Package 'DBI'

May 9, 2013

Version 0.2-7

Date 2013-05-08

Title R Database Interface

Author R Special Interest Group on Databases (R-SIG-DB)

Maintainer David A. James <daj025@gmail.com>

Depends R (>= 2.15.0), methods

Imports methods

Description A database interface (DBI) definition for communication
between R and relational database management systems. All
classes in this package are virtual and need to be extended by
the various R/DBMS implementations.

License LGPL (>= 2)

Collate DBI.R Util.R zzz.R

NeedsCompilation no

Repository CRAN

Date/Publication 2013-05-09 12:09:19

R topics documented:

---

dbCallProc   *Call an SQL stored procedure*

Description

Calls a stored procedure on a remote RDBMS

Usage

```
dbCallProc(conn, ...)
```

Arguments

conn   a DBIConnection object.

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

Details

Not yet implemented.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or
<http://developer.r-project.org/db>.

---

dbCommit *DBMS Transaction Management*

Description

Commit/rollback SQL transactions

Usage

```
dbCommit(conn, ...)
dbRollback(conn, ...)
```

Arguments

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

...   any database-specific arguments.

Details

Not all database engines implement transaction management, older versions of MySQL, for instance.

Value

a logical indicating whether the operation succeeded or not.

Side Effects

The current transaction on the connections con is committed or rolled back.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or
http://developer.r-project.org/db.

See Also

dbConnect dbSendQuery dbGetQuery fetch dbCommit dbGetInfo dbReadTable

Examples

```
## Not run:
ora <- dbDriver("Oracle")
con <- dbConnect(ora)
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(Not run)
```

---

dbConnect   *Create a connection to a DBMS*

Description

Connect to a DBMS going through the appropriate authorization procedure.

Usage

```
    dbConnect(drv, ...)
    dbDisconnect(conn, ...)
```

Arguments

drv   an object that inherits from `DBIDriver`, a character string specifying the DBMS
                driver, e.g., "RPgSQL", "ROracle", "Informix", or possibly another `dbConnect`
                object.

conn   a connection object as produced by `dbConnect`.

...   authorization arguments needed by the DBMS instance; these typically include
                `user`, `password`, `dbname`, `host`, `port`, etc. For details see the appropriate `DBIDriver`.

Details

Some implementations may allow you to have multiple connections open, so you may invoke this
function repeatedly assigning its output to different objects.

The authorization mechanism is left unspecified, so check the documentation of individual drivers
for details.

Value

An object that extends `DBIConnection` in a database-specific manner. For instance `dbConnect("MySQL")`
produces an object of class `MySQLConnection`. This object is used to direct commands to the
database engine.

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

Side Effects

A connection between R/Splus and the database server is established, and the R/Splus program
becomes a client of the database engine. Typically the connections is through the TCP/IP protocol,
but this will depend on vendor-specific details.

notes

Make sure you close the connection using `dbDisconnect(conn)` when it is not longer needed.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or
http://developer.r-project.org/db.

See Also

[dbConnect dbSendQuery dbGetQuery fetch dbCommit dbGetInfo dbReadTable](#)

Examples

```
## Not run:
# create an RODBC instance and create one connection.
m <- dbDriver("RODBC")

# open the connection using user, passsword, etc., as
# specified in the file \file{$HOME/.my.cnf}
con <- dbConnect(m, dsn="data.source", uid="user", pwd="password"))

# 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(Not run)
```

---

dbDataType	*Determine the SQL Data Type of an S object*

Description

Determine an (approximately) appropriate SQL data type for an S object.

Usage

```
dbDataType(dbObj, obj, ...)
```

Arguments

dbObj	a `DBIDriver` object, e.g., `ODBCDriver`, `OracleDriver`.
obj	R/Splus object whose SQL type we want to determine.
...	any other parameters that individual methods may need.

Details

This is a generic function. The default method determines the SQL type of an R/Splus object according to the SQL 92 specification, which may serve as a starting point for driver implementations.

Value

A character string specifying the SQL data type for `obj`.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or
<http://developer.r-project.org/db>.

See Also

[isSQLKeyword](#) [make.db.names](#)

Examples

```
## Not run:
ora <- dbDriver("Oracle")
sql.type <- dbDataType(ora, x)

## End(Not run)
```

---

dbDriver   *Database Interface (DBI) Classes and drivers*

Description

These *virtual* classes and their methods define the interface to database management systems (DBMS).
They are extended by packages or drivers that implement the methods in the context of specific
DBMS (e.g., Berkeley DB, MySQL, Oracle, ODBC, PostgreSQL, SQLite).

Usage

```
    dbDriver(drvName, ...)
    dbUnloadDriver(drv, ...)      ## free up all resources
```

Arguments

drvName   character name of the driver to instantiate.

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

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

Details

The virtual class `DBIDriver` defines the operations for creating connections and defining data type mappings. Actual driver classes, for instance RPgSQL, RMySQL, etc. implement these operations in a DBMS-specific manner.

More generally, the DBI defines a very small set of classes and methods that allows users and applications access DBMS with a common interface. The virtual classes are `DBIDriver` that individual drivers extend, `DBIConnection` that represent instances of DBMS connections, and `DBIResult` that represent the result of a DBMS statement. These three classes extend the basic class of `DBIObject`, which serves as the root or parent of the class hierarchy.

Value

In the case of `dbDriver`, an driver object whose class extends `DBIDriver`. This object may be used to create connections to the actual DBMS engine.

In the case of `dbUnloadDriver`, a logical indicating whether the operation succeeded or not.

Side Effects

The client part of the database communication is initialized (typically dynamically loading C code, etc.) but note that connecting to the database engine itself needs to be done through calls to `dbConnect`.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or http://developer.r-project.org/db.

See Also

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

Examples

```
## Not run:
# create a MySQL instance for capacity of up to 25 simultaneous
# connections.
m <- dbDriver("MySQL", max.con = 25)
p <- dbDriver("PgSQL")

# open the connection using user, password, etc., as
con <- dbConnect(m, user="ip", password = "traffic", 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(Not run)
```

---

dbGetInfo   *Database interface meta-data*

Description

Extract meta-data associated with various objects

Usage

```
dbGetInfo(dbObj, ...)         # meta-data for any DBIObject
dbGetDBIVersion()             # DBI version
dbGetStatement(res, ...)      # statement that produced result "res"
dbGetRowCount(res, ...)       # number of rows fetched so far
dbGetRowsAffected(res, ...) # number of affected rows (e.g., DELETE)
dbColumnInfo(res, ...)        # result set data types
dbHasCompleted(res, ...)      # are there more rows to fetch on "res"?
```

Arguments

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

res   refers to a DBIResult object.

...   any driver-specific arguments.

Details

These functions implement a minimal set of meta-data describing the most important aspects of the
R/Splus to DBMS interface.

The dbGetInfo works very similarly to the function options in that it attempts to extract what the
user may request, possibly NULL if it can't locate the specific piece of meta-data.

Value

dbGetDBIVersion returns a character string with the version of the database interface API.

dbGetInfo produces either a character vector or a named list of (name, value) pairs.

dbGetStatement returns a character string with the statement associated with the result set res.

dbGetRowCount returns the number of rows fetched so far.

dbGetRowsAffected returns the number of affected rows (e.g., how many rows were deleted, in-
serted). Some drivers may set this to the total number of rows a query produces.

dbColumnInfo returns a data.frame with one row per output field in res. The columns should report
field name, field data type, scale and precision (as understood by the DBMS engine), whether the
field can store NULL values, and possibly other DBMS-specific information.

dbHasCompleted a logical describing whether the operations has been completed by the DBMS or
not.

Note

Meta-data associated with a driver should include the version of the package, plus the version of the underlying client library. Connection objects should report the version of the DBMS engine, database name, user, possibly password, etc. Results should include the statement being executed, how many rows have been fetched so far (in the case of queries), how many rows were affected (deleted, inserted, changed, or total number of records to be fetched).

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or http://developer.r-project.org/db.

See Also

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

Examples

```
## Not run:
drv <- dbDriver("SQLite")
con <- dbConnect(drv)

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(Not run)
```

DBIConnection-class   *Class DBIConnection*

Description

Base class for all DBMS connection classes. Individual drivers (ODBC, Oracle, PostgreSQL, MySQL, etc.) extend this class in a database-specific manner.

Objects from the Class

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

Extends

Class "DBIObject", directly.

Generator

The main generator is dbConnect.

Methods

The following methods take objects from classes derived from DBIConnection:
  Create and close connections:

dbConnect signature(drv = "DBIConnection"): ...

dbDisconnect signature(conn = "DBIConnection"): ...
  Execute SQL commands:

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

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

dbCallProc signature(conn = "DBIConnection"): ...
  Transaction management:

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

dbRollback signature(conn = "DBIConnection"): ...
  Meta-data:

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

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

summary signature(object = "DBIConnection"): ...
  Exceptions:

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

dbListFields signature(conn = "DBIConnection", name = "character"): ...
  Convenience functions:

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

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

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

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

dbWriteTable signature(conn = "DBIConnection", name = "character", value = "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://developer.r-project.org/db>.

See Also

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

Examples

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

pg <- dbDriver("PostgreSQL")
con <- dbConnect(pg, "user", "password")

## End(Not run)
```

DBIDriver-class   *Class DBIDriver*

Description

Base class for all DBMS drivers (e.g., ODBC, Oracle, MySQL, PostgreSQL).

Objects from the Class

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

Extends

Class "DBIObject", directly.

Generator

The generator for classes that extend DBIDriver is dbDriver.

Methods

The following methods are defined for classes that extend DBIDriver:

dbUnloadDriver signature(drv = "DBIDriver"): …

dbConnect signature(drv = "DBIDriver"): …

dbGetInfo signature(dbObj = "DBIDriver"): …

dbListConnections signature(drv = "DBIDriver"): …

summary signature(object = "DBIDriver"): …

Author(s)

R-SIG-DB

References

See the Database Interface definition document DBI.pdf in the base directory of this package or
http://developer.r-project.org/db.

See Also

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

The function dbConnect is the main generator.

In addition see the help of the methods above.

Examples

```
## Not run:
drv <- dbDriver("ODBC")
summary(drv)
dbListConnections(drv)

## End(Not run)
```

DBIObject-class   *Class DBIObject*

Description

Base class for all other DBI classes (e.g., drivers, connections).

Objects from the Class

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

Methods

Methods defined for classes that extend DBIObject:

dbDataType signature(dbObj = "DBIObject"): ...

isSQLKeyword signature(dbObj = "DBIObject"): ...

make.db.names signature(dbObj = "DBIObject"): ...

SQLKeywords signature(dbObj = "DBIObject"): ...

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

Plus many other specific to the other DBI classes.

Author(s)

R-SIG-DB

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or
http://developer.r-project.org/db.

See Also

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

Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv, group = "rs-dbi")
res <- dbSendQuery(con, "select * from vitalSuite")
is(drv, "DBIObject")   ## True
is(con, "DBIObject")   ## True
is(res, "DBIObject")

## End(Not run)
```

DBIResult-class   *Class DBIResult*

Description

Base class for all DBMS-specific result objects.

Objects from the Class

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

Extends

Class "DBIObject", directly.

Generator

The main generator is dbSendQuery.

Methods

Fetching methods:

signature(res = ″DBIResult″, n = ″numeric″): ...

fetch signature(res = ″DBIResult″, n = ″missing″): ... Close result set:

dbClearResult signature(res = ″DBIResult″): ...

Meta-data:

dbColumnInfo signature(res = ″DBIResult″): ...

dbGetException signature(conn = ″DBIResult″): ...

dbGetInfo signature(dbObj = ″DBIResult″): ...

dbGetRowCount signature(res = ″DBIResult″): ...

dbGetRowsAffected signature(res = ″DBIResult″): ...

dbGetStatement signature(res = ″DBIResult″): ...

dbHasCompleted signature(res = ″DBIResult″): ...

dbListFields signature(conn = ″DBIResult″, name = ″missing″): ...

summary signature(object = ″DBIResult″): ...

coerce signature(from = ″DBIConnection″, to = ″DBIResult″): ...

Author(s)

R-SIG-DB

References

See the Database Interface definition document DBI.pdf in the base directory of this package or
http://developer.r-project.org/db.

See Also

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

Examples

```
## Not run:
 drv <- dbDriver("Oracle")
 con <- dbConnect(drv, "user/password@dbname")
 res <- dbSendQuery(con, "select * from LASERS where prdata > '2002-05-01'")
 summary(res)
 while(dbHasCompleted(res)){
    chunk <- fetch(res, n = 1000)
    process(chunk)
 }

## End(Not run)
```

---

dbListTables   *List items from a remote DBMS and from objects that implement the*
                                     database interface DBI.

Description

List remote tables, fields of a remote table, opened connections and pending statements in a connection.

Usage

```
dbListTables(conn, ...)
dbListFields(conn, name, ...)
dbListConnections(drv, ...)
dbListResults(conn, ...)
```

Arguments

drv	a driver object (e.g., `ODBC`, `Oracle`)
conn	a connection object
name	a character string with the name of the remote table.
...	optional arguments for the actual driver implementation.

Value

`dbListTables` returns a character vector with the names of the tables in the remote database associated with the connection in `conn` object.

`dbListFields` returns a character vector with the names of the fields of the `res` result object (it must be a query statement).

`dbListConnections` returns a list of all currently open connections on driver `drv`. Drivers that implement single connections would return the one single connection object.

`dbListResults` returns a list of objects for all pending results (statements) on the `conn` connection.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or http://developer.r-project.org/db.

See Also

[dbGetInfo](), [dbColumnInfo](), [dbDriver](), [dbConnect](), [dbSendQuery]()

Examples

```
## Not run:
odbc <- dbDriver("ODBC")
# after working awhile...
for(con in dbListConnections(odbc)){
   dbGetStatement(dbListResults(con))
}

## End(Not run)
```

---

dbReadTable   *Convenience functions for Importing/Exporting DBMS tables*

Description

These functions mimic their R/Splus counterpart get, assign, exists, remove, and objects, ex-cept that they generate code that gets remotely executed in a database engine.

Usage

```
dbReadTable(conn, name, ...)
dbWriteTable(conn, name, value, ...)
dbExistsTable(conn, name, ...)
dbRemoveTable(conn, name, ...)
```

Arguments

conn	a database connection object.
name	a character string specifying a DBMS table name.
value	a data.frame (or coercible to data.frame).
...	any optional arguments that the underlying database driver supports, e.g.

> **row.names** in the case of dbReadTable, this argument can be a string or an in-dex 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 argu-ment should be a logical specifying whether the row.names should be out-put to the output DBMS table; if TRUE, the extra field name will be whatever the S 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.

Value

dbReadTable returns a data.frame; all other functions return TRUE or FALSE denoting whether the operation was successful or not.

Side Effects

A DBMS statement is generated and remotely executed on a database engine; the result set it produces is fetched in its entirety. These operations may failed if the underlying database driver runs out of available connections and/or result sets, or the operation violates DBMS integrity constraints (e.g., attempting to write duplicate values on a field that's defined as a primary key).

The semantics of assign are slightly extended to allow overwriting or appending to an existing table.

Note

The translation of identifiers between R/Splus and SQL is done through calls to make.names and make.db.names, but we cannot guarantee that the conversion is reversible. For details see make.db.names.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

See Also

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

Examples

```
## Not run:
conn <- dbConnect("MySQL", group = "vitalAnalysis")
con2 <- dbConnect("ODBC", "dsn", "user", "pwd")
if(dbExistsTable(con2, "fuel_frame")){
   fuel.frame <- dbReadTable(con2, "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(Not run)
```

---

dbSendQuery   *Execute a statement on a given database connection*

Description

Submits and executes an arbitrary SQL statement on a specific connection. Also, clears (closes) a result set.

Usage

```
dbSendQuery(conn, statement, ...)
dbGetQuery(conn, statement, ...)
dbClearResult(res, ...)
dbGetException(conn, ...)
```

Arguments

conn	a connection object.
statement	a character vector of length 1 with the SQL statement.
res	a result set object (i.e., the value of dbSendQuery).
...	database-specific parameters may be specified.

Details

The function dbSendQuery only submits and synchronously executes the SQL statement to the database engine. It does *not* extracts any records — for that you need to use the function [fetch](#) (make sure you invoke dbClearResult when you finish fetching the records you need).

The function dbGetQuery does all these in one operation (submits the statement, fetches all output records, and clears the result set).

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

Value

dbSendQuery returns a result set object, i.e., an object that inherits from DBIResult; if the statement generates output (e.g., a SELECT statement) the result set can be used with [fetch](#) to extract records.

dbGetQuery returns a data.frame with the output (if any) of the query.

dbClearResult returns a logical indicating whether clearing the result set was successful or not.

dbGetException returns a list with elements errNum (an integer error number) and errMsg (a character string) describing the last error in the connection conn.

Side Effects

The statement is submitted for synchronous execution to the server connected through the conn object. The DBMS executes the statement, possibly generating vast amounts of data. Where these data reside is driver-specific: some drivers may choose to leave the output on the server and transfer them piecemeal to R/Splus, others may transfer all the data to the client – but not necessarily to the memory that R/Splus manages. See the individual drivers' [dbSendQuery](#) method for implementation details.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

See Also

dbDriver dbConnect fetch dbCommit dbGetInfo dbReadTable

Examples

```
## Not run:
drv <- dbDriver("MySQL")
con <- dbConnect(drv)
res <- dbSendQuery(con, "SELECT * from liv25")
data <- fetch(res, n = -1)

## End(Not run)
```

---

dbSetDataMappings   *Set data mappings between an DBMS and R/Splus*

Description

Sets one or more conversion functions to handle the translation of DBMS data types to R/Splus objects. This is only needed for non-primitive data, since all DBI drivers handle the common base types (integers, numeric, strings, etc.)

Usage

```
    dbSetDataMappings(res, flds, ...)
```

Arguments

res	a DBIResult object as returned by dbSendQuery.
flds	a field description object as returned by dbColumnInfo.
...	any additional arguments are passed to the implementing method.

Details

The details on conversion functions (e.g., arguments, whether they can invoke initializers and/or destructors) have not been specified.

Value

a logical specifying whether the conversion functions were successfully installed or not.

Side Effects

Conversion functions are set up to be invoked for each element of the corresponding fields in the result set.

Note

No driver has yet implemented this functionality.

References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or
http://developer.r-project.org/db.

See Also

dbSendQuery, fetch, dbColumnInfo.

Examples

```
## Not 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(Not run)
```

---

fetch   *Fetch records from a previously executed query*

Description

Fetch records from a previously executed query.

Usage

```
fetch(res, n, ...)
```

Arguments

res   a result set object (one whose class extends `DBIResult`). This object needs to be
             the result of a statement that produces output, such as SQL's `SELECT` or `SELECT`-
             like statement, this object `res` is typically produced by a call to or dbSendQuery.

n   maximum number of records to retrieve per fetch. Use `n = -1` to retrieve all
             pending records. Some implementations may recognize other special values.

...   any other database-engine specific arguments.

Details

See the notes for the various database server implementations.

Value

a data.frame with as many rows as records were fetched and as many columns as fields in the result set.

Side Effects

As the R/Splus client fetches records the remote database server updates its cursor accordingly.

Note

Make sure you close the result set with dbClearResult as soon as you finish retrieving the records you want.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

See Also

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

Examples

```
## Not run:
# Run an SQL statement by creating first a resultSet object
drv <- dbDriver("ODBC")
con <- dbConnect(drv, ...)
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(Not run)
```

---

make.db.names   *Make R/Splus identifiers into legal SQL identifiers*

Description

Produce legal SQL identifiers from a character vector.

Usage

```
make.db.names(dbObj, snames, keywords, unique=TRUE, allow.keywords=TRUE, ...)
SQLKeywords(dbObj, ...)
isSQLKeyword(dbObj, name, keywords=.SQL92Keywords,
                  case=c("lower", "upper", "any")[3], ...)
```

Arguments

dbObj   any DBI object (e.g., DBIDriver).

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

name   a character vector with database identifier candidates we need to determine
                   whether they are legal SQL identifiers or not.

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, by default it's .SQL92Keywords defined
                   by the DBI.

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

...   any other argument are passed to the driver implementation.

Details

The algorithm in make.db.names first invokes make.names and then replaces each occurrence of
a dot "." by an underscore "_". If allow.keywords is FALSE and identifiers collide with SQL
keywords, a small integer is appended to the identifier in the form of "_n".

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.

Value

`make.db.names` returns a character vector of legal SQL identifiers corresponding to its `snames` argument.

`SQLKeywords` returns a character vector of all known keywords for the database-engine associated with `dbObj`.

`isSQLKeyword` returns a logical vector parallel to `name`.

Bugs

The current mapping is not guaranteed to be fully reversible: some SQL identifiers that get mapped into S identifiers with `make.names` and then back to SQL with `make.db.names` will not be equal to the original SQL identifiers (e.g., compound SQL identifiers of the form `username.tablename` will loose the dot ".").

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.

See the Database Interface definition document `DBI.pdf` in the base directory of this package or `http://developer.r-project.org/db`.

See Also

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

Examples

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

con <- dbConnect("Oracle", user="iptraffic", pass = pwd)

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]))

# Oracle's extensions to SQL keywords
oracle.keywords <- c("CLUSTER", "COLUMN", "MINUS", "DBNAME")
isSQLKeyword(nam, c(.SQL92Keywords, oracle.keywords))
[1]  T  T  T  F

## End(Not run)
```

---

print.list.pairs   *Support functions*

Description

Some of these functions are conditionally elevated to generic functions (e.g., print, summary). Others are low-level support functions.

Usage

```
## S3 method for class 'list.pairs'
print(x, ...)
```

Arguments

x   a list of key, value pairs

...   additional arguments to be passed to cat

Value

the (invisible) value of x.

References

See the Database Interface definition document DBI.pdf in the base directory of this package or http://developer.r-project.org/db.

See Also

print.default, summary.default, cat.

Examples

```
## Not run:
print.list.pairs(list(a = 1, b = 2))

## End(Not run)
```

Index