

# Package ‘DatabaseConnector’

February 21, 2019

**Type** Package

**Title** Connecting to Various Database Platforms

**Version** 2.3.0

**Date** 2019-02-21

**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', and 'SQLite'). Also includes support for fetching data as 'ffdf' objects. Uses 'Java Database Connectivity' ('JDBC') to connect to databases (except SQLite).

**Imports** DatabaseConnectorJars,

rJava,  
bit,  
ff,  
ffbase (>= 0.12.1),  
SqlRender (>= 1.6.0),  
methods,  
utils,  
DBI (>= 1.0.0),  
urltools

**Suggests** aws.s3,

uuid,  
R.utils,  
testthat,  
DBITest,  
knitr,  
rmarkdown,  
RSQLite

**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** 6.1.1

**R topics documented:**

connect	3
createConnectionDetails	6
createZipFile	9
DatabaseConnector	10
DatabaseConnectorDriver	10
dbAppendTable,DatabaseConnectorConnection,character,data.frame-method	10
dbClearResult,DatabaseConnectorResult-method	11
dbColumnInfo,DatabaseConnectorResult-method	12
dbConnect,DatabaseConnectorDriver-method	13
dbCreateTable,DatabaseConnectorConnection,character,data.frame-method	13
dbDisconnect,DatabaseConnectorConnection-method	14
dbExecute,DatabaseConnectorConnection,character-method	15
dbExistsTable,DatabaseConnectorConnection,character-method	16
dbFetch,DatabaseConnectorResult-method	17
dbGetQuery,DatabaseConnectorConnection,character-method	18
dbGetRowCount,DatabaseConnectorResult-method	19
dbGetRowsAffected,DatabaseConnectorResult-method	19
dbGetStatement,DatabaseConnectorResult-method	20
dbHasCompleted,DatabaseConnectorResult-method	21
dbIsValid,DatabaseConnectorDbiConnection-method	21
dbIsValid,DatabaseConnectorJdbcConnection-method	22
dbListFields,DatabaseConnectorConnection,character-method	23
dbListTables,DatabaseConnectorConnection-method	24
dbQuoteIdentifier,DatabaseConnectorConnection,character-method	25
dbQuoteString,DatabaseConnectorConnection,character-method	25
dbReadTable,DatabaseConnectorConnection,character-method	26
dbRemoveTable,DatabaseConnectorConnection,character-method	27
dbSendQuery,DatabaseConnectorDbiConnection,character-method	28
dbSendQuery,DatabaseConnectorJdbcConnection,character-method	29
dbSendStatement,DatabaseConnectorConnection,character-method	30
dbUnloadDriver,DatabaseConnectorDriver-method	31
dbWriteTable,DatabaseConnectorConnection,character,data.frame-method	32
disconnect	33
executeSql	34
getTableNames	35
insertTable	35
jdbcDrivers	37
lowLevelExecuteSql	38
lowLevelQuerySql	38
lowLevelQuerySql.ffdf	39
querySql	39
querySql.ffdf	40
renderTranslateExecuteSql	41
renderTranslateQuerySql	42
renderTranslateQuerySql.ffdf	43
show,DatabaseConnectorConnection-method	44
show,DatabaseConnectorDriver-method	45

---

connect	<i>connect</i>
---------	----------------

---

## Description

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

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

## Arguments

connectionDetails	An object of class <code>connectionDetails</code> as created by the <a href="#">createConnectionDetails</a> function.
dbms	The type of DBMS running on the server. Valid values are <ul style="list-style-type: none"> <li>• "oracle" for Oracle</li> <li>• "postgresql" for PostgreSQL</li> <li>• "redshift" for Amazon Redshift</li> <li>• "sql server" for Microsoft SQL Server</li> <li>• "pdw" for Microsoft Parallel Data Warehouse (PDW)</li> <li>• "netezza" for IBM Netezza</li> <li>• "bigquery" for Google BigQuery</li> <li>• "sqlite" for SQLite</li> </ul>
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.
schema	(optional) The name of the schema to connect to.
extraSettings	(optional) Additional configuration settings specific to the database provider to configure things as security for SSL. These must follow the format for the JDBC connection for the RDBMS specified in dbms.
oracleDriver	Specify which Oracle drive you want to use. Choose between "thin" or "oci".
connectionString	The JDBC connection string. If specified, the server, port, extraSettings, and oracleDriver fields are ignored. If user and password are not specified, they are assumed to already be included in the connection string.
pathToDriver	Path to the JDBC driver JAR files. Currently only needed for BigQuery, Impala and Netezza. See <a href="#">jdbcDrivers</a> for details on how to get the drivers.

## Details

This function creates a connection to a database.

**Value**

An object that extends `DBConnection` 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)
- `schema`. This field contains the schema (i.e. 'user' in Oracle terms) containing the tables
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "(`PROTOCOL=tcps`)")
- `oracleDriver` The driver to be used. Choose between "thin" or "oci".

Microsoft SQL Server:

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

Microsoft PDW:

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

PostgreSQL:

- `user`. The user used to log in to the server
- `password`. The password used to log on to the server

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

**Redshift:**

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

**Netezza:**

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

**Impala:**

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

**SQLite:**

- `server`. The path to the SQLite file

To be able to use Windows authentication for SQL Server (and PDW), you have to install the JDBC driver. Download the .exe from [Microsoft](#) and run it, thereby extracting its contents to a folder. In the extracted folder you will find the file `sqljdbc_4.0/enu/auth/x64/sqljdbc_auth.dll` (64-bits) or `sqljdbc_4.0/enu/auth/x86/sqljdbc_auth.dll` (32-bits), which needs to be moved to location on the system path, for example to `c:/windows/system32`.

**Examples**

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

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

conn <- connect(dbms = "oracle",
               server = "127.0.0.1/xe",
               user = "system",
               password = "xxx",
               schema = "test",
               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, schema, extraSettings, oracle)
- 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)

	<ul style="list-style-type: none"> <li>• "netezza" for IBM Netezza</li> <li>• "bigquery" for Google BigQuery</li> <li>• "sqlite" for SQLite</li> </ul>
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.
schema	(optional) The name of the schema to connect to.
extraSettings	(optional) Additional configuration settings specific to the database provider to configure things as security for SSL. These must follow the format for the JDBC connection for the RDBMS specified in dbms.
oracleDriver	Specify which Oracle drive you want to use. Choose between "thin" or "oci".
connectionString	The JDBC connection string. If specified, the server, port, extraSettings, and oracleDriver fields are ignored. If user and password are not specified, they are assumed to already be included in the connection string.
pathToDriver	Path to the JDBC driver JAR files. Currently only needed for BigQuery, Impala and Netezza. See <a href="#">jdbcDrivers</a> for details on how to get the 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)
- schema. This field contains the schema (i.e. 'user' in Oracle terms) containing the tables
- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "(PROTOCOL=tcp)")
- oracleDriver The driver to be used. Choose between "thin" or "oci".

Microsoft SQL Server:

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

- `server`. This field contains the host name of the server
- `port`. Not used for SQL Server
- `schema`. The database containing the tables. If both database and schema are specified (e.g. 'my\_database.dbo', then only the database part is used, the schema is ignored.
- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "encrypt=true; trustServerCertificate=false;")

#### Microsoft PDW:

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

#### PostgreSQL:

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

#### Redshift:

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

#### Netezza:

- `user`. The user used to log in to the server
- `password`. The password used to log on to the server
- `server`. This field contains the host name of the server and the database holding the relevant schemas: <host>/<database>
- `port`. Specifies the port on the server (default = 5480)
- `schema`. The schema containing the tables.



- `extraSettings` The configuration settings for the connection (i.e. SSL Settings such as "ssl=true")
- `pathToDriver` The path to the folder containing the Netezza JDBC driver JAR file (nzjdbc.jar).

#### Impala:

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

#### SQLite:

- `server`. The path to the SQLite file

To be able to use Windows authentication for SQL Server (and PDW), you have to install the JDBC driver. Download the .exe from [Microsoft](#) and run it, thereby extracting its contents to a folder. In the extracted folder you will find the file `sqljdbc_4.0/enu/auth/x64/sqljdbc_auth.dll` (64-bits) or `sqljdbc_4.0/enu/auth/x86/sqljdbc_auth.dll` (32-bits), which needs to be moved to location on the system path, for example to `c:/windows/system32`.

### Examples

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

conn <- connect(connectionDetails)
dbGetQuery(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

## End(Not run)
```

---

<code>createZipFile</code>	<i>Compress files and/or folders into a single zip file</i>
----------------------------	---

---

### 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.

---

DatabaseConnector	<i>DatabaseConnector</i>
-------------------	--------------------------

---

**Description**

DatabaseConnector

---

DatabaseConnectorDriver	<i>Create a DatabaseConnectorDriver object</i>
-------------------------	--

---

**Description**

Create a DatabaseConnectorDriver object

**Usage**

DatabaseConnectorDriver()

---

dbAppendTable, DatabaseConnectorConnection, character, data.frame-method	<i>Insert rows into a table</i>
--	---------------------------------

---

**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, ...,
  row.names = NULL)
```

**Arguments**

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
name	Name of the table, escaped with <a href="#">dbQuoteIdentifier()</a> .
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	Specifically for Oracle, 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 <a href="#">DBIResult</a> .
...	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 <a href="#">DBIResult</a> .
...	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 <a href="#">connect</a> 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, ..., row.names = NULL,
  temporary = FALSE)
```

**Arguments**

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
name	Name of the table, escaped with <a href="#">dbQuoteIdentifier()</a> .
fields	Either a character vector or a data frame. A named character vector: Names are column names, values are types. Names are escaped with <a href="#">dbQuoteIdentifier()</a> . Field types are unescaped. A data frame: field types are generated using <a href="#">dbDataType()</a> .
oracleTempSchema	Specifically for Oracle, 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 be 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 default 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**

<code>conn</code>	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
<code>name</code>	A character string specifying a DBMS table name.
<code>database</code>	Name of the database.
<code>schema</code>	Name of the schema.
<code>...</code>	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, datesAsString = FALSE,  
        ...)
```

## Arguments

<code>res</code>	An object inheriting from <a href="#">DBIResult</a> , created by <a href="#">dbSendQuery()</a> .
<code>datesAsString</code>	Should dates be represented as strings? (instead of Date objects)
<code>...</code>	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

<code>conn</code>	A <a href="#">DBIConnection</a> object, as returned by <code>dbConnect()</code> .
<code>statement</code>	a character string containing SQL.
<code>...</code>	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 <a href="#">DBIResult</a> .
...	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 <a href="#">DBIObject</a> , i.e. <a href="#">DBIDriver</a> , <a href="#">DBIConnection</a> , or a <a href="#">DBIResult</a>
...	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

<code>conn</code>	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
<code>name</code>	a character string with the name of the remote table.
<code>database</code>	Name of the database.
<code>schema</code>	Name of the schema.
<code>...</code>	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 <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
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, including temporary tables if supported by the database. As soon a table is removed from the database, it is also removed from the list of database tables.

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 subclass of <a href="#">DBIConnection</a> , representing an active connection to an DBMS.
x	A character vector, <a href="#">SQL</a> or <a href="#">Id</a> 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 subclass of <a href="#">DBIConnection</a> , representing an active connection to an DBMS.
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 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.)

**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, ...)
```

**Arguments**

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
name	A character string specifying the unquoted DBMS table name, or the result of a call to <a href="#">dbQuoteIdentifier()</a> .
database	Name of the database.
schema	Name of the schema.
oracleTempSchema	Specifically for Oracle, 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, otherwise non-syntactic column names can be returned unquoted.

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, ...)
```

**Arguments**

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
name	A character string specifying a DBMS table name.
database	Name of the database.
schema	Name of the schema.
oracleTempSchema	Specifically for Oracle, 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 <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
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, if the syntax of the query is invalid, or if the query is not a non-NA string.

**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 <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
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, if the syntax of the query is invalid, or if the query is not a non-NA string.

**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 <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
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, 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`, `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 a new driver object given the name of a database or the corresponding R package. It works through convention: all DBI-extending packages should provide an exported object with the same name as the package. `dbDriver()` just looks for this object in the right places: if you know what database you are connecting to, you should call the function directly.

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

**Usage**

```
## 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.

**Usage**

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

**Arguments**

<code>conn</code>	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
<code>name</code>	A character string specifying the unquoted DBMS table name, or the result of a call to <a href="#">dbQuoteIdentifier()</a> .
<code>value</code>	a <a href="#">data.frame</a> (or coercible to <code>data.frame</code> ).
<code>overwrite</code>	Overwrite an existing table (if exists)?
<code>append</code>	Append to existing table?
<code>temporary</code>	Should the table created as a temp table?
<code>oracleTempSchema</code>	Specifically for Oracle, a schema with write privileges where temp tables can be created.
<code>...</code>	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**

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

**Usage**

```
disconnect(connection)
```

**Arguments**

connection      The connection to the database server.

**Examples**

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

conn <- connect(connectionDetails)
count <- querySql.ffdf(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

## End(Not run)
```

executeSql

*Execute SQL code***Description**

This function executes SQL consisting of one or more statements.

**Usage**

```
executeSql(connection, sql, profile = FALSE, progressBar = TRUE,
  reportOverallTime = TRUE, errorReportFile = file.path(getwd(),
  "errorReport.txt"))
```

**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 'errorReport.txt' in the current working directory.

**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)
```

---

getTableNames	<i>List all tables in a database schema.</i>
---------------	--

---

### Description

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

### Usage

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

### Arguments

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

### Details

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

### Value

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

---

insertTable	<i>Insert a table on the server</i>
-------------	-------------------------------------

---

### Description

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

### Usage

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

**Arguments**

connection	The connection to the database server.
tableName	The name of the table where the data should be inserted.
data	The data frame or fdf containing the data to be inserted.
dropTableIfExists	Drop the table if the table already exists before writing?
createTable	Create a new table? If false, will append to existing table.
tempTable	Should the table created as a temp table?
oracleTempSchema	Specifically for Oracle, a schema with write privileges where temp tables can be created.
useMppBulkLoad	If using Redshift or PDW, use more performant bulk loading techniques. Setting the system environment variable "USE_MPP_BULK_LOAD" to TRUE is another way to enable this mode. Please note, Redshift requires valid S3 credentials; PDW requires valid DWLoader installation. This can only be used for permanent tables, and cannot be used to append to an existing table.
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. If using Redshift or PDW, bulk uploading techniques may be more performant than relying upon a batch of insert statements, depending upon data size and network throughput. 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 either 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.

**Examples**

```
## Not run:
connectionDetails <- createConnectionDetails(dbms = "mysql",
                                             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(conn, "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,
            tableName = "scratch.somedata",
            data = data,
            dropTableIfExists = TRUE,
            createTable = TRUE,
            tempTable = FALSE,
            useMppBulkLoad = TRUE) # or, Sys.setenv('USE_MPP_BULK_LOAD' = TRUE)

## End(Not run)
```

---

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.

## PostgresSql

Go to [the PostgreSQL JDBC site](#) and download the current version. The file is called something like 'postgresql-42.2.2.jar'.

## Oracle

Go to [the Oracle JDBC site](#). Select 'Accept License Agreement' and download the jar file. The file is called something like 'ojdbc7.jar'.

## SQL Server and PDW

Go to [the Microsoft SQL Server JDBC site](#), click 'Download' and select the tar.gz file. Click 'Next' to start the download. Decompress the file and find a file called something like 'sqljdbc41.jar' in the a folder named something like 'sqljdbc\_6.0/enu/jre7'.

## RedShift

Go to the [Amazon RedShift JDBC driver page](#) and download the latest JDBC driver. The file is called something like 'RedshiftJDBC42-1.2.12.1017.jar'.

## 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.

---

lowLevelExecuteSql     *Execute SQL code*

---

**Description**

This function executes a single SQL statement.

**Usage**

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

**Arguments**

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

---

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

---

**Description**

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

**Usage**

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

**Arguments**

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

**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

---

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

---

### Description

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

### Usage

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

### Arguments

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

### Details

Retrieves data from the database server and stores it in an ffd 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.

### Value

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

---

<code>querySql</code>	<i>Retrieve data to a data.frame</i>
-----------------------	--------------------------------------

---

### Description

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

### Usage

```
querySql(connection, sql, errorReportFile = file.path(getwd(),
  "errorReport.txt"), snakeCaseToCamelCase = FALSE)
```

### Arguments

<code>connection</code>	The connection to the database server.
<code>sql</code>	The SQL to be send.
<code>errorReportFile</code>	The file where an error report will be written if an error occurs. Defaults to 'errorReport.txt' in the current working directory.
<code>snakeCaseToCamelCase</code>	If true, field names are assumed to use snake_case, and are converted to camel-Case.

**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)
```

---

querySql.ffdf

*Retrieves data to an ffdf object*


---

**Description**

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

**Usage**

```
querySql.ffdf(connection, sql, errorReportFile = file.path(getwd(),
  "errorReport.txt"), snakeCaseToCamelCase = FALSE)
```

**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 'errorReport.txt' in the current working directory.
snakeCaseToCamelCase	If true, field names are assumed to use snake_case, and are converted to camel-Case.

**Details**

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



**Value**

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

**Examples**

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

conn <- connect(connectionDetails)
count <- querySql.ffdf(conn, "SELECT COUNT(*) FROM person")
disconnect(conn)

## End(Not run)
```

---

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(), "errorReport.txt"),
                           oracleTempSchema = NULL, ...)
```

**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 'errorReport.txt' in the current working directory.
oracleTempSchema	A schema that can be used to create temp tables in when using Oracle or Impala.
...	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)
```

---

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(), "errorReport.txt"),
                        snakeCaseToCamelCase = FALSE, oracleTempSchema = NULL, ...)
```

**Arguments**

<code>connection</code>	The connection to the database server.
<code>sql</code>	The SQL to be send.
<code>errorReportFile</code>	The file where an error report will be written if an error occurs. Defaults to 'errorReport.txt' in the current working directory.
<code>snakeCaseToCamelCase</code>	If true, field names are assumed to use snake_case, and are converted to camel-Case.
<code>oracleTempSchema</code>	A schema that can be used to create temp tables in when using Oracle or Impala.
<code>...</code>	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)
```

---

```
renderTranslateQuerySql.ffdf
```

*Render, translate, and query to ffdf*

---

**Description**

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

**Usage**

```
renderTranslateQuerySql.ffdf(connection, sql,
                              errorReportFile = file.path(getwd(), "errorReport.txt"),
                              snakeCaseToCamelCase = FALSE, oracleTempSchema = NULL, ...)
```

**Arguments**

<code>connection</code>	The connection to the database server.
<code>sql</code>	The SQL to be send.
<code>errorReportFile</code>	The file where an error report will be written if an error occurs. Defaults to 'errorReport.txt' in the current working directory.
<code>snakeCaseToCamelCase</code>	If true, field names are assumed to use snake_case, and are converted to camel-Case.
<code>oracleTempSchema</code>	A schema that can be used to create temp tables in when using Oracle or Impala.
<code>...</code>	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.ffdf`.

**Value**

An `ffdf` object

**Examples**

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

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

disconnect(conn)

## 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 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 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

- character, [25, 26](#)
- connect, [3, 7, 13, 37](#)
- createConnectionDetails, [3, 6, 37](#)
- createZipFile, [9](#)
  
- data.frame, [17, 18, 32](#)
- DatabaseConnector, [10](#)
- DatabaseConnector-package  
(DatabaseConnector), [10](#)
- DatabaseConnectorDriver, [10](#)
- dbAppendTable, [14–16, 18, 22–24, 27–31, 33](#)
- dbAppendTable, DatabaseConnectorConnection, character, data.frame-method, [10](#)
- dbBind, [12, 17, 19–23, 25, 26](#)
- dbCanConnect, [22, 23, 32](#)
- dbClearResult, [12, 17, 19–23, 25, 26](#)
- dbClearResult(), [15, 17–23, 28–31](#)
- dbClearResult, DatabaseConnectorResult-method, [11](#)
- dbColumnInfo, [12, 17, 19–23, 25, 26](#)
- dbColumnInfo(), [24](#)
- dbColumnInfo, DatabaseConnectorResult-method, [12](#)
- dbConnect, [22, 23, 32](#)
- dbConnect(), [11, 14–16, 18, 23, 24, 26, 28–30, 32](#)
- dbConnect, DatabaseConnectorDriver-method, [13](#)
- dbCreateTable, [11, 15, 16, 18, 22–24, 27–31, 33](#)
- dbCreateTable(), [10](#)
- dbCreateTable, DatabaseConnectorConnection, character, data.frame-method, [13](#)
- dbDataType, [11, 14–16, 18, 22–24, 27–33](#)
- dbDataType(), [14](#)
- dbDisconnect, [11, 14, 16, 18, 22–24, 27–31, 33](#)
- dbDisconnect(), [22](#)
- dbDisconnect, DatabaseConnectorConnection-method, [14](#)
- dbDriver, [22, 23](#)
- dbExecute, [11, 14–16, 18, 22–24, 27–31, 33](#)
- dbExecute(), [10, 13, 18, 29, 30](#)
- dbExecute, DatabaseConnectorConnection, character-method, [15](#)
- dbExistsTable, [11, 14–16, 18, 22–24, 27–31, 33](#)
- dbExistsTable, DatabaseConnectorConnection, character-method, [16](#)
- dbFetch, [12, 19–23, 25, 26](#)
- dbFetch(), [12, 18–21, 28–30](#)
- dbFetch, DatabaseConnectorResult-method, [17](#)
- dbGetException, [11, 14–16, 18, 22–24, 27–31, 33](#)
- dbGetException, DatabaseConnectorConnection, character, data.frame-method, [27–31, 33](#)
- dbGetInfo, [11, 12, 14–33](#)
- dbGetQuery, [11, 14–16, 22–24, 27–31, 33](#)
- dbGetQuery(), [15, 16, 27–29, 31](#)
- dbGetQuery, DatabaseConnectorConnection, character-method, [18](#)
- dbGetRowCount, [12, 17, 20–23, 25, 26](#)
- dbGetRowCount, DatabaseConnectorResult-method, [19](#)
- dbGetRowsAffected, [12, 17, 19–23, 25, 26](#)
- dbGetRowsAffected(), [15, 30, 31](#)
- dbGetRowsAffected, DatabaseConnectorResult-method, [19](#)
- dbGetStatement, [12, 17, 19–23, 25, 26](#)
- dbGetStatement, DatabaseConnectorResult-method, [20](#)
- dbHasCompleted, [12, 17, 19, 20, 22, 23, 25, 26](#)
- dbHasCompleted, DatabaseConnectorResult-method, [21](#)
- DBIConnection, [11, 14–16, 18, 21–26, 28–30, 32](#)
- character, data.frame-method, [32](#)
- DBIDriver, [21, 22](#)
- DBIObject, [21, 22](#)
- DBIResult, [11, 12, 17, 19–22, 29–31](#)
- dbIsReadOnly, [11, 12, 14–33](#)
- dbIsValid, [11, 12, 14–21, 24–33](#)
- dbIsValid, DatabaseConnectorDbiConnection-method, [21](#)
- dbIsValid, DatabaseConnectorJdbcConnection-method, [22](#)
- dbListConnections, [22, 23, 32](#)
- dbListFields, [11, 14–16, 18, 22–24, 27–31,](#)

- 33
- dbListFields, DatabaseConnectorConnection, character-method, 33
- dbListObjects, 11, 14–16, 18, 22–24, 27–31, 33
- dbListResults, 11, 14–16, 18, 22–24, 27–31, 33
- dbListTables, 11, 14–16, 18, 22–24, 27–31, 33
- dbListTables, DatabaseConnectorConnection-method, 24
- dbQuoteIdentifier, 12, 17, 19–23, 26
- dbQuoteIdentifier(), 11, 14, 16, 26–28, 32, 33
- dbQuoteIdentifier, DatabaseConnectorConnection, character-method, 25
- dbQuoteLiteral, 12, 17, 19–23, 25, 26
- dbQuoteString, 12, 17, 19–23, 25
- dbQuoteString, DatabaseConnectorConnection, character-method, 25
- dbReadTable, 11, 14–16, 18, 22–24, 28–31, 33
- dbReadTable, DatabaseConnectorConnection, character-method, 26
- dbRemoveTable, 11, 14–16, 18, 22–24, 27, 29–31, 33
- dbRemoveTable, DatabaseConnectorConnection, character-method, 27
- dbSendQuery, 11, 14–16, 18, 22–24, 27, 28, 31, 33
- dbSendQuery(), 16–22, 31
- dbSendQuery, DatabaseConnectorDbiConnection, character-method, 28
- dbSendQuery, DatabaseConnectorJdbcConnection, character-method, 29
- dbSendStatement, 11, 14–16, 18, 22–24, 27–30, 33
- dbSendStatement(), 15, 17–22, 29–31
- dbSendStatement, DatabaseConnectorConnection, character-method, 30
- dbUnloadDriver, DatabaseConnectorDriver-method, 31
- dbUnquoteIdentifier, 12, 17, 19–23, 25, 26
- dbUnquoteIdentifier(), 25
- dbWriteTable, 11, 14–16, 18, 22–24, 27–31
- dbWriteTable(), 24, 27
- dbWriteTable, DatabaseConnectorConnection, character, data.frame-method, 32
- disconnect, 33
- executeSql, 34, 42
- getTableNames, 35
- Id, 25
- insertTable, 35
- jdbcDrivers, 3, 7, 37
- lowLevelExecuteSql, 38
- lowLevelQuerySql, 38
- lowLevelQuerySql.ffdf, 39
- print, 44, 45
- querySql, 38, 39, 43
- querySql.ffdf, 39, 40, 44
- renderTranslateExecuteSql, 41
- renderTranslateQuerySql, 42
- renderTranslateQuerySql.ffdf, 43
- rownames, 27
- setClass, 44, 45
- setGeneric, 44, 45
- setIs, 44, 45
- show, DatabaseConnectorConnection-method, 44
- show, DatabaseConnectorDriver-method, 45
- showDefault, 44, 45
- showMethods, 45
- SQL, 25, 26
- sqlAppendTable(), 10
- sqlAppendTableTemplate(), 10
- sqlColumnToRownames(), 27
- sqlCreateTable(), 13
- sqlRownamesToColumn(), 11, 14