The ROracle Package

November 12, 2003

Version 0.5-3

Date 2003-11-05

Title Oracle database interface for R

Author David A. James <dj@research.bell-labs.com> Jake Luciani <jake@agere.com>

Maintainer David A. James <dj@research.bell-labs.com>

Description Oracle database interface (DBI) driver for R. This is a DBI-compliant Oracle driver based on the ProC/C++ embedded SQL. It implements the DBI version 0.1-4 plus one extension.

Depends R (>= 1.6.0), methods, DBI (>= 0.1-4)

License LGPL version 2 or newer

URL stat.bell-labs.com/RS-DBI, www.omegahat.org

R topics documented:

`DBIPreparedStatement-class` *Class DBIPreparedStatement*

Description

Base class for all DBMS-specific prepared statement objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class `"DBIObject"`, directly. Class `"DBIResult"`, directly.

Generator

The main generator is `dbPrepareStatement`.

Methods

[**ROracle** dbExecStatement] `signature(ps = "DBIPreparedStatement", data = "data.frame")`:
...

Author(s)

R-SIG-DB

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

DBI classes: `DBIObject-class` `DBIDriver-class` `DBIConnection-class` `DBIResult-class`

Examples

```
## Don't run:
 drv <- dbDriver("Oracle")
 con <- dbConnect(drv, "user/password@dbname")
 ## to do...
## End Don't run
```

`OraConnection-class` *Class OraConnection*

Description

Oracle connection class.

Generators

Extends

Class `"DBIConnection"`, directly. Class `"OraObject"`, directly. Class `"DBIObject"`, by class "DBIConnection". Class `"dbObjectId"`, by class "OraObject".

Methods

coerce `signature(from = "OraConnection", to = "OraResult")`: ...

dbCallProc `signature(conn = "OraConnection")`: ...

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

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

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

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

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

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

dbGetQuery `signature(conn = "OraConnection", statement = "character")`: ...

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

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

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

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

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

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

dbSendQuery `signature(conn = "OraConnection", statement = "character")`: ...

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

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

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
ora <- dbDriver("Oracle")
con <- dbConnect(ora, "user/password@dbname")
## End Don't run
```

`OraDriver-class` *Class OraDriver*

Description

An Oracle driver implementing the R/S-Plus database (DBI) API.

Generators

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

Extends

Class `"DBIDriver"`, directly. Class `"OraObject"`, directly. Class `"DBIObject"`, by class "DBIDriver". Class `"dbObjectId"`, by class "OraObject".

Methods

coerce `signature(from = "OraObject", to = "OraDriver")`: ...

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

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

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

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

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

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
ora <- dbDriver("Oracle")
con <- dbConnect(ora, "user/password@dbname")
## End Don't run
```

`OraObject-class` *Class OraObject*

Description

Base class for all Oracle-specific DBI classes

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class `"DBIObject"`, directly. Class `"dbObjectId"`, directly.

Methods

coerce `signature(from = "OraObject", to = "OraDriver")`: ...

dbDataType `signature(dbObj = "OraObject")`: ...

isSQLKeyword `signature(dbObj = "OraObject", name = "character")`: ...

make.db.names `signature(dbObj = "OraObject", snames = "character")`: ...

SQLKeywords `signature(dbObj = "OraObject")`: ...

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
ora <- dbDriver("Oracle")
con <- dbConnect(ora, "user/password@dbname")
## End Don't run
```

OraPreparedStatement-class

Oracle Prepared Statement

Description

A class that encapsulates the information on an Oracle prepared statement

Objects from the Class

Use the method `dbPrepareStatement` to create an Oracle prepared statement and `dbExecStatement` to re-bind new data and execute the cached statement.

Slots

`Id`: an opaque reference into the prepared statement.

Extends

Class `"DBIPreparedStatement"`, directly. Class `"OraResult"`, directly. Class `"DBIObject"`, by class "DBIPreparedStatement". Class `"DBIResult"`, by class "OraResult". Class `"OraObject"`, by class "OraResult". Class `"dbObjectId"`, by class "OraResult".

Methods

dbExecStatement `signature(ps = "OraPreparedStatement", data = "data.frame")`: executes a prepared statement re-binding new data to it.

dbGetInfo `signature(dbObj = "OraPreparedStatement")`: returns a list of metadata associated with the prepared statement.

summary `signature(object = "OraPreparedStatement")`: writes a brief summary of the status of the prepared statement.

Background

Oracle's prepared statements (like other RDBMS') are SQL statements that are parsed and cached to increase performance when the SQL code is to be executed repeatedly but with different data; for instance when inserting the rows of a `data.frame` into a table the SQL for each row is exactly the same, only the row data changes.

The function `dbPrepareStatement` creates objects that extend the base class `DBIPreparedStatement`. These objects are simple references into C structures that store the various aspects (the text of the SQL statement, sets of buffers for transferring data back and forth, etc).

The function `dbExecStatement` takes a prepared statement object and a `data.frame` and binds one or more of its columns to the RDBMS table or object according to the specification in the prepared statement.

Note

As of the DBI version 0.1-5 prepared statements are not part of the R/S Database Interface definition (DBI).

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

DBI classes: `OraObject-class` `OraDriver-class` `OraConnection-class` `OraResult-class` `OraPreparedStatement-class`

Examples

```
## Don't run:
  ora <- dbDriver("Oracle")
  con <- dbConnection(ora, "user/password")
  ps <- dbPrepareStatement(con,
           "INSERT into QUAKES (lat, long_1) VALUES (:1, :2)",
           bind = c("numeric", "numeric"))
  dbExecStatement(ps, quakes)
  dbCommit(con)
## End Don't run
```

`OraResult-class` *Class OraResult*

Description

Oracle's query results class. This classes encapsulates the result of an SQL statement (either `select` or not).

Generators

The main generator is `dbSendQuery`.

Extends

Class `"DBIResult"`, directly. Class `"OraObject"`, directly. Class `"DBIObject"`, by class "DBIResult". Class `"dbObjectId"`, by class "OraObject".

Methods

coerce `signature(from = "OraConnection", to = "OraResult")`: ...

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

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

dbGetException `signature(conn = "OraResult")`: ...

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

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

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

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

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

dbListFields `signature(conn = "OraResult", name = "missing")`: ...

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

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

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

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
ora <- dbDriver("Oracle")
con <- dbConnect(ora, "user/password@dbname")
## End Don't run
```

`Oracle` *Instantiate an Oracle client from the current R/S-Plus session*

Description

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

Usage

```
Oracle(max.con = 10, fetch.default.rec = 500, force.reload = F)
```

Arguments

`max.con`	maximum number of connections that we intend to have open. This can be up to 10, a limit hard-coded in the current implementation.
`fetch.default.rec`	number of records to fetch at one time from the database. (The `fetch` method uses this number as a default.)
`force.reload`	should we reload (reinitialize) the client code? Setting this to `TRUE` allows you to change default settings. Notice that all connections should be closed before re-loading.

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.

Value

An object `OraDriver` whose class extends `DBIDriver` and the mixin (helper) class `dbObjectId`. This object is used to create connections, using the function `dbConnect`, to one or several Oracle database engines.

Side Effects

The R/S-Plus client part of the database communication is initialized, but note that 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 an "Oracle SID" (i.e., a database name); by default the Oracle SID is taken from the environment variable `$ORACLE_SID`. The function `dbConnect` allows authentication strings similar to the Oracle monitor `SQL*Plus`, namely, a string of any of the following forms:

1. `"user/passsword"`
2. `"user/password@dbname"`
3. `"/"` (provided the Oracle server is set up to use the underlying operating system users and passwords);

Prepared statements and data.frame bindings

As of version 0.5-0, `ROracle` implements R/S-Plus data binding to prepared SQL statements. This is done in two stages with the functions `dbPrepareStatement` and `dbExecStatement`.

In the first stage of preparing a statement column numbers are embedded inside the SQL statement, e.g., `"insert into my_table (id, name, val) VALUES (:1, :3, :2)"` and the S class of those columns are specified in the `bind=` argument to `dbPrepareStatement`

In the second stage `dbExecStatement` binds the pre-specified columns from a supplied `data=` data frame to the SQL statement and the SQL statement is executed once for each row in the input data frame. This step can be repeated with new data as many times as needed.

It is very important to note that typically a prepared statement implicitly will define a new transaction which needs to be explicitly committed with `dbCommit` or rolled back with `dbRollback`.

The current implementation allows only primitive types `c("numeric", "integer", "logical", "character")` for binding.

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 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 more details on the R/S-Plus database interface see the PDF file `DBI.pdf` under the `doc` directory of this package, http://stat.bell-labs/RS-DBI, and the Omega Project for Statistical Computing at http://www.omegahat.org.

See the documentation at the Oracle Web site http://www.oracle.com for details.

Author(s)

David A. James

See Also

On database managers:

`dbDriver` `Oracle` `dbUnloadDriver`

On connections:

`dbConnect` `dbDisconnect`

On queries, prepared statements, and result objects:

`dbSendQuery` `fetch` `dbGetQuery` `dbClearResult` `dbPrepareStatement` `dbExecStatement`

On transaction management:

`dbCommit` `dbRollback`

On meta-data:

`dbGetInfo` `summary` `dbListTables` `dbListFields` `dbListConnections` `dbListResults` `dbGetException` `dbGetStatement` `dbHasCompleted` `dbGetRowCount` `dbGetAffectedRows`

Examples

```
## Don't run:
## create a Oracle instance and create one connection.
ora <- Oracle()     ## or dbDriver("Oracle")
con <- dbConnect(ora, user = "opto", password="pure-light", db="oras")

## you can also use Oracle's user/password@dbname convention
con2 <- dbConnect(ora, user = "opto/pure-light@oras")

## or if you have defined the ORACLE_SID shell variable
con3 <- dbConnect(ora, user = "opto", password = "pure-light")

## clone an existing connection
w <- dbConnect(con)

## execute a statement and fetch its output in chunks of no more
## than 5000 rows at a time

rs <- dbSendQuery(con, "select * from HTTP_ACCESS where IP_ADDRESS='127.0.0.1'")

while(!dbHasCompleted(rs)){
   df <- fetch(rs, n = 5000)
   process(df)
}
dbHasCompleted(rs)
[1] TRUE
dbClearResult(rs)      ## done with this query
```

```
[1] TRUE

## prepare and bind columns 2, 3, and 7 to the Oracle table
## fields "cell", "erlangs", "blocking"
ps <- dbPrepareStatement(con,
         "INSERT into my_table (cell, erlangs, blocking) VALUE (:2,:3,:7)",
         bind = my.data.frame)

## execute one sql INSERT per row using columns 2, 3 and 7
ps <- dbExecStatement(ps, my.data.frame)
ps <- dbExecStatement(ps, more.data)

dbCommit(con)  ## ok, everything looks fine

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

<OraDriver:(24694)>
  Driver name:  Oracle (ProC/C++)
  Max  connections: 10
  Conn. processed: 9
  Default records per fetch: 500
  Open connections: 2

## a full description of the ora connection
summary(con, verbose = T)

<OraConnection:(25272,0)>
  User: opto
  Dbname: oras
  Oracle Server version:
    Oracle8 Enterprise Edition Release 8.0.4.0.0 - Production
    PL/SQL Release 8.0.4.0.0 - Production
    CORE Version 4.0.4.0.0 - Production
    TNS for Solaris: Version 8.0.4.0.0 - Production
    NLSRTL Version 3.3.1.0.0 - Production

dbDisconnect(con)     ## done with this connection
[1] TRUE
## End Don't run
```

`S4R` *R compatibility with S version 4/S-Plus 5+ support functions*

Description

These objects ease the task of porting functions into R from S Version 4 and S-Plus 5.0 and later. See the documentation of the lower-case functions there. May be obsolete in the future.

Usage

```
usingR(major, minor)
```

`dbCallProc-methods` *Call an SQL stored procedure*

Description

Not yet implemented.

Methods

conn a `OraConnection` object.

`...` additional arguments are passed to the implementing method.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

`dbCommit-methods` *DBMS Transaction Management*

Description

Commits or roll backs the current transaction in an Oracle connection

Methods

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

`...` currently unused.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
drv <- dbDriver("Oracle")
con <- dbConnect(drv, "user/password@SID")
rs <- dbSendQuery(con,
      "delete * from PURGE as p where p.wavelength<0.03")
if(dbGetInfo(rs, what = "rowsAffected") > 250){
  warning("dubious deletion -- rolling back transaction")
  dbRollback(con)
}
## End Don't run
```

`dbConnect-methods` *Create a connection object to an Oracle DBMS*

Description

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

Methods

drv an object of class `OraDriver`, or the character string ”Oracle” or an `OraConnection`.

conn an `OraConnection` object as produced by `dbConnect`.

username string of the Oracle login name.

password string with the Oracle password.

dbname string with the Oracle SID, System Identification (database name). The default takes this fromt the environment variable `ORACLE_SID`.

... Must specify user, password and optionally dbname. Also you may specify an Oracle connection string, e.g., ”user/password@SID”.

Side Effects

A connection between R/S-Plus and an Oracle server is established. The current implementation supports up to 10 simultaneous connections.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
# create an Oracle instance and create one connection.
drv <- dbDriver("Oracle")

# open the connection using user, passsword, etc., as
con <- dbConnect(drv, "user/password@dbname")

# Run an SQL statement by creating first a resultSet object
rs <- dbSendQuery(con, statement = paste(
                      "SELECT w.laser_id, w.wavelength, p.cut_off",
                      "FROM WL w, PURGE P",
                      "WHERE w.laser_id = p.laser_id",
                      "SORT BY w.laser_id")
# we now fetch records from the resultSet into a data.frame
data <- fetch(rs, n = -1)   # extract all rows
dim(data)
## End Don't run
```

`dbDataType-methods` *Determine the SQL Data Type of an S object*

Description

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

Methods

dbObj a `OraDriver` object, e.g., `ODBCDriver`, `OracleDriver`.

obj R/S-Plus object whose SQL type we want to determine.

`...` any other parameters that individual methods may need.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

`isSQLKeyword` `make.db.names`

Examples

```
## Don't run:
data(quakes)
drv <- dbDriver("Oracle")
sql.type <- dbDataType(drv, quakes)
## End Don't run
```

dbDriver-methods — *Oracle implementation of the Database Interface (DBI) classes and drivers*

Description

Oracle driver initialization and closing

Methods

drvName character name of the driver to instantiate.

drv an object that inherits from `OraDriver` as created by `dbDriver`.

... any other arguments are passed to the driver `drvName`.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
# create an Oracle instance and set 10000 of rows per fetch.
m <- dbDriver("Oracle", fetch.default.records=10000)

con <- dbConnect(m, username="usr", password = "pwd",
           dbname = "iptraffic")
rs <- dbSubmitQuery(con,
         "select * from HTTP_ACCESS where IP_ADDRESS = '127.0.0.1'")
df <- fetch(rs, n = 50)
df2 <- fetch(rs, n = -1)
dbClearResult(rs)

pcon <- dbConnect(p, "user", "password", "dbname")
dbListTables(pcon)
## End Don't run
```

dbGetInfo-methods — *Database interface meta-data*

Description

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

Methods

dbObj any object that implements some functionality in the R/S-Plus interface to databases (a driver, a connection or a result set).

res an `OraResult`.

... currently not being used.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
drv <- dbDriver("Oracle")
con <- dbConnect(drv, "user/passwd@dbname")

dbListTables(con)

rs <- dbSendQuery(con, query.sql)
dbGetStatement(rs)
dbHasCompleted(rs)

info <- dbGetInfo(rs)
names(dbGetInfo(drv))

# DBIConnection info
names(dbGetInfo(con))

# DBIResult info
names(dbGetInfo(rs))
## End Don't run
```

`dbListTables-methods` *List items from an Oracle DBMS and from objects*

Description

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

Methods

drv an `OraDriver`.

conn an `OraConnection`.

name a character string with the table name.

... currently not used.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

`Oracle`, `dbGetInfo`, `dbColumnInfo`, `dbDriver`, `dbConnect`, `dbSendQuery`

Examples

```
## Don't run:
drv <- dbDriver("Oracle")
# after working awhile...
for(con in dbListConnections(drv)){
   dbGetStatement(dbListResults(con))
}
## End Don't run
```

`dbObjectId-class` *Class dbObjectId*

Description

A helper (mixin) class to provide external references in an R/S-Plus portable way.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`Id`: Object of class `"integer"` this is an integer vector holding an opaque reference into a C struct (may or may not be a C pointer, may or may not have length one).

Methods

coerce `signature(from = "dbObjectId", to = "integer")`: ...

coerce `signature(from = "dbObjectId", to = "numeric")`: ...

coerce `signature(from = "dbObjectId", to = "character")`: ...

format `signature(x = "dbObjectId")`: ...

print `signature(x = "dbObjectId")`: ...

show `signature(object = "dbObjectId")`: ...

Note

A cleaner mechanism would use external references, but historically this class has existed mainly for R/S-Plus portability.

Examples

```
## Don't run:
  pg <- dbDriver("PostgreSQL")
  con <- dbConnect(pg, "user", "password")
  is(pg, "dbObjectId")   ## True
  is(con, "dbObjectId")  ## True
  isIdCurrent(con)       ## True
  q("yes")
  \$ R
  isIdCurrent(con)       ## False
## End Don't run
```

---

dbPrepareStatement-methods

Create a prepared SQL statement for repeated execution

Description

These methods parse and cache SQL statements and binds R data for repeated execution.

Details

Prepared statements are SQL statements that are parsed and cached to increase performance when the SQL code is to be executed repeatedly but with different data.

There are three distinct operations involved with prepared statements: parsing and caching the SQL statement, binding `data.frame` columns to the SQL, and executing the code (possibly repeatedly).

The function `dbPrepareStatement` takes a connection where to parse and cache the SQL code. Part of this operation is to embed references to `data.frame` column numbers in the SQL code and to specify their classes through the `bind=` argument. The `ROracle` package uses `:n` inside the SQL statement to bind the $n'th$ column, but other RDBMSs use the question mark to signal a place holder, e.g., `?n`.

The object that `dbPrepareStatement` produces is then used together with a `data.frame` (which should agree with the bound specification) in calls to `dbExecStatement` to be executed for each row of the `data.frame`. This can be repeated with new data.

Embedding column names, instead of column numbers, is not supported, since some valid S names are not legal SQL names (e.g., S names with dots `"."` in them).

Value

An object whose class extends `DBIPreparedStatement`.

In the current `ROracle` implementation the `OraPreparedStatement` class specializes (extends) `OraResultSet`, thus prepared statment objects inherit all result set methods, e.g., `fetch`, `dbClearResult`, `dbGetStatement`, `dbGetRowsAffected`.

Methods

conn a database connection

statement a string with an SQL statement, possibly with embedded column number specifications of the form `:columnNum` (e.g., `:1,:2,:6`) for binding those columns in the `data` argument to `dbExecStatement`.

bind a character vector parallel to the column specifications describing their S classes (e.g., `"character"`, `"numeric"`). You may supply a data.frame, in which case `bind=` is set to `sapply(data, class)`.

ps a prepared statement object as produced by `dbPrepareStatement`.

data a `data.frame` whose columns are to be bound to the SQL statement.

... other arguments are passed to the driver implementation. For instance, the argument `ora.buf.size` is used to specify the size of Oracle's internal binding buffers (ROracle sets these to 500 elements by default).

Note

These functions are `ROracle` extensions to the DBI as of version 0.1-7.

See Also

`DBIPreparedStatement-class` `OraPreparedStatement-class` `OraResult-class` `dbSendQuery` `dbGetQuery` `dbGetInfo` `summary`

Examples

```
## Don't run:
  con <- dbConnection("Oracle", "user/password")

  ps <- dbPrepareStatement(con,
           "INSERT into QUAKES (lat, long1, mag) VALUES (:1, :2, :4)",
           bind = c("numeric", "numeric", "numeric"))

  dbExecStatement(ps, data = quakes)
  dbExecStatement(ps, data = more.quakes)
  ...
  dbExecStatement(ps, data = yet.more.quakes)

  ## how many rows have we (tentatively) inserted?
  summary(ps)

  ## everything looks fine, so let's commit and wrap up
  dbCommit(con)
  dbClearResult(ps)
## End Don't run
```

`dbPrepareStatement` *Create a prepared SQL statement for repeated execution*

Description

These functions parse and cache SQL statements and binds S data for repeated execution.

Usage

```
dbPrepareStatement(conn, statement, bind, ...)
dbExecStatement(ps, data, ...)
```

Arguments

`conn`	a database connection
`statement`	a string with an SQL statement, possibly with embedded column number specifications of the form `:columnNum` (e.g., `:1,:2,:6`) for binding those columns in the `data` argument to `dbExecStatement`.
`bind`	a character vector parallel to the column specifications describing their S classes (e.g., `"character"`, `"numeric"`). You may supply a data.frame, in which case `bind=` is set to `sapply(data, class)`.
`ps`	a prepared statement object as produced by `dbPrepareStatement`.
`data`	a `data.frame` whose columns are to be bound to the SQL statement.
`...`	other arguments are passed to the driver implementation. For instance, the argument `ora.buf.size` is used to specify the size of Oracle's internal binding buffers (ROracle sets these to 500 elements by default).

Details

Prepared statements are SQL statements that are parsed and cached to increase performance when the SQL code is to be executed repeatedly but with different data.

There are three distinct operations involved with prepared statements: parsing and caching the SQL statement, binding `data.frame` columns to the SQL, and executing the code (possibly repeatedly).

The function `dbPrepareStatement` takes a connection where to parse and cache the SQL code. Part of this operation is to embed references to `data.frame` column numbers in the SQL code and to specify their classes through the `bind=` argument. The `ROracle` package uses `:n` inside the SQL statement to bind the *n'th* column, but other RDBMSs use the question mark to signal a place holder, e.g., `?n`.

The object that `dbPrepareStatement` produces is then used together with a `data.frame` (which should agree with the bound specification) in calls to `dbExecStatement` to be executed for each row of the `data.frame`. This can be repeated with new data.

Embedding column names, instead of column numbers, is not supported, since some valid S names are not legal SQL names (e.g., S names with dots `"."` in them).

Value

An object whose class extends `DBIPreparedStatement`.

In the current `ROracle` implementation the `OraPreparedStatement` class specializes (extends) `OraResultSet`, thus prepared statment objects inherit all result set methods, e.g., `fetch`, `dbClearResult`, `dbGetStatement`, `dbGetRowsAffected`.

Note

These functions are `ROracle` extensions to the DBI as of version 0.1-7.

See Also

`OraPreparedStatement-class` `OraResult-class` `dbSendQuery` `dbGetQuery` `dbGetInfo` `summary`

Examples

```
## Don't run:
  con <- dbConnection("Oracle", "user/password")

  ps <- dbPrepareStatement(con,
           "INSERT into QUAKES (lat, long1, mag) VALUES (:1, :2, :4)",
           bind = c("numeric", "numeric", "numeric"))

  dbExecStatement(ps, data = quakes)
  dbExecStatement(ps, data = more.quakes)
  ...
  dbExecStatement(ps, data = yet.more.quakes)

  ## how many rows have we (tentatively) inserted?
  summary(ps)

  ## everything looks fine, so let's commit and wrap up
  dbCommit(con)
  dbClearResult(ps)
## End Don't run
```

`dbReadTable-methods` *Convenience functions for Importing/Exporting DBMS tables*

Description

These functions mimic their R/S-Plus counterpart `get`, `assign`, `exists`, `remove`, and `objects`, except that they generate code that gets remotely executed in a database engine.

Value

A data.frame in the case of `dbReadTable`; otherwise a logical indicating whether the operation was successful.

Methods

conn an `OraConnection` database connection object.

name a character string specifying a table name.

value a data.frame (or coercible to data.frame).

row.names in the case of `dbReadTable`, this argument can be a string or an index specifying the column in the DBMS table to be used as `row.names` in the output data.frame (a `NULL`, `""`, or 0 specifies that no column should be used as `row.names` in the output). In the case of `dbWriteTable`, this argument should be a logical specifying whether the `row.names` should be output to the output DBMS table; if `TRUE`, an extra field whose name will be whatever the R/S-Plus identifier `"row.names"` maps to the DBMS (see `make.db.names`).

overwrite a logical specifying whether to overwrite an existing table or not. Its default is `FALSE`.

append a logical specifying whether to append to an existing table in the DBMS. Its default is `FALSE`.

... any optional arguments.

Note

Note that the data.frame returned by `dbReadTable` only has primitive data, e.g., it does not coerce character data to factors.

Oracle table names are *not* case sensitive, e.g., table names `ABC` and `abc` are considered equal.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
conn <- dbConnect("Oracle", "user/password@SID")
if(dbExistsTable(con, "fuel_frame")){
   dbRemoveTable(conn, "fuel_frame")
   dbWriteTable(conn, "fuel_frame", fuel.frame)
}
if(dbExistsTable(conn, "RESULTS")){
   dbWriteTable(conn, "RESULTS", results2000, append = T)
else
   dbWriteTable(conn, "RESULTS", results2000)
}
## End Don't run
```

`dbSendQuery-methods` *Execute a statement on a given database connection*

Description

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

Methods

conn an `OraConnection` object.

statement a character vector of length 1 with the SQL statement.

res an `OraResult` object.

... additional parameters.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

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

Examples

```
## Don't run:
drv <- dbDriver("Oracle")
con <- dbConnect(drv, "usr", "password", "sid")
res <- dbSendQuery(con, "SELECT * from liv25")
data <- fetch(res, n = -1)
## End Don't run
```

`dbSetDataMappings-methods` *Set data mappings between Oracle and R/S-Plus*

Description

Not yet implemented

Methods

res a `OraResult` object as returned by `dbSendQuery`.

flds a data.frame with field descriptions as returned by `dbColumnInfo`.

`...` any additional arguments are passed to the implementing method.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

`Oracle`, `dbSendQuery`, `fetch`, `dbColumnInfo`.

Examples

```
## Don't run:
makeImage <- function(x) {
  .C("make_Image", as.integer(x), length(x))
}

res <- dbSendQuery(con, statement)
flds <- dbColumnInfo(res)
flds[3, "Sclass"] <- makeImage

dbSetDataMappings(rs, flds)

im <- fetch(rs, n = -1)
## End Don't run
```

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

Description

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

Details

The ROracle implementations retrieves only n records, and if n is missing it only returns up to fetch.default.rec as specified in the call to Oracle (500 by default).

Methods

res an OraResult object.

n maximum number of records to retrieve per fetch. Use n = -1 to retrieve all pending records; use a value of n = 0 for fetching the default number of rows fetch.default.rec defined in the Oracle initialization invocation.

... currently not used.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://stat.bell-labs.com/RS-DBI.

See Also

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

Examples

```
## Don't run:
drv <- dbDriver("Oracle")
con <- dbConnect(drv, "user/password@SID")
res <- dbSendQuery(con, statement = paste(
                      "SELECT w.laser_id, w.wavelength, p.cut_off",
                      "FROM WL w, PURGE P",
                      "WHERE w.laser_id = p.laser_id",
                      "ORDER BY w.laser_id"))
# we now fetch the first 100 records from the resultSet into a data.frame
data1 <- fetch(res, n = 100)
dim(data1)

dbHasCompleted(res)

# let's get all remaining records
data2 <- fetch(res, n = -1)
## End Don't run
```

`isIdCurrent` *Check whether an dbObjectId handle object is valid or not*

Description

Support function that verifies that an dbObjectId holding a reference to a foreign object is still valid for communicating with the RDBMS

Usage

```
isIdCurrent(obj)
```

Arguments

`obj`	any `dbObjectId` (e.g., `dbDriver`, `dbConnection`, `dbResult`).

Details

`dbObjectId` are R/S-Plus remote references to foreign (C code) objects. This introduces differences to the object's semantics such as persistence (e.g., connections may be closed unexpectedly), thus this function provides a minimal verification to ensure that the foreign object being referenced can be contacted.

Value

a logical scalar.

See Also

`dbDriver` `dbConnect` `dbSendQuery` `dbGetQuery` `fetch`

Examples

```
## Don't run:
cursor <- dbSendQuery(con, sql.statement)
isIdCurrent(cursor)
## End Don't run
```

`make.db.names-methods` *Make R/S-Plus identifiers into legal SQL identifiers*

Description

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

Methods

dbObj any Oracle object (e.g., `OraDriver`).

snames a character vector of R/S-Plus identifiers (symbols) from which we need to make SQL identifiers.

name a character vector of SQL identifiers we want to check against keywords from the DBMS.

unique logical describing whether the resulting set of SQL names should be unique. Its default is `TRUE`. Following the SQL 92 standard, uniqueness of SQL identifiers is determined regardless of whether letters are upper or lower case.

allow.keywords logical describing whether SQL keywords should be allowed in the resulting set of SQL names. Its default is `TRUE`

keywords a character vector with SQL keywords, namely `.SQL92Keywords` defined in the `DBI` package.

case a character string specifying whether to make the comparison as lower case, upper case, or any of the two. it defaults to `any`.

`...` currently not used.

References

The set of SQL keywords is stored in the character vector `.SQL92Keywords` and reflects the SQL ANSI/ISO standard as documented in "X/Open SQL and RDA", 1994, ISBN 1-872630-68-8. Users can easily override or update this vector.

Oracle does add some keywords to the SQL 92 standard, they are listed in the `.OraKeywords` object.

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://stat.bell-labs.com/RS-DBI`.

See Also

`Oracle`, `dbReadTable`, `dbWriteTable`, `dbExistsTable`, `dbRemoveTable`, `dbListTables`.

Examples

```
## Don't run:
# This example shows how we could export a bunch of data.frames
# into tables on a remote database.

con <- dbConnect("Oracle", "user", "password")

export <- c("trantime.email", "trantime.print", "round.trip.time.email")
tabs <- make.db.names(export, unique = T, allow.keywords = T)

for(i in seq(along = export) )
   dbWriteTable(con, name = tabs[i],  get(export[i]))
## End Don't run
```

`oraParseConParams` *Parse an Oracle connection string*

Description

Parse an oracle connections string of the form "user/password@dbname" to determine the three Oracle's connection parameters "username", "passwd" and "dbname".

Usage

```
oraParseConParams(username="", password="", dbname=ifelse(usingR(), Sys.getenv("ORACLE_SID")
```

Arguments

`username`	a character string of the form "username/passwd@dbname". Default is "".
`password`	an optional password. If non-empty and there's also a password in the connection string `username`, this password overrides the one in `username`. Default is "".
`dbname`	an optional database name (Oracle SID). If non-empty and there's also a database name in the connection string `username`, this database name overrides the one in `username`.

Details

Both `username` and `password` may be emtpy, in which case the `username` is set to "/"; this instructs Oracle to use the operating system user/password authentication (Oracle needs to be set up to do this.)

Value

A 3-element character vector with the `username`, `passwd`, and `dbname` suitable for a call to `dbConnect`.

References

http://stat.bell-labs.com/RS-DBI

See Also

`dbConnect`, `Oracle`

Examples

```
## Don't run:
   conParams <- parse.OraConParams("user/pwd@dbname")
## End Don't run
```

`oraSupport` *Support Functions*

Description

These functions are the workhorse behind the ROracle package, but users need not invoke these directly.

Usage

```
## OraDriver-related
oraInitDriver(max.con=10, fetch.default.rec = 500, force.reload=FALSE)
oraDriverInfo(obj, what)
oraDescribeDriver(obj, verbose = FALSE, ...)
oraCloseDriver(drv, ...)

## OraConnection-related
oraNewConnection(drv, username="", password="",
   dbname = if(usingR()) Sys.getenv("ORACLE_SID") else getenv("ORACLE_SID"),
   max.results = 1)
oraCloneConnection(drv, ...)
oraConnectionInfo(obj, what)
oraDescribeConnection(obj, verbose = FALSE, ...)
oraCloseConnection(con, ..., force = FALSE)
ora9.workaround(con)

## OraResult-related
oraExecStatement(ps, data = NULL, ora.buf.size = -1)
oraFetch(res, n=0, ..., ora.buf.size)
oraQuickSQL(con, statement, ...)
oraExecDirect(con, statement, ora.buf.size = 500)
oraResultInfo(obj, what)
oraDescribeResult(obj, verbose = FALSE, ...)
oraCloseResult(res, ...)

## OraPreparedStatement-related
oraPrepareStatement(con, statement, bind)
oraExecStatement(ps, data, ora.buf.size)
oraDescribePreparedStatement(obj, verbose, ...)
oraPreparedStatementInfo(obj, what, ...)
oraBoundParamsInfo(obj)

## transactions
oraCommit(conn, ...)
oraRollback(conn, ...)

## data mappings and convenience functions
oraDataType(obj, ...)
oraReadTable(con, name, row.names = "row.names", check.names = TRUE, ...)
oraWriteTable(con, name, value, field.oraTypes, row.names = TRUE,
   overwrite=FALSE, append=FALSE, ...)
```

```
oraTableFields(con, name, ...)
```

Arguments

`max.con`	positive integer specifying maximum number of open connections. The current default of 10 is hardcoded in the C code.
`fetch.default.rec`	default number of rows to fetch (move to R/S-Plus). This default is used in `oraFetch`. The default is 500.
`force.reload`	logical indicating whether to re-initialize the driver. This may be useful if you want to change the defaults (e.g., `fetch.default.rec`). Note that the driver is a singleton (subsequent inits just returned the previously initialized driver, thus this argument).
`obj`	any of the Oracle DBI objects (e.g., `OraConnection`, `OraResult`).
`what`	character vector of metadata to extract, e.g., ”version”, ”statement”, ”isSelect”.
`verbose`	logical controlling how much information to display. Defaults to `FALSE`.
`drv`	an `OraDriver` object as produced by `oraInit`.
`con`	an `OraConnection` object as produced by `oraNewConnection` and `oraCloneConnection`.
`conn`	an `OraConnection` object as produced by `oraNewConnection` and `oraCloneConnection`.
`res`	an `OraResult`, for instance as produced by `oraExecDirect`.
`ps`	an `OraPreparedStatement` object as produce by `oraPrepareStatement`.
`data`	a `data.frame` whose columns are to be bound to a prepared statement.
`bind`	a characte vector with the classes of the bound `data.frame` columns.
`ora.buf.size`	an integer less than or equal to `RS_ORA_MAX_BUFFER_SIZE` (initially set to 4096) specifying how many rows per fetch should Oracle move at a time. The ProC/C++ Oracle implementation limits the size of these buffers to `65767/sizeof(field)` per column, thus the somewhat low maximum of 4096 rows.
`username`	a character string with the Oracle's user name. It can also be any of the Oracle-recognize login strings, e.g., ”user/password” or ”user/password@dbname”.
`password`	character string with the Oracle's password.
`dbname`	character string with the Oracle System Identification (SID).
`max.results`	positive integer indicating the maximum number of results that Oracle connections will hold open. The current default of 1 is hardcoded in the C code.
`force`	logical indicating whether to close a connection that has open result sets. The default is `FALSE`.
`statement`	character string holding one (and only one) SQL statement.
`n`	number of rows to fetch from the given result set. A value of -1 indicates to retrieve all the rows. The default of 0 specifies to extract whatever the `fetch.default.rec` was specified during driver initialization `oraInit`.
`name`	character vector of names (table names, fields, keywords).
`value`	a data.frame.
`field.oraTypes`	a list specifying the mapping from R/S-Plus fields in the data.frame `value` to SQL data types. The default is `sapply(value,SQLDataType)`, see `OraSQLType`.

`row.names`	a logical specifying whether to prepend the `value` data.frame row names or not. The default is `TRUE`.
`check.names`	a logical specifying whether to convert DBMS field names into legal S names. Default is `TRUE`.
`overwrite`	logical indicating whether to replace the table `name` with the contents of the data.frame `value`. The defauls is `FALSE`.
`append`	logical indicating whether to append `value` to the existing table `name`.
`...`	placeholder for future use.

Value

`oraInitDriver` returns an `OraDriver` object.

`oraDriverInfo` returns a list of name-value metadata pairs.

`oraDescribeDriver` returns `NULL` (displays the object's metadata).

`oraCloseDriver` returns a logical indicating whether the operation succeeded or not.

`oraNewConnection` returns an `OraConnection` object.

`oraCloneConnection` returns an `OraConnection` object.

`oraConnectionInfo`returns a list of name-value metadata pairs.

`oraDescribeConnection` returns `NULL` (displays the object's metadata).

`oraCloseConnection` returns a logical indicating whether the operation succeeded or not.

`oraExecStatement` returns an `OraResult` object.

`oraFetch` returns a data.frame.

`oraQuickSQL` returns either a data.frame if the `statement` is a `select`-like or NULL otherwise.

`oraDescribeResult` returns `NULL` (displays the object's metadata).

`oraCloseResult` returns a logical indicating whether the operation succeeded or not.

`oraPrepareStatement` returns a prepared statement.

`oraExecStatement` executes (and optionally binds new data) a prepared statement.

`oraExecDirect` executes a simple (no binding) SQL statement.

`oraPreparedStatementInfo` list of prepared statement metadata.

`oraDescribePreparedStatement` a simple print out of the prepared statement status

`oraBoundParamsInfo` data frame with as many rows as bound parameters with the columns number and class for the `data.frame` bindings.

`oraReadTable` returns a data.frame with the contents of the DBMS table.

`oraWriteTable` returns a logical indicating whether the operation succeeded or not.

`oraTableFields` returns a character vector with the table `name` field names.

`oraDataType` retuns a character string with the closest

`oraResultInfo` returns a list of name-value metadata pairs.

`oraCommit` commits the current transaction in the connection.

`oraRollback` roll backs the current transaction in the connection.

Constants

`.OraPkgName` (currently `"ROracle"`), `.OraPkgVersion` (the R package version), `.OraPkgRCS` (the RCS revision), `.Oracle.NA.string` (character that Oracle uses to denote `NULL` on input), `.OraSQLKeywords` (a lot!) `.conflicts.OK`.

`safe.write` *Write a data.frame avoiding exceeding memory limits*

Description

This function batches calls to `write.table` to avoid exceeding memory limits for very large data.frames.

Usage

```
safe.write(value, file, batch, ...)
```

Arguments

`value`	a data.frame;
`file`	a file object (connection, file name, etc).
`batch`	maximum number of rows to write at a time.
`...`	any other arguments are passed to `write.table`.

Details

The function has a while loop invoking `write.table` for subsets of `batch` rows of `value`. Since this is a helper function for `oraWriteTable` has hardcoded other arguments to `write.table`.

Value

`NULL`, invisibly.

Note

No error checking whatsoever is done.

See Also

`write.table`

Examples

```
## Don't run:
   ctr.file <- file("dump.sqloader", "w")
   safe.write(big.data, file = ctr.file, batch = 25000)
## End Don't run
```

`summary-methods` *Summarize an Oracle object*

Description

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

Methods

object = "DBIObject" Provides relevant metadata information on `object`, for instance, the Oracle server file, the SQL statement associated with a result set, etc.

from object to be coerced

to coercion class

x object to `format` or `print` or `show`

Index