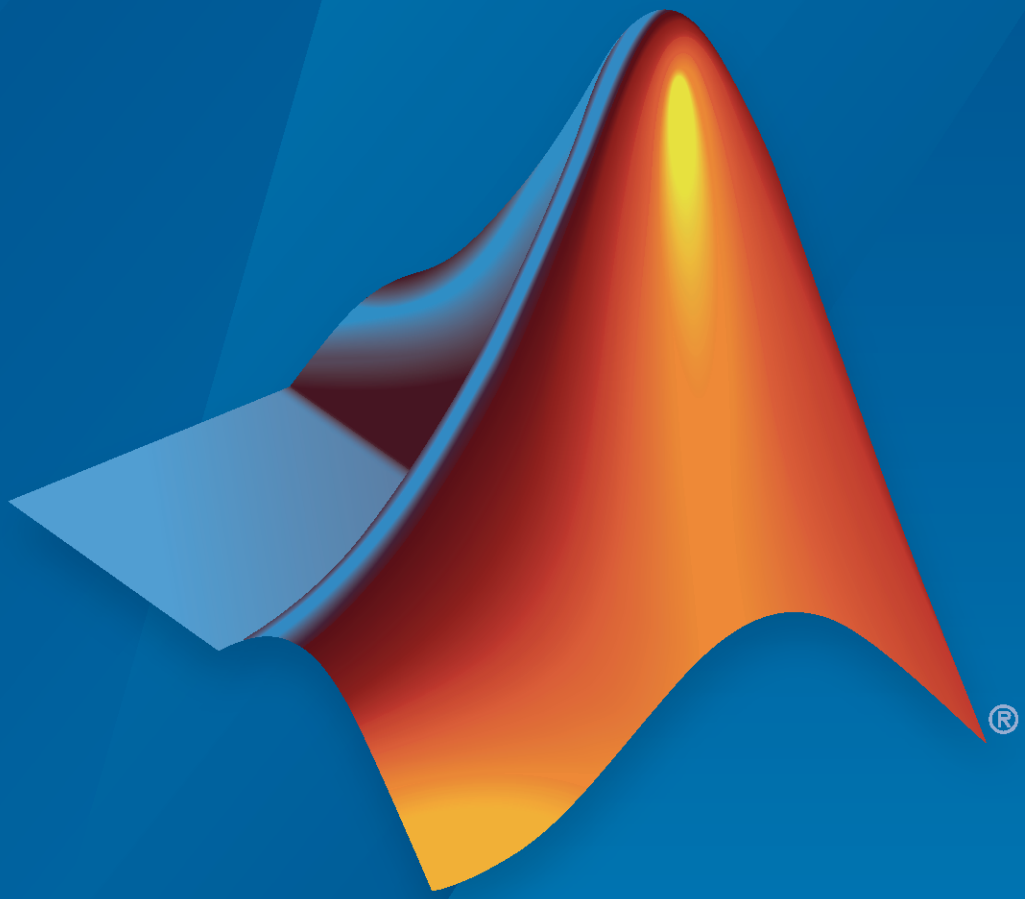
Database Toolbox™ Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

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

Database Toolbox™ Release Notes

Contents

R2020b

R2020a

R2019b

R2019a

R2018b

R2018a

R2017b

R2017a

R2016b

R2016a

R2015b

R2015a

Bug Fixes

R2014b

R2014a

Bug Fixes

R2013b

R2013a

R2012b

R2012a

R2011b

R2011a

R2010b

R2010a

R2009b

Bug Fixes

R2009a

Bug Fixes

R2008b

Bug Fixes

R2008a

Bug Fixes

R2007b

Bug Fixes

R2007a

R2006b

R2006a

R14SP3

R14SP2

Bug Fixes

R2020b

MySQL native interface

The MySQL® native interface provides another way to connect to a MySQL database. Use this interface as an alternative to an ODBC or JDBC database connection.

Install the MySQL Connector/C++ driver, create a data source by using the `databaseConnectionOptions` function, and then connect to the MySQL database by using the `mysql` function. For details about this native interface, see “MySQL Native Interface”.

PostgreSQL native interface

The PostgreSQL native interface provides another way to connect to a PostgreSQL database. Use this interface as an alternative to an ODBC or JDBC database connection.

Create a data source by using the `databaseConnectionOptions` function, and then connect to the PostgreSQL database by using the `postgresql` function. For details about this native interface, see “PostgreSQL Native Interface”.

Preserve non-ASCII characters in column names of imported data

When you import data from a database table, you can preserve the names of the columns in the table. If a column name contains non-ASCII characters, you can preserve these characters in the imported data. Otherwise, by default, the column name is converted to ASCII characters only. For details, see the `VariableNamingRule` property of the `SQLImportOptions` object.

Set JDBC connection options with updated workflow

You can configure and save a JDBC data source by setting JDBC connection options using the `SQLConnectionOptions` object and its functions. This table describes the functions for creating and working with the `SQLConnectionOptions` object.

Function	Description
`databaseConnectionOptions`	Create `SQLConnectionOptions` object containing connection options
`setoptions`	Set JDBC connection options
`rmoptions`	Remove JDBC connection options
`testConnection`	Test JDBC database connection
`saveAsDataSource`	Save JDBC data source
`reset`	Reset JDBC connection options to their defaults

To manage data sources, you can view saved data sources using the `listDataSources` function. Then, you can delete a data source using the `deleteDataSource` function.

Compatibility Considerations

getdatasources function will be removed
Still runs

The `getdatasources` function will be removed in a future release. Use the `listDataSources` function instead.

configureJDBCDataSource function will be removed

Still runs

The `configureJDBCDataSource` function will be removed in a future release. Use the `databaseConnectionOptions` function instead. Some differences between the workflows might require updates to your code.

Update Code

Use the `databaseConnectionOptions` function to create an `SQLConnectionOptions` object.

In prior releases, you configured a JDBC data source using the `configureJDBCDataSource` function and the `JDBCConnectionOptions` object. For example:

```
opts = configureJDBCDataSource('Vendor','Microsoft SQL Server');
opts = setConnectionOptions(opts, ...
    'DataSourceName','SQLServerDataSource', ...
    'Server','dbtb04','PortNumber',54317, ...
    'JDBCDriverLocation','C:\Drivers\sqljdbc4.jar', ...
    'AuthType','Windows');
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsJDBCDataSource(opts)
```

Now you can set JDBC connection options using the `databaseConnectionOptions` function, and save the data source using the `SQLConnectionOptions` object instead.

```
vendor = "Microsoft SQL Server";
opts = databaseConnectionOptions("jdbc",vendor);
opts = setoptions(opts, ...
    'DataSourceName',"SQLServerDataSource", ...
    'JDBCDriverLocation',"C:\Drivers\sqljdbc4.jar", ...
    'DatabaseName',"toystore_doc",'Server',"dbtb04", ...
    'PortNumber',54317,'AuthenticationType',"Windows");
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsDataSource(opts)
```

Also, you can open the JDBC Data Source Configuration dialog box by using the **Database Explorer** app.

JDBCConnectionOptions object will be removed

Still runs

The `JDBCConnectionOptions` object will be removed in a future release. Use the `SQLConnectionOptions` object instead. Some differences between the workflows might require updates to your code.

Update Code

Use the `SQLConnectionOptions` object to set JDBC connection options.

In prior releases, you configured a JDBC data source using the `JDBCConnectionOptions` object. For example:

```
opts = configureJDBCDataSource('Vendor','Microsoft SQL Server');
opts = setConnectionOptions(opts, ...
    'DataSourceName','SQLServerDataSource', ...
    'Server','dbtb04','PortNumber',54317, ...
    'JDBCDriverLocation','C:\Drivers\sqljdbc4.jar', ...
    'AuthType','Windows');
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsJDBCDataSource(opts)
```

Now you can set JDBC connection options and save the data source using the `SQLConnectionOptions` object instead.

```
vendor = "Microsoft SQL Server";
opts = databaseConnectionOptions("jdbc",vendor);
opts = setoptions(opts, ...
    'DataSourceName',"SQLServerDataSource", ...
    'JDBCDriverLocation',"C:\Drivers\sqljdbc4.jar", ...
    'DatabaseName',"toystore_doc",'Server',"dbtb04", ...
    'PortNumber',54317,'AuthenticationType',"Windows");
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsDataSource(opts)
```

addConnectionOptions function will be removed
Still runs

The `addConnectionOptions` function will be removed in a future release. Use the `setoptions` function instead. Some differences between the workflows might require updates to your code.

Update Code

Use the `setoptions` function with the `SQLConnectionOptions` object to set JDBC driver-specific connection options.

In prior releases, you configured a JDBC data source using the `JDBCConnectionOptions` object, and added options using the `addConnectionOptions` function. For example:

```
opts = configureJDBCDataSource('Vendor','Microsoft SQL Server');
opts = setConnectionOptions(opts, ...
    'DataSourceName','SQLServerDataSource', ...
    'Server','dbtb04','PortNumber',54317, ...
    'JDBCDriverLocation','C:\Drivers\sqljdbc4.jar', ...
    'AuthType','Windows');
opts = addConnectionOptions(opts,'loginTimeout',20);
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsJDBCDataSource(opts)
```

Now you can set JDBC driver-specific connection options using the `setoptions` function with the `SQLConnectionOptions` object instead.

```
vendor = "Microsoft SQL Server";
opts = databaseConnectionOptions("jdbc",vendor);
opts = setoptions(opts, ...
    'DataSourceName',"SQLServerDataSource", ...
    'JDBCDriverLocation',"C:\Drivers\sqljdbc4.jar", ...
    'DatabaseName',"toystore_doc",'Server',"dbtb04", ...
    'PortNumber',54317,'AuthenticationType',"Windows", ...
    'loginTimeout',20);
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsDataSource(opts)
```

deleteJDBCDataSource function will be removed

Still runs

The `deleteJDBCDataSource` function will be removed in a future release. Use the `deleteDataSource` function instead.

rmConnectionOptions function will be removed

Still runs

The `rmConnectionOptions` function will be removed in a future release. Use the `rmoptions` function instead. Some differences between the workflows might require updates to your code.

Update Code

Use the `rmoptions` function with the `SQLConnectionOptions` object to remove JDBC connection options.

In prior releases, you configured a JDBC data source using the `JDBCConnectionOptions` object, and removed options using the `rmConnectionOptions` function. For example:

```
opts = configureJDBCDataSource('Vendor','Microsoft SQL Server');
opts = setConnectionOptions(opts, ...
    'DataSourceName','SQLServerDataSource', ...
    'Server','dbtb04','PortNumber',54317, ...
    'JDBCDriverLocation','C:\Drivers\sqljdbc4.jar', ...
    'AuthType','Windows');
opts = addConnectionOptions(opts,'loginTimeout',20);
username = "";
password = "";
status = testConnection(opts,username,password);
opts = rmConnectionOptions(opts,'loginTimeout');
status = testConnection(opts,username,password);
saveAsJDBCDataSource(opts)
```

Now you can set JDBC driver-specific connection options with the `SQLConnectionOptions` object instead, and then remove options using the `rmoptions` function.

```
vendor = "Microsoft SQL Server";
opts = databaseConnectionOptions("jdbc",vendor);
opts = setoptions(opts, ...
    'DataSourceName',"SQLServerDataSource", ...
    'JDBCDriverLocation',"C:\Drivers\sqljdbc4.jar", ...
    'DatabaseName',"toystore_doc",'Server',"dbtb04", ...
    'PortNumber',54317,'AuthenticationType',"Windows", ...
    'loginTimeout',20);
```

```
username = "";
password = "";
status = testConnection(opts,username,password);
opts = rmoptions(opts,'loginTimeout');
status = testConnection(opts,username,password);
saveAsDataSource(opts)
```

saveAsJDBCDataSource function will be removed
Still runs

The `saveAsJDBCDataSource` function will be removed in a future release. Use the `saveAsDataSource` function instead. Some differences between the workflows might require updates to your code.

Update Code

Use the `saveAsDataSource` function with the `SQLConnectionOptions` object to save the JDBC data source.

In prior releases, you configured and saved a JDBC data source using the `JDBCConnectionOptions` object and the `saveAsJDBCDataSource` function. For example:

```
opts = configureJDBCDataSource('Vendor','Microsoft SQL Server');
opts = setConnectionOptions(opts, ...
    'DataSourceName','SQLServerDataSource', ...
    'Server','dbtb04','PortNumber',54317, ...
    'JDBCDriverLocation','C:\Drivers\sqljdbc4.jar', ...
    'AuthType','Windows');
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsJDBCDataSource(opts)
```

Now you can set JDBC connection options and save the data source using the `SQLConnectionOptions` object and the `saveAsDataSource` function instead.

```
vendor = "Microsoft SQL Server";
opts = databaseConnectionOptions("jdbc",vendor);
opts = setoptions(opts, ...
    'DataSourceName',"SQLServerDataSource", ...
    'JDBCDriverLocation',"C:\Drivers\sqljdbc4.jar", ...
    'DatabaseName',"toystore_doc",'Server',"dbtb04", ...
    'PortNumber',54317,'AuthenticationType',"Windows");
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsDataSource(opts)
```

setConnectionOptions function will be removed
Still runs

The `setConnectionOptions` function will be removed in a future release. Use the `setoptions` function instead. Some differences between the workflows might require updates to your code.

Update Code

Use the `setoptions` function with the `SQLConnectionOptions` object to set the JDBC connection options.

In prior releases, you configured a JDBC data source using the `setConnectionOptions` function and the `JDBCConnectionOptions` object. For example:

```
opts = configureJDBCDataSource('Vendor','Microsoft SQL Server');
opts = setConnectionOptions(opts, ...
    'DataSourceName','SQLServerDataSource', ...
    'Server','dbtb04','PortNumber',54317, ...
    'JDBCDriverLocation','C:\Drivers\sqljdbc4.jar', ...
    'AuthType','Windows');
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsJDBCDataSource(opts)
```

Now you can set JDBC connection options using the `setoptions` function, and save the data source using the `SQLConnectionOptions` object instead.

```
vendor = "Microsoft SQL Server";
opts = databaseConnectionOptions("jdbc",vendor);
opts = setoptions(opts, ...
    'DataSourceName',"SQLServerDataSource", ...
    'JDBCDriverLocation',"C:\Drivers\sqljdbc4.jar", ...
    'DatabaseName',"toystore_doc",'Server',"dbtb04", ...
    'PortNumber',54317,'AuthenticationType',"Windows");
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsDataSource(opts)
```

testConnection function will be removed
Still runs

The `testConnection` function will be removed in a future release. Use the `testConnection` function of the `SQLConnectionOptions` object instead. Some differences between the workflows might require updates to your code.

Update Code

Use the `testConnection` function with the `SQLConnectionOptions` object to test the specified JDBC connection options.

In prior releases, you configured a JDBC data source using the `JDBCConnectionOptions` object. For example:

```
opts = configureJDBCDataSource('Vendor','Microsoft SQL Server');
opts = setConnectionOptions(opts, ...
    'DataSourceName','SQLServerDataSource', ...
    'Server','dbtb04','PortNumber',54317, ...
    'JDBCDriverLocation','C:\Drivers\sqljdbc4.jar', ...
    'AuthType','Windows');
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsJDBCDataSource(opts)
```

Now you can set JDBC connection options and save the data source using the `SQLConnectionOptions` object instead.

```
vendor = "Microsoft SQL Server";
opts = databaseConnectionOptions("jdbc",vendor);
opts = setoptions(opts, ...
    'DataSourceName',"SQLServerDataSource", ...
    'JDBCDriverLocation',"C:\Drivers\sqljdbc4.jar", ...
    'DatabaseName',"toystore_doc",'Server',"dbtb04", ...
    'PortNumber',54317,'AuthenticationType',"Windows");
username = "";
password = "";
status = testConnection(opts,username,password);
saveAsDataSource(opts)
```

Functionality being removed or changed

database.ODBCConnection syntax has been removed

Errors

The `database.ODBCConnection` syntax has been removed. Use the syntaxes of the `database` function instead.

R2020a

Version: 9.2.1

Bug Fixes

Compatibility Considerations

Functionality being removed or changed

tables function has been removed

Errors

The `tables` function has been removed. Use the `sqlfind` function or access the properties of the `connection` object instead.

Some differences between these functions require updates to your code.

Update Code

In prior releases, you retrieved information about database tables by using the `tables` function. For example:

```
conn = database(datasource,username,password);
t = tables(conn,catalog,schema);
```

Now the `sqlfind` function returns information about the table types in a database.

```
conn = database(datasource,username,password);
data = sqlfind(conn,pattern);
```

Also, you can access the database properties and catalog and schema information by using the `connection` object. For example:

```
conn = database(datasource,username,password);
catalogs = conn.Catalogs;
schemas = conn.Schemas;
```

columns function has been removed

Errors

The `columns` function has been removed. Use the `sqlfind` function or access the properties of the `connection` object instead.

Some differences between these functions require updates to your code.

Update Code

In prior releases, you retrieved information about database columns by using the `columns` function. For example:

```
conn = database(datasource,username,password);
columnlist = columns(conn,catalog);
```

Now the `sqlfind` function returns information about the table types in the database (including information about database columns).

```
conn = database(datasource,username,password);
data = sqlfind(conn,pattern);
```

Also, you can access the database properties and catalog and schema information by using the `connection` object. For example:

```
conn = database(datasource,username,password);
catalogs = conn.Catalogs;
schemas = conn.Schemas;
```

R2019b

Version: 9.2

New Features

Bug Fixes

Database Toolbox interface for MongoDB SSL-enabled connection

The Database Toolbox interface for MongoDB® now supports the creation of an SSL-enabled MongoDB connection. For details about creating an SSL-enabled connection, see the `mongo` function.

Database Toolbox interface for MongoDB supports new MongoDB versions

The Database Toolbox interface for MongoDB now supports these versions of MongoDB:

- MongoDB 2.6
- MongoDB 3.0
- MongoDB 3.2
- MongoDB 3.4
- MongoDB 3.6
- MongoDB 4.0

To install the Database Toolbox interface for MongoDB, see Database Toolbox Interface for MongoDB Installation.

Graph database supports directed graph output

The MATLAB® interface to Neo4j® now supports searching a Neo4j database and returning graph or relationship information as a directed graph. You can specify the data return format of the output arguments in the `searchGraph` and `searchRelation` functions by using the `'DataReturnFormat'` name-value pair argument.

Database Toolbox Interface for Apache Cassandra Database SSL-enabled connection

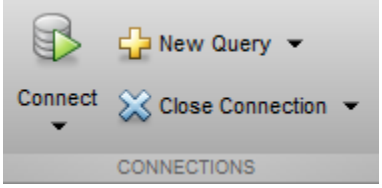
The Database Toolbox Interface for Apache Cassandra® Database now supports the creation of an SSL-enabled connection to a Cassandra® database. For details about creating an SSL-enabled connection, see the `cassandra` function.

Export data from MATLAB into wide column database

The Database Toolbox Interface for Apache Cassandra Database now supports exporting data from MATLAB into a Cassandra database, which is a wide column database. You can insert and update data in a Cassandra database by using the `upsert` function.

Database Explorer app streamlined connection workflow

The **Database Explorer** app has a streamlined workflow for creating and closing database connections and creating SQL queries. In the new **Connections** section of the **Database Explorer** tab, you can create a database connection to a configured data source by selecting it from the **Connect** list. Then, you can create an SQL query using the database connection by selecting it from the **New Query** list. Close the database connection by selecting it from the **Close Connection** list.



Configure JDBC data source programmatically

You can configure a JDBC data source at the command line. The `JDBCConnectionOptions` object contains all properties for a configured JDBC data source. These functions enable you to configure a JDBC data source, set various JDBC connection options, test the connection, and save the data source, and delete the data source:

- `configureJDBCDataSource`
- `setConnectionOptions`
- `testConnection`
- `saveAsJDBCDataSource`
- `deleteJDBCDataSource`

Specify driver-specific connection options

When you configure a JDBC data source, you can now specify JDBC driver-specific connection options by using the JDBC Data Source Configuration dialog box or the command line. For details about the dialog box, see the `configureJDBCDataSource` function. For details about specifying options at the command line, see the `addConnectionOptions` function. Or, for details about removing options, see the `rmConnectionOptions` function.

Create and execute SQL prepared statements

You can create an SQL prepared statement using the `databasePreparedStatement` function, which creates an `SQLPreparedStatement` object. With this object, you can bind values to parameters in the SQL prepared statement by using the `bindParamValues` function. Then, you can execute the SQL prepared statement with one or more parameter values by using the `fetch` or `execute` function. For details, see SQL Prepared Statements.

Use an SQL prepared statement for improved performance and security.

R2019a

Configure JDBC data sources using configureJDBCDataSource function

You can create and edit JDBC data sources using the `configureJDBCDataSource` function. After you specify the location of the JDBC driver file in the JDBC Data Source Configuration dialog box, the software adds the JDBC driver file to the Java® class path automatically. For examples, see Configuring Driver and Data Source. Then, you can use a JDBC data source in the `database` function to connect to a database.

Initialize parallel pool with database connection using createConnectionForPool function

Use the `createConnectionForPool` function to initialize a parallel pool with a database connection in one step. With Parallel Computing Toolbox™, you can use a JDBC or ODBC data source to work with databases in parallel.

Customize import options for text data types and missing data

You can customize import options for text data types and missing data using the `setoptions` function.

After customizing any import options, you can preview the data using the `preview` function before you import the data.

Customize import options using the Database Explorer app

You can customize general, text, datetime, and categorical import options using the **Database Explorer** app. For details, see Customize Import Options Using Database Explorer App.

Update node labels, node properties, and relationship properties in Neo4j database

You can use these functions to add or delete node labels and properties, and add or delete relationship properties, in a Neo4j database:

- `addNodeLabel`
- `removeNodeLabel`
- `removeNodeProperty`
- `removeRelationProperty`
- `setNodeProperty`
- `setRelationProperty`

These functions enable you to make simple updates without modifying the entire node or relationship.

To improve performance, all functions that update the graph database include a new syntax without output arguments. For details, see Graph Database.

Database Toolbox Interface for Neo4j Bolt Protocol enables interaction with Neo4j database using Bolt protocol

The Database Toolbox Interface for Neo4j Bolt Protocol enables you to interact with a Neo4j database using the Bolt protocol instead of the REST API. In general, you can experience improved performance because the Bolt protocol sends data of a smaller size. To use this interface, you must first install the Database Toolbox Interface for Neo4j Bolt Protocol. For details, see Database Toolbox Interface for Neo4j Bolt Protocol Installation. For details about the workflow, see Graph Database Workflow for Neo4j Database Interfaces.

Functionality being removed or changed

tableprivileges function has been removed

Errors

The `tableprivileges` function with the `dmd` object has been removed with no replacement.

columnprivileges function has been removed

Errors

The `columnprivileges` function with the `dmd` object has been removed with no replacement.

get function has been removed

Errors

The `get` function with the `dmd` object has been removed with no replacement.

runsqlscript function will be removed

Still runs

The `runsqlscript` function will be removed in a future release. Use the `executeSQLScript` function instead. Some differences between the output arguments might require updates to your code.

Update Code

In prior releases, the output argument of the `runsqlscript` function was a `cursor` array. For example:

```
datasource = 'MS SQL Server Auth';
conn = database(datasource,'','');
scriptfile = 'compare_sales.sql';
results = runsqlscript(conn,scriptfile)
```

```
results =

  1×2 cursor array with properties:

    Data
    RowLimit
    SQLQuery
    Message
    Type
    Statement
```

```
    Position
```

Now the `executeSQLScript` function returns a structure array.

```
datasource = 'MS SQL Server Auth';
conn = database(datasource,'','');
scriptfile = 'compare_sales.sql';
results = executeSQLScript(conn,scriptfile)
```

```
results = 1×2 struct array with fields:
    SQLQuery
    Data
    Message
```

You can also change the data return format of the results in the structure array by using the `'DataReturnFormat'` name-value pair argument.

setdbprefs function is not recommended
Still runs

The `setdbprefs` function is not recommended. Use the following replacement functionality to specify the data return format, error handling, and missing data. Some differences between the workflows might require updates to your code.

- Data return format — For the `'DataReturnFormat'` database preference, these values are not recommended:
  - `'numeric'`
  - `'cellarray'`
  - `'structure'`
- Error handling — The `'ErrorHandling'` database preference is not recommended.
- Missing data — The `'NullNumberWrite'`, `'NullStringWrite'`, and `'NullNumberRead'` database preferences for handling `NULL` data values are not recommended.

There are no plans to remove the `setdbprefs` function at this time.

Update Code

To set the data return format in prior releases, you specified returning imported data as a numeric matrix by setting the `'DataReturnFormat'` database preference to the value `'numeric'`. For example:

```
setdbprefs('DataReturnFormat','numeric')
results = fetch(conn,sqlquery);
```

Now you can set the same value by using the `'DataReturnFormat'` name-value pair argument of the `fetch` function.

```
results = fetch(conn,sqlquery,'DataReturnFormat','numeric');
```

Or, you can customize import options.

```
opts = databaseImportOptions(conn,tablename);
varnames = "quantity";
opts = setoptions(opts,varnames,'Type','int64');
```

To specify error handling in prior releases, you set the `'ErrorHandling'` database preference to the value `'report'` or `'store'` by using the `setdbprefs` function. For example:

```
setdbprefs('ErrorHandling','store')
```

Now you specify error handling by using the `'ErrorHandling'` name-value pair argument of the `database` function or the `'ErrorHandling'` name-value pair argument of the `executeSQLScript` function.

```
conn = database(datasource,username,password,'ErrorHandling','store');
```

To specify the handling of missing data in prior releases, you set the `'NullNumberWrite'` database preference to a specific value, for example. This table shows database preference settings that are not recommended and the functionality you can use instead.

Discouraged Functionality	Recommended Replacement
`setdbprefs('NullNumberWrite','NaN')`	`data` input argument of `sqlwrite`
`setdbprefs('NullStringWrite','null')`	`data` input argument of `sqlwrite`
`setdbprefs('NullNumberRead','0')`	`SQLImportOptions` object

R2018b

Database Toolbox Interface for Apache Cassandra Database: Explore keyspaces and tables of a Cassandra database and import data as MATLAB types

With the Database Toolbox Interface for Apache Cassandra Database, you can explore the Cassandra database, which is a wide-column store. You can access the database keyspaces, tables, and columns, and execute queries using the Cassandra Query Language (CQL). Also, you can import data from partitions of a Cassandra database table into MATLAB. The data import converts CQL data types into MATLAB types. For details, see Columnar Database.

Customize Import Options: Define and customize an import strategy for relational database data and minimize post-processing steps on imported data

With a relational database connection, you can create the `SQLImportOptions` object using the `databaseImportOptions` function. Use this object to customize options for importing data from a database into MATLAB. You can also retrieve the current import options using the `getoptions` function. Then, you can customize import options using the `setoptions` function. To set the import options back to their default values, use the `reset` function. For a workflow example, see Customize Options for Importing Data from Database into MATLAB.

View row count of SQL query results in Database Explorer app

The **Show Row Count** check box in the **Preview** section of the **Database Explorer** app enables you to view the total row count of the SQL query results. You can select and clear this check box when previewing data. For details, see Data Preview Using Database Explorer App.

Execute Non-SELECT SQL Statements

The `execute` function executes SQL queries that contain non-`SELECT` SQL statements. Use the `execute` function for these SQL statements instead of the `exec` function.

Functionality being removed or changed

cursor object is not recommended
Still runs

The `cursor` object is not recommended. Use the `fetch` function instead. Some differences between the workflows might require updates to your code.

There are no plans to remove the `cursor` object at this time.

Scrollable cursors have no replacement functionality.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object and import data. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

close function is not recommended
Still runs

The `close` function with the `cursor` object is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `close` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, import data, and close the `cursor` object. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
attributes = attr(curs);
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

attr function is not recommended
Still runs

The `attr` function is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `attr` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, import data, and find the attributes of the database columns. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
attributes = attr(curs);
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

If you return imported data as a table, access details about the variables in the table by using a function such as `summary`.

cols function is not recommended

Still runs

The `cols` function is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `cols` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, import data, and find the number of database columns in the imported data. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
numcols = cols(curs);
close(curs)
```

Now you can import data in one step using the `fetch` function.

```
results = fetch(conn,sqlquery);
sz = size(results);
```

If you return imported data as a table, access details about the variables in the table by using functions such as `summary` or `size`. Or, view the dimensions of the returned data in the Workspace browser.

columnnames function is not recommended

Still runs

The `columnnames` function is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `columnnames` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, import data, and find the names of the database columns. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
columnlist = columnnames(curs);
close(curs)
```

Now you can import data in one step using the `fetch` function, and then access properties of the output table.

```
results = fetch(conn,sqlquery);
results.Properties.VariableNames
```

If you return imported data as a table, access details about the variables in the table by using the table properties or a function such as `summary`.

get function is not recommended
Still runs

The `get` function with the `cursor` object is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `get` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, retrieve object properties, and import data. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
s = get(curs);
results = curs.Data;
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

isopen function is not recommended
Still runs

The `isopen` function with the `cursor` object is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `isopen` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, determine if the database cursor is open, and import data. For example:

```
curs = exec(conn,sqlquery);
i = isopen(curs);
curs = fetch(curs);
results = curs.Data;
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

set function is not recommended
Still runs

The `set` function with the `cursor` object is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `set` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, set object properties, and import data. For example:

```
curs = exec(conn,sqlquery);
set(curs,'RowLimit',5)
curs = fetch(curs);
s = get(curs);
results = curs.Data;
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

querytimeout function is not recommended

Still runs

The `querytimeout` function is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `querytimeout` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, import data, and determine the current database timeout value. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
timeout = querytimeout(curs);
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

There is no replacement functionality for the `querytimeout` function.

rows function is not recommended

Still runs

The `rows` function is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `rows` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, import data, and find the number of rows in the imported data. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
numrows = rows(curs);
close(curs)
```

Now you can import data in one step using the `fetch` function.

```
results = fetch(conn,sqlquery);
sz = size(results);
```

If you return imported data as a table, access details about the variables in the table by using functions such as `summary` or `size`. Or, view the dimensions of the returned data in the Workspace browser.

width function is not recommended

Still runs

The `width` function is not recommended. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

There are no plans to remove the `width` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object, import data, and determine the field size of a specified column number. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
colsize = width(curs,1);
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

There is no replacement functionality for the `width` function.

exec function is not recommended

Still runs

The `exec` function with the `connection` object is not recommended. For SQL statements that return data, use the `fetch` function with the `connection` object or the `select` function instead. For other SQL statements, use the `execute` function instead. Some differences between the workflows might require updates to your code.

There are no plans to remove the `exec` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object and import data. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

You can also import data in one step using the `select` function.

```
data = select(conn,selectquery);
```

The scrollable cursor functionality has no replacement.

fetch function with the cursor object is not recommended
Still runs

The `fetch` function with the `cursor` object is not recommended. Use the `fetch` function with the `connection` object or the `select` function instead. Some differences between the workflows might require updates to your code.

There are no plans to remove the `fetch` function at this time.

Update Code

Use the `fetch` function with the `connection` object to import data from a database in one step.

In prior releases, you wrote multiple lines of code to create the `cursor` object and import data. For example:

```
curs = exec(conn,sqlquery);
curs = fetch(curs);
results = curs.Data;
close(curs)
```

Now you can import data in one step using the `fetch` function.

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

You can also import data in one step using the `select` function.

```
data = select(conn,selectquery);
```

The scrollable cursor functionality has no replacement.

fetch function returns results as a table
Behavior change

The `fetch` function returns results as a table by default instead of a cell array. When the `fetch` function finds no data to import, the function returns an empty table instead of a cell array containing the `'No Data'` character vector.

fetch function ignores 'DataReturnFormat', 'NullNumberRead', and 'NullStringRead' database preferences
Behavior change

The `fetch` function ignores these database preferences:

- `'DataReturnFormat'`
- `'NullNumberRead'`
- `'NullStringRead'`

You can set the data type of the imported data using the `'DataReturnFormat'` name-value pair argument of the `fetch` function. For more customization of data types and fill values for missing data in the imported data, use the `SQLImportOptions` object.

fetchmulti function is not recommended
Still runs

The `fetchmulti` function is not recommended. There is no replacement for the `fetchmulti` function.

'DataReturnFormat' database preference values will be removed
Still runs

These values for the `'DataReturnFormat'` database preference of the `setdbprefs` function will be removed in a future release:

- `'numeric'`
- `'cellarray'`
- `'structure'`
- `'dataset'`

Use the `'DataReturnFormat'` name-value pair argument of the `fetch` function instead. Some differences between these syntaxes require updates to your code.

Update Code

In prior releases, you specified returning imported data as a structure by setting the `'DataReturnFormat'` database preference to the value `'structure'`.

```
setdbprefs('DataReturnFormat','structure')
results = fetch(conn,sqlquery);
```

Now set the same value by using the `'DataReturnFormat'` name-value pair argument of the `fetch` function.

```
results = fetch(conn,sqlquery,'DataReturnFormat','structure');
```

insert function will be removed
Still runs

The `insert` function that uses the `connection` object will be removed in a future release. Use the `sqlwrite` function instead.

Some differences between these functions require updates to your code.

Update Code

In prior releases, you specified a cell array when exporting data from the MATLAB workspace into a database. For example:

```
colnames = {'Month','salesTotal','Revenue'};
data = {'March',50,2000};
tablename = 'yearlySales';
insert(conn,tablename,colnames,data)
```

Now the `sqlwrite` function requires you to specify the data to export as a table.

```
colnames = {'Month','salesTotal','Revenue'};
d = {'March',50,2000};
data = cell2table(d,'VariableNames',colnames);
tablename = 'yearlySales';
sqlwrite(conn,tablename,data)
```

dmd function will be removed

Warns

The `dmd` function will be removed in a future release. Use the `sqlfind` function or access the properties of the `connection` object instead.

Some differences between these functions require updates to your code.

Update Code

In prior releases, you retrieved information about a database connection by using the `dmd` function. For example:

```
conn = database(datasource,username,password);
dbmeta = dmd(conn);
```

Now the `sqlfind` function returns information about the table types in the database.

```
conn = database(datasource,username,password);
data = sqlfind(conn,pattern);
```

Also, you can access the database properties and catalog and schema information by using the `connection` object. For example:

```
conn = database(datasource,username,password);
catalogs = conn.Catalogs;
schemas = conn.Schemas;
```

get function will be removed

Warns

The `get` function with the `dmd` object will be removed in a future release. There is no replacement for the `get` function.

tables function will be removed

Warns

The `tables` function will be removed in a future release. Use the `sqlfind` function or access the properties of the `connection` object instead.

Some differences between these functions require updates to your code.

Update Code

In prior releases, you retrieved information about database tables by using the `tables` function. For example:

```
conn = database(datasource,username,password);
t = tables(conn,catalog,schema);
```

Now the `sqlfind` function returns information about the table types in a database.

```
conn = database(datasource,username,password);
data = sqlfind(conn,pattern);
```

Also, you can access the database properties and catalog and schema information by using the `connection` object. For example:

```
conn = database(datasource,username,password);
catalogs = conn.Catalogs;
schemas = conn.Schemas;
```

columns function will be removed
Warns

The `columns` function will be removed in a future release. Use the `sqlfind` function or access the properties of the `connection` object instead.

Some differences between these functions require updates to your code.

Update Code

In prior releases, you retrieved information about database columns by using the `columns` function. For example:

```
conn = database(datasource,username,password);
columnlist = columns(conn,catalog);
```

Now the `sqlfind` function returns information about the table types in the database (including information about database columns).

```
conn = database(datasource,username,password);
data = sqlfind(conn,pattern);
```

Also, you can access the database properties and catalog and schema information by using the `connection` object. For example:

```
conn = database(datasource,username,password);
catalogs = conn.Catalogs;
schemas = conn.Schemas;
```

tableprivileges function will be removed
Warns

The `tableprivileges` function will be removed in a future release. There is no replacement for the `tableprivileges` function.

columnprivileges function will be removed
Warns

The `columnprivileges` function will be removed in a future release. There is no replacement for the `columnprivileges` function.

dexplore function has been removed
Errors

The `dexplore` function has been removed. Enter `databaseExplorer` at the command line to open the **Database Explorer** app instead.

R2018a

Neo4j Graph Database Upgrade: Write a directed graph to a Neo4j database, and create, update, and delete nodes and relationships

Using the MATLAB Interface to Neo4j, you can connect to a Neo4j graph database and import graph data into MATLAB. With this upgrade, you can export data in a directed graph from MATLAB into a Neo4j database. Also, you can create, update, and delete nodes and relationships in the Neo4j database. For details about using the MATLAB interface to Neo4j, see Graph Database. For a basic workflow, see Working with MATLAB Interface to Neo4j.

SQL-Speaking Functions: Read and write data and perform joins on database tables directly from MATLAB without writing SQL queries

If you are not familiar with writing SQL queries, you can use these command line functions to import data from and insert data into a database:

- `sqlread`
- `sqlinnerjoin`
- `sqlouterjoin`
- `sqlwrite`

The `sqlwrite` function replaces the `datainsert`, `fastinsert`, and `insert` functions, which insert data into a database. For details, see Inserting Data Using Command Line. For examples that show how to migrate to the `sqlwrite` function, see Append Data to Existing Database Table Using Insert Functionality and Insert Data into New Database Table Using Insert Functionality.

Also, you can find information (including table types) about a database by using the `sqlfind` function.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`datainsert`	Still runs	`sqlwrite`	Replace all instances of the `datainsert` function with the `sqlwrite` function.
`fastinsert`	Still runs	`sqlwrite`	Replace all instances of the `fastinsert` function with the `sqlwrite` function.
`dmd`	Still runs	`sqlfind` and `connection` object properties	Replace all instances of the `dmd` function with the `sqlfind` function or access the properties of the `connection` object.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`tables`	Still runs	`sqlfind` and `connection` object properties	Replace all instances of the `tables` function with the `sqlfind` function or access the properties of the `connection` object.
`columns`	Still runs	`sqlfind` and `connection` object properties	Replace all instances of the `columns` function with the `sqlfind` function or access the properties of the `connection` object.
`tableprivileges`	Still runs	`sqlfind` and `connection` object properties	Replace all instances of the `tableprivileges` function with the `sqlfind` function or access the properties of the `connection` object.
`columnprivileges`	Still runs	`sqlfind` and `connection` object properties	Replace all instances of the `columnprivileges` function with the `sqlfind` function or access the properties of the `connection` object.
`setdbprefs('NullNumberWrite','NaN')`	Still runs	`sqlwrite`	Replace all instances of `setdbprefs('NullNumberWrite','NaN')` with the `sqlwrite` function. Specify `NULL` values using the `data` input argument of the `sqlwrite` function.
`setdbprefs('NullStringWrite','null')`	Still runs	`sqlwrite`	Replace all instances of `setdbprefs('NullStringWrite','null')` with the `sqlwrite` function. Specify `NULL` values using the `data` input argument of the `sqlwrite` function.
`setdbprefs('NullNumberRead','0')`	Still runs	Default value `NaN` for numeric `NULL` values	Replace all instances of `setdbprefs('NullNumberRead','0')` with `NaN` values for numeric `NULL` values.
`setdbprefs('ErrorHandling','store')`	Still runs	`setdbprefs('ErrorHandling','report')`	Replace all instances of `setdbprefs('ErrorHandling','store')` with `setdbprefs('ErrorHandling','report')`.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`setdbprefs('DataReturnFormat','numeric')`	Still runs	`setdbprefs('DataReturnFormat','table')`	Replace all instances of `setdbprefs('DataReturnFormat','numeric')` with `setdbprefs('DataReturnFormat','table')`.
`setdbprefs('DataReturnFormat','cellarray')`	Still runs	`setdbprefs('DataReturnFormat','table')`	Replace all instances of `setdbprefs('DataReturnFormat','cellarray')` with `setdbprefs('DataReturnFormat','table')`.
`setdbprefs('DataReturnFormat','structure')`	Still runs	`setdbprefs('DataReturnFormat','table')`	Replace all instances of `setdbprefs('DataReturnFormat','structure')` with `setdbprefs('DataReturnFormat','table')`.
`database.ODBCConnection` syntax in the `database` function	Warns	`database` syntax	Replace all instances of the `database.ODBCConnection` syntax with the `database` syntax in the `database` function.
`dexplore`	Warns	`databaseExplorer`	To open the **Database Explorer** app, use the `databaseExplorer` function.
`catalogs`	Errors	`Catalogs` property of the `connection` object	Replace all instances of the `catalogs` function with access of the `Catalogs` property.
`exportedkeys`	Errors	Nothing	No replacement
`get`	Errors	`connection` object properties	Access any `connection` object property.
`importedkeys`	Errors	Nothing	No replacement
`indexinfo`	Errors	Nothing	No replacement
`isreadonly`	Errors	`ReadOnly` property of the `connection` object	Replace all instances of the `isreadonly` function with access of the `ReadOnly` property.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`logintimeout`	Errors	Name-value pair argument `'LoginTimeout'` in the `database` function	Remove all instances of the `logintimeout` function and specify the timeout value by using the `'LoginTimeout'` name-value pair argument.
`ping`	Errors	`connection` object properties	Access these `connection` object properties: • `MaxDatabaseConnections` • `DatabaseProductName` • `DatabaseProductVersion` • `DriverName` • `DriverVersion`
`primarykeys`	Errors	Nothing	No replacement
`procedures`	Errors	Nothing	No replacement
`procedurecolumns`	Errors	Nothing	No replacement
`schemas`	Errors	`Schemas` property of the `connection` object	Replace all instances of the `schemas` function with access of the `Schemas` property.
`set`	Errors	`connection` object properties	You can set only the `AutoCommit` and `ReadOnly` properties.
`sql2native`	Errors	Nothing	No replacement
`supports`	Errors	Nothing	No replacement
`setdbprefs('DataReturnFormat','dataset')`	Errors	`setdbprefs('DataReturnFormat','table')`	Replace all instances of `setdbprefs('DataReturnFormat','dataset')` with `setdbprefs('DataReturnFormat','table')`.
`setdbprefs('FetchBatchSize','1000')`	Errors	Nothing	The software sets the default value.
`setdbprefs('FetchInBatches','yes')`	Errors	Nothing	The software performs batching by default.

R2017b

Database Explorer App: Visually explore relational databases without knowing SQL

The **Database Explorer** app has been replaced. You can configure ODBC and JDBC data sources, explore data in a relational database, create SQL queries interactively, and import data into the MATLAB workspace.

Compatibility Considerations

- To open the new Database Explorer app, use the `databaseExplorer` function instead of the `dexplore` function.
- For JDBC data sources only, set up new JDBC data sources to connect to a database using the JDBC driver. For details, see Configuring Driver and Data Source.
- The syntax `setdbprefs('JDBCDataSourceFile')` has been removed.
- The new Database Explorer app has new workflows for connecting to databases and creating SQL queries. For details, see Create SQL Queries Using Database Explorer App.

Database Toolbox Interface for MongoDB: Easily interact with MongoDB

With the Database Toolbox interface for MongoDB, you can import data stored in a collection of documents in MongoDB for analysis in MATLAB. After connecting to MongoDB, you can also explore and manage collections, and export data from MATLAB into MongoDB.

For details about installing the Database Toolbox interface for MongoDB, see Database Toolbox Interface for MongoDB Installation. For details about the interface, see Document Database.

splitsqlquery Function: Split a SQL query into multiple SQL page queries to access large amounts of data

Using the `splitsqlquery` function, you can split a SQL query into multiple SQL page queries and then access large data in chunks. You can access large data using Database Toolbox functions or using a parallel pool (requires Parallel Computing Toolbox).

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`catalogs`	Warns	`Catalogs` property of the `connection` object	Replace all instances of the `catalogs` function with access of the `Catalogs` property.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`database.ODBCConnection` syntax in the `database` function	Warns	`database` syntax	Replace all instances of the `database.ODBCConnection` syntax with the `database` syntax in the `database` function.
`dexplore`	Warns	`databaseExplorer`	To open the **Database Explorer** app, use the `databaseExplorer` function.
`exportedkeys`	Warns	Nothing	No replacement
`get`	Warns	`connection` object properties	Access any `connection` object property.
`importedkeys`	Warns	Nothing	No replacement
`indexinfo`	Warns	Nothing	No replacement
`isreadonly`	Warns	`ReadOnly` property of the `connection` object	Replace all instances of the `isreadonly` function with access of the `ReadOnly` property.
`logintimeout`	Warns	Name-value pair argument `'LoginTimeout'` in the `database` function	Remove all instances of the `logintimeout` function and specify the timeout value using the name-value pair argument `'LoginTimeout'`.
`ping`	Warns	`connection` object properties	Access these `connection` object properties: • `MaxDatabaseConnections` • `DatabaseProductName` • `DatabaseProductVersion` • `DriverName` • `DriverVersion`
`primarykeys`	Warns	Nothing	No replacement
`procedure`	Warns	Nothing	No replacement
`procedurecolumns`	Warns	Nothing	No replacement
`schemas`	Warns	`Schemas` property of the `connection` object	Replace all instances of the `schemas` function with access of the `Schemas` property.

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`set`	Warns	`connection` object properties	You can set only the `AutoCommit` and `ReadOnly` properties.
`setdbprefs('DataReturnFormat','dataset')`	Warns	`table` data type	Replace all instances of `setdbprefs('DataReturnFormat','dataset')` with `setdbprefs('DataReturnFormat','table')`.
`setdbprefs('FetchBatchSize','1000')`	Warns	Nothing	No replacement
`setdbprefs('FetchInBatches','yes')`	Warns	Nothing	No replacement
`sql2native`	Warns	Nothing	No replacement
`supports`	Warns	Nothing	No replacement
`namecolumn`	Errors	Nothing	No replacement
`resultset`	Errors	Nothing	No replacement
`resultset` object as input argument in `close`	Errors	Nothing	No replacement
`resultset` object as input argument in `get`	Errors	Nothing	No replacement
`resultset` object as input argument in `namecolumn`	Errors	Nothing	No replacement
`rsmd`	Errors	Nothing	No replacement
`rsmd` object as input argument in `get`	Errors	Nothing	No replacement
`setdbprefs('ErrorHandling','empty')`	Errors	Nothing	Replace all instances of `setdbprefs('ErrorHandling','empty')` with `setdbprefs('ErrorHandling','report')`.
`setdbprefs('JDBCDataSourceFile')`	Errors	Nothing	Configure JDBC data sources by using the Database Explorer app. For details, see Configuring Driver and Data Source.

Compatibility Considerations

In the **Database Toolbox Preferences** pane:

- The **Cursor Fetch** pane has been removed. The **Fetch In Batches** and **Batch Size** preferences have been removed. Use the `setdbprefs` function at the command line instead.

- The `dataset` option in the **Data Return Format** list will be removed. Use `table` instead.
- The `empty` option in the **Error Handling** list will be removed. Use `report` instead.

R2017a

One-step data import

You can import data from a database in one step using the `select` function. This function enables maximum memory savings by importing numeric values using data types as defined in the database. For different ways to import data, see Data Import Using Database Explorer App or Command Line.

Expanded data type support

You can import data with a larger variety of data types into MATLAB. For the full list, see Data Type Support.

Connection object and database connection changes

The `connection` object has additional properties grouped into these categories:

- `Database Properties`
- `Catalog and Schema Information`
- `Database and Driver Information`

Additional properties include:

- `ReadOnly`
- `MaxDatabaseConnections`
- `DefaultCatalog`
- `Catalogs`
- `Schemas`
- `DatabaseProductName`
- `DatabaseProductVersion`
- `DriverName`
- `DriverVersion`

For ODBC drivers, the native ODBC interface is now the default database connection type.

Compatibility Considerations

These `connection` object properties have changed:

- `Handle` and `Constructor` is removed.
- `DataSource` replaces `Instance`.
- `LoginTimeout` replaces `TimeOut`.

For details about object properties, see the `connection` object.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`catalogs`	Still runs	`Catalogs` property of the `connection` object	Replace all instances of the `catalogs` function with access of the `Catalogs` property.
`get`	Still runs	`connection` object properties	Access any `connection` object property.
`isreadonly`	Still runs	`ReadOnly` property of the `connection` object	Replace all instances of the `isreadonly` function with access of the `ReadOnly` property.
`logintimeout`	Still runs	Name-value pair argument `'LoginTimeout'` in the `database` function	Remove all instances of the `logintimeout` function and specify the timeout value using the name-value pair argument `'LoginTimeout'`.
`ping`	Still runs	`connection` object properties	Access these `connection` object properties: • `MaxDatabaseConnections` • `DatabaseProductName` • `DatabaseProductVersion` • `DriverName` • `DriverVersion`
`schemas`	Still runs	`Schemas` property of the `connection` object	Replace all instances of the `schemas` function with access of the `Schemas` property.
`set`	Still runs	`connection` object properties	You can set only the `AutoCommit` and `ReadOnly` properties.
`sql2native`	Still runs	Nothing	No replacement
`setdbprefs('FetchInBatches','yes')`	Still runs	Nothing	No replacement
`setdbprefs('FetchBatchSize','1000')`	Still runs	Nothing	No replacement

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`database.ODBCConnection` syntax in the `database` function	Warns	`database` syntax	Replace all instances of the `database.ODBCConnection` syntax with the `database` syntax in the `database` function.
`resultset`	Warns	Nothing	No replacement
`resultset` object as input argument in `close`	Warns	Nothing	No replacement
`resultset` object as input argument in `get`	Warns	Nothing	No replacement
`resultset` object as input argument in `namecolumn`	Warns	Nothing	No replacement
`rsmd`	Warns	Nothing	No replacement
`rsmd` object as input argument in `get`	Warns	Nothing	No replacement
`setdbprefs('DataReturnFormat','dataset')`	Warns	`table` data type	Replace all instances of `setdbprefs('DataReturnFormat','dataset')` with `setdbprefs('DataReturnFormat','table')`.
`setdbprefs('ErrorHandling','empty')`	Warns	Nothing	Replace all instances of `setdbprefs('ErrorHandling','empty')` with `setdbprefs('ErrorHandling','report')`.
`isconnection`	Errors	`isopen`	Replace all instances of the `isconnection` function with `isopen`.
`querybuilder`	Errors	`dexplore`	Use the **Database Explorer** app.
`setdbprefs('DefaultRowPreFetch','10000')`	Errors	Nothing	No replacement
`setdbprefs('TempDirForRegistryOutput','')`	Errors	Nothing	No replacement
`setdbprefs('UseRegistryForSources','')`	Errors	Nothing	No replacement

Compatibility Considerations

The JDBC/ODBC bridge has been removed for the command line. For ODBC drivers, the syntaxes of the `database` function and other command-line functions use the native ODBC interface by default. The Database Explorer app still uses the JDBC/ODBC bridge for ODBC database connection.

Visual Query Builder has been removed. Use the Database Explorer app instead. When using the **Database Explorer** app with JDBC drivers for the first time, you must configure data sources.

R2016b

Graph Database Interface: Retrieve graph data from Neo4j

To import graph data in a Neo4j database into MATLAB, use the MATLAB interface to Neo4j. To perform graph network analysis with graph data in MATLAB using the `digraph` object, create a Neo4j database connection. Or, you can explore the graph using MATLAB functionality. If you are familiar with the Cypher® query language, you can execute Cypher queries. For details about using the MATLAB interface to Neo4j, see Graph Database.

DatabaseDatastore functionality and object properties changes

To analyze data using common MATLAB functions, such as `mean` and `histogram`, you can create a tall array using the `DatabaseDatastore` object. For details about using tall arrays with Database Toolbox, see Analyze Large Data in Database Using Tall Arrays.

To create a `DatabaseDatastore` object, use `databaseDatastore` instead of `datastore`.

The `DatabaseDatastore` object has two additional object properties. The `VariableNames` property provides the variable names of the retrieved data table. The `ReadSize` property specifies the number of rows to read from the retrieved data table.

The `read`, `readall`, and `preview` functions return data as a `table`.

Compatibility Considerations

- The `DatabaseDatastore` object properties have changed:
  - The `Cursor` property has been removed.
  - Two properties are added: `VariableNames` and `ReadSize`.

  For details about the object properties, see `DatabaseDatastore`.
- This syntax for the `read` function has been removed.

  ```
  data = read(dbds,rowcount)
  ```

  To specify the number of rows to retrieve, set the `ReadSize` property of the `DatabaseDatastore` object instead.
- The data retrieval functionality has changed:
  - The functions retrieve data as a `table` by default. Setting the database preference `DataReturnFormat` to `'table'` is not required.
  - `read` and `preview` throw an error when there is no more data in the `DatabaseDatastore` object for reading.
  - If `read` finds no more data to read, `hasdata` returns logical 0.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`bestrowid`	Errors	Nothing	No replacement

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`clearwarnings`	Errors	Nothing	No replacement
`crossreference`	Errors	Nothing	No replacement
`driver`	Errors	Nothing	No replacement
`drivermanager`	Errors	Nothing	No replacement
`isdriver`	Errors	Nothing	No replacement
`isjdbc`	Errors	Nothing	No replacement
`isnullcolumn`	Errors	Nothing	No replacement
`isurl`	Errors	Nothing	No replacement
`register`	Errors	Nothing	No replacement
`unregister`	Errors	Nothing	No replacement
`versioncolumns`	Errors	Nothing	No replacement
`rsmd`	Still runs	Nothing	No replacement
`resultset`	Still runs	Nothing	No replacement
`tables(conn)`	Errors	Use syntaxes with at least two input arguments instead.	Remove all instances of the `tables(conn)` syntax. Replace these instances with any of the remaining syntaxes. For details, see `tables`.

R2016a

MATLAB Interface to SQLite: Create, read, and write data from SQLite database files without external drivers and administration

To import and update data without an existing database or database administration, use the MATLAB Interface to SQLite. For details, see Working with the MATLAB Interface to SQLite. For the supported functions, see this table.

Function	Purpose
`sqlite`	Create SQLite connection.
`exec`	Perform database operation in the SQLite database file.
`fetch`	Run SQL query and import data from the SQLite database file.
`insert`	Export data into the SQLite database file.
`close`	Close SQLite connection.

fetch Function Speed Improvement: Import data faster using the JDBC driver

When you connect to a database using the JDBC driver, importing data is faster using the `fetch` function.

Support for 32-bit Windows removed

The Database Toolbox no longer supports connection to a database using a 32-bit driver.

Compatibility Considerations

Use a 64-bit database. Or, install a 64-bit driver that works with the 32-bit database. For details, consult with your database administrator.

For Microsoft® Access™, see https://www.mathworks.com/matlabcentral/answers/235949-how-to-connect-to-32-bit-microsoft-access-database-from-64-bit-matlab.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`clearwarnings`	Warns	Nothing	No replacement
`versioncolumns`	Warns	Nothing	No replacement
`crossreference`	Warns	Nothing	No replacement
`bestrowid`	Warns	Nothing	No replacement
`isnullcolumn`	Warns	Nothing	No replacement

R2015b

ODBC Interface Functions: Export and retrieve database information using native ODBC connections

More Database Toolbox functions support the native ODBC interface for exporting data and retrieving database information and metadata. For a list of supported functions, see Connecting to a Database Using the Native ODBC Interface.

Read and Write Performance Improvements: Import and export data more quickly

Data import and export functions can retrieve and write data faster. Particularly, `datainsert` and `update` call the `TRANSACTION` command of SQL to insert or update records faster for these databases: Microsoft SQL Server®, MySQL, Oracle®, and PostgreSQL.

Data Export Functions: Insert or replace data using table, structure, and dataset arrays

The `datainsert` and `update` functions can export data in tabular, `dataset`, and structure arrays to databases.

Functionality being removed or changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`driver`	Warns	Nothing	No replacement
`drivermanager`	Warns	Nothing	No replacement
`isconnection`	Warns	`isopen`	Replace all instances of `isconnection` with `isopen`.

R2015a

Version: 5.2.1

Bug Fixes

R2014b

Version: 5.2

New Features

DatabaseDatastore for applying mapreduce to data contained in relational databases

Create a `DatabaseDatastore` to work with large amounts of data in relational databases. Write custom functions to implement `mapreduce` to process large amounts of data. To create a `DatabaseDatastore`, you must create a `DatabaseDatastore` object. This object is a type of datastore.

Function	Purpose
`datastore`	Create a `DatabaseDatastore`.
`hasdata`	Determine if a `DatabaseDatastore` contains more data in the cursor object.
`preview`	Display the first eight records in a `DatabaseDatastore`.
`read`	Read data in a `DatabaseDatastore`.
`readall`	Read every record in a `DatabaseDatastore`.
`reset`	Reset the cursor position in a `DatabaseDatastore`.

Scrollable cursors for accessing data using relative and absolute position inputs

Fetch data sequentially or scroll up or down in the data without executing the query again. Scrolling within the data offers advantages when you are working with a large data set. An advantage of scrollable cursors is reading data in the middle of a large data set using the cursor position offset.

Create a scrollable cursor using `exec`. Retrieve data from a scrollable cursor using `fetch`. Use relative and absolute position inputs in `fetch` to retrieve data starting from a specific location in the data set.

R2014a

Version: 5.1

Bug Fixes

R2013b

Version: 5.0

New Features

Bug Fixes

Fast access to ODBC connections via a native ODBC driver

Support for native ODBC database connection for Windows® platforms. The native ODBC interface is available only for the command line. To use this interface, see Using the Native ODBC Database Connection. The native ODBC interface supports the following functions:

- `database`
- `fetch`
- `exec`
- `insert`
- `fastinsert`
- `close`

table data type support

You can return a `table` data type rather than a cell array. Use the `setdbprefs` command to set the database preference for the `DataReturnFormat` property to `'table'`.

R2013a

Version: 4.1

New Features

Bug Fixes

fetch function accepts user-defined batch sizes

`setdbprefs` is updated with new properties (`FetchInBatches` and `FetchBatchSize`) that support `fetch` when requesting large data.

R2012b

Database Explorer app for interactively exchanging data with databases

`dexplore` starts Database Explorer, which is the Database Toolbox GUI for connecting to a database and importing data to the MATLAB workspace. Alternatively, you can start Database Explorer by selecting **Database Explorer** from the **Database Connectivity and Reporting** section of the **Apps** tab in the MATLAB Toolstrip.

Functionality Being Removed or Changed

Functionality	What Happens When You Use It?	Use This Instead	Compatibility Considerations
`querybuilder`	Warns	`dexplore`	Continue to use `querybuilder` for exporting data.

R2012a

Version: 3.11

New Features

Bug Fixes

Execute .SQL Files

The new `runsqlscript` function lets you execute SQL commands from a `.SQL` file on a connected database, and store the results in a `cursor` array. You can input the results from executing `runsqlscript` to functions that accept `cursor` array inputs.

Improvements to the Database Constructor

When using a JDBC driver, you can input individual connection properties to the database constructor, `database`.

R2011b

Preferences Now Persistent Across MATLAB Sessions

The preferences you set using the Preference dialog box or the `setdbprefs` function now persist across MATLAB sessions.

Compatibility Considerations

In releases before R2011b, if you changed your preferences during a MATLAB session, these preferences would not remain in the next MATLAB session.

Change in Behavior for the update Function

`update` lets you update images, Booleans, doubles, and strings in a manner consistent with `fastinsert`.

Warning and Error ID Changes

Many warning and error IDs have changed from their previous versions. These warnings or errors typically appear during a function call.

Compatibility Considerations

If using warning or error IDs, you might need to change the strings you use. For example, if you turned off a warning for a certain ID, the warning might now appear under a different ID. If you use a `try/catch` statement in your code, replace the old identifier with the new identifier. There is no definitive list of the differences, or of the IDs that changed.

R2011a

Version: 3.9

New Features

Bug Fixes

New datainsert Function Exports MATLAB Cell Array Data into a Database Table

The new `datainsert` function inserts data from the MATLAB workspace into a database table, much like the `fastinsert` function. The new `datainsert` function is faster.

R2010b

Version: 3.8

New Features

Bug Fixes

Now Possible to Import Data into MATLAB Dataset Object

If you have Statistics Toolbox™ installed, you can now return a `dataset` object rather than a cell array. Use the `setdbprefs` command to set the database preference for the `DataReturnFormat` property to `'dataset'`.

R2010a

Version: 3.7

New Features

Bug Fixes

New Connection Object Methods

Several new connection object methods provide database-specific information. The new methods are:

- `database.catalogs`
- `database.columns`
- `database.schemas`
- `database.tables`

See the individual reference pages for more information on how to use these methods.

Enhanced Error Messages

New enhanced error messages provide more information about the error. For example, the 2009b error message `Drivers not Found/Loaded` is now `Drivers not Found/Loaded. Please verify that login information and database url are valid` in 2010b. This error will appear when the driver input is valid but the database URL is invalid.

Improved Write Performance

New bulk insert code templates provide significant performance upgrades.

R2009b

Version: 3.6

Bug Fixes

R2009a

Version: 3.5.1

Bug Fixes

R2008b

Version: 3.5

Bug Fixes

R2008a

Version: 3.4.1

Bug Fixes

R2007b

Version: 3.4

Bug Fixes

R2007a

Version: 3.3

New Features

Bug Fixes

setdbprefs Accepts Structure Input

The `setdbprefs` function now accepts a structure as input. For example, you can run the following commands to assign values to `s`:

```
s.DataReturnFormat = 'numeric';
s.ErrorHandling = 'report';
```

You can also do this for other `setdbprefs` properties whose values you want to change. Then set the preferences using the values in `s` by running the command:

```
setdbprefs(s)
```

For more information, see the `setdbprefs` reference page.

Visual Query Builder Generated M-File Includes Placeholder for Password and Assigns Preferences to Structure

When you run a query in the Visual Query Builder and select **File** > **Generate M-File**, the resulting M-file now includes a placeholder string `password` in the `database` statement. If a password is required for the connection, such as for connections established via JDBC drivers, substitute the password for the `password` string. If no password is required, the M-file will run as is. For more information, see About Generated Files.

The generated M-file assigns values for the preferences to the structure `s`. For more information, see the `setdbprefs` reference page.

Preference Added for Temporary Registry Output; Ensures Full Output for getdatasources

When you use `getdatasources` to view the data sources for your system, ensure that you view all data sources by specifying a temporary, writable, output directory using the new preference, `TempDirForRegistryOutput`. This is useful when you add data sources and do not have write access for the MATLAB current directory, where the toolbox temporarily writes ODBC registry settings. Without write access, `getdatasources` does not always return data sources you added. In that event, run `setdbprefs` to specify a value for the `TempDirForRegistryOutput` preference, where the value is the full path name to a directory for which you have write access.

R2006b

Enhanced fetch Combines exec with Existing fetch

The new function, `database.fetch`, executes the specified SQL query and imports results into the MATLAB workspace, given the connection handle `conn`. It is provided for convenience, to combine capabilities of the existing `exec` and `cursor.fetch` functions. In statements and code, do not specify `database.fetch` or `cursor.fetch` but rather, just specify `fetch` with the appropriate objects provided as arguments; the toolbox runs `database.fetch` or `cursor.fetch` as appropriate based on the arguments.

Unlike `cursor.fetch`, `database.fetch` does not return a cursor object on which you can run subsequent Database Toolbox functions, but rather returns all data to a MATLAB variable. For more information about `database.fetch` and how it differs from `cursor.fetch`, see the `fetch` reference page, as well as the `database.fetch` and `cursor.fetch` reference pages.

Import Data from Multiple Resultsets

The new function, `fetchmulti`, imports data into the MATLAB workspace from multiple resultsets, which you retrieve via an `exec` call to a stored procedure that contains two or more `select` statements.

Run Stored Procedures to Return Output Parameters

The new function, `runstoredprocedure`, executes a stored procedure using input parameters specified in a cell array to return output parameters. This allows you to retrieve the value of a variable into a MATLAB variable. `runstoredprocedure` overcomes a limitation of `exec`; when you run a stored procedures via `exec`, you can only retrieve resultsets.

Specify Catalog and Schema Using Visual Query Builder

You can now specify the catalog and schema for a data source using the Visual Query Builder. The `default` is none, meaning you do not need to select values for them.

Preferences Option to Find Additional Data Sources

The new `setdbprefs` option, `UseRegistryForSources`, instructs the Visual Query Builder to search the Microsoft Windows registry to find any ODBC data sources not uncovered using the system `ODBC.INI`.

MATLAB Change to Assignment of Nonscalar Structure Array Fields Might Impact Database Toolbox Users

In Version 7.3 (R2006b) of the MATLAB software, a change was made to how a nonscalar structure array field is assigned to a single MATLAB variable. For more information, see Assigning Nonscalar Structure Array Fields to a Single Variable in the MATLAB Release Notes.

Compatibility Considerations

As a result of this change in the MATLAB software, you may need to modify your Database Toolbox M-files.

R2006a

Version: 3.1.1

Bug Fixes

R14SP3

Version: 3.1

New Features

Bug Fixes

fastinsert Function Added

There is a new function, `fastinsert`, that you can use instead of the `insert` function to export data about three times more quickly than `insert`. It also allows exporting for all object types, so that any data you can retrieve from a database you now can export to the database, including binary objects.

While there are no known problems with `fastinsert`, if you receive unexpected results, return to using `insert` and report the problem with `fastinsert` via Technical Support.

Note that the Visual Query Builder insert feature uses the `insert` function instead of `fastinsert`.

JDBC Drivers Now Supported for Visual Query Builder on Microsoft Windows Systems

You now can use the Visual Query Builder (VQB) with JDBC drivers on Windows platforms. Previously, only ODBC drivers were supported.

The `confds` function now displays an enhanced dialog box you use to define JDBC data sources. With it, you save and load data source information via MATLAB MAT-files.

For details, see Setting Up JDBC Data Sources.

Define Data Sources from Within the Visual Query Builder

The Visual Query Builder now includes two new items under the **Query** menu:

- **Define ODBC Data Source**—Directly access your Windows ODBC Data Source Administrator dialog box where you define ODBC data sources.
- **Define JDBC Data Source**—Access the Define JDBC Data Source dialog box for defining JDBC data sources to use with the VQB. The function equivalent is `confds`. When you define a JDBC data source, the information is saved in a MAT-file so you can use it again in a later session. Later, open the MAT-file using the Define JDBC Data Source dialog box, or using `setdbprefs('JDBCDataSourceFile','fullpathtomatfile')`.

For details, see Configuring Your Environment.

setdbprefs Function Enhanced

New arguments are supported for defining the JDBC data source MAT-file. For details, see the `setdbprefs` reference page.

Dynamically Add JDBC Drivers File

You can dynamically add a JDBC drivers file to the MATLAB Java `classpath` using the MATLAB `javaaddpath` function. You can use this method instead of adding a pointer to the JDBC drivers file in your `classpath.txt` file. The advantage of using `javaaddpath` is that you do not have to restart the MATLAB software session after running the `javaaddpath` statement. The disadvantage is that this only applies to the current session and so you need to run the `javaaddpath` statement in each new session. For details, see Setting Up JDBC Data Sources.

64-Bit FLOAT for Microsoft SQL Server Software Is Fully Supported

You now can retrieve 64-bit `FLOAT` data using Microsoft SQL Server software.

Generate M-File from VQB

After running a query using the Visual Query Builder, you can generate an M-file consisting of Database Toolbox functions that perform the query. This is useful if you know how to run queries with the VQB and want to determine the equivalent functions, particularly the SQL statements in `exec` and `insert`.

update Function Enhanced to Export Multiple Records

The `update` function has been enhanced so that you can export multiple records based on different `where` clauses. The number of `where` clauses must equal the number of records in the array of data you are exporting. For details, see the reference page for `update`.

logintimeout Function Now Supported on Linux Platforms

The `logintimeout` function is now supported on Linux® platforms.

R14SP2

Version: 3.0.2

Bug Fixes