

# Package ‘DatabaseConnector’

April 19, 2018

**Type** Package

**Title** A Package for Connecting to Various DBMSs

**Version** 2.1.0

**Date** 2018-04-19

**Author** Martijn J. Schuemie and Marc A. Suchard

**Maintainer** Martijn Schuemie <schuemie@ohdsi.org>

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

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

**Suggests** aws.s3,  
uuid,  
R.utils,  
testthat,  
DBItest

**License** Apache License

**RoxygenNote** 6.0.1.9000

## R topics documented:

connect . . . . .	2
createConnectionDetails . . . . .	6
DatabaseConnector . . . . .	9
DatabaseConnectorDriver . . . . .	10
dbClearResult,DatabaseConnectorResult-method . . . . .	10
dbColumnInfo,DatabaseConnectorResult-method . . . . .	11
dbConnect,DatabaseConnectorDriver-method . . . . .	11
dbDisconnect,DatabaseConnectorConnection-method . . . . .	12
dbExecute,DatabaseConnectorConnection,character-method . . . . .	13

dbExistsTable,DatabaseConnectorConnection,character-method . . . . .	13
dbFetch,DatabaseConnectorResult-method . . . . .	14
dbGetQuery,DatabaseConnectorConnection,character-method . . . . .	15
dbGetRowCount,DatabaseConnectorResult-method . . . . .	16
dbGetRowsAffected,DatabaseConnectorResult-method . . . . .	17
dbGetStatement,DatabaseConnectorResult-method . . . . .	17
dbHasCompleted,DatabaseConnectorResult-method . . . . .	18
dbIsValid,DatabaseConnectorConnection-method . . . . .	19
dbListFields,DatabaseConnectorConnection,character-method . . . . .	20
dbListTables,DatabaseConnectorConnection-method . . . . .	20
dbQuoteIdentifier,DatabaseConnectorConnection,character-method . . . . .	21
dbQuoteString,DatabaseConnectorConnection,character-method . . . . .	22
dbReadTable,DatabaseConnectorConnection,character-method . . . . .	23
dbRemoveTable,DatabaseConnectorConnection,character-method . . . . .	24
dbSendQuery,DatabaseConnectorConnection,character-method . . . . .	25
dbSendStatement,DatabaseConnectorConnection,character-method . . . . .	26
dbUnloadDriver,DatabaseConnectorDriver-method . . . . .	27
dbWriteTable,DatabaseConnectorConnection,character,data.frame-method . . . . .	28
disconnect . . . . .	29
executeSql . . . . .	29
getTableNames . . . . .	30
insertTable . . . . .	31
lowLevelExecuteSql . . . . .	32
lowLevelQuerySql . . . . .	33
lowLevelQuerySql.ffdf . . . . .	33
querySql . . . . .	34
querySql.ffdf . . . . .	35
show,DatabaseConnectorConnection-method . . . . .	36
show,DatabaseConnectorDriver-method . . . . .	37

## Index 39

---

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

---

## Description

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

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

## Arguments

connectionDetails

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

dbms

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

- "mysql" for MySQL

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

## Details

This function creates a connection to a database.

## Value

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

## DBMS parameter details

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

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

Oracle:

- user. The user name used to access the server
- password. The password for that user

- **server.** This field contains the SID, or host and servicename, SID, or TNSName: '`<sid>`', '`<host>/<sid>`', '`<host>/<service name>`', or '`<tnsname>`'
- **port.** Specifies the port on the server (default = 1521)
- **schema.** This field contains the schema (i.e. 'user' in Oracle terms) containing the tables
- **extraSettings** The configuration settings for the connection (i.e. SSL Settings such as "`(PROTOCOL=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).
- **domain.** Optionally, the domain can be specified here.(See note below).
- **password.** The password used to log on to the server
- **server.** This field contains the host name of the server
- **port.** Not used for SQL Server
- **schema.** The database containing the tables. If both database and schema are specified (e.g. '`my_database.dbo`'), then only the database part is used, the schema is ignored.
- **extraSettings** The configuration settings for the connection (i.e. SSL Settings such as "`encrypt=true; trustServerCertificate=false;`")

#### Microsoft PDW:

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

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

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

#### Redshift:

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

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

Netezza:

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

Impala:

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

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

## Examples

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

```

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

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

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

## End(Not run)

```

---

```
createConnectionDetails
```

```
createConnectionDetails
```

---

## Description

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

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

## Arguments

<code>dbms</code>	The type of DBMS running on the server. Valid values are <ul style="list-style-type: none"> <li>• "mysql" for MySQL</li> <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> </ul>
<code>user</code>	The user name used to access the server.
<code>domain</code>	For SQL Server only: the Windows domain (optional).
<code>password</code>	The password for that user.
<code>server</code>	The name of the server.
<code>port</code>	(optional) The port on the server to connect to.

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

### Details

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

### Value

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

### DBMS parameter details

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

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

Oracle:

- user. The user name used to access the server
- password. The password for that user
- server. This field contains the SID, or host and servicename, SID, or TNSName: '<sid>', '<host>/<sid>', '<host>/<service name>', or '<tnsname>'
- port. Specifies the port on the server (default = 1521)
- schema. This field contains the schema (i.e. 'user' in Oracle terms) containing the tables
- extraSettings The configuration settings for the connection (i.e. SSL Settings such as "(PROTOCOL=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).
- domain. Optionally, the domain can be specified here.(See note below).

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

#### Microsoft PDW:

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

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

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

#### Redshift:

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

#### Netezza:

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



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

Impala:

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

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

## Examples

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

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

## End(Not run)
```

---

DatabaseConnector

*DatabaseConnector*


---

## Description

DatabaseConnector

---

DatabaseConnectorDriver

*Create a DatabaseConnectorDriver object*


---

### Description

Create a DatabaseConnectorDriver object

### Usage

```
DatabaseConnectorDriver()
```

---

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](#), [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 should describe 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

A data.frame with one row per output field in res. Methods MUST include name, field.type (the SQL type), and data.type (the R data type) columns, and MAY contain other database specific information like scale and precision or whether the field can store NULLs.

### See Also

Other DBIResult generics: [DBIResult-class](#), [dbBind](#), [dbClearResult](#), [dbFetch](#), [dbGetInfo](#), [dbGetRowCount](#), [dbGetRowsAffected](#), [dbGetStatement](#), [dbHasCompleted](#), [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 <code>link{DatabaseConnectorDriver}</code> 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)
```

---

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](#), [dbDataType](#), [dbExecute](#), [dbExistsTable](#), [dbGetException](#), [dbGetInfo](#), [dbGetQuery](#), [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()`.

#### Methods in other packages:

- `DBI::dbExecute("DBIConnection", "character")`

### Usage

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

### Arguments

<code>conn</code>	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
<code>statement</code>	a character string containing SQL.
<code>...</code>	Other parameters passed on to methods.

### 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](#), [dbDataType](#), [dbDisconnect](#), [dbExistsTable](#), [dbGetException](#), [dbGetInfo](#), [dbGetQuery](#), [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.

#### Methods in other packages:

- `DBI::dbExistsTable("DBIConnection", "Id")`

**Usage**

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbExistsTable(conn, name,
  database = NULL, schema = 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.
...	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](#), [dbDataType](#), [dbDisconnect](#), [dbExecute](#), [dbGetException](#), [dbGetInfo](#), [dbGetQuery](#), [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.

**Methods in other packages:**

- [DBI::dbFetch\("DBIResult"\)](#)

**Usage**

```
## S4 method for signature 'DatabaseConnectorResult'
dbFetch(res, datesAsString = FALSE, ...)
```

**Arguments**

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

### Methods in other packages:

- [DBI::dbGetQuery\("DBIConnection", "character"\)](#)

## Usage

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

## Arguments

<code>conn</code>	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
<code>statement</code>	a character string containing SQL.
<code>...</code>	Other parameters passed on to methods.

## Details

This method is for SELECT queries only. 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](#), [dbDataType](#), [dbDisconnect](#), [dbExecute](#), [dbExistsTable](#), [dbGetException](#), [dbGetInfo](#), [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**

<code>res</code>	An object inheriting from <a href="#">DBIResult</a> .
<code>...</code>	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](#), [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 <a href="#">DBIResult</a> .
...	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](#), [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 <a href="#">DBIResult</a> .
...	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](#), [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**

<code>res</code>	An object inheriting from <a href="#">DBIResult</a> .
<code>...</code>	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](#), [dbIsValid](#), [dbQuoteIdentifier](#), [dbQuoteLiteral](#), [dbQuoteString](#), [dbUnquoteIdentifier](#)

---

dbIsValid, DatabaseConnectorConnection-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 'DatabaseConnectorConnection'
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](#), [dbConnect](#), [dbDataType](#), [dbDriver](#), [dbGetInfo](#), [dbListConnections](#)

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

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

---

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

---

### Description

List field names of a remote table

### Usage

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

### Arguments

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
name	a character string with the name of the remote table.
database	Name of the database.
schema	Name of the schema.
...	Other parameters passed on to methods.

### Value

a character vector

### See Also

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

Other DBIConnection generics: [DBIConnection-class](#), [dbDataType](#), [dbDisconnect](#), [dbExecute](#), [dbExistsTable](#), [dbGetException](#), [dbGetInfo](#), [dbGetQuery](#), [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](#), [dbDataType](#), [dbDisconnect](#), [dbExecute](#), [dbExistsTable](#), [dbGetException](#), [dbGetInfo](#), [dbGetQuery](#), [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\(\)](#).

**Methods in other packages:**

- [DBI::dbQuoteIdentifier\("DBIConnection", "ANY"\)](#)
- [DBI::dbQuoteIdentifier\("DBIConnection", "Id"\)](#)
- [DBI::dbQuoteIdentifier\("DBIConnection", "SQL"\)](#)
- [DBI::dbQuoteIdentifier\("DBIConnection", "character"\)](#)

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

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.)

**See Also**

Other DBIResult generics: [DBIResult-class](#), [dbBind](#), [dbClearResult](#), [dbColumnInfo](#), [dbFetch](#), [dbGetInfo](#), [dbGetRowCount](#), [dbGetRowsAffected](#), [dbGetStatement](#), [dbHasCompleted](#), [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.

**Methods in other packages:**

- [DBI::dbQuoteString\("DBIConnection", "ANY"\)](#)
- [DBI::dbQuoteString\("DBIConnection", "SQL"\)](#)
- [DBI::dbQuoteString\("DBIConnection", "character"\)](#)

**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](#), [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.

**Methods in other packages:**

- [DBI::dbReadTable\("DBIConnection", "Id"\)](#)
- [DBI::dbReadTable\("DBIConnection", "character"\)](#)

**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](#), [dbDataType](#), [dbDisconnect](#), [dbExecute](#), [dbExistsTable](#), [dbGetException](#), [dbGetInfo](#), [dbGetQuery](#), [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.

#### Methods in other packages:

- [DBI::dbRemoveTable\("DBIConnection", "Id"\)](#)

### Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character'
dbRemoveTable(conn, name,
  database = NULL, schema = NULL, 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 a DBMS table name.
<code>database</code>	Name of the database.
<code>schema</code>	Name of the schema.
<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**

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](#), [dbDataType](#), [dbDisconnect](#), [dbExecute](#), [dbExistsTable](#), [dbGetException](#), [dbGetInfo](#), [dbGetQuery](#), [dbIsValid](#), [dbListFields](#), [dbListObjects](#), [dbListResults](#), [dbListTables](#), [dbReadTable](#), [dbSendQuery](#), [dbSendStatement](#), [dbWriteTable](#)

---

dbSendQuery, DatabaseConnectorConnection, 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 'DatabaseConnectorConnection,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](#), [dbDataType](#), [dbDisconnect](#), [dbExecute](#), [dbExistsTable](#), [dbGetException](#), [dbGetInfo](#), [dbGetQuery](#), [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\(\)](#).

**Methods in other packages:**

- [DBI::dbSendStatement\("DBIConnection", "character"\)](#)

**Usage**

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

**Arguments**

<code>conn</code>	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
<code>statement</code>	a character string containing SQL.
<code>...</code>	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](#), [dbDataType](#), [dbDisconnect](#), [dbExecute](#), [dbExistsTable](#), [dbGetException](#), [dbGetInfo](#), [dbGetQuery](#), [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**

<code>drv</code>	an object that inherits from <code>DBIDriver</code> as created by <code>dbDriver</code> .
<code>...</code>	any other arguments are passed to the driver <code>drvName</code> .

**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`, a 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](#), [dbConnect](#), [dbDataType](#), [dbGetInfo](#), [dbIsValid](#), [dbListConnections](#)

Other `DBIDriver` generics: [DBIDriver-class](#), [dbConnect](#), [dbDataType](#), [dbGetInfo](#), [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.

### Methods in other packages:

- [DBI::dbWriteTable\("DBIConnection", "Id"\)](#)

## Usage

```
## S4 method for signature 'DatabaseConnectorConnection,character,data.frame'
dbWriteTable(conn,
  name, value, overwrite = FALSE, append = FALSE, temporary = FALSE,
  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> .
value	a <a href="#">data.frame</a> (or coercible to data.frame).
overwrite	Overwrite an existing table (if exists)?
append	Append to existing table?
temporary	Should the table created as a temp table?
oracleTempSchema	Specifically for Oracle, a schema with write privileges where temp tables can be created.
...	Other parameters passed on to methods.

## Value

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

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

## See Also

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

---

disconnect	<i>Disconnect from the server</i>
------------	-----------------------------------

---

### Description

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

### Usage

```
disconnect(connection)
```

### Arguments

`connection`      The connection to the database server.

### Examples

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

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

## End(Not run)
```

---

executeSql	<i>Execute SQL code</i>
------------	-------------------------

---

### Description

This function executes SQL consisting of one or more statements.

### Usage

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

### Arguments

<code>connection</code>	The connection to the database server.
<code>sql</code>	The SQL to be executed
<code>profile</code>	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.
<code>progressBar</code>	When true, a progress bar is shown based on the statements in the SQL code.
<code>reportOverallTime</code>	When true, the function will display the overall time taken to execute all statements.

## Details

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

## Examples

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

conn <- connect(connectionDetails)
executeSql(conn, "CREATE TABLE x (k INT); CREATE TABLE y (k INT);")
disconnect(conn)

## End(Not run)
```

---

getTableNames	<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)
```

## Arguments

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

## 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.

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")
```

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)
```

---

lowLevelExecuteSql	<i>Execute SQL code</i>
--------------------	-------------------------

---

## 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	<i>Low level function for retrieving data to a data frame</i>
------------------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

A data frame containing the data retrieved from the server

---

lowLevelQuerySql.ffdf	<i>Low level function for retrieving data to an ffdf object</i>
-----------------------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

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

---

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

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

A data frame.

**Examples**

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

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

## End(Not run)
```

---

querySql.ffdf	<i>Retrieves data to an ffdf object</i>
---------------	---

---

## Description

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

## Usage

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

## Arguments

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

## Details

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

## Value

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

## Examples

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

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

## End(Not run)
```

---

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.

## Examples

```
## following the example shown in the setMethod documentation ...
setClass("track", slots = c(x="numeric", y="numeric"))
setClass("trackCurve", contains = "track", slots = c(smooth = "numeric"))

t1 <- new("track", x=1:20, y=(1:20)^2)

tc1 <- new("trackCurve", t1)

setMethod("show", "track",
  function(object)print(rbind(x = object@x, y=object@y))
)
## The method will now be used for automatic printing of t1
```

```

t1

## Not run:  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
x    1    2    3    4    5    6    7    8    9   10   11   12
y    1    4    9   16   25   36   49   64   81  100  121  144
      [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
x    13   14   15   16   17   18   19   20
y   169  196  225  256  289  324  361  400

## End(Not run)
## and also for tc1, an object of a class that extends "track"
tc1

## Not run:  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
x    1    2    3    4    5    6    7    8    9   10   11   12
y    1    4    9   16   25   36   49   64   81  100  121  144
      [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
x    13   14   15   16   17   18   19   20
y   169  196  225  256  289  324  361  400

## End(Not run)

```

---

```

show,DatabaseConnectorDriver-method
      Show an Object

```

---

## Description

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

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

## Usage

```

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

```

## Arguments

object            Any R object

## Details

Objects from an S4 class (a class defined by a call to [setClass](#)) will be displayed automatically 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.

**Examples**

```
## following the example shown in the setMethod documentation ...
setClass("track", slots = c(x="numeric", y="numeric"))
setClass("trackCurve", contains = "track", slots = c(smooth = "numeric"))

t1 <- new("track", x=1:20, y=(1:20)^2)

tc1 <- new("trackCurve", t1)

setMethod("show", "track",
  function(object)print(rbind(x = object@x, y=object@y))
)
## The method will now be used for automatic printing of t1

t1

## Not run:   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
x    1    2    3    4    5    6    7    8    9   10   11   12
y    1    4    9   16   25   36   49   64   81  100  121  144
  [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
x   13   14   15   16   17   18   19   20
y  169  196  225  256  289  324  361  400

## End(Not run)
## and also for tc1, an object of a class that extends "track"
tc1

## Not run:   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
x    1    2    3    4    5    6    7    8    9   10   11   12
y    1    4    9   16   25   36   49   64   81  100  121  144
  [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
x   13   14   15   16   17   18   19   20
y  169  196  225  256  289  324  361  400

## End(Not run)
```

# Index

character, [22](#)  
connect, [2](#), [7](#), [11](#)  
createConnectionDetails, [2](#), [6](#)

data.frame, [15](#), [16](#), [28](#)  
DatabaseConnector, [9](#)  
DatabaseConnector-package  
    (DatabaseConnector), [9](#)  
DatabaseConnectorDriver, [10](#)  
dbBind, [10](#), [11](#), [15–19](#), [22](#), [23](#)  
dbClearResult, [11](#), [15–19](#), [22](#), [23](#)  
dbClearResult(), [13](#), [15–19](#), [25](#), [26](#)  
dbClearResult, DatabaseConnectorResult-method,  
    [10](#)  
dbColumnInfo, [10](#), [15–19](#), [22](#), [23](#)  
dbColumnInfo(), [20](#)  
dbColumnInfo, DatabaseConnectorResult-method,  
    [11](#)  
dbConnect, [19](#), [27](#)  
dbConnect(), [12–15](#), [20](#), [21](#), [23–26](#), [28](#)  
dbConnect, DatabaseConnectorDriver-method,  
    [11](#)  
dbDataType, [12–14](#), [16](#), [19–21](#), [24–28](#)  
dbDisconnect, [13](#), [14](#), [16](#), [19–21](#), [24–28](#)  
dbDisconnect(), [19](#)  
dbDisconnect, DatabaseConnectorConnection-method,  
    [12](#)  
dbDriver, [19](#)  
dbExecute, [12](#), [14](#), [16](#), [19–21](#), [24–28](#)  
dbExecute(), [15](#), [16](#), [26](#)  
dbExecute, DatabaseConnectorConnection, character-method,  
    [13](#)  
dbExistsTable, [12](#), [13](#), [16](#), [19–21](#), [24–28](#)  
dbExistsTable, DatabaseConnectorConnection, character-method,  
    [13](#)  
dbFetch, [10](#), [11](#), [16–19](#), [22](#), [23](#)  
dbFetch(), [15–18](#), [25](#)  
dbFetch, DatabaseConnectorResult-method,  
    [14](#)  
dbGetException, [12–14](#), [16](#), [19–21](#), [24–28](#)  
dbGetInfo, [10–28](#)  
dbGetQuery, [12–14](#), [19–21](#), [24–28](#)  
dbGetQuery(), [13](#), [23](#), [25](#), [27](#)  
dbGetQuery, DatabaseConnectorConnection, character-method,  
    [15](#)  
dbGetRowCount, [10](#), [11](#), [15](#), [17–19](#), [22](#), [23](#)  
dbGetRowCount, DatabaseConnectorResult-method,  
    [16](#)  
dbGetRowsAffected, [10](#), [11](#), [15](#), [16](#), [18](#), [19](#),  
    [22](#), [23](#)  
dbGetRowsAffected(), [13](#), [26](#)  
dbGetRowsAffected, DatabaseConnectorResult-method,  
    [17](#)  
dbGetStatement, [10](#), [11](#), [15–19](#), [22](#), [23](#)  
dbGetStatement, DatabaseConnectorResult-method,  
    [17](#)  
dbHasCompleted, [10](#), [11](#), [15–19](#), [22](#), [23](#)  
dbHasCompleted, DatabaseConnectorResult-method,  
    [18](#)  
DBI::dbFetch(DBIResult), [14](#)  
DBIConnection, [12–15](#), [19–26](#), [28](#)  
DBIDriver, [19](#)  
DBIObject, [19](#)  
DBIResult, [10](#), [11](#), [14](#), [16–19](#), [25](#), [26](#)  
dbIsValid, [10–18](#), [20–28](#)  
dbIsValid, DatabaseConnectorConnection-method,  
    [19](#)  
dbListConnections, [19](#), [27](#)  
dbListFields, [12–14](#), [16](#), [19](#), [21](#), [24–28](#)  
dbListFields, DatabaseConnectorConnection, character-method,  
    [20](#)  
dbListObjects, [12–14](#), [16](#), [19–21](#), [24–28](#)  
dbListResults, [12–14](#), [16](#), [19–21](#), [24–28](#)  
dbListTables, [12–14](#), [16](#), [19](#), [20](#), [24–28](#)  
dbListTables, DatabaseConnectorConnection-method,  
    [20](#)  
dbQuoteIdentifier, [10](#), [11](#), [15–19](#), [23](#)  
dbQuoteIdentifier(), [14](#), [23–25](#), [28](#)  
dbQuoteIdentifier, DatabaseConnectorConnection, character-method,  
    [21](#)  
dbQuoteLiteral, [10](#), [11](#), [15–19](#), [22](#), [23](#)  
dbQuoteString, [10](#), [11](#), [15–19](#), [22](#)  
dbQuoteString, DatabaseConnectorConnection, character-method,  
    [22](#)  
dbReadTable, [12–14](#), [16](#), [19–21](#), [25–28](#)  
dbReadTable, DatabaseConnectorConnection, character-method,

23  
dbRemoveTable, 12–14, 16, 19–21, 24, 26–28  
dbRemoveTable(DatabaseConnectorConnection, character-method,  
24  
dbSendQuery, 12–14, 16, 19–21, 24, 25, 27, 28  
dbSendQuery(), 13–19, 26, 27  
dbSendQuery(DatabaseConnectorConnection, character-method,  
25  
dbSendStatement, 12–14, 16, 19–21, 24–26,  
28  
dbSendStatement(), 13, 15–19, 25, 26  
dbSendStatement(DatabaseConnectorConnection, character-method,  
26  
dbUnloadDriver(DatabaseConnectorDriver-method,  
27  
dbUnquoteIdentifier, 10, 11, 15–19, 22, 23  
dbUnquoteIdentifier(), 21  
dbWriteTable, 12–14, 16, 19–21, 24–27  
dbWriteTable(), 21, 24  
dbWriteTable(DatabaseConnectorConnection, character, data.frame-method,  
28  
disconnect, 29  
executeSql, 29  
getTableNames, 30  
Id, 21  
insertTable, 31  
lowLevelExecuteSql, 32  
lowLevelQuerySql, 33  
lowLevelQuerySql.ffdf, 33  
print, 36, 37  
querySql, 33, 34  
querySql.ffdf, 33, 35  
rownames, 23  
setClass, 36, 37  
setGeneric, 36, 37  
setIs, 36, 37  
show, DatabaseConnectorConnection-method,  
36  
show, DatabaseConnectorDriver-method,  
37  
showDefault, 36, 37  
showMethods, 36, 38  
SQL, 21, 22  
sqlColumnToRownames(), 23