select sql statement is the SELECT SQL statement that is returned.

Results to Cluster

Takes the query results, a 2D array of strings with a table definition cluster, and converts it into an array of clusters of the proper database datatype.

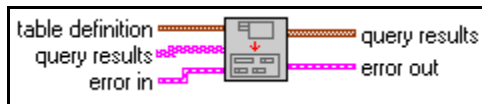


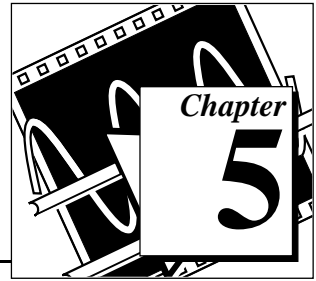
table definition specifies a table definition. Add properly typed controls to the cluster. Each control represents one column from the **query results** control, and the cluster order specifies the data retrieval and conversion.



query results specifies the query results in a 2D array of strings.

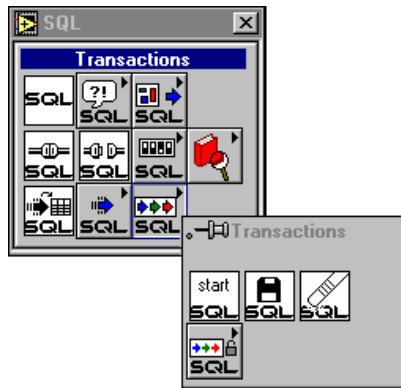


query results are the output of **query results** in an array of clusters. The cluster type must match the **table definition** cluster exactly for the results to be placed in the cluster.



Transaction Operation VIs

All Transaction Operation VIs can be executed at any time during an active session. Each database system conducts transaction operations differently. Refer to the *SQL Toolkit Driver Help* for details on each database system. To access the **Transactions** palette, select **Functions»SQL»Transactions**.



The **Transactions** palette includes the **Transaction Locking** subpalette.

Transaction VI Descriptions

The following Transaction VIs are available.

Commit Transaction

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



I16

connection reference specifies an active database connection number.

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 (see *Start Transaction*, later in this section) was executed are made permanent to the database contents. Transactions are supported only by certain database systems.

Rollback Transaction

Causes the database to discard all changes permanently that have been made during the current transaction.



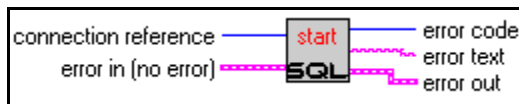
116

connection reference specifies an active database connection number.

Use Rollback Transaction 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 (see *Start Transaction*, later in this section) are discarded. Database contents revert to their state immediately before the transaction began and all intervening changes are ignored. Not all database systems support transactions.

Start Transaction

Begins a database transaction series on the specified database connection.



116

connection reference specifies an active database connection number.

Use this VI to start a transaction series. When a transaction is started, changes to database contents by subsequent SQL operations such as INSERT, UPDATE, and DELETE are recorded to a temporary buffer. The changes can be made permanent to the database by executing Commit Transaction (see *Commit Transaction* earlier in this section). Alternatively, you can discard the changes by executing RollBack Transaction (see *Rollback Transaction* earlier in this section). Transactions are supported only by certain database systems.

Transaction Locking VI Descriptions

Although these VIs are not Transaction VIs themselves, they greatly affect the behavior of transactions.

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

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

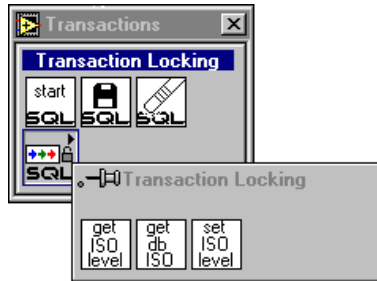
- *Dirty Reads*—User 1 modifies data while user 2 uses that same data before user 1 can commit the changes. User 2, therefore, uses incorrect data.
- *Non-Repeatable Reads*—User 1 reads records while another user 2 modifies records. User 1 rereads the records and finds that a record has changed or been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

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

- *Read Uncommitted* (lowest level)—Locks are obtained on modifications only and held to the EOT. Reading does not involve any locking.
- *Read Committed*—Locks are obtained on reading and modification, but locks are released after reading and held until EOT for modifications.
- *Repeatable Read*—Locks are obtained on reading and modifications. Locks are held until EOT for both reading and modifying records. Locks on non-modified access are released after reading.
- *Serialized*—All read or modified data is locked until EOT.
- *Versioning* (highest level)—A different implementation of Serializable that provides greater concurrency through the use of non-locking record versioning protocols.

The higher the isolation level, the greater the locking strategy, but the less user concurrency.

These VIs obtain information about the available isolation levels supported by a database, the current isolation level in use, and the procedure for setting a new isolation level. You access the **Transaction Locking** palette by selecting **Functions»SQL»Transactions»Transaction Locking**.



The following Transaction Locking VIs are available.

Get ISO Level

Determines the isolation level for the connection.



connection reference specifies an active database connection number.

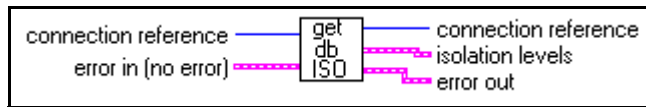


isolation level specifies the locking-strategy used.

- 0 – Versioning
- 1 – Serializable
- 2 – Repeatable Read
- 3 – Read Committed

Get Supported ISO Levels

Returns the isolation levels supported by the database system.



T16

connection reference specifies an active database connection number.

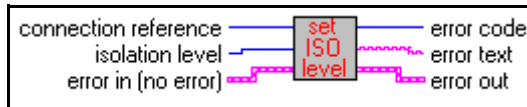
T16

isolation level specifies the locking-strategy used.

- 0 – Versioning
- 1 – Serializable
- 2 – Repeatable Read
- 3 – Read Committed

Set ISO Level

Sets the isolation level for the specified connection.



T16

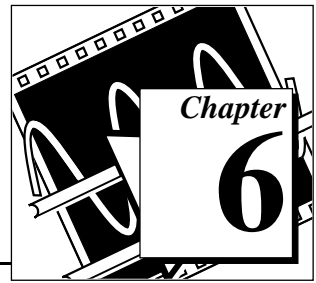
connection reference specifies an active database connection number.

T16

isolation level specifies the locking-strategy used.

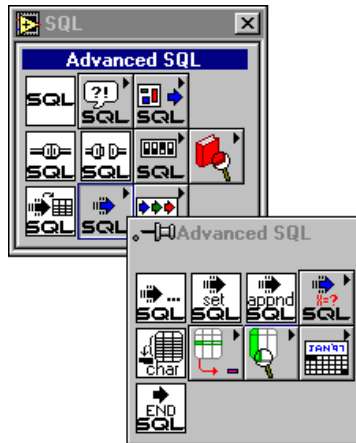
- 0 – Versioning
- 1 – Serializable
- 2 – Repeatable Read
- 3 – Read Committed

Advanced SQL VIs



This chapter introduces the Advanced SQL VIs, which provide detailed low-level access to database services and which, in many cases, form the elemental building blocks of the other SQL Toolkit VIs.

You access the **Advanced SQL** palette by selecting **Functions»SQL»Advanced SQL**.

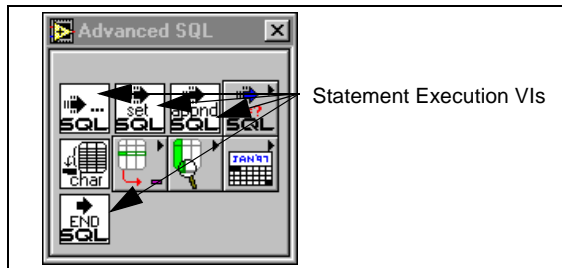


The **Advanced SQL** palette includes the Statement Execution VIs and the following subpalettes.

- Dynamic SQL
- Fetch Data Parsing
- Column Information
- Date Conversion

Statement Execution VI Descriptions

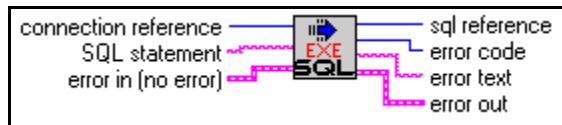
The Statement Execution VIs are available on the **Advanced SQL** palette.



The following Statement Execution VIs are available.

Execute SQL

Executes an 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.

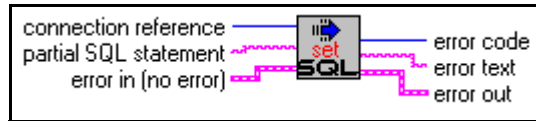
I16

sql reference returns a unique identifier for the specified SQL statement that must be used in subsequent operations.

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 an SQL `SELECT` statement must be fetched using, for example, the Fetched Query Results VI (see [Fetch Query Results](#) in the [Fetch Data Parsing VI Descriptions](#) section later in this chapter for more information).

Set SQL (Windows 3.1) and Append SQL (Windows 3.1)

Handle SQL statements larger than 20,000 bytes.



116

connection reference specifies an active database connection number.

abc

partial SQL statement is the first part of the large SQL VI for the set SPL VI and is the remaining part(s) of the SQL statement for the Append SQL VI.

In some applications, an insert statement containing binary data might be very large and cannot be executed with the Execute SQL VI alone. In such cases, You first must call the Set SQL VI with a partial SQL statement as input that is not larger than 20,000 bytes. To assign the remainder of the SQL statement, calls are made to the Append SQL VI, with a partial SQL statement for input. When enough of the Append SQL VI calls are made to complete the desired SQL statement, a call is then made to the Execute SQL VI with a null SQL statement as input.

End SQL

Terminates the specified SQL reference execution.



116

sql reference specifies a previously executed SQL operation number. Set to -1 to terminate all currently active SQL operations.

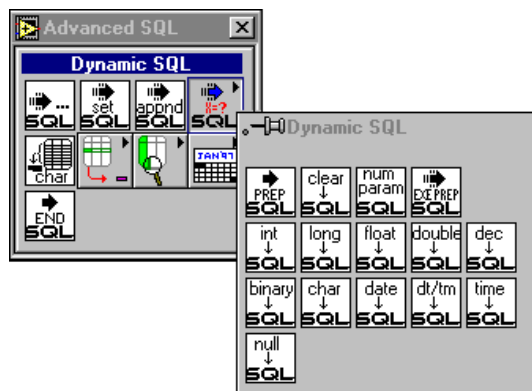
This VI deactivates the SQL reference and cleans up after an SQL operation finishes, including all row and column fetches. It also releases all local SQL Toolkit and remote database system resources, and disposes of query results, if any exist. Execute End SQL when random fetch options are enabled, so it closes properly and deletes any temporary log files.

Dynamic SQL VI Descriptions

Dynamic SQL is most beneficial when a statement needs to be executed more than once with only the data varying in the statement. An **Insert SQL statement**, where more than one record is to be inserted, is a good example of when it is best to use dynamic SQL. Depending on the data and the database structure you use, inserts using dynamic SQL can be several times faster than executing a new SQL statement each iteration of an acquisition loop. (See *Optimizing Your SQL Sessions* in Chapter 2, *Using SQL Toolkit VIs*, for more information.)

Generally, the method is as follows: You call Prepare SQL prior to the acquisition loop with a dynamic SQL statement for input. The dynamic SQL statement is the same as ANSI SQL except there are *placeholder* characters to indicate where values are assigned, for example: INSERT into table (col1, col2) values (?,?). As data is acquired, you can set the parameters by using the Set SQL Parameter calls. When you have defined all parameters, a call is made to Execute Prepared SQL. After completion of the acquisition, a call to End SQL is made to clear the SQL references. The same SQL reference is used throughout the acquisition loop, which makes the application run much faster.

You access the Dynamic SQL VIs by selecting **Functions»SQL»Advanced SQL»Dynamic SQL**. This displays the **Dynamic SQL** subpalette.



Prepare SQL

Parses an SQL statement containing statement parameters.



I16

connection reference specifies an active database connection number.

abc

sql statement is the desired SQL operation containing statement parameters for values, for example, `INSERT into table (col1, col2) values (?, ?)`.

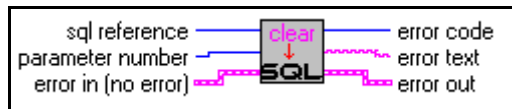
I16

sql reference is a unique identifier for the specified SQL statement that must be used as input to `Execute Prepared SQL` and other subsequent operations.

You prepare SQL to parse and prepare SQL statements containing statement parameters for values; i.e., `INSERT into table (col1, col2) values (?, ?)`. `Set SQL Parameter` calls must be made to set the parameter values before the statement can be executed. When all parameters are set, a call to `Execute Prepared SQL` executes the SQL statement against the database.

Clear SQL Parameter

Clears the value of the specified parameter in an SQL statement.



I16

sql reference specifies a previously executed SQL operation number returned from `Prepare SQL`.

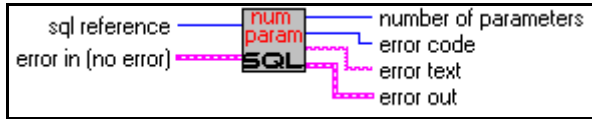
I16

parameter number specifies the parameter number as described in the SQL statement; for example, `INSERT into table (col1, col2) values (?, ?)`. The first “?” contained in the statement is parameter 1; the second “?” is parameter 2.

After you assign a value to an SQL statement parameter, you can clear it by making a call to the `Clear SQL Parameter VI`. You can overwrite parameter values by making another `Set SQL Parameter` call. Under some operating systems, however, you must clear a character string type parameter before you assign a new value to the parameter.

Get Number of Parameters

Returns the number of statement parameters that appeared in the specified SQL statement.



I16

sql reference specifies a previously executed SQL operation number returned from the Prepare SQL VI.

I16

number of parameters is the number of statement parameters that appeared in the specified SQL statement.

Execute Prepared SQL

Executes a previously prepared SQL statement as returned from Prepare SQL.

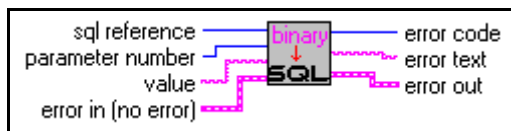


I16

sql reference specifies a previously executed SQL operation number returned from Prepare SQL.

- Set SQL Integer,
- Set SQL Long,
- Set SQL Float,
- Set SQL Double,
- Set SQL Binary,
- Set SQL Character,
- Set SQL Date,
- Set SQL Date-Time, and
- Set SQL Time

Set values for SQL statements that contain statement parameters.





sql reference specifies a previously executed SQL operation number returned from Prepare SQL.



parameter number specifies the parameter number as described in the SQL statement; for example, `INSERT into table (col1, col2) values (?,?)`. The first “?” contained in the statement is parameter 1; the second “?” is parameter 2.



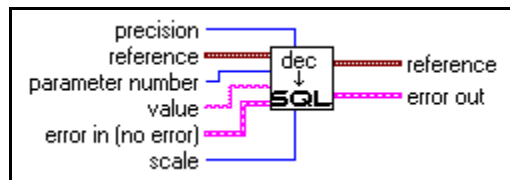
value specifies the value to set for the specified parameter. This input’s type varies depending on what Set SQL Parameter function is being called. For Date-Time functions, this input is the 26-byte date string format. For double and float calls, this input is of type double and single, respectively. For long and integer calls, this type is of I32 and I16, respectively.

Prior to calling the Set SQL Parameter functions, call to Prepare SQL with SQL statement parameters. Making the Set SQL Parameter calls assigns a value to the parameters. After you assign all parameters, Execute Prepared SQL executes the statement against the database.

The dynamic SQL is most beneficial when a statement needs to be executed more than once with only the data varying in the statement. An **Insert SQL statement**, where more than one record is to be inserted, is a good example of when it is best to use dynamic SQL. Depending on the data and the database structure you use, inserts using dynamic SQL can be several times faster than executing a new SQL statement each iteration of an acquisition loop. See *Optimizing Your SQL Sessions* in Chapter 2, *Using SQL Toolkit VIs*, for more information.)

Set SQL Decimal Parameter

Sets the specified SQL parameter to the specified decimal formatted string value.



reference specifies a previously executed SQL operation number returned from Prepare SQL.



parameter number specifies the parameter number as described in the SQL statement; for example, `INSERT into table (col1, col2) values (?,?)`. The first “?” contained in the statement is parameter 1; the second “?” is parameter 2.

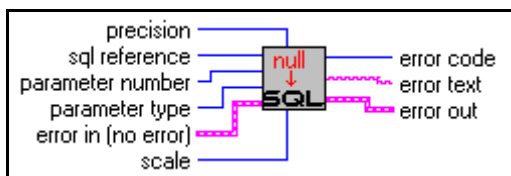
I16

value specifies the value to set for the specified parameter. This input's type varies depending on what Set SQL Parameter function is being called. For Date-Time functions, this input is the 26-byte date string format. For double and float calls, this input is of type double and single, respectively. For long and integer calls, this type is of I32 and I16, respectively.

precision/scale is not used.

Set SQL Parameter to Null

Sets the specified SQL parameter to null.

**I16**

sql reference specifies a previously executed SQL operation number returned from Prepare SQL.

I16

parameter number specifies the parameter number as described in the SQL statement; for example, `INSERT into table (col1, col2) values (?,?)`. The first “?” contained in the statement is parameter 1; the second “?” is parameter 2.

I16

parameter type specifies the parameter's data type.

0 – fixed length string

1 – variable length string

3 – 4-byte signed integer

4 – 2-byte signed integer

5 – 4-byte floating-point number

6 – 8-byte floating-point number

7 – 26-byte date time value: `YYY-MM-DD HH:MM:SS.FFFFFFFF`

8 – 26-byte date value

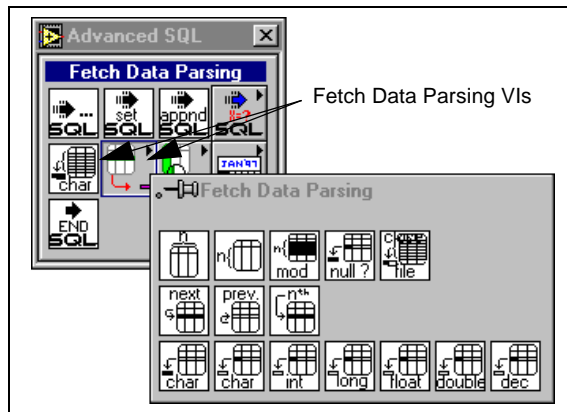
9 – 26-byte time value

precision/scale is not used.

Fetch Data Parsing VI Descriptions

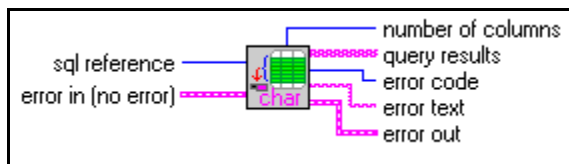
You can call several of these VIs by higher-level fetch VIs. Others are for specific application needs.

You access the Fetch Data Parsing VIs by selecting **Functions»SQL»Advanced SQL»Fetch Data Parsing**. The **Fetch Data Parsing** palette appears.



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 from which to fetch query results.

[I16]

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

[abc]

query results returns the entire results of a SQL `SELECT` statement in a two-dimensional array of strings, given sufficient memory.

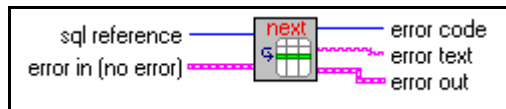
Fetch Query Results attempts to retrieve all data rows resulting from the active set of an SQL `SELECT`, subject to available memory. Prior to executing this VI, an SQL statement

must have been executed successfully. Row data are 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.

Once you access Query Data, you can extract and convert the columns from string to other data types using LabVIEW's built-in string conversion function. You can convert from and to the SQL Toolkit date type using SQL Date to G Date Format VI and G Date to SQL Date Format VI.

Fetch Next Record, Fetch Previous Record, and Fetch Record N

Position the cursor to the current record from the results of an SQL `SELECT` operation.



sql reference specifies a previously executed SQL `SELECT` operation number.



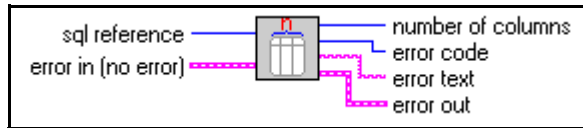
fetch record number specifies the number of the record to fetch from the SQL `SELECT` results (Fetch Record N only).

The Fetch Next Record VI 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 an SQL `SELECT` reference, this VI returns the first record from the database. Use Fetch Column Data functions to retrieve the columns from the current record into LabVIEW. Fetch Previous Record functions the same way, except it moves from record to record in the reverse order. Fetch Record N sets the current record to the specified record.

With Set Fetch Options, you must use enable the Fetch Previous Record and Fetch Record N functions.

Get Number of Columns and Get Number of Records

Determine the number of columns and the number of records returned by a referenced SQL `SELECT` operation.



I16

sql reference specifies a previously executed SQL `SELECT` operation number.

I16

number of columns/number of records is the number of columns or the number of records, depending on the function called, returned by the referenced SQL `SELECT` operation.

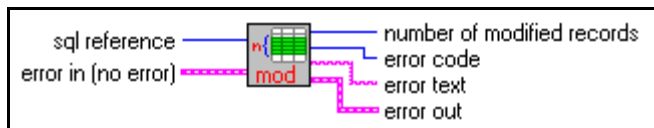
You can use Get Number of Records only if you enabled random fetching by Set Fetch Options.



Note: *When this VI is executed, SQL Toolkit actually retrieves all selected records from the database system into temporary log files. Ensure that sufficient files and disk space are available to accommodate the retrieved data.*

Get Number of Modified Records

Returns the number of database records that have SQL `INSERT`, `UPDATE`, and `DELETE` operations modified.



I16

sql reference specifies a previously executed SQL `SELECT` operation number.

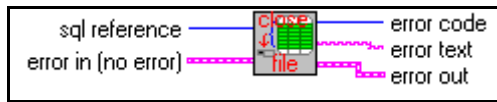
I32

number of modified records returns the number of records that were modified by the referenced SQL operation.

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

Close Fetch Log File

Closes any temporary fetch log files for the specified SQL reference.

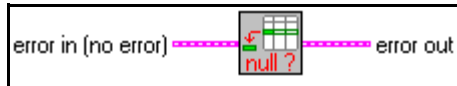


sql reference specifies a previously executed SQL SELECT operation number.

When log files are closed, the contents of the log files are preserved. The files reopen automatically when the next fetch operation is performed on that SQL reference.

Handle Null Value

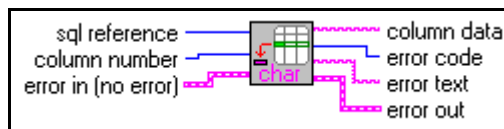
Changes a result code of -2 (null value) to a result code of 0.



This VI executes continuous fetch operations without halting on a -2 result code. The Fetch Query Results VI already includes this functionality. However, if you modify Fetch Query Results VI to retrieve data directly into standard data types, call Handle Null Value after each call so that you can ignore any null value result codes. You can halt fetch operations if a null value is encountered. If so, do not use this function.

Fetch Character Column Data, Fetch Double Column Data, Fetch Float Column Data, Fetch Integer Column Data, Fetch Long Column Data, and Fetch Column Data

Retrieve the contents of a specific column from the current record.





sql reference specifies a previously executed SQL `SELECT` operation number.



column number specifies from which column number to retrieve the contents.



column data is the contents of the column as a character string, integer or floating-point, depending on the Fetch Column Data function called.



Note: *Prior to using any of these VIs execute the `Fetch Next Record`, `Fetch Previous Record` or `Fetch Record N`.*

Retrieve column contents in order, such as column 1, 2, 3. You cannot retrieve the contents of a column twice. You can skip columns, but once you retrieve a subsequent column, you cannot retrieve earlier columns. For example, if you fetch columns 1, 2, and 4 in order, you cannot fetch column 3.

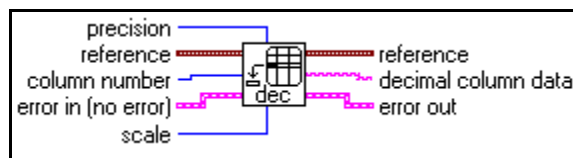
The Fetch Column Data VI is a generic fetch call that retrieves any column's contents regardless of its data type. It converts the column contents to a string before returning. The higher level fetch VI, Fetch Query Results, calls Fetch Column Data because of its dynamic ability. You do not need to know the underlying data types of the results called by `SELECT`.

In some applications, you may want to retrieve the results directly into a standard data type. You use the rest of these functions for this purpose. The disadvantage of this is you must know the data type of the results called by `SELECT`, also you must copy the Fetch Query Results and modify them specifically for your application.

Fetch Double Column Data and Fetch Float Column Data VIs retrieve data and place the data into LabVIEW's double-precision and single-precision data types, respectively. Fetch Long Column Data and Fetch Integer Column Data VIs retrieve data and place the data into LabVIEW's I32 and I16 data types, respectively. Fetch Character Column Data VI is identical to Fetch Column Data VI and is included only for consistency. Use it to retrieve any column's contents as a character string. It is also the only way to retrieve date and binary types, which you cannot retrieve directly.

Fetch Decimal Column Data

Retrieves the contents of a specified formatted decimal string.





reference specifies a previously executed SQL operation number returned from Prepare SQL.



parameter number specifies the parameter number as described in the SQL statement; for example, `INSERT into table (col1, col2) values (?,?)`. The first “?” contained in the statement is parameter 1; the second “?” is parameter 2.



value specifies the value to set for the specified parameter. This input's type varies depending on what Set SQL Parameter function is being called. For Date-Time functions, this input is the 26-byte date string format. For double and float calls, this input is of type double and single, respectively. For long and integer calls, this type is of I32 and I16, respectively.

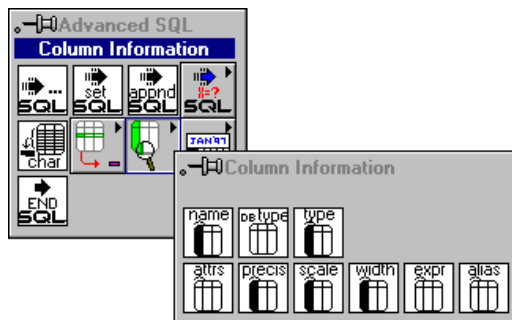
precision is not used.

Column Information VI Descriptions

You use the Column Information VIs to obtain information about data sources, database names, tables, columns, and column types. Each VI returns the results in a 2D array of strings.

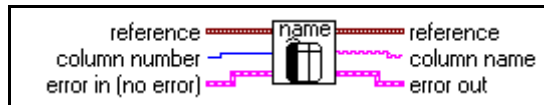
The interface level Column Information VIs consist of two groups of VIs. The first set of VIs obtain information about database columns. The second set of VIs obtain information about data sources, database names, tables, columns, and column types.

To access the Column Information VIs, select **Functions»SQL»Advanced SQL»Column Information**.



Get Column Name, Get Column Alias, Get Column Expression, Get Column Scale, Get Column Width, and Get Column Precision

Return information about the specified column returned by the referenced SQL operation.



reference specifies a previously executed SQL operation number.



column number specifies the column number to retrieve the information for.



column name/alias/expression/scale/width/precision each provide information about the column, depending on the function executed, from the referenced SQL operation.

column name is the actual column's name.

column alias is the column's alias name if it has one.

column expression is the column's expression.

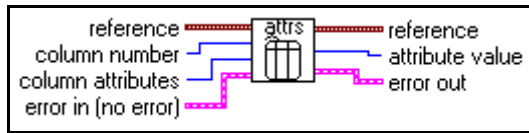
column scale is the maximum number of fractional decimal digits, if applicable.

column precision is the maximum number of decimal digits, if applicable.

column width is the largest data width that a column can contain in the database, not the selected column's width.

Get Column Attributes

Determines whether a specified column has a specified attribute returned by the referenced SQL `SELECT` operation.



I16

sql reference specifies a previously executed SQL operation number.

I16

column number specifies the column number to retrieve the information for.

I16

column attributes specifies the attribute type to check for.

Value	Description
0 – Update	Determines if a column can be updated.
1 – Null	Determines if a column can have a null value.
2 – Search	Determines if a column can be used in a WHERE clause.
3 – Signed	Determines if a column is signed or unsigned.
4 – Money	Determines if a column is of type Money.
5 – Auto Increment	Determines if a column is automatically incremented on an update or insert.

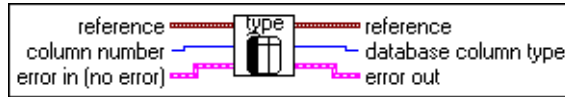
I16

attribute value returns the information about the specified column's specified attribute.

Checked For	Attribute Value	Description
Update	0	The column cannot be updated.
	1	The column can be updated.
	100	Unknown
Null	0	The column cannot be null.
	1	The column can be null.
	100	Unknown
Search	0	The column cannot appear in a WHERE clause.
	1	The column can appear in a WHERE clause only when used with the LIKE operator.
	2	The column can appear in a WHERE clause except with the LIKE operator.
	3	The column can appear anywhere within a WHERE clause.
	100	Unknown
Signed	0	The column is signed.
	1	The column is unsigned.
Money	0	The column is not of type Money.
	1	The column is of type Money.
Auto Increment	0	The column is not automatically incremented.
	1	The column is automatically incremented.

Get SQL Toolkit Column Type and Get Database Column Type

Returns the underlying database data type or the SQL Toolkit data type of the specified column.



I16

reference specifies a previously executed SQL operation number.

I16

column number specifies the column number for which the data type is being retrieved.

I16

sql toolkit column type/database column type is the SQL Toolkit data type code or the database type code, depending on the function executed, of the specified column.

sql toolkit column type is the SQL Toolkit data type code.

database column type is the actual data type code as defined by the database.

Table 6-1. Data Column Type Codes and Descriptions

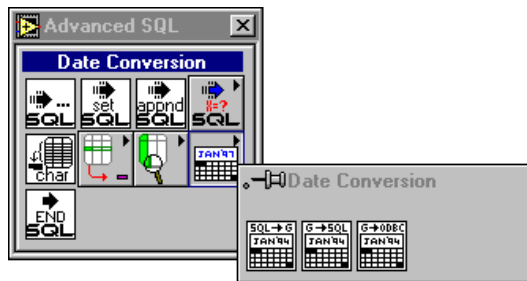
Type Code	Data Type Description
1	fixed length character string
2	variable length character string
3	Binary Coded Decimal (BCD) number
4	4-byte long integer
5	2-byte integer
6	4-byte single precision floating-point
7	8-byte double precision floating-point
8	26-byte date-time string of the form: YYYY-MM-DD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

**Request,
Request Database Information,
Request DSN Information,
Request Procedure Col Information,
Request Type Information, and
Request Table Information**

Obtain information about data sources, database names, tables, columns, and column types. Each VI generates an SQL reference when executed. To obtain the results, make a call to the Fetch Query Results VI to retrieve the data. Because this generates an SQL reference, you must call End SQL to clear SQL resources.

Date Conversion VI Descriptions

These VIs provide additional functions for database operations. To access the Date Conversion VIs, select **Functions»SQL»Advanced SQL»Date Conversion**.



**G Date to ODBC Date,
G Date to SQL Date, and
SQL Date to G Date**

Perform conversions between SQL Toolkit date-time strings and G's date-time cluster representation.





seconds specifies the date-time as the number of seconds elapsed since 12:00 a.m., Friday, January 1, 1904.



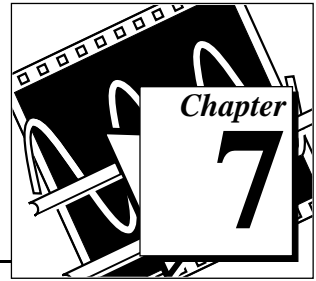
SQL date is the date string taking the form of:
[dt 'YYYY-MM-DD HH:MM:SS.SSSSS']



ODBC date is the date string taking the form of:
YYYY-MM-DD HH:MM:SS.SSSSS

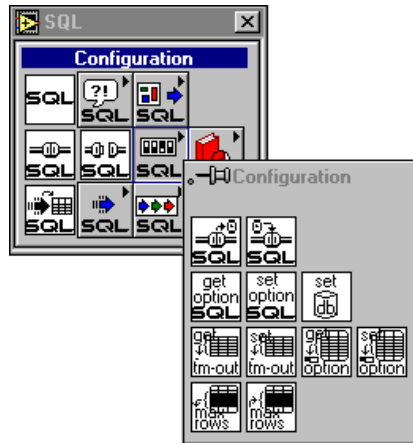


date/time record contains several elements representing the LabVIEW date.



Configuration VIs

This chapter describes the Configuration VIs, which you can access by selecting **Functions»SQL»Configuration**.

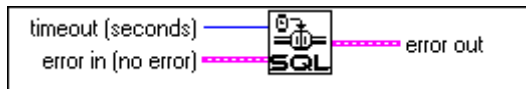


Configuration VI Descriptions

The following Configuration VIs are available.

Set Login Timeout and Get Login Timeout

Sets and retrieves the number of seconds to wait for a login request to complete before returning.



timeout specifies the timeout in seconds. A value of 0 indicates there is no timeout.

Set Query Timeout and Get Query Timeout

Sets and determines the number of seconds to wait for an SQL statement to execute before aborting the query.



I16

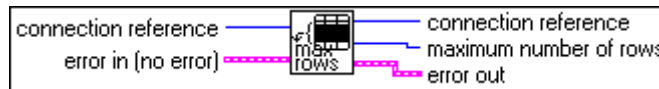
connection reference specifies an active database connection number.

I32

timeout specifies the timeout in seconds. A value of 0 indicates there is no timeout.

Set Maximum Rows and Get Maximum Rows

Sets and retrieves the maximum number of rows that can be returned by an SQL `SELECT` statement.



I16

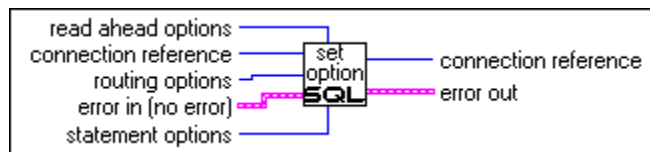
connection reference specifies an active database connection number.

I16

maximum number of rows specifies the maximum number of rows to be returned by an SQL `SELECT` statement. A setting of **0** indicates all rows will be retrieved.

Set Statement Options

Sets the options that determine the fetching commands and statement behaviors that are allowed when connected to data sources that support only one SQL statement per connection.





read ahead options is an enumeration of the following options:

execution reads entire result set into log file.

update (default) reads remainder of the result set into a log file whenever a record is locked, updated, or deleted.

commit updates avoids all read-ahead activity by requiring you to commit all updates before fetching more records.



connection reference specifies an active database connection number.



routing options is an enumeration of the following options:

read routes the statement through a connection used for read-only statements.

update routes the statement through a connection used for statements that modify a database.

default (default) lets the driver manager decide what connection to send the statement to.



statement options is an enumeration of the following options:

local cannot affect any other active SQL statement in this application.

non-local (default) can affect other SQL statements in the same application.

Get Statement Options

Returns the settings that determine which fetching commands and statement behaviors are allowed when connected to data sources that support only one SQL statement per connection.



connection reference specifies an active database connection number.



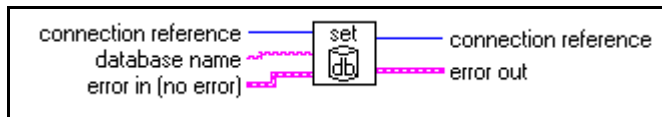
statement options is an enumeration of the following options.

local cannot affect any other active SQL statement in this application.

non-local (default) can affect other SQL statements in the same application.

Set Database

Specifies the default database to use for subsequent operations on the referenced connection.



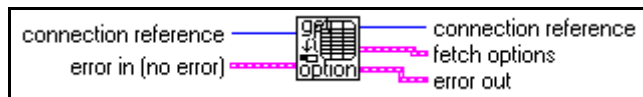
connection reference specifies an active database connection number.



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

Set Fetch Options and Get Fetch Options

Specify and determine how you can use the fetch VIs to retrieve records from the active set returned by SQL SELECT operations.



connection reference specifies an active database connection number.



fetch options specifies or determines, depending on the function called, 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**

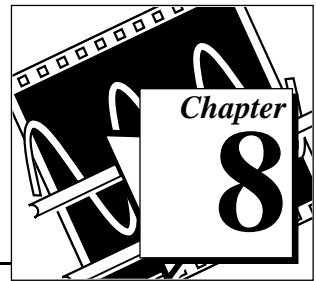
This VI selects which VIs can retrieve query data from an SQL `SELECT` operation. You can call this VI any time after you establish an active connection. You set options on each connection reference individually. The Fetch Options cluster elements determine three related aspects of how SQL Toolkit manages the active set, fetching, log files, and active set persistence, after you complete a transaction.

When **forward fetching** is `True`, you can use the Fetch Next Record VI. When forward/random/previous fetching is `True`, you can use Fetch Next Record, Fetch Previous Record, Fetch Record N, and Get Number of Records VIs.

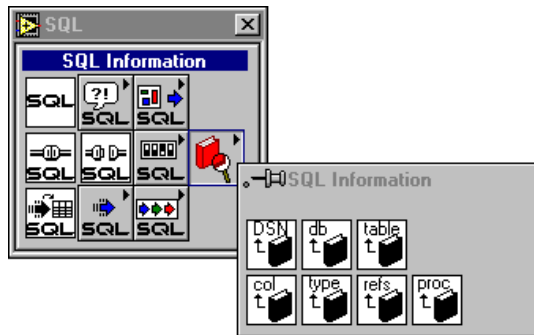
Depending on the capabilities of the underlying database, SQL Toolkit might need to write the active set to local disk files to support random fetching. When Fetch Previous Record, Fetch Record N, or Get Number of Records are first executed after an SQL `SELECT` operation, they transfer the entire query result to the temporary files. Subsequent fetch executions retrieve rows from those temporary files into LabVIEW. When **log file when needed** is `True`, you use log files only if required. When **log file always** is `True`, the active set is always logged to disk (this feature is useful for debugging). Ensure that there is sufficient disk space available, especially for large queries. Additionally, under Windows, a sufficient number of open DOS files must be permitted. If the number of open files exceed the DOS limit, use the Close Fetch Log File VI to close any temporary files not currently in use. Doing so preserves the contents of files closed, and reopens the files on subsequent fetches to those SQL references, provided the references remain active.

To use RAM and disk space effectively, SQL Toolkit supports three options to manage the active set upon concluding a transaction (see Chapter 5, *Transaction Operation VIs*, for more information). When **disable fetch after EOT** is `True`, the entire active set is disposed after the current transaction ends. 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 ends.

SQL Information VIs



This chapter provides descriptions of the SQL Information VIs. To access these VIs select **Functions»SQL»SQL Information**. This displays the **SQL Information** palette.

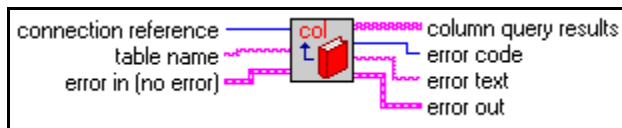


SQL Information VI Descriptions

The following SQL Information VIs are available.

Get Column Information

Returns information (such as the table qualifier, table user, table name, column, type, width, DB type, DB type name, precision, scale, nullable, and comments) about the columns from the specified table.



connection reference specifies an active database connection number.



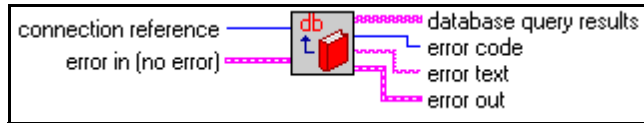
table name specifies the table name for which the column information is returned.

[abc]

column query results are the results of the column information query in a 2D array of strings.

Get Database Information

Returns information about the databases that the specified connection can access.



116

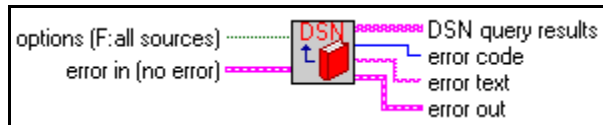
connection reference specifies an active database connection number.

[abc]

database query results displays the results of the database information query in a 2D array of strings.

Get DSN Information

Returns information (such as the name, extension, connection reference, and comments) about the available data sources.



TF

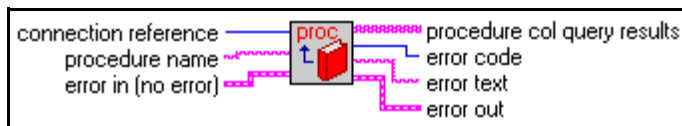
options specifies which data sources to retrieve information from. If set to `True`, it retrieves information only from connected data sources. If set to `False`, it retrieves information from all data sources.

[abc]

DSN query results displays the results of the data source information query in a 2D array of strings.

Get Procedure Col Information

Returns information (such as the procedure qualifier, owner, name, column name, column type, data type, database type name, width, precision, scale, nullable, and comments) about the stored procedure.



I16

connection reference specifies an active database connection number.

abc

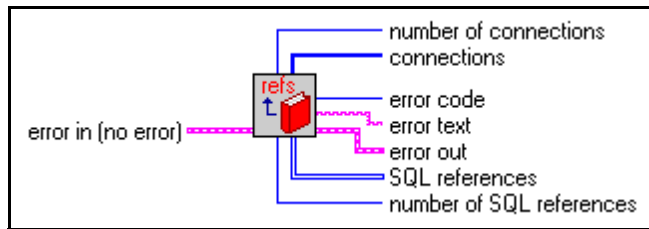
procedure name specifies the name of the stored procedure.

[abc]

procedure col query results displays the results of the stored procedure information in a 2D array of strings.

Get References

Returns the active connection and SQL references.



I16

number of connections is the number of active connections.

[I16]

connections are the active connection references.

I16

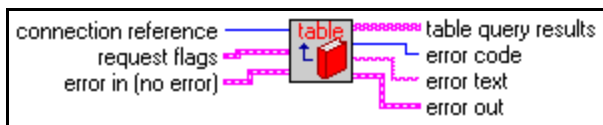
number of sql references is the number of active SQL references.

[I16]

sql references are the active SQL references.

Get Table Information

Returns table information (the qualifier, user, name, type, and comments) from the specified connection.



I16

connection reference specifies an active database connection number.



request flags specifies which type of tables from which to retrieve information. If no flags are set to True, database names are returned.



get table names



get view names



get stored procedure names



get system table names



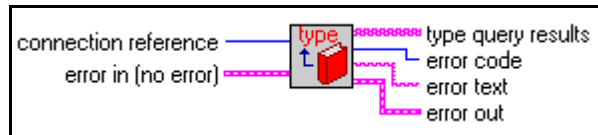
get synonym names



table query results are the results of the table information query in a 2D array of strings.

Get Type Information

Returns information (such as the type name, type, data type, width, precision, scale, literal prefix, literal suffix, create parameters, nullable, case sensitive, searchable, unsigned, money, auto increment, and local type name) about the data types supported by the specified database.

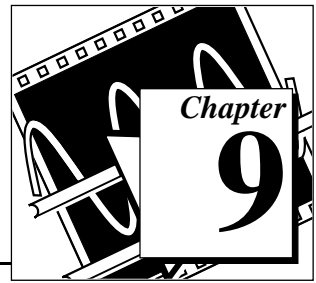


connection reference specifies an active database connection number.



type query results are the results of the data type information in a 2D array of strings.

Application Examples



This chapter presents example applications that incorporate SQL Toolkit VIs. The first, a quality control test station application, details a generic top-down structured design process for developing LabVIEW and BridgeVIEW applications that incorporate database functionality. The other examples act as tutorials and/or starting points for your application of SQL Toolkit technology.

You can find these VIs in the SQL library, which is typically located in the `\EXAMPLES\SQL` directory.

SQL Toolkit Examples

The following sections introduce many working example programs that show how to use the SQL Toolkit libraries to develop typical G application VIs. The examples illustrate the basic steps and VI calls necessary to enter data into a database, and retrieve data from the database. The examples are constructed from both the high-level Access VIs and the low-level Interface VIs, and show how the VIs are combined to achieve database operation. Look in the examples directory to find the SQL Toolkit Examples, if installed. You can explore and execute these VIs, and use and adapt them for your own applications.



Note: *All SQL Toolkit examples use the dBASE driver included with the product, creating new tables and/or using existing example tables provided with the product.*

SQL Toolkit Demo

This example (`General.11b\SQL Toolkit Demo`) illustrates the typical database operations that G performs. Run this demonstration to see the actions performed, then explore the block diagrams. First, a connection is made to the data source, which creates the table. Then, data appears in the table, and two queries are performed. Finally, the database is disconnected, and the table optionally is dropped (deleted).

Quick SQL

This example (General.11b\Quick SQL) illustrates SQL Toolkit programming at its simplest. The VI executes any valid SQL statement and displays results of a `SELECT` statement. Figure 9-1 shows the Quick SQL front panel.

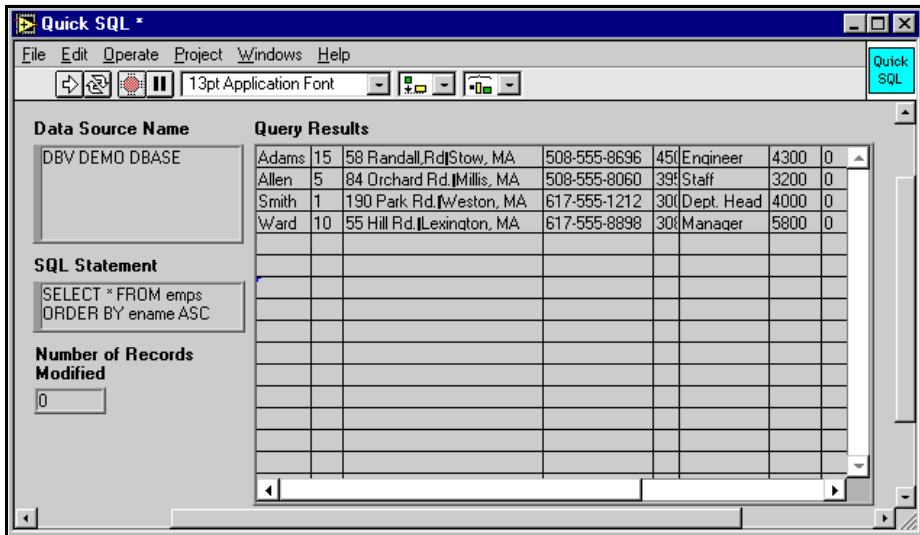


Figure 9-1. Quick SQL Panel

The Quick SQL block diagram (Figure 9-2) employs a single VI (see [Complete SQL Session](#) in Chapter 2, [Using SQL Toolkit VIs](#)), that connects to the dBASE table emps, executes the SELECT SQL statement, retrieves the query results, disposes of the SQL reference, and disconnects from dBASE, terminating the session.

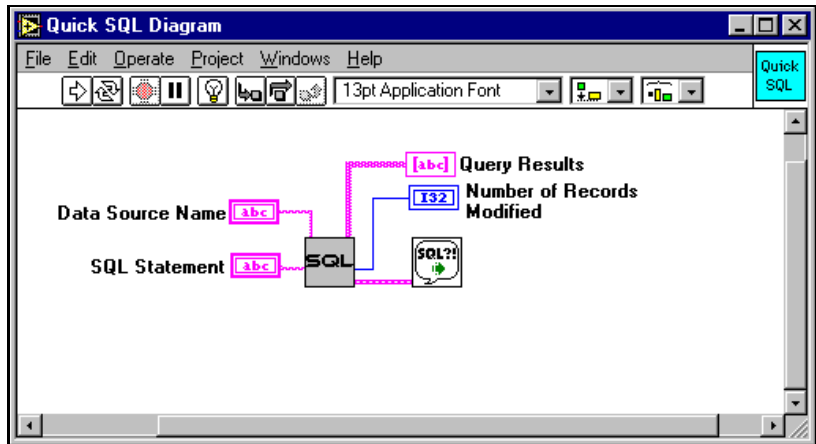


Figure 9-2. Quick SQL Block Diagram

Sample Transaction

This example (General.11b\Sample Transaction) shows how you can implement transactions with the SQL Toolkit. As seen in Figure 9-3, to start a transaction on a specified connection, place the Start Transaction VI in the execution flow after Connect VI and before any SQL Execution VIs. The database then caches all subsequent SQL command operations until either the Commit Transaction VI executes to

make all database changes permanent, or the Rollback Transaction VI runs to discard the database operations since the start of the transaction.

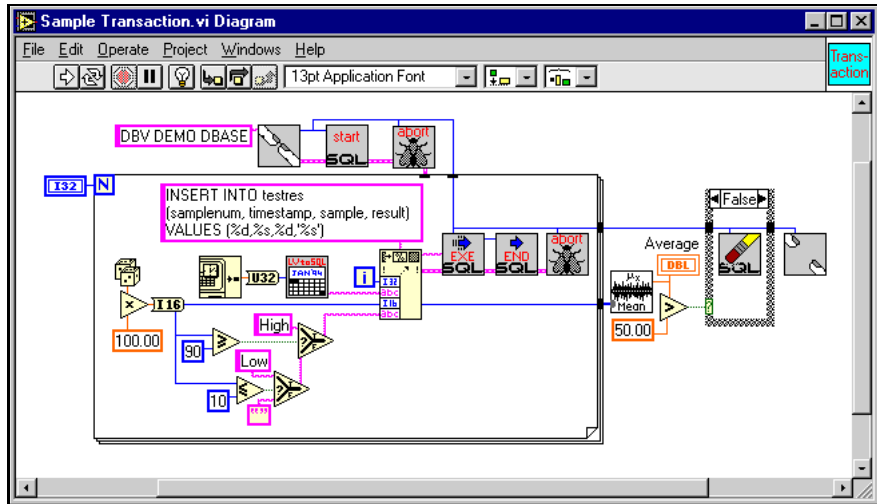


Figure 9-3. Sample Transaction Demo Diagram

Employee Record

This example (General.11b\Employee Record) presents a typical human resources database that contains records about employees in a small engineering company. This VI shows you how to build SQL statements interactively based on user input. Basic employee data is in the emps dBASE table supplied with the SQL Toolkit. You can insert new employees, query and examine existing employee records, and delete records. Explore the block diagrams to see the simple string formatting used to create the SQL statements.

Weather Examples

These examples (General.llb\Weather Station DAQ, Weather Analyzer, and SQL Optimization Demo) present a typical G science application. Weather Station DAQ, as seen in Figures 9-4 and 9-5, simulates a weather data acquisition system that collects data such as temperature, wind speed, barometric pressure, humidity, and precipitation. Daily statistics for a year's data are stored to the dBASE table climate. The Front panel displays show each value from the simulation in graphical form. Deselect **Show Indicators** and run the demo to see how fast the data insert for each day can be.

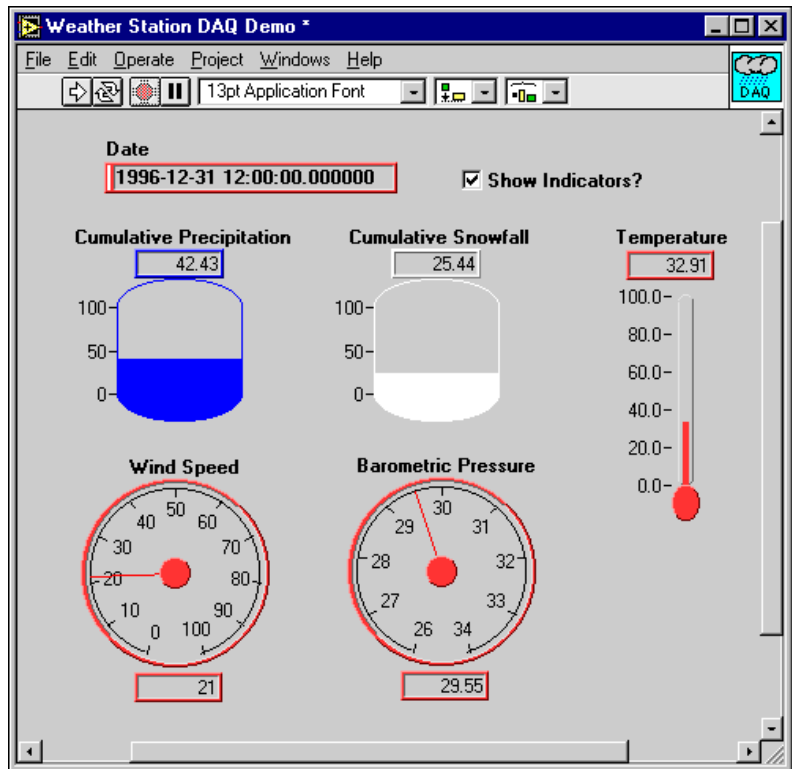


Figure 9-4. Weather Station DAQ Panel

Figure 9-5 shows the dynamic SQL method of inserting data into the database using statement parameters.

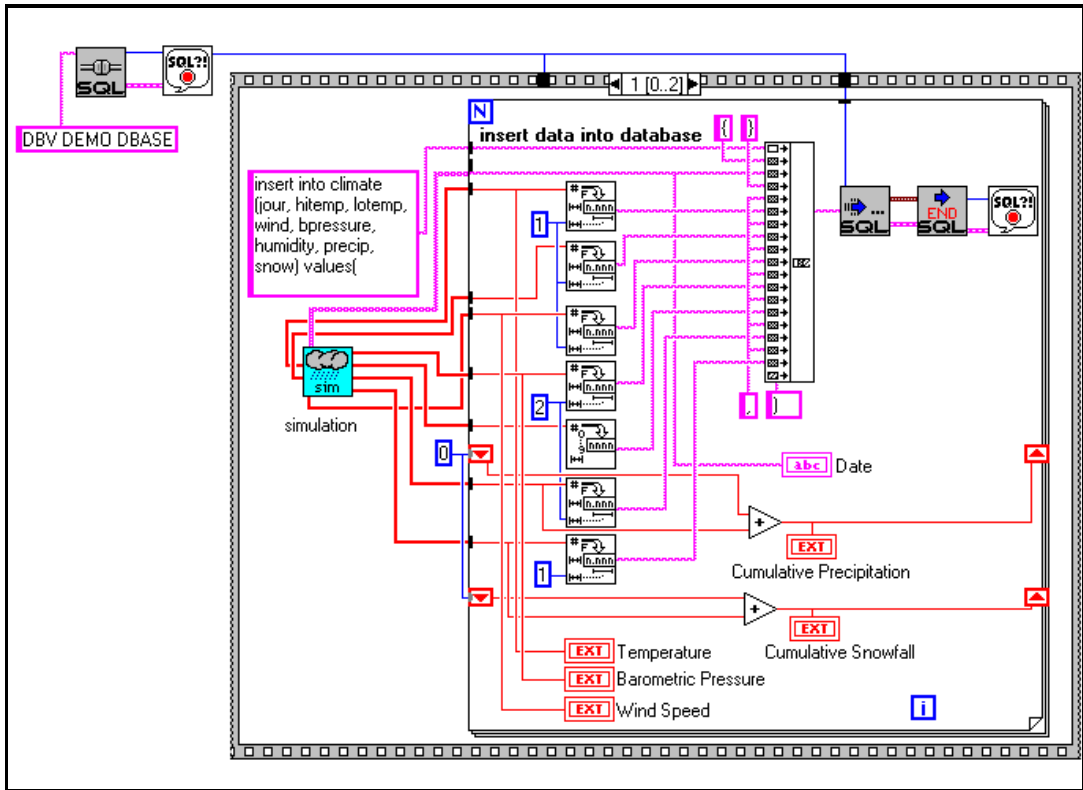


Figure 9-5. Weather Station DAQ Diagram

The Weather Analyzer VI shows a typical database query application where interactive input on a front panel creates SQL statements. End users now can perform database queries without learning SQL. The Weather Analyzer uses the climate table created by Weather Station DAQ.

The Weather Analyzer VI illustrates two approaches to developing a user interface for computing and viewing common weather statistics. The first approach, as shown in Figure 9-6, employs specific canned queries where the user quickly examines weather parameters in standard views with a minimum of input.

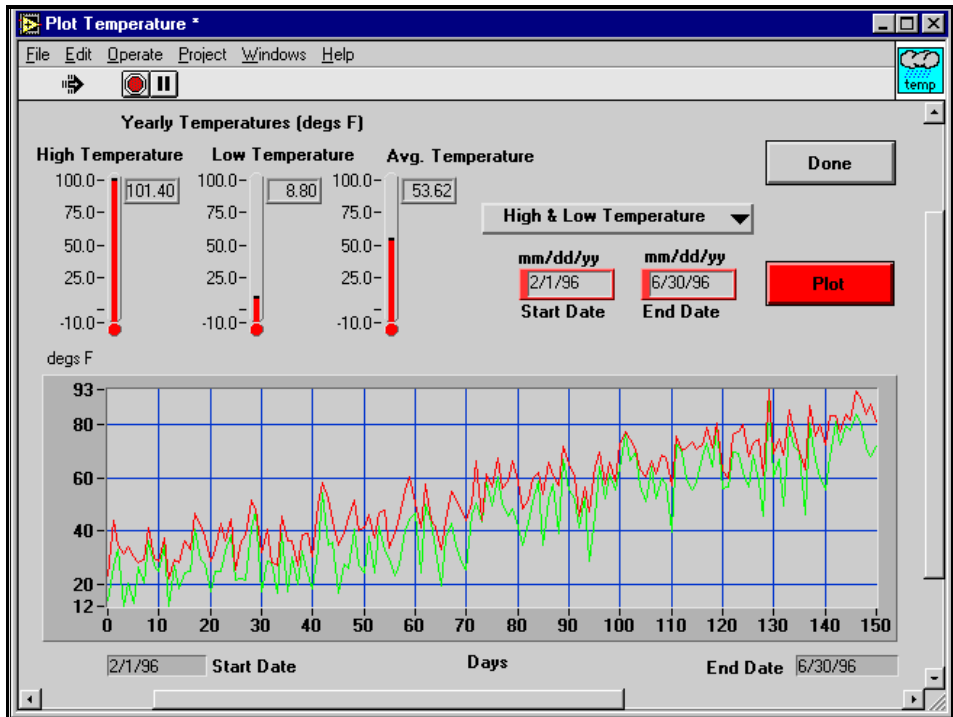


Figure 9-6. Canned Query Example Panel

The SQL statement generator Weather Query VI is another method of querying the climate database, as shown in Figure 9-7. You create very detailed queries using the various interactive controls to extract specific information from the database. As you build the SQL statement, a string indicator displays it at the bottom of the dialog window. This may help you learn SQL `SELECT` syntax and usage.

You can then execute statements and display results in a window containing a LabVIEW table.

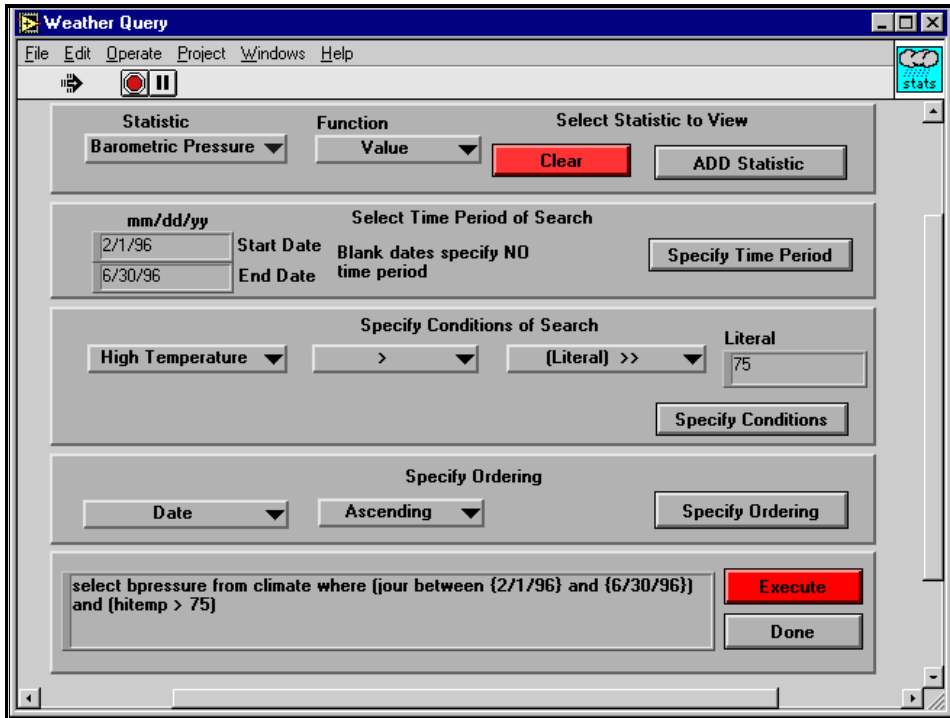


Figure 9-7. Weather Query Generator Panel

Insert Examples

These examples (Insert .11b) show four methods of inputting data into a table. Each uses a different SQL technique that vary in ease-of-use, performance, and flexibility. The Insert Benchmark VI can compare the methods by performance. The block diagrams that follow may give you a better understanding of how to use these techniques.

The traditional ANSI SQL INSERT method is shown in Figure 9-8. LabVIEW string formatting functions create a properly formatted INSERT statement that the Execute SQL VI receives.

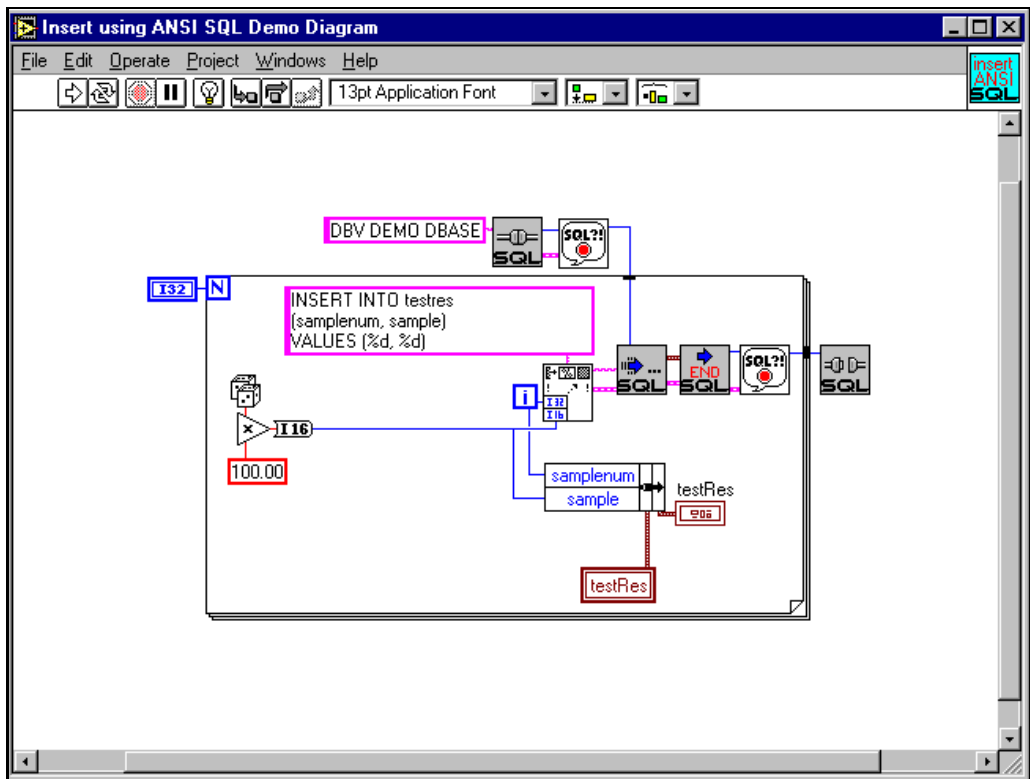


Figure 9-8. Insert Using ANSI SQL Demo Diagram

The SQL Toolkit can execute parameterized Dynamic SQL statements. As seen in the Figure 9-9, the INSERT SQL containing wildcards (question marks) is preprocessed using Prepare SQL VI. Individual values are bound to parameters using the Set SQL Parameter VIs. Then the prepared statement is executed. Dynamic SQL can be four times faster than string-only SQL.

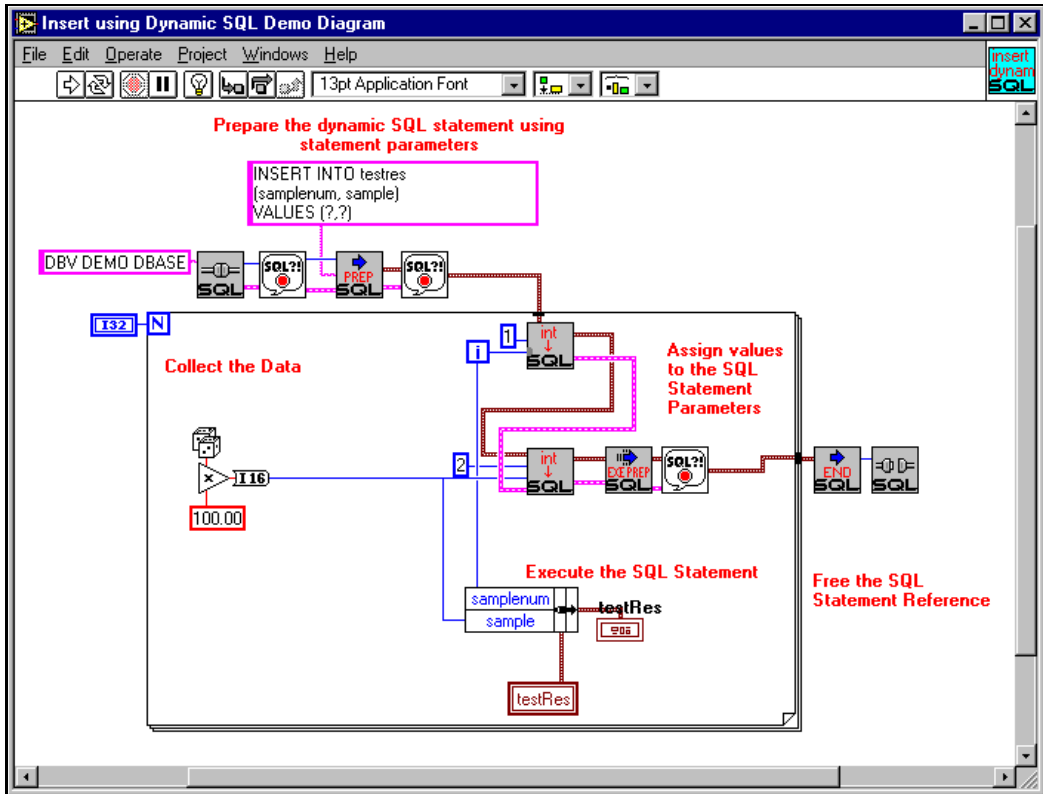


Figure 9-9. Example Dynamic SQL INSERT Diagram

The SQL Toolkit automatically formats SQL statements using G cluster definitions, which can improve usability. By using the Cluster to INSERT SQL VI, you can create a custom version of the VI for the cluster of information input to the database. The VI then creates an SQL statement to insert the values for each of the controls into like-named columns in a table named the same as the cluster. This method, as shown in Figure 9-10, simplifies database programming.

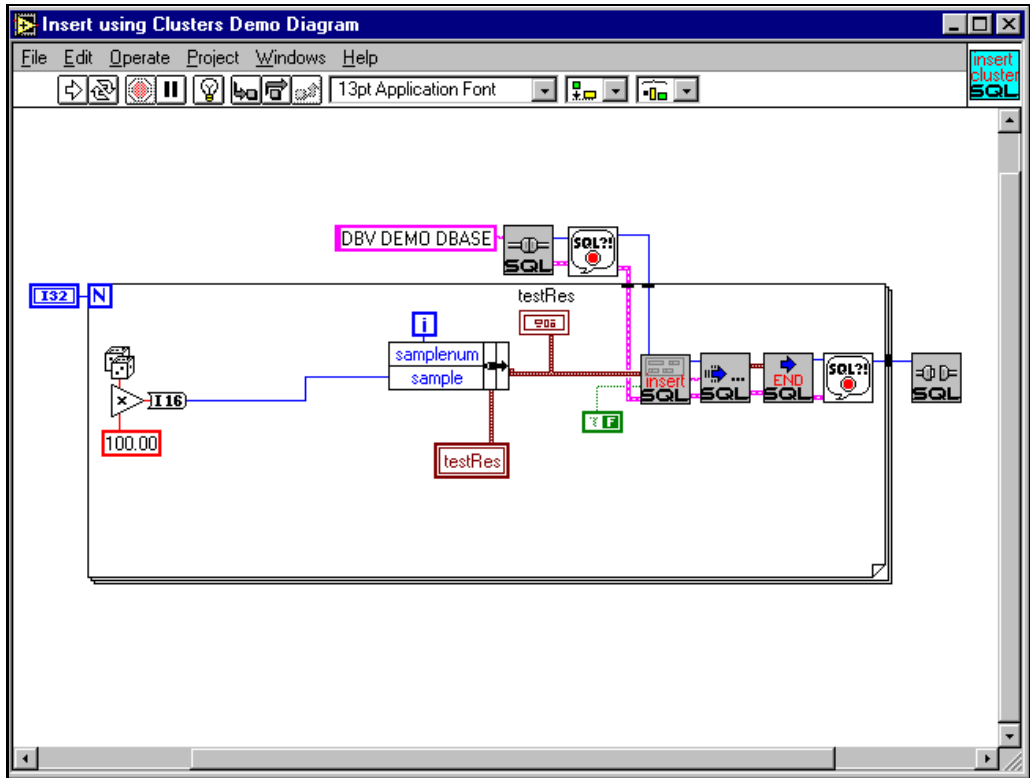


Figure 9-10. Example of Cluster to SQL INSERT Diagram

Figure 9-11 shows a combination of these methods.

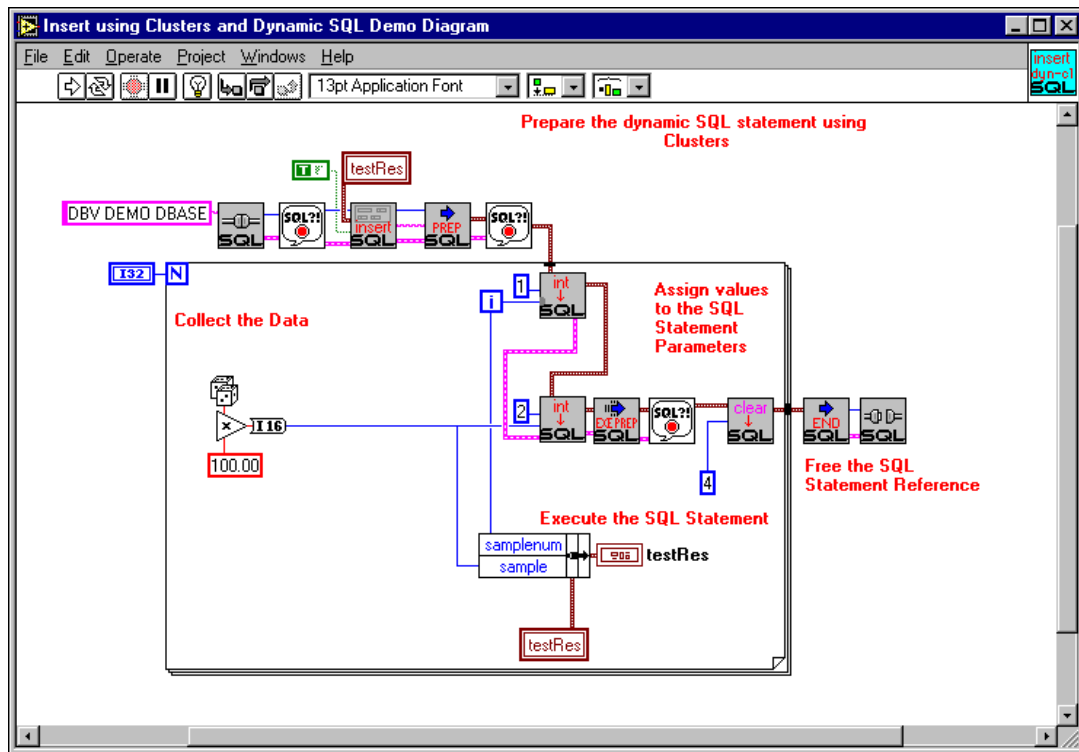


Figure 9-11. Insert Using Clusters and Dynamic SQL Demo Diagram

Cluster conversion VIs create a preprocessed Dynamic SQL parameter statement. Then the same methods for the Dynamic SQL INSERT execution of prepared SQL statements is used. This method is easy to use, like the cluster method, and has improved performance, like the Dynamic SQL method.

Query Examples

These examples (Query.11b) demonstrate three ways to retrieve results from a database. Similar in performance, they offer a variety of formatting for returned information. Standard SQL SELECT operations of SQL Toolkit options return information in a two-dimensional array of strings (table) that you convert into standard data types by user programming as shown in Figure 9-12.

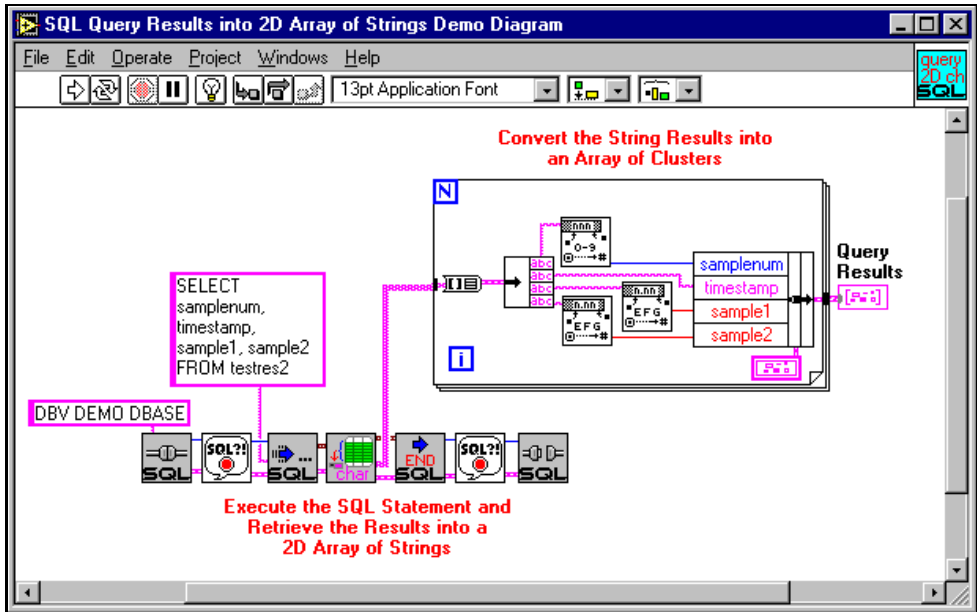


Figure 9-12. Example ANSI SQL SELECT to 2D Array of Strings Diagram

You combine an identical SQL `SELECT` statement with another VI you created to fetch the query results into standard data types, which creates an array of clusters. By using low level VIs from the **Functions»SQL»Fetch Data Parsing**, you can directly map the fetch values into standard types. See the Fetch My Query Results subVI of the SQL Query Results to Standard Data Types example VI, as shown in Figure 9-13.

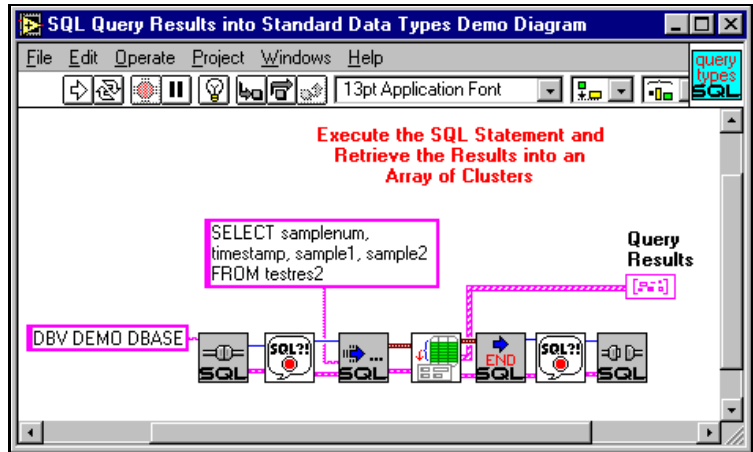


Figure 9-13. SQL Query Results into Standard Data Types Demo

Finally, you can retrieve query results automatically into an array of clusters using the Cluster VIs. As shown in Figure 9-14, an SQL `SELECT` statement is automatically generated using a copy of the Cluster to `SELECT` SQL Template VI and then executed using Easy

SQL. Then the query results are converted into an array of type matched clusters using a copy of the Results to Cluster Template VI.

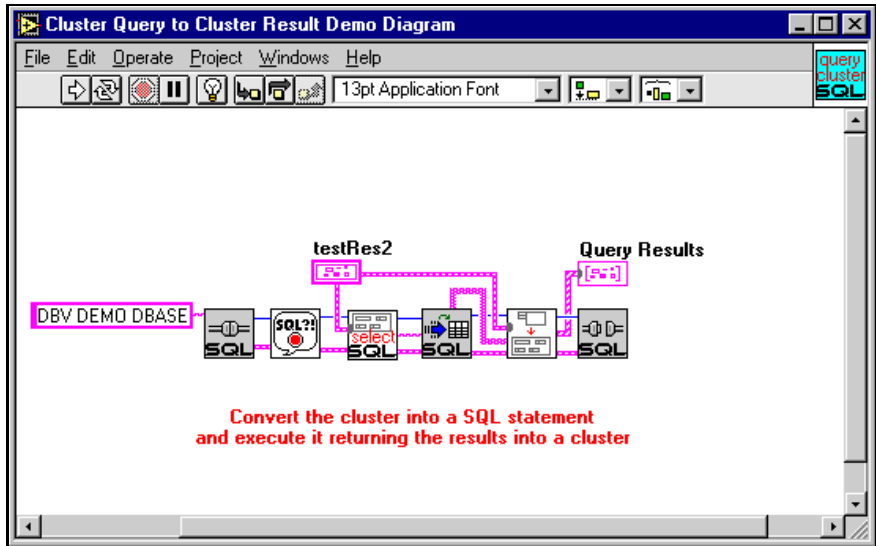


Figure 9-14. Cluster Query to Cluster Results Demo

Database Browser Utility

The Database Browser VI, which you can access from the Project menu, provides an interactive tool to examine and interact with available database resources. Figure 9-15 shows the main user interface to the Database Browser.



Figure 9-15. Database Browser VI User Interface Dialog Box

Nine operations are supported, from viewing available data sources to executing prototype SQL statements. You can create queries interactively for easy data mining. This VI is particularly useful during integration and unit testing of your application.

You can open the Database Browser VI and run it just as any other VI. You can run it when your development application is running, during a break or pause point, or after an error condition occurs. The program can probe the internal SQL Toolkit data structures, as well as the ODBC resources.

To view defined data sources on your system, click on **View DSN Attributes** of the Database Browser VI. The following dialog box appears.

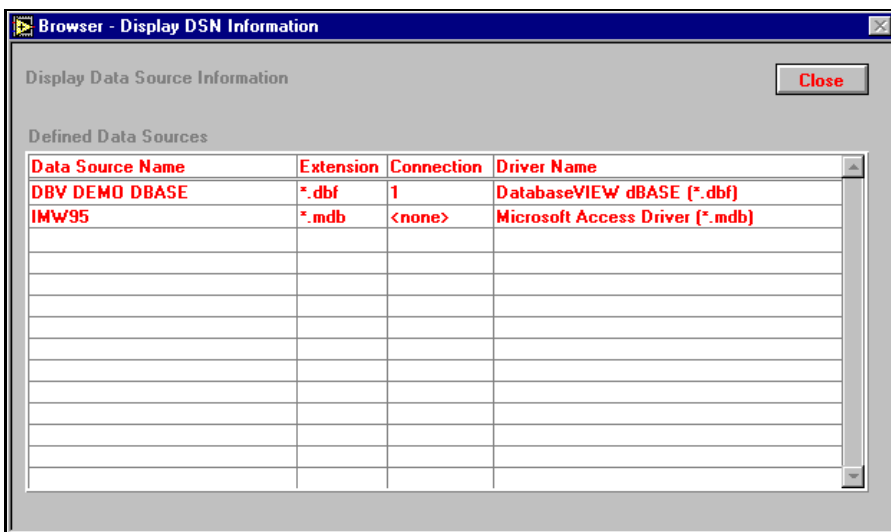


Figure 9-16. Data Source Attributes Dialog Box

The Data Source Name (DSN), the file extension of the database (if any), any active connection reference numbers, and the driver associated with the DSN displays.

To determine supported data types of a particular data source, click on **View Data Types** of the Database Browser VI. The following dialog box appears.

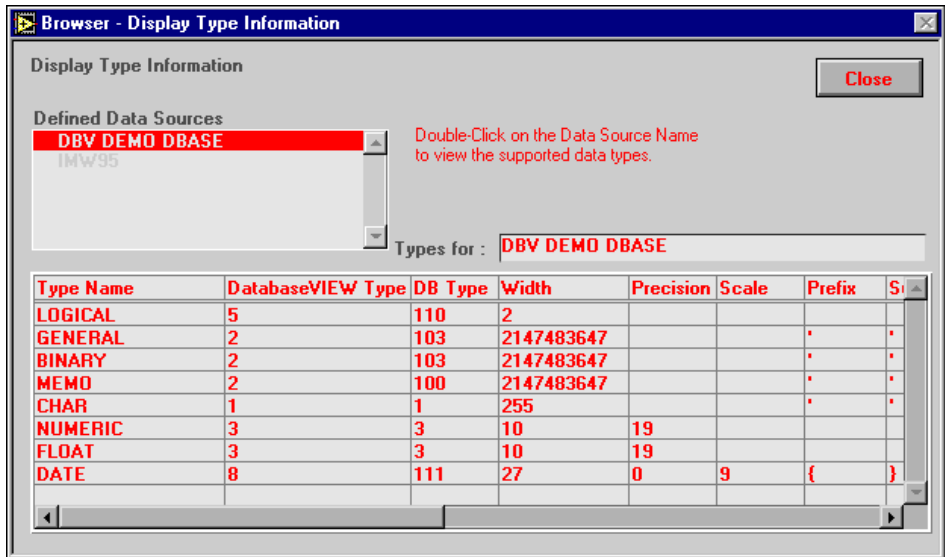


Figure 9-17. Datatype Information Dialog Box

You can select an individual data source to reveal a list of data type attributes for the database fields you can create.

You can access existing tables through a data source name, which appears when you click on **View Tables**. The following window appears.

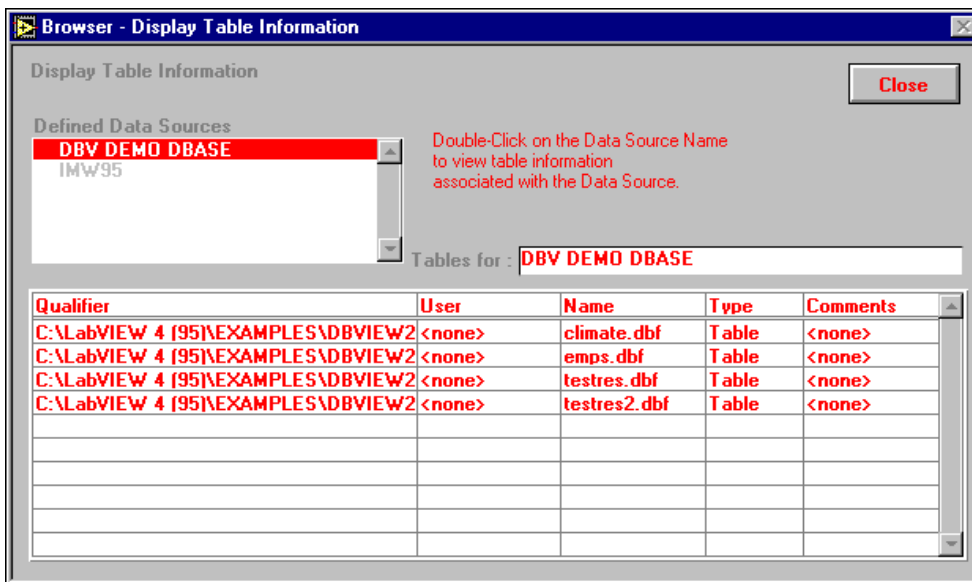


Figure 9-18. Table Information Dialog Box

The dialog lists all tables for the chosen data source, along with **User**, **Type**, and **Comments**.

To display information about the columns of a database table, click on **View Table Attributes**. The following dialog box appears where you can select a data source and a table.

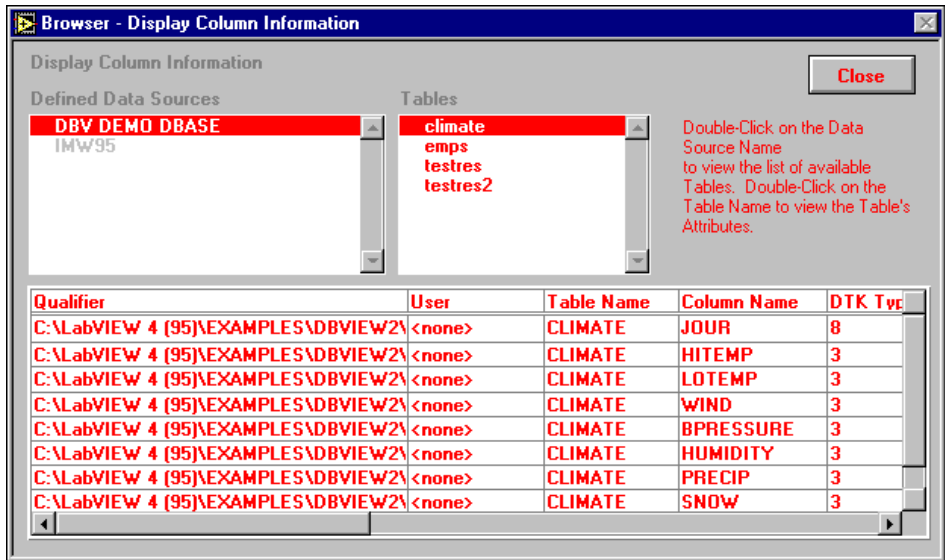


Figure 9-19. Column Information Dialog Box

A list appears with information about the **Table Name**, **User**, **DTK Type**, and attributes of each database column.

When the database connections and/or SQL operations are active, you can view the internal SQL Toolkit reference numbers by clicking on **View References**.

As shown in Figure 9-19, active connection and SQL reference numbers are displayed in lists. These numbers are the same reference numbers that are automatically generated and used in your block diagram. Each connection reference can have zero, one, or many SQL references associated with it. The following figure shows the **Display References** dialog box.

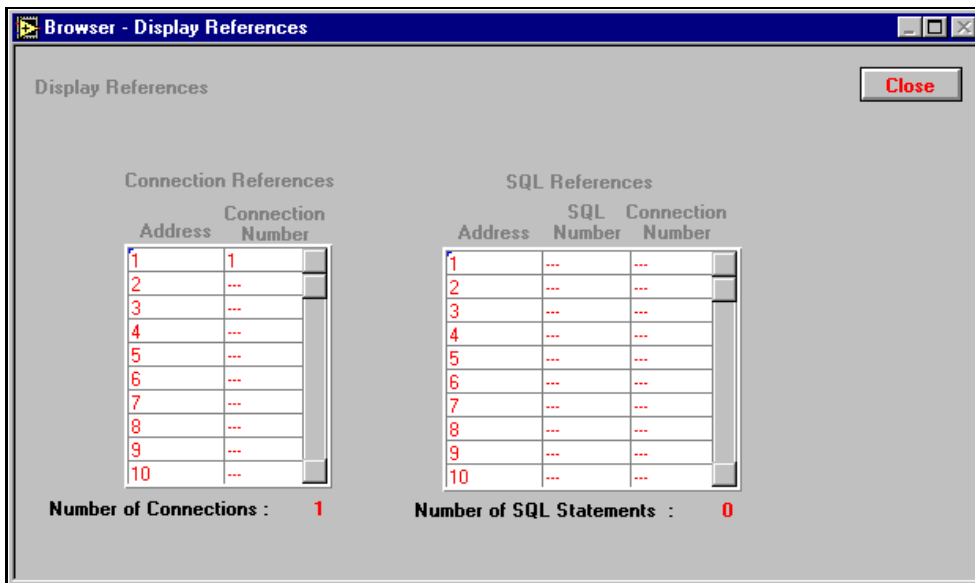


Figure 9-20. Display References Dialog Box

The Database Browser VI also assists in the testing of SQL statements during program development. You can execute SQL statements using the **Execute SQL** dialog box, which appears when you click **Execute SQL**. The following dialog box appears.

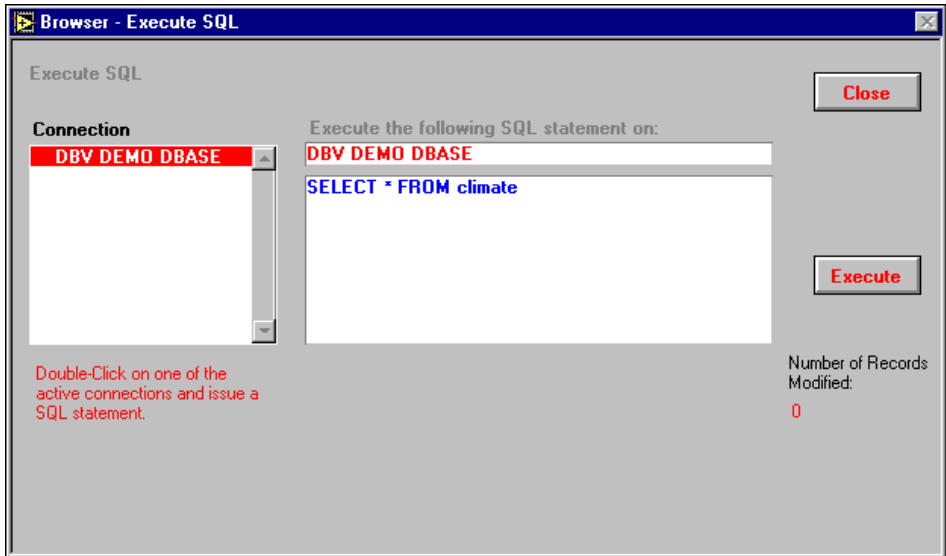


Figure 9-21. Execute SQL Dialog Box

Successful statement execution results in a confirmation message. A dialog box displays any errors made during execution. To execute SQL statements, select a desired data source, enter the SQL, and click on **Execute**. Use the interactive Query Builder to retrieve information from a database table. You access it by choosing **Query Builder** on the Database Browser User Interface Panel as shown in Figure 9-15. Then, to interactively build an SQL `SELECT` statement to query the database, choose the Data Source, table, column(s), and optional conditions. The following figure shows the **Query Builder** dialog box with a completed `SELECT` statement that retrieves the entire contents of the climate table.

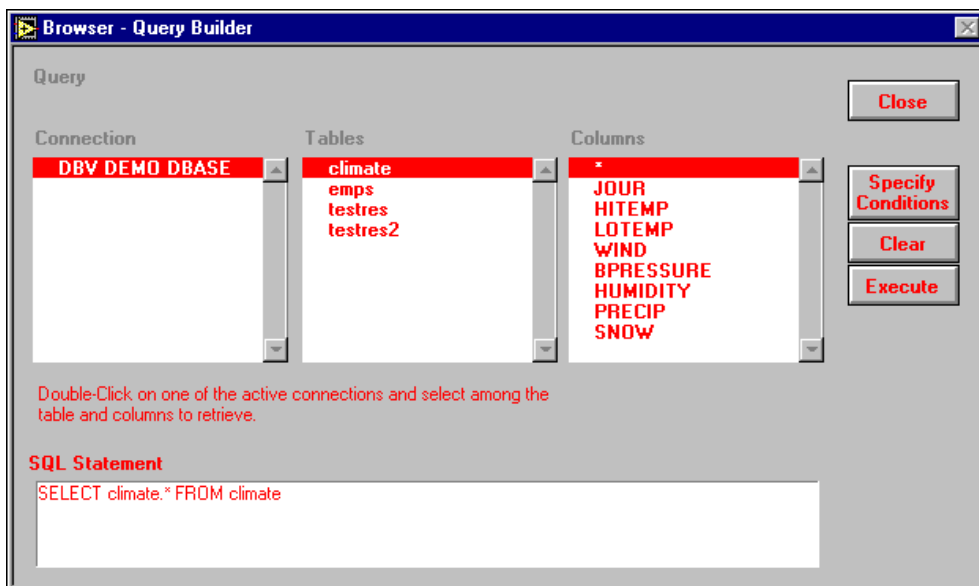
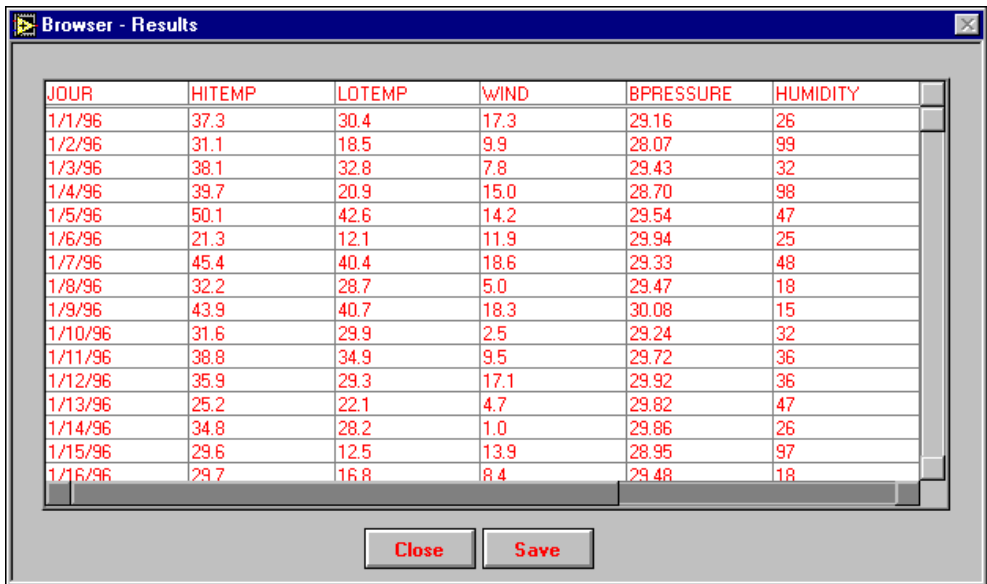


Figure 9-22. Query Builder Panel

A dialog box displays the results of this query in a window as shown below.



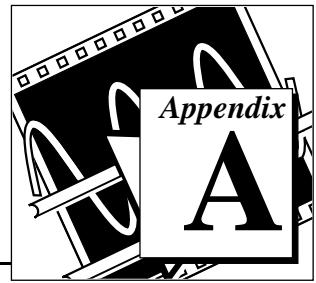
JOUR	HITEMP	LOTEMP	WIND	BPRESSURE	HUMIDITY
1/1/96	37.3	30.4	17.3	29.16	26
1/2/96	31.1	18.5	9.9	28.07	99
1/3/96	38.1	32.8	7.8	29.43	32
1/4/96	39.7	20.9	15.0	28.70	98
1/5/96	50.1	42.6	14.2	29.54	47
1/6/96	21.3	12.1	11.9	29.94	25
1/7/96	45.4	40.4	18.6	29.33	48
1/8/96	32.2	28.7	5.0	29.47	18
1/9/96	43.9	40.7	18.3	30.08	15
1/10/96	31.6	29.9	2.5	29.24	32
1/11/96	38.8	34.9	9.5	29.72	36
1/12/96	35.9	29.3	17.1	29.92	36
1/13/96	25.2	22.1	4.7	29.82	47
1/14/96	34.8	28.2	1.0	29.86	26
1/15/96	29.6	12.5	13.9	28.95	97
1/16/96	29.7	16.8	8.4	29.48	18

Figure 9-23. Climate Query Results Panel



Note: *To store these results to a tab-delimited text file, click the Save button.*

SQL Quick-Reference



This appendix lists and briefly explains the more common SQL commands, operators, and functions available for SQL Toolkit 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 *SQL Toolkit Driver Help for SQL* for 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. [Related Documentation](#) in *About This Manual* lists general-purpose and tutorial texts on SQL.

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

SQL Commands

The following table lists SQL commands you can use with the SQL Toolkit VIs.

Table A-1. SQL Commands

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

Table A-1. SQL Commands (Continued)

SQL Command	Syntax	Description and Examples
SELECT	<pre> 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, ...}] </pre>	<p>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.</p> <pre> SELECT col1, col2, col3 FROM tabl WHERE col1 >= (col3 * col2) ORDER BY col3 ASC </pre>
UPDATE	<pre> UPDATE table_defn [options] SET col_name = expr, ... [WHERE where_clause] </pre>	<p>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.</p> <pre> UPDATE tabl SET col1 = (col1 * 1.5) WHERE col1 <1000 </pre>

SQL Object Definitions

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

Table A-2. SQL Objects

Object Abbr.	Object Name	Description and Examples
table_defn	Table Definition	Describes a table on which to perform an operation. It may be simply 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. 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

Table A-2. SQL Objects (Continued)

Object Abbr.	Object Name	Description and Examples
date_expr	Date Expression	Any expression that results in a date datatype
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 A-3. SQL Clauses

Clause Name & Syntax	Applicable Commands	Description and Examples
<pre>FROM table_defn [options] [table_alias]</pre>	<pre>SELECT DELETE</pre>	<p>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.</p> <pre>SELECT FROM modtest</pre> <p>Table alias is used to specify a column name prefix to be used in subsequent clauses.</p> <pre>SELECT FROM C:\qcdata\modtes\MT</pre>
<pre>WHERE expr1 comparison_oper expr2 [logical_oper expr3 comparison_oper expr4]...</pre>	<pre>SELECT DELETE UPDATE</pre>	<p>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.</p> <pre>SELECT * FROM C:\qcdata\modtes\MT WHERE (MT.fld1 = 3 AND MT.fld2 <= 365)</pre> <p>(Note use of table alias MT.)</p>
<pre>GROUP BY col_expr{, col_expr,...}</pre>	<pre>SELECT</pre>	<p>Specifies one or more column expressions to use to group active set rows.</p> <pre>SELECT * FROM C:\qcdata\modtest\MT WHERE (MT.fld1 = 3 AND MT.fld2 <= 365) GROUP BY MT.fld4</pre>

Table A-3. SQL Clauses (Continued)

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

SQL Operators

The following table lists SQL expressions and operators.

Table A-4. SQL Operators

Operator Class	Operators	Description	Example
Constants		numeric constant	1234, 1234.5678
	' ' " "	character constant	'abcd', "abcd"
	{ }	date - time constant	{4/17/61}, {14:32:56}
	.T. .F.	logical constant	.T., .F.
Numeric	()	operator precedence	(A + B) * (C - D)
	+ -	sign	- A
	* /	multiply/divide	A * B, A / B
	+ -	add/subtract	A + B, A - B
	** ^	exponentiation	A**B, A^B
Character	+	concatenate, keep trailing blanks	'txt a '+'txt b' (gives 'txt a txt b')
	-	concatenate, move trailing blanks to end	'txt a '-'txt b' (gives 'txt atxt b')

Table A-4. SQL Operators (Continued)

Operator Class	Operators	Description	Example
Comparison (true / false)	=	equal	WHERE a = b
	<>	not equal	WHERE a <> b
	>=	greater than or equal	WHERE a >= b
	<=	less than or equal	WHERE a <= b
	IN	contained in the set ()	WHERE a IN ('apple', 'orange') WHERE a IN (SELECT...)
	NOT IN	not contained in the set ()	WHERE a NOT IN ('peach', 'pear')
	ANY, ALL	compare with a list of rows	WHERE a = ANY (SELECT ...)
	BETWEEN	within a range of values	WHERE c BETWEEN a AND e
	EXISTS	existence of at least one row	WHERE EXISTS (SELECT ...)
	[NOT] LIKE	character pattern match	WHERE a LIKE 'tar%'
[NOT] NULL	empty (no value)	WHERE a NOT NULL	
Date	+ -	add/subtract	testdate+5 (result is a new date) testdate - {1/30/18} (result is a number of days)

Table A-4. SQL Operators (Continued)

Operator Class	Operators	Description	Example
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 tabl

SQL Functions

The following table lists SQL functions.

Table A-5. SQL Functions

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

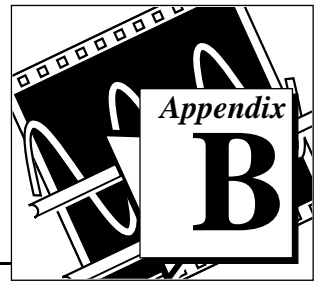
Table A-5. SQL Functions (Continued)

Function	Description
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 and number_expr2
MIN(column_expr)	minimum value of column_expr
MIN(number_expr1, number_expr2)	smaller of number_expr1 and number_expr2
SUM(column_expr)	sum of values in column_expr
DTC(date_expr, fmt_value[, 'separator_char'])	<p>convert date_expr from date to character string using fmt template and (optional) separator character (default is '/'). fmt_values are as follows:</p> <p>0 => MM/DD/YY 1 => DD/MM/YY 2 => YY/MM/DD 10 => MM/DD/YYYY 11 => DD/MM/YYYY 12 => YYYY/MM/DD</p>
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

Table A-5. SQL Functions (Continued)

Function	Description
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
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

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.



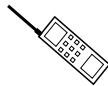
E-Mail Support (currently U.S. only)

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	03 9879 5166	03 9879 6277
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	09 725 725 11	09 725 725 55
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
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

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 for G 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 _____

HiQ, NI-DAQ, LabVIEW, or LabWindows 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 for G Reference Manual*

Edition Date: September 1997

Part Number: 321525B-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 _____

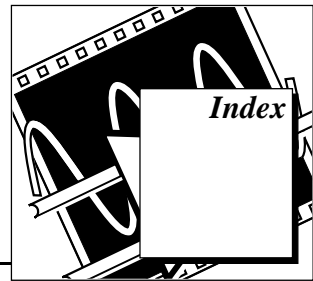
Company _____

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



A

Abort-Continue Error Dialog VI, 4-2 to 4-3

Abort Error Dialog VI, 4-1 to 4-2

advanced VIs

Column Information VIs, 6-14 to 6-19

Get Column Alias, 6-15

Get Column Attributes, 6-16 to 6-17

Get Column Expression, 6-15

Get Column Name, 6-15

Get Column Precision, 6-15

Get Column Scale, 6-15

Get Column Width, 6-15

Get Database Column Type, 6-18

Get SQL Toolkit Column Type, 6-18

Request, 6-19

Request Database Information, 6-19

Request DSN Information, 6-19

Request Procedure Col Information,
6-19

Request Table Information, 6-19

Request Type Information, 6-19

Date Conversion VIs, 6-19 to 6-20

G Date to ODBC Date, 6-19 to 6-20

G Date to SQL Date, 6-19 to 6-20

SQL Date to G Date, 6-19 to 6-20

Dynamic SQL VIs, 6-4 to 6-8

Clear SQL Parameter, 6-5

Execute Prepared SQL, 2-11, 6-6

Get Number of Parameters, 6-6

Prepare SQL, 6-5

Set SQL Binary, 6-6 to 6-7

Set SQL Character, 6-6 to 6-7

Set SQL Date, 6-6 to 6-7

Set SQL Date-Time, 6-6 to 6-7

Set SQL Decimal Parameter, 6-7 to 6-8

Set SQL Double, 6-6 to 6-7

Set SQL Float, 6-6 to 6-7

Set SQL Integer, 6-6 to 6-7

Set SQL Long, 6-6 to 6-7

Set SQL Parameter to Null, 6-8

Set SQL Time, 6-6 to 6-7

Fetch Data Parsing VIs, 6-9 to 6-14

Close Fetch Log File, 6-12

Fetch Character Column Data,
6-12 to 6-13

Fetch Column Data, 6-12 to 6-13

Fetch Decimal Column Data,
6-13 to 6-14

Fetch Double Column Data,
6-12 to 6-13

Fetch Float Column Data, 6-12 to 6-13

Fetch Integer Column Data,
6-12 to 6-13

Fetch Long Column Data, 6-12 to 6-13

Fetch Next Record, 6-10

Fetch Previous Record, 6-10

Fetch Query Results, 6-9 to 6-10

Fetch Record N, 6-10

Get Number of Columns, 6-11

Get Number of Modified Records, 6-11

Get Number of Records, 6-11

Handle Null Value, 6-12

Statement Execution VIs, 6-2 to 6-3

Append SQL, 6-3

End SQL, 6-3

Execute SQL, 6-2

Set SQL, 6-3

ALTER TABLE command, SQL, 1-11

Append SQL VI, 6-3

application examples, 9-1 to 9-23

Database Browser utility, 9-15 to 9-23

employee record, 9-4

- insert examples, 9-9 to 9-12
- query examples, 9-13 to 9-15
- Quick SQL, 9-2 to 9-3
- SQL Toolkit demo, 9-1
- transaction sample, 9-3 to 9-4
- weather examples, 9-5 to 9-8

B

- bulletin board support services, B-1

C

- character operators, SQL (table), A-8
- clauses, SQL (table), A-6 to A-7
- Clear SQL Parameter VI, 6-5
- Close Fetch Log File VI, 6-12
- Cluster to INSERT SQL VI
 - description, 4-4
 - using, 2-3 to 2-4
- Cluster to SELECT SQL VI
 - description, 4-4 to 4-5
 - using, 2-4 to 2-5
- Column Information dialog box (figure), Database Browser utility, 9-19
- Column Information VIs, 6-14 to 6-19
 - Get Column Alias, 6-15
 - Get Column Attributes, 6-16 to 6-17
 - Get Column Expression, 6-15
 - Get Column Name, 6-15
 - Get Column Precision, 6-15
 - Get Column Scale, 6-15
 - Get Column Width, 6-15
 - Get Database Column Type, 6-18
 - Get SQL Toolkit Column Type, 6-18
 - Request, 6-19
 - Request Database Information, 6-19
 - Request DSN Information, 6-19
 - Request Procedure Col Information, 6-19
 - Request Table Information, 6-19
 - Request Type Information, 6-19

- commands, SQL
 - common commands (table), 1-10
 - quick-reference (table), A-2 to A-3
 - variations on, 1-11
- Commit Transaction VI, 5-1 to 5-2
- communication
 - network communications, 1-8 to 1-9
 - with databases, 1-7
- communication stack, 1-7
- comparison operators, SQL (table), A-9
- Complete SQL Session VI
 - description, 3-2 to 3-3
 - maintaining database connections (example), 2-8
 - using, 2-1 to 2-2
- Configuration VIs, 7-1 to 7-5
 - Get Fetch Options, 7-4 to 7-5
 - Get Login Timeout, 7-1
 - Get Maximum Rows, 7-2
 - Get Query Timeout, 7-2
 - Get Statement Options, 7-3 to 7-4
 - Set Database, 7-4
 - Set Fetch Options, 7-4 to 7-5
 - Set Login Timeout, 7-1
 - Set Maximum Rows, 7-2
 - Set Query Timeout, 7-2
 - Set Statement Options, 7-2 to 7-3
- Connect VI
 - description, 3-1 to 3-2
 - example, 2-8
- connection with database
 - establishing, 1-7
 - maintaining during SQL session, 2-8
- constants, SQL (table), A-8
- CREATE TABLE command, SQL
 - description (table), 1-10, A-2
 - SQL-to-G functional correspondence (table), 1-12
 - variant implementations of, 1-11
- customer communication, *xiv*, B-1 to B-2

D

- Data Definition/Control Language (DDL/DCL) statements, 1-9
- Data Manipulation Language (DML) statements, 1-9
- Data Source Attributes dialog box (figure), Database Browser utility, 9-16
- data types for SQL Toolkit
 - codes and descriptions (table), 1-5 to 1-6
 - retrieving data directly, 2-12 to 2-13
- Database Browser utility, 9-15 to 9-23
 - Climate Query Results panel (figure), 9-23
 - Column Information dialog box (figure), 9-19
 - Data Source Attributes dialog box (figure), 9-16
 - Datatype Information dialog box (figure), 9-17
 - dialog box (figure), 9-15
 - Display References dialog box (figure), 9-20
 - Execute SQL dialog box (figure), 9-21
 - features, 2-6 to 2-7
 - Query Builder panel (figure), 9-22
 - Table Information dialog box (figure), 9-18
- databases
 - accessing with SQL Toolkit, 1-4
 - advantages and common features, 1-13
 - communicating with, 1-7
 - conceptual fit with SQL Toolkit, 1-11
 - engine-based, 1-7
 - establishing connections, 1-7
 - file-based, 1-7
 - functional integration with SQL Toolkit, 1-12
 - local and remote connections, 1-7
 - maintaining connections during SQL session, 2-8
 - models and examples, 1-4 to 1-6
 - transactions, 2-14
- Datatype Information dialog box (figure), Database Browser utility, 9-17
- Date Conversion VIs, 6-19 to 6-20
 - G Date to ODBC Date, 6-19 to 6-20
 - G Date to SQL Date, 6-19 to 6-20
 - SQL Date to G Date, 6-19 to 6-20
- date operators, SQL (table), A-9
- DELETE command, SQL
 - description (table), 1-10, A-2
 - SQL-to-G functional correspondence (table), 1-12
- Disconnect VI
 - description, 3-4
 - example, 2-8
- Display References dialog box (figure), Database Browser utility, 9-20
- documentation
 - conventions used in manual, *xii-xiii*
 - how to use SQL Toolkit documentation set, *xiii*
 - organization of manual, *xi-xii*
 - related documentation, *xiii-xiv*
- DROP TABLE command, SQL (table), A-2
- Dynamic SQL VIs, 6-4 to 6-8
 - Clear SQL Parameter, 6-5
 - Execute Prepared SQL, 2-11, 6-6
 - Get Number of Parameters, 6-6
 - optimizing SQL sessions, 2-10 to 2-11
 - overview, 6-4
 - Prepare SQL, 6-5
 - Set SQL Binary, 6-6 to 6-7
 - Set SQL Character, 6-6 to 6-7
 - Set SQL Date, 6-6 to 6-7
 - Set SQL Date-Time, 6-6 to 6-7
 - Set SQL Decimal Parameter, 6-7 to 6-8
 - Set SQL Double, 6-6 to 6-7
 - Set SQL Float, 6-6 to 6-7
 - Set SQL Integer, 6-6 to 6-7
 - Set SQL Long, 6-6 to 6-7
 - Set SQL Parameter to Null, 6-8
 - Set SQL Time, 6-6 to 6-7

E

e-mail support, B-2
 Easy SQL VI
 description, 3-3 to 3-4
 eliminating unnecessary data retrieval
 (example), 2-9
 maintain database connections (example),
 2-2
 efficient SQL operations. *See* SQL sessions,
 optimizing.
 electronic support services, B-1 to B-2
 employee record application example, 9-4
 End SQL VI, 6-3
 error-handling VIs, 4-1 to 4-3
 Abort-Continue Error Dialog, 4-2 to 4-3
 Abort Error Dialog, 4-1 to 4-2
 General Error Handler, 2-2
 overview, 4-1
 usage tips, 2-2 to 2-3
 examples. *See* application examples.
 Execute Prepared SQL VI
 description, 6-6
 optimizing SQL sessions, 2-11
 Execute SQL dialog box (figure), Database
 Browser utility, 9-21
 Execute SQL VI, 6-2

F

fax and telephone support, B-2
 Fax-on-Demand support, B-2
 Fetch Character Column Data VI, 6-12 to 6-13
 Fetch Column Data VI
 description, 6-12 to 6-13
 retrieving data directly into G data types,
 2-13
 Fetch Data Parsing VIs, 6-9 to 6-14
 Close Fetch Log File, 6-12
 Fetch Character Column Data,
 6-12 to 6-13
 Fetch Column Data, 2-13, 6-12 to 6-13
 Fetch Decimal Column Data,
 6-13 to 6-14

Fetch Double Column Data, 6-12 to 6-13
 Fetch Float Column Data, 6-12 to 6-13
 Fetch Integer Column Data, 6-12 to 6-13
 Fetch Long Column Data, 6-12 to 6-13
 Fetch Next Record, 6-10
 Fetch Previous Record, 6-10
 Fetch Query Results, 2-12 to 2-13,
 6-9 to 6-10
 Fetch Record N, 6-10
 Get Number of Columns, 6-11
 Get Number of Modified Records, 6-11
 Get Number of Records, 6-11
 Handle Null Value, 6-12
 retrieving data directly into G data types,
 2-12 to 2-13
 Fetch Decimal Column Data VI, 6-13 to 6-14
 Fetch Double Column Data VI, 6-12 to 6-13
 Fetch Float Column Data VI, 6-12 to 6-13
 Fetch Integer Column Data VI, 6-12 to 6-13
 Fetch Long Column Data VI, 6-12 to 6-13
 Fetch Next Record VI, 6-10
 Fetch Previous Record VI, 6-10
 Fetch Query Results VI
 description, 6-9 to 6-10
 retrieving data directly into G data types,
 2-12 to 2-13
 Fetch Record N VI, 6-10
 FOR UPDATE OF clause, SQL (table), A-7
 FROM clause, SQL (table), A-6
 FTP support services, B-1
 functions, SQL (table), A-11 to A-13

G

G Date to ODBC Date VI, 6-19 to 6-20
 G Date to SQL Date VI, 6-19 to 6-20
 G programming language, 1-2 to 1-3
 General Error Handler VI, 2-2
 Get Column Alias VI, 6-15
 Get Column Attributes VI, 6-16 to 6-17
 Get Column Expression VI, 6-15
 Get Column Information VI, 8-1 to 8-2
 Get Column Name VI, 6-15

Get Column Precision VI, 6-15
 Get Column Scale VI, 6-15
 Get Column Width VI, 6-15
 Get Database Column Type VI, 6-18
 Get Database Information VI, 8-2
 Get DSN Information VI, 8-2
 Get Fetch Options VI, 7-4 to 7-5
 Get ISO Level VI, 5-4
 Get Login Timeout VI, 7-1
 Get Maximum Rows VI, 7-2
 Get Number of Columns VI, 6-11
 Get Number of Modified Records VI, 6-11
 Get Number of Parameters VI, 6-6
 Get Number of Records VI, 6-11
 Get Procedure Col Information VI, 8-2 to 8-3
 Get Query Timeout VI, 7-2
 Get References VI, 8-3
 Get SQL Toolkit Column Type VI, 6-18
 Get Statement Options VI, 7-3 to 7-4
 Get Supported ISO Levels VI, 5-5
 Get Table Information VI, 8-3 to 8-4
 Get Type Information VI, 8-4
 GROUP BY clause, SQL (table), A-6

H

Handle Null Value VI, 6-12
 HAVING clause, SQL (table), A-7
 help for SQL Toolkit, online, 1-14
 high-level VIs, 3-1 to 3-4

- Complete SQL Session
 - description, 3-2 to 3-3
 - maintaining database connections (example), 2-8
 - using, 2-1 to 2-2
- Connect
 - description, 3-1 to 3-2
 - example, 2-8
- Disconnect
 - description, 3-4
 - example, 2-8

Easy SQL

- description, 3-3 to 3-4
- eliminating unnecessary data retrieval (example), 2-9
- maintaining database connections (example), 2-8

I

insert application example, 9-9 to 9-12
 INSERT command, SQL. *See also*

- Cluster to INSERT SQL VI.
 - description (table), 1-10
 - SQL-to-G functional correspondence (table), 1-12
- INSERT INTO command, SQL (table), A-2
- isolation levels, 5-3

L

locking transactions. *See* transaction locking VIs.
 logical operators, SQL (table), A-10

M

manual. *see* documentation.

N

network communications, 1-8 to 1-9
 numeric operators, SQL (table), A-8

O

object definitions, SQL (table), A-4 to A-5
 ODBC standard. *See* Open Database Connectivity (ODBC) standard.
 online help for SQL Toolkit, 1-14
 Open Database Connectivity (ODBC) standard, 1-6
 operators, SQL (table), A-8 to A-10

optimizing SQL sessions. *See* SQL sessions, optimizing.
 ORDER BY clause, SQL (table), A-7

P

performance considerations. *See* SQL sessions, optimizing.
 Prepare SQL VI, 6-5

Q

Query Builder panel (figure), Database Browser utility, 9-22
 query examples, 9-13 to 9-15
 Quick SQL example, 9-2 to 9-3

R

Request Database Information VI, 6-19
 Request DSN Information VI, 6-19
 Request Procedure Col Information VI, 6-19
 Request Table Information VI, 6-19
 Request Type Information VI, 6-19
 Request VI, 6-19
 Results to Cluster VI
 description, 4-5
 using, 2-5 to 2-6
 Rollback Transaction VI, 5-2

S

SELECT command, SQL. *See also*
 Cluster to Select SQL VI.
 description (table), 1-10, A-3
 SQL-to-G functional correspondence (table), 1-12
 SELECT statements, SQL, 1-9
 Set Database VI, 7-4
 Set Fetch Options VI, 7-4 to 7-5
 Set ISO Level VI, 5-5
 Set Login Timeout VI, 7-1
 Set Maximum Rows VI, 7-2

set operators, SQL (table), A-10
 Set Query Timeout VI, 7-2
 Set SQL Binary VI, 6-6 to 6-7
 Set SQL Character VI, 6-6 to 6-7
 Set SQL Date VI, 6-6 to 6-7
 Set SQL Date-Time VI, 6-6 to 6-7
 Set SQL Decimal Parameter VI, 6-7 to 6-8
 Set SQL Double VI, 6-6 to 6-7
 Set SQL Float VI, 6-6 to 6-7
 Set SQL Integer VI, 6-6 to 6-7
 Set SQL Long VI, 6-6 to 6-7
 Set SQL Parameter to Null VI, 6-8
 Set SQL Time VI, 6-6 to 6-7
 Set SQL VI, 6-3
 Set Statement Options VI, 7-2 to 7-3
 SQL (Structured Query Language). *See also* SQL quick-reference.
 classes of SQL statements, 1-9
 definition, 1-1, 1-9
 Dynamic SQL, 2-10 to 2-11
 example query results (table), 1-10
 frequently used commands (table), 1-10
 functional mapping between SQL statements and equivalent G operations (table), 1-12
 variants of SQL, 1-11
 SQL Date to G Date VI, 6-19 to 6-20
 SQL Information VIs, 8-1 to 8-4
 Get Column Information, 8-1 to 8-2
 Get Database Information, 8-2
 Get DSN Information, 8-2
 Get Procedure Col Information, 8-2 to 8-3
 Get References, 8-3
 Get Table Information, 8-3 to 8-4
 Get Type Information, 8-4
 SQL quick-reference, A-1 to A-13
 clauses (table), A-6 to A-7
 commands (table), A-2 to A-3
 frequently used commands (table), 1-10
 functions (table), A-11 to A-13
 object definitions (table), A-4 to A-5
 operators (table), A-8 to A-10

- SQL sessions, optimizing, 2-7 to 2-14
 - components of SQL sessions, 2-7
 - efficient SQL operations and databases, 2-14
 - eliminating unnecessary data retrieval, 2-9 to 2-10
 - maintaining database connections, 2-8
 - retrieving data directly into G data types, 2-12 to 2-13
 - speeding up execution using dynamic SQL, 2-10 to 2-11
- SQL-to-Cluster Template VIs, 4-3 to 4-5
 - Cluster to INSERT SQL
 - description, 4-4
 - using, 2-3 to 2-4
 - Cluster to SELECT SQL
 - description, 4-4 to 4-5
 - using, 2-4 to 2-5
 - overview, 4-3
 - Results to Cluster
 - description, 4-5
 - using, 2-5 to 2-6
 - usage tips, 2-3 to 2-6
- SQL Toolkit demo, 9-1
- SQL Toolkit for G. *See also* SQL VIs.
 - accessing databases, 1-4
 - advantages of databases, 1-13
 - cluster metaphor, 1-11 to 1-12
 - common SQL commands, 1-10
 - correspondence with database model, 1-11
 - data integration: conceptual fit, 1-11
 - data types (table), 1-5 to 1-6
 - database browser dialog, 2-6 to 2-7
 - database communication: local and remote connections, 1-7
 - database models, 1-4 to 1-6
 - examples (*See* application examples.)
 - features, 1-1 to 1-2
 - function integration: operational fit, 1-12
 - G programming language, 1-2 to 1-3
 - network communications, 1-8 to 1-9
 - ODBC standard, 1-6
 - online help, 1-14
 - operating environment, 1-2 to 1-13
 - SQL standard, 1-9 to 1-10
 - SQL variants, 1-11
- SQL VIs
 - Column Information VIs, 6-14 to 6-19
 - Get Column Alias, 6-15
 - Get Column Attributes, 6-16 to 6-17
 - Get Column Expression, 6-15
 - Get Column Name, 6-15
 - Get Column Precision, 6-15
 - Get Column Scale, 6-15
 - Get Column Width, 6-15
 - Get Database Column Type, 6-18
 - Get SQL Toolkit Column Type, 6-18
 - Request, 6-19
 - Request Database Information, 6-19
 - Request DSN Information, 6-19
 - Request Procedure Col Information, 6-19
 - Request Table Information, 6-19
 - Request Type Information, 6-19
 - Configuration VIs, 7-1 to 7-5
 - Get Fetch Options, 7-4 to 7-5
 - Get Login Timeout, 7-1
 - Get Maximum Rows, 7-2
 - Get Query Timeout, 7-2
 - Get Statement Options, 7-3 to 7-4
 - Set Database, 7-4
 - Set Fetch Options, 7-4 to 7-5
 - Set Login Timeout, 7-1
 - Set Maximum Rows, 7-2
 - Set Query Timeout, 7-2
 - Set Statement Options, 7-2 to 7-3
 - Date Conversion VIs, 6-19 to 6-20
 - G Date to ODBC Date, 6-19 to 6-20
 - G Date to SQL Date, 6-19 to 6-20
 - SQL Date to G Date, 6-19 to 6-20
 - Dynamic SQL VIs, 6-4 to 6-8
 - Clear SQL Parameter, 6-5
 - Execute Prepared SQL, 2-11, 6-6
 - Get Number of Parameters, 6-6

- optimizing SQL sessions,
 - 2-10 to 2-11
- Prepare SQL, 6-5
- Set SQL Binary, 6-6 to 6-7
- Set SQL Character, 6-6 to 6-7
- Set SQL Date, 6-6 to 6-7
- Set SQL Date-Time, 6-6 to 6-7
- Set SQL Decimal Parameter,
 - 6-7 to 6-8
- Set SQL Double, 6-6 to 6-7
- Set SQL Float, 6-6 to 6-7
- Set SQL Integer, 6-6 to 6-7
- Set SQL Long, 6-6 to 6-7
- Set SQL Parameter to Null, 6-8
- Set SQL Time, 6-6 to 6-7
- error-handling VIs, 4-1 to 4-3
 - Abort-Continue Error Dialog,
 - 4-2 to 4-3
 - Abort Error Dialog, 4-1 to 4-2
 - General Error Handler, 2-2
 - usage tips, 2-2 to 2-3
- examples (*See* application examples)
- Fetch Data Parsing VIs, 6-9 to 6-14
 - Close Fetch Log File, 6-12
 - Fetch Character Column Data,
 - 6-12 to 6-13
 - Fetch Column Data, 2-13,
 - 6-12 to 6-13
 - Fetch Decimal Column Data,
 - 6-13 to 6-14
 - Fetch Double Column Data,
 - 6-12 to 6-13
 - Fetch Float Column Data,
 - 6-12 to 6-13
 - Fetch Integer Column Data,
 - 6-12 to 6-13
 - Fetch Long Column Data,
 - 6-12 to 6-13
 - Fetch Next Record, 6-10
 - Fetch Previous Record, 6-10
 - Fetch Query Results, 2-12 to 2-13,
 - 6-9 to 6-10
 - Fetch Record N, 6-10
 - Fetch Number of Columns, 6-11
 - Fetch Number of Modified Records,
 - 6-11
 - Fetch Number of Records, 6-11
 - Handle Null Value, 6-12
 - retrieving data directly into G data
 - types, 2-12 to 2-13
- high-level VIs, 3-1 to 3-4
 - Complete SQL Session, 2-1 to 2-2,
 - 2-8, 3-2 to 3-3
 - Connect, 2-8, 3-1 to 3-2
 - Disconnect, 2-8, 3-4
 - Easy SQL, 2-8, 2-9, 3-3 to 3-4
- SQL Information VIs, 8-1 to 8-4
 - Get Column Information, 8-1 to 8-2
 - Get Database Information, 8-2
 - Get DSN Information, 8-2
 - Get Procedure Col Information,
 - 8-2 to 8-3
 - Get References, 8-3
 - Get Table Information, 8-3 to 8-4
 - Get Type Information, 8-4
- SQL-to-Cluster Template VIs, 4-3 to 4-5
 - Cluster to INSERT SQL, 2-3 to 2-4,
 - 4-4
 - Cluster to SELECT SQL, 2-4 to 2-5,
 - 4-4 to 4-5
 - Results to Cluster, 2-5 to 2-6, 4-5
 - usage tips, 2-3 to 2-6
- Statement Execution VIs, 6-2 to 6-3
 - Append SQL, 6-3
 - End SQL, 6-3
 - Execute SQL, 6-2
 - Set SQL, 6-3
- transaction locking VIs, 5-3 to 5-5
 - Get ISO Level, 5-4
 - Get Supported ISO Levels, 5-5
 - Set ISO Level, 5-5
- transaction VIs, 5-1 to 5-5
 - Commit Transaction, 5-1 to 5-2
 - Rollback Transaction, 5-2
 - Start Transaction, 5-2

- transaction management, 2-14
 - usage tips, 2-15
- Start Transaction VI, 5-2
- Statement Execution VIs, 6-2 to 6-3
 - Append SQL, 6-3
 - End SQL, 6-3
 - Execute SQL, 6-2
 - Set SQL, 6-3
- Structured Query Language. *See* SQL (Structured Query Language).

T

- Table Information dialog box (figure), Database Browser utility, 9-18
- tables
 - concept of tables (figure), 1-5
 - definition, 1-4
- telephone and fax support, B-2
- transaction locking VIs, 5-3 to 5-5
 - Get ISO Level, 5-4
 - Get Supported ISO Levels, 5-5
 - isolation levels, 5-3
 - overview, 5-3
 - Set ISO Level, 5-5
- transaction VIs, 5-1 to 5-5
 - application example, 9-3 to 9-4
 - Commit Transaction, 5-1 to 5-2
 - overview, 5-1
 - Rollback Transaction, 5-2
 - Start Transaction, 5-2
 - transaction management, 2-14
 - usage tips, 2-15

U

- UPDATE command, SQL
 - description (table), 1-10, A-3
 - SQL-to-G functional correspondence (table), 1-12

V

- virtual instruments (VIs)
 - definition, 1-2
 - examples (figure), 1-3
- VIs (virtual instruments). *See* SQL VIs; virtual instruments (VIs).

W

- weather application examples, 9-5 to 9-8
- WHERE clause, SQL (table), A-6