Package 'DatabaseConnector'

May 6, 2015

Type Package

Title A wrapper around RJDBC containing drivers for various DBMSs.

Version 1.2.1

Date 2015-01-03

Author Martijn J. Schuemie and Marc A. Suchard

Maintainer Martijn Schuemie <schuemie@ohdsi.org>

Description A wrapper around RJDBC containing drivers for various DBMSs.

Depends RJDBC (>= 0.2-5)

Imports rJava,
ff,
ffbase,
SqlRender

License Apache License

R topics documented:

`connect` *connect*

Description

`connect` creates a connection to a database server.

Usage

```
connect(dbms = "sql server", user, password, server, port, schema)
connect(connectionDetails)
```

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
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.
connectionDetails	An object of class connectionDetails as created by the createConnectionDetails function.

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

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 or SID: '<sid>', '<host>/<sid>', '<host>/<service name>'
- port. Specifies the port on the server (default = 1521)
- schema. This field contains the schema (i.e. 'user' in Oracle terms) containing the tables

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.
- 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.

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

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.

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 = 5432)
- schema. The schema containing the tables.

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.

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.

In order to enable Netezza support, place your Netezza jdbc driver at `inst/java/nzjdbc.jar` in this package.

Examples

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

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

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

## End(Not run)
```

`createConnectionDetails` *createConnectionDetails*

Description

`createConnectionDetails` creates a list containing all details needed to connect to a database.

Usage

```
createConnectionDetails(dbms = "sql server", user, domain, password, server,
  port, schema)
```

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

`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.

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

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 or SID: '<sid>', '<host>/<sid>', '<host>/<service name>'
- `port`. Specifies the port on the server (default = 1521)
- `schema`. This field contains the schema (i.e. 'user' in Oracle terms) containing the tables

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.
- `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.

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

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.

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 = 5432)
- `schema`. The schema containing the tables.

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.

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.

In order to enable Netezza support, place your Netezza jdbc driver at `inst/java/nzjdbc.jar` in this package.

Examples

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

## End(Not run)
```

DatabaseConnector *DatabaseConnector*

Description

DatabaseConnector

---

dbGetQuery.ffdf *Retrieve data from server as ffdf object.*

Description

This allows very large data sets to be retrieved without running out of memory.

Usage

```
dbGetQuery.ffdf(connection, query = "", batchSize = 5e+05,
  datesAsString = FALSE)
```

Arguments

connection	The connection to the database server.
query	The SQL statement to retrieve the data
batchSize	The number of rows that will be retrieved at a time from the server. A larger batchSize means less calls to the server so better performance, but too large a batchSize could lead to out-of-memory errors.
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)

Examples

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

## End(Not run)
```

---

dbGetQueryPostgreSql *Retrieve data from server using PostgreSQL specific commands.*

Description

This function is tailored to retrieve large datasets from a PostgreSQL database. Specifically, it temporarily disables auto commit and calls `setFetchSize` on the Statement object. Without these settings, all rows would be fetched from the server, resulting in out-of-memory errors.

Usage

```
dbGetQueryPostgreSql(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

Examples

```
## Not run:
  connectionDetails <- createConnectionDetails(dbms="postgresql", server="localhost/ohdsi",user="postgres",p
  conn <- connect(connectionDetails)
  dbGetQueryPostgreSql(conn,"SELECT * FROM person")
  dbDisconnect(conn)

## End(Not run)
```

dbInsertTable *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

```
dbInsertTable(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"))
  dbInsertTable(conn,"my_table",data)
  dbDisconnect(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);")
  dbDisconnect(conn)

## End(Not run)
```

querySql *Send SQL query*

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")
  dbDisconnect(conn)

## End(Not run)
```

querySql.ffdf *Send SQL query*

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

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

An ffdf object.

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")
  dbDisconnect(conn)

## End(Not run)
```

Index