Package 'DatabaseConnector'

October 13, 2017

Type Package

Title A Package for Connecting to Various DBMSs

Version 2.0.1

Date 2017-10-13

Author Martijn J. Schuemie and Marc A. Suchard

Maintainer Martijn Schuemie <schuemie@ohdsi.org>

Description
Package for connecting to various DBMSs. Also includes support for fetching data as ffdf objects.

Imports rJava,
bit,
ff,
ffbase (>= 0.12.1),
SqlRender,
methods,
utils

License Apache License

Suggests testthat

RoxygenNote 6.0.1

R topics documented:

connect   *connect*

Description

connect creates a connection to a database server .There are four ways to call this function:

- connect(dbms, user, domain, password, server, port, schema, extraSettings,   oracleDriver =
- connect(connectionDetails)
- connect(dbms, connectionString)
- connect(dbms, connectionString, user, password)

Arguments

connectionDetails

An object of class connectionDetails as created by the [createConnectionDetails](#) function.

dbms   The type of DBMS running on the server. Valid values are

- "mysql" for MySQL
- "oracle" for Oracle
- "postgresql" for PostgreSQL
- "redshift" for Amazon Redshift
- "sql server" for Microsoft SQL Server
- "pdw" for Microsoft Parallel Data Warehouse (PDW)
- "netezza" for IBM Netezza
- "bigquery" for Google BigQuery

user   The user name used to access the server.

domain   For SQL Server only: the Windows domain (optional).

password   The password for that user.

server   The name of the server.

port   (optional) The port on the server to connect to.

schema   (optional) The name of the schema to connect to.

extraSettings   (optional) Additional configuration settings specific to the database provider to configure things as security for SSL. These must follow the format for the JDBC connection for the RDBMS specified in dbms.

oracleDriver   Specify which Oracle drive you want to use. Choose between ″thin″ or ″oci″.

connectionString

The JDBC connection string. If specified, the server, port, extraSettings, and oracleDriver fields are ignored. If user and password are not specified, they are assumed to already be included in the connection string.

pathToDriver   Path to the JDBC driver JAR files. Currently only needed for Impala and Netezza.

Details

This function creates a connection to a database.

Value

An object that extends `DBIConnection` in a database-specific manner. This object is used to direct commands to the database engine.

DBMS parameter details

Depending on the DBMS, the function arguments have slightly different interpretations: MySQL:

- `user`. The user name used to access the server
- `password`. The password for that user
- `server`. The host name of the server
- `port`. Specifies the port on the server (default = 3306)
- `schema`. The database containing the tables
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "SSL Mode=Required")

Oracle:

- `user`. The user name used to access the server
- `password`. The password for that user
- `server`. This field contains the SID, or host and servicename, SID, or TNSName: '<sid>', '<host>/<sid>', '<host>/<service name>', or '<tnsname>'
- `port`. Specifies the port on the server (default = 1521)
- `schema`. This field contains the schema (i.e. 'user' in Oracle terms) containing the tables
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "(PROTOCOL=tcps)")
- `oracleDriver` The driver to be used. Choose between "thin" or "oci".

Microsoft SQL Server:

- `user`. The user used to log in to the server. If the user is not specified, Windows Integrated Security will be used, which requires the SQL Server JDBC drivers to be installed (see details below).
- `domain`. Optionally, the domain can be specified here.(See note below).
- `password`. The password used to log on to the server
- `server`. This field contains the host name of the server
- `port`. Not used for SQL Server
- `schema`. The database containing the tables. If both database and schema are specified (e.g. 'my_database.dbo', then only the database part is used, the schema is ignored.
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "encrypt=true; trustServerCertificate=false;")

Microsoft PDW:

- `user`. The user used to log in to the server. If the user is not specified, Windows Integrated Security will be used, which requires the SQL Server JDBC drivers to be installed (see details below).
- `password`. The password used to log on to the server
- `server`. This field contains the host name of the server

- `port`. Not used for SQL Server
- `schema`. The database containing the tables
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "encrypt=true; trustServerCertificate=false;")

Connections where the domain need to be specified are not supported. PostgreSQL:

- `user`. The user used to log in to the server
- `password`. The password used to log on to the server
- `server`. This field contains the host name of the server and the database holding the relevant schemas: <host>/<database>
- `port`. Specifies the port on the server (default = 5432)
- `schema`. The schema containing the tables.
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "ssl=true")

Redshift:

- `user`. The user used to log in to the server
- `password`. The password used to log on to the server
- `server`. This field contains the host name of the server and the database holding the relevant schemas: <host>/<database>
- `port`. Specifies the port on the server (default = 5439)
- `schema`. The schema containing the tables.
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "ssl=true&sslfactory=com.amazon.redshift.ssl.NonValidatingFactory")

Netezza:

- `user`. The user used to log in to the server
- `password`. The password used to log on to the server
- `server`. This field contains the host name of the server and the database holding the relevant schemas: <host>/<database>
- `port`. Specifies the port on the server (default = 5480)
- `schema`. The schema containing the tables.
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "ssl=true")
- `pathToDriver` The path to the folder containing the Netezza JDBC driver JAR file (nzjdbc.jar).

Impala:

- `user`. The user name used to access the server
- `password`. The password for that user
- `server`. The host name of the server
- `port`. Specifies the port on the server (default = 21050)
- `schema`. The database containing the tables
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "SSLKeyStorePwd=*****")

- pathToDriver The path to the folder containing the Impala JDBC driver JAR files.

To be able to use Windows authentication for SQL Server (and PDW), you have to install the JDBC driver. Download the .exe from [Microsoft](#) and run it, thereby extracting its contents to a folder. In the extracted folder you will find the file sqljdbc_4.0/enu/auth/x64/sqljdbc_auth.dll (64-bits) or sqljdbc_4.0/enu/auth/x86/sqljdbc_auth.dll (32-bits), which needs to be moved to location on the system path, for example to c:/windows/system32. When using a Windows domain to log in to SQL Server, DatabaseConnector must rely on a non-Microsoft driver. This driver has know issues with retrieving dates. We therefor recommend to either use Windows integrated security, or if a different user is needed, try running RStudio using that user: runas /netonly /user:domain\username "C:\path\to\rstudio\bin\rstudio.exe".

Examples

```
## Not run:
conn <- connect(dbms = "mysql",
                server = "localhost",
                user = "root",
                password = "xxx",
                schema = "cdm_v4")
dbGetQuery(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

conn <- connect(dbms = "sql server", server = "RNDUSRDHIT06.jnj.com", schema = "Vocabulary")
dbGetQuery(conn, "SELECT COUNT(*) FROM concept")
disconnect(conn)

conn <- connect(dbms = "oracle",
                server = "127.0.0.1/xe",
                user = "system",
                password = "xxx",
                schema = "test")
dbGetQuery(conn, "SELECT COUNT(*) FROM test_table")
disconnect(conn)

conn <- connect(dbms = "postgresql",
                connectionString = "jdbc:postgresql://127.0.0.1:5432/cmd_database")
dbGetQuery(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)


## End(Not run)
```

---

createConnectionDetails

createConnectionDetails

Description

createConnectionDetails creates a list containing all details needed to connect to a database. There are three ways to call this function:

- createConnectionDetails(dbms, user, domain, password, server, port, schema, extraSettings
- createConnectionDetails(dbms, connectionString)
- createConnectionDetails(dbms, connectionString, user, password)

Arguments

dbms	The type of DBMS running on the server. Valid values are

- "mysql" for MySQL
- "oracle" for Oracle
- "postgresql" for PostgreSQL
- "redshift" for Amazon Redshift
- "sql server" for Microsoft SQL Server
- "pdw" for Microsoft Parallel Data Warehouse (PDW)
- "netezza" for IBM Netezza
- "bigquery" for Google BigQuery

user	The user name used to access the server.
domain	For SQL Server only: the Windows domain (optional).
password	The password for that user.
server	The name of the server.
port	(optional) The port on the server to connect to.
schema	(optional) The name of the schema to connect to.
extraSettings	(optional) Additional configuration settings specific to the database provider to configure things as security for SSL. These must follow the format for the JDBC connection for the RDBMS specified in dbms.
oracleDriver	Specify which Oracle drive you want to use. Choose between "thin" or "oci".
connectionString	
	The JDBC connection string. If specified, the server, port, extraSettings, and oracleDriver fields are ignored. If user and password are not specified, they are assumed to already be included in the connection string.
pathToDriver	Path to the JDBC driver JAR files. Currently only needed for Impala and Netezza.

Details

This function creates a list containing all details needed to connect to a database. The list can then be used in the connect function.

Value

A list with all the details needed to connect to a database.

DBMS parameter details

Depending on the DBMS, the function arguments have slightly different interpretations: MySQL:

- user. The user name used to access the server
- password. The password for that user
- server. The host name of the server
- port. Specifies the port on the server (default = 3306)
- schema. The database containing the tables
- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "SSL Mode=Required")

Oracle:

- user. The user name used to access the server
- password. The password for that user
- server. This field contains the SID, or host and servicename, SID, or TNSName: '<sid>', '<host>/<sid>', '<host>/<service name>', or '<tnsname>'
- port. Specifies the port on the server (default = 1521)
- schema. This field contains the schema (i.e. 'user' in Oracle terms) containing the tables
- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "(PROTOCOL=tcps)")
- oracleDriver The driver to be used. Choose between "thin" or "oci".

Microsoft SQL Server:

- user. The user used to log in to the server. If the user is not specified, Windows Integrated Security will be used, which requires the SQL Server JDBC drivers to be installed (see details below).
- domain. Optionally, the domain can be specified here.(See note below).
- password. The password used to log on to the server
- server. This field contains the host name of the server
- port. Not used for SQL Server
- schema. The database containing the tables. If both database and schema are specified (e.g. 'my_database.dbo', then only the database part is used, the schema is ignored.
- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "encrypt=true; trustServerCertificate=false;")

Microsoft PDW:

- user. The user used to log in to the server. If the user is not specified, Windows Integrated Security will be used, which requires the SQL Server JDBC drivers to be installed (see details below).
- password. The password used to log on to the server
- server. This field contains the host name of the server
- port. Not used for SQL Server
- schema. The database containing the tables
- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "encrypt=true; trustServerCertificate=false;")

Connections where the domain need to be specified are not supported. PostgreSQL:

- user. The user used to log in to the server
- password. The password used to log on to the server
- server. This field contains the host name of the server and the database holding the relevant schemas: <host>/<database>
- port. Specifies the port on the server (default = 5432)
- schema. The schema containing the tables.
- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "ssl=true")

Redshift:

- user. The user used to log in to the server

- password. The password used to log on to the server

- server. This field contains the host name of the server and the database holding the relevant schemas: <host>/<database>

- port. Specifies the port on the server (default = 5439)

- schema. The schema containing the tables.

- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "ssl=true&sslfactory=com.amazon.redshift.ssl.NonValidatingFactory")

Netezza:

- user. The user used to log in to the server

- password. The password used to log on to the server

- server. This field contains the host name of the server and the database holding the relevant schemas: <host>/<database>

- port. Specifies the port on the server (default = 5480)

- schema. The schema containing the tables.

- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "ssl=true")

- pathToDriver The path to the folder containing the Netezza JDBC driver JAR file (nzjdbc.jar).

Impala:

- user. The user name used to access the server

- password. The password for that user

- server. The host name of the server

- port. Specifies the port on the server (default = 21050)

- schema. The database containing the tables

- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "SSLKeyStorePwd=*****")

- pathToDriver The path to the folder containing the Impala JDBC driver JAR files.

To be able to use Windows authentication for SQL Server (and PDW), you have to install the JDBC driver. Download the .exe from Microsoft and run it, thereby extracting its contents to a folder. In the extracted folder you will find the file sqljdbc_4.0/enu/auth/x64/sqljdbc_auth.dll (64-bits) or sqljdbc_4.0/enu/auth/x86/sqljdbc_auth.dll (32-bits), which needs to be moved to location on the system path, for example to c:/windows/system32. When using a Windows domain to log in to SQL Server, DatabaseConnector must rely on a non-Microsoft driver. This driver has know issues with retrieving dates. We therefor recommend to either use Windows integrated security, or if a different user is needed, try running RStudio using that user: runas /netonly /user:domain\username "C:\path\to\rstudio\bin\rstudio.exe".

Examples

```
## Not run:
connectionDetails <- createConnectionDetails(dbms = "mysql",
                                             server = "localhost",
                                             user = "root",
                                             password = "blah",
                                             schema = "cdm_v4")
conn <- connect(connectionDetails)
dbGetQuery(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

## End(Not run)
```

DatabaseConnector *DatabaseConnector*

Description

DatabaseConnector

---

disconnect *Disconnect from the server*

Description

This function sends SQL to the server, and returns the results in an ffdf object.

Usage

```
disconnect(connection)
```

Arguments

connection   The connection to the database server.

Examples

```
## Not run:
library(ffbase)
connectionDetails <- createConnectionDetails(dbms = "mysql",
                                             server = "localhost",
                                             user = "root",
                                             password = "blah",
                                             schema = "cdm_v4")
conn <- connect(connectionDetails)
count <- querySql.ffdf(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

## End(Not run)
```

executeSql   *Execute SQL code*

Description

This function executes SQL consisting of one or more statements.

Usage

```
executeSql(connection, sql, profile = FALSE, progressBar = TRUE,
  reportOverallTime = TRUE)
```

Arguments

connection	The connection to the database server.
sql	The SQL to be executed
profile	When true, each separate statement is written to file prior to sending to the server, and the time taken to execute a statement is displayed.
progressBar	When true, a progress bar is shown based on the statements in the SQL code.
reportOverallTime	
	When true, the function will display the overall time taken to execute all statements.

Details

This function splits the SQL in separate statements and sends it to the server for execution. If an error occurs during SQL execution, this error is written to a file to facilitate debugging. Optionally, a progress bar is shown and the total time taken to execute the SQL is displayed. Optionally, each separate SQL statement is written to file, and the execution time per statement is shown to aid in detecting performance issues.

Examples

```
## Not run:
connectionDetails <- createConnectionDetails(dbms = "mysql",
                                             server = "localhost",
                                             user = "root",
                                             password = "blah",
                                             schema = "cdm_v4")
conn <- connect(connectionDetails)
executeSql(conn, "CREATE TABLE x (k INT); CREATE TABLE y (k INT);")
disconnect(conn)

## End(Not run)
```

---

getTableNames *List all tables in a database schema.*

Description

This function returns a list of all tables in a database schema.

Usage

```
getTableNames(connection, databaseSchema)
```

Arguments

connection   The connection to the database server.

databaseSchema The name of the database schema. See details for platform-specific details.

Details

The `databaseSchema` argument is interpreted differently according to the different platforms: SQL Server and PDW: The databaseSchema schema should specify both the database and the schema, e.g. 'my_database.dbo'. PostgreSQL and Redshift: The databaseSchema should specify the schema. Oracle: The databaseSchema should specify the Oracle 'user'. MySql and Impala: The databaseSchema should specify the database.

Value

A character vector of table names. To ensure consistency across platforms, these table names are in upper case.

---

insertTable *Insert a table on the server*

Description

This function sends the data in a data frame or ffdf to a table on the server. Either a new table is created, or the data is appended to an existing table.

Usage

```
insertTable(connection, tableName, data, dropTableIfExists = TRUE,
  createTable = TRUE, tempTable = FALSE, oracleTempSchema = NULL)
```

Arguments

connection	The connection to the database server.
tableName	The name of the table where the data should be inserted.
data	The data frame or ffdf containing the data to be inserted.
dropTableIfExists	
	Drop the table if the table already exists before writing?
createTable	Create a new table? If false, will append to existing table.
tempTable	Should the table created as a temp table?
oracleTempSchema	
	Specifically for Oracle, a schema with write priviliges where temp tables can be created.

Details

This function sends the data in a data frame to a table on the server. Either a new table is created, or the data is appended to an existing table.

Examples

```
## Not run:
connectionDetails <- createConnectionDetails(dbms = "mysql",
                                             server = "localhost",
                                             user = "root",
                                             password = "blah",
                                             schema = "cdm_v4")
conn <- connect(connectionDetails)
data <- data.frame(x = c(1, 2, 3), y = c("a", "b", "c"))
insertTable(conn, "my_table", data)
disconnect(conn)

## End(Not run)
```

---

lowLevelExecuteSql   *Execute SQL code*

Description

This function executes a single SQL statement.

Usage

```
lowLevelExecuteSql(connection, sql)
```

Arguments

connection	The connection to the database server.
sql	The SQL to be executed

---

lowLevelQuerySql *Low level function for retrieving data to a data frame*

Description

This is the equivalent of the [querySql](#) function, except no error report is written when an error occurs.

Usage

```
lowLevelQuerySql(connection, query = "", datesAsString = FALSE)
```

Arguments

connection	The connection to the database server.
query	The SQL statement to retrieve the data
datesAsString	Should dates be imported as character vectors, our should they be converted to R's date format?

Details

Retrieves data from the database server and stores it in a data frame.

Value

A data frame containing the data retrieved from the server

---

lowLevelQuerySql.ffdf *Low level function for retrieving data to an ffdf object*

Description

This is the equivalent of the [querySql.ffdf](#) function, except no error report is written when an error occurs.

Usage

```
lowLevelQuerySql.ffdf(connection, query = "", datesAsString = FALSE)
```

Arguments

connection	The connection to the database server.
query	The SQL statement to retrieve the data
datesAsString	Should dates be imported as character vectors, our should they be converted to R's date format?

Details

Retrieves data from the database server and stores it in an ffdf object. This allows very large data sets to be retrieved without running out of memory.

Value

A ffdf object containing the data. If there are 0 rows, a regular data frame is returned instead (ffdf cannot have 0 rows)

---

querySql   *Retrieve data to a data.frame*

Description

This function sends SQL to the server, and returns the results.

Usage

```
querySql(connection, sql)
```

Arguments

connection   The connection to the database server.

sql   The SQL to be send.

Details

This function sends the SQL to the server and retrieves the results. If an error occurs during SQL execution, this error is written to a file to facilitate debugging.

Value

A data frame.

Examples

```
## Not run:
connectionDetails <- createConnectionDetails(dbms = "mysql",
                                             server = "localhost",
                                             user = "root",
                                             password = "blah",
                                             schema = "cdm_v4")
conn <- connect(connectionDetails)
count <- querySql(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

## End(Not run)
```

querySql.ffdf *Retrieves data to an ffdf object*

Description

This function sends SQL to the server, and returns the results in an ffdf object.

Usage

```
querySql.ffdf(connection, sql)
```

Arguments

connection   The connection to the database server.

sql   The SQL to be send.

Details

Retrieves data from the database server and stores it in an ffdf object. This allows very large data sets to be retrieved without running out of memory. If an error occurs during SQL execution, this error is written to a file to facilitate debugging.

Value

A ffdf object containing the data. If there are 0 rows, a regular data frame is returned instead (ffdf cannot have 0 rows)

Examples

```
## Not run:
library(ffbase)
connectionDetails <- createConnectionDetails(dbms = "mysql",
                                             server = "localhost",
                                             user = "root",
                                             password = "blah",
                                             schema = "cdm_v4")
conn <- connect(connectionDetails)
count <- querySql.ffdf(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

## End(Not run)
```

Index