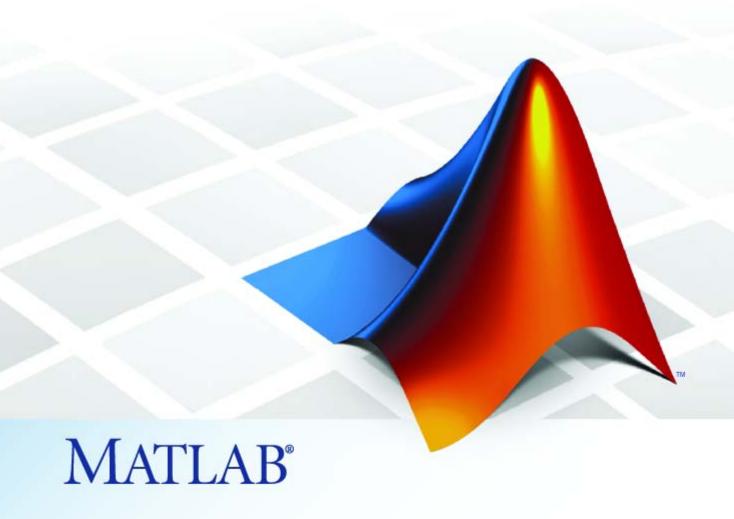
Database Toolbox™ 3 User's Guide





How to Contact The MathWorks



www.mathworks.com

comp.soft-sys.matlab

www.mathworks.com/contact TS.html Technical Support

Web

Newsgroup



suggest@mathworks.com bugs@mathworks.com

doc@mathworks.com

service@mathworks.com info@mathworks.com

Product enhancement suggestions

Bug reports

Documentation error reports

Order status, license renewals, passcodes Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Database ToolboxTM User's Guide

© COPYRIGHT 1998–2008 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

May 1998 Online Only July 1998 First Printing Online only June 1999 December 1999 Second printing Online only September 2000 Third printing June 2001 July 2002 Online only November 2002 Fourth printing June 2004 Fifth printing October 2004 Online only March 2005 Online only September 2005 Online only March 2006 Online only September 2006 Online only October 2006 Sixth printing March 2007 Online only September 2007 Seventh printing March 2008 Online only October 2008 Online only

New for Version 1 for MATLAB® 5.2 For Version 1 Revised for Version 2 (Release 11) For Version 2 (Release 11) Revised for Version 2.1 (Release 12) Revised for Version 2.2 (Release 12.1) Revised for Version 2.2.1 (Release 13) Version 2.2.1 Revised for Version 3.0 (Release 14) Revised for Version 3.1 (Release 14SP1) Revised for Version 3.0.2 (Release 14SP2) Revised for Version 3.1 (Release 14SP3) Revised for Version 3.1.1 (Release 2006a) Revised for Version 3.2 (Release 2006b) Revised for Version 3.2 (Release 2006b) Revised for Version 3.3 (Release 2007a) Revised for Version 3.4 (Release 2007b) Revised for Version 3.4.1 (Release 2008a) Revised for Version 3.5 (Release 2008b)

Contents

Before	You	Begin
--------	-----	-------

	Delote fou De	giii
1		
]	Product Overview	1-2
1	Working with Databases	1-3
	Connecting to Databases	1-3
	Supported Platforms	1-3
	Supported Databases	1-3
	Supported Drivers	1-4
	Structured Query Language (SQL)	1-4
S	Supported Data Types	1-5
1	Data Retrieval Restrictions	1-7
_	Spaces in Table Names or Column Names	1-7
	Quotation Marks in Table Names or Column Names	1-7
	Reserved Words in Column Names	1-7
	Working with Data Sour	ces
<u>'</u>		
\$	Setting up ODBC Data Sources	2-2
Ş	Setting up JDBC Data Sources	2-3
A	Accessing Existing JDBC Data Sources	2-4
ľ	Modifying Existing JDBC Data Sources	2-5
1	Removing JDBC Data Sources	2-6

	Troubleshooting JDBC Driver Problems	2-1
	Database Toolbox Functions vs. Visual Qu Bui	ıery lder
3		
	When to Use Visual Query Builder Tasks You Can Perform Using Visual Query Builder Limitations of Visual Query Builder	3-2 3-2 3-2
	When to Use Database Toolbox Functions	3-3
a	Using Visual Query Bui	lder
4		
	Getting Started with Visual Query Builder What Is Visual Query Builder? Using Queries to Import Data Using Queries to Export Data	4-2 4-2 4-2 4-4
	Working with Preferences Specifying Preferences Saving Preferences	4-6 4-6 4-9
	Displaying Query Results How to Display Query Results Displaying Data Relationally Charting Query Results Displaying Query Results in an HTML Report Using the MATLAB® Report Generator Software to Customize Display of Query Results	4-10 4-10 4-10 4-14 4-16
	Fine-Tuning Queries Using Advanced Query Options	4-22
	Data	4-22

	Retrieving Data That Meets Specified Criteria	4-24 4-27
	Grouping Statements	
	Displaying Results in a Specified Order	4-31
	Using Having Clauses To Refine Group By Results	4-34
	Creating Subqueries for Values from Multiple Tables Creating Queries That Include Results from Multiple	4-37
	Tables	4-42
	Additional Advanced Query Options	4-45
	Retrieving BINARY and OTHER Sun Java Data	
	Types	4-46
	Importing and Exporting BOOLEAN Data	4-48
	Importing BOOLEAN Data from Databases to the MATLAB	
	Workspace	4-48
	Exporting BOOLEAN Data from the MATLAB Workspace	4 - 4
	to Databases	4-51
	Saving Queries in M-Files	4-52
	About Generated M-Files	4-52
	VQB Query Elements in M-Files	4-53
- [Using Database Toolbox Funct	ions
5 [Using Database Toolbox Functi	ions
5 [Using Database Toolbox Functions Getting Started with Database Toolbox Functions	ions 5-2
5	Getting Started with Database Toolbox Functions	
5 [Getting Started with Database Toolbox Functions Importing Data from Databases into the MATLAB	
5	Getting Started with Database Toolbox Functions	5-2
5	Getting Started with Database Toolbox Functions Importing Data from Databases into the MATLAB	5-2
5	Getting Started with Database Toolbox Functions Importing Data from Databases into the MATLAB Workspace	5-2 5-3
5	Getting Started with Database Toolbox Functions Importing Data from Databases into the MATLAB Workspace	5-2 5-3 5-9
5	Getting Started with Database Toolbox Functions Importing Data from Databases into the MATLAB Workspace	5-2 5-3
5	Getting Started with Database Toolbox Functions Importing Data from Databases into the MATLAB Workspace	5-2 5-3 5-9
5 [Getting Started with Database Toolbox Functions Importing Data from Databases into the MATLAB Workspace	5-2 5-3 5-9

Retrieving BINARY or OTHER Su		
Types	• • • • • • • • • • • • • • • • • • • •	
Working with Database Metadata		
_		
Resultset Metadata Objects		
Using Driver Functions		
About Objects and Methods in the	e Database Toolbox	
Software		
	Function Refere	eı
Utilities		
Utilities Database Connection SQL Cursor		
Database Connection		

	et Metadata Object	6
Visual Q	uery Builder	6
	Functions — Alphabetical	Li
	Examp	ol
Visual Q	uery Builder GUI: Importing Data	A
Visual Q	uery Builder GUI: Displaying Results	A
	uery Builder GUI: Advanced Query Options	A
Visual Q		
_	uery Builder GUI: Exporting Data	A

Before You Begin

- ullet "Product Overview" on page 1-2
- "Working with Databases" on page 1-3
- "Supported Data Types" on page 1-5
- "Data Retrieval Restrictions" on page 1-7

Product Overview

Database ToolboxTM software is one of an extensive collection of MATLAB® toolboxes. This toolbox enables you to use MATLAB functions to import and export data between the MATLAB workspace and relational databases. With this toolbox, you can bring data from a database into the MATLAB workspace, use MATLAB's computational and analytic tools to work with the data, and then store the results back in the database or in another database.

For example, a financial analyst working on a mutual fund can import a company's financial data into the MATLAB workspace, run selected analyses, and store the results for future tracking. The analyst can then export the saved results to a database.

The Database Toolbox product includes Visual Query Builder (VQB), an easy-to-use graphical user interface (GUI) for exchanging data with your database. You can use VQB instead of or in addition to Database Toolbox functions.

Note This toolbox is not intended for use as a tool for database administration. Use your database system application to perform database administrative tasks, such as creating tables.

Working with Databases

In this section...

"Connecting to Databases" on page 1-3

"Supported Platforms" on page 1-3

"Supported Databases" on page 1-3

"Supported Drivers" on page 1-4

"Structured Query Language (SQL)" on page 1-4

Connecting to Databases

Before you can use this toolbox to connect to a database, you must set up data sources. For more information, see "Configuring Your Environment" in the *Database Toolbox Getting Started Guide*.

Supported Platforms

This toolbox runs on all platforms that the MATLAB software supports.

For more information, see Database Toolbox system requirements at http://www.mathworks.com/products/database/requirements.html.

Note This toolbox does not support running MATLAB software sessions with the -nojvm startup option enabled on UNIX® platforms. (UNIX is a registered trademark of the Open Group in the United States and other countries.)

Supported Databases

This toolbox supports importing and exporting data from any ODBC- and/or JDBC-compliant database management system, including:

- IBM DB2®
- IBM® Informix®
- Ingres®
- Microsoft® Access™
- Microsoft® Excel®
- Microsoft® SQL ServerTM
- MySQL®
- Oracle®
- Postgre SQL
- Sybase[®] SQL Anywhere[®]
- Sybase SQL Server®

If you are upgrading an earlier version of a database, you need not do anything special for this toolbox. Simply configure the data sources for the new version of the database application as you did for the original version.

Supported Drivers

This toolbox requires a database driver. Typically, you install a driver when you install a database. For instructions about how to install a database driver, consult your database administrator.

On Microsoft[®] Windows[®] platforms, the toolbox supports Open Database Connectivity (ODBC) drivers and Sun[™] Java[™] Database Connectivity (JDBC) drivers.

On UNIX platforms, the toolbox supports Java Database Connectivity (JDBC) drivers. If your database does not ship with JDBC drivers, download drivers from the Sun JDBC Web Site at http://industry.java.sun.com/products/jdbc/drivers.

Structured Query Language (SQL)

This toolbox supports American National Standards Institute (ANSI) standard SQL commands.

Supported Data Types

You can import the following data types into the MATLAB workspace and export them back to your database:

- BOOLEAN
- CHAR
- DATE
- DECIMAL
- DOUBLE
- FLOAT
- INTEGER
- LONGCHAR
- NUMERIC
- REAL
- SMALLINT
- TIME
- TIMESTAMP
- TINYINT

Note The Database Toolbox software interprets this data type as BOOLEAN and imports it into the MATLAB workspace as logical true (1) or false (0). For more information about how the Database Toolbox software handles BOOLEAN data, see "Importing and Exporting BOOLEAN Data" on page 4-48.

- VARCHAR
- NTEXT

You can import data of types not included in this list into the MATLAB workspace. However, you may need to manipulate such data before you can process it in MATLAB.

Data Retrieval Restrictions

In this section...

"Spaces in Table Names or Column Names" on page 1-7

"Quotation Marks in Table Names or Column Names" on page 1-7

"Reserved Words in Column Names" on page 1-7

Spaces in Table Names or Column Names

Microsoft Access supports the use of spaces in table and column names, but most other databases do not. Queries that retrieve data from tables and fields whose names contain spaces require delimiters around table names and field names. In AccessTM, enclose the table names or field names in quotation marks, for example, "order id". Other databases use different delimiters, such as brackets, []. In Visual Query Builder, table names and field names that include spaces appear in quotation marks.

Quotation Marks in Table Names or Column Names

Do not include quotation marks in table names or column names. The Database Toolbox software does not support data retrieval from table and column names that contain quotation marks.

Reserved Words in Column Names

You cannot use the Database Toolbox software to import or export data in columns whose names contain database reserved words, such as DATE or TABLE.

Working with Data Sources

- "Setting up ODBC Data Sources" on page 2-2
- "Setting up JDBC Data Sources" on page 2-3
- "Accessing Existing JDBC Data Sources" on page 2-4
- "Modifying Existing JDBC Data Sources" on page 2-5
- "Removing JDBC Data Sources" on page 2-6
- "Troubleshooting JDBC Driver Problems" on page 2-7

Setting up ODBC Data Sources

For instructions on setting up ODBC data sources, see "Setting Up Data Sources for Use with ODBC Drivers" in the Database ToolboxGetting Started Guide.

Setting up JDBC Data Sources

For instructions on setting up JDBC data sources, see "Setting Up Data Sources for Use with JDBC Drivers" in the *Database ToolboxGetting Started Guide*.

Accessing Existing JDBC Data Sources

To access an existing data source from Visual Query Builder in future MATLAB software sessions:

- 1 In Visual Query Builder, select Query > Define JDBC data source.
- 2 In the Define JDBC data sources dialog box, click Use Existing File.
- **3** In the Specify Existing JDBC data source MAT-file dialog box, select the MAT-file that contains the data sources you want to use and click **Open**.
 - The data sources in the selected MAT-file appear in the Define JDBC data sources dialog box.
- **4** Click **OK** to close the Define JDBC data sources dialog box. The data sources now appear in the Visual Query Builder **Data source** list.

Modifying Existing JDBC Data Sources

- 1 Access the existing data source as described in "Accessing Existing JDBC Data Sources" on page 2-4.
- 2 Select the data source in the Define JDBC Data Sources dialog box.
- 3 Modify the data in the Driver and URL fields.
- 4 Click Add/Update.
- **5** Click **OK** to save your changes and close the Define JDBC data sources dialog box.

Removing JDBC Data Sources

- 1 Access the existing data source as described in "Accessing Existing JDBC Data Sources" on page 2-4.
- 2 Click Remove.
- $\boldsymbol{3}$ Click \boldsymbol{OK} to save your changes and close the Define JDBC data sources dialog box.

Troubleshooting JDBC Driver Problems

This section describes how to address common data source access problems, in which selecting a data source in the Visual Query Builder list produces an error, or the data source is not in the list as expected. There are several potential causes for these issues:

- The database is unavailable, or there are connectivity problems. Try reselecting the data source in VQB. If you are still unable to access the data source, contact your database administrator.
- You ran the clear all command in the MATLAB Command Window after you defined a JDBC data source. In this case, redefine the data source by following the instructions in "Setting Up Data Sources for Use with JDBC Drivers" in the *Database Toolbox Getting Started Guide*.

Database Toolbox Functions vs. Visual Query Builder

- "When to Use Visual Query Builder" on page 3-2
- "When to Use Database Toolbox Functions" on page 3-3

When to Use Visual Query Builder

In this section...

"Tasks You Can Perform Using Visual Query Builder" on page 3-2

"Limitations of Visual Query Builder" on page 3-2

Tasks You Can Perform Using Visual Query Builder

You can use Visual Query Builder to:

- Import data from relational databases into the MATLAB workspace by selecting information from lists to build queries.
- Display retrieved information in relational tables, reports, and charts.
- Export data from the MATLAB workspace into new records in a database.
- Easily build SQL queries and exchange data between databases and the MATLAB workspace.
- View and edit SQL statements for queries generated with VQB.
- Automatically generate a MATLAB M-file that consists of Database Toolbox functions that perform queries you built using VQB.

Limitations of Visual Query Builder

- You cannot use Visual Query Builder to replace existing data in a database with data from the MATLAB workspace. Use the update function instead.
- You cannot use Visual Query Builder to export binary data. Instead, use the fastinsert function.

When to Use Database Toolbox Functions

Database Toolbox functions can do everything that Visual Query Builder can, and more. You can use these functions to:

- Replace existing records in databases with data from the MATLAB workspace.
- Retrieve large data sets or partial data sets in a single fetch command, or in discrete amounts using multiple fetches.
- Dynamically import data into the MATLAB workspace.
- Modify SQL queries in MATLAB statements.
- Write MATLAB M-files and applications that access databases.
- Perform other functions that are not available with Visual Query Builder, including:
 - Exporting binary data or other data types that you can import into the MATLAB workspace, but cannot export from the MATLAB workspace using VQB.
 - Accessing database metadata.

Using Visual Query Builder

- "Getting Started with Visual Query Builder" on page 4-2
- "Working with Preferences" on page 4-6
- "Displaying Query Results" on page 4-10
- "Fine-Tuning Queries Using Advanced Query Options" on page 4-22
- "Retrieving BINARY and OTHER Sun Java Data Types" on page 4-46
- "Importing and Exporting BOOLEAN Data" on page 4-48
- "Saving Queries in M-Files" on page 4-52

Getting Started with Visual Query Builder

In this section...

"What Is Visual Query Builder?" on page 4-2

"Using Queries to Import Data" on page 4-2

"Using Queries to Export Data" on page 4-4

What Is Visual Query Builder?

Visual Query Builder (VQB) is an easy-to-use graphical user interface (GUI) for exchanging data with your database. You can use VQB to:

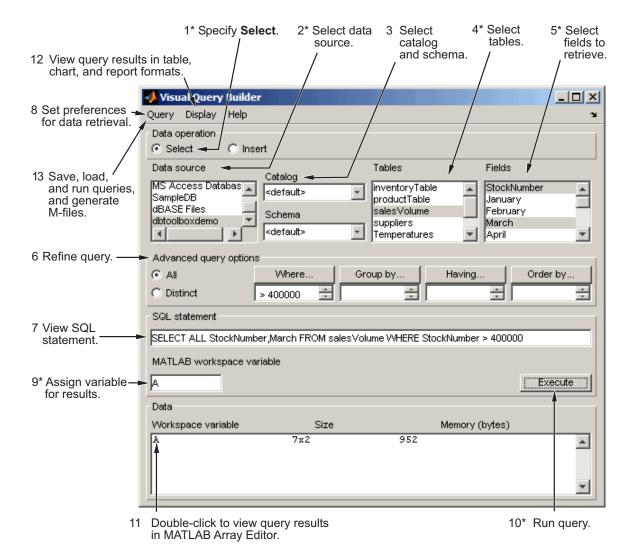
- Build queries to retrieve data by selecting information from lists instead of using MATLAB functions.
- Store data retrieved from a database in a MATLAB cell array, structure, or numeric matrix.
- Process the retrieved data using the MATLAB suite of functions.
- Display retrieved information in relational tables, reports, and charts.
- Export data from the MATLAB workspace into new rows in a database.

Using Queries to Import Data

The following steps summarize how to use VQB to import data.

To start the Visual Query Builder, type querybuilder at the MATLAB prompt.

^{*}Required step



For a step-by-step example of how to use queries to import data into the MATLAB workspace from a database, see "Using Queries to Import Database

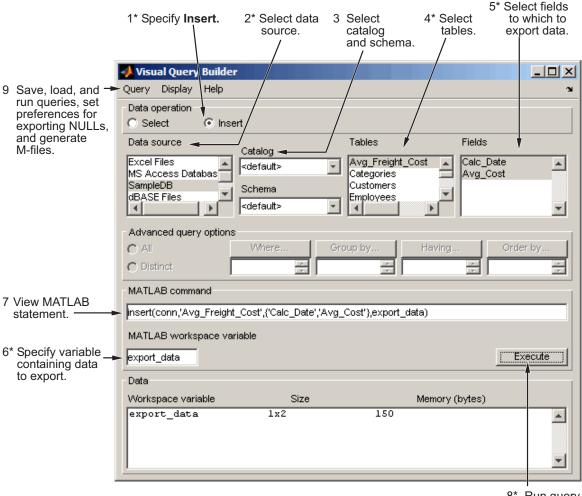
Data into the MATLAB Workspace" in the Database Toolbox Getting Started Guide.

Using Queries to Export Data

The following steps summarize how to use VQB to export data.

To start the Visual Query Builder, type querybuilder at the MATLAB prompt.

*Required step



8* Run query.

For a step-by-step example of how to use queries to export data from the MATLAB workspace to a database, see "Using Queries to Export MATLAB Workspace Data to a Database" in the Database Toolbox Getting Started Guide.

Working with Preferences

In this section...

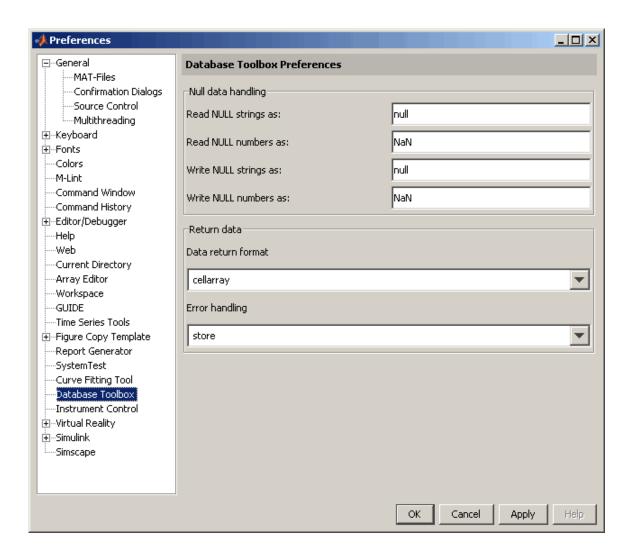
"Specifying Preferences" on page 4-6

"Saving Preferences" on page 4-9

Specifying Preferences

This section describes how to set VQB preferences to specify:

- How NULL data in a database is represented after you import it into the MATLAB workspace
- The format of data retrieved from databases
- The method of error notification
- 1 Click **Query > Preferences**. The Preferences dialog box appears.



2 Specify the Preferences settings as described in the following table.

Preference	Value	Description
Read NULL numbers as	0	If you accept the default value for this field, NULL data imported from databases into the MATLAB workspace appears as NaN. Setting this field to 0 causes NULL data imported into the MATLAB workspace to appear as 0s.
Data return format	numeric	Select a data format based on the type of data you are importing, memory considerations, and your preferred method of working with retrieved data.
		Cell arrays and structures support mixed data types, but require more memory and process more slowly than numeric matrices. Select numeric if:
		The data you are retrieving is numeric, or
		• You need to convert nonnumeric data to the format specified in the Read NULL numbers as field.
Error handling	report	• Set this field to store or empty to direct errors to a dialog box rather than to the MATLAB Command Window.
		• Set this field to report to display query errors in the MATLAB Command Window.

- 3 Click OK.
- **4** Assign the query results to a workspace variable, A.
- **5** Click **Execute** to rerun the query.

Information about the retrieved data appears in the **Data** area.

6 To see the results, enter A in the Command Window.

A =

125970	1400	1100	981
212569	2400	1721	1414
389123	1800	1200	890
400314	3000	2400	1800
400339	4300	0	2600
400345	5000	3500	2800
400455	1200	900	800
400876	3000	2400	1500
400999	3000	1500	1000
888652	0	900	821

NULL values appear as 0s instead of NaNs.

For more information about Preferences, see the setdbprefs function reference page.

Saving Preferences

Preferences apply only to the current MATLAB software session. They are not saved with queries. Default Preferences apply when you start a new session, or after you clear all variables (using, for example, the clear all command). It is a good practice to check Preferences settings before you run queries.

Displaying Query Results

In this section...

"How to Display Query Results" on page 4-10

"Displaying Data Relationally" on page 4-10

"Charting Query Results" on page 4-14

"Displaying Query Results in an HTML Report" on page 4-16

"Using the MATLAB® Report Generator Software to Customize Display of Query Results" on page 4-17

How to Display Query Results

To display query results, perform one of the following actions:

- Enter the variable name to which to assign the query results in the MATLAB Command Window.
- Double-click the variable in the VQB Data area to view the data in the Variable Editor.

The examples in this section use the saved query basic.gry. To load and configure this query:

- 1 Click Query > Preferences, and set Read NULL numbers as to 0.
- 2 Click Query > Load.
- 3 In the Load SQL Statement dialog box, select basic.gry from the File name field and click Open.
- **4** In VQB, enter a value for the **MATLAB workspace variable**, for example, A, and click Execute.

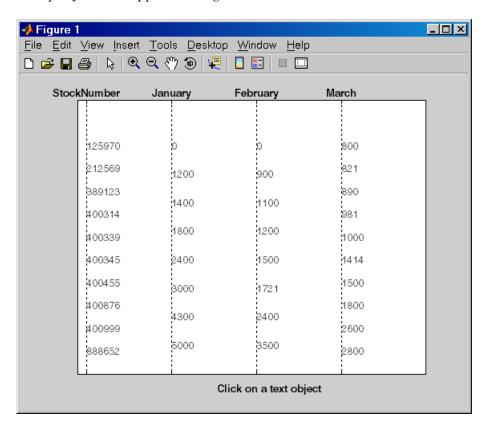
Displaying Data Relationally

To display the results of basic.gry:

1 Execute basic.gry.

2 Click Display > Data.

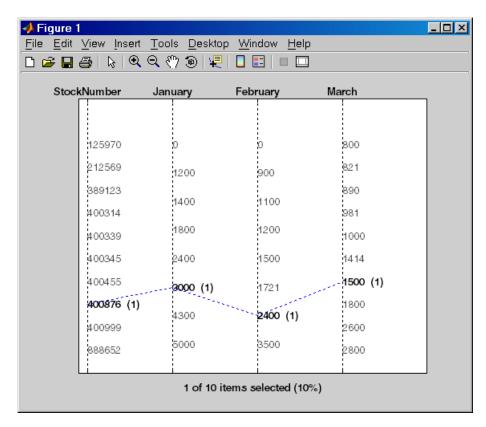
The query results appear in a figure window.



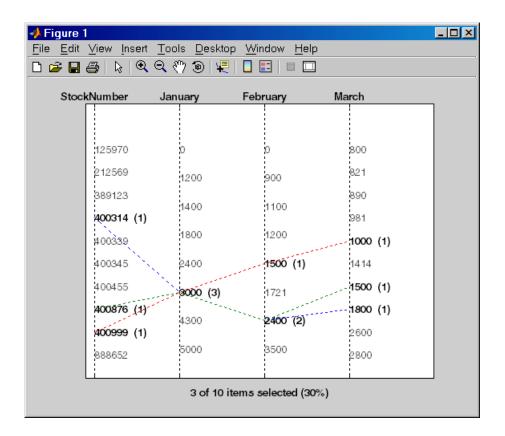
This display shows only unique values for each field, so you should not read each row as a single record. In this example, there are 10 entries for **StockNumber**, 8 entries for **January** and **February**, and 10 entries for **March**. The number of entries in each field corresponds to the number of unique values in the field.

3 Click a value in the figure window, for example, **StockNumber** 400876, to see its associated values.

The data associated with the selected value appears in bold font and is connected with a dotted line. The data shows that sales for item 400876 are 3000 in January, 2400 in February, and 1500 in March.



4 As another example, click 3000 under January. It shows three different items with sales of 3000 units in January: 400314, 400876, and 400999.

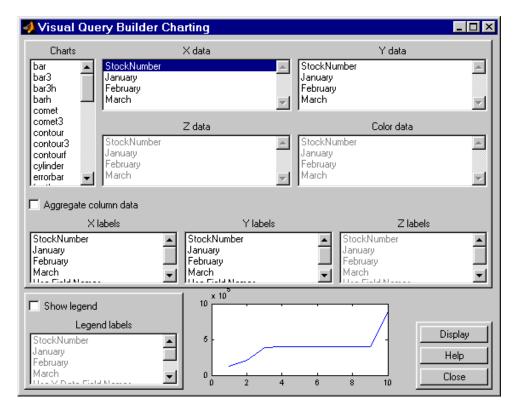


Charting Query Results

To chart the results of basic.qry:

1 Click **Display > Chart**.

The Visual Query Builder Charting dialog box appears.



2 Select a type of chart from the **Charts** list. In this example, choose a pie chart by specifying pie.

A preview of the pie chart, with each stock item displayed in a different color, appears at the bottom of the dialog box.

3 Select the data to display in the chart from the X data, Y data, and Z data list boxes. In this example, select March from the X data list box to display a pie chart of March data.

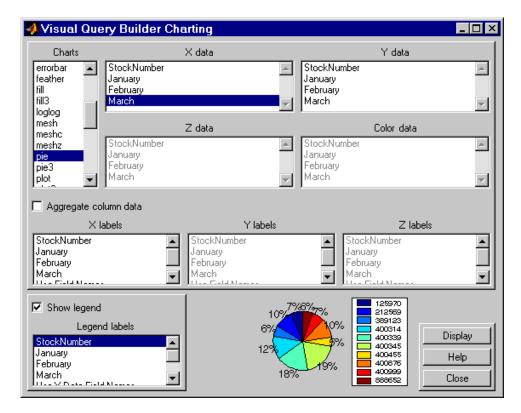
The pie chart preview now shows percentages for March data.

4 To display a legend, which maps colors to the stock numbers, select the **Show legend** check box.

The **Legend labels** field becomes active.

5 Select StockNumber from the Legend labels list box.

A legend appears in the chart preview. Drag and move the legend in the preview as needed.

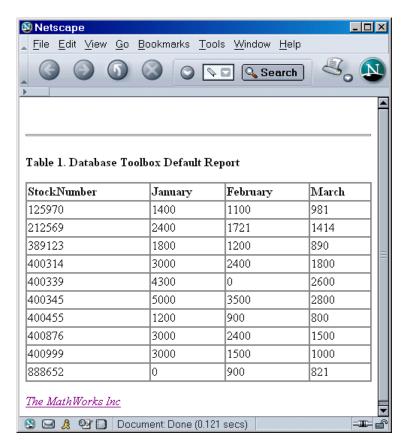


6 Click **Close** to close the Charting dialog box.

Displaying Query Results in an HTML Report

To display results for basic.gry in an HTML report, click **Display > Report**.

The query results appear as a table in a Web browser. Each row represents a record from the database. In this example, sales for item 400876 are 3000 in January, 2400 in February, and 1500 in March.

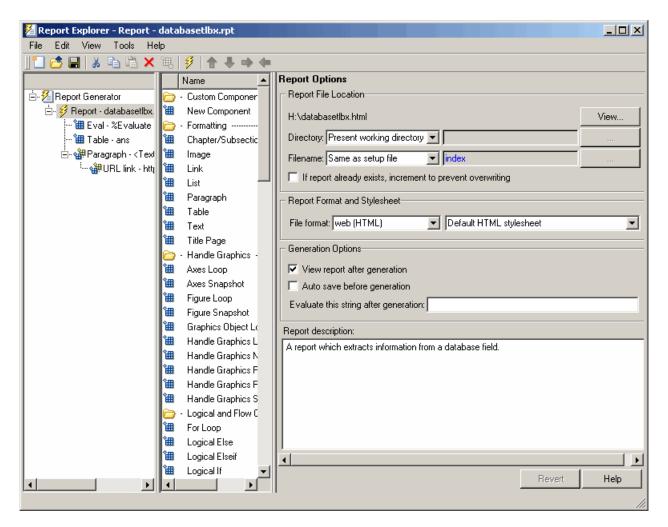


Note Because some browsers do not start automatically, you may need to open your Web browser before displaying the query results.

Using the MATLAB Report Generator Software to Customize Display of Query Results

To use the MATLAB® Report Generator TM software to customize the display of the results of basic.qry:

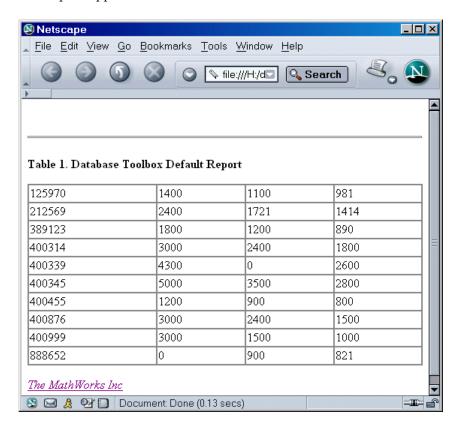
- 1 Click Display > Report Generator.
- 2 The Report Explorer opens, listing sample report templates that you can use to create custom reports. Select the template <code>matlabroot/toolbox/database/vqb/databasetlbx.rpt</code> from the Options pane in the middle of the Report Explorer window.



- 3 Open the report template for editing by clicking Open a Report file or stylesheet.
 - a In the Outline pane on the left, under Report Generator > databasetlbx.rpt, select Table.
 - **b** In the Properties pane on the right, do the following:

- i In Table Content > Workspace Variable Name, enter the name of the variable to which you assigned the query results in VQB, for example, 'A'.
- ii Under Header/Footer Options, set Number of header rows to 0.
- c Click Apply.
- **4** Click **File > Report** to run the report.

The report appears in a Web browser.



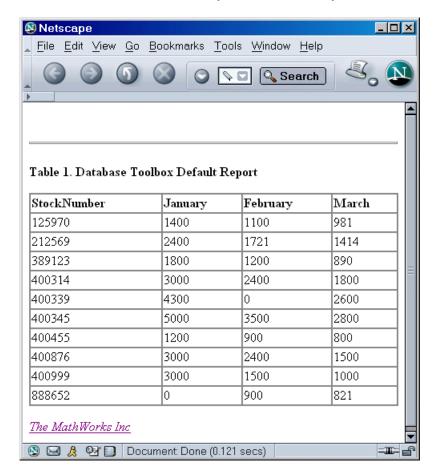
5 Field names do not automatically display as column headers in the report. To display the field names:

a Modify the workspace variable A as follows:

```
A = [{'Stock Number', 'January', 'February', 'March'};A]
```

b In the MATLAB Report Generator properties pane, change **Number of header rows** to 1 and regenerate the report. The report now displays field names as headings.

Each row represents a record from the database. For example, sales for item 400876 are 3000 in January, 2400 in February, and 1500 in March.



For more information about the MATLAB Report Generator product, click the **Help** button in the Report Explorer or see the MATLAB Report Generator documentation.

Note Because some browsers are not configured to launch automatically, you may need to open your Web browser before displaying the report.

Fine-Tuning Queries Using Advanced Query Options

In this section...

"Retrieving All Occurrences vs. Unique Occurrences of Data" on page 4-22

"Retrieving Data That Meets Specified Criteria" on page 4-24

"Grouping Statements" on page 4-27

"Displaying Results in a Specified Order" on page 4-31

"Using Having Clauses To Refine Group By Results" on page 4-34

"Creating Subqueries for Values from Multiple Tables" on page 4-37

"Creating Queries That Include Results from Multiple Tables" on page 4-42

"Additional Advanced Query Options" on page 4-45

Note For more information about advanced query options, select **Help** in any of the dialog boxes for the options.

Retrieving All Occurrences vs. Unique Occurrences of Data

To use the dbtoolboxdemo data source to demonstrate how to retrieve all versus distinct occurrences of data:

- 1 Set the **Data return format** preference to cellarray.
- 2 Set Read NULL numbers as to NaN.
- 3 In Data operation, choose Select.
- 4 In Data source, select dbtoolboxdemo.

Do not specify **Catalog** or **Schema**.

- **5** In **Tables**, select SalesVolume.
- **6** In **Fields**, select January.

- **7** To retrieve all occurrences of January:
 - a In Advanced query options, select All.
 - **b** Assign the query results to the **MATLAB** workspace variable All.
 - **c** Click **Execute** to run the query.
- **8** To retrieve only unique occurrences of data:
 - a In Advanced query options, select Distinct.
 - **b** Assign the query results to a MATLAB workspace variable Distinct.
 - **c** Click **Execute** to run the query.
- **9** In the MATLAB Command Window, enter All, Distinct to display the query results:

```
[1400]
    [2400]
    [1800]
    [3000]
    [4300]
    [5000]
    [1200]
    [3000]
    [3000]
    [ NaN]
Distinct =
    [ NaN]
    [1200]
    [1400]
    [1800]
    [2400]
    [3000]
    [4300]
    [5000]
```

A11 =

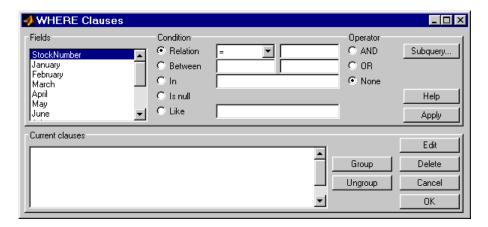
The value 3000 appears three times in All, but appears only once in Distinct.

Retrieving Data That Meets Specified Criteria

Use basic .qry and the Where field in Advanced query options to retrieve stock numbers greater than 400000 and less than 500000:

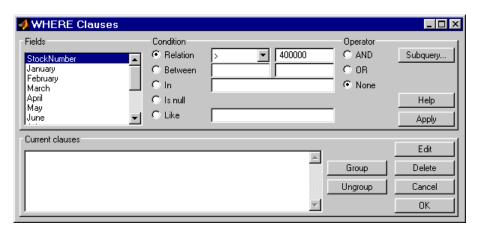
- 1 Load basic.gry.
- **2** Set the **Data return format** preference to cellarray.
- 3 Set Read NULL numbers as to NaN.
- 4 In Advanced query options, click Where.

The WHERE Clauses dialog box appears.



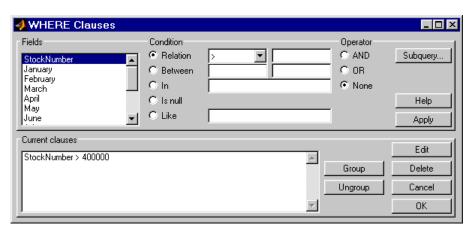
- 5 In Fields, select the field whose values you want to restrict, StockNumber.
- **6** In Condition, specify that StockNumber must be greater than 400000.
 - a Select Relation.
 - **b** In the drop-down list to the right of **Relation**, select >.
 - **c** In the field to the right of the drop-down list, enter 400000.

The WHERE Clauses dialog box now looks as follows.



d Click Apply.

The clause that you defined, StockNumber > 400000, appears in the Current clauses area.



- 7 Add the condition that StockNumber must also be less than 500000.
 - **a** In Current clauses, select StockNumber > 400000.
 - **b** In Current clauses, click Edit or double-click the StockNumber entry.
 - c For Operator, select AND.
 - d Click Apply.

The Current clauses field now displays:

```
StockNumber > 400000 AND
```

- e In Fields, select StockNumber.
- f In Condition, select Relation.
- **g** In the drop-down list to the right of **Relation**, select <.
- **h** In the field to the right of the drop-down list, enter 500000.
- i Click Apply.

The Current clauses field now displays:

```
StockNumber > 400000 AND
StockNumber < 500000
```

8 Click OK.

The WHERE Clauses dialog box closes. The Where field and SQL statement display the Where Clause you specified.

- **9** Assign the query results to the MATLAB workspace variable A.
- 10 Click Execute.

11 To view the results, enter A in the Command Window:

A =

[400314]	[3000]	[2400]	[1800]
[400339]	[4300]	[NaN]	[2600]
[400345]	[5000]	[3500]	[2800]
[400455]	[1200]	[900]	[800]
[400876]	[3000]	[2400]	[1500]
[400999]	[3000]	[1500]	[1000]

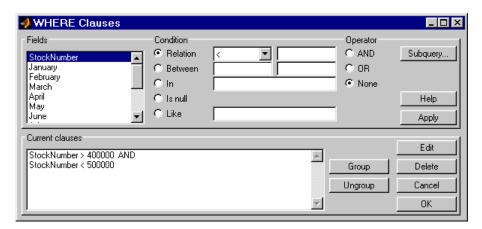
12 Save this query as basic_where.qry.

Grouping Statements

Use the WHERE Clauses dialog box to group query statements. In this example, modify basic_where.qry to retrieve data where sales in January, February, or March exceed 1500 units, if sales in each month exceed 1000 units.

To modify basic_where.qry:

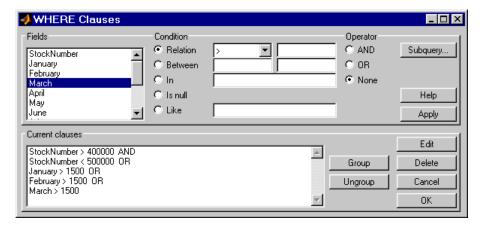
1 Click Where in VQB. The WHERE Clauses dialog box appears.



2 Modify the query to retrieve data if sales in January, February, or March exceed 1500 units.

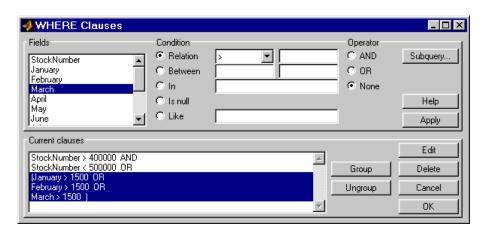
- a In Current clauses, select StockNumber < 500000 and click Edit.
- **b** For **Operator**, select **OR** and click **Apply**.
- c In Fields, select January. For Relation, select > and enter 1500 in its field. For Operator, select OR. Click Apply.
- **d** Repeat step c twice, specifying February and March in **Fields**.

The WHERE Clauses dialog box now looks as follows.



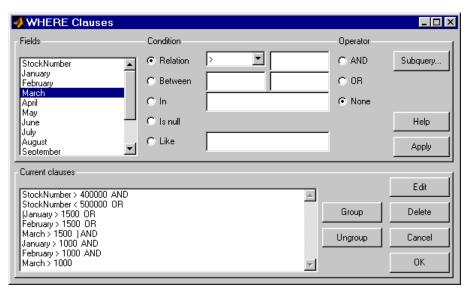
- **3** Group the criteria that require sales in each month to exceed 1500 units.
 - a In Current clauses, select the statement January > 1500 OR. Click Shift+click to select February > 1500 OR and March > 1500 also.
 - b Click Group.

An opening parenthesis is added before January and a closing parenthesis is added after March > 1500, indicating that these statements are evaluated as a group.



- **4** Modify the query to retrieve data if sales in each month exceed 1000 units.
 - a Select March > 1500) in Current clauses and click Edit.
 - **b** Select AND for **Operator** and click **Apply**.
 - c Select January in Fields. Select > for Relation and enter 1000 in its field. Select AND for Operator. Click Apply.
 - f d Repeat step c twice, specifying February and March in ${f Fields}.$

The WHERE Clauses dialog box now looks as follows.



e Click OK.

The WHERE Clauses dialog box closes. The SQL statement dialog box displays the modified where clause.

- **5** Assign the query results to the MATLAB workspace variable AA.
- **6** Click **Execute** to run the query.

7 To view the results, enter AA in the MATLAB Command Window.

AA =

[212569]	[2400]	[1721]	[1414]
[400314]	[3000]	[2400]	[1800]
[400339]	[4300]	[NaN]	[2600]
[400345]	[5000]	[3500]	[2800]
[400455]	[1200]	[900]	[800]
[400876]	[3000]	[2400]	[1500]
[400999]	[3000]	[1500]	[1000]

Removing Grouping of Statements

To use the WHERE Clauses dialog box to remove grouping criteria from the previous example:

- 1 In Current clauses, select (January > 1000 AND.
- $2 \, \mathrm{Click} \, \mathrm{Shift} + \mathrm{click} \, \mathrm{to} \, \mathrm{select} \, \mathrm{February} \, > \, 1000 \, \mathrm{AND} \, \mathrm{and} \, \mathrm{March} \, > \, 1000) \, \mathrm{also}.$
- 3 Click Ungroup.

The parentheses are removed from the statements, indicating that their grouping is removed.

Displaying Results in a Specified Order

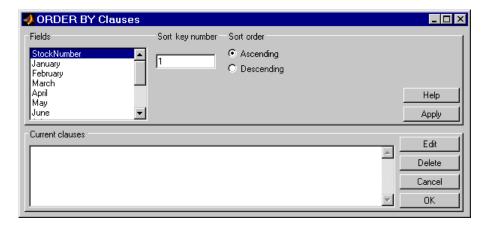
Use **Order by** in **Advanced query options** to specify the order in which query results display.

This example uses the basic_where.qry query you created in "Retrieving Data That Meets Specified Criteria" on page 4-24. The results of basic_where.qry are sorted so that January is the primary sort field, February the secondary, and March the last. Results for January and February appear in ascending order, and results for March appear in descending order.

To specify the order in which results appear in basic_where.qry:

- 1 Load basic_where.qry.
- **2** Set the **Data return format** preference to cellarray.
- 3 Set Read NULL numbers to NaN.
- 4 In Advanced query options, select Order by.

The **Order By Clauses** dialog box appears.



5 Enter values for the Sort key number and Sort order fields for the appropriate Fields.

To specify January as the primary sort field and display results in ascending order:

- a In Fields, select January.
- **b** For **Sort key number**, enter 1.
- c For Sort order, select Ascending.
- d Click Apply.

The **Current clauses** area now displays:

January ASC

- **6** To specify February as the second sort field and display results in ascending order:
 - a In Fields, select February.
 - **b** For **Sort key number**, enter 2.
 - c For Sort order, select Ascending.
 - d Click Apply.

The **Current clauses** area now displays:

January ASC February ASC

- 7 To specify March as the third sort field and display results in descending order:
 - a In Fields, select March.
 - **b** For **Sort key number**, enter 3.
 - c For Sort order, select Descending.
 - d Click Apply.

The **Current clauses** area now displays:

January ASC February ASC March DESC

8 Click OK.

The Order By Clauses dialog box closes. The **Order by** field and the **SQL** statement in VQB display the specified Order By clause.

- 9 Assign the query results to the MATLAB workspace variable B.
- **10** Click **Execute** to run the query.

11 To view the results, enter B in the MATLAB Command Window. Enter A to display the unordered query results and compare them to B. Your results look as follows:

A =

[400314]	[3000]	[2400]	[1800]
[400339]	[4300]	[NaN]	[2600]
[400345]	[5000]	[3500]	[2800]
[400455]	[1200]	[900]	[800]
[400876]	[3000]	[2400]	[1500]
[400999]	[3000]	[1500]	[1000]

B =

[400455]	[1200]	[900]	[800]
[400999]	[3000]	[1500]	[1000]
[400314]	[3000]	[2400]	[1800]
[400876]	[3000]	[2400]	[1500]
[400339]	[4300]	[NaN]	[2600]
[400345]	[5000]	[3500]	[2800]

For B, results are first sorted by January sales, in ascending order. The lowest value for January sales, 1200 (for item number 400455), appears first. The highest value, 5000 (for item number for 400345), appears last.

For items 400999, 400314, and 400876, January sales were 3000. Therefore, the second sort key, February sales, applies. February sales appear in ascending order: 1500, 2400, and 2400 respectively.

For items 400314 and 400876, February sales were 2400, so the third sort key, March sales, applies. March sales appear in descending order: 1800 and 1500, respectively.

Using Having Clauses To Refine Group By Results

Using the HAVING Clauses Dialog Box

Use the **Having** function to refine the results of a **Group By** clause.

0K

HAVING Clauses _ _ × Fields Condition Operator Relation C AND Subquery... January O OR February C Between O In None C Is null Help C Like Apply Current clauses Edit Group Delete Ungroup Cancel

After specifying a group-by clause in **Advanced query options**, click **Having**. The HAVING Clauses dialog box appears.

- 1 From the **Fields** list box, select the entry whose value to restrict.
- **2** Define the **Condition** for the selected field, as described in "Retrieving Data That Meets Specified Criteria" on page 4-24.
- **3** Select **Operator** to add another condition.
- **4** Click **Apply** to create the clause.

The subquery appears in the **Current clauses** area.

- **5** Repeat steps 1 through 4 to add more conditions as needed.
- **6** Change the clauses as needed:
 - To edit a clause:
 - 1 Select the clause from Current clauses and click Edit.
 - **2** Modify the **Fields**, **Condition**, and **Operator** fields as needed.
 - 3 Click Apply.
 - To group clauses:

- 1 Select the clauses to group from Current clauses. Use Ctrl+click or **Shift**+click to select multiple clauses.
- **2** Click **Group**. Parentheses are added around the set of clauses.

To ungroup clauses, select the clauses and then click **Ungroup**.

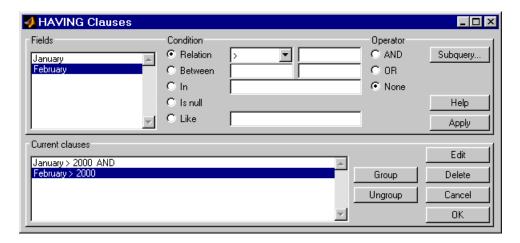
- To delete a clause, Select the clause from Current clauses and click **Delete**. Use Ctrl+click or Shift+click to select multiple clauses.
- **7** Specify a subquery in the HAVING Clauses dialog box, as needed. For more information, see "Creating Subqueries for Values from Multiple Tables" on page 4-37.
- 8 Click OK.

The Having Clauses dialog box closes. The SQL statement in the Visual Query Builder dialog box updates to reflect the specified having clause.

Example: Using Having Clauses

This example restricts the results from basic where gry to sales greater than 2000 for January and February:

- 1 In Advanced query options, click Having. The HAVING Clauses dialog box appears.
- **2** For January:
 - **a** Select > as the **Relation Condition**.
 - **b** Enter 2000 as the **Relation** value.
 - c Select the AND Operator.
 - d Click Apply.
- **3** For February:
 - Select > as the Relation Condition.
 - **b** Enter2000 as the **Relation** value.
 - **c** Click **Apply**. The HAVING Clauses dialog box appears as follows.



4 Click OK.

The Having Clauses dialog box closes. The **SQL statement** field in the VQB dialog box reflects the specified Having clause.

5 Assign a **MATLAB workspace variable** C, and click **Execute** to run the query.

C =

[3000]	[2400]
[5000]	[3500]

Compare these results to those in "Displaying Results in a Specified Order" on page 4-31.

Creating Subqueries for Values from Multiple Tables

Use the **Where** feature in **Advanced query options** to create subqueries. Creating subqueries in this way is referred to as *nested SQL*.

This example uses basic.qry, which you created in "Saving Queries" in the Database Toolbox Getting Started Guide.

The salesVolume table has sales volumes and stock number fields, but no product description field. The productTable has product description and

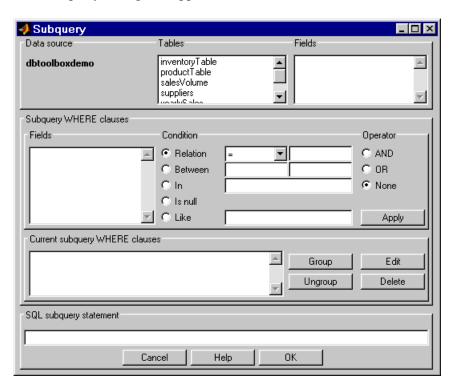
stock number fields, but no sales volumes. This example retrieves the stock number for the product whose description is Building Blocks from the productTable table. It then gets the sales volume values for that stock number from the salesVolume table.

- 1 Load basic.qry.
- 2 Set the Data return format Preference to cellarray and Read NULL numbers as to NaN.
- 3 Click Where in Advanced query options.

The WHERE Clauses dialog box appears.

4 Click Subquery.

The Subquery dialog box appears.

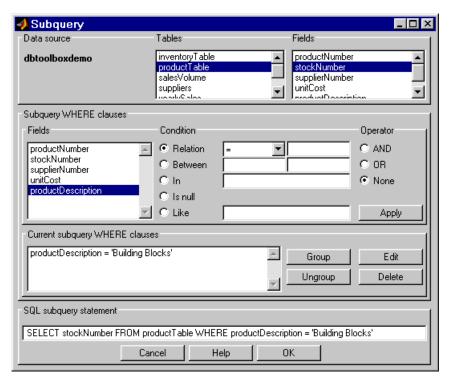


- **5** In **Tables**, select productTable, which includes the association between the stock number and the product description. The fields in that table appear.
- **6** In **Fields**, select **stockNumber**, the field that is common to this table and the table from which you are retrieving results.

The statement SELECT stockNumber FROM productTable is created in the SQL subquery statement.

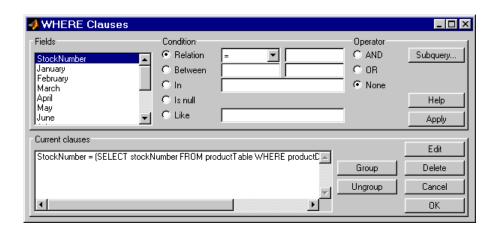
- **7** Limit the query to product descriptions that are Building Blocks.
 - a In Fields in Subquery WHERE clauses, select productDescription.
 - **b** For Condition, select Relation.
 - c In the drop-down list to the right of **Relation**, select =.
 - **d** In the field to the right of the drop-down list, enter 'Building Blocks'.
 - e Click Apply.

The clause appears in the Current subquery WHERE clauses field and is added to the SQL subquery statement.



- 8 Click OK to close the Subquery dialog box.
- **9** In the WHERE Clauses dialog box, click **Apply**.

This updates the Current clauses area using the subquery criteria specified in steps 3 through 8.



10 In the WHERE Clauses dialog box, click **OK**.

The WHERE Clauses dialog box closes. The **SQL statement** in the VQB dialog box updates.

- 11 Assign the query results to the MATLAB workspace variable C.
- 12 Click Execute.
- 13 Type C at the prompt in the MATLAB Command Window to see the results.

```
C = [400345] [5000] [3500] [2800]
```

- 14 The results are for item 400345, which has the product description Building Blocks, although that is not evident from the results. Create and run a query to verify that the product description is Building Blocks:
 - a For Data source, select dbtoolboxdemo.
 - **b** In **Tables**, select productTable.
 - c In Fields, select stockNumber and productDescription.
 - **d** Assign the query results to the **MATLAB** workspace variable P.
 - e Click Execute.

f Type P at the prompt in the MATLAB Command Window to view the results.

P = [125970] 'Victorian Doll' [212569] 'Train Set' [389123] 'Engine Kit' [400314] 'Painting Set' [400339] 'Space Cruiser' 'Building Blocks' [400345] [400455] 'Tin Soldier' [400876] 'Sail Boat'

'Slinky'

'Teddy Bear'

[400999]

[888652]

The results show that item 400345 has the product description Building Blocks. In the next section, you create a query that includes product description in the results.

Note You can include only one subquery in a query using VQB; you can include multiple subqueries using Database Toolbox functions.

Creating Queries That Include Results from Multiple Tables

A query whose results include values from multiple tables is said to perform a *join* operation in SQL.

This example retrieves sales volumes by product description. It is like the one in "Creating Subqueries for Values from Multiple Tables" on page 4-37, but this example creates a query that returns product description rather than stock number.

The sales Volume table has sales volume and stock number fields, but no product description field. The product Table table has product description and stock number fields, but no sales volume field. To create a query that retrieves data from both tables and equates the stock number from productTable with the stock number from salesVolume:

- 1 Set the **Data return format** preference to cellarray and the **Read NULL numbers as** preference to NaN.
- **2** For **Data operation**, choose **Select**.
- **3** For **Data source**, select dbtoolboxdemo.

The Catalog, Schema, and Tables for dbtoolboxdemo appear.

Do not specify Catalog or Schema.

4 In **Tables**, select the tables from which you want to retrieve data. For this example, click **Ctrl**+click and select both product**Table** and **salesVolume**.

The fields (columns) in those tables appear in **Fields**. Field names appear in the format fieldName.tableName. Therefore, productTable.stockNumber indicates the stock number in the product table and salesVolume.StockNumber indicates the stock number in the sales volume table.

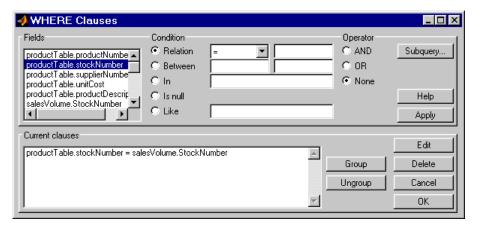
- **5** In **Fields**, click **Ctrl**+click to select the following fields:
 - productTable.productDescription
 - salesVolume.January
 - salesVolume.February
 - salesVolume.March
- 6 In this example, the Where clause equates the productTable.stockNumber with the salesVolume.StockNumber, so that product description is associated with sales volumes in the query results.

In **Advanced query options**, click **Where** to associate fields from different tables. The WHERE Clauses dialog box appears.

- **7** In the WHERE clauses dialog box:
 - **a** In **Fields**, select productTable.stockNumber.
 - **b** For Condition, select Relation.
 - **c** In the drop-down list to the right of **Relation**, select =.

- d In the field to the right of the drop-down list, enter salesVolume.StockNumber.
- e Click Apply.

The clause appears in the Current clauses field.



- f Click OK to close the WHERE Clauses dialog box. The Where field and SQL statement in VQB display the Where clause.
- **8** Assign the query results to the **MATLAB workspace variable** P1.
- **9** Click **Execute** to run the query.
- 10 Type P1 in the MATLAB Command Window.

P1 =

'Victorian Doll'	[1400]	[1100]	[981]
'Train Set'	[2400]	[1721]	[1414]
'Engine Kit'	[1800]	[1200]	[890]
'Painting Set'	[3000]	[2400]	[1800]
'Space Cruiser'	[4300]	[NaN]	[2600]
'Building Blocks'	[5000]	[3500]	[2800]
'Tin Soldier'	[1200]	[900]	[800]
'Sail Boat'	[3000]	[2400]	[1500]
'Slinky'	[3000]	[1500]	[1000]
'Teddy Bear'	[NaN]	[900]	[821]

Additional Advanced Query Options

For more information on advanced query options, choose an option and click **Help** in its dialog box. For example, click **Group by** in **Advanced query options**, and then click **Help** in the Group by Clauses dialog box.

Retrieving BINARY and OTHER Sun Java Data Types

This example shows how to retrieve data of types BINARY and OTHER, which may require manipulation before it can undergo MATLAB processing. To retrieve images using the SampleDB data source and a sample file that parses image data, matlabroot/toolbox/database/vqb/parsebinary.m:

- 1 For Data Operation, select Select.
- 2 In Data source, select SampleDB.
- **3** In **Tables**, select Employees.
- 4 In Fields, select EmployeeID and Photo (which contains bitmap images).
- **5** Select Query > Preferences.
- **6** In the **Data return format** field, specify cellarray.
- **7** As the MATLAB workspace variable, specify A.
- **8** Click **Execute** to run the query.

9 Type A in the MATLAB Command Window to view the query results.

```
A =
    [1]
            [21626x1 int8]
    [2]
            [21626x1 int8]
            [21722x1 int8]
    [3]
    [4]
            [21626x1 int8]
    [5]
            [21626x1 int8]
           [21626x1 int8]
    [6]
            [21626x1 int8]
    [7]
    [8]
            [21626x1 int8]
    [9]
            [21626x1 int8]
```

10 Assign the first element in A to the variable photo.

```
photo = A\{1,2\};
```

- 11 Make sure your current directory is writable.
- **12** Run the sample program parsebinary, which writes the retrieved data to a file, strips ODBC header information, and displays photo as a bitmap image.

```
cd I:\MATLABFiles\myfiles
parsebinary(photo, 'BMP');
```

For more information on parsebinary, enter help parsebinary, or view the parsebinary M-file in the MATLAB Editor/Debugger by entering open parsebinary in the Command Window.

Importing and Exporting BOOLEAN Data

In this section...

"Importing BOOLEAN Data from Databases to the MATLAB Workspace" on page 4-48

"Exporting BOOLEAN Data from the MATLAB Workspace to Databases" on page 4-51

Importing BOOLEAN Data from Databases to the **MATLAB Workspace**

BOOLEAN data is imported from databases into the MATLAB workspace as data type logical. This data has a value of 0 (false) or 1 (true), and is stored in a cell array or structure.

This example imports data from the Products table in the Nwind database into the MATLAB workspace.

- 1 Set Data return format to cellarray.
- 2 For Data operation, choose Select.
- **3** In **Data source**, select SampleDB.
- 4 In Tables, select Products.
- **5** In **Fields**, select ProductName and Discontinued.
- **6** Assign the query results to the MATLAB workspace variable D.
- **7** Click **Execute** to run the query.

VQB retrieves a 77-by-2 array.

8 Enter D in the MATLAB Command Window. 77 records are returned; only the first five records appear here due to space constraints.

D =	
'Chai'	[0]
'Chang'	[0]

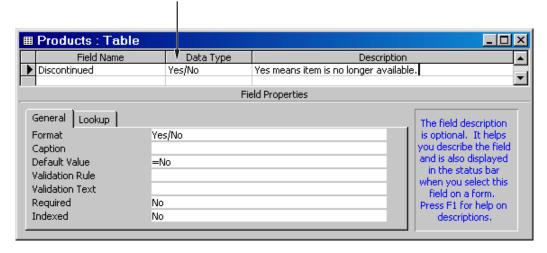
'Aniseed Syrup' [0] [1x28 char] [0] [1x22 char] [1]

9 Compare these results to the data in Microsoft Access.

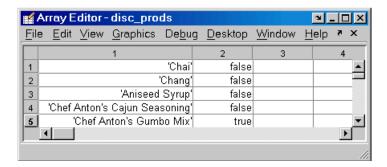
Discontinued field is BOOLEAN, where a check means true or Yes. <

duc	ts : Table									×
rodi	Product Name	Supplier	Category	Quantity F	Unit Pric	Units In :	Units	Reord	Discontinued	▲
1	Chai	Exotic Liq	Bevera	10 boxes :	\$18.00	39	0	10		
/ 2	Chang	Exotic Liq	Bevera	24 - 12 oz	\$19.00	17	40	25		
/ 3	Aniseed Syrup	Exotic Liq	Condii	12 - 550 m	\$10.00	13	70	25		
4	Chef Anton's Cajur	New Orlea	Condii	48 - 6 oz j	\$22.00	53	0	0		
5	Chef Anton's Guml	New Orlea	Condii	36 boxes	\$21.35	0	0	0	✓	~
: <u>I</u> ∢	4 6 ▶	▶1 ▶ * of 77		•					Þ	Γ

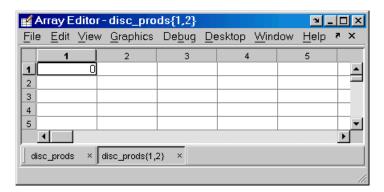
Design view in Access for the **Discontinued** field shows it is a Yes/No (B00LEAN) data type.



10 In the VQB Data area, double-click D to view its contents in the Variable Editor. The logical value for the first product, Chai, appears as false instead of 0.



11 In the Variable Editor, double-click false. Its logical value, 0, appears in a separate window.



For more information about MATLAB logical data types, see Logical Types in the MATLAB Programming Fundamentals documentation.

Exporting BOOLEAN Data from the MATLAB Workspace to Databases

Logical data is exported from the MATLAB workspace to a database as type BOOLEAN. This example adds two rows of data to the Products table in the Nwind database.

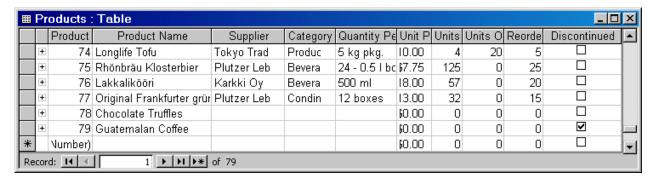
1 In the MATLAB workspace, create P, the structure you want to export.

```
P.ProductName{1,1}='Chocolate Truffles';
P.Discontinued{1,1}=logical(0);
P.ProductName{2,1}='Guatemalan Coffee';
P.Discontinued{2,1}=logical(1);
```

- 2 For Data operation, choose Insert.
- **3** In **Data source**, select SampleDB.
- 4 In Tables, select Products.
- **5** In **Fields**, select ProductName and Discontinued.
- **6** Assign results to the **MATLAB workspace variable** P.
- **7** Click **Execute** to run the query.

VQB inserts two new rows into the Products table.

View the table in Microsoft Access to verify that the data was correctly inserted.



Saving Queries in M-Files

```
In this section...
"About Generated M-Files" on page 4-52
"VQB Query Elements in M-Files" on page 4-53
```

About Generated M-Files

Select **Query > Generate M-File** to create an M-file that contains the equivalent Database Toolbox functions required to run an existing query that was created in VQB. Edit the M-file to include MATLAB or related toolbox functions, as needed. To run the query, execute the M-file.

The following is an example of an M-file generated by VQB.

```
% Set preferences with setdbprefs.
s.DataReturnFormat = 'cellarray';
s.ErrorHandling = 'store';
s.NullNumberRead = 'NaN';
s.NullNumberWrite = 'NaN';
s.NullStringRead = 'null';
s.NullStringWrite = 'null';
s.JDBCDataSourceFile = '';
s.UseRegistryForSources = 'yes';
s.TempDirForRegistryOutput = '';
setdbprefs(s)
% Make connection to database. Note that the password has been omitted.
% Using ODBC driver.
conn = database('dbtoolboxdemo','','password');
% Read data from database.
e = exec(conn, 'SELECT ALL StockNumber, January, February FROM salesVolume');
e = fetch(e);
close(e)
% Close database connection.
close(conn)
```

VQB Query Elements in M-Files

The following VQB query elements do not appear in generated M-files.

• Generated M-files do not include MATLAB workspace variables to which you assigned query results in the VQB query. The M-file assigns the query results to e; access these results using the variable e.Data. For example, you can add a statement to the M-file that assigns a variable name to e.Data as follows:

myVar = e.Data

• For security reasons, generated M-files do not include passwords required to connect to databases. Instead, the database statement includes the string 'password' as a placeholder. To run M-files to connect to databases that require passwords, substitute your password for the string password in the database statement.

Using Database Toolbox Functions

- "Getting Started with Database Toolbox Functions" on page 5-2
- "Importing Data from Databases into the MATLAB Workspace" on page 5-3
- "Viewing Information About Imported Data" on page 5-9
- "Exporting Data from the MATLAB Workspace to a New Record in a Database" on page 5-11
- "Replacing Existing Data in Databases with Data Exported from the MATLAB Workspace" on page 5-15
- "Exporting Multiple Records from the MATLAB Workspace" on page 5-17
- "Retrieving BINARY or OTHER Sun Java SQL Data Types" on page 5-21
- "Working with Database Metadata" on page 5-23
- "Using Driver Functions" on page 5-29
- "About Objects and Methods in the Database Toolbox Software" on page 5-31

Getting Started with Database Toolbox Functions

The following sections provide examples of how to use Database Toolbox functions. M-files that include functions used in some of these examples are available in matlab/toolbox/database/dbdemos.

Follow these simple examples consecutively when you first start using the product. Once you are familiar with Database Toolbox usage, refer to these examples as needed.

Importing Data from Databases into the MATLAB Workspace

This example imports country data from the customers table in the Nwind sample database into the MATLAB workspace using the following functions:

- database
- exec
- fetch (cursor.fetch)
- logintimeout
- ping
- setdbprefs

For more information on these functions, see matlab\toolbox\database\dbdemos\dbimportdemo.m.

1 Before you connect to a database, set the maximum time that you want to allow the MATLAB software session to try to connect to a database to 5 seconds.

```
logintimeout(5)
ans=
5
```

Note If you are connecting to a database using a JDBC connection, you must use different function syntax in this step. For more information, see the logintimeout function reference page.

- **2** Use the database function to define a MATLAB variable, conn, to represent the returned connection object. Pass the following arguments to this function:
 - The name of the database, SampleDB
 - The username and password, which are specified as empty strings because SampleDB does not require a user name or password

```
conn = database('SampleDB', '', '')
```

Enter conn at the command prompt to see the data.

```
conn =
   Instance: 'SampleDB'
   UserName: ''
      Driver: []
         URL: []
Constructor: [1x1 com.mathworks.toolbox.database.databaseConnect]
    Message: []
     Handle: [1x1 sun.jdbc.odbc.JdbcOdbcConnection]
    TimeOut: 5
 AutoCommit: 'on'
        Type: 'Database Object'
```

Note If you are connecting to a database using a JDBC connection, you need to specify different syntax for the database function. For more information, see the database reference page.

3 Use ping to check that the database connection status is successful.

- **4** Use the exec function to open a cursor and execute an SQL statement. Pass the following arguments to exec:
 - conn, the name of the connection object
 - select country from customers, a SQL statement that selects the country column of data from the customers table

```
curs = exec(conn, 'select country from customers')
```

The exec function returns the MATLAB variable curs.

```
curs =
     Attributes: []
            Data: 0
 DatabaseObject: [1x1 database]
        RowLimit: 0
       SQLQuery: 'select country from customers'
         Message: []
            Type: 'Database Cursor Object'
       ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
          Cursor: [1x1 com.mathworks.toolbox.database.sqlExec]
       Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
           Fetch: 0
```

5 The returned data contains strings, so you must convert it to a format that supports strings. Use setdbprefs to specify the format cellarray:

```
setdbprefs('DataReturnFormat','cellarray')
```

- 6 Import data into the MATLAB workspace using the fetch function. Pass the following arguments to this function:
 - curs, the cursor object returned by exec
 - 10, the maximum number of rows you want fetch to return

```
curs = fetch(curs, 10)
curs =
      Attributes: []
            Data: {10x1 cell}
 DatabaseObject: [1x1 database]
        RowLimit: 0
       SQLQuery: 'select country from customers'
         Message: []
            Type: 'Database Cursor Object'
       ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
          Cursor: [1x1 com.mathworks.toolbox.database.sqlExec]
       Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
           Fetch: [1x1 com.mathworks.toolbox.database.fetchTheData]
```

The curs object contains an element, Data, that contains the rows of data in the cell array.

The Attributes field is always empty. To view cursor attributes, use the attr function.

7 Assign the data element, curs. Data to the variable AA to display the Data element of curs:

```
AA = curs.Data
AA =

'Germany'
'Mexico'
'UK'
'Sweden'
'Germany'
'France'
'Spain'
'France'
'Canada'
```

- 8 To import more rows of data, rerun fetch until you retrieve all data.
- **9** Continue with the next example. To stop working now and resume working on the next example at a later time, close the cursor and the connection as follows:

close(curs) close(conn)

Viewing Information About Imported Data

This example shows how to view information about imported data and close the connection to the database using the following Database Toolbox functions:

- attr
- close
- cols
- columnnames
- rows
- width

For more information on these functions, see matlab\toolbox\database\dbdemos\dbinfodemo.m.

1 Open the cursor and connection if needed:

```
conn = database('SampleDB', '', '');
curs = exec(conn, 'select country from customers');
setdbprefs('DataReturnFormat', 'cellarray');
curs = fetch(curs, 10);
```

2 Use rows to return the number of rows in the data set:

```
numrows = rows(curs)
numrows =
10
```

3 Use cols to return the number of columns in the data set:

```
numcols = cols(curs)
numcols =
1
```

4 Use column ames to return the names of the columns in the data set:

```
colnames = columnnames(curs)
colnames =
```

'country'

5 Use width to return the column width, or size of the field, for the specified column number:

```
colsize = width(curs, 1)
colsize =
    15
```

6 Use attr to view multiple attributes for a column:

```
attributes = attr(curs)
attributes =
      fieldName: 'country'
       typeName: 'VARCHAR'
      typeValue: 12
    columnWidth: 15
      precision: []
          scale: []
       currency: 'false'
       readOnly: 'false'
       nullable: 'true'
        Message: []
```

Tip To import multiple columns, include a colnum argument in attr to specify the number of columns whose information you want.

7 Close the cursor.

```
close(curs)
```

8 Continue with the next example. To stop working now and resume working on the next example at a later time, close the connection.

```
close(conn)
```

Exporting Data from the MATLAB Workspace to a New Record in a Database

This example does the following:

- 1 Retrieves freight costs from an orders table.
- 2 Calculates the average freight cost and records the date on which the calculation was made.
- **3** Stores this data in a cell array.
- **4** Exports this data to an empty table.

You learn to use the following Database Toolbox functions:

- get
- fastinsert
- setdbprefs

For more information on these functions, see matlab\toolbox\database\dbdemos\dbinsertdemo.m.

1 Connect to the data source, SampleDB, if needed:

```
conn = database('SampleDB', '', '');
```

2 Use setdbprefs to set the format for retrieved data to numeric:

```
setdbprefs('DataReturnFormat', 'numeric')
```

3 Import three rows of data the freight column of data from the orders table.

```
curs = exec(conn, 'select freight from orders');
curs = fetch(curs, 3);
```

4 Assign the data to the **MATLAB workspace variable** AA:

```
AA = curs.Data
```

```
AA =
   32.3800
   11.6100
   65.8300
```

5 Calculate average freight cost and assign the number of rows in the array to numrows:

```
numrows = rows(curs);
```

6 Calculate the average of the data and assign the result to the variable meanA:

```
meanA = sum(AA(:))/numrows
meanA =
   36.6067
```

7 Assign the date on which the calculation was made to the variable D:

```
D = '20-Jan-2002';
```

8 Assign the date and mean to a cell array to export to a database. Put the date in the first cell of exdata:

```
exdata(1,1) = \{D\}
exdata =
  '20-Jan-2002'
```

Put the mean in the second cell of exdata:

```
exdata(1,2) = \{meanA\}
exdata =
    '20-Jan-2002'
                       [36.6067]
```

9 Define the names of the columns to which to export data. In this example, the column names are Calc Date and Avg Cost, from the Avg Freight Cost table in the SampleDB database. Assign the cell array containing the column names to the variable colnames:

```
colnames = {'Calc_Date', 'Avg Cost'};
```

10 Use the get function to determine the current status of the AutoCommit database flag. This status determines whether the exported data is automatically committed to the database. If the flag is off, you can undo an update; if it is on, data is automatically committed to the database.

```
get(conn, 'AutoCommit')
ans =
  on
```

The AutoCommit flag is set to on, so the exported data is automatically committed to the database.

- 11 Use the fastinsert function to export the data into the Avg_Freight_Cost table. Pass the following arguments to this function:
 - conn, the connection object for the database
 - Avg_Freight_Cost, the name of the table to which you are exporting data
 - The cell arrays colnames and exdata

```
fastinsert(conn, 'Avg_Freight_Cost', colnames, exdata)
```

fastinsert appends the data as a new record at the end of the Avg Freight Cost table.

12 In Microsoft Access, view the Avg_Freight_Cost table to verify the results.



The Avg_Cost value was rounded to a whole number to match the properties of that field in Access.

13 Close the cursor.

close(curs)

14 Continue with the next example. To stop now and resume working with the next example at a later time, close the connection.

close(conn)

Replacing Existing Data in Databases with Data Exported from the MATLAB Workspace

This example updates the date field that you previously imported into the Avg Freight Cost table using the following Database Toolbox functions:

- close
- update

For more information on these functions, see matlab\toolbox\database\dbdemos\dbupdatedemo.m.

1 If you have completed the previous example, skip this step. Otherwise, enter the following commands:

```
conn = database('SampleDB', '', '');
colnames = {'Calc_Date', 'Avg_Cost'};
D = '20-Jan-2002';
meanA = 36.6067;
exdata = {D, meanA}
exdata =
    '20-Jan-2002' [36.6067]
```

2 Change the date in the Avg_Freight_Cost table from 20-Jan-2002 to 19-Jan-2002:

```
D = '19-Jan-2002'
```

3 Assign the new date value to the newdata cell array.

```
newdata(1,1) = {D}
newdata =
'19-Jan-2002'
```

4 Specify the record to update in the database by defining a SQL where statement and assigning it to the variable whereclause. The record to update is the record whose Calc_Date is 20-Jan-2002. Because the date string is within a string, it is embedded within two single quotation marks rather than one.

```
whereclause = 'where Calc_Date = ''20-Jan-2002'''
whereclause =
where Calc_Date = '20-Jan-2002'
```

5 Export the data, replacing the record whose Calc Date is 20-Jan-2002.

```
update(conn,'Avg_Freight_Cost',colnames,newdata,whereclause)
```

6 In Microsoft Access, view the Avg Freight Cost table to verify the results.



7 Close the cursor and disconnect from the database.

```
close(conn)
```

Exporting Multiple Records from the MATLAB Workspace

This example does the following:

- 1 Imports monthly sales figures for all products from the tutorial database into the MATLAB workspace.
- 2 Computes total sales for each month.
- **3** Exports the totals to a new table.

You use the following Database Toolbox functions:

- fastinsert
- setdbprefs

For more information on these functions, see matlab\toolbox\database\dbdemos\dbinsert2demo.m.

- **1** Ensure that the tutorial database is writable, that is, not read-only.
- **2** Use the database function to connect to the data source, assigning the returned connection object as conn. Pass the following arguments to this function:
 - dbtoolboxdemo, the name of the data source
 - username and password, which are passed as empty strings because no user name or password is required to access the database

```
conn = database('dbtoolboxdemo', '', '');
```

3 Use the setdbprefs function to specify preferences for the retrieved data. Set the data return format to numeric and specify that NULL values read from the database are converted to 0 in the MATLAB workspace.

```
setdbprefs...
({'NullNumberRead';'DataReturnFormat'},{'0';'numeric'})
```

When you specify DataReturnFormat as numeric, the value for NullNumberRead must also be numeric.

4 Import data from the salesVolume table.

```
curs = exec(conn, 'select * from salesVolume');
curs = fetch(curs);
```

5 Use columnnames to view the column names in the fetched data set:

```
columnnames(curs)
ans =
'StockNumber', 'January', 'February', 'March', 'April',
'May', 'June', 'July', 'August', 'September', 'October',
 'November', 'December'
```

6 View the data for January (column 2).

```
curs.Data(:,2)
ans =
        1400
        2400
        1800
        3000
        4300
        5000
        1200
        3000
        3000
           0
```

7 Assign the dimensions of the matrix containing the fetched data set to m and n.

```
[m,n] = size(curs.Data)
m =
     10
n =
     13
```

8 Use m and n to compute monthly totals. The variable tmp is the sales volume for all products in a given month c. The variable monthly is the total sales volume of all products for that month. For example, if c is 2, row 1 of monthly is the total of all rows in column 2 of curs.Data, where column 2 is the sales volume for January.

```
for c = 2:n
  tmp = curs.Data(:,c);
  monthly(c-1,1) = sum(tmp(:));
end
```

View the result.

```
monthly
25100
15621
14606
11944
9965
8643
6525
5899
8632
13170
48345
172000
```

9 Create a string array containing the column names into which you want to insert the data, and assign the array to the variable colnames.

```
colnames{1,1} = 'salesTotal';
```

10 Use fastinsert to insert the data into the yearlySales table:

```
fastinsert(conn, 'yearlySales', colnames, monthly)
```

11 To verify that the data was imported correctly, view the yearlySales table in the tutorial database.

▦	_ 🗆 ×		
	Month	salesTotal	Revenue
		25100	\$0.00
		15621	\$0.00
		14606	\$0.00
		11944	\$0.00
		9965	\$0.00
		8643	\$0.00
		6525	\$0.00
		5899	\$0.00
		8632	\$0.00
		13170	\$0.00
		48345	\$0.00
		172000	\$0.00
*		0	\$0.00
Re	cord: 14 🕕	1	* of 12

12 Close the cursor and the database connection.

```
close(curs)
close(conn)
```

Retrieving BINARY or OTHER Sun Java SQL Data Types

This example retrieves images from the SampleDB data source using a sample file that parses image data, matlabroot/toolbox/database/vqb/parsebinary.m.

1 Connect to the SampleDB data source.

```
conn = database('SampleDB', '', '');
```

2 Specify cellarray as the data return format preference.

```
setdbprefs('DataReturnFormat','cellarray');
```

3 Import the EmployeeID and Photo columns of data from the Employees table.

```
curs = exec(conn, 'select EmployeeID,Photo from Employees')
curs = fetch(curs);
```

4 View the data you imported.

[9]

```
curs.Data
ans =
    [1]
            [21626x1 int8]
            [21626x1 int8]
    [2]
            [21722x1 int8]
    [3]
    [4]
            [21626x1 int8]
    [5]
            [21626x1 int8]
            [21626x1 int8]
    [6]
            [21626x1 int8]
    [7]
            [21626x1 int8]
    [8]
```

Note Some OTHER data type fields may be empty, indicating that the data could not pass through the JDBC/ODBC bridge.

5 Assign the image element you want to the variable photo.

[21626x1 int8]

```
photo = curs.Data{1,2};
```

6 Run parsebinary. This program writes the retrieved data to a file, strips ODBC header information from it, and displays photo as a bitmap image in a figure window. Ensure that your current directory is writable so that the output of parsebinary can be written to it.

```
cd 'I:\MATLABFiles\myfiles
parsebinary(photo, 'BMP');
```

For more information on parsebinary, enter help parsebinary or view the M-file in the MATLAB Editor/Debugger by entering open parsebinary.

Working with Database Metadata

In this section... "Accessing Metadata" on page 5-23 "Resultset Metadata Objects" on page 5-28

Accessing Metadata

In this example, you use the following Database Toolbox functions to access metadata:

- dmd
- get
- supports
- tables
- 1 Connect to the dbtoolboxdemo data source.

2 Use the dmd function to create a database metadata objectdbmeta and return its handle, or identifier:

```
dbmeta = dmd(conn)
dbmeta = DMDHandle: ...
```

```
[1x1 sun.jdbc.odbc.JdbcOdbcDatabaseMetaData]
```

3 Use the get function to assign database properties data, dbmeta, to the variable v:

```
v = get(dbmeta)
v =
              AllProceduresAreCallable: 1
                AllTablesAreSelectable: 1
DataDefinitionCausesTransactionCommit: 1
   DataDefinitionIgnoredInTransactions: 0
            DoesMaxRowSizeIncludeBlobs: 0
                              Catalogs: {4x1 cell}
                       CatalogSeparator: '.'
                           CatalogTerm: 'DATABASE'
                   DatabaseProductName: 'ACCESS'
                DatabaseProductVersion: '04.00.0000'
           DefaultTransactionIsolation: 2
                    DriverMajorVersion: 2
                    DriverMinorVersion: 1
                            DriverName: [1x31 char]
                         DriverVersion: '2.0001 (04.00.6200)'
                   ExtraNameCharacters: [1x29 char]
                 IdentifierQuoteString: '`'
                      IsCatalogAtStart: 1
                MaxBinaryLiteralLength: 255
                  MaxCatalogNameLength: 260
                  MaxCharLiteralLength: 255
                   MaxColumnNameLength: 64
                   MaxColumnsInGroupBy: 10
                     MaxColumnsInIndex: 10
                   MaxColumnsInOrderBy: 10
                    MaxColumnsInSelect: 255
                     MaxColumnsInTable: 255
                        MaxConnections: 64
                   MaxCursorNameLength: 64
                        MaxIndexLength: 255
                MaxProcedureNameLength: 64
                            MaxRowSize: 4052
                   MaxSchemaNameLength: 0
```

```
MaxStatementLength: 65000
                   MaxStatements: 0
              MaxTableNameLength: 64
               MaxTablesInSelect: 16
               MaxUserNameLength: 0
                NumericFunctions: [1x73 char]
                   ProcedureTerm: 'QUERY'
                         Schemas: {}
                      SchemaTerm: ''
              SearchStringEscape: '\'
                     SQLKeywords: [1x461 char]
                 StringFunctions: [1x91 char]
      StoresLowerCaseIdentifiers: 0
StoresLowerCaseQuotedIdentifiers: 0
      StoresMixedCaseIdentifiers: 0
StoresMixedCaseQuotedIdentifiers: 1
      StoresUpperCaseIdentifiers: 0
StoresUpperCaseQuotedIdentifiers: 0
                 SystemFunctions: ''
                      TableTypes: {13x1 cell}
               TimeDateFunctions: [1x111 char]
                        TypeInfo: {16x1 cell}
                             URL: ...
    'jdbc:odbc:dbtoolboxdemo'
                        UserName: 'admin'
           NullPlusNonNullIsNull: 0
             NullsAreSortedAtEnd: 0
           NullsAreSortedAtStart: 0
              NullsAreSortedHigh: 0
               NullsAreSortedLow: 1
           UsesLocalFilePerTable: 0
                  UsesLocalFiles: 1
```

Tip For more information about the database metadata properties returned by get, see the methods of the DatabaseMetaData object on the Sun Java Web site at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html.

4 Some information is too long to fit in the display area of the field, so the size of the field data appears instead. The Catalogs element is shown as a 4-by-1 cell array. View the Catalog information.

```
v.Catalogs
ans =
 'D:\Work\databasetoolboxfiles\Nwind'
 'D:\Work\databasetoolboxfiles\Nwind orig'
 'D:\Work\databasetoolboxfiles\tutorial'
 'D:\Work\databasetoolboxfiles\tutorial copy'
```

5 Use the supports function to see what properties this database supports:

```
a = supports(dbmeta)
a =
                          AlterTableWithAddColumn: 1
                         AlterTableWithDropColumn: 1
                              ANSI92EntryLevelSQL: 1
                                    ANSI92FullSQL: 0
                            ANSI92IntermediateSQL: 0
                       CatalogsInDataManipulation: 1
                       CatalogsInIndexDefinitions: 1
                   CatalogsInPrivilegeDefinitions: 0
                         CatalogsInProcedureCalls: 0
                       CatalogsInTableDefinitions: 1
                                   ColumnAliasing: 1
                                           Convert: 1
                                    CoreSQLGrammar: 0
                             CorrelatedSubqueries: 1
    DataDefinitionAndDataManipulationTransactions: 1
                 DataManipulationTransactionsOnly: 0
                   DifferentTableCorrelationNames: 0
                             ExpressionsInOrderBy: 1
                               ExtendedSQLGrammar: 0
                                    FullOuterJoins: 0
                                           GroupBy: 1
                              GroupByBeyondSelect: 1
                                  GroupByUnrelated: 0
                     IntegrityEnhancementFacility: 0
                                  LikeEscapeClause: 0
```

```
LimitedOuterJoins: 0
            MinimumSQLGrammar: 1
         MixedCaseIdentifiers: 1
   MixedCaseQuotedIdentifiers: 0
           MultipleResultSets: 0
         MultipleTransactions: 1
           NonNullableColumns: 0
      OpenCursorsAcrossCommit: 0
    OpenCursorsAcrossRollback: 0
   OpenStatementsAcrossCommit: 1
OpenStatementsAcrossRollback: 1
             OrderByUnrelated: 0
                   OuterJoins: 1
             PositionedDelete: 0
             PositionedUpdate: 0
    SchemasInDataManipulation: 0
    SchemasInIndexDefinitions: 0
SchemasInPrivilegeDefinitions: 0
      SchemasInProcedureCalls: 0
    SchemasInTableDefinitions: 0
              SelectForUpdate: 0
             StoredProcedures: 1
      SubqueriesInComparisons: 1
           SubqueriesInExists: 1
              SubqueriesInIns: 1
      SubgueriesInQuantifieds: 1
        TableCorrelationNames: 1
                 Transactions: 1
                        Union: 1
                     UnionAll: 1
```

A 1 for a given property indicates that the database supports that property; a 0 means that the database does not support the property.

Tip For more information about properties that the database supports, see the methods of the DatabaseMetaData object on the Sun Java Web site at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html.

- **6** Alternatively, use the tables function to retrieve metadata, such as the names and types of the tables in a catalog in the database. Pass the following arguments to this function:
 - dbmeta, the name of the database metadata object.
 - tutorial, the name of the catalog from which you want to retrieve table names.

```
t = tables(dbmeta, 'tutorial')
    'MSysAccessObjects'
                              'SYSTEM TABLE'
    'MSysIMEXColumns'
                              'SYSTEM TABLE'
    'MSysIMEXSpecs'
                              'SYSTEM TABLE'
    'MSysObjects'
                              'SYSTEM TABLE'
    'MSysQueries'
                              'SYSTEM TABLE'
    'MSysRelationships'
                              'SYSTEM TABLE'
    'inventoryTable'
                              'TABLE'
    'productTable'
                              'TABLE'
    'salesVolume'
                              'TABLE'
    'suppliers'
                              'TABLE'
    'yearlySales'
                              'TABLE'
    'display'
                              'VIEW'
```

7 Close the database connection.

```
close(conn)
```

Resultset Metadata Objects

Use the resultset function to create resultset objects for cursor object. Then, use the rsmd function to get metadata information about the resultset objects.

For more information, see the resultset and rsmd function reference pages.

Using Driver Functions

This example uses the following Database Toolbox functions to create driver and drivermanager objects, and to get and set their properties:

- drivermanager
- driver
- get
- isdriver
- set

Note There is no equivalent M-file demo available for this example, because this example relies on a specific system-to-JDBC connection and database. Your configuration is different from the one in this example, so you cannot run these examples exactly as written. Instead, substitute appropriate values for your own system. See your database administrator for more information.

1 Connect to the database.

```
c = database('orc1','scott','tiger',...
'oracle.jdbc.driver.OracleDriver',...
'jdbc:oracle:thin:@144.212.123.24:1822:');
```

2 Use the driver function to construct a driver object and return its handle, for a specified database URL string of the form jdbc:subprotocol:subname.

```
d = driver('jdbc:oracle:thin:@144.212.123.24:1822:')
DriverHandle: [1x1 oracle.jdbc.driver.OracleDriver]
```

3 Use the **get** function to get information, such as version data, for the driver object.

```
v = get(d)
v =
   MajorVersion: 1
   MinorVersion: 0
```

4 Use isdriver to verify that d is a valid JDBC driver object.

```
isdriver(d)
ans =
 1
```

This result shows that d is a valid JDBC driver object. If it is a not valid JDBC driver object, the returned result is 0.

5 Use the drivermanager function to create a drivermanager object dm.

```
dm = drivermanager
```

6 Get properties of the drivermanager object.

```
v = get(dm)
v =
       Drivers: {'sun.jdbc.odbc.JdbcOdbcDriver@761630' ...
   [1x38 char]}
 LoginTimeout: 0
     LogStream: []
```

7 Set the LoginTimeout value to 10 for all drivers loaded during this session.

```
set(dm, 'LoginTimeout',10)
```

Verify the LoginTimeout value.

```
v = get(dm)
v =
      Drivers: {'sun.jdbc.odbc.JdbcOdbcDriver@761630'}
LoginTimeout: 10
    LogStream: []
```

About Objects and Methods in the Database Toolbox Software

This toolbox is an object-oriented application. You do not need to be familiar with the product's object-oriented implementation to use it; this information is provided for reference purposes.

The Database Toolbox software includes the following objects:

- Cursor
- Database
- Database metadata
- Driver
- Drivermanager
- Resultset
- Resultset metadata

Each object has its own method directory, whose name begins with an @ sign, in the matlabroot/toolbox/database/database directory. M-file functions in the directory for each object provide methods for operating on the object.

Object-oriented characteristics of the toolbox enable you to:

Use constructor functions to create and return information about objects.

For example, to create a cursor object containing query results, run the fetch (cursor.fetch) function. The object and stored information about the object are returned. Because objects are MATLAB structures, you can view elements of the returned object.

This example uses the fetch function to create a cursor object curs.

```
curs =
     Attributes: []
            Data: {10x1 cell}
 DatabaseObject: [1x1 database]
       RowLimit: 0
       SQLQuery: 'select country from customers'
         Message: []
            Type: 'Database Cursor Object'
       ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
          Cursor: [1x1 com.mathworks.toolbox.database.sqlExec]
       Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
           Fetch: [1x1 com.mathworks.toolbox.database.fetchTheData]
```

View the contents of the Data element in the cursor object.

```
curs.Data
ans =
   'Germany'
   'Mexico'
   'Mexico'
   'UK'
   'Sweden'
   'Germany'
   'France'
   'Spain'
   'France'
```

Use overloaded functions.

Objects allow the use of overloaded functions, which simplify usage because you only need to use one function to operate on objects. For example, use the get function to view properties of an object.

 Create custom methods that operate on Database Toolbox objects and store them in the MATLAB workspace as M-files. For more information, see "Methods — Defining Class Operations" in the Developing MATLAB Classes documentation.

Function Reference

Utilities (p. 6-2) Settings for login time, retrieval

format, and more

Database Connection (p. 6-2) Create, test, close, and set

parameters for database connection

SQL Cursor (p. 6-3) Set parameters for and execute

query

Data Import (p. 6-3) Import data from database to

the MATLAB workspace, and get information about imported data

Database Metadata Object (p. 6-4) Information about database data

Data Export (p. 6-5) Export data from the MATLAB

workspace to database

Driver Object (p. 6-5) Construct and get information about

database driver

Drivermanager Object (p. 6-6) Construct and get information about

database drivermanager

Resultset Object (p. 6-6) Construct and get information about

resultset

Resultset Metadata Object (p. 6-6) Construct and get information about

resultset metadata

Visual Query Builder (p. 6-7) Start query builder GUI and

configure JDBC data source



Utilities

logintimeout Set or get time allowed to establish

database connection

setdbprefs Set preferences for retrieval format,

errors, NULLs, and more

Database Connection

close Close database connection, cursor,

or resultset object

database Connect to database

Retrieve object properties get

getdatasources Return names of ODBC and JDBC

data sources on system

isconnection Detect whether database connections

are valid

Detect whether database connection isreadonly

is read-only

Get status information about ping

database connection

set Set properties for database, cursor,

or drivermanager object

setdbprefs Set preferences for retrieval format,

errors, NULLs, and more

sql2native Convert JDBC SQL grammar to

SQL grammar native to system

SQL Cursor

close Close database connection, cursor,

or resultset object

exec Execute SQL statement and open

cursor

get Retrieve object properties

querytimeout Get time specified for SQL queries

to succeed

runstoredprocedure Call stored procedure with input and

output parameters

set Set properties for database, cursor,

or drivermanager object

Data Import

attr Retrieve attributes of columns in

fetched data set

cols Retrieve number of columns in

fetched data set

columnnames Retrieve names of columns in fetched

data set

cursor.fetch Import data into MATLAB

workspace from cursor object created

by exec

database.fetch Execute SQL statement to import

data into MATLAB workspace

fetch cursor.fetch or database.fetch

fetchmulti Import data from multiple resultsets

querybuilder Start SQL query builder GUI to

import and export data

Return number of rows in fetched rows

data set

width Return field size of column in fetched

data set

Database Metadata Object

bestrowid Unique identifier for row in database

table

columnprivileges List database column privileges

columns Returns database table column

names

crossreference Retrieve information about primary

and foreign keys

dmd Construct database metadata object

exportedkeys Retrieve information about exported

foreign keys

get Retrieve object properties

importedkeys Return information about imported

foreign keys

indexinfo Return indices and statistics for

database tables

Get primary key information for primarykeys

database table or schema

procedurecolumns Get stored procedure parameters

and result columns of catalogs

Get stored procedures for catalogs procedures

supports Detect whether property is supported

by database metadata object

tableprivileges Return database table privileges tables Return database table names

versioncolumns Automatically update table columns

Data Export

commit Make database changes permanent

insert Add MATLAB data to database

tables (deprecated; use fastinsert

instead)

querybuilder Start SQL query builder GUI to

import and export data

rollback Undo database changes

update Replace data in database table with

MATLAB data

Driver Object

driver Construct database driver object

get Retrieve object properties

isdriver Detect whether driver is a valid

JDBC driver object

isjdbc Detect whether driver is JDBC

compliant

isurl Detect whether database URL is

valid

register Load database driver unregister Unload database driver

Drivermanager Object

drivermanager Construct database drivermanager

object

Retrieve object properties get

set Set properties for database, cursor,

or drivermanager object

Resultset Object

clearwarnings Clear warnings for database

connection or resultset

close Close database connection, cursor,

or resultset object

get Retrieve object properties

isnullcolumn Detect whether last record read in

resultset is NULL

namecolumn Map resultset column name to

resultset column index

resultset Construct resultset object

Resultset Metadata Object

get Retrieve object properties

rsmd Construct resultset metadata object

Visual Query Builder

confds Configure JDBC data source for

Visual Query Builder

 ${\tt querybuilder} \hspace{1.5cm} {\tt Start} \; {\tt SQL} \; {\tt query} \; {\tt builder} \; {\tt GUI} \; {\tt to}$

import and export data

Functions — Alphabetical List

Purpose

Retrieve attributes of columns in fetched data set

Syntax

```
attributes = attr(curs, colnum)
attributes = attr(curs)
```

Description

- attributes = attr(curs, colnum) retrieves attribute information for:
 - The column number colnum
 - in the fetched data set curs
- attributes = attr(curs) retrieves attribute information for all columns in the fetched data set curs and stores the data in a cell array.
- attributes = attr(colnum) displays attributes of column colnum.

A list of returned attributes appears in the following table.

Attribute	Description
fieldName	Name of the column
typeName	Data type
typeValue	Numerical representation of the data type
columnWidth	Size of the field
precision	Precision value for floating and double data types; an empty value is returned for strings
scale	Precision value for real and numeric data types; an empty value is returned for strings
currency	If true, data format is currency
readOnly	If true, data cannot be overwritten
nullable	If true, data can be NULL
Message	Error message returned by fetch

See Also

cols, columnnames, columns, cursor.fetch,dmd, get, tables, width

Purpose

Unique identifier for row in database table

Syntax

```
b = bestrowid(dbmeta, 'cata', 'sch')
b = bestrowid(dbmeta, 'cata', 'sch', 'tab')
```

Description

• b = bestrowid(dbmeta, 'cata', 'sch') returns the optimal set of columns in a table that uniquely identifies:

-

a row in the schema sch, in the catalog cata, for the database whose database metadata object is dbmeta.

b = bestrowid(dbmeta, 'cata', 'sch', 'tab') returns the optimal set of columns that uniquely identifies a row in table tab, in the schema sch, in the catalog cata, for the database whose database metadata object is dbmeta.

Examples

Run bestrowid, passing it the following arguments:

- dbmeta, the database metadata object
- \bullet msdb, the catalog
- geck, the schema
- builds, the table

```
b = bestrowid(dbmeta, 'msdb', 'geck', 'builds')
b =
    'build id'
```

The result indicates that each entry in the build_id column is unique and identifies the row.

See Also

columns, dmd, get, tables

Purpose

Clear warnings for database connection or resultset

Syntax

clearwarnings(conn)
clearwarnings(rset)

Description

- clearwarnings (conn) clears warnings reported for the database connection object conn.
- clearwarnings(rset) clears warnings reported for the resultset object rset.

Tip For command-line help on clearwarnings, use the overloaded methods:

help database/clearwarnings help resultset/clearwarnings

Examples

clearwarnings (conn) clears reported warnings for the database connection object conn.

See Also

database, get, resultset

close

Purpose Close database connection, cursor, or resultset object

Syntax close(object)

Description close (object) closes object, which frees up resources.

Allowable objects for close are listed in the following table.

Object	Description	Action Performed by close(object)
conn	Database connection object	Closes conn
curs	Cursor object	Closes curs
rset	Resultset object	Closes rset

Database connections, cursors, and resultsets remain open until you close them using the close function. Always close a cursor, connection, or resultset when you finish using it. Close a cursor before closing the connection used for that cursor.

Note The MATLAB software session closes open cursors and connections when exiting, but the database might not free up the cursors and connections.

Tip For command-line help on close, use the overloaded methods:

help database/close

help cursor/close

help resultset/close

Examples Close the cursor curs and the connection conn.

close(curs)
close(conn)

See Also cursor.fetch, database, exec, resultset

cols

Purpose Retrieve number of columns in fetched data set

Syntax numcols = cols(curs)

Description numcols = cols(curs) returns the number of columns in the fetched

data set curs.

Examples Display three columns in the fetched data set curs.

numcols = cols(curs)

numcols = 3

See Also attr, columnnames, columnprivileges, columns, cursor.fetch, get,

rows, width

Purpose Retrieve names of columns in fetched data set

Syntax FIELDSTRING = columnnames(CURSOR)

FIELDSTRING = columnnames(CURSOR, BCELLARRAY)

Description

FIELDSTRING = columnnames (CURSOR) returns the column names of the data selected from a database table. The column names are enclosed in guotes and separated by commas.

FIELDSTRING = columnnames(CURSOR, BCELLARRAY) returns the column names as a cell array of strings when BCELLARRAY is set to true.

Examples

1 Run a SQL query to return all columns from the Microsoft Access Northwind database employees table:

```
'select * from employees'
```

2 Use columnnames to retrieve all column names for the selected columns:

```
fieldString = columnnames(cursor)
fieldString = 'EmployeeID','LastName','FirstName','Title',
'TitleOfCourtesy','BirthDate','HireDate','Address',
'City','Region','PostalCode','Country','HomePhone',
```

See Also

attr, cols, columnprivileges, columns, cursor.fetch, get, width

columnprivileges

Purpose

List database column privileges

Syntax

```
lp = columnprivileges(dbmeta, 'cata', 'sch', 'tab')
lp = columnprivileges(dbmeta, 'cata', 'sch', 'tab', 'l')
```

Description

- lp = columnprivileges(dbmeta, 'cata', 'sch', 'tab') returns a list of privileges for:
 - All columns in the table tab
 - In the schema sch
 - In the catalog cata
 - For the database whose database metadata object is dbmeta
- lp = columnprivileges(dbmeta, 'cata', 'sch', 'tab', 'l') returns a list of privileges for:
 - column 1 in the table tab
 - In the schema sch
 - In the catalog cata
 - For the database whose database metadata object is dbmeta

Examples

- 1 Use columnprivileges, passing in the following arguments:
 - The database metadata object.dbmeta
 - The catalog msdb
 - The schema geck
 - The table builds
 - The column name build id

columnprivileges

This result shows:

- The table name, builds, in column 1
- The column name, build_id, in column 2
- The column privileges, 1p, in column 3
- 2 View the contents of the third column in 1p.

```
lp{1,3}
ans =
    'INSERT' 'REFERENCES' 'SELECT' 'UPDATE'
```

See Also

cols, columns, columnnames, dmd, get

Purpose

Returns database table column names

Syntax

```
1 = columns(dbmeta, 'cata')
1 = columns(dbmeta, 'cata', 'sch')
1 = columns(dbmeta, 'cata', 'sch', 'tab')
```

Description

- 1 = columns(dbmeta, 'cata') returns a list of:
 - All column names in the catalog cata
 - For the database whose database metadata object is dbmeta
- 1 = columns(dbmeta, 'cata', 'sch') returns a list of:
 - All column names in the schema sch
 - In the catalog cata
 - For the database whose database metadata object is dbmeta
- 1 = columns(dbmeta, 'cata', 'sch', 'tab') returns a list of columns for:
 - The table tab
 - In the schema sch
 - In the catalog cata
 - For the database whose database metadata object is dbmeta

Examples

- 1 Run columns, passing it the following arguments:
 - The database metadata object dbmeta
 - The catalog orcl
 - The schema schSCOTT

```
'EMP' {1x8 cell}
'SALGRADE' {1x3 cell}
'TRIAL' {1x3 cell}
```

The results show the names of the five tables in dbmeta, and cell arrays containing the column names in each table.

2 View the column names for the BONUS table:

```
1{1,2}
ans =
    'ENAME' 'JOB' 'SAL' 'COMM'
```

See Also

attr, bestrowid, cols, columnnames, columnprivileges, dmd, get, version columns $% \left(1\right) =\left(1\right) \left(1\right) \left$

Purpose Make database changes permanent

Syntax commit(conn)

Description commit (conn) makes permanent changes made to the database

connection conn since the last commit or rollback function was run. To

run this function, the AutoCommit flag for conn must be off.

Examples Example 1: Check the Status of the Autocommit Flag

Check that the status of the AutoCommit flag for connection conn is off.

```
get(conn,'AutoCommit')
ans =
  off
```

Example 2: Commit Data to a Database

1 Insert exdata into the columns DEPTNO, DNAME, and LOC in the table DEPT, for the data source conn.

```
fastinsert(conn, 'DEPT', {'DEPTNO';'DNAME';'LOC'},... exdata)
```

2 Commit this data.

```
commit(conn)
```

See Also

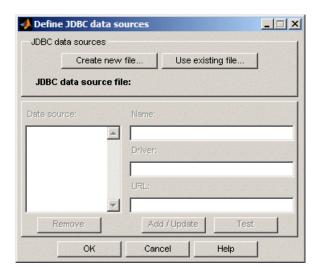
database, exec, fastinsert, get, rollback, update

Purpose Configure JDBC data source for Visual Query Builder

GUI Select Define JDBC data sources from the Visual Query Builder Alternatives Query menu.

Syntax confds

Description confds displays the VQB Define JDBC data sources dialog box. Use confds only to build and run queries using Visual Query Builder with JDBC drivers.



For information about how to use the Define JDBC data sources dialog box to configure JDBC drivers, see "Setting Up Data Sources for Use with JDBC Drivers" in the *Database Toolbox Getting Started Guide*.

Tip Use the database function to define JDBC data sources programmatically.

confds

See Also

database, querybuilder

Purpose

Retrieve information about primary and foreign keys

Syntax

```
f = crossreference(dbmeta, 'pcata', 'psch', 'ptab', 'fcata',
   'fsch', 'ftab')
```

Description

f = crossreference(dbmeta, 'pcata', 'psch', 'ptab',
'fcata', 'fsch', 'ftab') returns information about the relationship
between foreign keys and primary keys for the database whose database
metadata object is dbmeta. The primary key information is for:

- The table ptab
- In the primary schema psch
- Of the primary catalog pcata

The foreign key information is for:

- The foreign table ftab
- In the foreign schema fsch
- Of the foreign catalog fcata

Examples

Run crossreference to get primary and foreign key information given the following arguments:

- The database metadata object.dbmeta
- The primary and foreign catalog orcl
- The primary and foreign schema SCOTT
- The table DEPT that contains the referenced primary key
- $\bullet\,$ The table EMP that contains the foreign key

```
f = crossreference(dbmeta, 'orcl', 'SCOTT', 'DEPT',...
  'orcl', 'SCOTT', 'EMP')
f = Columns 1 through 7
```

```
'orcl' 'SCOTT' 'DEPT' 'DEPTNO' 'orcl'...
'SCOTT' 'EMP'

Columns 8 through 13
'DEPTNO' '1' 'null' '1' 'FK_DEPTNO'...
'PK_DEPT'
```

The results show the following primary and foreign key information.

Column	Description	Value
1	Catalog that contains primary key, referenced by foreign imported key	orcl
2	Schema that contains primary key, referenced by foreign imported key	SCOTT
3	Table that contains primary key, referenced by foreign imported key	DEPT
4	Column name of primary key, referenced by foreign imported key	DEPTNO
5	Catalog that has foreign key	orcl
6	Schema that has foreign key	SCOTT
7	Table that has foreign key	EMP
8	Foreign key column name that references the primary key in another table	DEPTNO
9	Sequence number within foreign key	1
10	Update rule, that is, what happens to the foreign key when the primary key updates	null
11	Delete rule, that is, what happens to the foreign key when the primary key is deleted	1

Column	Description	Value
12	Foreign imported key name	FK_DEPTNO
13	Primary key name in referenced table	PK_DEPT

There is only one foreign key in the schema SCOTT. The table DEPT contains a primary key DEPTNO that is referenced by the field DEPTNO in the table EMP. The field DEPTNO in the table EMP table is a foreign key.

Tip For a description of the codes for update and delete rules, see the getCrossReference property on the Sun Java Web site at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html.

See Also

dmd, exportedkeys, get, importedkeys, primarykeys

Purpose

Import data into MATLAB workspace from cursor object created by exec

GUI Alternatives

Retrieve data using Visual Query Builder. For more information about Visual Query Builder, see Chapter 4, "Using Visual Query Builder".

Syntax

```
curs = fetch(curs, RowLimit)
curs = fetch(curs)
```

Description

- curs = fetch(curs, RowLimit) imports rows of data into the object curs from the open SQL cursor curs, up to the maximum RowLimit.
- curs = fetch(curs) imports rows of data from the open SQL cursor curs into the object curs, up to RowLimit. Use the set function to specifyRowLimit.

Data is stored in a MATLAB cell array, structure, or numeric matrix. It is a best practice to assign the object returned by fetch to the variable curs from the open SQL cursor. This practice results in only one open cursor object, which consumes less memory than multiple open cursor objects.

The next time fetch is run, records are imported starting with the row following the specified RowLimit. If you do not specify a RowLimit, fetch imports all remaining rows of data.

Fetching large amounts of data can result in memory or speed issues. In this case, use RowLimit to limit how much data you retrieve at once.

Remarks

This page documents fetch for a cursor object. For more information about the use of fetch, cursor.fetch, and database.fetch, see fetch. Unless otherwise noted, fetch in this documentation refers to cursor.fetch, rather than database.fetch.

Examples Example 1: Import All Rows of Data

1 Use fetch to import all data into the cursor object curs, and store the data in a cell array contained in the cursor object field curs.Data.

curs = fetch(curs)
curs =
 Attributes: []
 Data: {91x1 cell}

DatabaseObject: [1x1 database]
 RowLimit: 0
 SQLQuery: 'select country from customers'
 Message: []
 Type: 'Database Cursor Object'
 ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
 Cursor: [1x1 com.mathworks.toolbox.database.sqlExec]
 Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
 Fetch: [1x1 com.mathworks.toolbox.database.fetchTheData]

2 Display data in curs.Data. Due to space constraints, only a portion of the returned data appears here.

Example 2 - Import a Specified Number of Rows

1

use the RowLimit argument to retrieve only the first three rows of data.

```
curs = fetch(curs, 3)
curs =
    Attributes: []
        Data: {3x1 cell}

DatabaseObject: [1x1 database]
    RowLimit: 0
    SQLQuery: 'select country from customers'
    Message: []
        Type: 'Database Cursor Object'
    ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
        Cursor: ...
[1x1 com.mathworks.toolbox.database.sqlExec]
        Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
        Fetch: ...
[1x1 com.mathworks.toolbox.database.fetchTheData]
```

b View the data.

```
curs.Data
ans =
    'Germany'
    'Mexico'
'Mexico'
```

2

a Rerun the fetch function to return the second three rows of data.

```
curs = fetch(curs, 3);
```

b View the data.

```
curs.Data
ans =
   'UK'
```

```
'Sweden'
'Germany'
```

Example 3 — Import Rows Iteratively until You Retrieve All Data

Use the RowLimit argument to retrieve the first ten rows of data, and then rerun the import using a while loop, retrieving ten rows at a time. Continue until you have retrieved all data, which occurs when curs.Data is 'No Data'.

```
% Initialize RowLimit (fetchsize)
fetchsize = 10
% Check for more data. Retrieve and display all data.
while ~strcmp(curs.Data, 'No Data')
  curs=fetch(curs,fetchsize);
  curs.Data(:)
end
ans =
   'No Data'
```

Example 4 - Import Numeric Data

Import a column of numeric data, using the setdbprefs function to specifynumeric as the format for the retrieved data.

```
conn = database('SampleDB', '', '');
curs=exec(conn, 'select all UnitsInStock from Products');
setdbprefs('DataReturnFormat', 'numeric')
curs=fetch(curs,3);
curs.Data
ans =
    39
    17
    13
```

Example 5 — Import BOOLEAN Data

1 Import data that includes a BOOLEAN field, using the setdbprefs function to specify cellarray as the format for the retrieved data.

```
conn = database('SampleDB', '', '');
curs=exec(conn, 'select ProductName, ...
Discontinued fromProducts');
setdbprefs('DataReturnFormat','cellarray')
curs=fetch(curs,5);
A=curs.Data
A =
    'Chai'
                        [0]
    'Chang'
                        [0]
    'Aniseed Syrup'
                        [0]
        [1x28 char]
                        [0]
        [1x22 char]
                       [1]
```

2 View the class of the second column of A:

```
class(A{1,2})
ans =
logical
```

See Also

attr, cols, columnnames, database, database.fetch, exec, fetch, fetchmulti, get, logical, rows, resultset, set, width, Chapter 4, "Using Visual Query Builder",

"Retrieving BINARY or OTHER Sun Java SQL Data Types" on page 5-21

Connect to database

GUI Alternatives

Connect to databases using Visual Query Builder. For more information on Visual Query Builder, see Chapter 4, "Using Visual Query Builder".

Syntax

```
conn = database('datasourcename','username','password')
conn = database('databasename','username',...
'password','driver','databaseurl')
```

Description

conn = database('datasourcename', 'username', 'password') connects a MATLAB software session to a database via an ODBC driver and assigns the returned connection object to conn. The arguments passed to this function are as follows:

- datasourcename: The data source to which you connect.
- username and password are the user name and password required to connect to the database. If a user name or password are not required to connect to your database, specify empty strings for these arguments.

conn = database('databasename', 'username',...
'password', 'driver', 'databaseurl') connects a MATLAB software session to a database and assigns the returned connection object to conn. The arguments passed to this function are as follows:

- databasename: The name of the database to which you connect.
- driver: The name of your JDBC driver.

Note The JDBC driver is sometimes referred to as the *class* that implements the Sun Java SQL driver for your database.

- username and password: The user name and password required to connect to the database. If a user name or password are not required to connect to your database, specify empty strings for these arguments.
- Find the correct driver name

databaseurl: A JDBC URL object of the form jdbc:subprotocol:subname. subprotocol is a database type, such as Oracle. subname may contain other information used by driver, such as the location of the database and/or a port number. subname may take the form //hostname:port/databasename.

If database establishes a database connection, it returns information about the connection object, as shown in the following example:

```
Instance: 'SampleDB'
UserName: ''
Driver: []
    URL: []
Constructor: [1x1 com.mathworks.toolbox.database.databaseConnect]
    Message: []
    Handle: [1x1 sun.jdbc.odbc.JdbcOdbcConnection]
    TimeOut: 0
AutoCommit: 'off'
    Type: 'Database Object'
```

Examples

Example 1 — Establish an ODBC Connection

Connect to an ODBC data source called Pricing, specifying user name mike, and password bravo.

```
conn = database('Pricing', 'mike', 'bravo');
```

Example 2 — Establish an ODBC Connection without Specifying a User Name and Password

Connect to an ODBC data source SampleDB where a user name and password are not required to access the database.

```
conn = database('SampleDB','','');
```

Example 3 — Establish a JDBC Connection

In this example, you establish a JDBC connection by passing the following arguments to the database function:

- oracle, the database to which you connect
- scott and tiger, the required user name and password
- oracle.jdbc.driver.OracleDriver, the oci7 JDBC driver name
- jdbc:oracle:oci7, the URL that specifies the location of the database server

```
conn = database('oracle','scott','tiger',...
'oracle.jdbc.driver.OracleDriver','jdbc:oracle:oci7:');
```

The JDBC name and URL take different forms for different databases, as shown in the examples in the following table.

JDBC Name and URL Example Syntax

Database	JDBC Name and URL Example Syntax
IBM Informix	JDBC driver: com.informix.jdbc.IfxDriver
	Database URL: jdbc:informix-sqli://161.144.202.206:3000: INFORMIXSERVER=stars
MySQL	JDBC driver: twz1.jdbc.mysql.jdbcMysqlDriver
	Database URL: jdbc:z1MySQL://natasha:3306/metrics
	JDBC driver: com.mysql.jdbc.Driver
	Database URL: jdbc:mysql://devmetrics.mrkps.com/testing
Oracle oci7 drivers	JDBC driver: oracle.jdbc.driver.OracleDriver
	Database URL: jdbc:oracle:oci7:@rex

database

JDBC Name and URL Example Syntax (Continued)

Database	JDBC Name and URL Example Syntax
Oracle oci8 drivers	JDBC driver: oracle.jdbc.driver.OracleDriver
	Database URL: jdbc:oracle:oci8:@111.222.333.44:1521:
	Database URL: jdbc:oracle:oci8:@frug
Oracle thin drivers	JDBC driver: oracle.jdbc.driver.OracleDriver
	Database URL: jdbc:oracle:thin:@144.212.123.24:1822:
Oracle 10 connections with JDBC (thin drivers)	JDBC driver: oracle.jdbc.driver.OracleDriver Database URL: jdbc:oracle:thin: (do not specify the target name and port)
	In this example, the target machine on which the database server resides is 144.212.123.24 and the port number is 1822.
PostgreSQL	JDBC driver: org.postgresql.Driver
	Database URL: jdbc:postgresql://masd/MOSE
PostgreSQL with SSL connection	JDBC driver: org.postgresql.Driver
	Database URL: jdbc:postgresql:servername:dbname:ssl= true&sslfactory=org.postgresql.ssl.NonValidatingFactory& (the trailing & is required)
Microsoft SQL Server	$ m JDBC\ driver:\ com.microsoft.jdbc.sqlserver.SQLServerDriver$
	Database URL: jdbc:sqlserver://localhost:port;database=databasename
Sybase SQL Server and Sybase SQL Anywhere	JDBC driver: com.sybase.jdbc.SybDriver
	Database URL: jdbc:sybase:Tds:yourhostname:yourportnumber/

database

See Also

close, dmd, exec, fastinsert, get, getdatasources, isconnection, isreadonly, logintimeout, ping, supports, update , $Chapter\ 4$, "Using Visual Query Builder"

Execute SQL statement to import data into MATLAB workspace

Syntax

```
results = fetch(conn, sqlquery)
results = fetch(conn, sqlquery, RowInc)
```

Description

- results = fetch(conn, sqlquery) executes the SQL statement sqlquery and imports data for the open connection object conn.results is a cell array, structure, or numeric matrix, based on specifications set by setdbprefs.
- results = fetch(conn, sqlquery, RowInc) executes the SQL statement sqlquery and imports RowInc rows of data at a time, given the open connection object conn. Data is stored in a MATLAB cell array, structure, or numeric matrix, based on specifications set by setdbprefs.

RowInc, manages speed and memory issues. It is a good practice to use RowInc when importing large amounts of data.

For more information on SQL statements, see exec.

Remarks

- This page documents fetch for a database object. For more information about the relationship with cursor.fetch, see fetch.
- The order of records in your database does not remain constant. Use the values in column names to identify records. Use the SQL ORDER BY command in your sqlquery statement to sort data.

Examples Example 1 - Import Data

1 Import the country column from the customers table in the SampleDB database.

```
conn= database('SampleDB','','');
setdbprefs('DataReturnFormat','cellarray')
results=fetch(conn, 'select country from customers')
results =
```

```
'Germany'
'Mexico'
'UK'
'Sweden'
...
'Finland'
'Brazil'
'USA'
'Finland'
```

2 View the size of the cell array into which the results were returned.

```
size(results)ans =
91 1
```

Tip Try running this example using the rowinc argument to address memory and speed issues.

Example 2— Import Two Columns of Data and View Information

1 Import the ProductName and Discontinued columns from the SampleDB database.

```
conn = database('SampleDB', '', '');
setdbprefs('DataReturnFormat','cellarray')
results=fetch(conn, 'select ProductName, Discontinued from Products');
```

2 View the size of the cell array into which the results were returned.

```
size(results)
ans =
77 2
```

3 To see the results for the first row of data, run:

```
results(1,:)
ans =
    'Chai' [0]
```

4 View the data type of the second element in the first row of data.

```
class(results{1,2})
ans =
logical
```

See Also

cursor.fetch, database, exec, fetch, logical,

"Retrieving BINARY or OTHER Sun Java SQL Data Types" on page 5-21

Purpose Construct database metadata object

Syntax dbmeta = dmd(conn)

Description dbmeta = dmd(conn)) constructs a database metadata object for the

database connection conn. Use get and supports to obtain properties of dbmeta. Use dmd and get(dbmeta) to obtain information you need about a database, such as table names required to retrieve data.

For a list of functions that operate on database metadata objects, enter:

help dmd/Contents

Examples

• dbmeta = dmd(conn) creates a database metadata object dbmeta for the database connection conn.

• v = get(dbmeta) lists properties of the database metadata object.

See Also

columns, database, get, supports, tables

driver

Purpose Construct database driver object

Syntax d = driver('s')

Description d = driver('s') constructs a database driver object d from s, where s

is a database URL string of the form jdbc:odbc:<name> or <name>. The

driver object ${\tt d}$ is the first driver that recognizes ${\tt s}$.

Examples d = driver('jdbc:odbc:thin:@144.212.123.24:1822:') creates

driver object ${\tt d}.$

See Also get, isdriver, isjdbc, isurl, register

drivermanager

Purpose Construct database drivermanager object

Syntax dm = drivermanager

Description dm = drivermanager constructs a database drivermanager object

which comprises the properties for all loaded database drivers. Use get

and set to obtain and change the properties of dm.

• dm = drivermanager creates a database drivermanager object dm.

• get(dm) returns properties of the drivermanager object dm.

See Also get, register, set

Execute SQL statement and open cursor

GUI Alternatives

Query databases using Visual Query Builder. For more information on Visual Query Builder, see Chapter 4, "Using Visual Query Builder".

Syntax

curs = exec(conn, 'sqlquery')

Description

curs = exec(conn, 'sqlquery') executes the SQL statement sqlquery for the database connection conn, and opens a cursor.

Running exec returns the cursor object to the variable curs and returns additional information about the cursor object. The sqlquery argument can be a stored procedure for that database connection, of the form {call sp_name (parm1,parm2,...)}.

Remarks

- After opening a cursor, use fetch to import data from the cursor. Use resultset, rsmd, and statement to get properties of the cursor.
- Use querytimeout to specify the maximum amount of time for which exec tries to execute the SQL statement.
- You can have multiple cursors open at one time.
- A cursor stays open until you close it using the close function.
- Unless noted in this reference page, the exec function supports all valid SQL statements, such as nested queries.
- The order of records in your database is not constant. Use values in column names to identify records. Use the SQL ORDER BY command to sort records.
- Before you modify database tables, ensure that the database is not open for editing. If you try to edit the database while it is open, you receive the following MATLAB error:

[Vendor][ODBC Driver] The database engine could not lock table 'TableName' because it is already in use by another person or process.

- For Microsoft Excel, tables in sqlquery are Excel® worksheets. By default, some worksheet names include \$. To select data from a worksheet with this name format, use a SQL statement of the form: select * from "Sheet1\$" (or 'Sheet1\$').
- You may experience issues with text field formats in the Microsoft SQL Server database management system. Workarounds for these issues include:
 - Converting fields of format NVARCHAR, TEXT, NTEXT, and VARCHAR to CHARin the database.
 - Usingsqlquery to convert data to VARCHAR. For example, run a sqlquery statement of the form 'select convert(varchar(20), field1) from table1'
- The PostgreSQL database management system supports multidimensional fields, but SQL select statements fail when retrieving these fields unless you specify an index.
- Some databases require that you include a symbol, such as #, before and after a date in a query. For example:

```
curs = exec(conn, 'select * from mydb where mydate > #03/05/2005#')
```

Examples Example 1 — Select Data from a Database Table

Select data from the customers table that you access using the database connection conn. Assign the returned cursor object to the variable curs.

```
curs = exec(conn, 'select * from customers')
curs =
   Attributes: []
        Data: 0
DatabaseObject: [1x1 database]
   RowLimit: 0
   SQLQuery: 'select * from customers'
   Message: []
        Type: 'Database Cursor Object'
   ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
        Cursor: [1x1 com.mathworks.toolbox.database.sqlExec]
   Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
   Fetch: 0
```

Example 2 — Select One Column of Data from Database Table

Select country data from the customers table that you access using the database connection conn. Assign the SQL statement to the variable sqlquery and assign the returned cursor to the variable curs.

```
sqlquery = 'select country from customers';
curs = exec(conn, sqlquery);
```

Example 3 — Use a Variable in a Query

1 Select data from the customers table that you access using the database connection conn, where country is a variable. In this example, you are prompted to specify your country. Your input is assigned to the variable UserCountry.

```
UserCountry = input('Enter your country: ', 's')
```

2 You are prompted as follows:

```
Enter your country:
```

Enter:

Mexico

3 To perform the query using your input, run:

```
curs = exec(conn, ...
['select * from customers where country= ' '''' UserCountry ''''])
curs=fetch(curs)
```

The select statement is created by using square brackets to concatenate the two strings select * from customers where country = and 'UserCountry'. The pairs of four quotation marks are needed to create the pair of single quotation marks that appears in the SQL statement around UserCountry. The outer two marks delineate the next string to concatenate, and two marks are required inside them to denote a quotation mark inside a string.

Tip Without using a variable, the function to retrieve the data would be:

```
curs = exec(conn, ['select * from customers where country = '...
''Mexico''])
curs=fetch(curs)
```

Example 4 — Roll Back or Commit Data Exported to Database Table

Use exec to roll back or commit data after running a fastinsert, insert, or update for which the AutoCommit flag is off.

• To roll back data for the database connection conn.

```
exec(conn, 'rollback')
```

• To commit the data, run:

```
exec(conn, 'commit');
```

Example 5 — Change Database Connection Catalog

Change the catalog for the database connection conn to intlprice.

```
curs = exec(conn, 'Use intlprice');
```

Example 6 — Create a Table and Add a New Column

This example creates a table and adds a new column to it.

1 Use the SQL CREATE command to create the table.

```
mktab = 'CREATE TABLE Person(LastName varchar, ...
FirstName varchar,Address varchar,Age int)'
```

2 Create the table for the database connection object conn.

```
exec(conn, mktab);
```

3 Use the SQL ALTER command to add a new column, City, to the table.

```
a = exec(conn, ...
'ALTER TABLE Person ADD City varchar(30)')
```

Example 7 — Run a Simple Stored Procedure

• Execute the stored procedure sp_customer_list for the database connection conn.

```
curs = exec(conn,'sp customer list');
```

• Run a stored procedure with input parameters.

```
curs = exec(conn, '{call sp name (parm1, parm2,...)}');
```

Example 8 — Return a Cursor Object Using a Stored Procedure

The following example calls a database stored procedure that returns a cursor object.

1 Specify data to return as a structure.

```
setdbprefs('DataReturnFormat','structure');
```

2 Define a stored procedure.

```
ssql_cmd1 = '{?= call get_int_by_id(1,1, ...
    to_date('07/02/05','MM/DD/YY'),...
    to_date('07/07/05','MM/DD/YY'))}';
```

3 Execute the stored procedure and open a cursor object.

4 Import data from the cursor to a MATLAB variable, **a**.

```
a = fetch(curs);
```

5 View a.Data.

```
a.Data
ans =
     TS_DT: {'2005-07-02 00:00:00.0'}
     INT_VALUE: 1
```

6 Define another stored procedure.

```
sql_cmd2='{?= call nrg.ts_get_int_by_id(1,1,...
    to_date(''07/02/05'',''MM/DD/YY''),...
    to_date(''07/20/05'',''MM/DD/YY''))}';
```

7 Repeat steps 1 through 5 using this new stored procedure.

```
curs = exec(conn, ssql cmd2)
curs =
       Attributes: []
             Data: 0
   DatabaseObject: [1x1 database]
         RowLimit: 0
         SQLQuery: [1x97 char]
          Message: []
             Type: 'Database Cursor Object'
        ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
           Cursor: ...
     [1x1 com.mathworks.toolbox.database.sqlExec]
        Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
            Fetch: 0
a = fetch(curs)
a =
       Attributes: []
             Data: [1x1 struct]
```

```
DatabaseObject: [1x1 database]
         RowLimit: 0
         SQLQuery: [1x97 char]
          Message: []
             Type: 'Database Cursor Object'
        ResultSet: ...
        [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
           Cursor: ...
       [1x1 com.mathworks.toolbox.database.sqlExec]
        Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
            Fetch: ...
       [1x1 com.mathworks.toolbox.database.fetchTheData]
a.Data
ans =
       TS DT: {2x1 cell}
   INT VALUE: [2x1 double]
```

8 Examine the attributes of a.

```
a.Data.TS_DT

ans =

'2005-07-02 00:00:00.0'

'2005-07-10 00:00:00.0'

a.Data.INT_VALUE

ans =

1

6
```

See Also

close, cursor.fetch, database, database.fetch, fastinsert, fetch, procedures, querybuilder, querytimeout, resultset, rsmd, set, update, Chapter 4, "Using Visual Query Builder", "Data Retrieval Restrictions" on page 1-7

Retrieve information about exported foreign keys

Syntax

```
e = exportedkeys(dbmeta, 'cata', 'sch'
e = exportedkeys(dbmeta, 'cata', 'sch', 'tab')
```

Description

- e = exportedkeys(dbmeta, 'cata', 'sch' returns foreign exported key information (that is, information about primary keys that are referenced by other tables) for:
 - The schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta
- e = exportedkeys(dbmeta, 'cata', 'sch', 'tab') returns exported foreign key information (that is, information about the primary key which is referenced by other tables), for:
 - The table tab
 - In the schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta

Examples

Get foreign exported key information for the schema SCOTT for the database metadata object dbmeta.

```
e = exportedkeys(dbmeta, 'orcl', 'SCOTT')
e =
  Columns 1 through 7
                                              'orcl' ...
    'orcl'
              'SCOTT'
                        'DEPT'
                                  'DEPTNO'
    'SCOTT'
                'FMP'
  Columns 8 through 13
    'DEPTNO'
                 '1'
                        'null'
                                   '1'
                                          'FK DEPTNO'...
  'PK DEPT'
```

The results show the foreign exported key information.

Column	Description	Value
1	Catalog containing primary key that is exported	null
2	Schema containing primary key that is exported	SCOTT
3	Table containing primary key that is exported	DEPT
4	Column name of primary key that is exported	DEPTNO
5	Catalog that has foreign key	null
6	Schema that has foreign key	SCOTT
7	Table that has foreign key	EMP
8	Foreign key column name, that is the column name that references the primary key in another table	DEPTNO
9	Sequence number within the foreign key	1
10	Update rule, that is, what happens to the foreign key when the primary key updates	null
11	Delete rule, that is, what happens to the foreign key when the primary key is deleted	1
12	Foreign key name	FK_DEPTNO
13	Primary key name that is referenced by foreign key	PK_DEPT

In the schema SCOTT, only one primary key is exported to (referenced by) another table. DEPTNO, the primary key of the table DEPT, is referenced by the field DEPTNO in the table EMP. The referenced table is DEPT and the referencing table is EMP. In the DEPT table, DEPTNO is an exported key. Reciprocally, the DEPTNO field in the table EMP is an imported key.

exportedkeys

For a description of codes for update and delete rules, see the getExportedKeys property on the Sun Java Web site at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html.

See Also

crossreference, dmd, get, importedkeys, primarykeys

Add MATLAB data to database table

GUI Alternatives

Export data using Visual Query Builder with **Data operation** set to **Insert**. For more information on Visual Query Builder, see Chapter 4, "Using Visual Query Builder".

Syntax

fastinsert(conn, 'tablename', colnames, exdata)

Description

fastinsert(conn, 'tablename', colnames, exdata) exports records from the MATLAB variable exdata into new rows in an existing database table tablename via the connection conn. The variable exdata can be a cell array, numeric matrix, or structure. You do not specify the type of data you are exporting; the data is exported in its current MATLAB format. Specify column names for tablename as strings in the MATLAB cell array colnames. If exdata is a structure, field names in the structure must exactly match colnames.

The status of the AutoCommit flag determines whether fastinsert automatically commits the data to the database. Use get to view the AutoCommit flag status for the connection and use set to change it. Use commit or issue an SQL commit statement using exec to commit the data to the database. Use rollback or issue an SQL rollback statement using exec to roll back the data.

Useupdate to replace existing data in a database.

Remarks

- The fastinsert function replaces the insert function. The two functions have the same syntax, but fastinsert provides better performance and supports more object types than insert. If fastinsert does not work as expected, try running insert.
- The fastinsert function only supports dates in U.S. format; that is, dates in mm/dd/yyyy format.
- The order of records in your database is not constant. Use values in column names to identify records.
- If an error message like the following appears when you runfastinsert, the table may be open in edit mode.

[Vendor][ODBC Product Driver] The database engine could not lock table 'TableName' because it is already in use by another person or process.

In this case, close the table in the database and rerun the fastinsert function.

Example 1 - Insert a Record

Insert a record consisting of two columns, City and Avg_Temp, into the Temperatures table. City is San Diego and Avg_Temp is 88 degrees. The database connection is conn. Assign the data to the cell array exdata.

```
exdata = {'San Diego', 88}
```

2 Create a cell array containing the column names in Temperatures.

```
colnames = {'City', 'Avg_Temp'}
```

3 Insert the data into the database.

```
fastinsert(conn, 'Temperatures', colnames, exdata)
```

The row of data is added to the Temperatures table.

Example 2 — Insert Multiple Records

Insert a cell array, exdata, that contains multiple rows of data and three columns, Date, Avg_Length, and Avg_Wt, into the Growth table. The database connection is conn.

Insert the data.

```
fastinsert(conn, 'Growth', ...
{'Date';'Avg_Length';'Avg_Wt'}, exdata)
```

The records are inserted into the table.

Example 3 — Import Records, Perform Calculations, and Export Data

Import data from a database into the MATLAB workspace, perform calculations on it, and then export the results to a database.

1 Import all data from the products table into a cell array.

```
conn = database('SampleDB', '', '');
curs = exec(conn, 'select * from products');
setdbprefs('DataReturnFormat', 'cellarray')
curs = fetch(curs);
```

2 Assign the first column of data to the variable prod_name.

```
prod_name = curs.Data(:,1);
```

3 Assign the sixth column of data to the variable price.

```
price = curs.Data(:,6);
```

4 Convert the cell array price to a numeric format, and calculate off 25% of the price. Assign the result of the calculation to the variable new_price.

```
new price =.75*[price{:}]
```

5 Export prod_name, price, and new_price to the Sale table. Because prod_name is a character array and price is numeric, you must export the data as a cell array. To do so, convert new_price from a numeric array back to a cell array. To convert the columns of data in new price to a cell array, run:

```
new price = num2cell(new price);
```

6 Create an array, exdata, that contains the three columns of data to export. Put prod_name in column 1, price in column 2, and new_price in column 3.

```
exdata(:,1) = prod_name(:,1);
exdata(:,2) = price;
exdata(:,3) = new price;
```

7 Assign the column names to a string array, colnames.

```
colnames={'product name', 'price', 'sale price'};
```

8 Export the data to the Sale table.

```
fastinsert(conn, 'Sale', colnames, exdata)
```

All rows of data are inserted into the Sale table.

Example 4 — Insert Numeric Data

Export tax_rate, a numeric matrix consisting of two columns, into the Tax table.

```
fastinsert(conn, 'Tax', {'rate', 'max_value'}, tax_rate)
```

Example 5 — Insert and Commit Data

1 Use the SQL commit function to commit data to a database after it has been inserted. The AutoCommit flag is off.

Insert the cell array exdata into the column names colnames of the Error_Rate table.

```
fastinsert(conn, 'Error_Rate', colnames, exdata)
```

2 Alternatively, commit the data using a SQL commit statement with the exec function.

```
cursor = exec(conn, 'commit');
```

Example 6 - Insert BOOLEAN Data

Insert BOOLEAN data (which is represented as MATLAB type logical) into a database.

```
conn = database('SampleDB', '', '');
P.ProductName{1}='Chocolate Truffles';
P.Discontinued{1}=logical(0);
fastinsert(conn, 'Products',...
{'ProductName';'Discontinued'}, P)
```

2 View the new record in the database to verify that the Discontinued field is BOOLEAN. In some databases, the MATLAB logical value 0 is shown as a BOOLEAN false, No, or a cleared check box.

See Also

commit, database, exec, insert, logical, querybuilder, rollback, set, update, Chapter 4, "Using Visual Query Builder"

cursor.fetch or database.fetch

About fetch, cursor.fetch, and database.fetch

There are two fetch functions in this toolbox, cursor.fetch and database.fetch. The fetch function runs one of these functions, depending on what object you provide to it as an argument. Use the syntax fetch with the appropriate object argument rather than explicitly specifying cursor.fetch or database.fetch.

For example, cursor.fetch runs when you pass a cursor object, curs, to fetch as an argument.

```
conn=database(...)
curs=exec(conn, sqlquery)
fetch(curs)
```

The database.fetch function runs when you pass a database object, conn, to fetch as an argument.

```
conn=database(...)
fetch(conn, sqlquery)
```

In this example, the results are effectively identical. database.fetch runs exec and returns results to the cursor object. It then runs cursor.fetch, returns results, and closes the cursor object. This shows that you can use a single call to the database.fetch function to get the same results as if you had called two functions, exec and cursor.fetch.

cursor.fetch returns a cursor object on which you can run many other functions, such as get and rows. For this reason, cursor.fetch is recommended for use in most situations. To import data into the MATLAB workspace without meta information about the data, use database.fetch instead of cursor.fetch.

Throughout the documentation, references to fetch denote cursor.fetch unless explicitly stated otherwise.

Explicitly specify database.fetch or cursor.fetch only when running help or doc. To get help for database.fetch, run help

database.fetch. Similarly, to view the reference pages for either version of fetch, run doc database.fetch or doc cursor.fetch.

See Also

cursor.fetch, database, database.fetch, exec

Import data from multiple resultsets

Syntax

curs = fetchmulti(curs)

Description

curs = fetchmulti(curs) imports data from the open SQL cursor object curs into the object curs, where the open SQL cursor object contains multiple resultsets.

Multiple resultsets are retrieved via exec with a sqlquery statement that runs a stored procedure consisting of two select statements.

cursmulti.Data contains data from each resultset associated with cursmulti.Statement. cursmulti.Data is a cell array consisting of cell arrays, structures, or numeric matrices as specified in setdbprefs; the data type is the same for all resultsets.

Examples

Use exec to run a stored procedure that includes multiple select statements and fetchmulti to retrieve the resulting multiple resultsets.

```
conn = database(...)
setdbprefs('DataReturnFormat','cellarray')
curs = exec(conn, '{call sp 1}');
curs = fetchmulti(curs)
Attributes: []
              Data: {{10x1 cell} {12x4 cell}}
    DatabaseObject: [1x1 database]
          RowLimit: 0
          SQLQuery: '{call sp 1}'
           Message: []
              Type: 'Database Cursor Object'
         ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
          [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
            Cursor: ...
     [1x1 com.mathworks.toolbox.database.sqlExec]
         Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
           [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
             Fetch: ...
     [1x1 com.mathworks.toolbox.database.fetchTheData]
```

See Also

cursor.fetch, database, exec, setdbprefs

Retrieve object properties

Syntax

```
v = get(object)
v = get(object, 'property')
v.property
```

Description

- v = get(object) returns a structure that contains object and its corresponding properties, and assigns the structure to v.
- v = get(object, 'property') retrieves the value of property for object and assigns the value to v.
- v.property returns the value of property after you have created v by running get.

Use set(object) to view a list of writable properties for object.

Allowable objects include:

- "Database Connection Objects" on page 7-58, which are created using database
- "Cursor Objects" on page 7-59, which are created using exec or fetch (cursor.fetch)
- "Driver Objects" on page 7-60, which are created using driver
- "Database Metadata Objects" on page 7-60, which are created using dmd
- "Drivermanager Objects" on page 7-61, which are created using drivermanager
- "Resultset Objects" on page 7-61, which are created using resultset
- "Resultset Metadata Objects" on page 7-62, which are created using rsmd

If you call these objects from applications that use Sun Java, you can get more information about object properties from the Java Web site at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html.

Database Connection Objects

Allowable property names and returned values for database connection objects appear in the following table.

Property	Value	
'AutoCommit'	Status of the AutoCommit flag. It is either on or off, as specified by set	
'Catalog'	Name of the catalog in the data source. You may need to extract a single catalog name from 'Catalog' for functions such as columns, which accept only a single catalog.	
'Driver'	Driver used for a JDBC connection, as specified by database	
'Handle'	Identifies a JDBC connection object	
'Instance'	Name of the data source for an ODBC connection or the name of a database for a JDBC connection, as specified by database	
'Message'	Error message returned by database	
'ReadOnly'	1 if the database is read-only; 0 if the database is writable	
'TimeOut'	Value for LoginTimeout	
'TransactionIsolation'	Value of current transaction isolation mode	
'Type'	Object type, specifically Database Object	
'URL'	For JDBC connections only, the JDBC URL objectjdbc:subprotocol:subname, as specified by database	
'UserName'	User name required to connect to a given database, as specified by database	
'Warnings'	Warnings returned by database	

Cursor Objects

Allowable property names and returned values for cursor objects appear in the following table.

Property	Value	
'Attributes'	Cursor attributes. This field is always empty. Use the attr function to retrieve cursor attributes.	
'Data'	Data in the cursor object data element (the query results)	
'DatabaseObject'	Information about a given database object	
'RowLimit'	Maximum number of rows returned by fetch, as specified by set	
'SQLQuery'	SQL statement for a cursor, as specified by exec	
'Message'	Error message returned from exec or fetch	
'Type'	Object type, specifically Database Cursor Object	
'ResultSet'	Resultset object identifier	
'Cursor'	Cursor object identifier	
'Statement'	Statement object identifier	
	Note If you specify a value (in seconds) for the timeout argument, queries time out after the time exceeds the given value.	
'Fetch'	O for cursor created using exec; fetchTheData for cursor created using fetch	

Driver Objects

Allowable property names and examples of values for driver objects appear in the following table.

Property	Example of Value
'MajorVersion'	1
'MinorVersion'	1001

Database Metadata Objects

Database metadata objects have many properties. Some allowable property names and examples of their values appear in the following table.

Property	Example of Value
'Catalogs'	{4x1 cell}
'DatabaseProductName'	'ACCESS'
'DatabaseProductVersion'	'03.50.0000'
'DriverName'	'JDBC-ODBC Bridge (odbcjt32.dll)'
'MaxColumnNameLength'	64
'MaxColumnsInOrderBy'	10
'URL'	'jdbc:odbc:dbtoolboxdemo'
'NullsAreSortedLow'	1

Drivermanager Objects

Allowable property names and examples of values for drivermanager objects appear in the following table.

Property	Example of Value	
'Drivers'	{'oracle.jdbc.driver.OracleDriver@1d8e09ef' [1x37 char]}	
'LoginTimeout'	0	
'LogStream'	[]	

Resultset Objects

Allowable property names and examples of values for resultset objects appear in the following table.

Property	Example of Value
'CursorName'	{'SQL_CUR92535700x' 'SQL_CUR92535700x'}
'MetaData'	{1x2 cell}
'Warnings'	{[] []}

Resultset Metadata Objects

Allowable property names and examples of values for a resultset metadata objects appear in the following table.

Property	Example of Value	
'CatalogName'	{'''}	
'ColumnCount'	2	
'ColumnName'	{'Calc_Date' 'Avg_Cost'}	
'ColumnTypeName'	{'TEXT' 'LONG'}	
'TableName'	{''''}	
'isNullable'	{[1] [1]}	
'isReadOnly'	{[0] [0]}	

The empty strings for CatalogName and TableName indicate that databases do not return these values.

For command-line help on get, use the overloaded methods:

```
help cursor/get
help database/get
help dmd/get
help driver/get
help drivermanager/get
help resultset/get
help rsmd/get
```

Examples

Example 1 — Get Connection Property and Data Source Name

Connect to the database SampleDB, and then get the name of the data source for the connection and assign it to v.

```
conn = database('SampleDB', '', '');
v = get(conn, 'Instance')
```

Example 2 — Get Connection Property and AutoCommit Flag Status

Check the status of the ${\tt AutoCommit}$ flag for the database connection conn.

```
get(conn, 'AutoCommit')
ans =
  on
```

Example 3 — Display Data in Cursor

Display data in the cursor object curs by running:

```
get(curs, 'Data')
or:
    curs.Data
    ans =
        'Germany'
        'Mexico'
        'France'
        'Canada'
```

Example 4 — Get Database Metadata Object Properties

1 View the properties of the database metadata object for connection conn; due to space constraints, only a portion of the returned data appears here.

```
dbmeta = dmd(conn);
v = get(dbmeta)
v =
           AllProceduresAreCallable: 1
             AllTablesAreSelectable: 1
    DataDefinitionCausesTransaction: 1
    DataDefinitionIgnoredInTransact: 0
         DoesMaxRowSizeIncludeBlobs: 0
                           Catalogs: {4x1 cell}
              NullPlusNonNullIsNull: 0
                NullsAreSortedAtEnd: 0
              NullsAreSortedAtStart: 0
                 NullsAreSortedHigh: 0
                  NullsAreSortedLow: 1
              UsesLocalFilePerTable: 0
                     UsesLocalFiles: 1
```

2 To view names of the catalogs in the database, run:

```
v.Catalogs
ans =
    'D:\matlab\toolbox\database\dbdemos\db1'
    'D:\matlab\toolbox\database\dbdemos\origtutorial'
    'D:\matlab\toolbox\database\dbdemos\tutorial'
    'D:\matlab\toolbox\database\dbdemos\tutorial1'
```

See Also

columns, cursor.fetch, database, dmd, driver, drivermanager, exec, getdatasources, resultset, rows, rsmd, set

Purpose

Return names of ODBC and JDBC data sources on system

Syntax

d = getdatasources

Description

d = getdatasources returns the names of valid ODBC and JDBC data sources on the system as a cell array d of strings. The function gets the names of ODBC data sources from the ODBC.INI file located in the directory returned by running:

```
myODBCdir = getenv('WINDIR')
```

d is empty when the ODBC.INI file is valid, but no data sources are defined. d equals -1 when the ODBC.INI file cannot be opened.

The function also retrieves the names of data sources that are in the system registry but not in the ODBC. INI file.

If you do not have write access to myODBCdir, the results of getdatasources may not include data sources that you recently added. In this case, specify a temporary, writable, output directory via the preference TempDirForRegistryOutput. For more information about this preference, see setdbprefs.

getdatasources gets the names of JDBC data sources from the file that you define using setdbprefs or the Define JDBC data sources dialog box.

Examples

Get the names of databases on your system.

```
d = getdatasources
```

d =

'MS Access Database' 'SampleDB' 'dbtoolboxdemo'

See Also

database, get, setdbprefs

importedkeys

Purpose

Return information about imported foreign keys

Syntax

```
i = importedkeys(dbmeta, 'cata', 'sch')
i = importedkeys(dbmeta, 'cata', 'sch', 'tab')
```

Description

- i = importedkeys(dbmeta, 'cata', 'sch') returns foreign imported key information, that is, information about fields that reference primary keys in other tables, in:
 - The schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta
- i = importedkeys(dbmeta, 'cata', 'sch', 'tab') returns foreign imported key information, that is, information about fields in The table tab. In turn, fields in tab reference primary keys in other tables in:
 - The schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta

Examples

Get foreign key information for the schema SCOTT in the catalog orcl, for dbmeta.

```
i = importedkeys(dbmeta, 'orcl', 'SCOTT')
  Columns 1 through 7
                                               'orcl' ...
    'orcl'
              'SCOTT'
                         'DEPT'
                                   'DEPTNO'
    'SCOTT'
                'EMP'
  Columns 8 through 13
                 111
                                           'FK DEPTNO'...
    'DEPTNO'
                         'null'
                                    '1'
    'PK DEPT'
```

The results show foreign imported key information as described in the following table.

Column	Description	Value
1	Catalog containing primary key, referenced by foreign imported key	orcl
2	Schema containing primary key, referenced by foreign imported key	SCOTT
3	Table containing primary key, referenced by foreign imported key	DEPT
4	Column name of primary key, referenced by foreign imported key	DEPTNO
5	Catalog that has foreign imported key	orcl
6	Schema that has foreign imported key	SCOTT
7	Table that has foreign imported key	EMP
8	Foreign key column name, that is the column name that references the primary key in another table	DEPTNO
9	Sequence number within foreign key	1
10	Update rule, that is, what happens to the foreign key when the primary key updates	null
11	Delete rule, that is, what happens to the foreign key when the primary key is deleted	1
12	Foreign imported key name	FK_DEPTNO
13	Primary key name in referenced table	PK_DEPT

In the schema SCOTT, there is only one foreign imported key. The table EMP contains a field, DEPTNO, that references the primary key in the DEPT table, the DEPTNO field.

EMP is the referencing table and DEPT is the referenced table.

<code>DEPTNO</code> is a foreign imported key in the EMP table. Reciprocally, the <code>DEPTNO</code> field in the table <code>DEPT</code> is an exported foreign key and the primary key.

importedkeys

For a description of the codes for update and delete rules, see the getImportedKeys property on the Sun Java Web site at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html.

See Also

crossreference, dmd, exportedkeys, get, primarykeys

Purpose

Return indices and statistics for database tables

Syntax

```
x = indexinfo(dbmeta, 'cata', 'sch', 'tab')
```

Description

x = indexinfo(dbmeta, 'cata', 'sch', 'tab') returns indices and statistics for:

- The table tab
- In the schema sch
- Of the catalog cata
- for the database whose database metadata object is dbmeta

Examples

Get index and statistics information for the table DEPT in the schema SCOTT of the catalog orcl, for dbmeta.

```
x = indexinfo(dbmeta,'','SCOTT','DEPT')
x =
 Columns 1 through 8
 'orcl' 'SCOTT' 'DEPT'
                         '0'
                             'null'
                                      'null'
 'orcl' 'SCOTT' 'DEPT'
                         '0'
                             'null'
                                      'PK DEPT' '1'
  Columns 9 through 13
  'null'
           'null'
                    '4'
                        '1'
                             'null'
  'DEPTNO'
           'null'
                   '4'
                        '1' 'null'
```

The results contain two rows, meaning there are two index columns. The statistics for the first index column appear in the following table.

Column	Description	Value
1	Catalog	orcl
2	Schema	SCOTT

indexinfo

Column	Description	Value
3	Table	DEPT
4	Non-unique: 0 if index values can be non-unique, 1 otherwise	0
5	Index catalog	null
6	Index name	null
7	Index type	0
8	Column sequence number within index	0
9	Column name	null
10	Column sort sequence	null
11	Number of rows in the index table or number of unique values in the index	4
12	Number of pages used for the table or number of pages used for the current index	1
13	Filter condition	null

For more information about the index information, see the getIndexInfo property on the Sun Java Web site at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html.

See Also dmd, get, tables

insert

Purpose Add MATLAB data to database tables (deprecated; use fastinsert

instead)

Syntax insert(conn, 'tab', colnames, exdata)

Description insert(conn, 'tab', colnames, exdata)

The fastinsert function has replaced the insert function. fastinsert offers improved performance and supports more data types than insert.

Use insert if fastinsert does not work as expected, especially if you

have used insert successfully in the past.

The insert function uses the same syntax as fastinsert; for details,

see fastinsert.

See Also commit, fastinsert, querybuilder, rollback

isconnection

Purpose Detect whether database connections are valid

Syntax a = isconnection(conn)

Description a = isconnection(conn) returns 1 if the database connection conn

is valid, or returns 0 otherwise.

Examples Check if the database connection conn is valid.

a = isconnection(conn)

a =

1

See Also database, isreadonly, ping

Purpose Detect whether driver is a valid JDBC driver object

Syntax a = isdriver(d)

Description a = isdriver(d) returns 1 if d is a valid JDBC driver object. It returns

0 otherwise.

Examples Check if d is a valid JDBC driver object.

a = isdriver(d)
a =
1

See Also driver, get, isjdbc, isurl

isjdbc

Purpose Detect whether driver is JDBC compliant

Syntax a = isjdbc(d)

Description a = isjdbc(d) returns 1 if the driver object d is JDBC-compliant. It

returns 0 otherwise.

Examples Verify whether the database driver object d is JDBC compliant.

a = isjdbc(d)

a =

1

See Also driver, get, isdriver, isurl

Purpose

Detect whether last record read in resultset is NULL

Syntax

```
a = isnullcolumn(rset)
```

Description

a = isnullcolumn(rset) returns 1 if the last record read in the resultset rset is NULL. It returns 0 otherwise.

Examples

Example 1 — Result Is Not NULL

isnullcolumn returns not null.

1 Run:

```
curs = fetch(curs,1);
rset = resultset(curs);
isnullcolumn(rset)
ans =
    0
```

2 Verify this result.

```
curs.Data
ans =
[1400]
```

Example 2 - Result Is NULL

isnullcolumn returns null.

1 Run:

```
curs = fetch(curs,1);
rset = resultset(curs);
isnullcolumn(rset)
ans =
    1
```

2 Verify this result.

isnullcolumn

curs.Data
ans =
 [NaN]

See Also get, resultset

Purpose Detect whether database connection is read-only

Syntax a = isreadonly(conn)

Description a = isreadonly(conn) returns 1 if the database connection conn is

read-only. It returns 0 otherwise.

Examples Check whether conn is read-only.

a = isreadonly(conn)

The result indicates that the database connection conn is read-only:

a =

1

Therefore, you cannot run fastinsert, insert, or update functions on this database.

See Also database, isconnection

Purpose Detect whether database URL is valid

Description a = isurl('s', d) returns 1 if the database URL s for the driver

object d is valid. It returns 0 otherwise.

The URL s is of the form jdbc:odbc:name or name.

Examples Check whether the database URL,

jdbc:odbc:thin:@144.212.123.24:1822: is valid for driver object d.

a = isurl('jdbc:odbc:thin:@144.212.123.24:1822:', d)
a =
 1

This indicates that the database URL is valid for d.

See Also driver, get, isdriver, isjdbc

Purpose

Set or get time allowed to establish database connection

Syntax

```
timeout = logintimeout('driver', time)
```

timeout = logintimeout(time)
timeout = logintimeout('driver')

timeout = logintimeout

Description

- timeout = logintimeout('driver', time) sets the amount of time, in seconds, for a MATLAB software session to connect to a database via a given JDBC driver. Use logintimeout before running the database function. If the MATLAB software session cannot connect to the database within the specified time, it stops trying.
- timeout = logintimeout(time) sets the amount of time, in seconds, allowed for a MATLAB software session to try to connect to a database via an ODBC connection. Use logintimeout before running the database function. If the MATLAB software session cannot connect within the allowed time, it stops trying.
- timeout = logintimeout('driver') returns the time, in seconds, that was previously specified for the JDBC driver. A returned value of 0 means that the timeout value was not previously set. The MATLAB software session stops trying to connect to the database if it is not immediately successful.
- timeout = logintimeout returns the time, in seconds, that you previously specified for an ODBC connection. A returned value of 0 means that the timeout value was not previously set; the MATLAB software session stops trying to make a connection if it is not immediately successful.

Note If you do not specify a value for logintimeout and the MATLAB software session cannot establish a database connection, your MATLAB software session may freeze.

Note $Apple^{\mathbb{R}}$ $Mac\ OS^{\mathbb{R}}$ platforms do not support logintimeout.

Examples

Example 1 — Get Timeout Value for ODBC Connection

View the current connection timeout value.

```
logintimeout
ans =
    0
```

This indicates that you have not specified a timeout value.

Example 2 — Set Timeout Value for ODBC Connection

Set the timeout value to 5 seconds.

```
logintimeout(5)
ans =
    5
```

Example 3 — Get and Set Timeout Value for JDBC Connection

1 Check the timeout value for a database connection that is established using an Oracle JDBC driver.

```
logintimeout('oracle.jdbc.driver.OracleDriver')
ans =
    0
```

This indicates that the timeout value is currently 0.

2 Set the timeout to 5 seconds.

```
timeout = ...
logintimeout('oracle.jdbc.driver.OracleDriver', 5)
timeout =
    5
```

3 Verify the timeout value.

```
logintimeout('oracle.jdbc.driver.OracleDriver')
ans =
    5
```

See Also database, get, set

namecolumn

Purpose

Map resultset column name to resultset column index

Syntax

x = namecolumn(rset, n)

Description

x = namecolumn(rset, n) maps a resultset column name n to its resultset column index. rset is the resultset and n is a string or cell array of strings containing the column names.

Examples

1 Get the indices for the column names DNAME and LOC resultset object rset.

```
x = namecolumn(rset, {'DNAME';'LOC'})
x =
2 3
```

The results show that DNAME is column 2 and LOC is column 3.

2 Get the index for only the LOC column.

```
x = namecolumn(rset, 'LOC')
```

See Also

columnnames, resultset

Purpose Get status information about database connection

Syntax ping(conn)

Description ping(conn) returns status information about the database connection

conn if the connection is open. It returns an error message otherwise.

Examples Example 1 — Get Status Information About ODBC Connection

Check the status of the ODBC connection conn.

```
ping(conn)
ans =
    DatabaseProductName: 'ACCESS'
DatabaseProductVersion: '03.50.0000'
    JDBCDriverName: 'JDBC-ODBC Bridge (odbcjt32.dll)'
    JDBCDriverVersion: '1.1001 (04.00.4202)'
MaxDatabaseConnections: 64
    CurrentUserName: 'admin'
    DatabaseURL: 'jdbc:odbc:SampleDB'
AutoCommitTransactions: 'True'
```

Example 2 — Get Status Information About JDBC Connection

Check the status of the JDBC connection conn.

Example 3 — Unsuccessful Request for Information About Connection

In this example, the database connection conn has been terminated or is not successful. Run:

```
ping(conn)
Cannot Ping the Database Connection
```

See Also

database, dmd, get, isconnection, set, supports

Purpose

Get primary key information for database table or schema

Syntax

```
k = primarykeys(dbmeta, 'cata', 'sch')
k = primarykeys(dbmeta, 'cata', 'sch', 'tab')
```

Description

- k = primarykeys(dbmeta, 'cata', 'sch') returns primary key information for all tables in:
 - The schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta
- k = primarykeys(dbmeta, 'cata', 'sch', 'tab') returns primary key information for:
 - The table tab
 - In the schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta

Examples

Get primary key information for the DEPT table in:

- The schema SCOTT
- Of the catalog orcl
- For the database metadata object dbmeta

```
k = primarykeys(dbmeta, 'orcl', 'SCOTT', 'DEPT')
k =
    'orcl' 'SCOTT' 'DEPT' 'DEPTNO' '1' 'PK_DEPT'
```

primarykeys

The results show the primary key information as described in the following table.

Column	Description	Value
1	Catalog	orcl
2	Schema	SCOTT
3	Table	DEPT
4	Column name of primary key	DEPTNO
5	Sequence number within primary key	1
6	Primary key name	PK_DEPT

See Also

crossreference, dmd, exportedkeys, get, importedkeys

procedurecolumns

Purpose

Get stored procedure parameters and result columns of catalogs

Syntax

```
pc = procedurecolumns(dbmeta, 'cata', 'sch')
pc = procedurecolumns(dbmeta, 'cata')
```

Description

- pc = procedurecolumns(dbmeta, 'cata', 'sch') returns the stored procedure parameters and result columns for:
 - The schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta
- pc = procedurecolumns(dbmeta, 'cata') returns stored procedure parameters and result columns for:
 - The catalog cata
 - For the database whose database metadata object is dbmeta

Running the stored procedure generates results. One row is returned for each column.

Examples

Get stored procedure parameters for:

Columns 8 through 13

- The schema ORG
- In the catalog tutorial
- For the database metadata object dbmeta.

```
pc = procedurecolumns(dbmeta,'tutorial', 'ORG')
pc =
  Columns 1 through 7
  [1x19 char] 'ORG' 'display' 'Month' '3' ...
  '12' 'TEXT'
  [1x19 char] 'ORG' 'display' 'Day' '3' ...
  '4' 'INTEGER'
```

procedurecolumns

```
'50' '50' 'null' 'null' '1' 'null' '50' '4' 'null' 'null' '1' 'null'
```

The results show stored procedure parameter and result information. Because two rows of data are returned, there are two columns of data in the results. The results show that running the stored procedure display returns the Month and Day columns.

procedurecolumns

Following is a full description of the procedurecolumns results for the first row (Month).

Column	Description	Value for First Row
1	Catalog	'D:\orgdatabase\orcl'
2	Schema	'ORG'
3	Procedure name	'display'
4	Column/parameter name	'MONTH'
5	Column/parameter type	'3'
6	SQL data type	'12'
7	SQL data type name	'TEXT'
8	Precision	'50'
9	Length	'50'
10	Scale	'null'
11	Radix	'null'
12	Nullable	'1'
13	Remarks	'null'

For more information about the procedurecolumns results, see the getProcedureColumns property on the Sun Java Web site at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html.

See Also dmd, get, procedures

Purpose

Get stored procedures for catalogs

Syntax

```
p = procedures(dbmeta, 'cata')
p = procedures(dbmeta, 'cata', 'sch')
```

Description

- p = procedures(dbmeta, 'cata') returns stored procedures in the catalog cata, for the database whose database metadata object is dbmeta.
- p = procedures(dbmeta, 'cata', 'sch') returns the stored procedures in:
 - The schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta

Stored procedures are SQL statements that are saved with the database. Use the exec function to run a stored procedure. Specify the stored procedure as the sqlquery argument instead of explicitly entering the sqlquery statement as the argument.

Examples

1 Get the names of stored procedures for the catalog DBA, for the database metadata object dbmeta.

```
p = procedures(dbmeta, 'DBA')
p =
          'sp_contacts'
          'sp_customer_list'
          'sp_customer_products'
          'sp_product_info'
          'sp_retrieve_contacts'
          'sp_sales_order'
```

2

a Execute the stored procedure sp_customer_list for the database connection conn, and fetch all data.

```
curs = exec(conn, 'sp_customer_list');
curs = fetch(conn)
curs =
     Attributes: []
           Data: {10x2 cell}
 DatabaseObject: [1x1 database]
       RowLimit: 0
       SQLQuery: 'sp_customer_list'
        Message: []
           Type: 'Database Cursor Object'
      ResultSet: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]
         Cursor: ...
   [1x1 com.mathworks.toolbox.database.sqlExec]
      Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
          Fetch: ...
   [1x1 com.mathworks.toolbox.database.fetchTheData]
```

b View the results.

```
curs.Data
ans =
              'The Power Group'
    [101]
    [102]
              'AMF Corp.'
              'Darling Associates'
    [103]
    [104]
              'P.S.C.'
              'Amo & Sons'
    [105]
              'Ralston Inc.'
    [106]
              'The Home Club'
    [107]
    [108]
              'Raleigh Co.'
    [109]
              'Newton Ent.'
              'The Pep Squad'
    [110]
```

See Also

dmd, exec, get, procedurecolumns

querybuilder

Purpose Start SQL query builder GUI to import and export data

Syntax querybuilder

Description querybuilder starts Visual Query Builder (VQB), the Database

Toolbox GUI.

 $\label{eq:Note_to_populate_the_VQB} \textbf{Schema} \ \text{and} \ \textbf{Catalog} \ \text{fields, you must} \\ \text{associate your user name with schemas or catalogs before starting VQB}.$

Examples For more information on Visual Query Builder, including examples, see

the VQB **Help** menu or Chapter 4, "Using Visual Query Builder".

Purpose Get time specified for SQL queries to succeed

Syntax timeout = querytimeout(curs)

Description timeout = querytimeout(curs) returns the amount of time, in

seconds, allowed for SQL queries of the open cursor curs to succeed. If a given query cannot complete in the specified time, the toolbox stops

trying to perform the query.

The database administrator defines timeout values. If the timeout

value is zero, queries must complete immediately.

Examples Get the current database timeout setting for curs.

querytimeout(curs)
ans =
 10

Limitations

• If a given database does not have a database timeout feature, it returns the following:

[Driver]Driver not capable

• ODBC drivers for Microsoft Access and Oracle do not support querytimeout.

See Also exec

register

Purpose Load database driver

Syntax register(d)

Description register (d) loads the database driver object d. Use unregister to

unload the driver.

Although database automatically loads a driver, register allows you to use get to view properties of the driver before connecting to the database. The register function also allows you to run drivermanager

with set and get on properties for loaded drivers.

Examples 1 register(d) loads the database driver object d.

2 get(d) returns properties of the driver object.

See Also driver, drivermanager, get, set, unregister

Purpose Construct resultset object

Syntax rset = resultset(curs)

Description rset = resultset(curs) creates a resultset object rset for the cursor

curs. To get properties of rset, create a resultset metadata object using rsmd, or make calls to rset using applications based on Sun Java.

Run clearwarnings, isnullcolumn, and namecolumn on rset. Use

close to close the resultset, which frees up resources.

Examples Construct a resultset object rset.

rset = resultset(curs)

rset =

Handle: [1x1 sun.jdbc.odbc.JdbcOdbcResultSet]

See Also clearwarnings, close, cursor.fetch, exec, get, isnullcolumn,

namecolumn, rsmd

Undo database changes

Syntax

rollback(conn)

Description

rollback(conn) reverses changes made to a database using fastinsert, insert, or update via the database connection conn. The rollback function reverses all changes made since the last commit or rollback operation. To use rollback, the AutoCommit flag for conn must be off.

Note The rollback function does not roll back data in MySQL databases.

Examples

1 Ensure that the AutoCommit flag for connection conn is off by running:

```
get(conn,'AutoCommit')
ans =
  off
```

2 Insert data contained in exdata into the columns DEPTNO, DNAME, and LOC, in the table DEPT, for the data source conn.

```
fastinsert(conn, 'DEPT', ...
{'DEPTNO';'DNAME';'LOC'}, exdata)
```

3 Roll back the data that you inserted into the database by running:

```
rollback(conn)
```

The data in exdata is removed from the database. The database now contains the data it had before you ran the fastinsert function.

See Also

commit, database, exec, fastinsert, get, insert, update

Return number of rows in fetched data set

Syntax

numrows = rows(curs)

Description

numrows = rows(curs) returns the number of rows in the fetched data set curs, where curs has been generated by the cursor.fetch function.

Examples

There are four rows in the fetched data set curs.

```
numrows = rows(curs)
numrows =
4
```

To see the four rows of data in curs, run:

```
curs.Data
ans =
    'Germany'
    'Mexico'
    'France'
    'Canada'
```

See Also

cols, cursor.fetch, get, rsmd

rsmd

Purpose Construct resultset metadata object

Syntax rsmeta = rsmd(rset)

Description rsmeta = rsmd(rset) creates a resultset metadata object rsmeta, for

the resultset object rset. Get properties of rsmeta using get or make

calls to rsmeta using applications that are based on Sun Java.

Examples Create a resultset metadata object rsmeta.

rsmeta=rsmd(rset)

rsmeta =

Handle: [1x1 sun.jdbc.odbc.JdbcOdbcResultSetMetaData]

Use v = get(rsmeta) and v.property to view properties of the

resultset metadata object.

See Also exec, get, resultset

Call stored procedure with input and output parameters

Syntax

results = runstoredprocedure(conn, sp_name, parms_in,
 types out)

Description

results = runstoredprocedure(conn, sp_name, parms_in, types_out) calls a stored procedure with specified input parameters and returns output parameters, where:

- conn is the database connection handle
- sp name is the stored procedure to run
- parms_in is a cell array containing the input parameters for the stored procedure
- types out is the list of data types of the output parameters

Use runstoredprocedure to return the value of a variable to a MATLAB variable, which you cannot do when running a stored procedure via exec. Running a stored procedure via exec returns resultsets but cannot return output parameters.

Examples

These examples illustrate how runstoredprocedure differs from running stored procedures via exec.

1 The command:

```
x = runstoredprocedure(c, 'myprocnoparams')
```

Runs a stored procedure that has no input or output parameters.

2 The command:

```
x = runstoredprocedure(c, 'myprocinonly', {2500, 'Jones'})
```

Runs a stored procedure given input parameters 2500 and 'Jones'. It returns no output parameters.

runstoredprocedure

3 The command:

```
x = runstoredprocedure(c, 'myproc', {2500, 'Jones'}, {java.sql.Types.NUMERIC})
```

Runs the stored procedure myproc given input parameters 2500 and 'Jones'. It returns an output parameter of type java.sql.Types.NUMERIC, which could be any numeric Sun Java data type. The output parameter x is the value of a database variable n. The stored procedure myproc creates this variable, given the input values 2500 and 'Jones'. For example, myproc computes n, the number of days when Jones is 2500. It then returns the value of n to x.

See Also

cursor.fetch, exec

Set properties for database, cursor, or drivermanager object

Syntax

set(object, 'property', value)
set(object)

Description

- set(object, 'property', value) sets the value of property to value for the specified object.
- set(object) displays all properties for object.

Allowable values for object are:

- "Database Connection Objects" on page 7-102, created using database
- "Cursor Objects" on page 7-103, created using exec or fetch (cursor.fetch)
- "Drivermanager Objects" on page 7-103, created using drivermanager.

You cannot set all of these properties for all databases. You receive an error message when you try to set a property that the database does not support.

Database Connection Objects

The allowable values for *property* and value for a database connection object appear in the following table.

Property	Value	Description
'AutoCommit'	'on'	Database data is written and automatically committed when you run fastinsert, insert, or exec. You cannot use rollback to reverse this process.
	'off'	Database data is not committed automatically when you run fastinsert, insert, or update. Use rollback to reverse this process. When you are sure that your data is correct, use the committunction to commit it to the database.
'ReadOnly'	0	Not read-only; that is, writable
	1	Read-only
'TransactionIsolation'	positive integer	Current transaction isolation level

Note For some databases, if you insert data and then close the database connection without having committed the data to the database, the data gets committed automatically. Your database administrator can tell you whether your database behaves this way.

Cursor Objects

The allowable *property* and value for a cursor object appear in the following table.

Property	Value	Description
'RowLimit'	positive integer	Sets the RowLimit for fetch. Specify this property instead of passing RowLimit as an argument to the fetch function. When you define RowLimit forfetch by using set, fetch behaves differently depending on what type of database you are using.

Drivermanager Objects

The allowable *property* and value for a drivermanager object appear in the following table.

Property	Value	Description
'LoginTimeout'	positive integer	Sets the logintimeout value for all loaded database drivers.

For command-line help on set, use the overloaded methods:

help cursor/set help database/set help drivermanager/set

Examples

Example 1 — Set RowLimit for Cursor

This example does the following:

- Establishes a JDBC connection to a data source
- Runs fetch to retrieve data from the table EMP,

• Sets RowLimit to 5

Run the command:

```
conn=database('orcl','scott','tiger',...
 'oracle.jdbc.driver.OracleDriver',...
 'jdbc:oracle:thin:@144.212.123.24:1822:');
curs=exec(conn, 'select * from EMP');
set(curs, 'RowLimit', 5)
curs=fetch(curs)
curs =
    Attributes: []
           Data: {5x8 cell}
 DatabaseObject: [1x1 database]
       RowLimit: 5
       SQLQuery: 'select * from EMP'
       Message: []
           Type: 'Database Cursor Object'
      ResultSet: [1x1 oracle.jdbc.driver.OracleResultSet]
         Cursor: [1x1 com.mathworks.toolbox.database.sqlExec]
     Statement: [1x1 oracle.jdbc.driver.OracleStatement]
          Fetch: [1x1 com.mathworks.toolbox.database.fetchTheData]
```

As displayed above, the RowLimit property of curs is 5 and the Data property is 5x8 cell, indicating that fetch returned five rows of data.

In this example, RowLimit limits the maximum number of rows you can retrieve. Therefore, rerunning the fetch function returns no data.

Example 2 — Set the AutoCommit Flag to On

This example shows what happens when you run a database update function on a database whose AutoCommit flag is set to on.

1 Determine the status of the AutoCommit flag for the database connection conn.

```
get(conn, 'AutoCommit')
ans =
off
```

The flag is off.

2 Set the flag status to on and verify its value.

```
set(conn, 'AutoCommit', 'on');
get(conn, 'AutoCommit')
ans =
on
```

3 Insert a cell array exdata into column names colnames in the table Growth.

```
fastinsert(conn, 'Growth', colnames, exdata)
```

The data is inserted and committed to the database.

Example 3 — Set the AutoCommit Flag to Off and Commit Data

This example shows the results of running fastinsert and commit to insert and commit data into a database whose AutoCommit flag is off.

1 First set the AutoCommit flag to off for database connection conn.

```
set(conn, 'AutoCommit', 'off');
```

2 Insert a cell array exdata into the column names colnames in the table Avg_Freight_Cost.

```
fastinsert(conn, 'Avg_Freight_Cost', colnames, exdata)
```

3 Commit the data to the database.

```
commit(conn)
```

Example 4 — Set the AutoCommit Flag to Off and Roll Back Data

This example runs update to insert data into a database whose AutoCommit flag is off. It then uses rollback to roll back the data.

1 Set the AutoCommit flag to off for database connection conn.

```
set(conn, 'AutoCommit', 'off');
```

2 Update the data in colnames in the table Avg_Freight_Weight table, for the record selected by whereclause, with data from the cell array exdata.

```
update(conn, 'Avg_Freight_Weight', colnames, exdata,
whereclause)
```

3 Roll back the data.

```
rollback(conn)
```

The data in the table is now as it was before you ran update.

Example 5 — Set the LoginTimeout for a Drivermanager Object

1 Create a drivermanager object dm and set its LoginTimeout value to 3 seconds.

```
dm = drivermanager;
    set(dm,'LoginTimeout',3);

2 Verify this result.
    logintimeout
    ans =
        3
```

See Also

cursor.fetch, database, drivermanager, exec, fastinsert, get, insert, logintimeout, ping, update

setdbprefs

Purpose

Set preferences for retrieval format, errors, NULLs, and more

GUI Alternatives

Click **Query > Preferences** to set database preferences from Visual Query Builder.

Syntax

```
setdbprefs
s = setdbprefs
setdbprefs('property')
setdbprefs('property', 'value')
setdbprefs({'property1'; ...}, {'value1'; ...}
setdbprefs(s)
```

Description

- setdbprefs returns current values for database preferences.
- s = setdbprefs returns current values for database preferences to the structure s. You can save s to a MAT-file to use your specified preferences in future MATLAB software sessions.
- setdbprefs('property') returns the current value for the specified property.
- setdbprefs('property', 'value') sets the specified property to value in the current MATLAB software session. You can include this statement in a MATLAB startup file to set preferences automatically when a MATLAB software session starts.
- setdbprefs({'property1'; ...}, {'value1'; ...} sets properties starting with property1 to values starting with value1, in the current MATLAB software session.
- setdbprefs(s) sets preferences specified in the structure s to values that you specify.

Allowable properties appear in the following tables:

• DataReturnFormat and ErrorHandling Properties and Values for setdbprefs on page 7-109

- Null Data Handling Properties and Values for setdbprefs on page 7-110
- Other Properties and Values for setdbprefs (Not Accessible via Query > Preferences) on page 7-112

DataReturnFormat and ErrorHandling Properties and Values for setdbprefs

Property	Allowable Values	Description
'DataReturnFormat'	'cellarray' (default), 'numeric', or 'structure'	Format for data to import into the MATLAB workspace. Set the format based on the type of data being retrieved, memory considerations, and your preferred method of working with retrieved data.
	'cellarray' (default)	Imports nonnumeric data into MATLAB cell arrays.
	'numeric'	Imports data into MATLAB matrix of doubles. Nonnumeric data types are considered NULL and appear as specified in the NullNumberRead property. Use only when data to retrieve is in numeric format, or when nonnumeric data to retrieve is not relevant.
	'structure'	Imports data into a MATLAB structure. Use for all data types. Facilitates working with returned columns.

setdbprefs

DataReturnFormat and ErrorHandling Properties and Values for setdbprefs (Continued)

Property	Allowable Values	Description
'ErrorHandling'	'store' (default), 'report', or 'empty'	Specifies how to handle errors when importing data. Set this parameter before you run exec.
	'store' (default)	Errors from running database are stored in the Message field of the returned connection object. Errors from running exec are stored in the Message field of the returned cursor object.
	'report'	Errors from running database or exec display immediately in the MATLAB Command Window.
	'empty'	Errors from running database are stored in the Message field of the returned connection object. Errors from running exec are stored in the Message field of the returned cursor object. Objects that cannot be created are returned as empty handles ([]]).

Null Data Handling Properties and Values for setdbprefs

Property	Allowable Values	Description
'NullNumberRead'	User-specified, for example, '0'	Specifies how NULL numbers appear after being imported from a database into the MATLAB workspace. NaN is the default value. String values such as 'NULL' cannot be set if 'DataReturnFormat' is set to 'numeric'. Set this parameter before running fetch.

Null Data Handling Properties and Values for setdbprefs (Continued)

Property	Allowable Values	Description
'NullNumberWrite'	User-specified, for example, 'NaN' (default)	Numbers in the specified format, for example, NaN appears as NULL after being exported from the MATLAB workspace to a database.
'NullStringRead'	User-specified, for example, 'NULL' (default)	Specifies how NULL strings appear after being imported from a database into the MATLAB workspace. Set this parameter before running fetch.
'NullStringWrite'	User-specified, for example, 'NULL' (default)	Strings in the specified format, for example, NaN, appear as NULL after being exported from the MATLAB workspace to a database.

setdbprefs

Other Properties and Values for setdbprefs (Not Accessible via Query > Preferences)

Property	Allowable Values	Description
'JDBCDataSourceFile'	User-specified, for example, 'D:/file.mat'	Path to MAT-file containing JDBC data sources. For more information, see "Accessing Existing JDBC Data Sources" on page 2-4.
'UseRegistryForSources'	'yes' (default) or 'no'	When set to yes, VQB searches the Microsoft Windows registry for ODBC data sources that are not uncovered in the system ODBC. INI file. The following message may appear: Registry editing has been disabled by your administrator. This message is harmless and can safely be ignored.
'TempDirForRegistryOutput'	User-specified, for example, 'D:/work'	Directory where VQB writes ODBC registry settings when you run getdatasources. Use when you add data sources and do not have write access to the MATLAB current directory. The default is the Windows temporary directory, which is returned by the command getenv('temp')). If you specify a directory to which you do not have write access or which does not exist, the following error appears: Cannot export <directory-name>\ODBC.INI: Error opening the file. There may be a disk or file system error.</directory-name>

Remarks

When you run clear all, setdbprefs values are cleared and returned to their default values. It is a good practice to set or verify preferences values before each fetch.

Examples

Example 1 — Display Current Values

Run setdbprefs.

```
DataReturnFormat: 'cellarray'
ErrorHandling: 'store'
NullNumberRead: 'NaN'
NullNumberWrite: 'NULL'
NullStringRead: 'null'
NullStringWrite: 'null'
JDBCDataSourceFile: ''
UseRegistryForSources: 'yes'
TempDirForRegistryOutput: ''
```

These values show that:

- Data is imported from databases into MATLAB cell arrays.
- Errors that occur during a database connection or SQL query attempt are stored in the Message field of the connection or cursor data object.
- Each NULL number in the database is read into the MATLAB workspace as NaN. Each NaN in the MATLAB workspace is exported to the database as NULL. Each NULL string in the database is read into the MATLAB workspace as 'null'. Each 'null' string in the MATLAB workspace is exported to the database as a NULL string.
- A MAT-file that specifies the JDBC source file has not been created.
- Visual Query Builder looks in the Windows system registry for data sources that do not appear in the ODBC. INI file.
- No temporary directory for registry settings has been specified.

Example 2 — Change a Preference

```
Run setdbprefs ('NullNumberRead').

setdbprefs ('NullNumberRead')
NullNumberRead: 'NaN'
```

Each NULL number in the database is read into the MATLAB workspace as NaN.

Change the value of this preference to 0.

```
setdbprefs ('NullNumberRead', '0')
```

Each NULL number in the database is read into the MATLAB workspace as 0.

Example 3 — Change the DataReturnFormat Preference

1 Specify that database data be imported into MATLAB cell arrays.

```
setdbprefs ('DataReturnFormat', 'cellarray')
```

2 Import data into the MATLAB workspace.

```
conn = database('SampleDB', '', '');
curs=exec(conn, ...
  'select all ProductName,UnitsInStock fromProducts');
curs=fetch(curs,3);
curs.Data
ans =
   'Chai' [39]
   'Chang' [17]
   'Aniseed Syrup' [13]
```

3 Change the data return format from cellarray to numeric.

```
setdbprefs ('DataReturnFormat', 'numeric')
```

4 Perform the same import operation as you ran in the cell array example. Note the format of the returned data.

In the database, the values for ProductName are character strings, as seen in the previous example when DataReturnFormat was set to cellarray. Therefore, the ProductName values cannot be read when they are imported into the MATLAB workspace using the numeric format. Therefore, the MATLAB software treats them as NULL numbers and assigns them the current value for the NullNumberRead property of setdbprefs, NaN.

5 Change the data return format to structure.

```
setdbprefs ('DataReturnFormat','structure')
```

6 Then perform the same import operation as you ran in the cell array example.

```
curs.Data
ans =
    ProductName: {3x1 cell}
    UnitsInStock: [3x1 double]
```

7 View the contents of the structure to see the data.

```
curs.Data.ProductName
ans =
    'Chai'
    'Chang'
    'Aniseed Syrup'

curs.Data.UnitsInStock
```

setdbprefs

ans = 39 17 13

Example 4 — Change the Write Format for NULL Numbers

1 Specify NaN for the NullNumberWrite format.

```
setdbprefs('NullNumberWrite', 'NaN')
```

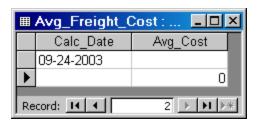
Numbers represented as NaN in the MATLAB workspace are exported to databases as NULL.

For example, the variable ex_data, contains a NaN.

```
ex_data = '09-24-2003' NaN
```

2 Insert ex_data into a database using fastinsert. TheNaN data is exported into the database as NULL.

fastinsert (conn, 'Avg_Freight_Cost', colnames, ex_data)



3 Change the value of NullNumberWrite to Inf.

```
setdbprefs('NullNumberWrite', 'Inf')
```

4 Attempt to insert ex_data. A MATLAB error appears because the NaN in ex_data cannot be read.

```
fastinsert(conn, 'Avg_Freight_Cost', colnames, ex_data
??? Error using ==> fastinsert
[Microsoft][ODBC Microsoft Access Driver]
Too few parameters.
Expected 1.
```

Example 5 — Specify Error Handling Settings

1 Specify the store format for the ErrorHandling preference.

```
setdbprefs ('ErrorHandling','store')
```

Errors generated from running database or exec are stored in the Message field of the returned connection or cursor object.

2 Now try to fetch data from a closed cursor by running:.

```
conn=database('SampleDB', '', '');
curs=exec(conn, 'select all ProductName from Products');
close(curs)
curs=fetch(curs,3);
curs=
        Attributes: []
              Data: 0
    DatabaseObject: [1x1 database]
          RowLimit: 0
          SQLQuery: 'select all ProductName from Products'
          Message: 'Error: Invalid cursor'
              Type: 'Database Cursor Object'
         ResultSet: 0
            Cursor: 0
         Statement: [1x1 sun.jdbc.odbc.JdbcOdbcStatement]
             Fetch: [1x1 com.mathworks.toolbox.database.fetchTheData]
```

The error generated by this operation appears in the Message field.

3 To specify the report format for the ErrorHandling preference, run:

```
setdbprefs ('ErrorHandling','report')
```

Errors generated by running database or exec display immediately in the Command Window.

4 Now try to fetch data from a closed cursor by running:

```
conn = database('SampleDB', '', '');
curs=exec(conn, 'select all ProductName from Products');
close(curs)
curs=fetch(curs,3);
??? Error using ==> cursor/fetch (errorhandling)
Invalid Cursor
Error in ==>
D:\matlab\toolbox\database\database\@cursor\fetch.m
On line 36 ==> errorhandling(initialCursor.Message);
```

The error generated by this operation appears immediately in the Command Window.

5 Specify the empty format for the ErrorHandling preference.

```
setdbprefs ('ErrorHandling','empty')
```

Errors generated while running database or exec are stored in the Message field of the returned connection or cursor object. In addition, objects that cannot be created are returned as empty handles, [].

6 Try to fetch data from a closed cursor.

```
conn = database('SampleDB', '', '');
curs=exec(conn, 'select all ProductName from Products');
close(curs)
curs=fetch(curs,3);
curs =
```

The error appears in the cursor object Message field. Furthermore, the Attributes field contains empty handles because no attributes could be created.

Example 6 - Change Multiple Settings

Specify that NULL strings are read from the database into a MATLAB matrix of doubles as 'NaN':

```
setdbprefs({'NullStringRead';'DataReturnFormat'},...
{'NaN';'numeric'})
```

See "Example 8 — Assign Values to a Structure" on page 7-120 for more information on another way to change multiple settings.

Example 7 — Specify JDBC Data Sources for Use by VQB

Instruct VQB to connect to the database using the data sources specified in the file myjdbcdatsources.mat.

```
setdbprefs('JDBCDataSourceFile',...
'D:/Work/myjdbcdatasources.mat')
```

.

Example 8 - Assign Values to a Structure

1 Assign values for preferences to fields in the structure s.

2 Set preferences using the values in **s**:

```
setdbprefs(s)
```

3 Runsetdbprefs to check your preferences settings:

```
setdbprefs
```

```
DataReturnFormat: 'numeric'
ErrorHandling: 'store'
NullNumberRead: '0'
NullNumberWrite: 'NaN'
NullStringRead: 'null'
NullStringWrite: 'null'
JDBCDataSourceFile: ''
UseRegistryForSources: 'yes'
TempDirForRegistryOutput: 'C:\Work'
```

Example 9 — Return Values to a Structure

Assign values for all preferences to s by running:

Now use the MATLAB tab completion feature when obtaining the value for a preference. For example, enter:

```
s.U
```

Press the **Tab** key, and then **Enter**. MATLAB completes the field and displays the value.

```
s.UseRegistryForSources
ans =
```

yes

Example 10 - Save Preferences

You can save your preferences to a MAT-file to use them in future MATLAB software sessions. For example, say that you need to reuse preferences that you set for the Seasonal Smoothing project. Assign the preferences to the variable SeasonalSmoothing and save them to a MAT-file SeasonalSmoothingPrefs in your current directory:

```
SeasonalSmoothing = setdbprefs;
save SeasonalSmoothingPrefs.mat SeasonalSmoothing
```

At a later time, load the data and restore the preferences:

```
load SeasonalSmoothingPrefs.mat
setdbprefs(SeasonalSmoothing);
```

Example 11 — Access Existing JDBC Data Sources

Use the following command to access an existing JDBC data source in future MATLAB software sessions:

```
setdbprefs('JDBCDataSourceFile','fullpathtomatfile')
```

For example, to use the data sources in the MAT-file D:/Work/myjdbcdatasources.mat, run this command in the MATLAB Command Window:

```
setdbprefs('JDBCDataSourceFile',...
'D:/Work/myjdbcdatasources.mat')
```

Tip Include this statement in a MATLAB startup file to access a given JDBC data source automatically when your MATLAB software session starts.

See Also

clear, cursor.fetch, getdatasources, "Working with Preferences" on page 4-6

sql2native

Purpose Convert JDBC SQL grammar to SQL grammar native to system

Syntax n = sql2native(conn, 'sqlquery')

Description n = sql2native(conn, 'sqlquery'), converts the SQL statement

string sqlquery from JDBC SQL grammar into the database system's

native SQL grammar for the connection $\mbox{conn.}$ The native SQL

statement is assigned to n.

Detect whether property is supported by database metadata object

Syntax

```
a = supports(dbmeta)
a = supports(dbmeta, 'property')
a.property
```

Description

- a = supports(dbmeta) returns a structure that contains the properties of dbmeta and its property values, 1 or 0. A value of 1 indicates that the property is supported, and 0 indicates that the property is not supported.
- a = supports(dbmeta, 'property') returns 1 or 0 for the property field of dbmeta. A value of 1 indicates that the property is supported, and 0 indicates that the property is not supported.
- a.property returns the value of property after you have created a using the supports function.

Examples

1 Check if dbmeta supports group-by clauses.

```
a = supports(dbmeta, 'GroupBy')
a =
1
```

2 View the value of all properties of dbmeta.

```
a = supports(dbmeta)
```

The returned result is a list of properties and their values.

3 See the value of the **GroupBy** property by running:

```
a.GroupBy
a =
1
```

See Also

database, dmd, get, ping

Return database table privileges

Syntax

```
tp = tableprivileges(dbmeta, 'cata')
tp = tableprivileges(dbmeta, 'cata', 'sch')
tp = tableprivileges(dbmeta, 'cata', 'sch', 'tab')
```

Description

- tp = tableprivileges(dbmeta, 'cata') returns a list of table privileges for all tables in the catalog cata, for the database whose database metadata object is dbmeta.
- tp = tableprivileges(dbmeta, 'cata', 'sch') returns a list of table privileges for all tables in:
 - The schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta
- tp = tableprivileges(dbmeta, 'cata', 'sch', 'tab') returns a list of privileges for:
 - The table tab
 - The schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta

Examples

Get table privileges for the builds table in the schema geck for the catalog msdb, for the database metadata object dbmeta.

See Also

dmd, get, tables

Return database table names

Syntax

```
t = tables(dbmeta, 'cata')
t = tables(dbmeta, 'cata', 'sch')
```

Description

- t = tables(dbmeta, 'cata') returns a list of tables and table types in the catalog cata, for the database whose database metadata object is dbmeta.
- t = tables(dbmeta, 'cata', 'sch') returns a list of tables and table types in:
 - The schema sch
 - Of the catalog cata
 - For the database whose database metadata object is dbmeta

Tip For command-line help on tables, use the overloaded method:

```
help dmd/tables
```

Examples

Get the table names and types for the schema SCOTT in the catalog orcl, for the database metadata object dbmeta.

```
t = tables(dbmeta,'orcl', 'SCOTT')
t =
    'BONUS' 'TABLE'
    'DEPT' 'TABLE'
    'EMP' 'TABLE'
    'SALGRADE' 'TABLE'
    'TRIAL' 'TABLE'
```

See Also

attr, bestrowid, dmd, get, indexinfo, tableprivileges

unregister

Purpose Unload database driver

Syntax unregister(d)

Description unregister(d) unloads the database driver object d, freeing up system

resources. If you do not unload a registered driver, it automatically

unloads when you end your MATLAB software session.

Examples unregister(d) unloads the database driver object d.

See Also register

Replace data in database table with MATLAB data

Syntax

```
update(conn, 'tab', colnames, exdata, 'whereclause')
update(conn, 'tab', colnames, ...
{datA,datAA, ...; datB,datBB, ...; datn, datNN}, ...
{'where col1 = val1'; where col2 = val2'; ... 'where coln = valn'}
```

Description

update(conn, 'tab', colnames, exdata, 'whereclause') exports the MATLAB variable exdata in its current format into the database table tab using the database connection conn.exdata can be a cell array, numeric matrix, or structure. Existing records in the database table are replaced as specified by the SQL whereclause command.

Specify column names for tab as strings in the MATLAB cell array colnames. If exdata is a structure, field names in the structure must exactly match field names in colnames.

The status of the AutoCommit flag determines whether update automatically commits the data to the database. View the AutoCommit flag status for the connection using get and change it using set. Commit the data by running commit or a SQL commit statement via the exec function. Roll back the data by runningrollback or a SQL rollback statement via the exec function.

To add new rows instead of replacing existing data, use fastinsert.

update(conn, 'tab', colnames, {datA, datAA, ...; datB, datBB, ...; datn,datNN}, {'where col1 = val1'; where col2 = val2'; ... 'where coln = valn'}) exports multiple records for n where clauses. The number of records in exdata must equal n.

Remarks

- The order of records in your database is not constant. Use values of column names to identify records.
- An error like the following may appear if your database table is open in edit mode:

[Vendor][ODBC Product Driver] The database engine could not lock table 'TableName' because it is already in use

by another person or process.

In this case, close the table and repeat the update function.

• An error like the following may appear if you try to run an update operation that is identical to one that you just ran:

```
??? Error using ==> database.update
Error:Commit/Rollback Problems
```

Examples Example 1 - Update an Existing Record

Update the record in the Birthdays table using the database connection conn, where First_Name is Jean, replacing the current value for Age with 40.

1 First define a cell array containing the column name that you are updating, Age.

```
colnames = {'Age'}
```

2 Define a cell array containing the new data, 40.

```
exdata(1,1) = {40}
```

3 Run the update.

```
update(conn, 'Birthdays', colnames, exdata, ...
'where First_Name = ''Jean''')
```

Example 2 - Roll Back Data after Updating a Record

Update the column Date in the Error_Rate table for the record selected by whereclause, using data contained in the cell array exdata. The AutoCommit flag is off. The data is rolled back after the update operation is run.

1 Set the AutoCommit flag to off for database connection conn.

```
set(conn, 'AutoCommit', 'off')
```

2 Update the Date column.

```
update(conn, 'Error_Rate', {'Date'}, exdata, whereclause)
```

3 Because the data was not committed, you can roll it back.

```
rollback(conn)
```

The update is reversed; the data in the table is the same as it was before you ran update.

Example 3 — Update Multiple Records with Different Constraints

Given the table TeamLeagues, where column names are 'Team', 'Zip Code', and 'New League':

```
'Team1' 02116
'Team2' 02138
'Team3' 02116
```

Assign teams with a zip code of 02116 to the A league and teams with a zip code of 02138 to the B league:

```
update(conn, 'TeamLeagues', {'League'}, {'A';'B'}, ... {'where Zip_Code =''02116''';'where Zip_Code =''02138'''})
```

See Also

commit, database, fastinsert, rollback, set

Purpose

Automatically update table columns

Syntax

```
v1 = versioncolumns(dbmeta, 'cata')
v1 = versioncolumns(dbmeta, 'cata', 'sch')
v1 = versioncolumns(dbmeta, 'cata', 'sch', 'tab')
```

Description

- v1 = versioncolumns (dbmeta, 'cata') returns a list of columns that automatically update when a row value updates in the catalog cata, in the database whose database metadata object is dbmeta.
- v1 = versioncolumns(dbmeta, 'cata', 'sch') returns a list of all columns that automatically update when a row value updates in:
 - The schema sch
 - In the catalog cata
 - For the database whose database metadata object is dbmeta
- vl = versioncolumns(dbmeta, 'cata', 'sch', 'tab') returns a list of columns that automatically update when a row value updates in:
 - The table tab
 - The schema sch
 - In the catalog cata
 - For the database whose database metadata object is dbmeta

Examples

Get a list of which columns automatically update when a row in the table BONUS updates, in the schema SCOTT, in the catalog orcl, for the database metadata object dbmeta.

```
v1 = versioncolumns(dbmeta,'orcl','SCOTT','BONUS')
v1 =
{}
```

The results are an empty set, indicating that no columns in the database automatically update when a row value updates.

versioncolumns

See Also

columns, dmd, get

Purpose Return field size of column in fetched data set

Syntax colsize = width(cursor, colnum)

Description colsize = width(cursor, colnum) returns the field size of the

specified column number colnum in the fetched data set curs.

Examples Get the width of the first column of the fetched data set, curs:

```
colsize = width(curs, 1)
colsize =
11
```

The field size of column one is 11 characters (bytes).

See Also attr, cols, columnnames, cursor.fetch, get

width



Examples

Use this list to find examples in the documentation.

Visual Query Builder GUI: Importing Data

"Working with Preferences" on page 4-6

"Retrieving All Occurrences vs. Unique Occurrences of Data" on page 4-22

"Retrieving Data That Meets Specified Criteria" on page 4-24

"Creating Subqueries for Values from Multiple Tables" on page 4-37

"Creating Queries That Include Results from Multiple Tables" on page 4-42

"Retrieving BINARY and OTHER Sun Java Data Types" on page 4-46

"Importing BOOLEAN Data from Databases to the MATLAB Workspace" on page 4-48

Visual Query Builder GUI: Displaying Results

"Displaying Data Relationally" on page 4-10

"Charting Query Results" on page 4-14

"Displaying Query Results in an HTML Report" on page 4-16

"Using the MATLAB® Report Generator Software to Customize Display of Query Results" on page 4-17

"Displaying Results in a Specified Order" on page 4-31

Visual Query Builder GUI: Advanced Query Options

"Example: Using Having Clauses" on page 4-36

Visual Query Builder GUI: Exporting Data

"Exporting BOOLEAN Data from the MATLAB Workspace to Databases" on page 4-51

Using Database Toolbox Functions

"Importing Data from Databases into the MATLAB Workspace" on page 5-3

"Viewing Information About Imported Data" on page 5-9

"Exporting Data from the MATLAB Workspace to a New Record in a Database" on page 5-11

"Replacing Existing Data in Databases with Data Exported from the MATLAB Workspace" on page 5-15

"Exporting Multiple Records from the MATLAB Workspace" on page 5-17

"Retrieving BINARY or OTHER Sun Java SQL Data Types" on page 5-21

"Working with Database Metadata" on page 5-23

"Using Driver Functions" on page 5-29

Index

A	cell arrays
advanced query options in VQB 4-22	assigning values to cells 5-12
All option in VQB 4-22	data format 7-108
array	for exporting data 5-12
data format 7-108	for query results 5-6
arrays	setting data format in VQB 4-6
data format in VQB 4-8	charting
attr 7-2	query results 4-14
example 5-10	Charting dialog box 4-14
Attributes 7-59	data (x, y, z, and color) 4-15
attributes of data	legends 4-15
attr function 7-2	preview 4-15
example 5-10	clearwarnings 7-5
AutoCommit	close 7-6
example 5-13	cols 7-8
setting status 7-102	example 5-9
status via get 7-58	ColumnCount 7-62
Status via got 1 30	ColumnName 7-62
D	columnnames 7-9
В	exporting example 5-18
bestrowid $7-4$	importing example 5-9
BINARY data types	columnprivileges $7 ext{-} 10$
retrieving with functions 5-21	columns $7 ext{-} 12$
retrieving with VQB 4-46	attributes 5-10
BOOLEAN data type	automatically updated 7-131
inserting 7-51	cross reference 7-17
retrieving 7-24	exported keys 7-44
VQB 4-48	foreign key information 7-66
	imported key information 7-66
C	names, exporting 5-12
	names, importing 5-9
catalog	names, via attr 7-2
changing 7-40	names, via columnnames 7-9
Catalog 7-58	names, via columns 7-12
CatalogName 7-62	number 7-8
	optimal set to identify row 7-4
	primary key information 7-85
	privileges 7-10
	viewing width 5-10
	width 7-133

ColumnTypeName 7-62	currency 7-2
columnWidth 7-2	Current clauses area in VQB
commit 7-14	example 4-25
example 5-13	cursor
via exec 7-39	attributes 7-59
Condition in VQB 4-24	close function 7-6
confds	creating via exec 7-36
function reference 7-15	creating via fetch 7-20
Configure Data Source dialog box 7-15	data element 7-59
connection	error messages 7-59
clearing warnings for 7-5	importing data 5-6
close function 7-6	object 7-20
creating 7-25	objects
database, opening (establishing) 7-25	example 5-5
database, opening (establishing),	opening 5-5
example 5-3	properties 7-101
information 7-83	properties, example 7-56
JDBC 7-58	resultset object 7-95
messages 7-58	Cursor 7-59
object 5-3	cursor.fetch $7\text{-}20$
opening 7-25	example 5-6
properties, getting 7-56	relative to fetch 7-52
properties, setting 7-101	
read-only 7-77	D
status 7-83	_
status, example 5-5	data
time allowed for 7-79	attributes 7-2
time allowed for, example 5-3	example 5-10
validity 7-72	cell array 5-12
warnings 7-58	column names 7-9
constructor functions 5-31	example 5-9
crossreference 7-17	column numbers 7-8

example 5-9	database
commit function 7-14	connecting to 7-25
committing 7-102	connecting to, example 5-3
displaying results in VQB 4-10	example 5-3
exporting 7-47 7-71	JDBC connection 7-58
exporting, example 5-13	metadata objects
field names 7-9	creating 7-33
importing 7-20	properties 7-56
importing, example 5-6	properties supported 7-124
information about 5-9	name 7-25
inserting into database 5-20	supported 1-3
replacing 5-15	URL 7-26
rolling back 7-96	Database Toolbox
rolling back, via set 7-102	about 1-2
rows 5-9	relationship of functions to VQB 3-1
rows function 7-97	Database Toolbox requirements 1-3
unique occurrences of 4-22	database.fetch 7-30
updating 7-128	relative to fetch 7-52
Data 7-59	database/fetch 7-52
data format 7-108	DatabaseObject 7-59
Database Toolbox 4-8	dbdemos 5-1
preferences for retrieval 7-108	demos 5-1
preferences in VQB 4-6	dbinfodemo 5-9
data sources	dbinsertdemo 5-11
defining	dbupdatedemo 5-15
JDBC 7-15	displaying
for connection 7-25	query results
JDBC	as chart 4-14
accessing 2-4	as report 4-16
modifying 2-5	in MATLAB Report Generator
removing 2-6	software 4-17
updating 2-5	relationally 4-10
ODBC connection 7-58	Distinct option in VQB 4-22
ODBC, on system 7-65	dmd $7-33$
data types 7-2	example 5-23
BINARY, retrieving with functions 5-21	dotted line in display of results 4-12
BINARY, retrieving with VQB 4-46	driver $7\text{-}34$
OTHER, retrieving with functions 5-21	example 5-29
OTHER, retrieving with VQB 4-46	object in get function 7-58
supported 1-5	

driver objects	exportedkeys 7-44
functions 6-6	exporting data
functions, example 5-29	cell arrays 5-12
properties 5-29	inserting 7-47 7-71
drivermanager 7-35	example 5-11
drivermanager objects	multiple records 5-20
example 5-29	replacing 7-128
properties 7-101	replacing, example 5-15
properties, via get 7-56	
drivers	F
JDBC 1-4 7-26	r
troubleshooting 2-7	fastinsert 7-47
JDBC compliance 7-74	example 5-13
loading 7-94	fetch $7\text{-}52$
ODBC 1-4	cursor $7\text{-}20$
properties 7-56	database 7-30
properties, drivermanager 7-35	Fetch 7-59
supported 1-4	fetchmulti $7\text{-}54$
unloading 7-127	fieldName $7-2$
validity 7-73	fields
Drivers 7-61	names 7-12
	size (width) 7-2
-	example 5-10
E	width $7 \text{-} 133$
editing clauses in VQB 4-26	foreign key information
empty field 5-21	crossreference 7-17
error	exportedkeys 7-44
messages	importedkeys 7-66
cursor object 7-59	format for data retrieved, preferences 7-108
database connection object 7-58	freeing up resources 7-6
modifying database 7-36	functions
error handling	equivalent to VQB queries 4-52
preferences 4-6	when to use 3-3
error notification, preferences 7-108	
examples	G
using functions 5-1	-
exec 7-36	get 5-30 7-56
example 5-5	AutoCommit status 5-13
with fetch 7-30	properties 5-29
executing queries 7-36	getdatasources 7-65

grouping statements 4-27	JDBC
removing 4-31	compliance 7-74
	connection object 7-58
н	driver instance 7-58
	driver name 7-26
Handle 7-58	drivers
Having Clauses dialog box 4-34	names 7-26
Having in VQB 4-34	supported 1-4
HTML report of query results 4-16	validity 7-73
MATLAB Report Generator software 4-17	MAT-file location preference 7-108
	SQL conversion to native grammar 7-123
1	URL 7-26
	via get 7-58
images	join operation in VQB 4-42
importing 5-21	•
VQB 4-46	L
importedkeys 7-66	L
importing data	legends
data types	in chart 4-15
BINARY and OTHER using functions 5-21	labels in chart 4-15
BINARY and OTHER using VQB 4-46	logical data types
empty field 5-21	inserting 7-51
using functions 7-20	retrieving 7-24
example 5-3	VQB 4-48
index for resultset column 7-82	logintimeout 7-79
indexinfo 7-69	example 5-3
insert 7-71	Macintosh platform support 7-79
inserting data into database 5-20	LoginTimeout
Instance 7-58	Database connection object 7-58
isconnection 7-72	Drivermanager objects 7-61
isdriver 5-30 7-73	example 5-30
isjdbc 7-74	LogStream 7-61
isNullable 7-62	
isnullcolumn 7-75	M
isreadonly 7-77	
isReadOnly 7-62	M-files 5-1
isurl 7-78	generated from VQB 4-52
	MajorVersion 7-60
J	MATLAB Report Generator software
Java TM Databasa Connectivity See JDRC	display of query results 4-17

memory problems	Open Database Connectivity. See ODBC
RowInc solution 7-30	Operator in VQB 4-26
RowLimit solution 7-20	Order By Clauses dialog box 4-32
Message	Order by option in VQB 4-31
attr 7-2	OTHER data types
cursor object 7-59	retrieving with functions 5-21
database connection object 7-58	retrieving with VQB 4-46
metadata objects	•
database 7-33	P
example 5-23	P
resultset 7-98	parentheses, adding to statements 4-27
resultset functions 5-28	password 7-25 to 7-26
methods 5-31	ping $7\text{-}83$
MinorVersion 7-60	AutoCommit $5\text{-}13$
	example 5-5
NI.	platforms 1-3
N	precision $7\text{-}2$
namecolumn 7-82	preferences
nested SQL 4-37	for Visual Query Builder 4-6
NULL values	primary key information 7-17
detecting in imported record 7-75	primarykeys $7\text{-}85$
function for handling 4-9	privileges
preferences for reading and writing 4-6	columns 7-10
reading from database 5-17	tables 7-125
representation in results 4-8	procedurecolumns 7-87
setdbprefs 7-108	procedures 7-90
writing to database 4-6	properties
nullable 7-2	database metadata objects 7-124
numeric data format 7-108	example 5-24
VQB 4-6	drivers 5-29
	getting 7-56
0	setting 7-101
objects 5-31	
creating 5-31	Q
properties, getting 7-56	queries
ObjectType 7-58	accessing subqueries in multiple tables 4-37
ODBC	accessing values in multiple tables 4-42
data sources on system 7-65	displaying results
drivers 1-4	as chart 4-14
WIIVOID I-I	as chart T-17

as report 4-16	rollback $7\text{-}96$
in MATLAB Report Generator	RowInc
software 4-17	database.fetch 7-30
relationally 4-10	RowLimit
ordering results 4-31	fetch $7\text{-}20$
refining 4-24	get 7-59
results 7-59	set 7-103
running via exec 7-36	rows 7-97
querybuilder 7-92	example 5-9
querytimeout 7-93	uniquely identifying 7-4
quotation marks	rsmd 7-98
in table and column names 1-7	runstoredprocedure 7-99
R	S
readonly 7-2	scale 7-2
ReadOnly 7-58	selecting data from database 7-38
refining queries 4-24	set 7-101
register 7-94	example 5-30
Relation in VQB 4-24	setdbprefs 7-108
relational display of query results 4-10	example 5-17
replacing data 5-15	VQB 4-9
update function 7-128	size 5-19
reporting query results	size of field 5-10
MATLAB Report Generator software 4-17	Sort key number in VQB 4-32
table 4-16	Sort order in VQB 4-32
reserved words	spaces
in table and column names 1-7	in table and column names 1-7
resultset 7-95	speed
clearing warnings for 7-5	inserting data 7-47
closing 7-6	SQL
column name and index 7-82	commands 1-4
metadata objects 5-28	conversion to native grammar 7-123
creating 7-98	join in VQB 4-42
properties 7-56	statement
object, functions 6-6	executing 7-36
properties 7-56	in exec 7-59
ResultSet 7-59	in exec, example 5-5
retrieving data	
restrictions 1-7	

in VQB 4-26	time
time allowed for query 7-93	allowed for connection 7-79
where clause 7-128	allowed for SQL query 7-93
example 5-15	TimeOut 7-58
where clause in exec 5-15	TransactionIsolation $7\text{-}58$
sql2native 7-123	Type 7-59
SQLQuery 7-59	typeName 7-2
Statement 7-59	typeValue $7\text{-}2$
status of connection 7-83	
example 5-5	U
stored procedures	•
in catalog or schema 7-90	undoing exported data update 5-13
information 7-87	ungrouping statements 4-31
running 7-40	unique occurrences of data 4-22
string and numeric data format 7-108	unregister 7-127
strings	update 7-128
within strings 5-15	example 5-15
structure data format 7-108	URL 7-58
VQB 4-6	JDBC database connection 7-26
subqueries	validity 7-78
in VQB 4-37	user name 7-25 to 7-26 7-58
Subquery dialog box 4-38	
supports 7-124	V
example 5-26	versioncolumns 7-131
system requirements 1-3	Visual Query Builder
	advanced query options 4-22
T	equivalent Database Toolbox functions 4-52
table	getting started 4-2
	limitations 3-2
creating	starting 7-92
using exec 7-40 TableName 7-62	steps to export (insert) data 4-4
tableprivileges 7-125	steps to import (retrieve) data 4-2
tables 7-126	when to use 3-2
example 5-28	VQB. See Visual Query Builder
index information 7-69	The state of the s
names 7-126	14/
privileges 7-125	W
selecting multiple for VQB 4-43	Warnings 7-58
selecting multiple for VQD 4-45	warnings, clearing 7-5

 $\begin{array}{c} \text{where clause} \ \, 7\text{-}128 \\ \text{example} \ \, 5\text{-}15 \\ \text{WHERE Clauses dialog box} \ \, 4\text{-}24 \\ \text{Where option in VQB} \ \, 4\text{-}24 \end{array}$

width 7-133 example 5-10 writable 7-58