Connecting to a database

Martijn J. Schuemie

2025-01-15

Contents

1 Introduction **1**

2 Obtaining drivers **1**
2.1 The JAR folder 2
2.2 Obtaining drivers for SQL Server, Oracle, PostgreSQL, PDW, Spark, RedShift, Snowflake, BigQuery, and Synapse 2
2.3 Obtaining drivers for Netezza and Impala 2
2.4 Obtaining drivers for SQLite or DuckDb 2

3 Creating a connection **2**

4 Using Windows Authentication for SQL Server **3**

5 Connecting to a SQLite database **3**

6 Connecting with Windows authentication from a non-windows machine **4**

1 Introduction

This vignette describes how you can use the `DatabaseConnector` package to connect to a database.

`DatabaseConnector` supports these database platforms:

- Microsoft SQL Server
- Oracle
- PostgresSql
- Microsoft Parallel Data Warehouse (PDW, a.k.a. Analytics Platform System)
- Amazon Redshift
- Apache Impala
- Google BigQuery
- IBM Netezza
- SQLite
- (DataBricks)Spark
- Snowflake
- Synapse

2 Obtaining drivers

Before `DatabaseConnector` can be used to connect to a database, the drivers for your platform need to be downloaded to a location in the local file system, which we'll refer to as the JAR folder.

2.1 The JAR folder

The JAR folder is just a folder in the local file system where the database drivers are stored. It is highly recommended to use the `DATABASECONNECTOR_JAR_FOLDER` environmental variable to point to this folder, which you can for example set using:

```
Sys.setenv("DATABASECONNECTOR_JAR_FOLDER" = "c:/temp/jdbcDrivers")
```

Even better would be to add this entry to your `.Renviron` file:

```
DATABASECONNECTOR_JAR_FOLDER = 'c:/temp/jdbcDrivers'
```

That way, the environmental variable will be automatically set whenever you start R. A convenient way to edit your `.Renviron` file is by using `usethis`:

```
install.packages("usethis")
usethis::edit_r_environ()
```

If you don't use the `DATABASECONNECTOR_JAR_FOLDER` environmental variable, you will need to provide the `pathToDriver` argument every time you call the `downloadJdbcDrivers`, `connect`, `dbConnect`, or `createConnectionDetails` functions.

2.2 Obtaining drivers for SQL Server, Oracle, PostgreSQL, PDW, Spark, RedShift, Snowflake, BigQuery, and Synapse

For your convenience these JDBC drivers are hosted on the OHDSI GitHub pages, and can be downloaded using the `downloadJdbcDrivers` function. You'll first need to specify the JAR folder as described in the previous section, for example using

```
Sys.setenv("DATABASECONNECTOR_JAR_FOLDER" = "c:/temp/jdbcDrivers")
```

And next download the driver. For example, for PostgreSQL:

```
library(DatabaseConnector)
downloadJdbcDrivers("postgresql")
```

```
## DatabaseConnector JDBC drivers downloaded to 'c:/temp/jdbcDrivers'.
```

Note that if we hadn't specified the `DATABASECONNECTOR_JAR_FOLDER` environmental variable, we would have to specify the `pathToDriver` argument when calling `downloadJdbcDrivers`.

2.3 Obtaining drivers for Netezza and Impala

Because of licensing reasons the drivers for BigQuery, Netezza and Impala are not included but must be obtained by the user. see these instructions on how to download these drivers, which you can also see by typing `?jdbcDrivers`.

2.4 Obtaining drivers for SQLite or DuckDb

For SQLite and DuckDb we actually don't use a JDBC driver. Instead, we use the RSQLite and duckdb packages, respectively, which can be installed using

```
install.packages("RSQLite")
install.packages("duckdb")
```

3 Creating a connection

To connect to a database a number of details need to be specified, such as the database platform, the location of the server, the user name, password, and path to the driver. We can call the `connect()` function and

specify these details directly:

```
conn <- connect(dbms = "postgresql",
                server = "localhost/postgres",
                user = "joe",
                password = "secret")
```

```
## Connecting using PostgreSQL driver
```

See this webpage or type `?connect` for information on which details are required for each platform. Note that we did not need to specify the `pathToDriver` argument because we previously already set the `DATABASECONNECTOR_JAR_FOLDER` environmental variable.

Don't forget to close any connection afterwards:

```
disconnect(conn)
```

Instead of providing the server name, it is also possible to provide the JDBC connection string if this is more convenient:

```
conn <- connect(dbms = "postgresql",
                connectionString = "jdbc:postgresql://localhost:5432/postgres",
                user = "joe",
                password = "secret")
```

```
## Connecting using PostgreSQL driver
```

Sometimes we may want to first specify the connection details, and defer connecting until later. This may be convenient for example when the connection is established inside a function, and the details need to be passed as an argument. We can use the `createConnectionDetails` function for this purpose:

```
details <- createConnectionDetails(dbms = "postgresql",
                                   server = "localhost/postgres",
                                   user = "joe",
                                   password = "secret")
conn <- connect(details)
```

```
## Connecting using PostgreSQL driver
```

4 Using Windows Authentication for SQL Server

In some organizations using Microsoft SQL Server and Windows, it is possible to use Windows Authentication to connect to the server, meaning you won't have to provide a user name and password, since your Windows credentials will be used. This will require downloading the SQL Server authentication DLL file, and placing it somewhere on your system path. If you don't have rights to add files to a place on your system path, you can place it anywhere, and set the `PATH_TO_AUTH_DLL` environmental variable, either using the `Sys.setenv()`, or by adding it to your `.Renviron` file. See this webpage or type `?connect` for details on where to get the DLL (and what specific version).

5 Connecting to a SQLite database

`DatabaseConnector` also supports SQLite through the RSQLite package, mainly for testing and demonstration purposes. Provide the path to the SQLite file as the `server` argument when connecting. If no file exists it will be created:

```
conn <- connect(dbms = "sqlite", server = tempfile())
```

```
## Connecting using SQLite driver
```

```r
# Upload cars dataset as table:
insertTable(connection = conn,
            tableName = "cars",
            data = cars)
```

```
## Inserting data took 0.0121 secs
```

```r
querySql(conn, "SELECT COUNT(*) FROM main.cars;")
```

```
##   COUNT(*)
## 1       50
```

```r
disconnect(conn)
```

6 Connecting with Windows authentication from a non-windows machine

Kerberos is used by Active Directory, you need to have the appropriate packages installed on your client machine. For MacOS, the Kerberos packages are usually already installed. On Linux, you'll have to install `krb5-user`. Most of this setup comes from this site.

1. On the non-windows client machine, create or update `/etc/krb5.conf` so that it points to your AD server. Here's an example of `krb5.conf`:

   ```
   [libdefaults]
   default_realm = DOMAIN.COMPANY.COM

   [realms]
   DOMAIN.COMPANY.COM = {
     kdc = dc-33.domain.company.com
   }
   ```

2. Run `kinit <username>@DOMAIN.COMPANY.COM` to get a ticket granting ticket from the kerberos domain controller (KDC). (NOTE: you want to make sure your KDC is accessible from your machine)

3. Download the latest MSSql JDBC driver.

4. Try to connect to the database with the following code in RStudio:

   ```r
   library(devtools)
   library(DatabaseConnector)
   connectionDetails <- createConnectionDetails(
     dbms="sql server",
     ...
     extraSettings="authenticationScheme=JavaKerberos")
   c <- connect(connectionDetails = connectionDetails)
   ```

   In RStudio, you should see that the variable c has a value and the new connection in the connections tab as well.

Note: If you are getting the below error on Mac:

> Error in rJava::.jcall(jdbcDriver, "Ljava/sql/Connection;", "connect", : com.microsoft.sqlserver.jdbc.SQLServerException: Kerberos Login failed: Integrated authentication failed. ClientConnectionId:13fb0d4e-4822-4de2-a125-8408334cb3ed due to javax.security.auth.login.LoginException (Cannot get any of properties: [user, USER] from con properties not available to garner authentication information from the user)

Instead of kinit you can also try `/System/Library/CoreServices` and find 'Ticket Viewer' Click 'Add Identity', then enter your user name. After you click 'Continue', a Kerberos ticket should have been generated.