**Connecting to Oracle Database 12c from Oracle R with Roracle**

<http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle>

Connecting to Oracle Database from Oracle R with ROracle

R is an open source software for computing statistics and generating statistical graphs. The Oracle R Distribution is a repackaged enhanced version of the open source R. ROracle is a high performance R package that makes use of Oracle Call Interface (OCI) libraries for Oracle Database connection handling. In this tutorial we shall discuss connecting to Oracle Database 12c from the Oracle R Distribution using ROracle. This tutorial has the following sections.

[Overview of ROracle Methods](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#OverviewROracle)

[Setting the Environment](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#SettingEnv)

[Loading the ROracle Library](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#LoadingROracle)

[Loading the Oracle Database Driver](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#LoadingOracleDBDriver)

[Creating a Connection](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#CreatingAConnection)

[Creating an Oracle Database Table](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#CreatingOraDBTable)

[Adding Table Data](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#AddingTableData)

[Running an SQL SELECT Query](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#RunningSQLSelect)

[Fetching and Displaying Query Result](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#FetchingAndDisplayingQueryData)

[Listing Fields, Column Info and Tables](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#ListingFields)

[Committing and Rolling Back a Transaction](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#CommittingRolling)

[Getting Metadata](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#GettingMetadata)

[Closing Resources](http://www.toadworld.com/platforms/oracle/w/wiki/11057.connecting-to-oracle-database-12c-from-oracle-r-with-roracle#ClosingResources)

Overview of ROracle Methods

ROracle provides various methods to connect to Oracle Database, run SQL statements, fetch results, commit/rollback a transaction, get metadata, and close connection resources. The main methods, which we shall discuss in this tutorial, are discussed in following table.

|  |  |
| --- | --- |
| **Method** | **Description** |
| Oracle | Creates and instantiates an Oracle client and returns an object with a connection to Oracle Database may be established. |
| dbConnect | Creates a connection object with which SQL queries may be run on Oracle Database. |
| dbSendQuery | Runs an SQL query statement but does not fetch the data, which has to be fetched with the fetch method. |
| dbGetQuery | Runs an SQL statement and fetches the data. A separate invocation of fetch is not required. |
| fetch | Fetches the data returned by a previously run query. |
| dbCommit | Commits the current transaction in an Oracle connection. |
| dbGetInfo | Get meta data |
| dbColumnInfo | Get column meta data |
| dbRollback | Rollback the current transaction in an Oracle connection |
| dbDisconnect | Disconnect the current connection and free up all resources associated with the connection object |
| dbDriver | Loads the Oracle Database driver. |
| dbUnloadDriver | Unloads the Oracle Database driver |
| dbListTables | Lists tables |
| dbListConnections | Lists connections |
| dbListResults | Returns a list all associated result sets |
| dbClearResult | Clears the result set and frees up the associated resources |
| dbWriteTable | Adds row/s to a new table or appends row/s to an existing table. Auto-commits current transaction and the data added or appended |
| dbExistsTable | Returns a boolean to indicate if a table exists |
| dbRemoveTable | Removes a table |
| dbGetStatement | Gets the SQL statement associated with a query |
| dbHasCompleted | Returns a boolean to indicate if a query has completed |
| dbGetRowsAffected | Returns the number of rows affected by a SQL query |
| dbGetRowCount | Returns the number of rows affected by a SQL SELECT query |
| show | A meta data method for a driver, result set, connection object |
| summary | A meta data method for summary of a driver, result set, connection object |

Setting the Environment

First, download the following software.

-Oracle Database 12c from *http://www.oracle.com/us/corporate/features/database-12c/index.html*.

-Oracle R Distribution .exe file from *http://www.oracle.com/technetwork/database/database-technologies/r/r-distribution/downloads/index.html*.

-ROracle zip file from *http://www.oracle.com/technetwork/database/database-technologies/r/roracle/downloads/index.html*.

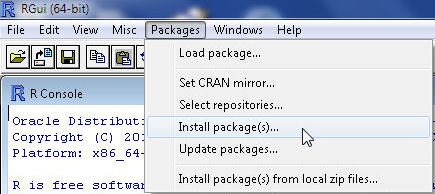
Windows OS is used in this article. Install Oracle Database 12c. Double-click on the Oracle R Distribution to install Oracle R. Start the Oracle R GUI. Select Packages>Install package(s) from local zip files.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/4174.image001.jpg)

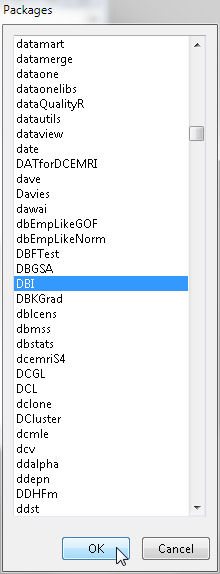
The ROracle package gets installed.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/8308.image002.jpg)

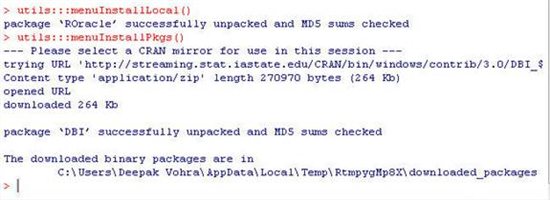
The ROracle package has a dependency on the DBI package. Next, install the DBI package. Select Packages>Install package(s).

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/0474.image003.jpg)

Select a CRAN mirror and click on OK. Select the DB1 package and click on OK.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/2248.image004.jpg)

The DBI package gets installed.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/7041.image005.jpg)

Loading the ROracle Library

To load the ROracle package run the following command in the R command shell.

library(ROracle)

The ROracle library gets loaded.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/0486.image006.jpg)

Loading the Oracle Database Driver

Next, load the Oracle Database driver. Each of the following commands loads the Oracle Database driver.

driver<- Oracle()

driver <- dbDriver("Oracle")

After the Oracle driver has been loaded a connection with Oracle Database may be established.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/3531.image007.jpg)

Creating a Connection

The

dbConnect(drv, username = "", password = "", dbname = "", prefetch = FALSE,

bulk\_read = 1000L, stmt\_cache = 0L, external\_credentials = FALSE,

sysdba = FALSE, ...) method is used to connect to Oracle Database from R using ROracle OCI. At the minimum a driver instance, and username and password must be provided to connect to a local Oracle database instance.

connection <- dbConnect(driver, username = "OE", password = "OE")

To verify that a connection has been established run the following dbListTables command.

dbListTables(connection, schema = "OE", all = FALSE, full = FALSE)

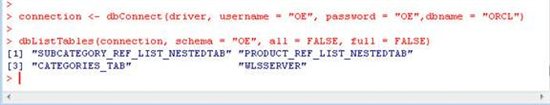
The database tables in the OE schema get listed.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/0676.image008.jpg)

An Oracle instance may also be specified in dbConnect with the dbname attribute.

connection <- dbConnect(driver, username = "OE", password = "OE",dbname = "ORCL")

Subsequently run the dbListTables command to list the Oracle Database tables in the OE schema.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/8611.image009.jpg)

Another option to connect to Oracle Database is to use a connect string constructed from the hostname, port and SID as follows.

host <- "localhost"

port <- 1521

sid <- "ORCL"

connect.string <- paste(

"(DESCRIPTION=",

"(ADDRESS=(PROTOCOL=tcp)(HOST=", host, ")(PORT=", port, "))",

"(CONNECT\_DATA=(SID=", sid, ")))", sep = "")

connection <- dbConnect(driver, username = "OE", password = "OE",

dbname = connect.string)

Subsequently the tables in OE schema get listed using the connection object.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/4118.image010.jpg)

Creating an Oracle Database Table

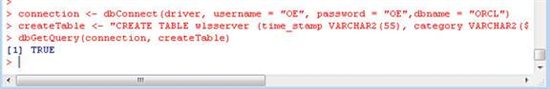
Next, create an Oracle Database table. A method specific to creating an Oracle Database table is not provided. Use one of the dbGetQuery or dbSendQuery to run an SQL statement to create a table. Create a connection object, and specify a SQL statement to create an Oracle Database table. Run the SQL statement using the dbGetQuery method with the connection object and the SQL statement string as args.

connection <- dbConnect(driver, username = "OE", password = "OE",dbname = "ORCL")

createTable <- "CREATE TABLE wlsserver (time\_stamp VARCHAR2(55), category VARCHAR2(15), type VARCHAR2(55), servername VARCHAR2(15), code VARCHAR2(15), msg VARCHAR2(255))"

dbGetQuery(connection, createTable)

The command returns TRUE to indicate that the SQL statement ran without error. A database table WLSSERVER gets created in the OE schema.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/2248.image011.jpg)

Adding Table Data

Next, add rows to the wlsserver table using INSERT SQL statement, which is run with the dbGetQuery method. Create an INSERT statement with bind variable position holders for the bind data. Create variables for the bind data. Invoke the dbGetQuery method with the connection object, the INSERT statement string and the bind data as args.

insertString <- "insert into wlsserver values(:1, :2, :3, :4, :5, :6)";

time\_stamp <- "Apr-8-2014-7:06:16-PM-PDT";

category <- "Notice";

type <- "WebLogicServer";

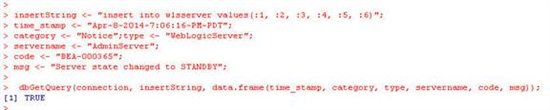
servername <- "AdminServer";

code <- "BEA-000365";

msg <- "Server state changed to STANDBY";

dbGetQuery(connection, insertString, data.frame(time\_stamp, category, type, servername, code, msg));

When the dbGetQuery is run one row gets added to the wlsserver table.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/7522.image012.jpg)

Multiple row data may be bound in a single data.frame bind data object. For example bind data for two rows is created as follows.

time\_stamp1 <- "Apr-8-2014-7:06:17-PM-PDT";

msg1 <- "Server state changed to STARTING";

time\_stamp2 <- "Apr-8-2014-7:06:18-PM-PDT";

msg2 <- "Server state changed to ADMIN";

log.data = data.frame(time\_stamp = c(time\_stamp1, time\_stamp2), category= c("Notice", "Notice"), type= c("WebLogicServer", "WebLogicServer"), servername= c("AdminServer", "AdminServer"), code= c("BEA-000365", "BEA-000365"), msg = c(msg1, msg2));

Invoke the dbSendQuery method with a connection object, an INSERT statement, and the bind data as args.

rs <- dbSendQuery(connection, "insert into WLSSERVER (time\_stamp, category, type, servername, code, msg) values (:1, :2, :3, :4, :5, :6)",

data = log.data)

When the dbSendQuery is run two rows get added.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/4186.image013.jpg)

Using the dbGetQuery method add two more rows. The same bind variable values may be reused if some of the column values are the same in different rows.

time\_stamp <- "Apr-8-2014-7:06:19-PM-PDT";

msg <- "Server state changed to RESUMING";

dbGetQuery(connection, insertString, data.frame(time\_stamp, category, type, servername, code, msg));

time\_stamp <- "Apr-8-2014-7:06:20-PM-PDT";

code <- "BEA-000331";

msg <- "Started WebLogic AdminServer";

dbGetQuery(connection, insertString, data.frame(time\_stamp, category, type, servername, code, msg));

When the preceding dbGetQuery method is run two more rows get added.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/7433.image014.jpg)

An INSERT statement may also be constructed without using bind position holders and a data.frame object as follows.

insertRow <- "INSERT INTO WLSSERVER (time\_stamp, category, type, servername, code, msg) values ('Apr-8-2014-7:06:21-PM-PDT', 'Notice', 'WebLogicServer', 'AdminServer', 'BEA-000365', 'Server state changed to RUNNING')"

Run the INSERT statement using the dbSendQuery method with a connection object and the INSERT statement string as args.

queryresult <-dbSendQuery(connection, insertRow);

Another row gets added.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/1362.image015.jpg)

The dbWriteTable method may be used to create a new table and add rows or append to an existing table. Specify the row data to be added as variables. Invoke the dbWriteTable method with a connection object, the database table name, a data.frame bind data object as args. Specify append=TRUE and overwrite=FALSE to append data and not overwrite. Specify the schema name with OE.

time\_stamp <- "Apr-8-2014-7:06:22-PM-PDT";

code <- "BEA-000360";

msg <- "Server started in RUNNING mode";

dbWriteTable(connection, "WLSSERVER", data.frame(time\_stamp, category, type, servername, code, msg), overwrite = FALSE,

append = TRUE, schema = "OE")

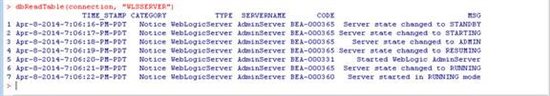
When the preceding dbWriteTable method is invoked another row gets added to the OE.WLSSERVER table.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/1258.image016.jpg)

The data added may be output by invoking the dbReadTable method with a connection object and table name as args.

dbReadTable(connection, "WLSSERVER")

The 7 rows added get listed.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/3527.image017.jpg)

Running an SQL SELECT Query

To select table data run a SELECT SQL statement using either the dbSendQuery method or the dbGetQuery method. For example, the following statement selects all rows in the OE.WLSSERVER table.

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

When the preceding statement is run a result set object is returned, but the data in the result set object is not fetched.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/7532.image018.jpg)

To fetch the data in the result set invoke the fetch method. The dbGetRowCount method returns the number of rows in the result set data. The dbGetRowsAffected method returns the number of rows affected by the by the SQL statement.

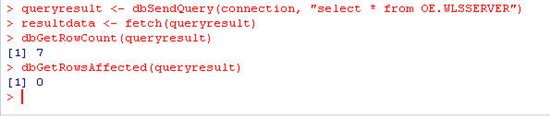
queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

resultdata <- fetch(queryresult)

dbGetRowCount(queryresult)

dbGetRowsAffected(queryresult)

As the query returns 7 rows the dbGetRowCount method returns ‘7’. As the SELECT statement does not affect any rows the dbGetRowsAffected method returns ‘0’.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/4503.image019.jpg)

Fetching and Displaying Query Result

Next, we shall fetch and display the data in the result set object. The following statement fetches the result set data into the resultdata object.

resultdata <- fetch(queryresult)

To display the data fetched with fetch do not assign the result set data to a variable.

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

fetch(queryresult)

The result set data gets displayed.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/7635.image020.jpg)

A specific number of rows may be fetched by specifying the number of rows as an arg to the fetch method.

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

fetch(queryresult, n = 5)

The preceding fetch fetches only 5 rows from the result set object.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/3021.image021.jpg)

The execute method may be used to run a query statement but not return a result set object (as the dbSendQuery does) or return the result set data (as the dbGetQuery method does).

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

execute(queryresult)

The execute method returns NULL.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/7433.image022.jpg)

The dbHasCompleted method may be used to find if a query statement has completed. A query statement has not completed till all the data in the query statement result set has been fetched. For example invoke the dbSendQuery method to run a query statement but do not fetch the result set data. Invoke the dbHasCompleted method subsequent to the dbSendQuery method invocation. The dbHasCompleted should return FALSE as the query result data has not been fetched yet.

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

dbHasCompleted(queryresult)

If the dbHasCompleted method is invoked after fetching data with the fetch method the query statement is indicated to have completed with a return value of TRUE.

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

resultdata <- fetch(queryresult)

dbHasCompleted(queryresult)

Without fetching result set the dbHasCompleted returns FALSE and after fetching the result set data the dbHasCompleted method returns TRUE.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/6646.image023.jpg)

All the data in the result set must be fetched for the dbHasCompleted to return TRUE. For example, fetch only 5 rows from a result set that has 7 rows and invoke the dbHasCompleted method.

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

fetch(queryresult, n = 5)

dbHasCompleted(queryresult)

The dbHasCompleted method returns FALSE as not all the data in the query result has been fetched yet.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/8103.image024.jpg)

In contrast to the dbSendQuery method the dbGetQuery method returns the result data as an array of column data. For example, run a query statement and output the result data as follows.

resultdata <- dbGetQuery(connection, "select \* from OE.WLSSERVER")

resultdata [1]

resultdata [2]

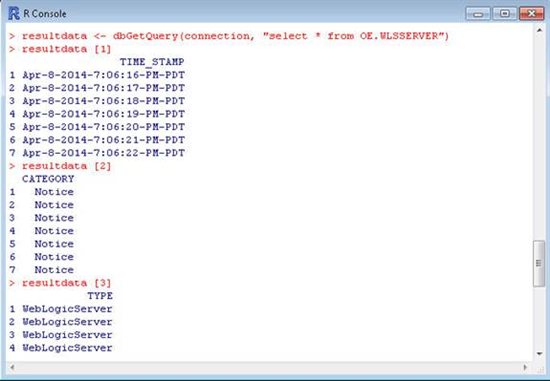
resultdata [3]

resultdata [4]

resultdata [5]

resultdata Devil

The resultdata [1] displays the first column, the resultdata [2] the second column and resultdata Devilthe 6th column.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/2352.image025.jpg)

The output from the dbGetQuery is listed:

resultdata <- dbGetQuery(connection, "select \* from OE.WLSSERVER")

> resultdata [1]

TIME\_STAMP

1 Apr-8-2014-7:06:16-PM-PDT

2 Apr-8-2014-7:06:17-PM-PDT

3 Apr-8-2014-7:06:18-PM-PDT

4 Apr-8-2014-7:06:19-PM-PDT

5 Apr-8-2014-7:06:20-PM-PDT

6 Apr-8-2014-7:06:21-PM-PDT

7 Apr-8-2014-7:06:22-PM-PDT

> resultdata [2]

CATEGORY

1 Notice

2 Notice

3 Notice

4 Notice

5 Notice

6 Notice

7 Notice

> resultdata [3]

TYPE

1 WebLogicServer

2 WebLogicServer

3 WebLogicServer

4 WebLogicServer

5 WebLogicServer

6 WebLogicServer

7 WebLogicServer

> resultdata [4]

SERVERNAME

1 AdminServer

2 AdminServer

3 AdminServer

4 AdminServer

5 AdminServer

6 AdminServer

7 AdminServer

> resultdata [5]

CODE

1 BEA-000365

2 BEA-000365

3 BEA-000365

4 BEA-000365

5 BEA-000331

6 BEA-000365

7 BEA-000360

> resultdata Devil

MSG

1 Server state changed to STANDBY

2 Server state changed to STARTING

3 Server state changed to ADMIN

4 Server state changed to RESUMING

5 Started WebLogic AdminServer

6 Server state changed to RUNNING

7 Server started in RUNNING mode

>

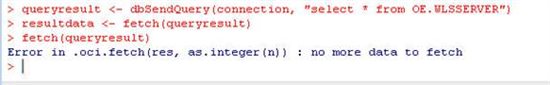
Once the result set data has been fetched with fetch method subsequent invocation of fetch generates an error. For example, re-invoke the fetch method after fetching the result set data.

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

resultdata <- fetch(queryresult)

fetch(queryresult)

An error “no more data to fetch” is returned.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/8304.image026.jpg)

Listing Fields, Column Info and Tables

The dbExistsTable method may be used to find if a table exists. For example the following dbExistsTable invocation returns a boolean to indicate if the OE.WLSSERVER table exists.

dbExistsTable(connection, "WLSSERVER", schema = "OE")

The fields in the OE.WLSSERVER method may be listed with the dbListFields method as follows.

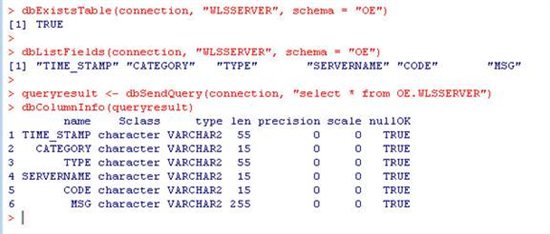
dbListFields(connection, "WLSSERVER", schema = "OE")

The column metadata in a query result set may be output using the dbColumnInfo method.

queryresult <- dbGetQuery(connection, "select \* from OE.WLSSERVER")

dbColumnInfo(queryresult)

The dbExistsTable method returns TRUE. The dbListFields method lists the fields in the OE.WSLSERVER table. The dbColumnInfo method lists column metadata in the result set for a query statement to select all rows in the OE.WLSSERVER table.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/7041.image027.jpg)

Committing and Rolling Back a Transaction

The dbCommit method is used to save permanently in the database all changes applied in a connection object. The dbRollback method is used to roll back the changes to the previous save point. For example, run a query to delete all rows from the OE.WLSSERVER table and rollback the transaction if the number of rows affected by the query is more than 0.

queryresult <- dbSendQuery(connection, "DELETE from WLSSERVER")

if(dbGetInfo(queryresult, what = "rowsAffected") > 0)

{

warning("Don't delete data -- rolling back transaction")

dbRollback(connection)

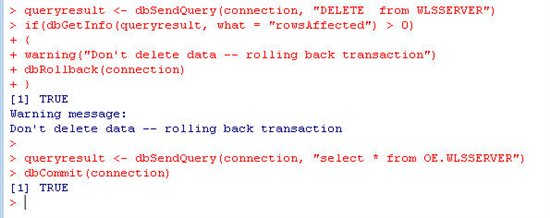
}

To commit a query run with dbSendQuery invoke the dbCommit method with the connection object used to run the query as the arg.

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

dbCommit(connection)

When the preceding statements are run the connection object used to delete all rows from a table is rolled back to the previous save point as the number of rows affected is more than 0. The dbCommit method returns TRUE after the changes in the connection object have been saved permanently regardless of the whether the query statement actually affects any rows.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/8306.image028.jpg)

Getting Metadata

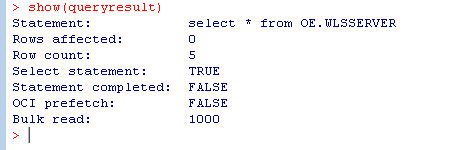
Several methods are provided to output the metadata. The following show method invocation outputs the result set metadata.

queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER")

fetch(queryresult, n = 5)

show(queryresult)

The output includes the query statement run, whether the statement is a Select statement, the number of rows affected, whether the statement completed, and the row count.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/2746.image029.jpg)

The summary method is used to output the summary of a driver, connection, result set or result set data object. For example, run a query to select all rows from OE.WLSSERVER table which have code BEA-000360. Subsequently fetch the result set data and output the summary of the result set data. Also output the dimension of the result set data using the dim method.

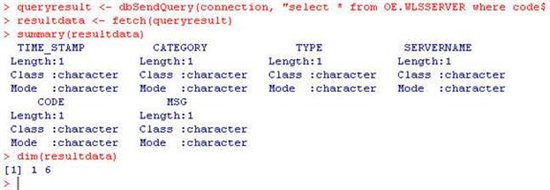
queryresult <- dbSendQuery(connection, "select \* from OE.WLSSERVER where code = :1",data = data.frame(code = "BEA-000360"))

summary(resultdata)

dim(resultdata)

The summary method returns the columns in the result set data and the number of rows.

The dim method returns the dimension as 1 6, which implies 1 row and 6 columns.

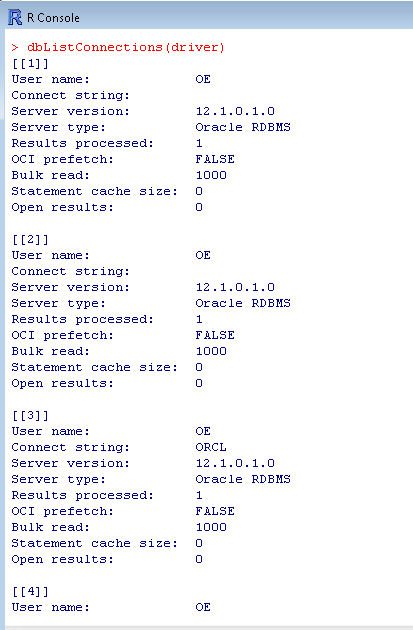
[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/0218.image030.jpg)

The dbListConnections method returns all connections associated with a driver object.

driver<- Oracle()

dbListConnections(driver)

The connection information includes the user name, the connect string if any, the Oracle database server version, the server type, the number of results processed and open results.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/5353.image031.jpg)

The dbGetInfo method returns metadata information about the a driver, connection or result set object. The names method may be used to output the attribute names in the metadata returned by the dbGetInfo method. For example, output the metadata about a driver and the attributes in the metadata as follows.

names(dbGetInfo(driver))

dbGetInfo(driver)

The driver name, version, client version, total number of connections, open connections, and connection metadata for each connection is returned.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/4812.image032.jpg)

The result sets associated with a connection are returned with the dbListResults method with the connection object as arg.

dbListResults(connection)

The output incudes each of the query statements, the number of rows affected by the query, the row count, whether the query statement is a Select statement, and whether the query statement completed are returned.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/3125.image033.jpg)

Metadata about a connection object may be output using the dbGetInfo method with the connection object as an arg.

dbGetInfo(connection)

The user name, the Oracle database instance, the database version, the server type, the total number of result sets associated with the connection, and metadata about each of the result set objects is output.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/6562.image034.jpg)

To output the query statement for a result set object invoke the dbGetStatement method. To output a result set metadata invoke the dbGetInfo method with the result set object as arg. The names method may be used to output the metadata attributes for a result set metadata.

dbGetStatement(queryresult)

names(dbGetInfo(queryresult))

dbGetInfo(queryresult)

The dbGetStatement method returns the associated query statement. The dbGetInfo also returns the query statement in addition to whether the statement is a Select statement, the number of rows affected, whether the statement completed, and fields in the result set.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/4380.image035.jpg)

Closing Resources

As a best practice all the resources associated with a Oracle database connection session must be closed after completing the connection session. To clear a result set object invoke the dbClearResult method. If a database table is to be deleted invoke the dbRemoveTable method. To disconnect a connection use the dbDisconnect method and to unload a database driver object invoke the dbUnloadDriver method.

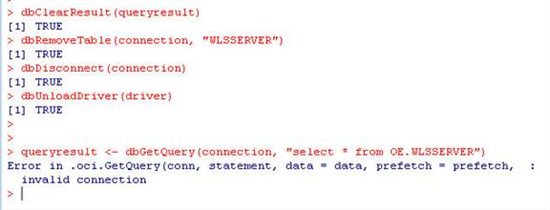
dbClearResult(queryresult)

dbRemoveTable(connection, "WLSSERVER")

dbDisconnect(connection)

dbUnloadDriver(driver)

Each of these methods return TRUE to indicate that the resource has been closed/removed. If a query making use of any of the closed resources is run an error is generated. For example, if a closed connection object is used an “invalid connection” error is generated.

[](http://www.toadworld.com/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-03/1665.image036.jpg)

In this tutorial we discussed connecting to Oracle Database from Oracle R using ROracle.

Glossary

[Oracle](http://www.toadworld.com/platforms/oracle/w/wiki/tags/Oracle)