Title

stata.com

jdbc — Load, write, or view data from a database with a Java API

Description

`jdbc` allows you to load data from a database, execute SQL statements on a database, and insert data into a database using Java Database Connectivity (JDBC). JDBC is an application programming interface (API) for the programming language Java and defines how a client (Stata) can access a database. `jdbc` is oriented toward relational databases or nonrelational database-management systems that have rectangular data. NoSQL databases will not work with `jdbc`.

`jdbc connect` stores all database connection settings for subsequent `jdbc` commands.

`jdbc add` stores database connection settings as a data source name for a Stata session.

`jdbc remove` removes a stored data source name for a Stata session.

`jdbc list` displays all stored data source names for a Stata session.

`jdbc showdbs` produces a list of all databases for a given URL.

`jdbc showtables` retrieves a list of table names available from a specified database.

`jdbc describe` lists column names and data types associated with a specified table.

`jdbc load` reads a database table into Stata's memory. You can load a table specified in the `table()` option or load an ODBC table generated by an SQL SELECT statement specified in the `exec()` option.

`jdbc insert` writes data from memory to a database table. The data can be appended to an existing table or replace an existing table.

`jdbc exec` allows for most SQL statements to be issued directly to any database. Statements that produce output, such as SELECT, have their output neatly displayed. By using Stata's ado-language, you can also generate SQL commands on the fly to do positional updates or whatever the situation requires.

Quick start

Store connection settings to database `myDB`

```
jdbc connect, jar("mysql-connector-java-5.1.49.jar")     ///
     driverclass("com.mysql.jdbc.Driver")                ///
     url("jdbc:mysql://https://www.stata.com/myDB:3306") ///
     user("stata") password("stata")
```

List available table names in database `myDB`

```
jdbc showtables
```

Describe the column names and data types in table `MyTable` from `myDB`

```
jdbc describe "MyTable"
```

Load MyTable into memory from myDB

```
jdbc load, table("MyTable")
```

Syntax

Store JDBC connection settings for all jdbc commands

```
jdbc connect { DataSourceName | , connect_options }
```

Add JDBC connection settings as a data source name for the current Stata session

```
jdbc add DataSourceName, connect_options
```

Remove JDBC connection settings and data source name for the current Stata session

```
jdbc remove { DataSourceName | _all }
```

List stored data source names and URLs for the current Stata session

```
jdbc list
```

List all databases for a given connection

```
jdbc showdbs
```

Retrieve available table names from specified data source

```
jdbc showtables [ "SearchString" ]
```

List column names and data types associated with specified table

```
jdbc describe "TableName"
```

Import data from a database

```
jdbc load, { table("TableName") | exec("SqlStmtList") } [ load_options ]
```

Export data to a database

```
jdbc insert [ varlist ] [ if ] [ in ], table("TableName") [ insert_options ]
```

Allow SQL statements to be issued directly to a database

```
jdbc exec "SqlStmtList"
```

DataSourceName is a name used to store connection settings.
SearchString is a database table name search string; SQL wildcard characters like % and _ are allowed.
TableName is the name of a table in the database.
SqlStmtList may be one valid SQL statement or a list of SQL statements separated by semicolons.

connect_options	Description
* `jar("`*JarFileName*`")`	JAR file name of JDBC driver
* `jarpath("`*DirectoryName*`")`	directory where the driver JAR file is stored along with driver dependencies
* `driverclass("`*ClassName*`")`	Java class name for JDBC driver
* `url("`*URL*`")`	database URL
* `user("`*UserID*`")`	user ID of user establishing connection
* `password("`*Password*`")`	password of user establishing connection
`connprop("`*ConnectionProperty*`")`	driver-specific connection property

*Either `jar("`*JarFileName*`")` or `jarpath("`*DirectoryName*`")` and `driverclass("`*ClassName*`")`, `url("`*URL*`")`, `user("`*UserID*`")`, and `password("`*Password*`")` are required with `jdbc add`. These options are also required with `jdbc connect` when *DataSourceName* is not specified.

load_options	Description
* `table("`*TableName*`")`	name of table stored in the database
* `exec("`*SqlStmtList*`")`	SQL SELECT statements to generate a table to be read into Stata
`rows(#)`	fetch # result set rows from database; default is `rows(10)`
`clear`	replace data in memory
`case(`<u>l</u>`ower` | <u>u</u>`pper` | <u>pre</u>`serve)`	import variable names as lowercase or uppercase; the default is to preserve the case

*Either `table("`*TableName*`")` or `exec("`*SqlStmtList*`")` must be specified.

insert_options	Description
* `table("`*TableName*`")`	name of table stored in the database
`rows(#)`	build memory result set with # of rows; default is `rows(1)`
`overwrite`	clear data in table before data in memory are written to the table

*`table("`*TableName*`")` is required.

JarFileName is the name of the JDBC driver JAR file.
ClassName is the Java class name stored in the JDBC driver JAR file.
URL is the database URL.
UserID is the user ID.
Password is the user's password.

Options

Options are presented under the following headings:

Options for jdbc connect and jdbc add

`jar("`*JarFileName*`")` specifies the JDBC driver JAR file installed along your ado-path. Either `jar()` or `jarpath()` is required with `jdbc add`. Also, if *DataSourceName* is not specified, either `jar()` or `jarpath()` is required with `jdbc connect` for `jdbc showdbs`, `jdbc showtables`, `jdbc describe`, `jdbc load`, `jdbc insert`, and `jdbc exec` to work. `jar()` may not be combined with `jarpath()`.

`jarpath("`*DirectoryName*`")` specifies the directory where the JDBC driver JAR files are installed along your ado-path. Either `jarpath()` or `jar()` is required with `jdbc add`. Also, if *DataSourceName* is not specified, either `jarpath()` or `jar()` is required with `jdbc connect` for `jdbc showdbs`, `jdbc showtables`, `jdbc describe`, `jdbc load`, `jdbc insert`, and `jdbc exec` to work. `jarpath()` may not be combined with `jar()`.

`driverclass("`*ClassName*`")` specifies the Java class name stored in the JDBC driver JAR file installed along your ado-path. `driverclass()` is required with `jdbc add`. Also, if *DataSourceName* is not specified, `driverclass()` is required with `jdbc connect` for `jdbc showdbs`, `jdbc showtables`, `jdbc describe`, `jdbc load`, `jdbc insert`, and `jdbc exec` to work.

`url("`*URL*`")` specifies the URL to the database the user is attempting to establish the connection to. `url()` is required with `jdbc add`. Also, if *DataSourceName* is not specified, `url()` is required with `jdbc connect` for `jdbc showdbs`, `jdbc showtables`, `jdbc describe`, `jdbc load`, `jdbc insert`, and `jdbc exec` to work. The driver *URL* syntax is as follows:

```
jdbc:Database_type://Host:Port/Database_name?connection_properties
```

`user("`*UserID*`")` specifies the user ID of the user attempting to establish the connection to a database. `user()` is required with `jdbc add`. Also, if *DataSourceName* is not specified, `user()` is required with `jdbc connect` for `jdbc showdbs`, `jdbc showtables`, `jdbc describe`, `jdbc load`, `jdbc insert`, and `jdbc exec` to work.

`password("`*Password*`")` specifies the password of the user attempting to establish the connection to a database. `password()` is required with `jdbc add`. Also, if *DataSourceName* is not specified, `password()` is required with `jdbc connect` for `jdbc showdbs`, `jdbc showtables`, `jdbc describe`, `jdbc load`, `jdbc insert`, and `jdbc exec` to work.

`connprop("`*ConnectionProperty*`")` specifies the driver-specific connection properties. A connection property is a key value pair that is separated by a colon and delimited by a semicolon. For example,

```
jdbc connect, ... connprop("characterEncoding:ISO8859-1;")
```

These properties can also be set in the `url()` option.

Options for jdbc load

`table("`*TableName*`")` specifies the name of the table stored in a specified database. Either the `table()` option or the `exec()` option—but not both—is required with the `jdbc load` command.

`exec("`*SqlStmtList*`")` allows you to issue an SQL SELECT statement to generate a table to be read into Stata. Multiple SQL statements can be issued, with the last SQL statement being a SELECT. Each statement should be delimited by a semicolon. For example,

```
local sql                                                              ///
    "CREATE TEMPORARY TABLE t(a INT, b INT); INSERT INTO t VALUES (1,2); ///
    SELECT * FROM t;"
jdbc load, exec("`sql'")
```

An error message is returned if the SQL statements are invalid SQL. Either the `table()` option or the `exec()` option—but not both—is required with the `jdbc load` command.

`rows(#)` specifies the number of rows to be fetched from the database result set for each network call. This option may help improve command performance. The default is `rows(10)`. Some drivers do not support this feature. Note that setting `rows()` to a large number might require you to change the amount of heap memory allocated for the JVM with the `java set heapmax` command.

`clear` permits the data to be loaded, even if there are data already in memory, and even if that data have changed since the data were last saved.

`case(lower | upper | preserve)` specifies the case of the variable names after loading. The default is `case(preserve)`.

Options for jdbc insert

`table("`*TableName*`")` specifies the name of the table stored in a specified database.

`rows(#)` specifies the number of result set rows to be sent to the database for each network call. This option may help improve command performance. The default result set size is 1. This option does not work with datasets that contain `strL`s. Some drivers do not support this feature. Note that setting the `rows(#)` to a large number might require you to change the amount of heap memory allocated for the JVM with the `java set heapmax` command.

`overwrite` allows data to be dropped from a database table before the Stata data in memory are written to the table. All data from the table are erased, not just the data from the variable columns that will be replaced.

Remarks and examples

stata.com

`jdbc` allows you to connect to, load data from, insert data into, and execute queries on a database using JDBC. First, you specify the connection settings with `jdbc connect`, including the URL for the database you are connecting to and your user ID and password. Thereafter, you can use `jdbc showdbs`, `jdbc showtables`, `jdbc describe`, `jdbc load`, `jdbc insert`, and `jdbc exec`. These commands allow you to execute statements on a database and load data to and from Stata; they will use the connection information you specified with `jdbc connect` to open a connection and perform the specified task.

If you will be connecting to multiple databases frequently, you can store the connection settings for each database under a data source name with `jdbc add`. Then, whenever you wish to connect to a database, simply use `jdbc connect`, and specify the data source name. This avoids having to specify all the connection information every time you wish to connect to a different database.

Remarks are presented under the following headings:

JDBC drivers

To use jdbc, you must first download and install your database vendor JDBC driver JAR file. To see information on Stata's current JDBC implementation, click here.

Once you have downloaded the appropriate driver, you must install the driver along Stata's ado-path. If the file is compressed, you can use Stata's unzipfile with the downloaded file to extract the .jar file. Once extracted, place the .jar file along your ado-path so Stata can add it to the Java virtual machine (JVM) class-path. You can use java query to check to see whether Stata has loaded your driver along the JVM class-path.

Most users should place the .jar files in the PERSONAL directory or the current working directory. System administrators may wish to place them in the SITE directory if they have a network installation and want to make them available to all users.

Connecting to a database

jdbc connect stores all database connection settings for commands jdbc showdbs, jdbc showtables, jdbc describe, jdbc load, jdbc insert, and jdbc exec. Options jar(), driverclass(), url(), user(), and password() are required, unless you have already saved that information under a data source name and you are specifying that *DataSourceName* with jdbc connect.

If you try to use these commands before setting your connection properties, you will receive the following error message:

```
. jdbc showtables
Connection failed
JDBC driver class not found
r(681);
```

❑ Technical note

Storing your database name, user ID, and password in a Stata do-file, ado-file, or log file can be a security risk. Your database vendor might have software called a wallet that can store this information securely on your machine.

❑

▷ Example 1: Creating a connection

Below, we create a connection string for the JDBC driver in Stata:

```
. jdbc connect, jar("mysql-connector-java-8.0.22.jar")
> driverclass("com.mysql.cj.jdbc.Driver")
> url("jdbc:mysql://localhost:3306/myDB")
> user("stata") password("stata_pass")
```

Going forward, when we issue the jdbc showdbs, jdbc showtables, jdbc describe, jdbc load, jdbc insert, or jdbc exec command, each will use this information to connect to the database myDB.

◁

▷ Example 2: Using macros

You can also use macros to make your do-file more readable and easier to change database settings.

```
. local jar "mysql-connector-java-8.0.22.jar"
. local driverclass "com.mysql.cj.jdbc.Driver"
. local url "jdbc:mysql://localhost:3306/myDB"
. local user "stata"
. local pass "stata_pass"
. jdbc connect, jar("`jar'") driverclass("`driverclass'")
> url("`url'") user("`user'") password("`pass'")
```

◁

Data source names

If you would like to have database connection settings stored and ready for `jdbc` to use every time you start a Stata session, you can place `jdbc add` in your `profile.do` to store these settings; see **[GSW] B.3 Executing commands every time Stata is started**, **[GSM] B.1 Executing commands every time Stata is started**, or **[GSU] B.1 Executing commands every time Stata is started**.

Use `jdbc list` to see the current session's stored connection settings and `jdbc remove` to remove stored settings.

Exploring a database

`jdbc showdbs`, `jdbc showtables`, and `jdbc describe` are used, respectively, to list database names, table names, and table columns of a connection. Use these commands to search for data to load from your connection.

▷ Example 3: Listing table names

`jdbc showtables` is used to list table names available from a specified database. To list all the tables stored in database `myDB`, type

```
. jdbc showtables
Database: myDB
------------------------------------------------------------------------------
Tables
------------------------------------------------------------------------------
auto
------------------------------------------------------------------------------
```

◁

▷ Example 4: Listing column names and data types

`jdbc describe` displays the column names and JDBC data types of the table listed. To describe the `auto` table, type

```
. jdbc describe "auto"
Table: auto
```

column name	column type
make	VARCHAR
price	INT
mpg	INT
rep78	SMALLINT
headroom	FLOAT
trunk	SMALLINT
weight	SMALLINT
length	SMALLINT
turn	SMALLINT
displacement	SMALLINT
gear_ratio	FLOAT
domestic	VARCHAR

◁

Loading data from a database

`jdbc load` is used to load a database table into Stata's memory; this can be an existing table or a subset of a table created by a series of SQL statements.

▷ Example 5: Loading a table

To load a database table listed in the `jdbc showtables` output, specify the table name in the `table()` option.

```
. jdbc load, table("auto")
74 observations loaded
. describe
Contains data
 Observations:            74
    Variables:            12
```

Variable name	Storage type	Display format	Value label	Variable label
make	str19	%19s		make
price	long	%12.0g		price
mpg	long	%12.0g		mpg
rep78	int	%8.0g		rep78
headroom	float	%9.0g		headroom
trunk	int	%8.0g		trunk
weight	int	%8.0g		weight
length	int	%8.0g		length
turn	int	%8.0g		turn
displacement	int	%8.0g		displacement
gear_ratio	float	%9.0g		gear_ratio
domestic	str18	%18s		domestic

```
Sorted by:
     Note: Dataset has changed since last saved.
```

◁

Example 6: Loading part of a table

If your database table is large and the memory on your computer is limited, it is a good idea to limit the amount of data loaded from the database using a SELECT statement in the `exec()` option. For example, instead of loading the whole table as we did above, we can just load the `mpg` column:

```
. jdbc load, exec("SELECT mpg FROM auto;")
74 observations loaded

. describe

Contains data
 Observations:            74
    Variables:             1
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
mpg             long    %12.0g                mpg
-------------------------------------------------------------------------------
Sorted by:
     Note: Dataset has changed since last saved.
```

❑ Technical note

When Stata loads a table, data are converted from JDBC data types to Stata data types. Stata does not support all JDBC data types. If the column cannot be read because of incompatible data types, Stata will issue a note and skip a column. The following table lists the supported JDBC data types and their corresponding Stata data types:

JDBC data type	Stata data type
BOOLEAN	byte
BIT	byte
TINYINT	byte
SMALLINT	int
INTEGER	long
ROWID	str
BIGINT	str
REAL	float
FLOAT	float
NUMERIC	double
DECIMAL	double
DOUBLE	double
DATE	double
TIME	double
TIMESTAMP	double
TIME_WITH_TIMEZONE	str
TIMESTAMP_WITH_TIMEZONE	str
BINARY	strL
VARBINARY	strL
LONGVARBINARY	strL
BLOB	strL
CHAR	str/strL
VARCHAR	str/strL
LONGVARCHAR	str/strL
NCHAR	str/strL
NVARCHAR	str/strL
LONGNVARCHAR	str/strL
NCLOB	str/strL
CLOB	str/strL
STRUCT	skipped
ARRAY	skipped
SQLXML	skipped
NULL	skipped
OTHER	skipped
REF_CURSOR	skipped
JAVA_OBJECT	skipped
DISTINCT	skipped
REF	skipped
DATALINK	skipped

Stata is a UTF-8 application, so all string data should be encoded as UTF-8. This can be set using a driver connection property. Check your database vendor or driver documentation to see how your string data is encoded by default to see whether this property should be set.

```
. jdbc connect, ... connprop("characterEncoding:UTF8;")
```

❑

Inserting data into a database

`jdbc insert` inserts data in memory into a database table. The database table and the Stata *varlist* must have the same column and variable names, number of columns, and compatible data types for the insert to work correctly. By default, observations are appended to the database table. When you insert data, mapping of the data types are the same as `jdbc load`, with one exception, Stata `bytes`. Stata `bytes` are mapped to SMALLINTs because some database vendors' (SQLServer) BYTE data type is unsigned.

▷ Example 7: Inserting data into a table

Below, we insert the data in memory into the table `auto`.

```
. jdbc insert, table(auto)
74 rows inserted
```

To replace the table with the data in memory, use the option `overwrite`.

```
. jdbc insert, table(auto) overwrite
74 rows affected
74 rows inserted
```

◁

Executing SQL on a database

You use `jdbc exec` to execute SQL commands on the database. If an SQL command returns a result set, like SELECT, that result set will be displayed in the Stata Results window.

▷ Example 8: Executing SQL commands

To use `jdbc insert`, you must have a table already created in your database. If you do not, you can use `jdbc exec` to create a table in your database. For example, one might create a table in a MySQL database with the SQL command below:

```
#delimit ;
local create_table_sql `"CREATE TABLE auto (
        make varchar(19) NOT NULL,
        price int,
        mpg int,
        rep78 smallint,
        headroom float,
        trunk smallint,
        weight smallint,
        length smallint,
        turn smallint,
        displacement smallint,
        gear_ratio float,
        domestic varchar(18)
);"' ;
jdbc exec "`create_table_sql'"
```

If your SQL statement contains double quotes, you must enclose your statement in compound double quotes, as we did with the statement above.

◁

Stored results

`jdbc showdbs` stores the following in `r()`:

Scalars

`r(n_dbs)`	number of databases displayed

`jdbc showtables` stores the following in `r()`:

Scalars

`r(n_tables)`	number of tables displayed

`jdbc describe` stores the following in `r()`:

Scalars

`r(k)`	number of columns displayed

`jdbc load` stores the following in `r()`:

Scalars

`r(k)`	number of variables loaded
`r(N)`	number of observations loaded

`jdbc insert` stores the following in `r()`:

Scalars

`r(k)`	number of columns inserted
`r(N)`	number of rows inserted

References

Crow, K. 2017. Importing WRDS data into Stata. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2017/09/19/importing-wrds-data-into-stata/.

——. 2022. Wharton Research Data Services, Stata 17, and JDBC. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2022/01/27/wharton-research-data-services-stata-17-and-jdbc/.

Also see

[D] **odbc** — Load, write, or view data from ODBC sources

[D] **import** — Overview of importing data into Stata

[D] **export** — Overview of exporting data from Stata