Package 'DatabaseConnector'

June 8, 2022

Type Package

Title Connecting to Various Database Platforms

Version 5.0.3

Date 2022-06-07

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', '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 (>= 2.10)

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

Suggests aws.s3,
R.utils,
withr,
testthat,
DBItest,
knitr,
rmarkdown,
RSQLite,
ssh,
Andromeda (>= 0.6.0),
dplyr

License Apache License

VignetteBuilder knitr

URL https://ohdsi.github.io/DatabaseConnector/, https://github.com/OHDSI/DatabaseConnector

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

Copyright See file COPYRIGHTS

RoxygenNote 7.2.0

Encoding UTF-8

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, password, server, port, extraSettings, oracleDriver, pathToDriver)
- connect(connectionDetails)
- connect(dbms, connectionString, pathToDriver))
- connect(dbms, connectionString, user, password, pathToDriver)

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

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=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 "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.

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 ssqljdbc_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)

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, 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

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 "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.

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 ssqljdbc_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:
connectionDetails <- createConnectionDetails(
  dbms = "postgresql",
  server = "localhost/postgres",
  user = "root",
  password = "blah"
)
conn <- connect(connectionDetails)
dbGetQuery(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

## End(Not run)
```

---

createZipFile   *Compress files and/or folders into a single zip file*

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
```
Create a DatabaseConnectorDriver object

Description

Create a DatabaseConnectorDriver object

Usage

```
DatabaseConnectorDriver()
```

```
dbAppendTable,DatabaseConnectorConnection,character,data.frame-method
```
Insert rows into a table

Description

The dbAppendTable() method assumes that the table has been created beforehand, e.g. with [dbCreateTable()](). The default implementation calls [sqlAppendTableTemplate()]() and then [dbExecute()]() with the param argument. Backends compliant to ANSI SQL 99 which use ? as a placeholder for prepared queries don't need to override it. Backends with a different SQL syntax which use ? as a placeholder for prepared queries can override [sqlAppendTable()](). Other backends (with different placeholders or with entirely different ways to create tables) need to override the dbAppendTable() method.

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character,data.frame'
dbAppendTable(
  conn,
  name,
  value,
  temporary = FALSE,
  oracleTempSchema = NULL,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  ...,
  row.names = NULL
)
```

Arguments

conn	A [DBIConnection]() object, as returned by [dbConnect()]().
name	Name of the table, escaped with [dbQuoteIdentifier()]().
value	A data frame of values. The column names must be consistent with those in the target table in the database.
temporary	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 ta-
                bles. To emulate temp tables, provide a schema with write privileges where
                temp tables can be created.

...   Other arguments used by individual methods.

row.names   Must be `NULL`.

Details

The `row.names` argument is not supported by this method. Process the values with [sqlRownamesToColumn()](#)
before calling this method.

See Also

Other DBIConnection generics: [DBIConnection-class](#), [dbCreateTable()](#), [dbDataType()](#), [dbDisconnect()](#),
[dbExecute()](#), [dbExistsTable()](#), [dbGetException()](#), [dbGetInfo()](#), [dbGetQuery()](#), [dbIsReadOnly()](#),
[dbIsValid()](#), [dbListFields()](#), [dbListObjects()](#), [dbListResults()](#), [dbListTables()](#), [dbReadTable()](#),
[dbRemoveTable()](#), [dbSendQuery()](#), [dbSendStatement()](#), [dbWriteTable()](#)

---

dbClearResult,DatabaseConnectorResult-method
                                Clear a result set

Description

Frees all resources (local and remote) associated with a result set. In some cases (e.g., very large
result sets) this can be a critical step to avoid exhausting resources (memory, file descriptors, etc.)

Usage

```
## S4 method for signature 'DatabaseConnectorResult'
dbClearResult(res, ...)
```

Arguments

res   An object inheriting from [DBIResult.](#)
...   Other arguments passed on to methods.

Value

dbClearResult() returns TRUE, invisibly, for result sets obtained from both dbSendQuery() and
dbSendStatement(). An attempt to close an already closed result set issues a warning in both
cases.

See Also

Other DBIResult generics: [DBIResult-class](#), [dbBind()](#), [dbColumnInfo()](#), [dbFetch()](#), [dbGetInfo()](#),
[dbGetRowCount()](#), [dbGetRowsAffected()](#), [dbGetStatement()](#), [dbHasCompleted()](#), [dbIsReadOnly()](#),
[dbIsValid()](#), [dbQuoteIdentifier()](#), [dbQuoteLiteral()](#), [dbQuoteString()](#), [dbUnquoteIdentifier()](#)

```
dbColumnInfo,DatabaseConnectorResult-method
```
Information about result types

Description

Produces a data.frame that describes the output of a query. The data.frame should have as many rows as there are output fields in the result set, and each column in the data.frame describes an aspect of the result set field (field name, type, etc.)

Usage

```
## S4 method for signature 'DatabaseConnectorResult'
dbColumnInfo(res, ...)
```

Arguments

res   An object inheriting from [DBIResult](#).

...   Other arguments passed on to methods.

Value

dbColumnInfo() returns a data frame with at least two columns ″name″ and ″type″ (in that order) (and optional columns that start with a dot). The ″name″ and ″type″ columns contain the names and types of the R columns of the data frame that is returned from [dbFetch()](#). The ″type″ column is of type character and only for information. Do not compute on the ″type″ column, instead use dbFetch(res, n = 0) to create a zero-row data frame initialized with the correct data types.

An attempt to query columns for a closed result set raises an error.

See Also

Other DBIResult generics: [DBIResult-class](#), [dbBind()](#), [dbClearResult()](#), [dbFetch()](#), [dbGetInfo()](#), [dbGetRowCount()](#), [dbGetRowsAffected()](#), [dbGetStatement()](#), [dbHasCompleted()](#), [dbIsReadOnly()](#), [dbIsValid()](#), [dbQuoteIdentifier()](#), [dbQuoteLiteral()](#), [dbQuoteString()](#), [dbUnquoteIdentifier()](#)

```
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

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

Arguments

drv	The result of the link{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

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

## End(Not run)
```

---

dbCreateTable,DatabaseConnectorConnection,character,data.frame-method
Create a table in the database

Description

The default dbCreateTable() method calls [sqlCreateTable()](#) and [dbExecute()](#). Backends compliant to ANSI SQL 99 don't need to override it. Backends with a different SQL syntax can override sqlCreateTable(), backends with entirely different ways to create tables need to override this method.

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character,data.frame'
dbCreateTable(
  conn,
  name,
  fields,
  oracleTempSchema = NULL,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  ...,
  row.names = NULL,
  temporary = FALSE
)
```

Arguments

conn	A [DBIConnection](#) object, as returned by [dbConnect()](#).
name	Name of the table, escaped with [dbQuoteIdentifier()](#).
fields	Either a character vector or a data frame.
	A named character vector: Names are column names, values are types. Names are escaped with [dbQuoteIdentifier()](#). Field types are unescaped.
	A data frame: field types are generated using [dbDataType()](#).
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.
...	Other arguments used by individual methods.
row.names	Must be NULL.
temporary	Should the table created as a temp table?

Details

The row.names argument is not supported by this method. Process the values with [sqlRownamesToColumn()](#) before calling this method.

The argument order is different from the sqlCreateTable() method, the latter will be adapted in a later release of DBI.

See Also

Other DBIConnection generics: [DBIConnection-class](#), [dbAppendTable()](#), [dbDataType()](#), [dbDisconnect()](#), [dbExecute()](#), [dbExistsTable()](#), [dbGetException()](#), [dbGetInfo()](#), [dbGetQuery()](#), [dbIsReadOnly()](#), [dbIsValid()](#), [dbListFields()](#), [dbListObjects()](#), [dbListResults()](#), [dbListTables()](#), [dbReadTable()](#), [dbRemoveTable()](#), [dbSendQuery()](#), [dbSendStatement()](#), [dbWriteTable()](#)

---

dbDisconnect,DatabaseConnectorConnection-method
Disconnect (close) a connection

Description

This closes the connection, discards all pending work, and frees resources (e.g., memory, sockets).

Usage

```
## S4 method for signature 'DatabaseConnectorConnection'
dbDisconnect(conn)
```

Arguments

conn	A [DBIConnection](#) object, as returned by [dbConnect()](#).

Value

dbDisconnect() returns TRUE, invisibly.

See Also

Other DBIConnection generics: DBIConnection-class, dbAppendTable(), dbCreateTable(),
dbDataType(), dbExecute(), dbExistsTable(), dbGetException(), dbGetInfo(), dbGetQuery(),
dbIsReadOnly(), dbIsValid(), dbListFields(), dbListObjects(), dbListResults(), dbListTables(),
dbReadTable(), dbRemoveTable(), dbSendQuery(), dbSendStatement(), dbWriteTable()

---

dbExecute,DatabaseConnectorConnection,character-method
                    Execute an update statement, query number of rows affected, and then
                    close result set

Description

Executes a statement and returns the number of rows affected. dbExecute() comes with a de-
fault implementation (which should work with most backends) that calls dbSendStatement(), then
dbGetRowsAffected(), ensuring that the result is always free-d by dbClearResult().

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbExecute(conn, statement, ...)
```

Arguments

conn	A DBIConnection object, as returned by dbConnect().
statement	a character string containing SQL.
...	Other parameters passed on to methods.

Details

You can also use dbExecute() to call a stored procedure that performs data manipulation or other
actions that do not return a result set. To execute a stored procedure that returns a result set use
dbGetQuery() instead.

Value

dbExecute() always returns a scalar numeric that specifies the number of rows affected by the
statement. An error is raised when issuing a statement over a closed or invalid connection, if the
syntax of the statement is invalid, or if the statement is not a non-NA string.

See Also

For queries: dbSendQuery() and dbGetQuery().

Other DBIConnection generics: DBIConnection-class, dbAppendTable(), dbCreateTable(),
dbDataType(), dbDisconnect(), dbExistsTable(), dbGetException(), dbGetInfo(), dbGetQuery(),
dbIsReadOnly(), dbIsValid(), dbListFields(), dbListObjects(), dbListResults(), dbListTables(),
dbReadTable(), dbRemoveTable(), dbSendQuery(), dbSendStatement(), dbWriteTable()

dbExistsTable,DatabaseConnectorConnection,character-method

Does a table exist?

Description

Returns if a table given by name exists in the database.

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbExistsTable(conn, name, database = NULL, schema = NULL, ...)
```

Arguments

conn	A [DBIConnection](#) object, as returned by [dbConnect()](#).
name	A character string specifying a DBMS table name.
database	Name of the database.
schema	Name of the schema.
...	Other parameters passed on to methods.

Value

dbExistsTable() returns a logical scalar, TRUE if the table or view specified by the name argument exists, FALSE otherwise.

This includes temporary tables if supported by the database.

An error is raised when calling this method for a closed or invalid connection. An error is also raised if name cannot be processed with [dbQuoteIdentifier()](#) or if this results in a non-scalar.

See Also

Other DBIConnection generics: [DBIConnection-class](#), [dbAppendTable()](#), [dbCreateTable()](#), [dbDataType()](#), [dbDisconnect()](#), [dbExecute()](#), [dbGetException()](#), [dbGetInfo()](#), [dbGetQuery()](#), [dbIsReadOnly()](#), [dbIsValid()](#), [dbListFields()](#), [dbListObjects()](#), [dbListResults()](#), [dbListTables()](#), [dbReadTable()](#), [dbRemoveTable()](#), [dbSendQuery()](#), [dbSendStatement()](#), [dbWriteTable()](#)

dbFetch,DatabaseConnectorResult-method

Fetch records from a previously executed query

Description

Fetch the next n elements (rows) from the result set and return them as a data.frame.

Usage

```
## S4 method for signature 'DatabaseConnectorResult'
dbFetch(res, n = -1, ...)
```

Arguments

res	An object inheriting from [DBIResult](), created by [dbSendQuery()]().
n	maximum number of records to retrieve per fetch. Use n = -1 or n = Inf to retrieve all pending records. Some implementations may recognize other special values.
...	Other arguments passed on to methods.

Details

fetch() is provided for compatibility with older DBI clients - for all new code you are strongly encouraged to use dbFetch(). The default implementation for dbFetch() calls fetch() so that it is compatible with existing code. Modern backends should implement for dbFetch() only.

Value

dbFetch() always returns a [data.frame]() with as many rows as records were fetched and as many columns as fields in the result set, even if the result is a single value or has one or zero rows. An attempt to fetch from a closed result set raises an error. If the n argument is not an atomic whole number greater or equal to -1 or Inf, an error is raised, but a subsequent call to dbFetch() with proper n argument succeeds. Calling dbFetch() on a result set from a data manipulation query created by [dbSendStatement()]() can be fetched and return an empty data frame, with a warning.

See Also

Close the result set with [dbClearResult()]() as soon as you finish retrieving the records you want.

Other DBIResult generics: [DBIResult-class](), [dbBind](), [dbClearResult](), [dbColumnInfo](), [dbGetInfo](), [dbGetRowCount](), [dbGetRowsAffected](), [dbGetStatement](), [dbHasCompleted](), [dbIsReadOnly](), [dbIsValid](), [dbQuoteIdentifier](), [dbQuoteLiteral](), [dbQuoteString](), [dbUnquoteIdentifier]()

---

dbGetQuery,DatabaseConnectorConnection,character-method
Send query, retrieve results and then clear result set

Description

Returns the result of a query as a data frame. dbGetQuery() comes with a default implementation (which should work with most backends) that calls [dbSendQuery()](), then [dbFetch()](), ensuring that the result is always free-d by [dbClearResult()]().

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbGetQuery(conn, statement, ...)
```

Arguments

conn	A [DBIConnection]() object, as returned by [dbConnect()]().
statement	a character string containing SQL.
...	Other parameters passed on to methods.

Details

This method is for SELECT queries only (incl. other SQL statements that return a SELECT-alike result, e. g. execution of a stored procedure).

To execute a stored procedure that does not return a result set, use dbExecute().

Some backends may support data manipulation statements through this method for compatibility reasons. However, callers are strongly advised to use dbExecute() for data manipulation statements.

Value

dbGetQuery() always returns a data.frame with as many rows as records were fetched and as many columns as fields in the result set, even if the result is a single value or has one or zero rows. An error is raised when issuing a query over a closed or invalid connection, if the syntax of the query is invalid, or if the query is not a non-NA string. If the n argument is not an atomic whole number greater or equal to -1 or Inf, an error is raised, but a subsequent call to dbGetQuery() with proper n argument succeeds.

See Also

For updates: dbSendStatement() and dbExecute().

Other DBIConnection generics: DBIConnection-class, dbAppendTable(), dbCreateTable(), dbDataType(), dbDisconnect(), dbExecute(), dbExistsTable(), dbGetException(), dbGetInfo(), dbIsReadOnly(), dbIsValid(), dbListFields(), dbListObjects(), dbListResults(), dbListTables(), dbReadTable(), dbRemoveTable(), dbSendQuery(), dbSendStatement(), dbWriteTable()

---

dbGetRowCount,DatabaseConnectorResult-method
The number of rows fetched so far

Description

Returns the total number of rows actually fetched with calls to dbFetch() for this result set.

Usage

```
## S4 method for signature 'DatabaseConnectorResult'
dbGetRowCount(res, ...)
```

Arguments

res	An object inheriting from DBIResult.
...	Other arguments passed on to methods.

Value

dbGetRowCount() returns a scalar number (integer or numeric), the number of rows fetched so far. After calling dbSendQuery(), the row count is initially zero. After a call to dbFetch() without limit, the row count matches the total number of rows returned. Fetching a limited number of rows increases the number of rows by the number of rows returned, even if fetching past the end of the result set. For queries with an empty result set, zero is returned even after fetching. For data manipulation statements issued with dbSendStatement(), zero is returned before and after calling dbFetch(). Attempting to get the row count for a result set cleared with dbClearResult() gives an error.

See Also

Other DBIResult generics: DBIResult-class, dbBind(), dbClearResult(), dbColumnInfo(), dbFetch(), dbGetInfo(), dbGetRowsAffected(), dbGetStatement(), dbHasCompleted(), dbIsReadOnly(), dbIsValid(), dbQuoteIdentifier(), dbQuoteLiteral(), dbQuoteString(), dbUnquoteIdentifier()

---

dbGetRowsAffected,DatabaseConnectorResult-method
The number of rows affected

Description

This method returns the number of rows that were added, deleted, or updated by a data manipulation statement.

Usage

```
## S4 method for signature 'DatabaseConnectorResult'
dbGetRowsAffected(res, ...)
```

Arguments

res	An object inheriting from DBIResult.
...	Other arguments passed on to methods.

Value

dbGetRowsAffected() returns a scalar number (integer or numeric), the number of rows affected by a data manipulation statement issued with dbSendStatement(). The value is available directly after the call and does not change after calling dbFetch(). For queries issued with dbSendQuery(), zero is returned before and after the call to dbFetch(). Attempting to get the rows affected for a result set cleared with dbClearResult() gives an error.

See Also

Other DBIResult generics: DBIResult-class, dbBind(), dbClearResult(), dbColumnInfo(), dbFetch(), dbGetInfo(), dbGetRowCount(), dbGetStatement(), dbHasCompleted(), dbIsReadOnly(), dbIsValid(), dbQuoteIdentifier(), dbQuoteLiteral(), dbQuoteString(), dbUnquoteIdentifier()

dbGetStatement,DatabaseConnectorResult-method
Get the statement associated with a result set

Description

Returns the statement that was passed to dbSendQuery() or dbSendStatement().

Usage

```
## S4 method for signature 'DatabaseConnectorResult'
dbGetStatement(res, ...)
```

Arguments

res   An object inheriting from DBIResult.

...   Other arguments passed on to methods.

Value

dbGetStatement() returns a string, the query used in either dbSendQuery() or dbSendStatement().
Attempting to query the statement for a result set cleared with dbClearResult() gives an error.

See Also

Other DBIResult generics: DBIResult-class, dbBind(), dbClearResult(), dbColumnInfo(),
dbFetch(), dbGetInfo(), dbGetRowCount(), dbGetRowsAffected(), dbHasCompleted(), dbIsReadOnly(),
dbIsValid(), dbQuoteIdentifier(), dbQuoteLiteral(), dbQuoteString(), dbUnquoteIdentifier()

---

dbHasCompleted,DatabaseConnectorResult-method
Completion status

Description

This method returns if the operation has completed. A SELECT query is completed if all rows have
been fetched. A data manipulation statement is always completed.

Usage

```
## S4 method for signature 'DatabaseConnectorResult'
dbHasCompleted(res, ...)
```

Arguments

res   An object inheriting from DBIResult.

...   Other arguments passed on to methods.

Value

dbHasCompleted() returns a logical scalar. For a query initiated by dbSendQuery() with non-empty result set, dbHasCompleted() returns FALSE initially and TRUE after calling dbFetch() without limit. For a query initiated by dbSendStatement(), dbHasCompleted() always returns TRUE. Attempting to query completion status for a result set cleared with dbClearResult() gives an error.

See Also

Other DBIResult generics: DBIResult-class, dbBind(), dbClearResult(), dbColumnInfo(), dbFetch(), dbGetInfo(), dbGetRowCount(), dbGetRowsAffected(), dbGetStatement(), dbIsReadOnly(), dbIsValid(), dbQuoteIdentifier(), dbQuoteLiteral(), dbQuoteString(), dbUnquoteIdentifier()

---

dbIsValid,DatabaseConnectorDbiConnection-method
Is this DBMS object still valid?

Description

This generic tests whether a database object is still valid (i.e. it hasn't been disconnected or cleared).

Usage

```
## S4 method for signature 'DatabaseConnectorDbiConnection'
dbIsValid(dbObj, ...)
```

Arguments

dbObj   An object inheriting from DBIObject, i.e. DBIDriver, DBIConnection, or a
                 DBIResult

...   Other arguments to methods.

Value

dbIsValid() returns a logical scalar, TRUE if the object specified by dbObj is valid, FALSE otherwise. A DBIConnection object is initially valid, and becomes invalid after disconnecting with dbDisconnect(). For an invalid connection object (e.g., for some drivers if the object is saved to a file and then restored), the method also returns FALSE. A DBIResult object is valid after a call to dbSendQuery(), and stays valid even after all rows have been fetched; only clearing it with dbClearResult() invalidates it. A DBIResult object is also valid after a call to dbSendStatement(), and stays valid after querying the number of rows affected; only clearing it with dbClearResult() invalidates it. If the connection to the database system is dropped (e.g., due to connectivity problems, server failure, etc.), dbIsValid() should return FALSE. This is not tested automatically.

See Also

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

Other DBIConnection generics: DBIConnection-class, dbAppendTable(), dbCreateTable(), dbDataType(), dbDisconnect(), dbExecute(), dbExistsTable(), dbGetException(), dbGetInfo(),

dbGetQuery(), dbIsReadOnly(), dbListFields(), dbListObjects(), dbListResults(), dbListTables(), dbReadTable(), dbRemoveTable(), dbSendQuery(), dbSendStatement(), dbWriteTable()

Other DBIResult generics: DBIResult-class, dbBind(), dbClearResult(), dbColumnInfo(), dbFetch(), dbGetInfo(), dbGetRowCount(), dbGetRowsAffected(), dbGetStatement(), dbHasCompleted(), dbIsReadOnly(), dbQuoteIdentifier(), dbQuoteLiteral(), dbQuoteString(), dbUnquoteIdentifier()

---

dbIsValid,DatabaseConnectorJdbcConnection-method
Is this DBMS object still valid?

Description

This generic tests whether a database object is still valid (i.e. it hasn't been disconnected or cleared).

Usage

```
## S4 method for signature 'DatabaseConnectorJdbcConnection'
dbIsValid(dbObj, ...)
```

Arguments

dbObj   An object inheriting from DBIObject, i.e. DBIDriver, DBIConnection, or a
                DBIResult

...   Other arguments to methods.

Value

dbIsValid() returns a logical scalar, TRUE if the object specified by dbObj is valid, FALSE otherwise. A DBIConnection object is initially valid, and becomes invalid after disconnecting with dbDisconnect(). For an invalid connection object (e.g., for some drivers if the object is saved to a file and then restored), the method also returns FALSE. A DBIResult object is valid after a call to dbSendQuery(), and stays valid even after all rows have been fetched; only clearing it with dbClearResult() invalidates it. A DBIResult object is also valid after a call to dbSendStatement(), and stays valid after querying the number of rows affected; only clearing it with dbClearResult() invalidates it. If the connection to the database system is dropped (e.g., due to connectivity problems, server failure, etc.), dbIsValid() should return FALSE. This is not tested automatically.

See Also

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

Other DBIConnection generics: DBIConnection-class, dbAppendTable(), dbCreateTable(), dbDataType(), dbDisconnect(), dbExecute(), dbExistsTable(), dbGetException(), dbGetInfo(), dbGetQuery(), dbIsReadOnly(), dbListFields(), dbListObjects(), dbListResults(), dbListTables(), dbReadTable(), dbRemoveTable(), dbSendQuery(), dbSendStatement(), dbWriteTable()

Other DBIResult generics: DBIResult-class, dbBind(), dbClearResult(), dbColumnInfo(), dbFetch(), dbGetInfo(), dbGetRowCount(), dbGetRowsAffected(), dbGetStatement(), dbHasCompleted(), dbIsReadOnly(), dbQuoteIdentifier(), dbQuoteLiteral(), dbQuoteString(), dbUnquoteIdentifier()

dbListFields,DatabaseConnectorConnection,character-method
List field names of a remote table

Description

List field names of a remote table

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbListFields(conn, name, database = NULL, schema = NULL, ...)
```

Arguments

conn	A [DBIConnection](#) object, as returned by [dbConnect()](#).
name	a character string with the name of the remote table.
database	Name of the database.
schema	Name of the schema.
...	Other parameters passed on to methods.

Value

dbListFields() returns a character vector that enumerates all fields in the table in the correct order. This also works for temporary tables if supported by the database. The returned names are suitable for quoting with dbQuoteIdentifier(). If the table does not exist, an error is raised. Invalid types for the name argument (e.g., character of length not equal to one, or numeric) lead to an error. An error is also raised when calling this method for a closed or invalid connection.

See Also

[dbColumnInfo()](#) to get the type of the fields.

Other DBIConnection generics: [DBIConnection-class](#), [dbAppendTable()](#), [dbCreateTable()](#), [dbDataType()](#), [dbDisconnect()](#), [dbExecute()](#), [dbExistsTable()](#), [dbGetException()](#), [dbGetInfo()](#), [dbGetQuery()](#), [dbIsReadOnly()](#), [dbIsValid()](#), [dbListObjects()](#), [dbListResults()](#), [dbListTables()](#), [dbReadTable()](#), [dbRemoveTable()](#), [dbSendQuery()](#), [dbSendStatement()](#), [dbWriteTable()](#)

dbListTables,DatabaseConnectorConnection-method
List remote tables

Description

Returns the unquoted names of remote tables accessible through this connection. This should include views and temporary objects, but not all database backends (in particular **RMariaDB** and **RMySQL**) support this.

Usage

```
## S4 method for signature 'DatabaseConnectorConnection'
dbListTables(conn, database = NULL, schema = NULL, ...)
```

Arguments

conn	A DBIConnection object, as returned by dbConnect().
database	Name of the database.
schema	Name of the schema.
...	Other parameters passed on to methods.

Value

dbListTables() returns a character vector that enumerates all tables and views in the database. Tables added with dbWriteTable() are part of the list. As soon a table is removed from the database, it is also removed from the list of database tables.

The same applies to temporary tables if supported by the database.

The returned names are suitable for quoting with dbQuoteIdentifier(). An error is raised when calling this method for a closed or invalid connection.

See Also

Other DBIConnection generics: DBIConnection-class, dbAppendTable(), dbCreateTable(), dbDataType(), dbDisconnect(), dbExecute(), dbExistsTable(), dbGetException(), dbGetInfo(), dbGetQuery(), dbIsReadOnly(), dbIsValid(), dbListFields(), dbListObjects(), dbListResults(), dbReadTable(), dbRemoveTable(), dbSendQuery(), dbSendStatement(), dbWriteTable()

---

dbQuoteIdentifier,DatabaseConnectorConnection,character-method
Quote identifiers

Description

Call this method to generate a string that is suitable for use in a query as a column or table name, to make sure that you generate valid SQL and protect against SQL injection attacks. The inverse operation is dbUnquoteIdentifier().

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbQuoteIdentifier(conn, x, ...)
```

Arguments

conn	A DBIConnection object, as returned by dbConnect().
x	A character vector, SQL or Id object to quote as identifier.
...	Other arguments passed on to methods.

Value

dbQuoteIdentifier() returns an object that can be coerced to character, of the same length as
the input. For an empty character vector this function returns a length-0 object. The names
of the input argument are preserved in the output. When passing the returned object again to
dbQuoteIdentifier() as x argument, it is returned unchanged. Passing objects of class SQL
should also return them unchanged. (For backends it may be most convenient to return SQL objects
to achieve this behavior, but this is not required.)

An error is raised if the input contains NA, but not for an empty string.

See Also

Other DBIResult generics: DBIResult-class, dbBind(), dbClearResult(), dbColumnInfo(),
dbFetch(), dbGetInfo(), dbGetRowCount(), dbGetRowsAffected(), dbGetStatement(), dbHasCompleted(),
dbIsReadOnly(), dbIsValid(), dbQuoteLiteral(), dbQuoteString(), dbUnquoteIdentifier()

---

dbQuoteString,DatabaseConnectorConnection,character-method
Quote literal strings

Description

Call this method to generate a string that is suitable for use in a query as a string literal, to make
sure that you generate valid SQL and protect against SQL injection attacks.

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbQuoteString(conn, x, ...)
```

Arguments

conn	A DBIConnection object, as returned by dbConnect().
x	A character vector to quote as string.
...	Other arguments passed on to methods.

Value

dbQuoteString() returns an object that can be coerced to character, of the same length as the input.
For an empty character vector this function returns a length-0 object.

When passing the returned object again to dbQuoteString() as x argument, it is returned un-
changed. Passing objects of class SQL should also return them unchanged. (For backends it may
be most convenient to return SQL objects to achieve this behavior, but this is not required.)

See Also

Other DBIResult generics: DBIResult-class, dbBind(), dbClearResult(), dbColumnInfo(),
dbFetch(), dbGetInfo(), dbGetRowCount(), dbGetRowsAffected(), dbGetStatement(), dbHasCompleted(),
dbIsReadOnly(), dbIsValid(), dbQuoteIdentifier(), dbQuoteLiteral(), dbUnquoteIdentifier()

dbReadTable,DatabaseConnectorConnection,character-method
Copy data frames from database tables

Description

Reads a database table to a data frame, optionally converting a column to row names and converting the column names to valid R identifiers.

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbReadTable(
  conn,
  name,
  database = NULL,
  schema = NULL,
  oracleTempSchema = NULL,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  ...
)
```

Arguments

conn	A [DBIConnection](#) object, as returned by [dbConnect()](#).
name	A character string specifying the unquoted DBMS table name, or the result of a call to [dbQuoteIdentifier()](#).
database	Name of the database.
schema	Name of the schema.
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.
...	Other parameters passed on to methods.

Value

dbReadTable() returns a data frame that contains the complete data from the remote table, effectively the result of calling [dbGetQuery()](#) with SELECT * FROM <name>. An error is raised if the table does not exist. An empty table is returned as a data frame with zero rows.

The presence of [rownames](#) depends on the row.names argument, see [sqlColumnToRownames()](#) for details:

- If FALSE or NULL, the returned data frame doesn't have row names.

- If TRUE, a column named "row_names" is converted to row names, an error is raised if no such column exists.

- If NA, a column named "row_names" is converted to row names if it exists, otherwise no translation occurs.
- If a string, this specifies the name of the column in the remote table that contains the row names, an error is raised if no such column exists.

The default is row.names = FALSE.

If the database supports identifiers with special characters, the columns in the returned data frame are converted to valid R identifiers if the check.names argument is TRUE, If check.names = FALSE, the returned table has non-syntactic column names without quotes.

An error is raised when calling this method for a closed or invalid connection. An error is raised if name cannot be processed with [dbQuoteIdentifier()](#) or if this results in a non-scalar. Unsupported values for row.names and check.names (non-scalars, unsupported data types, NA for check.names) also raise an error.

See Also

Other DBIConnection generics: [DBIConnection-class](#), [dbAppendTable()](#), [dbCreateTable()](#), [dbDataType()](#), [dbDisconnect()](#), [dbExecute()](#), [dbExistsTable()](#), [dbGetException()](#), [dbGetInfo()](#), [dbGetQuery()](#), [dbIsReadOnly()](#), [dbIsValid()](#), [dbListFields()](#), [dbListObjects()](#), [dbListResults()](#), [dbListTables()](#), [dbRemoveTable()](#), [dbSendQuery()](#), [dbSendStatement()](#), [dbWriteTable()](#)

---

dbRemoveTable,DatabaseConnectorConnection,character-method
            Remove a table from the database

Description

Remove a remote table (e.g., created by [dbWriteTable()](#)) from the database.

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbRemoveTable(
  conn,
  name,
  database = NULL,
  schema = NULL,
  oracleTempSchema = NULL,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  ...
)
```

Arguments

conn	A [DBIConnection](#) object, as returned by [dbConnect()](#).
name	A character string specifying a DBMS table name.
database	Name of the database.
schema	Name of the schema.
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.

... Other parameters passed on to methods.

Value

dbRemoveTable() returns TRUE, invisibly. If the table does not exist, an error is raised. An attempt to remove a view with this function may result in an error.

An error is raised when calling this method for a closed or invalid connection. An error is also raised if name cannot be processed with dbQuoteIdentifier() or if this results in a non-scalar.

See Also

Other DBIConnection generics: DBIConnection-class, dbAppendTable(), dbCreateTable(), dbDataType(), dbDisconnect(), dbExecute(), dbExistsTable(), dbGetException(), dbGetInfo(), dbGetQuery(), dbIsReadOnly(), dbIsValid(), dbListFields(), dbListObjects(), dbListResults(), dbListTables(), dbReadTable(), dbSendQuery(), dbSendStatement(), dbWriteTable()

---

dbSendQuery,DatabaseConnectorDbiConnection,character-method
Execute a query on a given database connection

Description

The dbSendQuery() method only submits and synchronously executes the SQL query to the database engine. It does *not* extract any records — for that you need to use the dbFetch() method, and then you must call dbClearResult() when you finish fetching the records you need. For interactive use, you should almost always prefer dbGetQuery().

Usage

```
## S4 method for signature 'DatabaseConnectorDbiConnection,character'
dbSendQuery(conn, statement, ...)
```

Arguments

conn   A DBIConnection object, as returned by dbConnect().
statement   a character string containing SQL.
...   Other parameters passed on to methods.

Details

This method is for SELECT queries only. Some backends may support data manipulation queries through this method for compatibility reasons. However, callers are strongly encouraged to use dbSendStatement() for data manipulation statements.

The query is submitted to the database server and the DBMS executes it, possibly generating vast amounts of data. Where these data live is driver-specific: some drivers may choose to leave the output on the server and transfer them piecemeal to R, others may transfer all the data to the client – but not necessarily to the memory that R manages. See individual drivers' dbSendQuery() documentation for details.

Value

dbSendQuery() returns an S4 object that inherits from DBIResult. The result set can be used with dbFetch() to extract records. Once you have finished using a result, make sure to clear it with dbClearResult(). An error is raised when issuing a query over a closed or invalid connection, or if the query is not a non-NA string. An error is also raised if the syntax of the query is invalid and all query parameters are given (by passing the params argument) or the immediate argument is set to TRUE.

See Also

For updates: dbSendStatement() and dbExecute().

Other DBIConnection generics: DBIConnection-class, dbAppendTable(), dbCreateTable(), dbDataType(), dbDisconnect(), dbExecute(), dbExistsTable(), dbGetException(), dbGetInfo(), dbGetQuery(), dbIsReadOnly(), dbIsValid(), dbListFields(), dbListObjects(), dbListResults(), dbListTables(), dbReadTable(), dbRemoveTable(), dbSendStatement(), dbWriteTable()

---

dbSendQuery,DatabaseConnectorJdbcConnection,character-method
Execute a query on a given database connection

Description

The dbSendQuery() method only submits and synchronously executes the SQL query to the database engine. It does *not* extract any records — for that you need to use the dbFetch() method, and then you must call dbClearResult() when you finish fetching the records you need. For interactive use, you should almost always prefer dbGetQuery().

Usage

```
## S4 method for signature 'DatabaseConnectorJdbcConnection,character'
dbSendQuery(conn, statement, ...)
```

Arguments

conn	A DBIConnection object, as returned by dbConnect().
statement	a character string containing SQL.
...	Other parameters passed on to methods.

Details

This method is for SELECT queries only. Some backends may support data manipulation queries through this method for compatibility reasons. However, callers are strongly encouraged to use dbSendStatement() for data manipulation statements.

The query is submitted to the database server and the DBMS executes it, possibly generating vast amounts of data. Where these data live is driver-specific: some drivers may choose to leave the output on the server and transfer them piecemeal to R, others may transfer all the data to the client – but not necessarily to the memory that R manages. See individual drivers' dbSendQuery() documentation for details.

Value

dbSendQuery() returns an S4 object that inherits from [DBIResult](#). The result set can be used with [dbFetch()](#) to extract records. Once you have finished using a result, make sure to clear it with [dbClearResult()](#). An error is raised when issuing a query over a closed or invalid connection, or if the query is not a non-NA string. An error is also raised if the syntax of the query is invalid and all query parameters are given (by passing the params argument) or the immediate argument is set to TRUE.

See Also

For updates: [dbSendStatement()](#) and [dbExecute()](#).

Other DBIConnection generics: [DBIConnection-class](#), [dbAppendTable()](#), [dbCreateTable()](#), [dbDataType()](#), [dbDisconnect()](#), [dbExecute()](#), [dbExistsTable()](#), [dbGetException()](#), [dbGetInfo()](#), [dbGetQuery()](#), [dbIsReadOnly()](#), [dbIsValid()](#), [dbListFields()](#), [dbListObjects()](#), [dbListResults()](#), [dbListTables()](#), [dbReadTable()](#), [dbRemoveTable()](#), [dbSendStatement()](#), [dbWriteTable()](#)

---

dbSendStatement,DatabaseConnectorConnection,character-method
Execute a data manipulation statement on a given database connection

---

Description

The dbSendStatement() method only submits and synchronously executes the SQL data manipulation statement (e.g., UPDATE, DELETE, INSERT INTO, DROP TABLE, ...) to the database engine. To query the number of affected rows, call [dbGetRowsAffected()](#) on the returned result object. You must also call [dbClearResult()](#) after that. For interactive use, you should almost always prefer [dbExecute()](#).

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbSendStatement(conn, statement, ...)
```

Arguments

conn	A [DBIConnection](#) object, as returned by [dbConnect()](#).
statement	a character string containing SQL.
...	Other parameters passed on to methods.

Details

[dbSendStatement()](#) comes with a default implementation that simply forwards to [dbSendQuery()](#), to support backends that only implement the latter.

Value

dbSendStatement() returns an S4 object that inherits from [DBIResult](). The result set can be used
with [dbGetRowsAffected()]() to determine the number of rows affected by the query. Once you
have finished using a result, make sure to clear it with [dbClearResult()](). An error is raised when
issuing a statement over a closed or invalid connection, or if the statement is not a non-NA string.
An error is also raised if the syntax of the query is invalid and all query parameters are given (by
passing the params argument) or the immediate argument is set to TRUE.

See Also

For queries: [dbSendQuery()]() and [dbGetQuery()]().

Other DBIConnection generics: [DBIConnection-class](), [dbAppendTable](), [dbCreateTable](),
[dbDataType](), [dbDisconnect](), [dbExecute](), [dbExistsTable](), [dbGetException](), [dbGetInfo](),
[dbGetQuery](), [dbIsReadOnly](), [dbIsValid](), [dbListFields](), [dbListObjects](), [dbListResults](),
[dbListTables](), [dbReadTable](), [dbRemoveTable](), [dbSendQuery](), [dbWriteTable]()

---

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 an 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

```
## 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()

---

dbWriteTable,DatabaseConnectorConnection,character,data.frame-method
Copy data frames to database tables

Description

Writes, overwrites or appends a data frame to a database table, optionally converting row names to a column and specifying SQL data types for fields. New code should prefer dbCreateTable() and dbAppendTable().

Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character,data.frame'
dbWriteTable(
  conn,
  name,
  value,
  overwrite = FALSE,
  append = FALSE,
  temporary = FALSE,
  oracleTempSchema = NULL,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  ...
)
```

Arguments

conn	A DBIConnection object, as returned by dbConnect().
name	A character string specifying the unquoted DBMS table name, or the result of a call to dbQuoteIdentifier().
value	a data.frame (or coercible to data.frame).
overwrite	Overwrite an existing table (if exists)?
append	Append to existing table?
temporary	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.

...   Other parameters passed on to methods.

Value

dbWriteTable() returns TRUE, invisibly. If the table exists, and both append and overwrite arguments are unset, or append = TRUE and the data frame with the new data has different column names, an error is raised; the remote table remains unchanged.

An error is raised when calling this method for a closed or invalid connection. An error is also raised if name cannot be processed with [dbQuoteIdentifier()](#) or if this results in a non-scalar. Invalid values for the additional arguments row.names, overwrite, append, field.types, and temporary (non-scalars, unsupported data types, NA, incompatible values, duplicate or missing names, incompatible columns) also raise an error.

See Also

Other DBIConnection generics: [DBIConnection-class](#), [dbAppendTable()](#), [dbCreateTable()](#), [dbDataType()](#), [dbDisconnect()](#), [dbExecute()](#), [dbExistsTable()](#), [dbGetException()](#), [dbGetInfo()](#), [dbGetQuery()](#), [dbIsReadOnly()](#), [dbIsValid()](#), [dbListFields()](#), [dbListObjects()](#), [dbListResults()](#), [dbListTables()](#), [dbReadTable()](#), [dbRemoveTable()](#), [dbSendQuery()](#), [dbSendStatement()](#)

---

disconnect   *Disconnect from the server*

Description

Close the connection to the server.

Usage

```
disconnect(connection)
```

Arguments

connection   The connection to the database server.

Examples

```
## 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" or "pdw" for Microsoft SQL Server
	• "oracle" for Oracle
	• "spark" for Spark
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
- RedShiftV1.2.27.1051
- SQL ServerV8.4.1.zip
- OracleV19.8
- SparkV2.6.21

Value

Invisibly returns the destination if the download was successful.

Examples

```
## 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

```
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.

Value

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

```
## 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

```
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

```
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).

SQL Server, Oracle, PostgreSQL, PDW, Spark, RedShift

Use the [downloadJdbcDrivers](#) function to download these drivers from the OHDSI GitHub pages.

Netezza

Read the instructions here on how to obtain the Netezza JDBC driver.

BigQuery

Go to Google's site and download the latest JDBC driver. Unzip the file, and locate the appropriate jar files.

Impala

Go to Cloudera's site, pick your OS version, and click "GET IT NOW!'. Register, and you should be able to download the driver.

SQLite

For SQLite we actually don't use a JDBC driver. Instead, we use the RSQLite package, which can be installed using install.packages("RSQLite").

---

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,
  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, our 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

lowLevelQuerySqlToAndromeda

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

```
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, our 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 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 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

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.

Value

A data frame.

Examples

```
## 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

```
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.
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

```
## 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

```
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

```
## 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, ...) {
  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)

## End(Not run)
```

---

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

```
renderTranslateQuerySql(
  connection,
  sql,
  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.

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`.

...   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",
  schema = "cdm_v4"
)
conn <- connect(connectionDetails)
persons <- renderTranslatequerySql(conn,
  sql = "SELECT TOP 10 * FROM @schema.person",
  schema = "cdm_synpuf"
)
disconnect(conn)

## End(Not run)
```

---

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`.
...	Parameters that will be used to render the SQL.

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

```
## 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,
  andromedaTableName = "foo"
)
disconnect(conn)

andromeda$foo

## End(Not run)
```

---

show,DatabaseConnectorConnection-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 'DatabaseConnectorConnection'
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.

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