Package ‘DatabaseConnector’

October 12, 2022

Type Package

Title Connecting to Various Database Platforms

Version 5.1.0

Date 2022-09-23

Description An R 'DataBase Interface' ('DBI') compatible interface to various database platforms ('PostgreSQL', 'Oracle', 'Microsoft SQL Server', 'Amazon Redshift', 'Microsoft Parallel Database Warehouse', 'IBM Netezza', 'Apache Impala', 'Google BigQuery', 'Snowflake', 'Spark', and 'SQLite'). Also includes support for fetching data as 'Andromeda' objects. Uses 'Java Database Connectivity' ('JDBC') to connect to databases (except SQLite).

SystemRequirements Java version 8 or higher (https://www.java.com/)

Depends R (&gt;= 2.10)

Imports rJava, SqlRender (&gt;= 1.8.3), methods, stringr, readr, rlang, utils, DBI (&gt;= 1.0.0), urltools, bit64

Suggests aws.s3, R.utils, withr, testthat, DBItest, knitr, rmarkdown, RSQLite, ssh, Andromeda, dplyr

License Apache License

VignetteBuilder knitr


BugReports https://github.com/OHDSI/DatabaseConnector/issues

Copyright See file COPYRIGHTS

RoxygenNote 7.2.1

Encoding UTF-8

NeedsCompilation no

Author Martijn Schuemie [aut, cre],
Marc Suchard [aut],
Observational Health Data Science and Informatics [cph],
Microsoft Inc. [cph] (SQL Server JDBC driver),
PostgreSQL Global Development Group [cph] (PostgreSQL JDBC driver),
Oracle Inc. [cph] (Oracle JDBC driver),
Amazon Inc. [cph] (RedShift JDBC driver)

Maintainer  Martijn Schuemie <schuemie@ohdsi.org>

Repository  CRAN

Date/Publication  2022-09-23 11:40:02 UTC

R topics documented:

connect ................................................................. 2
createConnectionDetails ........................................ 7
createZipFile ....................................................... 11
DatabaseConnectorDriver ........................................ 11
dbConnect,DatabaseConnectorDriver-method .................. 12
dbms ................................................................. 13
dbUnloadDriver,DatabaseConnectorDriver-method .......... 13
disconnect ......................................................... 14
downloadJdbcDrivers ................................................ 15
dropEmulatedTempTables ......................................... 16
executeSql ........................................................ 17
existsTable ......................................................... 18
getAvailableJavaHeapSpace ....................................... 19
getTableNames ...................................................... 19
insertTable ........................................................ 20
isSqlReservedWord ............................................... 22
jdbcDrivers ........................................................ 22
lowLevelExecuteSql ............................................... 23
lowLevelQuerySql ................................................ 24
lowLevelQuerySqlToAndromeda .................................. 25
querySql .......................................................... 26
querySqlToAndromeda .............................................. 27
renderTranslateExecuteSql ....................................... 29
renderTranslateQueryApplyBatched .............................. 30
renderTranslateQuerySql ......................................... 32
renderTranslateQuerySqlToAndromeda .......................... 34
show,DatabaseConnectorDriver-method ....................... 36

Index  37
connect

Description

class createConnectionDetails function.

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
- "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
- "sqlite" for SQLite
- "sqlite extended" for SQLite with extended types (DATE and DATETIME)
- "spark" for Spark
- "snowflake" for Snowflake
user
The user name used to access the server.
password
The password for that user.
server
The name of the server.
port
(optional) The port on the server 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 a folder containing the JDBC driver JAR files. See downloadJdbcDrivers for instructions on how to download the relevant drivers.

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: 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)
• extraSettings The configuration settings for the connection (i.e. SSL Settings such as "(PROTOCOL=tcp)")
• oracleDriver The driver to be used. Choose between "thin" or "oci".
• pathToDriver The path to the folder containing the Oracle JDBC driver JAR files.

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).
• 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
• extraSettings The configuration settings for the connection (i.e. SSL Settings such as "encrypt=true; trustServerCertificate=false;")
• pathToDriver The path to the folder containing the SQL Server JDBC driver JAR files.

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
• extraSettings The configuration settings for the connection (i.e. SSL Settings such as "encrypt=true; trustServerCertificate=false;")
• pathToDriver The path to the folder containing the SQL Server JDBC driver JAR files.

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)
• **extraSettings** The configuration settings for the connection (i.e. SSL Settings such as "ssl=true")
• **pathToDriver** The path to the folder containing the PostgreSQL JDBC driver JAR files.

**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)
• **extraSettings** The configuration settings for the connection (i.e. SSL Settings such as "ssl=true&sslfactory=com.amazon.redshift.ssl.NonValidatingFactory")
• **pathToDriver** The path to the folder containing the RedShift JDBC driver JAR files.

**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)
• **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)
• **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.

**SQLite:**
• **server.** The path to the SQLite file.

**Spark:**
• connectionString. The connection string (e.g. starting with 'jdbc:spark://my-org.dev.cloud.databricks.com...').
• user. The user name used to access the server.
• password. The password for that user.

Snowflake:
• connectionString. The connection string (e.g. starting with 'jdbc:snowflake://host[:port]?[db=database]').
• user. The user name used to access the server.
• password. The password for that user.

Windows authentication for SQL Server
To be able to use Windows authentication for SQL Server (and PDW), you have to install the JDBC driver. Download the version 9.2.0.zip from Microsoft and extract its contents to a folder. In the extracted folder you will find the file sqljdbc_9.2/enu/auth/x64/mssql-jdbc_auth-9.2.0.x64.dll (64-bits) or ssqjdbc_9.2/enu/auth/x86/mssql-jdbc_auth-9.2.0.x86.dll (32-bits), which needs to be moved to location on the system path, for example to c:/windows/system32. If you not have write access to any folder in the system path, you can also specify the path to the folder containing the dll by setting the environmental variable PATH_TO_AUTH_DLL, so for example Sys.setenv("PATH_TO_AUTH_DLL" = "c:/temp") Note that the environmental variable needs to be set before calling connect for the first time.

Examples
## Not run:
conn <- connect(
  dbms = "postgresql",
  server = "localhost/postgres",
  user = "root",
  password = "xxx"
)
dbGetQuery(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

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

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

cconn <- connect(
  dbms = "postgresql",
  connectionString = "jdbc:postgresql://127.0.0.1:5432/cmd_database"
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, password, server, port, extraSettings, oracleDriver, pathToDriver)
- createConnectionDetails(dbms, connectionString, pathToDriver)
- createConnectionDetails(dbms, connectionString, user, password, pathToDriver)

Arguments

dbms The type of DBMS running on the server. Valid values are
  - "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
  - "sqlite" for SQLite
  - "sqlite extended" for SQLite with extended types (DATE and DATETIME)
  - "spark" for Spark
  - "snowflake" for Snowflake

user The user name used to access the server.

password The password for that user.

server The name of the server.

port (optional) The port on the server 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 a folder containing the JDBC driver JAR files. See downloadJdbcDrivers for instructions on how to download the relevant drivers.

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: 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)
- 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".
- pathToDriver The path to the folder containing the Oracle JDBC driver JAR files.

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).
- 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
- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "encrypt=true; trustServerCertificate=false;")
- pathToDriver The path to the folder containing the SQL Server JDBC driver JAR files.

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
- **extraSettings**: The configuration settings for the connection (i.e. SSL Settings such as "en-\cr\crypt=true; trustServerCertificate=false;")
- **pathToDriver**: The path to the folder containing the SQL Server JDBC driver JAR files.

**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)
- **extraSettings**: The configuration settings for the connection (i.e. SSL Settings such as "ssl=true")
- **pathToDriver**: The path to the folder containing the PostgreSQL JDBC driver JAR files.

**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)
- **extraSettings**: The configuration settings for the connection (i.e. SSL Settings such as "ssl=true&sslfactory=com.amazon.redshift.ssl.NonValidatingFactory")
- **pathToDriver**: The path to the folder containing the RedShift JDBC driver JAR files.

**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)
- **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
createConnectionDetails

- **port**: Specifies the port on the server (default = 21050)
- **extraSettings**: The configuration settings for the connection (i.e. SSL Settings such as "SSLEnterpriseTrustStorePwd=*****")
- **pathToDriver**: The path to the folder containing the Impala JDBC driver JAR files.

SQLite:

- **server**: The path to the SQLite file.

Spark:

- **connectionString**: The connection string (e.g. starting with 'jdbc:spark://my-org.dev.cloud.databricks.com...').
- **user**: The user name used to access the server.
- **password**: The password for that user.

Snowflake:

- **connectionString**: The connection string (e.g. starting with 'jdbc:snowflake://host[:port]?[db=database]').
- **user**: The user name used to access the server.
- **password**: The password for that user.

**Windows authentication for SQL Server**

To be able to use Windows authentication for SQL Server (and PDW), you have to install the JDBC driver. Download the version 9.2.0.zip from Microsoft and extract its contents to a folder. In the extracted folder you will find the file sqljdbc_9.2/enu/auth/x64/mssql-jdbc_auth-9.2.0.x64.dll (64-bits) or sqljdbc_9.2/enu/auth/x86/mssql-jdbc_auth-9.2.0.x86.dll (32-bits), which needs to be moved to location on the system path, for example to c:/windows/system32. If you not have write access to any folder in the system path, you can also specify the path to the folder containing the dll by setting the environmental variable PATH_TO_AUTH_DLL, so for example Sys.setenv("PATH_TO_AUTH_DLL" = "c:/temp") Note that the environmental variable needs to be set before calling connect for the first time.

**Examples**

```r
## Not run:
connectionDetails <- createConnectionDetails(
  dbms = "postgresql",
  server = "localhost/postgres",
  user = "root",
  password = "blah"
)
conn <- connect(connectionDetails)
dbGetQuery(conn, "SELECT COUNT(*) FROM person")
disable(conn)
## End(Not run)
```
createZipFile

Description
Compress files and/or folders into a single zip file

Usage
createZipFile(zipFile, files, rootFolder = getwd(), compressionLevel = 9)

Arguments
- zipFile: The path to the zip file to be created.
- files: The files and/or folders to be included in the zip file. Folders will be included recursively.
- rootFolder: The root folder. All files will be stored with relative paths relative to this folder.
- compressionLevel: A number between 1 and 9. 9 compresses best, but it also takes the longest.

Details
Uses Java's compression library to create a zip file. It is similar to utils::zip, except that it does not require an external zip tool to be available on the system path.

DatabaseConnectorDriver

Description
Create a DatabaseConnectorDriver object

Usage
DatabaseConnectorDriver()
dbConnect,DatabaseConnectorDriver-method

Create a connection to a DBMS

Description

Connect to a database. This function is synonymous with the `connect` function, except a dummy driver needs to be specified.

Usage

```r
## S4 method for signature 'DatabaseConnectorDriver'
dbConnect(drv, 
...)
```

Arguments

- `drv`  
The result of the `DatabaseConnectorDriver` function
- `...`  
Other parameters. These are the same as expected by the `connect` function.

Value

Returns a `DatabaseConnectorConnection` object that can be used with most of the other functions in this package.

Examples

```r
## Not run:
conn <- dbConnect(DatabaseConnectorDriver(),
  dbms = "postgresql",
  server = "localhost/ohdsi",
  user = "joe",
  password = "secret"
)
querySql(conn, "SELECT * FROM cdm_synpuf.person;")
```

## End(Not run)
Get the database platform from a connection

**Description**

The SqlRender package provides functions that translate SQL from OHDSI-SQL to a target SQL dialect. These functions need the name of the database platform to translate to. The `dbms` function returns the dbms for any DBI connection that can be passed along to SqlRender translation functions (see example).

**Usage**

```r
dbms(connection)
```

**Arguments**

- **connection**  
  A DBI (or DatabaseConnector) connection

**Value**

The name of the database (dbms) used by SqlRender

**Examples**

```r
library(DatabaseConnector)
con <- connect(dbms = "sqlite", server = "::memory::")
dbms(con)
#> [1] "sqlite"
SqlRender::translate("DATEADD(d, 365, dateColumn)", targetDialect = dbms(con))
#> "CAST(STRFTIME('%s', DATETIME(dateColumn, 'unixepoch', (365)||' days')) AS REAL)"
disconnect(con)
```

**dbUnloadDriver,DatabaseConnectorDriver-method**

Load and unload database drivers

**Description**

These methods are deprecated, please consult the documentation of the individual backends for the construction of driver instances.

`dbDriver()` is a helper method used to create a new driver object given the name of a database or the corresponding R package. It works through convention: all DBI-extending packages should provide an exported object with the same name as the package. `dbDriver()` just looks for this object in the right places: if you know what database you are connecting to, you should call the function directly.

`dbUnloadDriver()` is not implemented for modern backends.
Usage

```r
# S4 method for signature 'DatabaseConnectorDriver'
dbUnloadDriver(drv, ...)
```

Arguments

- `drv` an object that inherits from `DBIDriver` as created by `dbDriver`.
- `...` any other arguments are passed to the driver `drvName`.

Details

The client part of the database communication is initialized (typically dynamically loading C code, etc.) but note that connecting to the database engine itself needs to be done through calls to `dbConnect`.

Value

In the case of `dbDriver`, an driver object whose class extends `DBIDriver`. This object may be used to create connections to the actual DBMS engine.

In the case of `dbUnloadDriver`, a logical indicating whether the operation succeeded or not.

See Also

Other `DBIDriver` generics: `DBIDriver-class`, `dbCanConnect()`, `dbConnect()`, `dbDataType()`, `dbGetInfo()`, `dbIsReadOnly()`, `dbIsValid()`, `dbListConnections()`

Other `DBIDriver` generics: `DBIDriver-class`, `dbCanConnect()`, `dbConnect()`, `dbDataType()`, `dbGetInfo()`, `dbIsReadOnly()`, `dbIsValid()`, `dbListConnections()`

---

**disconnect**

*Disconnect from the server*

Description

Close the connection to the server.

Usage

```r
disconnect(connection)
```

Arguments

- `connection` The connection to the database server.
downloadJdbcDrivers

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

## End(Not run)

downloadJdbcDrivers  Download DatabaseConnector JDBC Jar files

### Description

Download the DatabaseConnector JDBC drivers from https://ohdsi.github.io/DatabaseConnectorJars/

### Usage

downloadJdbcDrivers(
  dbms,
  pathToDriver = Sys.getenv("DATABASECONNECTOR_JAR_FOLDER"),
  method = "auto",
  ...
)

### Arguments

dbms  The type of DBMS to download Jar files for.
  • "postgresql" for PostgreSQL
  • "redshift" for Amazon Redshift
  • "sql server", "pdw" or "synapse" for Microsoft SQL Server
  • "oracle" for Oracle
  • "spark" for Spark
  • "snowflake" for Snowflake

pathToDriver  The full path to the folder where the JDBC driver .jar files should be downloaded to. By default the value of the environment variable "DATABASECONNECTOR_JAR_FOLDER" is used.

method  The method used for downloading files. See ?download.file for details and options.

...  Further arguments passed on to download.file
Details
The following versions of the JDBC drivers are currently used:

- PostgreSQLV42.2.18
- RedShiftV2.1.0.9
- SQL ServerV8.4.1.zip
- OracleV19.8
- SparkV2.6.21
- SnowflakeV3.13.22

Value
Invisibly returns the destination if the download was successful.

Examples

```r
## Not run:
downloadJdbcDrivers("redshift")
## End(Not run)
```

--

`dropEmulatedTempTables`

*Drop all emulated temp tables.*

Description

On some DBMSs, like Oracle and BigQuery, DatabaseConnector through SqlRender emulates temp tables in a schema provided by the user. Ideally, these tables are deleted by the application / R script creating them, but for various reasons orphan temp tables may remain. This function drops all emulated temp tables created in this session only.

Usage

```r
dropEmulatedTempTables(
  connection,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema")
)
```

Arguments

- `connection`: The connection to the database server.
- `tempEmulationSchema`: Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.
executeSql

Value
In invisibly returns the list of deleted emulated temp tables.

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,
  errorReportFile = file.path(getwd(), "errorReportSql.txt"),
  runAsBatch = FALSE
)

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.
  errorReportFile | The file where an error report will be written if an error occurs. Defaults to ‘errorReportSql.txt’ in the current working directory.
  runAsBatch | When true the SQL statements are sent to the server as a single batch, and executed there. This will be faster if you have many small SQL statements, but there will be no progress bar, and no per-statement error messages. If the database platform does not support batched updates the query is executed without batching.

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

```r
## Not run:
connectionDetails <- createConnectionDetails(
  dbms = "postgresql",
  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)

---

**existsTable**  
*Does the table exist?*

### Description

Checks whether a table exists. Accounts for surrounding escape characters. Case insensitive.

### Usage

`existsTable(connection, databaseSchema, tableName)`

### Arguments

- `connection`  
The connection to the database server.

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

- `tableName`  
The name of the table to check.

### 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'. Impala: the `databaseSchema` should specify the database. Oracle: The `databaseSchema` should specify the Oracle 'user'. All other: The `databaseSchema` should specify the schema.

### Value

A logical value indicating whether the table exits.
getAvailableJavaHeapSpace

Get available Java heap space

Description
For debugging purposes: get the available Java heap space.

Usage
getAvailableJavaHeapSpace()

Value
The Java heap space (in bytes).

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'. Impala: the databaseSchema should specify the database. Oracle: The databaseSchema should specify the Oracle 'user'. All other: The databaseSchema should specify the schema.

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 to a table on the server. Either a new table is created, or the data is appended to an existing table.

Usage

```r
insertTable(
  connection,
  databaseSchema = NULL,
  tableName,
  data,
  dropTableIfExists = TRUE,
  createTable = TRUE,
  tempTable = FALSE,
  oracleTempSchema = NULL,
  tempEmulationSchema =getOption("sqlRenderTempEmulationSchema"),
  bulkLoad = Sys.getenv("DATABASE_CONNECTOR_BULK UPLOAD"),
  useMppBulkLoad = Sys.getenv("USE_MPP_BULK LOAD"),
  progressBar = FALSE,
  camelCaseToSnakeCase = FALSE
)
```

Arguments

- **connection**: The connection to the database server.
- **databaseSchema**: (Optional) The name of the database schema where the table should be located.
- **tableName**: The name of the table where the data should be inserted.
- **data**: The data frame 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**: DEPRECATED: use tempEmulationSchema instead.
- **tempEmulationSchema**: Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.
- **bulkLoad**: If using Redshift, PDW, Hive or Postgres, use more performant bulk loading techniques. Does not work for temp tables (except for HIVE). See Details for requirements for the various platforms.
useMppBulkLoad  DEPRECATED. Use bulkLoad instead.
progressBar    Show a progress bar when uploading?
camelCaseToSnakeCase

If TRUE, the data frame column names are assumed to use camelCase and are converted to snake_case before uploading.

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. NA values are inserted as null values in the database.

Bulk uploading:
Redshift: The MPP bulk loading relies upon the CloudyR S3 library to test a connection to an S3 bucket using AWS S3 credentials. Credentials are configured directly into the System Environment using the following keys: Sys.setenv("AWS_ACCESS_KEY_ID" = "some_access_key_id", "AWS_SECRET_ACCESS_KEY" = "some_secret_access_key", "AWS_DEFAULT_REGION" = "some_aws_region", "AWS_BUCKET_NAME" = "some_bucket_name", "AWS_OBJECT_KEY" = "some_object_key", "AWS_SSE_TYPE" = "server_side_encryption_type").

PDW: The MPP bulk loading relies upon the client having a Windows OS and the DWLoader exe installed, and the following permissions granted: –Grant BULK Load permissions - needed at a server level USE master; GRANT ADMINISTER BULK OPERATIONS TO user; –Grant Staging database permissions - we will use the user db. USE scratch; EXEC sp_addrolemember 'db_ddladmin', user; Set the R environment variable DWLOADER_PATH to the location of the binary.

PostgreSQL: Uses the 'pg' executable to upload. Set the POSTGRES_PATH environment variable to the Postgres binary path, e.g. 'C:/Program Files/PostgreSQL/11/bin'.

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

## bulk data insert with Redshift or PDW
connectionDetails <- createConnectionDetails(
  dbms = "redshift",
  server = "localhost",
  user = "root",
  password = "blah",
  schema = "cdm_v5"
)
conn <- connect(connectionDetails)
data <- data.frame(x = c(1, 2, 3), y = c("a", "b", "c"))
insertTable(
  connection = connection,
  databaseSchema = "scratch",
  tableName = "somedata",
  data = data,
  dropTableIfExists = TRUE,
  createTable = TRUE,
  tempTable = FALSE,
  bulkLoad = TRUE
)  # or, Sys.setenv("DATABASE_CONNECTOR_BULK_UPLOAD" = TRUE)

## End(Not run)

### isSqlReservedWord

Test a character vector of SQL names for SQL reserved words

**Description**

This function checks a character vector against a predefined list of reserved SQL words.

**Usage**

```r
isSqlReservedWord(sqlNames, warn = FALSE)
```

**Arguments**

- `sqlNames`: A character vector containing table or field names to check.
- `warn`: (logical) Should a warn be thrown if invalid SQL names are found?

**Value**

A logical vector with length equal to `sqlNames` that is `TRUE` for each name that is reserved and `FALSE` otherwise.

### jdbcDrivers

How to download and use JDBC drivers for the various data platforms.

**Description**

Below are instructions for downloading JDBC drivers for the various data platforms. Once downloaded use the `pathToDriver` argument in the `connect` or `createConnectionDetails` functions to point to the driver. Alternatively, you can set the `DATABASECONNECTOR_JAR_FOLDER` environmental variable, for example in your `.Renviron` file (recommended).
**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

```r
lowLevelQuerySql(
  connection,
  query,
  datesAsString = FALSE,
  integerAsNumeric = getOption("databaseConnectorIntegerAsNumeric", default = TRUE),
  integer64AsNumeric = getOption("databaseConnectorInteger64AsNumeric", default = TRUE)
)
```

Arguments

- `connection` The connection to the database server.
- `query` The SQL statement to retrieve the data
- `datesAsString` Logical: Should dates be imported as character vectors, or should they be converted to R’s date format?
- `integerAsNumeric` Logical: should 32-bit integers be converted to numeric (double) values? If FALSE 32-bit integers will be represented using R’s native `integer` class.
- `integer64AsNumeric` Logical: should 64-bit integers be converted to numeric (double) values? If FALSE 64-bit integers will be represented using `bit64::integer64`.

Details

Retrieves data from the database server and stores it in a data frame. Null values in the database are converted to NA values in R.

Value

A data frame containing the data retrieved from the server
Low level function for retrieving data to a local Andromeda object

Description

This is the equivalent of the `querySqlToAndromeda` function, except no error report is written when an error occurs.

Usage

```r
lowLevelQuerySqlToAndromeda(
  connection,
  query,
  andromeda,
  andromedaTableName,
  datesAsString = FALSE,
  integerAsNumeric = getOption("databaseConnectorIntegerAsNumeric", default = TRUE),
  integer64AsNumeric = getOption("databaseConnectorInteger64AsNumeric", default = TRUE)
)
```

Arguments

- `connection`: The connection to the database server.
- `query`: The SQL statement to retrieve the data
- `andromeda`: An open Andromeda object, for example as created using `andromeda`.
- `andromedaTableName`: The name of the table in the local Andromeda object where the results of the query will be stored.
- `datesAsString`: Should dates be imported as character vectors, or should they be converted to R’s date format?
- `integerAsNumeric`: Logical: should 32-bit integers be converted to numeric (double) values? If FALSE 32-bit integers will be represented using R’s native `Integer` class.
- `integer64AsNumeric`: Logical: should 64-bit integers be converted to numeric (double) values? If FALSE 64-bit integers will be represented using `bit64::integer64`.

Details

Retrieves data from the database server and stores it in a local Andromeda object. This allows very large data sets to be retrieved without running out of memory. Null values in the database are converted to NA values in R. If a table with the same name already exists in the local Andromeda object it is replaced.
Value
Invisibly returns the andromeda. The Andromeda object will have a table added with the query results.

querySql
Retrieve data to a data.frame

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

Usage
querySql(
  connection, sql,
  errorReportFile = file.path(getwd(), "errorReportSql.txt"),
  snakeCaseToCamelCase = FALSE,
  integerAsNumeric = getOption("databaseConnectorIntegerAsNumeric", default = TRUE),
  integer64AsNumeric = getOption("databaseConnectorInteger64AsNumeric", default = TRUE)
)

Arguments
connection The connection to the database server.
sql The SQL to send.
errorReportFile The file where an error report will be written if an error occurs. Defaults to 'errorReportSql.txt' in the current working directory.

snakeCaseToCamelCase If true, field names are assumed to use snake_case, and are converted to camel-Case.

integerAsNumeric Logical: should 32-bit integers be converted to numeric (double) values? If FALSE 32-bit integers will be represented using R’s native Integer class.

integer64AsNumeric Logical: should 64-bit integers be converted to numeric (double) values? If FALSE 64-bit integers will be represented using bit64::integer64.

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. Null values in the database are converted to NA values in R.
querySqlToAndromeda

Value

A data frame.

Examples

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

querySqlToAndromeda  Retrieves data to a local Andromeda object

Description

This function sends SQL to the server, and returns the results in a local Andromeda object

Usage

```r
querySqlToAndromeda(
  connection,                     
  sql,                           
  andromeda,                     
  andromedaTableName,            
  errorReportFile = file.path(getwd(), "errorReportSql.txt"), 
  snakeCaseToCamelCase = FALSE, 
  integerAsNumeric =getOption("databaseConnectorIntegerAsNumeric", default = TRUE), 
  integer64AsNumeric =getOption("databaseConnectorInteger64AsNumeric", default = TRUE)
)
```

Arguments

- **connection**  The connection to the database server.
- **sql**  The SQL to be sent.
- **andromeda**  An open connection to a Andromeda object, for example as created using `andromeda`.
- **andromedaTableName**  The name of the table in the local Andromeda object where the results of the query will be stored.
querySqlToAndromeda

errorReportFile
The file where an error report will be written if an error occurs. Defaults to 'errorReportSql.txt' in the current working directory.

snakeCaseToCamelCase
If true, field names are assumed to use snake_case, and are converted to camelCase.

integerAsNumeric
Logical: should 32-bit integers be converted to numeric (double) values? If FALSE 32-bit integers will be represented using R’s native Integer class.

integer64AsNumeric
Logical: should 64-bit integers be converted to numeric (double) values? If FALSE 64-bit integers will be represented using bit64::integer64.

Details
Retrieves data from the database server and stores it in a local Andromeda 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. Null values in the database are converted to NA values in R. If a table with the same name already exists in the local Andromeda object it is replaced.

Value
Invisibly returns the andromeda. The Andromeda object will have a table added with the query results.

Examples
## Not run:
andromeda <- Andromeda::andromeda()
connectionDetails <- createConnectionDetails(
  dbms = "postgresql",
  server = "localhost",
  user = "root",
  password = "blah",
  schema = "cdm_v4"
)
conn <- connect(connectionDetails)
querySqlToAndromeda(
  connection = conn,
  sql = "SELECT * FROM person;",
  andromeda = andromeda,
  andromedaTableName = "foo"
)
disconnect(conn)

andromeda$foo

## End(Not run)
renderTranslateExecuteSql

Render, translate, execute SQL code

Description
This function renders, translates, and executes SQL consisting of one or more statements.

Usage
renderTranslateExecuteSql(
  connection,
  sql,
  profile = FALSE,
  progressBar = TRUE,
  reportOverallTime = TRUE,
  errorReportFile = file.path(getwd(), "errorReportSql.txt"),
  runAsBatch = FALSE,
  oracleTempSchema = NULL,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  ...
)

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.
errorReportFile The file where an error report will be written if an error occurs. Defaults to 'errorReportSql.txt' in the current working directory.
runAsBatch When true the SQL statements are sent to the server as a single batch, and executed there. This will be faster if you have many small SQL statements, but there will be no progress bar, and no per-statement error messages. If the database platform does not support batched updates the query is executed as ordinarily.
oracleTempSchema DEPRECATED: use tempEmulationSchema instead.
tempEmulationSchema Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.
... Parameters that will be used to render the SQL.
Details

This function calls the render and translate functions in the SqlRender package before calling executeSql.

Examples

```r
## Not run:
connectionDetails <- createConnectionDetails(
  dbms = "postgresql",
  server = "localhost",
  user = "root",
  password = "blah",
  schema = "cdm_v4"
)
conn <- connect(connectionDetails)
renderTranslateExecuteSql(connection,
  sql = "SELECT * INTO #temp FROM @schema.person;",
  schema = "cdm_synpuf"
)
disconnect(conn)
## End(Not run)
```

renderTranslateQueryApplyBatched

Render, translate, and perform process to batches of data.

Description

This function renders, and translates SQL, sends it to the server, processes the data in batches with a call back function. Note that this function should perform a row-wise operation. This is designed to work with massive data that won’t fit in to memory.

The batch sizes are determined by the java virtual machine and will depend on the data.

Usage

```r
renderTranslateQueryApplyBatched(
  connection,  
  sql,         
  fun,         
  args = list(),
  errorReportFile = file.path(getwd(), "errorReportSql.txt"),
  snakeCaseToCamelCase = FALSE,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  integerAsNumeric = getOption("databaseConnectorIntegerAsNumeric", default = TRUE),
  integer64AsNumeric = getOption("databaseConnectorInteger64AsNumeric", default = TRUE),
  ...  
)
```
Arguments

- **connection**: The connection to the database server.
- **sql**: The SQL to be send.
- **fun**: Function to apply to batch. Must take data.frame and integer position as parameters.
- **args**: List of arguments to be passed to function call.
- **errorReportFile**: The file where an error report will be written if an error occurs. Defaults to 'errorReportSql.txt' in the current working directory.
- **snakeCaseToCamelCase**: If true, field names are assumed to use snake_case, and are converted to camelCase.
- **tempEmulationSchema**: Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.
- **integerAsNumeric**: Logical: should 32-bit integers be converted to numeric (double) values? If FALSE 32-bit integers will be represented using R's native Integer class.
- **integer64AsNumeric**: Logical: should 64-bit integers be converted to numeric (double) values? If FALSE 64-bit integers will be represented using bit64::integer64.
- **...**: Parameters that will be used to render the SQL.

Details

This function calls the render and translate functions in the SqlRender package before calling querySql.

Value

Invisibly returns a list of outputs from each call to the provided function.

Examples

```r
## Not run:
connectionDetails <- createConnectionDetails(
  dbms = "postgresql",
  server = "localhost",
  user = "root",
  password = "blah",
  schema = "cdm_v4"
)
connection <- connect(connectionDetails)

# First example: write data to a large CSV file:
filepath <- "myBigFile.csv"
writeBatchesToCsv <- function(data, position, ...) {
  #...
renderTranslateQuerySql

**Render, translate, and query to data.frame**

**Description**

This function renders, and translates SQL, sends it to the server, and returns the results as a data.frame.

**Usage**

```r
renderTranslateQuerySql(
  connection,
  sql,
  errorReportFile = file.path(getwd(), "errorReportSql.txt"),
  snakeCaseToCamelCase = FALSE,
```

---

```
write.csv(data, filepath, append = position != 1)
return(NULL)
}
renderTranslateQueryApplyBatched(connection,
  "SELECT * FROM @schema.person;",
  schema = "cdm_synpuf",
  fun = writeBatchesToCsv
)

# Second example: write data to Andromeda
# (Alternative to querySqlToAndromeda if some local computation needs to be applied)
bigResults <- Andromeda::andromeda()
writeBatchesToAndromeda <- function(data, position, ...)
  data$p <- EmpiricalCalibration::computeTraditionalP(data$logRr, data$logSeRr)
  if (position == 1) {
    bigResults$rrs <- data
  } else {
    Andromeda::appendToTable(bigResults$rrs, data)
  }
return(NULL)
}
sql <- "SELECT target_id, comparator_id, log_rr, log_se_rr FROM @schema.my_results;"
renderTranslateQueryApplyBatched(connection,
  sql,
  fun = writeBatchesToAndromeda,
  schema = "my_results",
  snakeCaseToCamelCase = TRUE
)
disconnect(connection)
```
oracleTempSchema = NULL,
tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
integerAsNumeric = getOption("databaseConnectorIntegerAsNumeric", default = TRUE),
integer64AsNumeric = getOption("databaseConnectorInteger64AsNumeric", default = TRUE),
...)

Arguments

connection  The connection to the database server.
sql  The SQL to be send.
errorReportFile  The file where an error report will be written if an error occurs. Defaults to 'errorReportSql.txt' in the current working directory.
snakeCaseToCamelCase  If true, field names are assumed to use snake_case, and are converted to camel-Case.
oracleTempSchema  DEPRECATED: use tempEmulationSchema instead.
tempEmulationSchema  Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.
integerAsNumeric  Logical: should 32-bit integers be converted to numeric (double) values? If FALSE 32-bit integers will be represented using R's native Integer class.
integer64AsNumeric  Logical: should 64-bit integers be converted to numeric (double) values? If FALSE 64-bit integers will be represented using bit64::integer64.
...  Parameters that will be used to render the SQL.

Details

This function calls the render and translate functions in the SqlRender package before calling querySql.

Value

A data frame.

Examples

## Not run:
connectionDetails <- createConnectionDetails(
  dbms = "postgresql",
  server = "localhost",
  user = "root",
  password = "blah",
renderTranslateQuerySqlToAndromeda

Render, translate, and query to local Andromeda

Description

This function renders, and translates SQL, sends it to the server, and returns the results as an ffdf object.

Usage

renderTranslateQuerySqlToAndromeda(
  connection,
  sql,
  andromeda,
  andromedaTableName,
  errorReportFile = file.path(getwd(), "errorReportSql.txt"),
  snakeCaseToCamelCase = FALSE,
  oracleTempSchema = NULL,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  integerAsNumeric = getOption("databaseConnectorIntegerAsNumeric", default = TRUE),
  integer64AsNumeric = getOption("databaseConnectorInteger64AsNumeric", default = TRUE),
  ...
)

Arguments

- **connection**: The connection to the database server.
- **sql**: The SQL to be send.
- **andromeda**: An open Andromeda object, for example as created using `andromeda`.
- **andromedaTableName**: The name of the table in the local Andromeda object where the results of the query will be stored.
- **errorReportFile**: The file where an error report will be written if an error occurs. Defaults to 'errorReportSql.txt' in the current working directory.
snakeCaseToCamelCase
   If true, field names are assumed to use snake_case, and are converted to camelCase.
oracleTempSchema
   DEPRECATED: use tempEmulationSchema instead.
tempEmulationSchema
   Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.
integerAsNumeric
   Logical: should 32-bit integers be converted to numeric (double) values? If FALSE 32-bit integers will be represented using R’s native `Integer` class.
integer64AsNumeric
   Logical: should 64-bit integers be converted to numeric (double) values? If FALSE 64-bit integers will be represented using `bit64::integer64`.

Details
   This function calls the `render` and `translate` functions in the SqlRender package before calling `querySqlToAndromeda`.

Value
   Invisibly returns the andromeda. The Andromeda object will have a table added with the query results.

Examples
   ```r
   ## Not run:
   connectionDetails <- createConnectionDetails(
     dbms = "postgresql",
     server = "localhost",
     user = "root",
     password = "blah",
     schema = "cdm_v4"
   )
   conn <- connect(connectionDetails)
   renderTranslatequerySqlToAndromeda(conn,
     sql = "SELECT * FROM @schema.person",
     schema = "cdm_synpuf",
     andromeda = andromeda,
     andromedaTable = "foo"
   )
   disconnect(conn)
   andromeda$foo
   ## End(Not run)
```
show,DatabaseConnectorDriver-method

Show an Object

Description
Display the object, by printing, plotting or whatever suits its class. This function exists to be specialized by methods. The default method calls showDefault.

Formal methods for show will usually be invoked for automatic printing (see the details).

Usage
## S4 method for signature 'DatabaseConnectorDriver'
show(object)

Arguments
object Any R object

Details
Objects from an S4 class (a class defined by a call to setClass) will be displayed automatically is if by a call to show. S4 objects that occur as attributes of S3 objects will also be displayed in this form; conversely, S3 objects encountered as slots in S4 objects will be printed using the S3 convention, as if by a call to print.

Methods defined for show will only be inherited by simple inheritance, since otherwise the method would not receive the complete, original object, with misleading results. See the simpleInheritanceOnly argument to setGeneric and the discussion in setIs for the general concept.

Value
show returns an invisible NULL.

See Also
showMethods prints all the methods for one or more functions.
Index

andromeda, 25, 27, 34
connect, 2, 8, 12, 22
createConnectionDetails, 3, 7, 22
createZipFile, 11
DatabaseConnectorDriver, 11
dbCanConnect, 14
dbConnect, 14
dbConnect, DatabaseConnectorDriver-method, 12
dbDataType, 14
dbGetInfo, 14
dbIsReadOnly, 14
dbIsValid, 14
dbListConnections, 14
dbms, 13
dbUnloadDriver, DatabaseConnectorDriver-method, 13
disconnect, 14
downloadJdbcDrivers, 3, 8, 15, 23
dropEmulatedTempTables, 16
executeSql, 17, 30
existsTable, 18
getAvailableJavaHeapSpace, 19
getTableNames, 19
insertTable, 20
isSqlReservedWord, 22
jdbcDrivers, 22
lowLevelExecuteSql, 23
lowLevelQuerySql, 24
lowLevelQuerySqlToAndromeda, 25
print, 36
querySql, 24, 26, 31, 33
querySqlToAndromeda, 25, 27, 35
renderTranslateExecuteSql, 29
renderTranslateQueryApplyBatched, 30
renderTranslateQuerySql, 32
renderTranslateQuerySqlToAndromeda, 34
setClass, 36
setGeneric, 36
setIs, 36
show, DatabaseConnectorDriver-method, 36
showDefault, 36
showMethods, 36

37