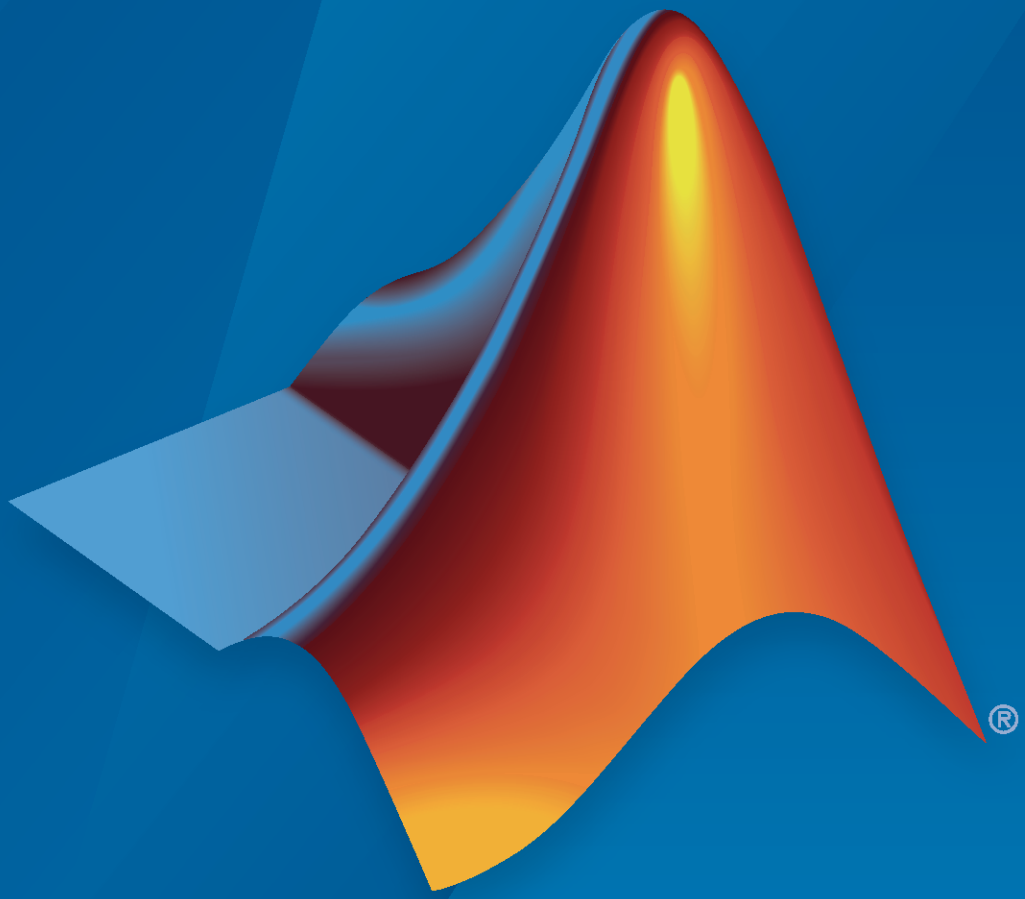
Database Toolbox™ Release Notes



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

R2026a

R2025b

R2025a

R2024b

R2024a

R2023b

R2023a

R2022b

R2022a

R2021b

R2021a

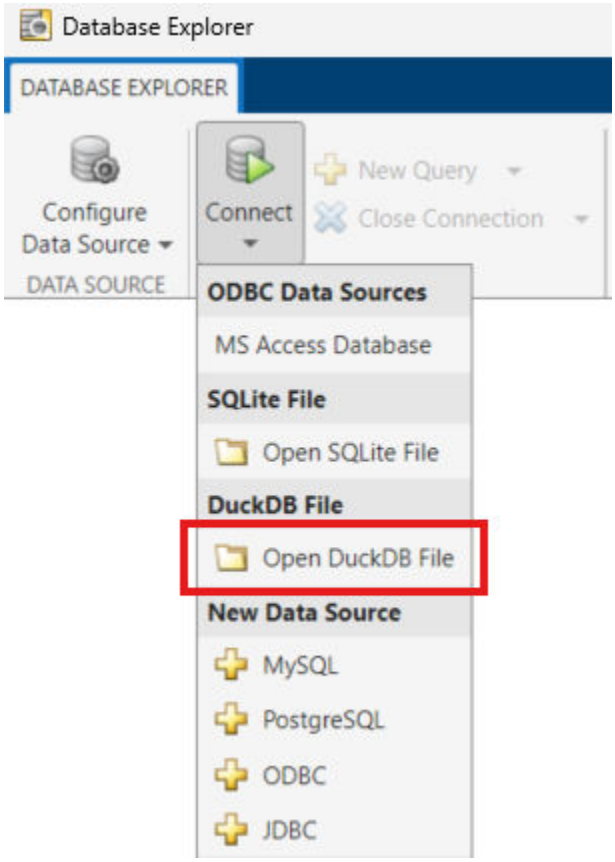
R2026a

★Relational Databases: Interact with DuckDB database engine from MATLAB

With the DuckDB™ native interface, you can create, connect, and interact with DuckDB database files and perform data analysis by directly connecting to the DuckDB engine. When you import or export data, the interface automatically handles data type conversions between DuckDB and MATLAB®. Connect to a DuckDB database file or the DuckDB engine by using the `duckdb` function.

★Relational Databases: Connect to DuckDB database files with Database Explorer

Database Explorer now supports the DuckDB native interface and allows you to interactively connect to a DuckDB database file.



Relational Databases: Import and export data with multithreading for MySQL, PostgreSQL, SQLite, and ODBC database connections

You can now use multithreading in your database workflows for increased efficiency when you import and export data. Follow these steps to use multithreading to export data to a MySQL® database table:

1. Prepare the data you want to export. Use the `parallel.pool.Constant` to avoid unnecessarily copying data multiple times in the multithreading environment.

2. Connect to the database server and create a table to store the data.
3. Use the following MATLAB code to divide the data into batches and export each batch on a separate thread by using the `sqlwrite` function. Each thread executes the `hsqlwritemultithreaded` helper function that sends the data to the server in a thread-safe manner.

```
numTasks = 5;
batchSize = floor(nRow/numTasks);
startRow = 1;
endRow = 1+batchSize;

writeFutures(1:numTasks) = parallel.FevalFuture;
pool = parpool("Threads");

for i=1:numTasks
    writeFutures(i) = parfeval(pool, @hsqlwritemultithreaded,1,server,databaseName,...
        username,password,portNumber,tablename,data,startRow,endRow);
    startRow = endRow;
    endRow = endRow + batchSize;
    if i==numTasks-1
        endRow = nRow+1;
    end
end
finishedArray = writeFutures.fetchOutputs("UniformOutput",false);

function finished = hsqlwritemultithreaded(server,databaseName,username,password,portNumber,table
    conn = mysql(username,password,"Server",server,"DatabaseName",databaseName,"PortNumber",portN
    sqlwrite(conn,tablename,data.Value(startRow:endRow-1,:));
    close(conn);
    finished = true;
end
```

Support Package: Connect to PostgreSQL database using updated ODBC driver

The PostgreSQL® ODBC driver for Database Toolbox now includes version 17.0.4 of the driver for the Windows® platform. For installation instructions, see “PostgreSQL ODBC Driver for Database Toolbox Support Package”.

MongoDB C Driver: Connect to MongoDB using version 2.1.0

Database Toolbox now includes version 2.1.0 of the MongoDB® C driver. For more information about the supported server, visit https://github.com/mongodb/mongo-c-driver/releases/tag/2.1.0.

R2025b

Version: 25.2

Bug Fixes

Quality and stability improvements

R2025b delivers quality and stability improvements, building on the new features introduced in R2025a.

R2025a

Version: 25.1

New Features

Bug Fixes

★Relational Databases: Connect to databases using stored credentials with the Database Explorer app

You can now save your username and password when connecting to a data source with the Database Explorer app. For more information, see:

- `database`
- `databaseDatastore`

For more information on how to store your credentials, see Choose How to Store Credentials for Database Connections.

★MongoDB C Driver: Connect to MongoDB using version 1.28.0

Database Toolbox now includes version 1.28.0 of the MongoDB C driver. For information about the supported server, visit https://www.mongodb.com/docs/drivers/c/.

Relational Databases: Retrieve metadata from SQL databases

Retrieve metadata about SQL database tables and queries before you import your data. For more information, see:

- `sqltableinfo`
- `sqlqueryinfo`
- `sqlStatistics`

Relational Databases: Control automatic table creation in sqlwrite

When you use the `sqlwrite` function to export data into a database table, you can use the new `checkTableExists` name-value argument to control automatic table creation.

R2024b

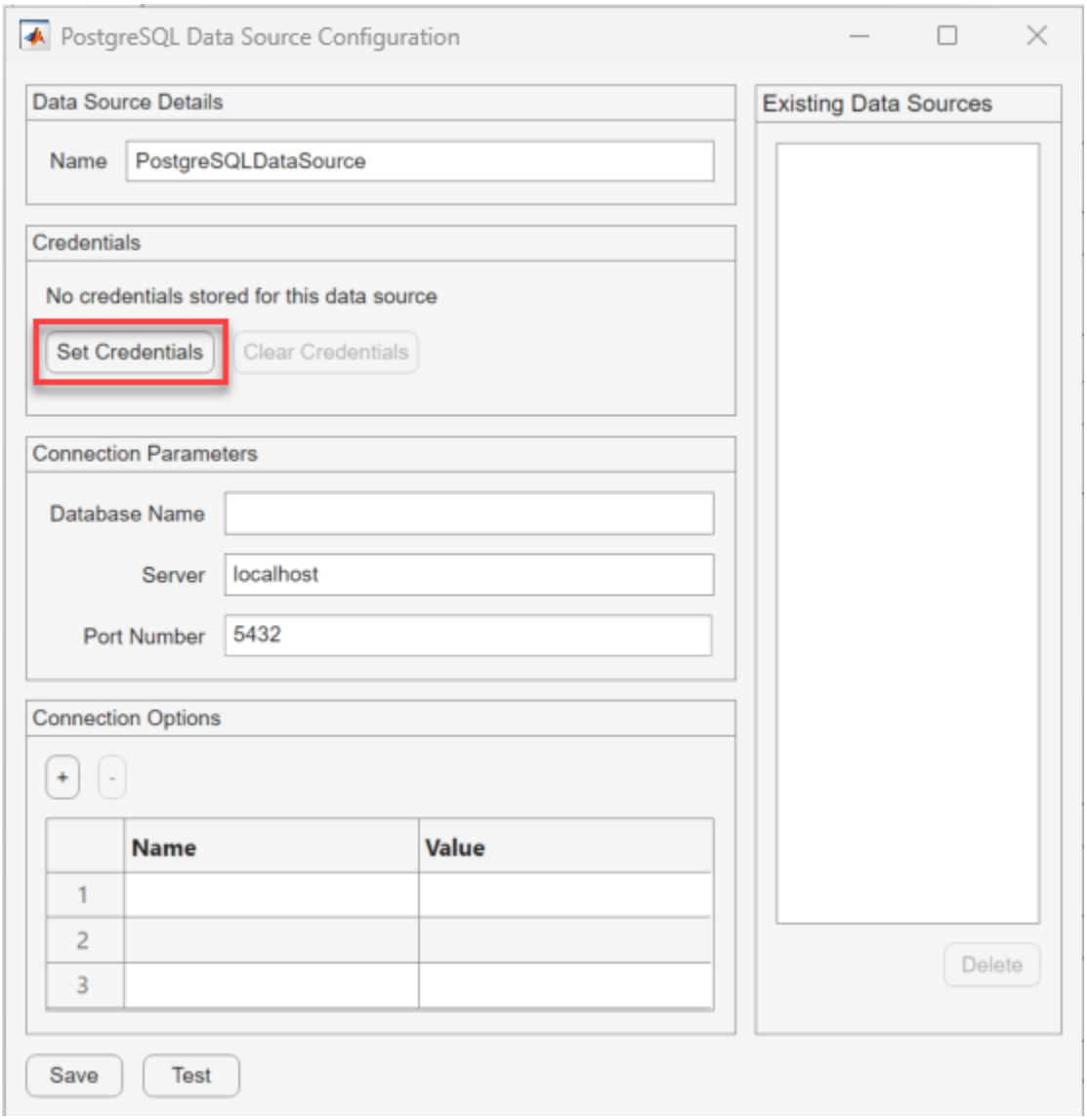
Version: 24.2

New Features

Bug Fixes

★Relational Databases: Store user credentials when connecting to data sources

When you connect to data sources in Database Explorer, you can now store your user credentials by clicking the **Set Credentials** button on the configuration dialog box of your data source, instead of entering them each time you connect to the database. For an example, see Configure PostgreSQL Native Interface Data Source.



★Store and retrieve user credentials by using setSecret and getSecret

For databases that require authentication, you can store credentials in your MATLAB vault by using the `setSecret` function instead of including them in your code. Specify the `datasource` argument and retrieve your credentials by using the `getSecret` function. For an example on using `setSecret`

and `getSecret`, see database. For more information on security consideration topics, see Keep Sensitive Information Out of Code.

ODBC: Fetch multiple SQL result sets

You can now use the `fetchmulti` function to import data into the MATLAB workspace from a stored procedure containing multiple database queries.

Relational Databases: Use foreign keys and link tables in ORM class hierarchies

Specify relationships between object relational mapping (ORM) classes in MATLAB by using `Mappable` attributes such as foreign keys and link tables.

Use the new `CascadeChanges` argument to enable inserting or updating objects in foreign keys and link tables when you use the `ormwrite` and `ormupdate` methods. Use the new `Depth` argument to set the number of nested foreign keys and link tables to read before stopping when you use the `ormread` method.

Relational Databases: Read PostgreSQL array types

Use any of the four PostgreSQL native interface import methods to read array data types into MATLAB without any extra data conversions. For details on how MATLAB reads each of the supported PostgreSQL data types, see the table of data type mappings in the Output Arguments section of the following import methods:

- fetch
- sqlread
- sqlinnerjoin
- sqlouterjoin

MongoDB C Driver: Connect to MongoDB using version 1.25.4

Database Toolbox now includes version 1.25.4 of the MongoDB C driver. For information about the supported server, visit https://www.mongodb.com/docs/drivers/c/.

R2024a

Version: 24.1

New Features

Bug Fixes

★Secure and parallelize databaseDatastore objects

You can now read large amounts of data in a database by securely parallelizing a `databaseDatastore` object when you work in a parallel environment.

★Support Package: Connect to MariaDB database using ODBC driver for Database Toolbox

The MariaDB® ODBC driver for Database Toolbox is now offered as a support package that includes version 3.1 of this driver for the Windows platform. For installation instructions, see MariaDB ODBC Driver for Database Toolbox Support Package Installation.

★Support Package: Connect to PostgreSQL database using ODBC driver for Database Toolbox

The PostgreSQL ODBC driver for Database Toolbox is now offered as a support package that includes version 15.0 of this driver for the Windows platform. For installation instructions, see PostgreSQL ODBC Driver for Database Toolbox Support Package Installation.

Support Package: Connect to Databricks database using ODBC driver for Database Toolbox

The Databricks® ODBC driver for Database Toolbox is now offered as a support package that includes version 2.7.3 of this driver for the Windows and macOS platforms. For installation instructions, see Databricks ODBC Driver for Database Toolbox Support Package Installation.

Configure ODBC data sources at the command line

You can create, modify, save, test, and delete ODBC data sources directly by using Database Toolbox instead of performing these operations outside of MATLAB. For more information about the enhanced ODBC configuration workflow for both the Windows and macOS platforms, see databaseConnectionOptions.

Relational Databases: Insert ORM objects with autoincrementing primary keys

Database Toolbox now supports autoincrementing primary keys for applicable databases and relaxes the need for specifying the values of the `PrimaryKey` property to insert an object on a database. You can immediately insert objects, perform other object relational mapping (ORM) operations, and streamline your workflows when interacting with these databases:

- PostgreSQL
- MySQL
- MariaDB
- SQLite
- Microsoft® SQL Server®

- Microsoft Access®
- Oracle®

MariaDB C Driver: Connect to MariaDB database using version 3.3.5

Database Toolbox now includes version 3.3.5 of the MariaDB C driver. To connect to a MariaDB database, see connection. For information about the supported server, see C & C++ Connectors on the MariaDB website.

R2023b

Version: 23.2

New Features

Bug Fixes

Relational Databases: Interact with databases using ORM

Object relational mapping (ORM) is a software layer that provides the capability to interact with relational databases using MATLAB objects. You can read, write, and update the details of objects that are stored as rows in a database table. Instantiate an object and map it to a database table using the `database.orm.mixin.Mappable` class. Use the following methods to read, write, and update the database:

- `ormwrite`
- `ormread`
- `ormupdate`

Use the `orm2sql` function to produce the SQL CREATE statement that corresponds to a given class definition. You can use this function to check whether the attributes in the class definition have the intended effect.

For more information, see Read and Write Objects to Relational Database Using ORM Workflow.

ODBC: Make database connections on macOS

ODBC support for macOS provides users the capability to connect to database servers on a Mac using the shipped driver from MathWorks® or a downloaded driver. Database vendors include: MySQL/MariaDB, PostgreSQL, and Microsoft SQL Server.

You can create an ODBC database connection on Windows, Linux®, and Mac platforms using the `odbc` function with a defined data source or a connection string.

For more information on setting up a connection to a relational database on the Apple macOS platform, see:

- Microsoft SQL Server ODBC for macOS
- Microsoft SQL Server ODBC for macOS DSN-Less Connection
- MySQL ODBC for macOS
- MySQL ODBC for macOS DSN-Less Connection
- PostgreSQL ODBC for macOS
- PostgreSQL ODBC for macOS DSN-Less Connection

Relational Databases: Filter rows in databaseDatastore

The `RowFilter` property for a `databaseDatastore` object provides database filtering directly from MATLAB. You can set filtering conditions either as a property after creating a `databaseDatastore` object or from a `databaseImportOptions` object.

MongoDB C Driver: Connect to MongoDB using version 1.23.0

Database Toolbox now includes version 1.23.0 of the MongoDB C driver. For information about the supported server, visit https://www.mongodb.com/docs/drivers/c/.

PostgreSQL ODBC Driver Support: Connect to PostgreSQL servers

Database Toolbox now provides a PostgreSQL ODBC driver for connecting to PostgreSQL servers.

- To connect to a server from an Intel®-based Mac, use the driver *`matlabroot`*`/bin/maci64/psqlodbcw.so`.
- To connect to a server from an Apple silicon Mac, use the driver *`matlabroot`*`/bin/maca64/psqlodbcw.so`.
- To connect to a server from a Linux machine, use the driver *`matlabroot`*`/bin/glnxa64/psqlodbcw.so`.

Tall Arrays: Projection and predicate pushdown for indexing tall arrays backed by databaseDatastore

Tall arrays backed by `databaseDatastore` now use projection and predicate pushdown for indexing operations, which reduces the amount of data that is imported into MATLAB by filtering the variables and rows selected in the indexing operation.

R2023a

Version: 11.0

New Features

Bug Fixes

Row Filter Support: Selectively import rows of data

You can specify row filter conditions to selectively import rows of data from a database table. Use the `RowFilter` name-value argument with the `sqlread`, `fetch`, `sqlinnerjoin`, and `sqlouterjoin` functions when working with ODBC-compliant and JDBC-compliant relational databases or native interfaces.

Use the `RowFilter` property of `SQLImportOptions` when defining import options for database data.

SQL Update: Update existing rows in database tables

The `sqlupdate` function enables you to update rows in database tables with the rows from the MATLAB table based on filter conditions. Update tables in ODBC-compliant and JDBC-compliant relational databases and native interfaces by using:

- `sqlupdate` for ODBC-compliant and JDBC-compliant databases
- `sqlupdate` for MySQL databases
- `sqlupdate` for PostgreSQL databases
- `sqlupdate` for SQLite databases

SQLite Connection: Customize import options

Create the `SQLImportOptions` object for an SQLite database connection using the `databaseImportOptions` function. Use this object to customize options for importing data from a database into MATLAB.

Large Data Import: Customize import with additional DatabaseDatastore properties

When importing large data programmatically, you can use these new properties of the `databaseDatastore` object for more control:

- `SelectedVariableNames` property enables you to specify a subset of selected variables to import.
- `VariableNamingRule` property lets you preserve non-ASCII characters in column names of imported data.

Also, you can now edit column names specified in the `VariableNames` property of the `databaseDatastore` object.

MySQL C++ Interface: Use MariaDB driver for simpler data source setup and connection to MySQL database

Database Toolbox now provides the MariaDB C Connector driver that streamlines the process for configuring a MySQL native interface data source for a MySQL database.

MariaDB ODBC Driver Support: Connect to MySQL and MariaDB servers

Database Toolbox now provides the MariaDB ODBC driver for connecting to MySQL servers and MariaDB servers. To connect to a server on Linux, use a DNS-less connection with the driver file *matlabroot*`/bin/glnxa64/libmaodbc.so`.

R2022b

Database Explorer App supports MATLAB interface to SQLite

The Database Explorer app supports the MATLAB interface to SQLite to create an SQLite connection. After you create the connection, you can import data from an SQLite database file into MATLAB.

Functionality being removed or changed

Database Toolbox Interface for Apache Cassandra Database has been removed

The Database Toolbox Interface for Apache Cassandra® Database is not available for use with Database Toolbox starting with the version R2022b. Use the Columnar Database instead.

These functions have been removed and replaced by the functions of the Apache® Cassandra® Database C++ interface.

Database Toolbox Interface for Apache Cassandra Database Function	Apache Cassandra Database C++ Interface Replacement Function
`cassandra`	`apacheCassandra`
`close`	`close`
`isopen`	`isopen`
`columninfo`	`columninfo`
`partitionRead`	`partitionRead`
`tablenames`	`tablenames`
`upsert`	`upsert`
`executecql`	`executecql`

Database Toolbox interface for MongoDB has been removed

The Database Toolbox interface for MongoDB is not available for use with Database Toolbox starting with the version R2022b. Use the Document Database instead.

These functions have be removed and replaced by the functions of the MongoDB C++ interface.

Database Toolbox Interface for MongoDB Function	MongoDB C++ Interface Replacement Function
`mongo`	`mongoc`
`isopen`	`isopen`
`close`	`close`
`count`	`count`
`distinct`	No replacement
`find`	`find`
`createCollection`	`createCollection`
`dropCollection`	`dropCollection`

Database Toolbox Interface for MongoDB Function	MongoDB C++ Interface Replacement Function
`insert`	`insert`
`remove`	`remove`
`update`	`update`

R2022a

MATLAB interface to SQLite functions for data interaction and database management

The MATLAB interface to SQLite has functions for importing data, exporting data, and database management. For details, see MATLAB Interface to SQLite. For added functionality, see this table.

Function	Purpose
`isopen`	Determine if SQLite connection is open.
`sqlread`	Import data into MATLAB from an SQLite database table.
`sqlwrite`	Insert MATLAB data into an SQLite database table.
`commit`	Make changes to an SQLite database file permanent.
`execute`	Execute an SQL statement using an SQLite database connection.
`rollback`	Undo changes made to an SQLite database file.

Database Toolbox supports MATLAB Online for SQLite databases

Database Toolbox supports MATLAB Online™ when you connect to an SQLite database. For details about interacting with SQLite databases, see MATLAB Interface to SQLite.

⚠Functionality being removed or changed

exec function will be removed
Still runs

The `exec` function will be removed in a future release. Use the `execute` function to manage the SQLite database.

insert function will be removed
Still runs

The `insert` function will be removed in a future release. Use the `sqlwrite` function to import data. Some differences between the workflows might require updates to your code.

Update Code

Use the `sqlwrite` function with the `sqlite` object to export data from a SQLite database.

In prior releases, you exported data using the `insert` function. For example:

```
tablename = 'inventoryTable';
colnames = {'productNumber','Quantity','Price','inventoryDate'};
insert(conn,tablename,colnames, ...
    {20,150,50.00,'11/3/2015 2:24:33 AM'})
```

Now you can export data using the `sqlwrite` function.

```
tablename = "productTable";
data = table(30,500000,1000,25,"Rubik's Cube", ...
```

```
    'VariableNames',["productNumber" "stockNumber" ...
    "supplierNumber" "unitCost" "productDescription"]);
sqlwrite(conn,tablename,data)
```

fetch function returns table
Behavior change

In prior releases, the `fetch` function, which imports data from an SQLite database, returned the `results` output argument as a cell array. In R2022a, the `fetch` function returns the `results` output argument as a table. Use the `table2cell` function to convert the data type back to a cell array, or adjust your code to accept the new data type.

runsqlscript function will be removed
Warns

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 argument.

insert function will be removed

Warns

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.

```
tablename = "yearlySales";
data = table("March",50,2000, ...
    'VariableNames',["Month" "salesTotal" "Revenue"]);
sqlwrite(conn,tablename,data)
```

Database Toolbox Interface for Apache Cassandra Database will be removed

Warns

The Database Toolbox Interface for Apache Cassandra Database will be removed in a future release. Use the Apache Cassandra Database C++ interface instead.

These functions will be removed and replaced by the functions of the Apache Cassandra Database C++ interface.

Database Toolbox Interface for Apache Cassandra Database Function	Apache Cassandra Database C++ Interface Replacement Function
`cassandra`	`apacheCassandra`
`close`	`close`
`isopen`	`isopen`
`columninfo`	`columninfo`
`partitionRead`	`partitionRead`
`tablenames`	`tablenames`
`upsert`	`upsert`
`executecql`	`executecql`

Database Toolbox interface for MongoDB will be removed

Warns

The Database Toolbox interface for MongoDB will be removed in a future release. Use the MongoDB C++ interface instead.

These functions will be removed and replaced by the functions of the MongoDB C++ interface.

Database Toolbox Interface for MongoDB Function	MongoDB C++ Interface Replacement Function
`mongo`	`mongoc`
`isopen`	`isopen`
`close`	`close`
`count`	`count`
`distinct`	No replacement
`find`	`find`
`createCollection`	`createCollection`
`dropCollection`	`dropCollection`
`insert`	`insert`
`remove`	`remove`
`update`	`update`

attr function has been removed
Errors

The `attr` function has been removed. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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 has been removed
Errors

The `cols` function has been removed. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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`. Alternatively, view the dimensions of the returned data in the Workspace browser.

columnnames function has been removed
Errors

The `columnnames` function has been removed. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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`.

fetchmulti function has been removed
Errors

The `fetchmulti` function has been removed. There is no replacement for the `fetchmulti` function.

querytimeout function has been removed
Errors

The `querytimeout` function has been removed. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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 has been removed
Errors

The `rows` function has been removed. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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`. Alternatively, view the dimensions of the returned data in the Workspace browser.

set function has been removed
Errors

The `set` function with the `cursor` object has been removed. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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);
```

width function has been removed

Errors

The `width` function has been removed. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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.

R2021b

MongoDB C++ interface

With the MongoDB C++ interface, 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, see MongoDB C++ Interface.

Database Toolbox includes the MongoDB C++ interface and does not require the installation of an add-on.

Functionality being removed or changed

Database Toolbox Interface for Apache Cassandra Database will be removed
Still runs

The Database Toolbox Interface for Apache Cassandra Database will be removed in a future release. Use the Apache Cassandra Database C++ interface instead.

These functions will be removed and replaced by the functions of the Apache Cassandra Database C++ interface.

Database Toolbox Interface for Apache Cassandra Database Function	Apache Cassandra Database C++ Interface Replacement Function
`cassandra`	`apacheCassandra`
`close`	`close`
`isopen`	`isopen`
`columninfo`	`columninfo`
`partitionRead`	`partitionRead`
`tablenames`	`tablenames`
`upsert`	`upsert`
`executecql`	`executecql`

Database Toolbox interface for MongoDB will be removed
Still runs

The Database Toolbox interface for MongoDB will be removed in a future release. Use the MongoDB C++ interface instead.

These functions will be removed and replaced by the functions of the MongoDB C++ interface.

Database Toolbox Interface for MongoDB Function	MongoDB C++ Interface Replacement Function
`mongo`	`mongoc`
`isopen`	`isopen`
`close`	`close`
`count`	`count`
`distinct`	No replacement
`find`	`find`

Database Toolbox Interface for MongoDB Function	MongoDB C++ Interface Replacement Function
`createCollection`	`createCollection`
`dropCollection`	`dropCollection`
`insert`	`insert`
`remove`	`remove`
`update`	`update`

R2021a

ODBC connection for Linux and DSN-less connection

The `odbc` function creates an ODBC database connection on the Windows and Linux platforms. The `odbc` function creates an ODBC database connection by using a defined data source or a connection string. When you specify a connection string, you create a DSN-less database connection. (DSN is a data source name.)

Apache Cassandra Database C++ Interface

With the Apache Cassandra database C++ interface, you can explore the Cassandra database. You can access the database keyspaces, tables, and columns, and then 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 Apache Cassandra Database C++ Interface.

Database Toolbox includes the Apache Cassandra database C++ interface and does not require the installation of an add-on.

⚠Functionality being removed or changed

attr function will be removed
Warns

The `attr` function will be removed in a future release. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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 will be removed
Warns

The `cols` function will be removed in a future release. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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 will be removed
Warns

The `columnnames` function will be removed in a future release. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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 will be removed
Warns

The `get` function with the `cursor` object will be removed in a future release. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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);
```

set function will be removed
Warns

The `set` function with the `cursor` object will be removed in a future release. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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 will be removed
Warns

The `querytimeout` function will be removed in a future release. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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 will be removed

Warns

The `rows` function will be removed in a future release. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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 will be removed

Warns

The `width` function will be removed in a future release. Use the `fetch` function to import data. Some differences between the workflows might require updates to your code.

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.

fetchmulti function will be removed

Warns

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