Package 'ROracle'

May 7, 2026

Version 1.5-1

Date 2025-10-21

Title OCI Based Oracle Database Interface for R

Maintainer Rajendra S. Pingte <rajendra.pingte@oracle.com>

Description Oracle Database interface (DBI) driver for R.
This is a DBI-compliant Oracle driver based on the OCI.

SystemRequirements curl Oracle Instant Client or Oracle Database Client

LazyLoad yes

Depends R (>= 4.4), methods, DBI (>= 0.2-5)

Imports utils

License LGPL

URL https://www.oracle.com

Collate dbi.R oci.R zzz.R

NeedsCompilation yes

Repository CRAN

Date/Publication 2025-10-27 19:30:02 UTC

Author Denis Mukhin [aut],
David A. James [aut],
Jake Luciani [aut],
Rajendra S. Pingte [aut, cre]

Contents

`dbCommit-methods` *DBMS Transaction Management*

Description

Commits or roll backs the current transaction in an Oracle connection

Usage

```
## S4 method for signature 'OraConnection'
dbCommit(conn, ...)
## S4 method for signature 'OraConnection'
dbRollback(conn, ...)
```

Arguments

`conn`	a `OraConnection` object, as produced by the function `dbConnect`
`...`	currently unused.

Details

dbCommit implementation saves all changes done on that connection. Changes can not be undone once saved permanently.

dbRollback implementation undo all changes done after last savepoint.

Side Effects

dbCommit saves changes permanently.

dbRollback undo all changes done after last save point.

References

For the Oracle Database documentation see https://docs.oracle.com/en/.

See Also

`Oracle`, `dbConnect`, `dbSendQuery`, `dbGetQuery`, `fetch`, `dbCommit`, `dbGetInfo`, `dbReadTable`.

Examples

```
## Not run:
  drv <- dbDriver("Oracle")
  con <- dbConnect(drv, "scott", "tiger")
  dbReadTable(con, "EMP")
  rs <- dbSendQuery(con, "delete from emp where deptno = 10")
  dbReadTable(con, "EMP")
  if(dbGetInfo(rs, what = "rowsAffected") > 1)
  {
    warning("dubious deletion -- rolling back transaction")
    dbRollback(con)
  }
  dbReadTable(con, "EMP")

## End(Not run)
```

---

dbConnect-methods *Create a Connection Object to an Oracle DBMS*

Description

These methods are straightforward implementations of the corresponding generic functions.

Usage

```
## S4 method for signature 'OraDriver'
dbConnect(drv, username = "", password = "",
          dbname = "", prefetch = FALSE,
          bulk_read = 1000L, bulk_write = 1000L, stmt_cache = 0L,
          external_credentials = FALSE, sysdba = FALSE, ...)
## S4 method for signature 'ExtDriver'
dbConnect(drv, prefetch = FALSE, bulk_read = 1000L,
          bulk_write = 1000L, stmt_cache = 0L,
          external_credentials = FALSE, sysdba = FALSE, ...)
## S4 method for signature 'OraConnection'
dbDisconnect(conn, ...)
```

Arguments

drv	An object of class OraDriver or ExtDriver.
conn	An OraConnection object as produced by dbConnect.
username	A character string specifying a user name.
password	A character string specifying a password.
dbname	A character string specifying a connect identifier (for more information refer to Chapter 8, Configuring Naming Methods, of Oracle Database Net Services Administrator's Guide). This is the same as part of the SQL*Plus connect string

that follows the '@' sign. If you are using Oracle Wallet to store username and password, then this string should be the connect string used to create the wallet mapping (for more information, refer to Configuring Clients to Use the External Password Store in Chapter 3 of Oracle Database Security Guide). Connect identifiers for an Oracle TimesTen IMDB instance are supported via the OCI tnsnames or easy connect naming methods. For additional information on TimesTen connections for OCI see chapter 3, TimesTen Support for OCI, of TimesTen In-Memory C Developer's Guide. Examples below show various ways to specify the connect identifier.

`prefetch` A logical value indicating TRUE or FALSE. When set to TRUE, ROracle will use OCI prefetch buffers to retrieve additional data from the server thus saving memory required in RODBI/ROOCI by allocating a single row buffer to fetch the data from OCI. Using prefetch results in a fetch call for every row. By default, prefetch is FALSE and array fetch is used to retrieve the data from the server.

`bulk_read` An integer value indicating the number of rows to fetch at a time. The default value is 1000L. When the prefetch option is selected, memory is allocated for prefetch buffers and OCI will fetch that many rows at a time. When prefetch is not used (the default), memory is allocated in RODBI/ROOCI define buffers. Setting this to a large value will result in more memory allocated based on the number of columns in the select list and the types of columns. For a column of type character, define buffers are allocated using the maximum width times the NLS maximum width. Applications should adjust this value based on the query result and a larger value will benefit queries that return a large result. An application can tune this value as needed.

`bulk_write` An integer value indicating the number of rows to insert, update or delete at a time. The default value is 1000L. When the bulk_write value is given in argument, memory is allocated for buffers and OCI will write that many rows at a time. When bulk_write argument is not given, the default value 1000 is used to allocate memory for the bind buffers. Setting this to a large value will result in more memory allocated based on the number of columns in the insert list and the types of columns.

`stmt_cache` An integer value indicating the number of statements to cache. It means that cursors are ready to be used without the need to parse the statements again. The default value is 0L. If stmt_cache value is greater than 0L then `prefetch` value must be set to TRUE.

`external_credentials`
A logical value indicating TRUE or FALSE. When set to TRUE, ROracle will begin OCI session authenticated with external credentials on the connection. The default value is FALSE.

`sysdba` A logical value indicating TRUE or FALSE. When set to TRUE, ROracle will begin OCI session with SYSDBA privileges on the connection. The default value is FALSE.

`...` Currently unused.

Details

dbConnect This connection object is used to execute operations on the database.

When prefetch is set to TRUE, it allows the use of the OCI prefetch buffer to retrieve additional data from the server.

The bulk_read argument is used to set an integer value indicating the number of rows to fetch at a time.

The bulk_write argument is used to set an integer value indicating the number of rows to write at a time.

The stmt_cache argument is used to enable or disable the statement caching feature. Its value specifies the statement cache size.

The external_credentials argument is used to begin OCI session authenticated with external credentials on the connection.

The sysdba argument is used to begin OCI session with SYSDBA privileges on the connection.

When establishing a connection with an ExtDriver driver, none of the arguments specifying credentials are used. A connection in this mode is a singleton object, that is, all calls to dbConnect return the same connection object.

dbDisconnect This implementation disconnects the connection between R and the database server. It frees all resources used by the connection object. It frees all result sets associated with this connection object.

Value

dbConnect	An object OraConnection whose class extends DBIConnection. This object is used to execute SQL queries on the database.
dbDisconnect	A logical value indicating whether the operation succeeded or not.

Side Effects

dbConnect Establishes a connection between R and an Oracle Database server.

dbDisconnect Frees resources used by the connection object.

References

For the Oracle Database documentation see https://docs.oracle.com/en/.

See Also

Oracle, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbReadTable.

Examples

```
## Not run:
  ## Create an Oracle Database instance and create one connection
  ## on the same machine.
  drv <- dbDriver("Oracle")

  ## Use username/password authentication.
  con <- dbConnect(drv, username = "scott", password = "tiger")
```

```
    ## Run a SQL statement by creating first a resultSet object.
    rs <- dbSendQuery(con, "select * from emp where deptno = 10")

    ## We now fetch records from the resultSet into a data.frame.
    data <- fetch(rs)       ## extract all rows
    dim(data)

## End(Not run)
  ## Not run:
    ## Create an Oracle Database instance and create one connection
    ## to a remote database using the SID in the connect string.
    drv <- dbDriver("Oracle")

    ## Refer to Oracle Database Net Services Administrator's Guide for
    ## details on connect string specification.
    host <- "myhost"
    port <- 1521
    sid <- "mysid"
    connect.string <- paste(
      "(DESCRIPTION=",
      "(ADDRESS=(PROTOCOL=tcp)(HOST=", host, ")(PORT=", port, "))",
        "(CONNECT_DATA=(SID=", sid, ")))", sep = "")

    ## Use username/password authentication.
    con <- dbConnect(drv, username = "scott", password = "tiger",
                     dbname = connect.string)

    ## Run a SQL statement by creating first a resultSet object.
    rs <- dbSendQuery(con, "select * from emp where deptno = 10")

    ## We now fetch records from the resultSet into a data.frame.
    data <- fetch(rs)       ## extract all rows
    dim(data)

## End(Not run)
  ## Not run:
    ## Create an Oracle Database instance and create one connection
    ## to a remote database using the service name.
    drv <- dbDriver("Oracle")

    ## Refer to Oracle Database Net Services Administrator's Guide for
    ## details on connect string specification.

    host <- "myhost"
    port <- 1521
    svc <- "mydb.example.com"
    connect.string <- paste(
      "(DESCRIPTION=",
      "(ADDRESS=(PROTOCOL=tcp)(HOST=", host, ")(PORT=", port, "))",
      "(CONNECT_DATA=(SERVICE_NAME=", svc, ")))", sep = "")
    ## Use username/password authentication.
    con <- dbConnect(drv, username = "scott", password = "tiger",
                     dbname = connect.string)
```

```
    ## Run a SQL statement by creating first a resultSet object.
    rs <- dbSendQuery(con, "select * from emp where deptno = 10")

    ## We now fetch records from the resultSet into a data.frame.
    data <- fetch(rs)       ## extract all rows
    dim(data)

## End(Not run)
  ## Not run:
    ## Create an Oracle Database instance and create one connection.
    drv <- dbDriver("Oracle")

    ## Use Oracle Wallet authentication.
    con <- dbConnect(drv, username ="", password="",
    dbname = "<wallet_connect_string>")

    ## Run a SQL statement by creating first a resultSet object.
    rs <- dbSendQuery(con, "select * from emp where deptno = 10")

    ## We now fetch records from the resultSet into a data.frame.
    data <- fetch(rs)       ## extract all rows
    dim(data)

## End(Not run)
  ## Not run:
    ## Create an Oracle Database instance and create one connection.
    drv <- dbDriver("Oracle")

    ## Connect to a TimesTen IMDB instance using the easy connect
    ## naming method where SampleDb is a direct driver TimesTen DSN.
    con <- dbConnect(drv, username ="scott", password="tiger",
                     dbname = "localhost/SampleDb:timesten_direct")

    ## Run a SQL statement by creating first a resultSet object.
    rs <- dbSendQuery(con, "select * from dual")

    ## We now fetch records from the resultSet into a data.frame.
    data <- fetch(rs)       ## extract all rows
    dim(data)

## End(Not run)
  ## Not run:
    ## Connect to an extproc (this assumes that the driver has already
    ## been initialized in the embedded R code by passing an external
    ## pointer representing the extproc context).
    con <- dbConnect(Extproc())

    ## Run a SQL statement by creating first a resultSet object.
    rs <- dbSendQuery(con, "select * from dual")

    ## We now fetch records from the resultSet into a data.frame.
    data <- fetch(rs)       ## extract all rows
```

```
    dim(data)

## End(Not run)
  ## Not run:
    ## Create an Oracle Database instance and create one connection.
    drv <- dbDriver("Oracle")

    ## Create connection with SYSDBA privileges.
    con <- dbConnect(drv, username ="scott", password="tiger",
                     sysdba = TRUE)

    ## Run a SQL statement by creating first a resultSet object.
    rs <- dbSendQuery(con, "select * from emp where deptno = 10")

    ## We now fetch records from the resultSet into a data.frame.
    data <- fetch(rs)       ## extract all rows
    dim(data)

## End(Not run)
  ## Not run:
    ## Create an Oracle Database instance and create one connection.
    drv <- dbDriver("Oracle")

    ## Use OS authentication as an example of external authentication
    ## Make sure that database user exist to allow an OS authentication

    ## Create connection authenticated with external credentials.
    con <- dbConnect(drv, username ="", password="",
                     external_credentials = TRUE)

    ## Above dbConnect() used OS credentials to connect with database.

    ## Run a SQL statement by creating first a resultSet object.
    rs <- dbSendQuery(con, "select * from emp where deptno = 10")

    ## We now fetch records from the resultSet into a data.frame.
    data <- fetch(rs)       ## extract all rows
    dim(data)

## End(Not run)
```

`dbDriver-methods` *Oracle Implementation of the Database Interface (DBI) Classes and Drivers*

Description

Oracle driver initialization and closing.

Usage

```
## S4 method for signature 'OraDriver'
dbUnloadDriver(drv, ...)
## S4 method for signature 'ExtDriver'
dbUnloadDriver(drv, ...)
```

Arguments

drv	An object that inherits from OraDriver or ExtDriver as created by dbDriver.
...	Any other arguments to pass to the driver drvName.

Details

dbDriver This object is a singleton, that is, subsequent invocations of dbDriver return the same initialized object.

This implementation allows you to connect to multiple host servers and run multiple connections on each server simultaneously.

When interruptible is set to TRUE, it allows for interrupting long-running queries on the server by executing the query in a thread. Main thread checks for Ctrl-C and issues OCIBreak/OCIReset to cancel the operation on the server. By default, interruptible is FALSE.

When unicode_as_utf8 is set to FALSE, NCHAR, NVARCHAR and NCLOB data is fetched using the character set specified by the NLS_LANG setting. By default, unicode_as_utf8 is set to TRUE.

When ora.attributes is set to TRUE, the result set from dbGetQuery and fetch contains DBMS-specific attributes like ora.encoding, ora.type, and ora.maxlength for the corresponding column.

dbUnloadDriver This implementation removes communication links between the R client and the database. It frees all connections and all result sets associated with those connection objects.

Value

dbDriver	An object OraDriver or ExtDriver whose class extends DBIDriver. This object is used to create connections, using the function dbConnect, to one or more Oracle Database engines.
dbUnloadDriver	Free all resources occupied by the driver object.

Side Effects

dbDriver The R client part of the database communication is initialized, but note that connecting to the database engine needs to be done through calls to dbConnect.

dbUnloadDriver Remove the communication link between the R client and the database.

References

For Oracle Database documentation, see https://docs.oracle.com/en/.

See Also

Oracle, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbListTables, dbReadTable.

Examples

```
## Not run:
  # first load the library
  library("ROracle")

  # create an Oracle instance
  drv <- dbDriver("Oracle")

  con <- dbConnect(drv, "scott", "tiger")
  dbListTables(con)

  # fetch all results from the emp table
  res <- dbGetQuery(con, "select * from emp")

  # dbSendQuery can be used to fetch data in chunks
  # as well as all of data at once
  res <- dbSendQuery(con, "select * from emp")

  # fetch all results
  fetch(res)

  # or a chunk at a time
  fetch(res, n = 5)

  # describing the schema for the emp table using dbGetInfo
  dbGetInfo(res, what = 'fields')

  # clear the result
  dbClearResult(res)

  # disconnect from the database
  dbDisconnect(con)

  # free the driver object
  dbUnloadDriver(drv)

## End(Not run)
```

---

dbGetInfo-methods *Database interface Metadata*

Description

These methods are straight-forward implementations of the corresponding generic functions.

Usage

```
## S4 method for signature 'OraDriver'
dbGetInfo(dbObj, ...)
## S4 method for signature 'ExtDriver'
dbGetInfo(dbObj, ...)
## S4 method for signature 'OraConnection'
dbGetInfo(dbObj, what, ...)
## S4 method for signature 'OraResult'
dbGetInfo(dbObj, what, ...)
## S4 method for signature 'OraResult'
dbGetStatement(res, ...)
## S4 method for signature 'OraResult'
dbGetRowCount(res, ...)
## S4 method for signature 'OraResult'
dbGetRowsAffected(res, ...)
## S4 method for signature 'OraResult'
dbColumnInfo(res, ...)
## S4 method for signature 'OraResult'
dbHasCompleted(res)
```

Arguments

dbObj	Any object that implements some functionality in the R interface to databases (a driver, a connection, or a result set).
what	A character string specifying an element of the output list.
res	An OraResult.
...	Currently unused.

Details

Table, schema, and column names are case sensitive, for example, table names ABC and abc are not the same. All database schema object names should not include double quotes as they are enclosed in double quotes when the corresponding SQL statement is generated.

The **ROracle** method dbGetInfo provides following details about the driver object:

- $driverName The name of the driver, "Oracle (OCI)"
- $driverVersion The version of the ROracle driver used
- $clientVersion The version of the Oracle Client library used
- $conTotal The number of connections instantiated by the driver
- $conOpen The number of connections open currently
- $interruptible TRUE when a long-running query can be interrupted
- $unicode_as_utf8 TRUE when character data is to be fetched in UTF8 encoded format
- $ora_attributes TRUE when each column in a result set data frame has corresponding Oracle DBMS attributes

- $connections Information about each connection currently open, see `dbGetInfo` of connection object for details

The **ROracle** method `dbGetInfo` provides following the details about the connection object:

- $username The name of the user on this connection
- $dbname The connect alias or the connect string used
- $serverVersion The version of the Oracle Database server
- $serverType The value "Oracle RDBMS"
- $resTotal The number of result sets on this connection
- $resOpen The number of result sets open on this connection
- $prefetch TRUE when prefetch mechanism is used to fetch data
- $bulk_read The number of rows to fetch at a time from DBMS
- $bulk_write The number of rows to write at a time to DBMS
- $stmt_cache TRUE when the statement cache is used
- $results Information about each result set currently open, see `dbGetInfo` of result set for details

The **ROracle** method `dbGetInfo` provides the following details about the result set object:

- $statement SQL statement used to produce the result set object
- $isSelect TRUE when a select statement is specified
- $rowsAffected The number of rows affected by DML statement
- $rowCount The number of rows in result set currently
- $completed TRUE if there are no more rows in the result set
- $prefetch TRUE when the prefetch mechanism used to fetch data
- $bulk_read The number of rows to fetch at a time from DBMS
- $bulk_write The number of rows to write at a time to DBMS
- $fields Information about each column in the result set, see `dbColumnInfo` for details

The **ROracle** method `dbColumnInfo` provides following details about each column in the result set:

- $name The name of the column
- $Sclass The R type of the object containing the data returned by the Oracle RDBMS
- $type The type of column as created in Oracle RDBMS
- $len Length of VARCHAR, CHAR and RAW column type in Oracle RDBMS. All other columns will have NA.
- $precision The precision of number column
- $scale The scale of number column
- $nullOK TRUE when a NULL value can be present in the column

The example below shows the driver, connection, result set, and column information for a table containing:

```
   create table foo(
     a number(21),
     b number,
     c char(20),
     d varchar(300),
     e binary_double,
     f binary_float,
     g clob,
     h blob,
     i bfile,
     j date,
     m timestamp,
     n timestamp with time zone,
     o timestamp with local time zone,
     r interval day to second,
     s raw(234),
     t boolean,
     u vector(*,*)
   );


   library(ROracle)
Loading required package: DBI
> # instantiate ROracle driver object
> drv <- Oracle()
> con <- dbConnect(drv, "scott", "tiger")
> rs <- dbSendQuery(con, "select * from foo")
> dbGetInfo(drv)
$driverName
[1] "Oracle (OCI)"

$driverVersion
[1] "1.5-1"

$clientVersion
[1] "23.6.0.24.10"

$conTotal
[1] 1

$conOpen
[1] 1

$interruptible
[1] FALSE

$unicode_as_utf8
[1] TRUE
```

```
$ora_attributes
[1] TRUE

$connections
$connections[[1]]
User name:             scott
Connect string:
Server version:        23.6.0.24.10
Server type:           Oracle RDBMS
Results processed:     1
OCI prefetch:          FALSE
Bulk read:             1000
Bulk write:            1000
Statement cache size:  0
Open results:          1


> dbGetInfo(con)
$username
[1] "scott"

$dbname
[1] ""

$serverVersion
[1] "23.6.0.24.10"

$serverType
[1] "Oracle RDBMS"

$resTotal
[1] 1

$resOpen
[1] 1

$prefetch
[1] FALSE

$bulk_read
[1] 1000

$bulk_write
[1] 1000

$stmt_cache
[1] 0
```

```
$results
$results[[1]]
Statement:            select * from foo
Rows affected:        0
Row count:            0
Select statement:     TRUE
Statement completed:  FALSE
OCI prefetch:         FALSE
Bulk read:            1000
Bulk write:           1000


> dbGetInfo(rs)
$statement
[1] "select * from foo"

$isSelect
[1] TRUE

$rowsAffected
[1] 0

$rowCount
[1] 0

$completed
[1] FALSE

$prefetch
[1] FALSE

$bulk_read
[1] 1000

$bulk_write
[1] 1000

$fields
   name    Sclass                        type len precision scale nullOK
1     A   numeric                      NUMBER  NA        21     0   TRUE
2     B   numeric                      NUMBER  NA         0  -127   TRUE
3     C character                        CHAR  20         0     0   TRUE
4     D character                    VARCHAR2 300         0     0   TRUE
5     E   numeric                BINARY_DOUBLE  NA         0     0   TRUE
6     F   numeric                 BINARY_FLOAT  NA         0     0   TRUE
7     G character                        CLOB  NA         0     0   TRUE
8     H       raw                        BLOB  NA         0     0   TRUE
9     I       raw                       BFILE  NA         0     0   TRUE
```

```
10    J   POSIXct                          DATE  NA         0     0   TRUE
11    M   POSIXct                     TIMESTAMP  NA         0     6   TRUE
12    N   POSIXct       TIMESTAMP WITH TIME ZONE  NA         0     6   TRUE
13    O   POSIXct TIMESTAMP WITH LOCAL TIME ZONE  NA         0     6   TRUE
14    R  difftime         INTERVAL DAY TO SECOND  NA         2     6   TRUE
15    S       raw                           RAW 234         0     0   TRUE
16    T   logical                       BOOLEAN  NA         0     0   TRUE
17    U      list                     VECTOR(*)  NA         0     0   TRUE

>
```

Value

Information about driver, connection or a result set object.

References

For the Oracle Database documentaion see https://docs.oracle.com/en/.

See Also

Oracle, dbDriver, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo, dbListTables, dbReadTable.

Examples

```
## Not run:
  drv <- dbDriver("Oracle")
  con <- dbConnect(drv, "scott", "tiger")

  rs <- dbSendQuery(con, "select * from emp")

  # Get the SQL statement for the result set object rs
  dbGetStatement(rs)

  # Are there any more rows in result set?
  dbHasCompleted(rs)

  # Information about columns in result set rs object
  dbColumnInfo(rs)

  # DBIDriver info
  names(dbGetInfo(drv))

  # DBIConnection info
  names(dbGetInfo(con))

  # DBIResult info
  names(dbGetInfo(rs))

## End(Not run)
```

dbListConnections-methods
List items from Oracle objects

Description

These methods are straight-forward implementations of the corresponding generic functions.

Usage

```
## S4 method for signature 'OraDriver'
dbListConnections(drv, ...)
## S4 method for signature 'ExtDriver'
dbListConnections(drv, ...)
## S4 method for signature 'OraConnection'
dbListResults(conn, ...)
```

Arguments

drv	an OraDriver or ExtDriver.
conn	an OraConnection.
...	currently unused.

Details

dbListConnections implementation return a list of all the associated connections. It shows information about all the associated connections.

dbListResults implementation return a list of all associated result sets. It shows information about all associated result sets.

Value

dbListConnections	A list of all connections associated with driver.
dbListResults	A list of all result sets associated with connection.

References

For the Oracle Database documentation see https://docs.oracle.com/en/.

See Also

Oracle, dbGetInfo, dbColumnInfo, dbDriver, dbConnect, dbSendQuery

Examples

```
## Not run:
  drv <- dbDriver("Oracle")
  con1 <- dbConnect(drv, "scott", "tiger")
  res1 <- dbSendQuery(con1, "select * from emp where deptno = 10")
  res2 <- dbSendQuery(con1, "select * from emp where deptno = 20")
  con2 <- dbConnect(drv, "scott", "tiger")
  res3 <- dbSendQuery(con2, "select * from dept")

  ## get all active statements
  for(con in dbListConnections(drv))
    for (res in dbListResults(con))
      print(dbGetStatement(res))

## End(Not run)
```

---

dbReadTable-methods *Convenience Functions for Manipulating DBMS Tables*

Description

These functions mimic their R counterparts except that they generate code that gets remotely executed in a database engine: `get`, `assign`, `exists`, `remove`, `objects`, and `names`.

Usage

```
## S4 method for signature 'OraConnection,character'
dbReadTable(conn, name, schema = NULL,
               row.names = NULL, sparse = FALSE, ...)
## S4 method for signature 'OraConnection,character,data.frame'
dbWriteTable(conn, name, value,
          row.names = FALSE, overwrite = FALSE, append = FALSE, ora.number = TRUE,
               schema = NULL, date = FALSE, sparse = FALSE, ...)
## S4 method for signature 'OraConnection,character'
dbExistsTable(conn, name, schema = NULL, ...)
## S4 method for signature 'OraConnection,character'
dbRemoveTable(conn, name, purge = FALSE,
               schema = NULL, ...)
## S4 method for signature 'OraConnection'
dbListTables(conn, schema = NULL, all = FALSE, full = FALSE, ...)
## S4 method for signature 'OraConnection,character'
dbListFields(conn, name, schema = NULL, ...)
```

Arguments

conn	An `OraConnection` database connection object.
name	A case-sensitive character string specifying a table name.

schema	A case-sensitive character string specifying a schema name (or a vector of character strings for dbListTables).
date	A boolean flag to indicate whether to use date or DateTimep. By default, DateTime will be used instead of timestamp.
row.names	In the case of dbReadTable, this argument can be a string, an index or a logical vector specifying the column in the DBMS table to be used as row.names in the output data.frame (a NULL specifies that no column should be used as row.names in the output). The default is NULL. In the case of dbWriteTable, this argument should be a logical value specifying whether the row.names should be output to the output DBMS table; if TRUE, an extra column whose name is "row.names" will be added to the output. The default is FALSE.
value	A data.frame containing the data to write to a table. (See Details section for supported column types.)
overwrite	A logical value specifying whether to overwrite an existing table or not. The default is FALSE.
append	A logical value specifying whether to append to an existing table in the DBMS. The default is FALSE.
ora.number	A logical value specifying whether to create a table with Oracle NUMBER or BINARY_DOUBLE columns while writing numeric data. Specify TRUE to create a table with Oracle NUMBER values or specify FALSE to create a table with Oracle BINARY_DOUBLE values. The default value is TRUE. Specify FALSE if one or more of the numeric data values are NaN.
purge	A logical value specifying whether to add the PURGE option to the SQL DROP TABLE statement.
all	A logical value specifying whether to look at all schemas.
full	A logical value specifying whether to generate schema names. When argument all is TRUE, the output is a vector containing schema names followed by the table names. Using matrix(..., ncol = 2) on the output produces a matrix where each row corresponds to a table and the columns represent the schema names and table names respectively.
sparse	A logical indicating whether to use sparseVector method from Matrix library to construct sparse vectors returned by Oracle database. When TRUE, sparse vectors are constructed using sparseVector method of Matrix package. When FALSE, dense vector is returned and one can use any of the R methods/packages to transform to sparse format. For dbWriteTable, specifying sparse=TRUE will create the vector column in the database as a sparse vector type, otherwise a dense vector type is created.
...	currently unused.

Details

Table, schema, and column names are case sensitive, e.g., table names ABC and abc are not the same. All database schema object names should not include double quotes as they are enclosed in double quotes when the corresponding SQL statement is generated.

The following attributes are used for correctly mapping BLOB, CLOB, NCLOB, NCHAR, VARCHAR2, NVARCHAR2, CHAR, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE and VECTOR columns in dbWriteTable:

- 1) Attribute Name: ora.type This attribute indicates the type of the underlying column and can be "CLOB", "BLOB", "CHAR", "VARCHAR2", "RAW", or "VECTOR". The user can specify TIMESTAMP, DATE, TIMESTAMP WITH TIME ZONE or any other column types supported by Oracle Database. ROracle does not parse the value; it is validated by the database. The user can provide one of the following values for ora.type: CLOB, BLOB, CHAR, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE, VECTOR and fractional_seconds_precision.
- 2) Attribute Name: ora.encoding When UTF-8 is specified, the database uses NCLOB, NCHAR or NVARCHAR based on ora.type.
- 3) Attribute Name: ora.maxlength One can specify the maximum length of CHAR, VARCHAR, NCHAR, NVARCHAR2, or RAW columns. For other data types, ora.maxlength does not apply and is ignored. The following default values are used for certain data types when ora.maxlength is not specified. CHAR 2000 NCHAR 1000 VARCHAR2 4000 NVARCHAR2 2000 RAW 2000
- 4) Attribute Name: ora.format For vector data type when ora.maxlength and ora.format are not specified flex dimension and format are used. Type VECTOR(*) column info indicates flex format and length NA denotes flex dimension.
- 5) Attribute Name: ora.fractional_seconds_precision One can specify the fractional part of the SECOND datetime field of TIMESTAMP, TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE columns. It can be a number in the range 0 to 9. The default value is 6.

ROracle methods such as dbReadTable, dbGetQuery, fetch, and dbWriteTable use the following mapping between R and Oracle data types:

- logical and integer map to Oracle INTEGER
- numeric maps to Oracle NUMBER if argument ora.number is TRUE or Oracle BINARY_DOUBLE if FALSE
- character maps to Oracle CLOB if attribute ora.type is "CLOB" or Oracle NCLOB if attribute ora.type is "CLOB" and ora.encoding is "UTF-8" or Oracle CHAR(ora.maxlength) if attribute ora.type is "CHAR" or Oracle NCHAR(ora.maxlength) if attribute ora.type is "CHAR" and ora.encoding is "UTF-8" or Oracle NVARCHAR2(ora.maxlength) if attribute ora.encoding is "UTF-8" or Oracle VARCHAR2(ora.maxlength)
- Date and POSIXct map to Oracle DATE **ROracle** - the ROracle package R - the R application
- POSIXct maps to Oracle TIMESTAMP WITH TIME ZONE(ora.fractional_seconds_precision) if attribute ora.type is "TIMESTAMP WITH TIME ZONE" or Oracle TIMESTAMP WITH LOCAL TIME ZONE(ora.fractional_seconds_precision) if attribute ora.type is "TIMESTAMP WITH LOCAL TIME ZONE" or Oracle TIMESTAMP(ora.fractional_seconds_precision) and if Date is FALSE
- difftime maps to Oracle INTERVAL DAY TO SECOND
- list of raw vectors map to Oracle BLOB if attribute ora.type is "BLOB" or Oracle RAW(ora.maxlength)
- list of raw, numeric, character, integer vectors map to Oracle VECTOR if attribute ora.type is "VECTOR" or Oracle RAW(ora.maxlength) CHAR(ora.maxlength), NUMBER(ora.maxlength)

- other R types such as `factor` are converted to `character`

ROracle returns values from database columns that are of data type: date, time stamp, time stamp with time zone and time stamp with local time zone data types in R's POSIXct format. POSIXct refers to a time that is internally stored as the number of seconds since the start of 1970 in UTC. Number of seconds are exchanged from R and ROracle driver in floating point double format. In POSIXct representation R uses the TZ environment variable or maps the OS time zone environment variable to its own, therefore the date will be displayed in this time zone format.

One can insert data into columns of the four data types listed above using a string with the correct format or POSIXct representation. String data is passed to the database directly and ROracle relies on databse to convert it to date time representation. ROracle driver converts the POSIXct representation to a string representation using the format "%Y-%m-%d %H:%M:%OS6" in a data frame that is used for DML operations. Data in this format corresponds to NLS_TIMESTAMP_TZ_FORMAT "YYYY-MM-DD HH24:MI:SSXFF" and is converted to SQLT_TIMESTAMP_LTZ to be bound to the Oracle database. An intermediate class "datetime" is created that represents character data to the ROracle driver internally.

Columns having a date and time stamp data type are fetched by ROracle using the SQLT_TIMESTAMP data type. Columns having a time stamp with time zone or a time stamp with local time zone data types are fetched using SQLT_TIMESTAMP_LTZ data type. Columns of data type time stamp with local time zone undergo conversion to the session time zone that the R application runs in, therefore setting the time zone environment TZ in R will affect the data values in this column. ROracle driver maps the TZ environment variable to the session time zone and issues an alter DDL to set the session time zone when the connection is made to the database.

To fetch data from columns with a timestamp with time zone or a timestamp with local time zone, the client and server must have the same time zone data file else an error will be reported.

When these data types are selected from the database, they are converted to string representation using the NLS_TIMESTAMP_TZ_FORMAT "YYYY-MM-DD HH24:MI:SSXFF" that corresponds to "%Y-%m-%d %H:%M:%OS6" in R. An intermediate class "datetime" is created to represent this character data in ROracle driver. ROracle driver then converts it to POSIXct using the as.POSIXct() function. An R application sees the data in POSIXct form in the data frame.

```
R session time zone:
 R has the concept of a time zone in which the R engine operates. The time
 zone can be set to a string such as 'PST8PDT', 'America/Los_Angeles' and so on.
 These strings are self-explanatory and specify the time zone in which the
 session is operating.
 The R session time zone can be set in one of two ways:
 1. By entering the following on the Linux or Unix command line before starting
    R:
    setenv TZ = America/Los_Angeles on Linux/UNIX
    NOTE: Do not use this option on Windows as it does not allow one to
          set Oracle compatible timezone names for the environment variable TZ.
 2. By entering the following at the R prompt:
    Sys.setenv(TZ = "PST8PDT")

 We recommend using the option 2 as the R script works without any
 porting issues on Linux/Unix as well as Windows. Option 2 also allows you
 to specify Oracle compatible timezone names even on Windows.
```

```
 The R session time zone determines the time zone for all POSIXct time
 zone unqualified date-time types. It is also the time zone to which all
 qualified date-time types are converted when they are displayed by R.

 The following example demonstrates this.
 Sys.setenv(TZ = "PST8PDT")
 dt <- c(as.POSIXct("2010/3/13", tz = "PST8PDT"),
         as.POSIXct("2010/3/13 3:47:30.123456", tz = "PST8PDT"),
         as.POSIXct("2010/3/22", tz = "PST8PDT"),
         as.POSIXct("2010/3/22 7:02:30", tz = "PST8PDT"),
         as.POSIXct("2010/3/13"),
         as.POSIXct("2010/3/13 3:47:30.123456"),
         as.POSIXct("2010/3/22"),
         as.POSIXct("2010/3/22 7:02:30"))
 dt
 [1] "2010-03-13 00:00:00.000000 PST" "2010-03-13 03:47:30.123456 PST"
 [3] "2010-03-22 00:00:00.000000 PDT" "2010-03-22 07:02:30.000000 PDT"
 [5] "2010-03-13 00:00:00.000000 PST" "2010-03-13 03:47:30.123456 PST"
 [7] "2010-03-22 00:00:00.000000 PDT" "2010-03-22 07:02:30.000000 PDT"

Note that the unqualified timestamps are also assumed to be in the R's
session time zone when they are displayed by R. Of course, R is also smart
enough to make the determination of whether the time falls into PST or PDT
based on when US Daylight savings begins, and displays PST or PDT
accordingly.

The following example makes this more obvious.
> Sys.setenv(TZ = "EST5EDT")
> dt <- c(as.POSIXct("2010/3/13", tz = "PST8PDT"),
+         as.POSIXct("2010/3/13 3:47:30.123456", tz = "PST8PDT"),
+         as.POSIXct("2010/3/22", tz = "PST8PDT"),
+         as.POSIXct("2010/3/22 7:02:30", tz = "PST8PDT"),
+         as.POSIXct("2010/3/13"),
+         as.POSIXct("2010/3/13 3:47:30.123456"),
+         as.POSIXct("2010/3/22"),
+         as.POSIXct("2010/3/22 7:02:30"))

> dt
[1] "2010-03-13 03:00:00.000000 EST" "2010-03-13 06:47:30.123456 EST"
[3] "2010-03-22 03:00:00.000000 EDT" "2010-03-22 10:02:30.000000 EDT"
[5] "2010-03-13 00:00:00.000000 EST" "2010-03-13 03:47:30.123456 EST"
[7] "2010-03-22 00:00:00.000000 EDT" "2010-03-22 07:02:30.000000 EDT"

Note that all the time zone unqualified timestamps are assumed to be in
the session time zone. However, even the time zone qualified timestamps
are converted to session time zone and displayed. Note that all the
values are displayed by R in the R session's time zone (with the
```

```
timezone name also modified to EST or EDT to account for
daylight savings as applicable). Refer to Date-Time Classes at
http://stat.ethz.ch/R-manual/R-devel/library/base/html/DateTimeClasses.html
and timezones at:
http://stat.ethz.ch/R-manual/R-devel/library/base/html/timezones.html
for details on how R handles dates and times and time zones)
Let's take an example where we use a longer time zone name
(often referred to as an 'Olson Name') as opposed to an abbreviation.
> Sys.setenv(TZ = "America/Los_Angeles")
> dt <- c(as.POSIXct("2010/3/13", tz = "PST8PDT"),
+         as.POSIXct("2010/3/13 3:47:30.123456", tz = "PST8PDT"),
+         as.POSIXct("2010/3/22", tz = "PST8PDT"),
+         as.POSIXct("2010/3/22 7:02:30", tz = "PST8PDT"),
+         as.POSIXct("2010/3/13"),
+         as.POSIXct("2010/3/13 3:47:30.123456"),
+         as.POSIXct("2010/3/22"),
+         as.POSIXct("2010/3/22 7:02:30"))
> dt
[1] "2010-03-13 00:00:00.000000 PST" "2010-03-13 03:47:30.123456 PST"
[3] "2010-03-22 00:00:00.000000 PDT" "2010-03-22 07:02:30.000000 PDT"
[5] "2010-03-13 00:00:00.000000 PST" "2010-03-13 03:47:30.123456 PST"
[7] "2010-03-22 00:00:00.000000 PDT" "2010-03-22 07:02:30.000000 PDT"

Note that in such a case, R doesn't use the long name when the
values are displayed, but instead still displays the values using
the abbreviations "PST" and "PDT". This is significant because Oracle
doesn't necessarily like these abbreviations. For example, an Oracle
databse doesn't recognize "PDT" as a valid time zone. See "R Time zone
and Oracle session time zone" for details on valid time zones.
```

The example below shows the effect of changing the time zone in R environment:

```
R> Sys.timezone()
[1] "PST8PDT"
# Selecting data and displaying it
res <- dbGetQuery(con, selStr)
R>     res[,1]
[1] 1 2 3 4 5 6
R>     res[,2]
[1] "2012-06-05 00:00:00 PDT" "2012-01-05 07:15:02 PST"
    "2012-01-05 00:00:00 PST" "2011-01-05 00:00:00 PST"
[5] "2013-01-05 00:00:00 PST" "2020-01-05 00:00:00 PST"
R>     res[,3]
[1] "2012-06-05 00:00:00 PDT" "2012-01-05 07:15:03 PST"
    "2012-01-05 00:00:00 PST" "2011-01-05 00:00:00 PST"
[5] "2013-01-05 00:00:00 PST" "2020-01-05 00:00:00 PST"
R>     res[,4]
[1] "2012-06-05 00:00:00 PDT" "2012-01-05 07:15:03 PST"
    "2012-01-05 00:00:00 PST" "2011-01-05 00:00:00 PST"
```

```
[5] "2013-01-05 00:00:00 PST" "2020-01-05 00:00:00 PST"
R>     res[,5]
[1] "2012-06-05 00:00:00 PDT" "2012-01-05 07:15:03 PST"
    "2012-01-05 00:00:00 PST" "2011-01-05 00:00:00 PST"
[5] "2013-01-05 00:00:00 PST" "2020-01-05 00:00:00 PST"
R> Sys.setenv(TZ='EST5EDT')
R>     res[,1]
[1] 1 2 3 4 5 6
R>     res[,2]
[1] "2012-06-05 03:00:00 EDT" "2012-01-05 10:15:02 EST"
    "2012-01-05 03:00:00 EST" "2011-01-05 03:00:00 EST"
[5] "2013-01-05 03:00:00 EST" "2020-01-05 03:00:00 EST"
R>     res[,3]
[1] "2012-06-05 03:00:00 EDT" "2012-01-05 10:15:03 EST"
    "2012-01-05 03:00:00 EST" "2011-01-05 03:00:00 EST"
[5] "2013-01-05 03:00:00 EST" "2020-01-05 03:00:00 EST"
R>     res[,4]
[1] "2012-06-05 03:00:00 EDT" "2012-01-05 10:15:03 EST"
    "2012-01-05 03:00:00 EST" "2011-01-05 03:00:00 EST"
[5] "2013-01-05 03:00:00 EST" "2020-01-05 03:00:00 EST"
R>     res[,5]
[1] "2012-06-05 03:00:00 EDT" "2012-01-05 10:15:03 EST"
    "2012-01-05 03:00:00 EST" "2011-01-05 03:00:00 EST"
[5] "2013-01-05 03:00:00 EST" "2020-01-05 03:00:00 EST"
```

Also `dbWriteTable` always auto commits a current transaction as well as the data it inserts, i.e. it acts as a DDL statement even if appends rows to an already existing table.

Value

A `data.frame` in the case of `dbReadTable`; a `vector` in the case of `dbListTables` and `dbListFields`; a `logical` in the case of `dbExistsTable` indicating whether the table exists; otherwise `TRUE` when the operation was successful or an exception.

References

For the Oracle Database documentation see https://docs.oracle.com/en/. For Datetime Data Types and Time Zone Support in Oracle see https://docs.oracle.com/cd/E11882_01/server.112/e10729/ch4datetime.htm.

See Also

Oracle, dbDriver, dbConnect, dbSendQuery, dbGetQuery, fetch, dbCommit, dbGetInfo.

Examples

```
## Not run:
  con <- dbConnect(Oracle(), "scott", "tiger")
  if (dbExistsTable(con, "FOO", "SCOTT"))
    dbRemoveTable(con, "FOO")
```

```
foo <- dbReadTable(con, "EMP")
row.names(foo) <- foo$EMPNO
foo <- foo[,-1]

dbWriteTable(con, "FOO", foo, row.names = TRUE)
dbWriteTable(con, "FOO", foo, row.names = TRUE, overwrite = TRUE)
dbReadTable(con, "FOO", row.names = 1)

dbGetQuery(con, "delete from foo")
dbWriteTable(con, "FOO", foo, row.names = TRUE, append = TRUE)
dbReadTable(con, "FOO", row.names = 1)
dbRemoveTable(con, "FOO")

dbListTables(con)
dbListFields(con, "EMP")

if (dbExistsTable(con, "RORACLE_TEST", "SCOTT"))
  dbRemoveTable(con, "RORACLE_TEST")

# Example of POSIXct usage.
# A table is created using:
createTab <- "create table RORACLE_TEST(row_num number, id1 date,
             id2 timestamp, id3 timestamp with time zone,
             id4 timestamp with local time zone )"

dbGetQuery(con, createTab)
# Insert statement.
insStr <- "insert into RORACLE_TEST values(:1, :2, :3, :4, :5)";

# Select statement.
selStr <- "select * from RORACLE_TEST";

# Insert time stamp without time values in POSIXct form.
x <- 1;
y <- "2012-06-05";
y <- as.POSIXct(y);
dbGetQuery(con, insStr, data.frame(x, y, y, y, y));

# Insert date & times stamp with time values in POSIXct form.
x <- 2;
y <- "2012-01-05 07:15:02";
y <- as.POSIXct(y);
z <- "2012-01-05 07:15:03.123";
z <- as.POSIXct(z);
dbGetQuery(con, insStr, data.frame(x, y, z,  z, z));

# Insert list of date objects in POSIXct form.
x <- c(3, 4, 5, 6);
y <- c('2012-01-05', '2011-01-05', '2013-01-05', '2020-01-05');
y <- as.POSIXct(y);
dbGetQuery(con, insStr, data.frame(x, y, y, y, y));
```

```
    dbCommit (con)

    # Selecting data and displaying it.
    res <- dbGetQuery(con, selStr)
    res[,1]
    res[,2]
    res[,3]
    res[,4]
    res[,5]

    # insert data in Date format
    a<-as.Date("2014-01-01")
    dbWriteTable(con, 'TEMP', data.frame(a), date = TRUE)

    # using attribute to map NCHAR, CLOB, BLOB, NCLOB columns correctly in
    # dbWriteTable
    str1 <- paste(letters, collapse="")
    lstr1 <- paste(rep(str1, 200), collapse="")
    raw.lst <- vector("list",1)
    lraw.lst <- vector("list",1)
    raw.lst[[1L]] <- charToRaw(str1)
    lraw.lst[[1L]] <- rep(charToRaw(str1), 200)
    a <- as.POSIXct("2014-01-01 14:12:09.0194733")
    b <- as.POSIXct("2014-01-01 14:12:09.01947")
    test.df <- data.frame(char=str1, nchar=str1, varchar=str1, clob=lstr1,
                     nclob=lstr1, stringsAsFactors=FALSE)
    test.df$raw.typ <- raw.lst
    test.df$blob <- lraw.lst
    test.df$char_max <- str1
    test.df$raw_max.typ <- raw.lst
    test.df$nvchar <- str1
    test.df$nvchar_max <- str1
    test.df$date_tz <- a
    test.df$date_ltz <- b

    # adding attributes
    attr(test.df$clob, "ora.type") <- "CLOB"
    attr(test.df$blob, "ora.type") <- "BLOB"
    attr(test.df$nclob, "ora.type") <- "CLOB"
    attr(test.df$nclob, "ora.encoding") <- "UTF-8"
    attr(test.df$char_max, "ora.maxlength") <- 3000
    attr(test.df$raw_max.typ, "ora.maxlength") <- 1000
    attr(test.df$nvchar, "ora.encoding") <- "UTF-8"
    attr(test.df$nvchar_max, "ora.encoding") <- "UTF-8"
    attr(test.df$nvchar_max, "ora.maxlength") <- 1500
    attr(test.df$char, "ora.type") <- "CHAR"
    attr(test.df$date_tz, "ora.type") <- "timestamp with time zone"
    attr(test.df$date_ltz, "ora.type") <- "timestamp with local time zone"
    attr(test.df$nchar, "ora.type") <- "CHAR"
    attr(test.df$nchar, "ora.encoding") <- "UTF-8"
    attr(test.df$date_tz, "ora.fractional_seconds_precision") <- 9
R> # displaying the data frame
R> test.df
```

```
char                        nchar
1 abcdefghijklmnopqrstuvwxyz abcdefghijklmnopqrstuvwxyz
varchar
1 abcdefghijklmnopqrstuvwxyz
clob
1 abcdefghijklmnopqrstuvwxyz...
nclob
1 abcdefghijklmnopqrstuvwxyz...
raw.typ
1 61, 62, 63, 64, 65, 66, 67, 68, 69, 6a, 6b, 6c, 6d, 6e, 6f, 70, 71, 72, 73,
  74, 75, 76, 77, 78, 79, 7a
blob
1 61, 62, 63, 64, 65, 66, 67, 68, 69, 6a, 6b, 6c, 6d, 6e, 6f, 70, 71, 72, 73,
  74, 75, 76, 77, 78, 79, 7a,...
char_max
1 abcdefghijklmnopqrstuvwxyz
raw_max.typ
1 61, 62, 63, 64, 65, 66, 67, 68, 69, 6a, 6b, 6c, 6d, 6e, 6f, 70, 71, 72, 73,
  74, 75, 76, 77, 78, 79, 7a
nvchar                      nvchar_max
1 abcdefghijklmnopqrstuvwxyz abcdefghijklmnopqrstuvwxyz
date_tz                     date_ltz
1 2014-01-01 14:12:09.019473 2014-01-01 14:12:09.01946

    dbWriteTable(con, name="TEST_TAB", value=test.df)
    res <- dbReadTable(con, name="TEST_TAB")
R> res
char
1 abcdefghijklmnopqrstuvwxyz
nchar
1 abcdefghijklmnopqrstuvwxyz
varchar
1 abcdefghijklmnopqrstuvwxyz
clob
1 abcdefghijklmnopqrstuvwxyz...
nclob
1 abcdefghijklmnopqrstuvwxyz...
raw.typ
1 61, 62, 63, 64, 65, 66, 67, 68, 69, 6a, 6b, 6c, 6d, 6e, 6f, 70, 71, 72, 73,
  74, 75, 76, 77, 78, 79, 7a
blob
1 61, 62, 63, 64, 65, 66, 67, 68, 69, 6a, 6b, 6c, 6d, 6e, 6f, 70, 71, 72, 73,
  74, 75, 76, 77, 78, 79, 7a,...
char_max
1 abcdefghijklmnopqrstuvwxyz
raw_max.typ
1 61, 62, 63, 64, 65, 66, 67, 68, 69, 6a, 6b, 6c, 6d, 6e, 6f, 70, 71, 72, 73,
  74, 75, 76, 77, 78, 79, 7a
nvchar                      nvchar_max
1 abcdefghijklmnopqrstuvwxyz abcdefghijklmnopqrstuvwxyz
date_tz                     date_ltz
1 2014-01-01 14:12:09.019473 2014-01-01 14:12:09.01946
```

```
## End(Not run)
  ## Not run:
    df <- data.frame(A=c(0,1,NaN,4), B=c(NA, 2,3,NaN))
    con <- dbConnect(Oracle(), "scott", "tiger")
    dbWriteTable(con,"TEST", df, row.names = FALSE, ora.number=FALSE)

## End(Not run)
```

---

dbSendQuery-methods *Execute a Statement on a Given Database Connection*

Description

These methods are straight-forward implementations of the corresponding generic functions except for the execute method, which is an ROracle specific DBI extension.

Usage

```
## S4 method for signature 'OraConnection,character'
dbSendQuery(conn, statement, data = NULL,
          prefetch = FALSE, bulk_read = 1000L, bulk_write = 1000L,
          sparse=FALSE, ...)
## S4 method for signature 'OraConnection,character'
dbGetQuery(conn, statement, data = NULL,
          prefetch = FALSE, bulk_read = 1000L, bulk_write = 1000L,
          sparse=FALSE, ...)
## S4 method for signature 'OraConnection,character'
oracleProc(conn, statement, data = NULL,
          prefetch = FALSE, bulk_read = 1000L, bulk_write = 1000L,
          sparse=FALSE, ...)
## S4 method for signature 'OraResult'
dbClearResult(res, ...)
## S4 method for signature 'OraConnection'
dbGetException(conn, ...)

execute(res, ...)
## S4 method for signature 'OraResult'
execute(res, data = NULL, ...)
```

Arguments

conn	An OraConnection object.
statement	A character vector of length 1 with the SQL statement.
res	An OraResult object.
data	A data.frame specifying bind data

`prefetch` A logical value that specifies whether ROracle uses prefetch buffers or an array fetch to retrieve data from the server. If `TRUE`, then ROracle uses OCI prefetch buffers to retrieve additional data from the server thus saving the memory required in RODBI/ROOCI by allocating a single row buffer to fetch the data from OCI. Using prefetch results in a fetch call for every row. If `FALSE` (the default), then ROracle uses an array fetch to retrieve the data.

`bulk_read` An integer value indicating the number of rows to fetch at a time. The default value is 1000L. When the prefetch option is selected, memory is allocated for prefetch buffers and OCI fetches the specified number of rows at a time. When prefetch is not used, which is the default, memory is allocated in RODBI/ROOCI define buffers. Setting this to a large value results in more memory being allocated based on the number of columns in the select list and the types of columns. For a column of type character, define buffers are allocated using the maximum width times the NLS maximum width. An application should adjust this value based on the query result. A larger value benefits queries that return a large result. The application can tune this value as needed.

`bulk_write` An integer value indicating the number of rows to write at a time. The default value is 1000L. When a bulk_write value is specified, memory is allocated for buffers and OCI writes that many rows at a time. If the bulk_write argument is not used, then the default value is used to allocate memory for the bind buffers. Setting `bulk_write` to a large value results in more memory being allocated based on the number of columns in the insert list and the types of columns.

`sparse` A logical indicating whether to use sparseVector method from Matrix library to construct sparse vectors returned by Oracle database. When TRUE, sparse vectors are constructed using sparseVector method of Matrix package. When FALSE, dense vector is returned and one can use any of the R methods/packages to transform to sparse format.

`...` Currently unused.

Details

dbGetQuery This function executes a query statement and fetches the result data from the database. It should not be used for calling PL/SQL queries.

dbSendQuery This function executes a query statement and returns a result set to the application. The application can then perform operations on the result set. It should not be used for calling PL/SQL queries.

oracleProc This function executes a PL/SQL stored procedure or function query statement and returns the result.

dbClearResult This function frees resources used by result set.

dbGetException This function retrieves error information.

execute This function executes the specified query statement.

Value

`dbSendQuery` An `OraResult` object whose class extends `DBIResult`. This object is used to fetch data from a database, using the function `fetch`.

Side Effects

dbGetQuery Query statement is executed and data is fetched from database.

dbSendQuery Query statement is executed, but data needs to be fetched through calls to fetch.

oracleProc PL/SQL stored procedure or function query statement is executed and result is returned.

dbClearResult Resources acquired by the result set are freed.

dbGetException Error information is retrieved and then cleaned from the driver.

execute Query statement is executed.

References

For the Oracle Database documentation see https://docs.oracle.com/en/.

See Also

Oracle, dbDriver, dbConnect, fetch, dbCommit, dbGetInfo, dbReadTable.

Examples

```
## Not run:
  drv <- dbDriver("Oracle")
  con <- dbConnect(drv, "scott", "tiger")
  res <- dbSendQuery(con, "select * from emp where deptno = :1",
                     data = data.frame(deptno = 10))
  data <- fetch(res, n = -1)
  res2 <- dbSendQuery(con, "select * from emp where deptno = :1",
                      data1 <- data.frame(deptno = 10), prefetch=TRUE,
                      bulk_read=2L)
  data1 <- fetch(res2, n = -1)
  data1

  res3 <- dbSendQuery(con, "select * from emp where deptno = :1",
                      data2 <- data.frame(deptno = 10), bulk_read=10L)
  data2 <- fetch(res3, n = -1)
  data2

  res4 <- dbSendQuery(con, "select * from emp where deptno = :1",
                      data3 <- data.frame(deptno = 10), bulk_write=10L)
  data3 <- fetch(res4, n = -1)
  data3

  res5 <- dbSendQuery(con, "select * from emp where ename = :1",
                      data4 <- data.frame(ename = 'SMITH'))
  data4 <- fetch(res5, n = -1)
  data4

## End(Not run)
```

ExtDriver-class *Class ExtDriver*

Description

An Oracle extproc driver class implementing the R database interface (DBI) API.

Generators

The main generators are `dbDriver` and `Extproc`.

Extends

Class `"DBIDriver"`, directly. Class `"DBIObject"`, by class `"DBIDriver"`, distance 2.

Methods

dbConnect `signature(drv = "ExtDriver")`: ...

dbGetInfo `signature(dbObj = "ExtDriver")`: ...

dbListConnections `signature(drv = "ExtDriver")`: ...

dbUnloadDriver `signature(drv = "ExtDriver")`: ...

summary `signature(object = "ExtDriver")`: ...

show `signature(object = "ExtDriver")`

See Also

DBI classes: `OraConnection-class` `OraResult-class`

Examples

```
## Not run:
con <- dbConnect(Extproc())

## End(Not run)
```

fetch-methods *Fetch records from a previously executed query*

Description

This method is a straight-forward implementation of the corresponding generic function.

Usage

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

Arguments

res	an OraResult object.
n	maximum number of records to retrieve per fetch. Use n = -1 to retrieve all pending records.
...	currently unused.

Details

The ROracle implementations retrieves only n records, and if n is missing it returns all records.

Value

number of records fetched from database.

References

For the Oracle Database documentation see https://docs.oracle.com/en/.

See Also

Oracle, dbConnect, dbSendQuery, dbGetQuery, dbClearResult, dbCommit, dbGetInfo, dbReadTable.

Examples

```
## Not run:
  drv <- dbDriver("Oracle")
  con <- dbConnect(drv, "scott", "tiger")
  res <- dbSendQuery(con, "select * from emp")

  # we now fetch the first 10 records from the resultSet into a data.frame
  data1 <- fetch(res, n = 10)
  dim(data1)

  dbHasCompleted(res)
```

```
    # let's get all remaining records
    data2 <- fetch(res, n = -1)

## End(Not run)
```

`Oracle` *Instantiate an Oracle client from the current* R *session*

Description

This function creates and initializes an Oracle client from the current R session. It returns an object that allows you to connect to one or more Oracle servers.

Usage

```
Oracle(interruptible = FALSE, unicode_as_utf8 = TRUE,
       ora.attributes = FALSE, ora.objects = FALSE,
       sparse = FALSE)
Extproc(extproc.ctx = NULL)
```

Arguments

`interruptible`	A logical indicating whether to allow user interrupts on long-running queries.
`extproc.ctx`	An external pointer wrapping extproc context.
`unicode_as_utf8`	A logical indicating whether to fetch NCHAR, NVARCHAR and NCLOB data encoded in UTF8.
`ora.attributes`	A logical indicating whether to include the attributes `ora.encoding`, `ora.type`, and `ora.maxlength` in the data frames returned by `dbGetQuery` and `fetch`.
`ora.objects`	A logical indicating whether to allow access to tables with object types, such as Collections, user-defined types and varrays.
`sparse`	A logical indicating whether to use sparseVector method from Matrix library to construct sparse vectors returned by Oracle database. When TRUE, sparse vectors are constructed using sparseVector method of Matrix package. When FALSE, a dense vector is returned and one can use any of the R methods/packages to transform to sparse format.

Details

This object is a singleton, that is, on subsequent invocations it returns the same initialized object.

This implementation allows you to connect to multiple host servers and run multiple connections on each server simultaneously.

When `interruptible` is set to TRUE, it allows for interrupting long-running queries on the server by executing the query in a thread. Main thread checks for Ctrl-C and issues OCIBreak/OCIReset to cancel the operation on the server. By default `interruptible` is FALSE.

When unicode_as_utf8 is set to FALSE, NCHAR, NVARCHAR and NCLOB data is fetched using the character set using the NLS_LANG setting. By default unicode_as_utf8 is set to TRUE.

When ora.attributes is set to TRUE attributes ora.encoding, ora.type, ora.format and ora.maxlength are added in result data frame returned from dbGetQuery and fetch. ora.maxlength in result data frame represents the dimension of vector column. ora.format in data frame refer to format of the vector column defined in the data base for the vector column. Format will be one of int8, float32, float64, binary or * which represents flex format. It should be used with dbWriteTable to create the same data types as in the Oracle DBMS as fetched from the source table.

Value

An object of class OraDriver for Oracle or ExtDriver for Extproc whose class extends DBIDriver. This object is used to create connections, using the function dbConnect, to one or more Oracle database engines.

Side Effects

The R client part of the database communication is initialized, but connecting to the database engine needs to be done through calls to dbConnect.

Oracle user authentication

In order to establish a connection to an Oracle server users need to provide a user name, a password, and possibly a connect identifier (for more information refer to chapter 8 (Configuring Naming Methods) of Oracle Database Net Services Administrator's Guide). This is the same as the part of the SQL*Plus connect string that follows the '@' sign.

Connections to an Oracle TimesTen IMDB instance are established using the OCI tnsnames or easy connect naming methods. For additional information on TimesTen connections for OCI see chapter 3 (TimesTen Support for Oracle Call Interface) of the Oracle TimesTen In-Memory C Developer's Guide.

Transactions

The current implementation directly supports transaction commits and rollbacks on a connection-wide basis through calls to dbCommit and dbRollback. Save points are not yet directly implemented, but you may be able to define them and rollback to them through calls to dynamic SQL with dbGetQuery.

Notice that Oracle (and ANSI/ISO compliant DBMS) transactions are implicitly started when data definition SQL statements are executed (create table, etc.), which helper functions like dbWriteTable may execute behind the scenes. You may want or need to commit or roll back your work before issuing any of these helper functions.

References

For Oracle Database documentation, see https://docs.oracle.com/en/database/.

Author(s)

David A. James and Denis Mukhin

See Also

On database managers:

dbDriver dbUnloadDriver dbListConnections

On connections:

dbConnect dbDisconnect dbSendQuery dbGetQuery dbGetException dbListResults

Convenience methods: dbListTables dbReadTable dbWriteTable dbExistsTable dbRemoveTable dbListFields

On transaction management:

dbCommit dbRollback

On queries and result objects:

fetch dbClearResult dbColumnInfo dbGetStatement dbHasCompleted dbGetRowsAffected dbGetRowCount

On meta-data:

show summary dbGetInfo

Examples

```
## Not run:
  library(ROracle)

  ## create a Oracle instance and create one connection.
  ora <- Oracle()         ## or dbDriver("Oracle")
  con <- dbConnect(ora, username = "scott", password = "tiger",
                   dbname = "inst1")

  ## if you are connecting to a local database
  con <- dbConnect(ora, username = "scott", password = "tiger")

  ## execute a statement and fetch its output in chunks of no more
  ## than 5000 rows at a time
  rs   <- dbSendQuery(con, "select * from emp where deptno = 10")
  while (!dbHasCompleted(rs)) {
    df <- fetch(rs, n = 5000)
    ## process df
  }
  dbClearResult(rs)       ## done with this query

  ## execute and fetch a statement with bind data
  df <- dbGetQuery(con, "select * from emp where deptno = :1",
                   data = data.frame(depno = 10))

  ## create a copy of emp table
  dbGetQuery(con, "create table foo as select * from emp")

  ## execute and bind an INSERT statement
  my.data = data.frame(empno = c(8001, 8002), ename = c('MUKHIN', 'ABOYOUN'))
  more.data = data.frame(empno = c(8003), ename = c('JAMES'))
  rs <- dbSendQuery(con, "insert into foo (empno, ename) values (:1, :2)",
```

```
                    data = my.data)

    ## execute with more data
    execute(rs, data = more.data)
    dbClearResult(rs)       ## done with this query

    ## ok, everything looks fine
    dbCommit(con)

    ## a concise description of the driver
    summary(ora)

    ## done with this connection
    dbDisconnect(con)

## End(Not run)
  ## Not run:
    library(ROracle)

    ## create an Oracle instance and create one connection to access data stored
    ## in object data types such as Collections, user defined types and varrays.
    ora <- Oracle(ora.attributes = TRUE, ora.objects = TRUE)

    con <- dbConnect(ora, username = "scott", password = "tiger",
                     dbname = "inst1")

    ## if you are connecting to a local database
    con <- dbConnect(ora, username = "scott", password = "tiger")

    ##
    ## object table with embedded object
    ##
    dbSendQuery(con,
                "CREATE OR REPLACE TYPE address AS OBJECT(\
                   no NUMBER,\
                   street VARCHAR(32)\
                )")

    dbSendQuery(con,
                "CREATE OR REPLACE TYPE employee AS OBJECT \
                 ( \
                   id                NUMBER, \
                   name              VARCHAR(16), \
                   birthday          DATE, \
                   resume            CLOB, \
                   addr              ADDRESS \
                 )")

    dbSendQuery(con, "CREATE TABLE emp_tab OF employee")

    # funtion to generate data
    myFun <- function(i = 1000, j = 2000, n = 1)
    {
```

```
  a <- do.call(paste0, replicate(i, sample(LETTERS, n, TRUE), FALSE))
  b <- do.call(paste0, replicate(j, sample(letters, n, TRUE), FALSE))
  paste0(a, " ", b)
}

# construct data frame to bind in insert statement into emp_tab table
df <- NULL
for (i in 101:200) {
  ID <- i + 10
  NAME <- paste("First", i, " Last", i+1, sep ="")
  z <- 946713600 + (i * 86400)
  BIRTHDAY <- as.POSIXct(z, origin = "1970-01-01")
  RESUME <- myFun(12, 2000, 1)
  attr(RESUME, "ora.type") <- "clob"

  ADDS <- list(data.frame(i+30, myFun(4, 10, 1), stringsAsFactors = FALSE))
  attr(ADDS, "ora.type") <- "ADDRESS"
  rowin <- data.frame(ID, NAME, BIRTHDAY, RESUME, stringsAsFactors = FALSE)
  rowin$ADDR <- ADDS
  attr(rowin, "ora.type") <- "EMPLOYEE"

  if (is.null(df))
    df <- rowin
  else
    df[nrow(df) + 1,] <- rowin
}

dbSendQuery(con,
            "insert into emp_tab(ID, NAME, BIRTHDAY, RESUME, ADDR) \
             values (:1, :2, :3, :4, :5)", df)


##
## table with id and embedded object
##
dbSendQuery(con, "CREATE OR REPLACE TYPE addss AS OBJECT(\
                      no NUMBER,\
                      street VARCHAR(32)\
                    )")

dbSendQuery(con, "CREATE OR REPLACE TYPE employee AS OBJECT\
                  (\
                  id                NUMBER,\
                  name              VARCHAR(16),\
                  birthday          DATE,\
                  resume            CLOB,\
                  addr              ADDSS\
                  )")

dbSendQuery(con, "create table emp_tab_b(id number, emp employee)")

## insert rows into table
dbSendQuery(con,
```

```
"insert into emp_tab_b values(1\
  employee(1, 'Sandy'\
         to_date('1972 08 23', 'YYYY MM DD')\
         'This is a comment'\
         ADDSS(500, 'Oracle pkwy'\
          ))")

dbSendQuery(con,
"insert into emp_tab_b values(2,\
  employee(2, 'Sandy 2',\
         to_date('1975 08 25', 'YYYY MM DD'),\
         'This is a comment2',\
         ADDSS(NULL, 'Oracle pkwy')\
          ))")

## funtion to generate data
myFun <- function(i = 1000, j = 2000, n = 1)
{
  a <- do.call(paste0, replicate(i, sample(LETTERS, n, TRUE), FALSE))
  b <- do.call(paste0, replicate(j, sample(letters, n, TRUE), FALSE))
  paste0(a, " ", b)
}

## construct data frame to bind in insert statement into emp_tab_b table
df <- NULL
for (i in 101:200) {
  ID <- i + 10
  NAME <- paste("First", i, " Last", i+1, sep ="")
  z <- 946713600 + (i * 86400)
  BIRTHDAY <- as.POSIXct(z, origin = "1970-01-01")
  RESUME <- myFun(12, 2000, 1)
  attr(RESUME, "ora.type") <- "clob"

  ADDS <- data.frame(i+30, myFun(4, 10, 1),stringsAsFactors = FALSE)
  rowin <- data.frame(ID, NAME, BIRTHDAY, RESUME, stringsAsFactors = FALSE)
  rowin$ADDR <- ADDS

  elem <- list(rowin)
  attr(elem, "ora.type") <- "EMPLOYEE"

  row <- data.frame(ID = i)
  row$EMP <- elem
  if (is.null(df))
    df <- row

  df[nrow(df) + 1,] <- row
}

dbSendQuery(con, "insert into emp_tab_b values(:1, :2)", df)

##
## Table with simple object type
##
```

```
dbSendQuery(con, "CREATE OR REPLACE TYPE a AS OBJECT (\
                    A1 NUMBER,\
                    A2 VARCHAR(20)\
                    )")

dbSendQuery(con, "create table obja (id number, a a)")

dbSendQuery(con, "insert into obja values(1, A(1, '111111'))")
## funtion to generate data
myFun <- function(i = 1000, j = 2000, n = 1)
{
  a <- do.call(paste0, replicate(i, sample(LETTERS, n, TRUE), FALSE))
  b <- do.call(paste0, replicate(j, sample(letters, n, TRUE), FALSE))
  paste0(a, " ", b)
}

## construct data frame to bind in insert statement into obja table
df <- NULL
for (i in 101:200) {
  A1 <- i + 10
  A2 <- paste("First", i, " Last", i+1, sep ="")
  rowin <- data.frame(A1, A2, stringsAsFactors = FALSE)

  elem <- list(rowin)
  attr(elem, "ora.type") <- "A"

  row <- data.frame(ID = i)
  row$EMP <- elem
  if (is.null(df))
    df <- row

  df[nrow(df) + 1,] <- row
}

dbSendQuery(con, "insert into obja values(:1, :2)", df)

##
## Table with a Varray type
##
dbSendQuery(con, "CREATE OR REPLACE TYPE varr is varray(10) of number")

dbSendQuery(con, "create table test(a number, b varr)")
dbSendQuery(con, "insert into TEST values(1, VARR(1, 2, 3, 4, 5, 6))")
dbSendQuery(con, "insert into TEST values(2, VARR(7, NULL, 9, 10, 11))")
dbSendQuery(con, "insert into TEST values(3, NULL))")
dbSendQuery(con, "insert into TEST values(4, VARR(12, NULL, 18, 19))")

## construct data frame to bind in insert statement into test table
df <- NULL
j <- 13
i <- 11
nrow <- 1
```

```
ncol <- 10
for (i in 101:200) {
  A <- i + 10
  VARR <- data.frame(matrix(rnorm(nrow*ncol),nrow, ncol))
  B <- list(VARR)
  rowin <- data.frame(A=A)
  rowin$B <- list(B)
  attr(rowin$B, "ora.type") <- "VARR"

  if (is.null(df))
    df <- rowin
  else
    df[nrow(df) + 1,] <- rowin
}

dbSendQuery(con, "insert into test(A, B) values (:1, :2)", df)

# check the data in table
dbGetQuery(con, "select * from test")


##
## Begin Table with a Vector type
##
dbSendQuery(con, "create table vectab (col_1 vector(*,*))")
dbSendQuery(con, "insert into vectab values('[1.1, 2.2, 3.3]')")
dbGetQuery(con, "select * from vectab")

# get vector data as populated by
res <- dbGetQuery(con, "select * from vectab")

# display column meta data
res <- dbSendQuery(con, 'SELECT * from vectab');
dbColumnInfo(res)

# fetch all data and display it
df1 <- dbGetQuery(con, 'SELECT * from vectab');
df1

# number of rows in data frame(result)
nrow(df1)
# number of columns in data frame(result)
ncol(df1)

# insert statement with bind variable
insBindStr <- "INSERT INTO vectab VALUES (:1)"

# re-insert data retrived earlier
dbSendQuery(con, insBindStr, df1) # insert all rows

# fetch all data and display it
res <- dbGetQuery(con, "select * from vectab")
res
```

```
# number of rows in data frame(result)
nrow(res)
# number of columns in data frame(result)
ncol(res)

# construct vector of list in number format
vecdf <- NULL
j <- 30
for (i in 1:5)
{
  id <- 100+i

  col_1 <- list(c(100+i+.1, 100+i+.2, 100+i+.3))
  row <- data.frame(id=id)
  row$col_1[[1]] <- col_1
  if (is.null(vecdf))
    vecdf <- row
  else
    vecdf[nrow(vecdf) + 1,] <- row

  str(vecdf)
}

# insert data in vecdf constructed above as a list of numbers
dbSendQuery(con, insBindStr, vecdf[2]) # insert all rows

# fetch all data and display it
res <- dbGetQuery(con, "select * from vectab")
res

# number of rows in data frame(result)
nrow(res)

mxl = attr(res, "ora.maxlength")
vecfmt = attr(res, "ora.format")

# construct vector of list using fixed format string format
vecdf <- NULL
j <- 30
for (i in 1:2)
{
  id <- 100+i

  if (i == 2)
    col_1 <- list(NULL)
  else if (i == 3)
    col_1 <- list('')
  else
    col_1 <- list('[9.4, 9.6,9.7]')

  attr(col_1, "ora.type") <- "vector"
  attr(col_1, "ora.maxlength") <- mxl
```

```
  attr(col_1, "ora.format") <- vecfmt

  row <- data.frame(id=id)
  row$col_1[[1]] <- col_1
  if (is.null(vecdf))
    vecdf <- row
  else
    vecdf[nrow(vecdf) + 1,] <- row

  str(vecdf)
}

# insert data in vecdf constructed above as a list with fixed string
dbSendQuery(con, insBindStr, vecdf[2]) # insert all rows
dbSendQuery(con, "commit")

# fetch all data and display it
res <- dbGetQuery(con, "select * from vectab")
res

# number of rows in data frame(result)
nrow(res)

# construct vector of list in integer format
vecdf <- NULL
j <- 30
for (i in 1:5)
{
  id <- 100+i

  col_1 <- list(c(as.integer(900+i), as.integer(900+i), as.integer(900+i)))
  row <- data.frame(id=id)
  row$col_1[[1]] <- col_1
  if (is.null(vecdf))
    vecdf <- row
  else
    vecdf[nrow(vecdf) + 1,] <- row

  str(vecdf)
}

# insert data in vecdf constructed above as a list integers
dbSendQuery(con, insBindStr, vecdf[2]) # insert all rows
dbSendQuery(con, "commit")

# fetch all data and display it
res <- dbGetQuery(con, "select * from vectab")
res

# number of rows in data frame(result)
nrow(res)

# construct vector of list using variable string format
```

```
vecdf <- NULL
j <- 30
for (i in 1:5)
{
  id <- 1000+i

  str <- '['
  for (k in 1:7)
  {
    str <- paste(str, as.character(k*3+.7*i))
    str <- paste(str, ',');
  }
  str <- paste(str, as.character(k*3+.7*i))
  str <- paste(str, ']')
  row <- data.frame(id=id)
  row$col_1[[1]] <- str
  if (is.null(vecdf))
    vecdf <- row
  else
    vecdf[nrow(vecdf) + 1,] <- row

  str(vecdf)
}

# insert data in vecdf constructed above as a list of variable strings
dbSendQuery(con, insBindStr, vecdf[2]) # insert all rows
dbSendQuery(con, "commit")

# fetch all data and display it
res <- dbGetQuery(con, "select * from vectab")
res

# number of rows in data frame(result)
nrow(res)

res <- dbGetQuery(con, "drop table vectab")

##
## Create a table with a sparse vector column
##
dbSendQuery(con, "CREATE TABLE sparse_vectab \
               (\
                 id NUMBER, \
                 c1 VECTOR(*, *, SPARSE)\
               )")

dbSendQuery(con, "INSERT INTO sparse_vectab VALUES \
               (101, '[10, [2, 3, 4], [10, 20, 30]]')")

library(Matrix)

# get vector data as populated by
res <- dbGetQuery(con, "select * from sparse_vectab", sparse = TRUE)
```

```
# display column meta data
res <- dbSendQuery(con, 'SELECT * from sparse_vectab');
dbColumnInfo(res)

# fetch all rows and display the sparse vector column
df1 <- dbGetQuery(con, 'SELECT * from sparse_vectab', sparse = TRUE);
df1$C1

# number of rows in data frame(result)
nrow(df1)
# number of columns in data frame(result)
ncol(df1)

# insert statement with bind variable
insBindStr <- "INSERT INTO sparse_vectab VALUES (:1, :2)"

# re-insert data retrived earlier
dbSendQuery(con, insBindStr, df1) # insert all rows

# Fetch all rows and display the sparse vector column
res <- dbGetQuery(con, "select * from sparse_vectab", sparse = TRUE)
res$C1

# retrieve and display all data as list of numeric
res <- dbGetQuery(con, "select * from sparse_vectab")
res$C1

# number of rows in data frame(result)
nrow(res)

# construct R dsparseVector objects for inserting
v1 <- new("dsparseVector",
      i = c(1L, 2L, 3L), #integer indices
      x = c(1.12, 2.23, 3.12),
      length = 320L)
v2 <- new("dsparseVector",
      i = c(1L, 2L, 3L, 4L), #integer indices
      x = c(1.12, 2.23, 3.12, 4.23),
      length = 1000L)
vecdf <- data.frame(id = c(9, 10))
vecdf$c1 <- list(v1, v2)
attr(vecdf$c1, "ora.type") <- "vector"
attr(vecdf$c1, "ora.format") <- "float32"

# insert data in vecdf constructed above as a list of dsparseVector
dbSendQuery(con, insBindStr, vecdf) # insert all rows
dbSendQuery(con, "commit")

# fetch all data and display it
res <- dbGetQuery(con, "select * from sparse_vectab", sparse = TRUE)
res$C1
```

```
# number of rows in data frame(result)
nrow(res)

res <- dbGetQuery(con, "drop table sparse_vectab")
##
## End of Table with a Vector type
##

## execute a statement and fetch its output in chunks of no more
## than 5000 rows at a time
rs   <- dbSendQuery(con, "select * from emp where deptno = 10")
while (!dbHasCompleted(rs)) {
  df <- fetch(rs, n = 5000)
  ## process df
}
dbClearResult(rs)       ## done with this query

## execute and fetch a statement with bind data
df <- dbGetQuery(con, "select * from emp where deptno = :1",
                 data = data.frame(depno = 10))

## create a copy of emp table
dbGetQuery(con, "create table foo as select * from emp")

## execute and bind an INSERT statement
my.data = data.frame(empno = c(8001, 8002), ename = c('MUKHIN', 'ABOYOUN'))
more.data = data.frame(empno = c(8003), ename = c('JAMES'))
rs <- dbSendQuery(con, "insert into foo (empno, ename) values (:1, :2)",
                  data = my.data)

## execute with more data
execute(rs, data = more.data)
dbClearResult(rs)       ## done with this query

## ok, everything looks fine
dbCommit(con)

## a concise description of the driver
summary(ora)

## done with this connection
dbDisconnect(con)

## End(Not run)
```

`OraConnection-class` *Class OraConnection*

Description

An Oracle connection class implementing the R database interface (DBI) API.

Generators

The method dbConnect is the main generator.

Extends

Class "DBIConnection", directly. Class "DBIObject", by class "DBIConnection", distance 2.

Methods

dbDisconnect signature(conn = "OraConnection"): ...

dbSendQuery signature(conn = "OraConnection", statement = "character", prefetch = FALSE, bulk_read = 1000L, bulk_write = 1000L): ...

dbGetQuery signature(conn = "OraConnection", statement = "character", prefetch = FALSE, bulk_read = 1000L, bulk_write = 1000L): ...

dbGetException signature(conn = "OraConnection"): ...

dbListResults signature(conn = "OraConnection"): ...

dbListTables signature(conn = "OraConnection"): ...

dbReadTable signature(conn = "OraConnection", name = "character"): ...

dbWriteTable signature(conn = "OraConnection", name = "character", value = "data.frame"): ...

dbExistsTable signature(conn = "OraConnection", name = "character"): ...

dbRemoveTable signature(conn = "OraConnection", name = "character"): ...

dbListFields signature(conn = "OraConnection", name = "character"): ...

dbCommit signature(conn = "OraConnection"): ...

dbRollback signature(conn = "OraConnection"): ...

dbGetInfo signature(dbObj = "OraConnection"): ...

summary signature(object = "OraConnection"): ...

show signature(object = "OraConnection")

See Also

DBI classes: OraDriver-class OraConnection-class OraResult-class

Examples

```
## Not run:
ora <- dbDriver("Oracle")
## connecting without a connect string
con <- dbConnect(ora, "scott", "tiger")

## connecting with a connection string with SID
host <- "myhost"
port <- 1521
sid <- "mysid"
connect.string <- paste(
```

```
  "(DESCRIPTION=",
  "(ADDRESS=(PROTOCOL=tcp)(HOST=", host, ")(PORT=", port, "))",
  "(CONNECT_DATA=(SID=", sid, ")))", sep = "")

## use username/password authentication
con <- dbConnect(drv, username = "scott", password = "tiger",
                 dbname = connect.string)

## connecting with a connection string with service name
host <- "myhost"
port <- 1521
svc <- "mydb.example.com"
connect.string <- paste(
  "(DESCRIPTION=",
  "(ADDRESS=(PROTOCOL=tcp)(HOST=", host, ")(PORT=", port, "))",
  "(CONNECT_DATA=(SERVICE_NAME=", svc, ")))", sep = "")
## use username/password authentication
con <- dbConnect(drv, username = "scott", password = "tiger",
                 dbname = connect.string)

## Please refer to "Oracle Database Net Services Administator's Guide", which
## has the topic "Connect Identifier and Connect Descriptor Syntax
## Characteristics"

dbListTables(con)

## End(Not run)
```

`OraDriver-class` *Class OraDriver*

Description

An Oracle driver class implementing the R database interface (DBI) API.

Generators

The main generators are `dbDriver` and `Oracle`.

Extends

Class `"DBIDriver"`, directly. Class `"DBIObject"`, by class `"DBIDriver"`, distance 2.

Methods

dbConnect `signature(drv = "OraDriver")`: ...

dbGetInfo `signature(dbObj = "OraDriver")`: ...

dbListConnections `signature(drv = "OraDriver")`: ...

dbUnloadDriver `signature(drv = "OraDriver")`: ...

summary `signature(object = "OraDriver")`: ...

show `signature(object = "OraDriver")`

See Also

DBI classes: `OraConnection-class` `OraResult-class`

Examples

```
## Not run:
# first load the library
library("ROracle")
ora <- dbDriver("Oracle")
con <- dbConnect(ora, "scott", "tiger")

## End(Not run)
```

`OraResult-class` *Class OraResult*

Description

An Oracle query results class. This class encapsulates the result of a SQL statement.

Generators

The main generator is `dbSendQuery`.

Extends

Class `"DBIResult"`, directly. Class `"DBIObject"`, by class `"DBIResult"`, distance 2.

Methods

dbClearResult `signature(res = "OraResult")`: ...

dbColumnInfo `signature(res = "OraResult")`: ...

dbGetInfo `signature(dbObj = "OraResult")`: ...

dbGetStatement `signature(res = "OraResult")`: ...

dbGetRowCount `signature(res = "OraResult")`: ...

dbGetRowsAffected `signature(res = "OraResult")`: ...

dbHasCompleted `signature(res = "OraResult")`: ...

fetch `signature(res = "OraResult", n = "numeric")`: ...

fetch `signature(res = "OraResult", n = "missing")`: ...

execute `signature(res = "OraResult")`: ...

summary `signature(object = "OraResult")`: ...

show `signature(object = "OraResult")`

See Also

DBI classes: OraDriver-class OraConnection-class OraResult-class

Examples

```
## Not run:
ora <- dbDriver("Oracle")
con <- dbConnect(ora, "scott", "tiger")
res <- dbSendQuery(con, "select * from emp")
fetch(res, n = 2)
fetch(res)
dbColumnInfo(res)
dbClearResult(res)

## End(Not run)
```

---

summary-methods *Summarize an Oracle object*

Description

These methods are straight-forward implementations of the corresponding generic functions.

Usage

```
## S4 method for signature 'OraDriver'
summary(object, ...)
## S4 method for signature 'ExtDriver'
summary(object, ...)
## S4 method for signature 'OraConnection'
summary(object, ...)
## S4 method for signature 'OraResult'
summary(object, ...)
```

Arguments

object	a driver, connection or result set object.
...	currently unused.

Value

description of object.

References

For the Oracle Database documentaion see https://docs.oracle.com/en/.

See Also

Oracle, dbConnect, dbSendQuery, dbGetQuery, dbClearResult, dbCommit, dbGetInfo, dbGetInfo.

Examples

```
  ## Not run:
    drv <- dbDriver("Oracle")
    con <- dbConnect(drv, "scott", "tiger")
    res <- dbSendQuery(con, "select * from emp")

    summary(drv)
    summary(con)
    summary(res)
    show(drv)
    show(con)
    show(res)

## End(Not run)
```

Index