How to create a seamless R programming environment on Windows 10

By Joseph DeArce 4/10/2017

This document describes the necessary steps for the installation of **ROracle** driver/package on Windows 10 using several additional software packages to create a seamless R data processing environment. These four other software packages are necessary for the creation of a complete programming environment from a R client to a database server. You might ask why I should go through the trouble of installing these complicated software products, my answer is simple **SPEED** this connection is faster than **ODBC**, **JDBC** and anything else because it's a native connection. If you're interested in processing large amounts of data and they reside in **RDBM** databases, and you use R then **ROracle** is for you.

In benchmark comparisons, **ROracle** performed up to 79 times faster than **RJDBC** and 2.5 times faster than **RODBC** for reading data across a range of 1000 to 1 million rows, and 10 to 1000 columns. **ROracle** shows scalability across **NUMBER**, **VARCHAR2**, **TIMESTAMP**, and **BINARY_DOUBLE** data types.

Hardware

Before we begin our installation, process lets describe our test system; it has the following configuration:

- An Acer Laptop running Windows 10 with a Core7i quad processor,
- 16GB of memory.
- 1TB hard drive (not SSD).
- Installed is an Oracle 12c Enterprise database with an SGA of 4GB
- This database has a custom configuration.
- There are five PDB's on the Oracle 12c R1 instance.

Prerequisites

There are several prerequisites for the installation of ROracle, see the list below:

1) The first step is to install the R language software (CRANS) from the main R software website, see the URL below. This site also has tutorials documentation, books, FAQs, and other resources.

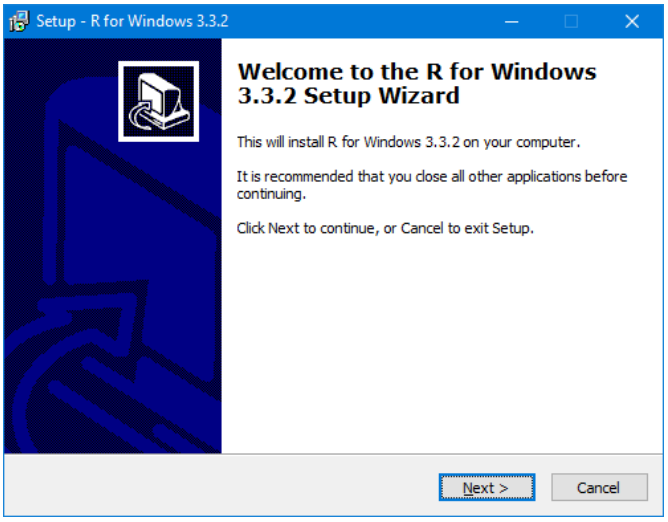
https://www.r-project.org/

Download page

https://cran.r-project.org/bin/windows/base/

On the main download page, there are many resources for novices, FAQs, installation instructions, a compatibility list, instructions on how to update packages and New Features. On other pages, there are manuals and tutorials. The current version is 3.2.2 when you have downloaded the file double-click on the file.

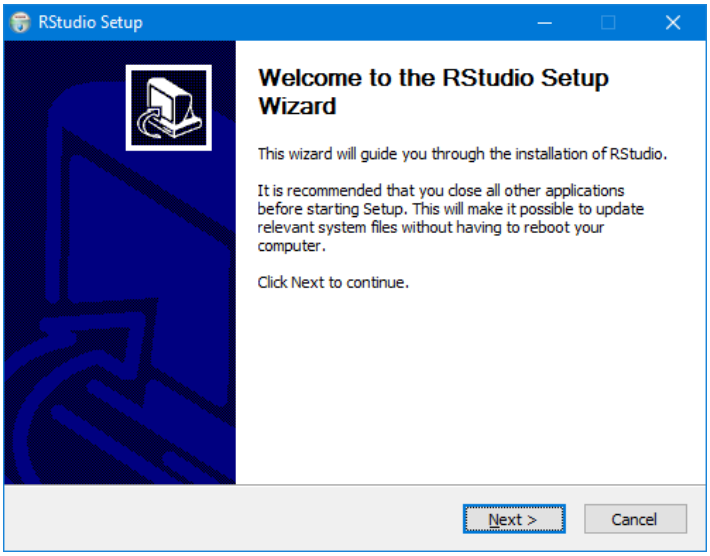
Follow the installation instructions.



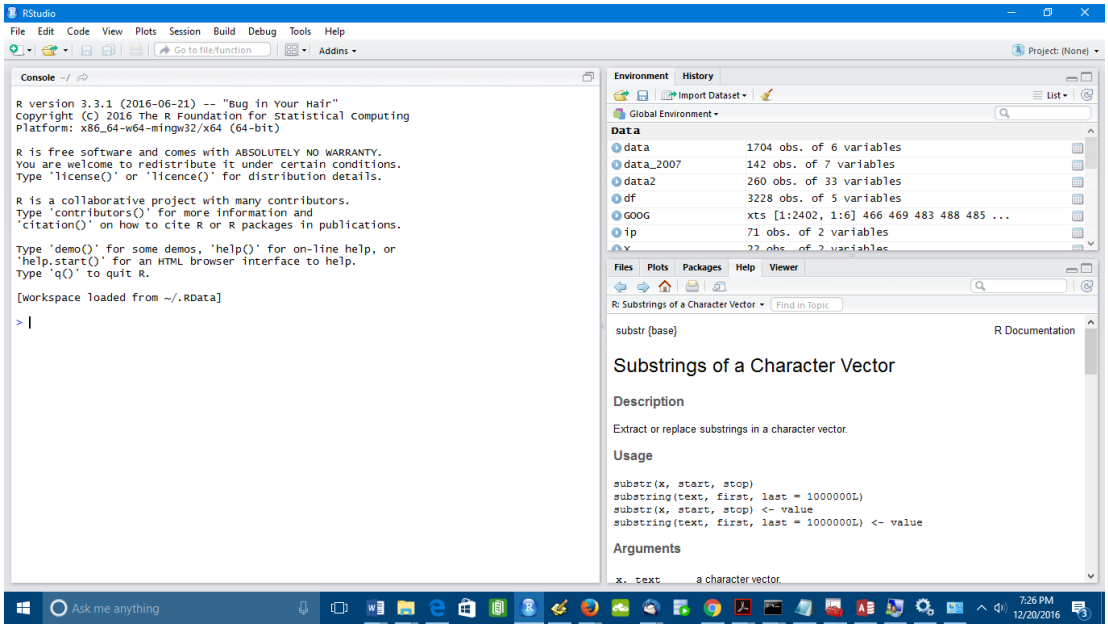
2) We must install an IDE for R we will do this to make programing in R easier and more intuitive. There are several IDEs to choose from, and a simple Google search will locate them for you. I picked RStudio which is one of the better-known ones. RStudio can be installed from the URL below.

https://www.rstudio.com/products/rstudio/download3/

Click on the download link and wait until the file is downloaded to your machine then double-click on it, you will see the screen below then follow the instructions of the installer. The most current version I found was "**RStudio 1.0.44 - Windows Vista/7/8/10**".



When the installer has finished and RStudio is installed, select it and bring it up. When RStudio comes up you will see the main screen for the program, see below for an example.



The programming environment is divided into three panels; the main programming panel takes up half the window, the two additional panels show the environment and history. The others have several tabs but open to the documentation page, plots and files will also be displayed here.

3) Installing Oracle Client 12c

You will need to install the Oracle client to get access to the OCI libraries which the ROracle depends on and create several environmental variables and add several configuration files to the install directory. You can get the client software from the main Oracle database page. Here you will be asked to agree to the Oracle OTN license which you must agree to, to be able to download the software.

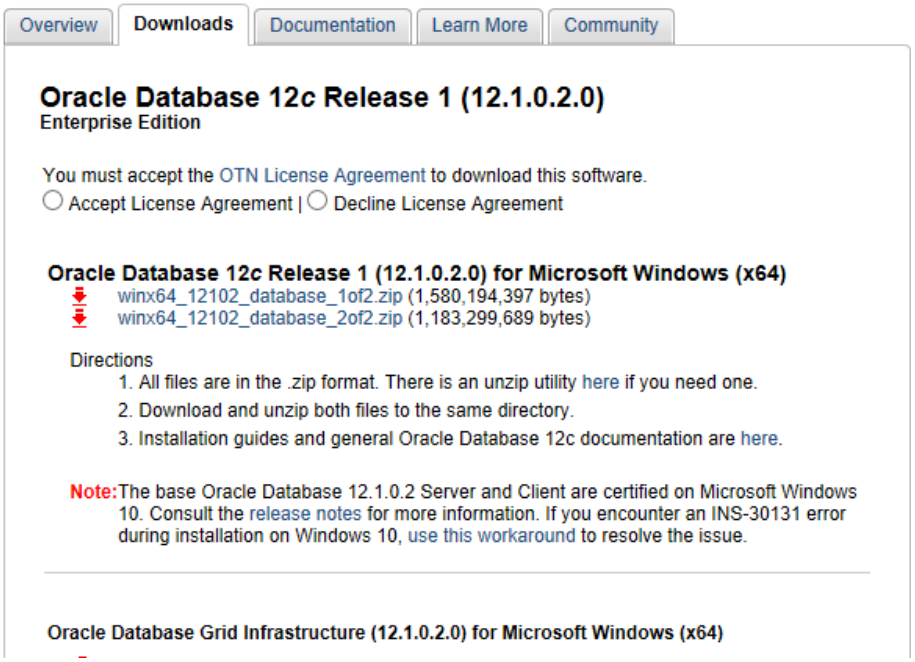
You should also sign up for an OTN account since it is free. The Oracle site can be very helpful for getting timely information on Oracle developments for all their products. Also, Oracle has many blogs by their development teams there. These blogs have a great deal of information and insights on development trends.

First step is to download the Oracle client from the main database website, use the URL below.

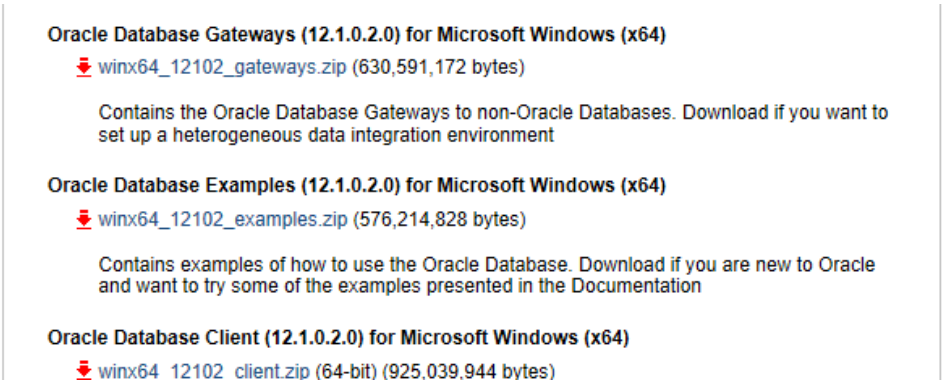
http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html

Click on the downloads tab and agree to the OTN license, then select the file you wish to download, in our case it's the 64-bit Oracle client for 12c.

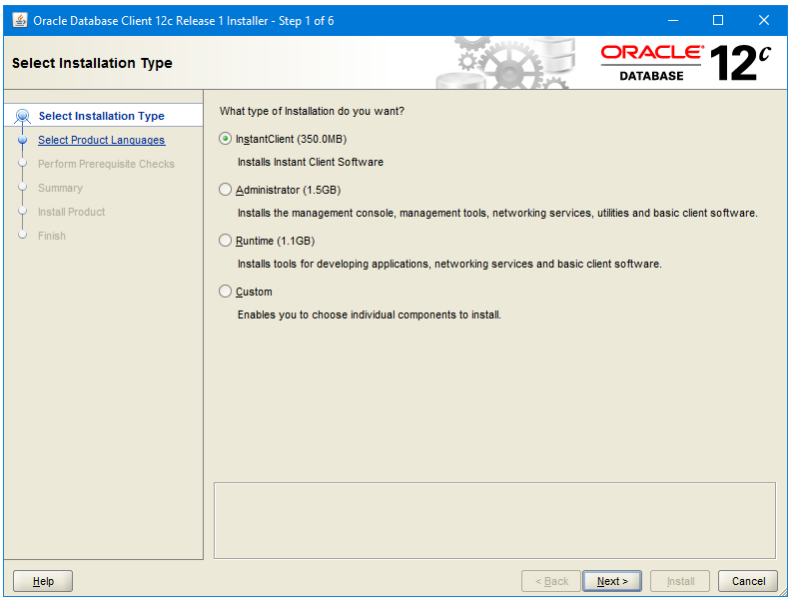
Select the **'Oracle Database 12c Release 1 (12.1.0.2)**' and then click the '**See All**' hyperlink.



Click on the '**Oracle Database Client (12.1.0.2) for Windows (64x)**' and wait until the file is downloaded.

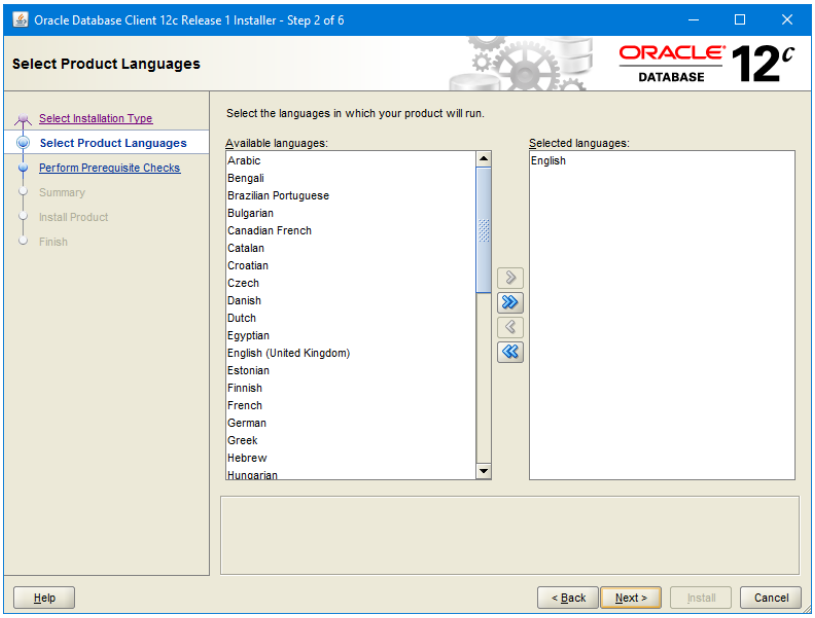


Move the file to your temporary working directory and unzip the software. Double-click on the setup.exe file to begin the install.

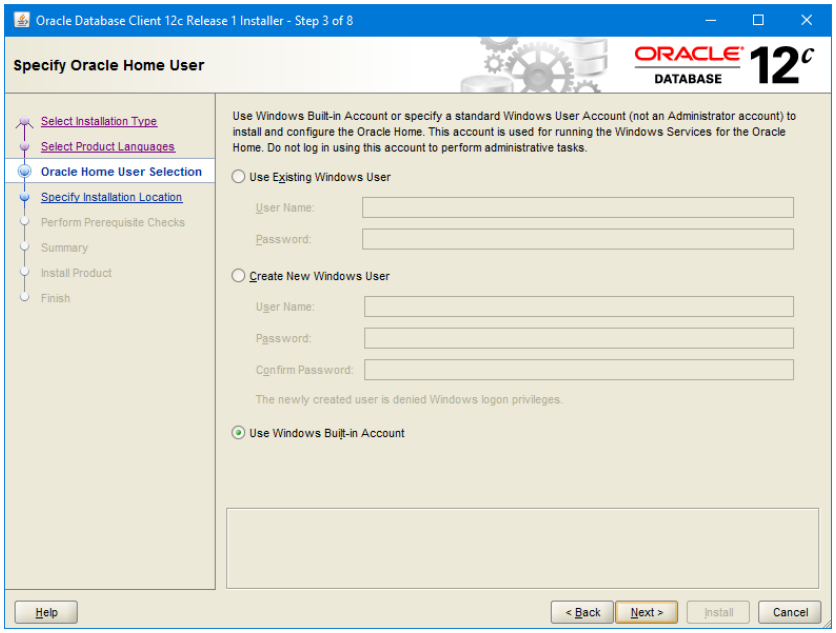


Select your install option, I choose the '**Administrator**' option, this option will lead to less steps being done than using the '**InstantClient**' option to install and configure the **ROracle** software. The instant Client option requires more steps and a more extensive configuration.

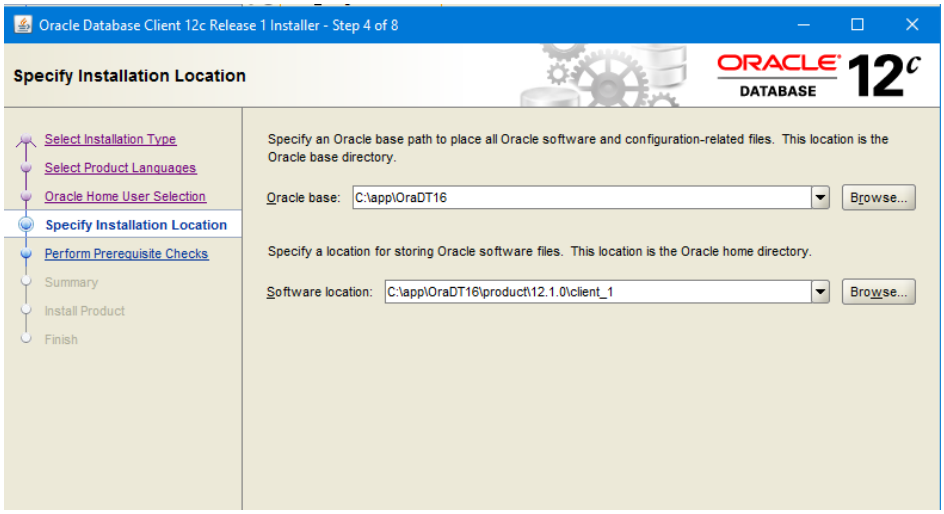
Chose a language and click '**Next**'**.**



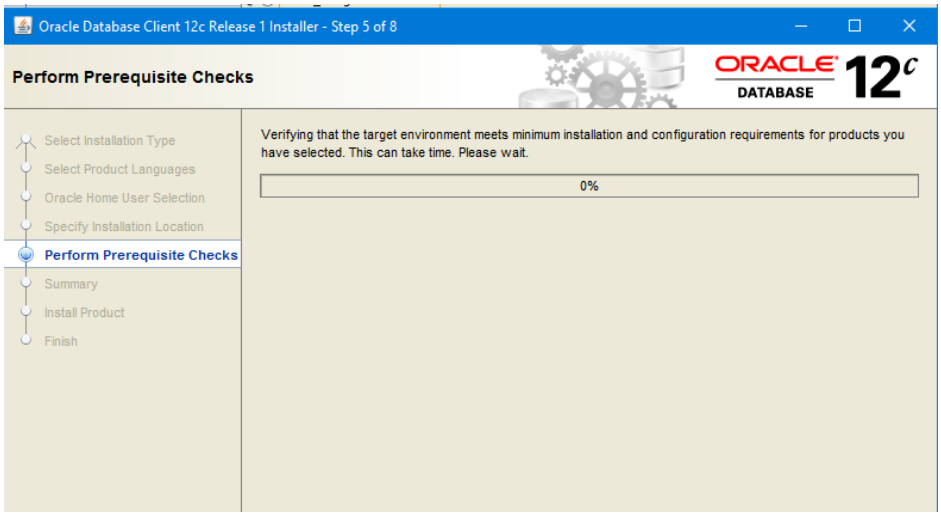
Here you must create a new user that does not have Admin rights to the server or the install directory. This must be a regular Windows user. You will have the option to choose an existing user or create a new one and when you're ready enter '**USER NAME**' and '**PASSWORD**' then click the '**Next**' button.



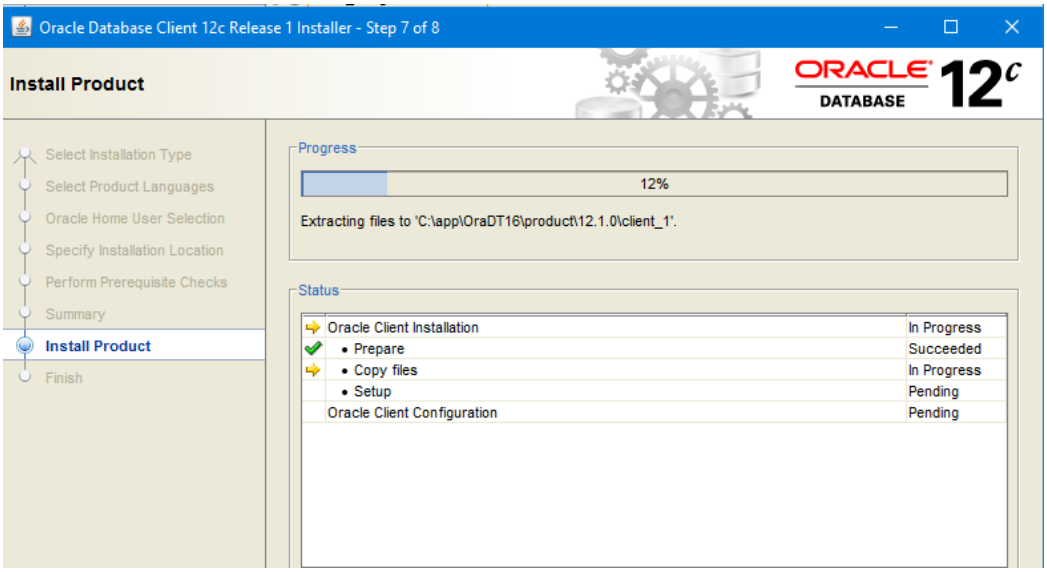
Here you can change the installation default locations.

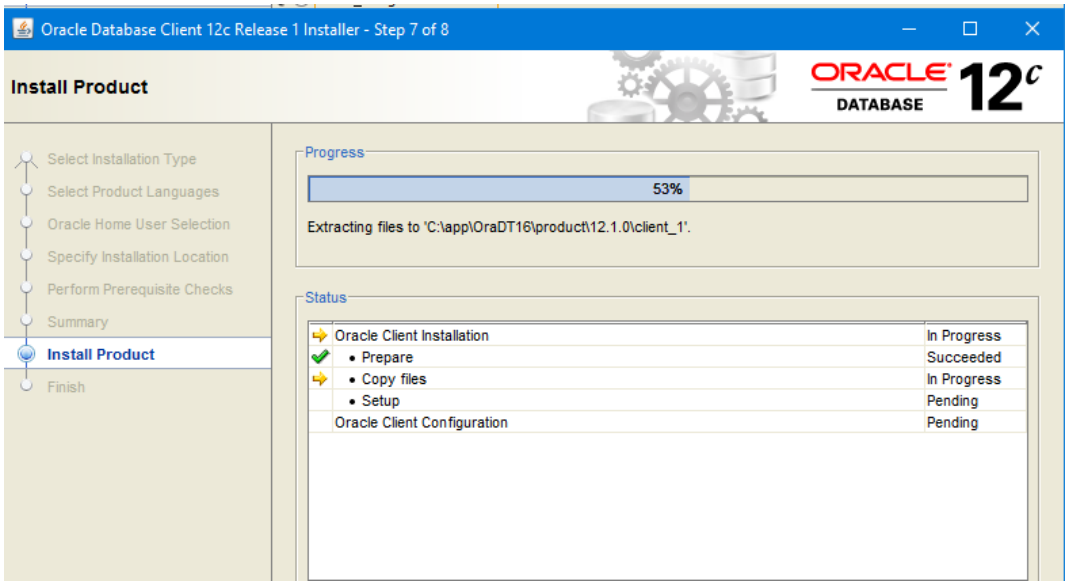


Click '**Next**' and you have now begun the install.

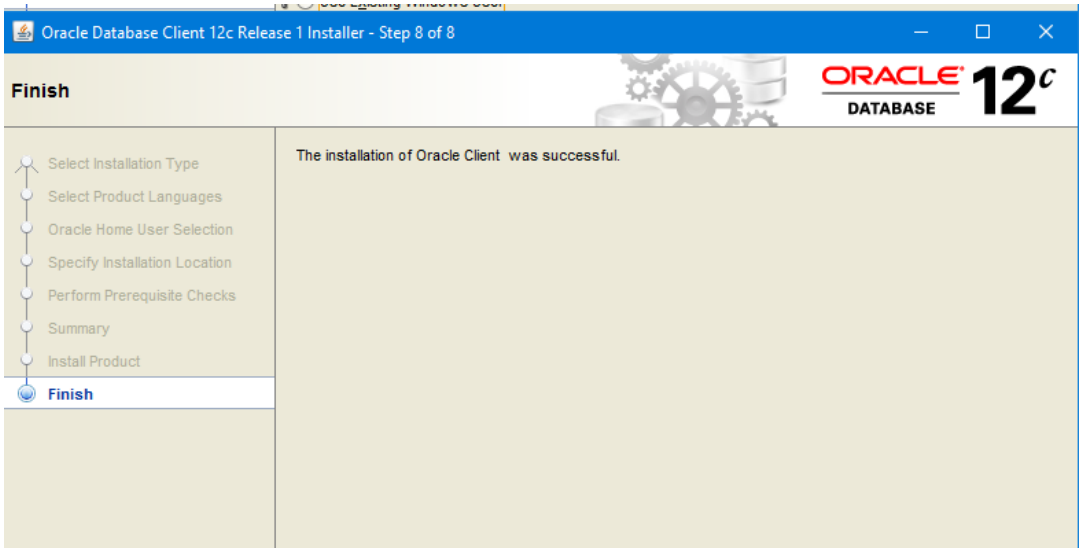


The install progress can be seen in the progress bar located at the top of the window. Towards the middle is the status panel which shows what part of the install has been **completed**, is **Pending**, **Succeeded**, or **In Progress**.





The install has now finished.

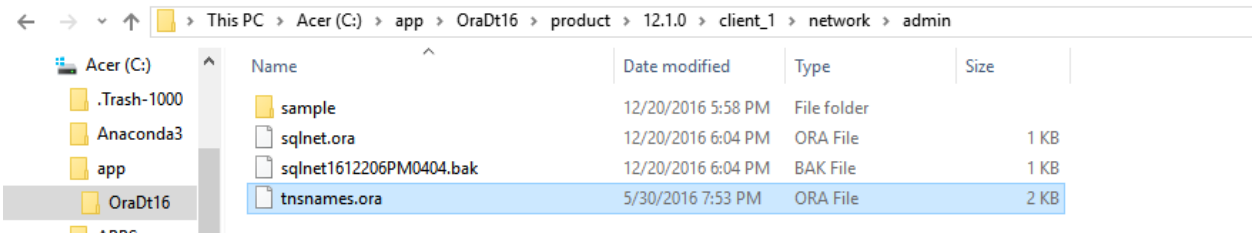


Oracle Client is now installed and you can now click the '**Exit**' button.

Oracle Post Installation and configuration

The Oracle 12c Client has now been installed, but we must now configure it to connect to an Oracle 12c database. To configure it you will need to follow these steps:

- The first step is to copy the **tnsnames.ora** file to the **NETWORK** directory in your client installation. This text file holds all your connection aliases to your various databases.



Untitled * | tnsnames.ora

```
# tnsnames.ora Network Configuration File: C:\app\OraDt16\product\12.1.0\dbhome_1\network\admin\tnsnames.ora
# Generated by Oracle configuration tools.

ORACLR_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
    (CONNECT_DATA =
      (SID = CLRExtProc)
      (PRESENTATION = RO)
    )
  )
```

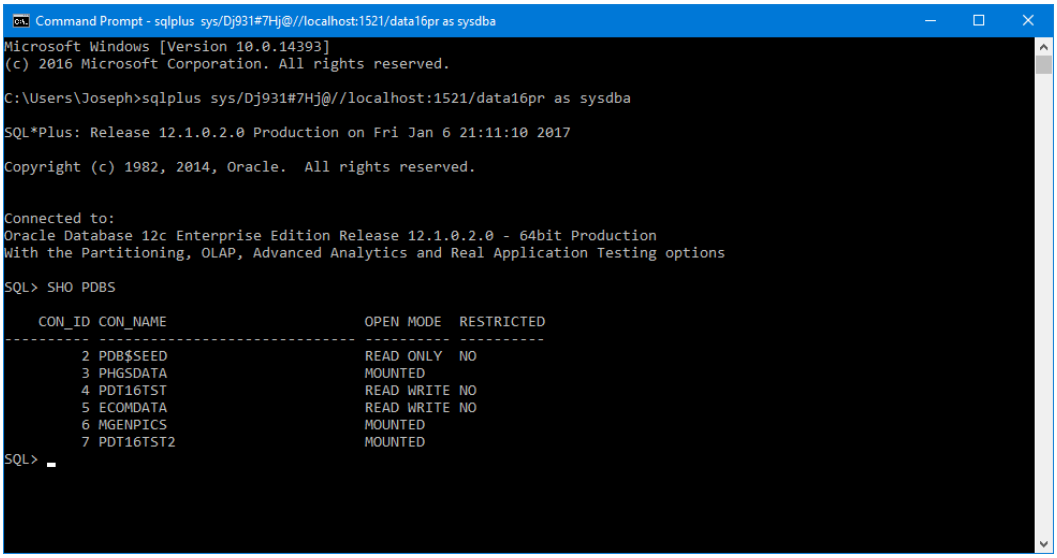
In Oracle 12c database we will be running the **SQL*Plus** client for several of our SQL queries/scripts, see below.

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Joseph>sqlplus sys/PASSWORD@//localhost:1521/data16pr as sysdba

SQL*Plus: Release 12.1.0.2.0 Production on Fri Jan 6 21:11:10 2017

Copyright (c) 1982, 2014, Oracle.  All rights reserved.
```

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> SHO PDBS

```
    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 PHGSDATA                       MOUNTED
         4 PDT16TST                       READ WRITE NO
         5 ECOMDATA                       READ WRITE NO
         6 MGENPICS                       MOUNTED
         7 PDT16TST2                      MOUNTED
SQL>
```



This ends the Software Installation prerequisites

Installation of ROracle

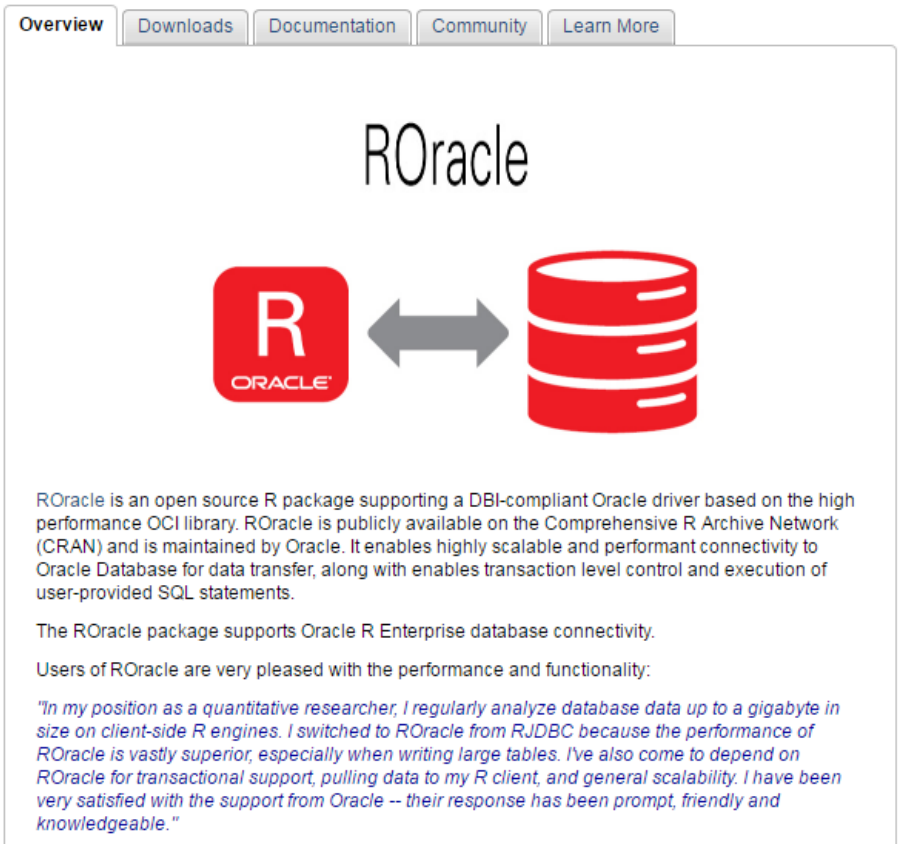
The main web page for **ROracle** on the **OTN** web site is the page below, besides downloads there is also other useful information, see the URL below.

In benchmark comparisons, **ROracle** performed up to 79 times faster than **RJDBC** and 2.5 times faster than **RODBC** for reading data across a range of 1000 to 1 million rows, and 10 to 1000 columns. **ROracle** shows scalability across **NUMBER**, **VARCHAR2**, **TIMESTAMP**, and **BINARY_DOUBLE** data types.

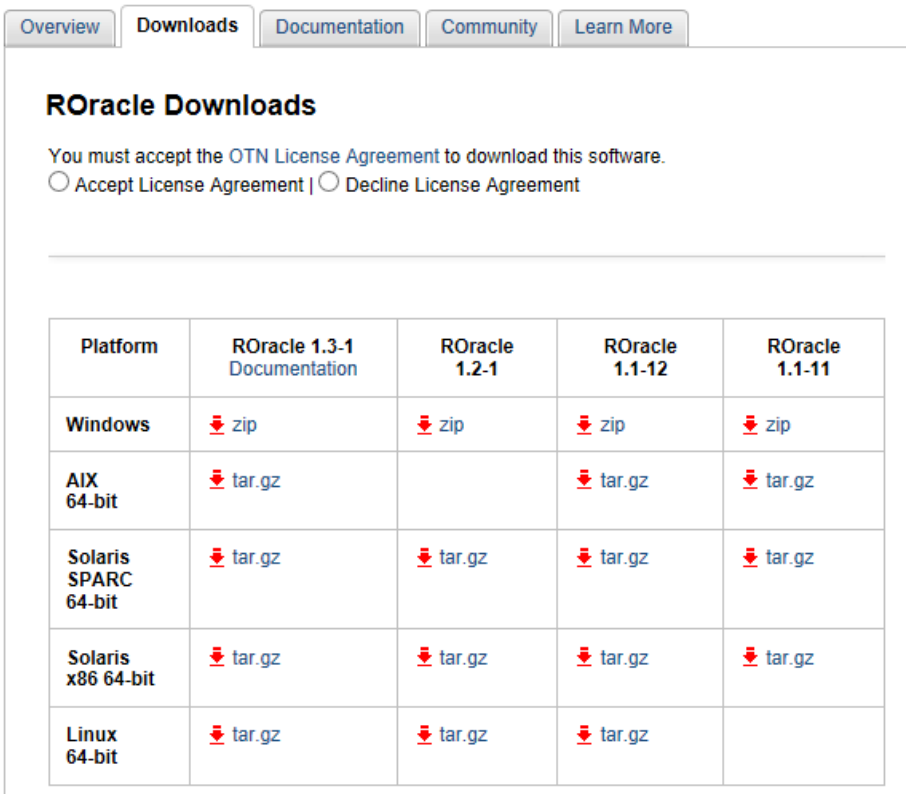
Similarly, for writing data to Oracle Database, **ROracle** was 61 times faster for 10 columns at 10 thousand rows than **RODBC**, and 630 times faster for the same data than **RJDBC**.

Here the **ROracle** page is divided into five tabs we want the downloads tab.

http://www.oracle.com/technetwork/database/database-technologies/r/roracle/downloads/index.html



Download the most current version of the **ROracle** package by clicking the download tab using the zip hyperlink for **ROarcle 1.3-1** see page below.



Overview | Downloads | Documentation | Community | Learn More

ROracle Downloads

You must accept the OTN License Agreement to download this software.
○ Accept License Agreement | ○ Decline License Agreement

Platform	ROracle 1.3-1 Documentation	ROracle 1.2-1	ROracle 1.1-12	ROracle 1.1-11
Windows	zip	zip	zip	zip
AIX 64-bit	tar.gz		tar.gz	tar.gz
Solaris SPARC 64-bit	tar.gz	tar.gz	tar.gz	tar.gz
Solaris x86 64-bit	tar.gz	tar.gz	tar.gz	tar.gz
Linux 64-bit	tar.gz	tar.gz	tar.gz	

Here we will choose the Windows version and we will get the most current version which is the file **ROracle_1.3-1.zip**. Here we will have to accept the license agreement and then click the zip hyper link below the **ROracle_1.3-1** heading and the download will proceed if it doesn't, just do it again.

The documentation tab is for downloading the documentation for this package, click the documentation tab, see the URL below to go to that page.

http://www.oracle.com/technetwork/database/database-technologies/r/roracle/documentation/index.html

There is only one file there and that is **ROracle.pdf** click the hyper link to begin downloading. Create a work directory on your PC call it **ROracle** move the files into it. In order to install this package bring up **RStudio** and we will run the command below.

The '**install.packages**' command/function will need three pieces of information the **path**, the **file name** and the other parameter will tell the function **not to use a repository** see below.

Directory Parameter
This is the directory where you downloaded the **ROracle** to and where the other files you downloaded are.

C:\DATATREE_NEW_HOME\R_ENTERPRISE\ROracle

Since one of the parameters is a directory path it will use escape characters, these will have to be suppressed. The function will deal with a single escape character by giving you an error, see below, except for the path which will be different.

```
> install.packages("C:\DATATREE_NEW_HOME\R_ENTERPRISE\ROracle\ROracle_1.3-1.zip", repos=NULL)
Error: '\D' is an unrecognized escape in character string starting ""C:\D"
```

So the escape character '\' will need to be doubled to '\\' so that the '**install.packages**' command/function will not give us an error. You will need to type a command like what you see below, but your **PATH** will be different.

install.packages("C:\\DATATREE_NEW_HOME\\R_ENTERPRISE\\ROracle\\ROracle_1.3-1.zip", repos=NULL)

We will run these commands in **RStudio**, the log below shows the results you will get.

```
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'license()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
> install.packages("C:\\DATATREE_NEW_HOME\\R_ENTERPRISE\\ROracle\\ROracle_1.3-1.zip",
repos=NULL)
Installing package into 'C:/Users/Joseph/Documents/R/win-library/3.3'
(as 'lib' is unspecified)
```

The **ROracle** package has been successfully unpacked and MD5 sums checked

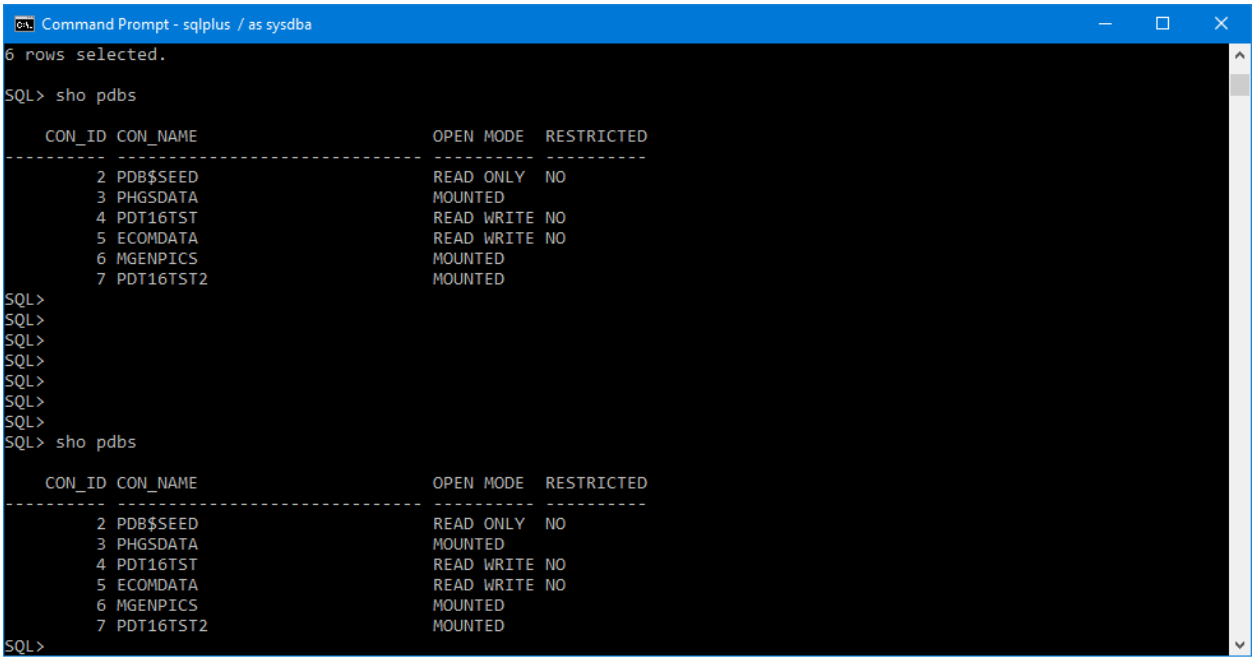
We have now installed the following software products:

- **R version 3.2.2**.
- **RStudio 1.0.44**.
- **Oracle 12c Client (12.1.0.2)** and configured it.
- **ROracle 1.3-**1 is installed as a package under the R language

You have now installed all the prerequisites and the **Roracle** package itself. Now we will need to test it, for this you will need an Oracle database to connect to.

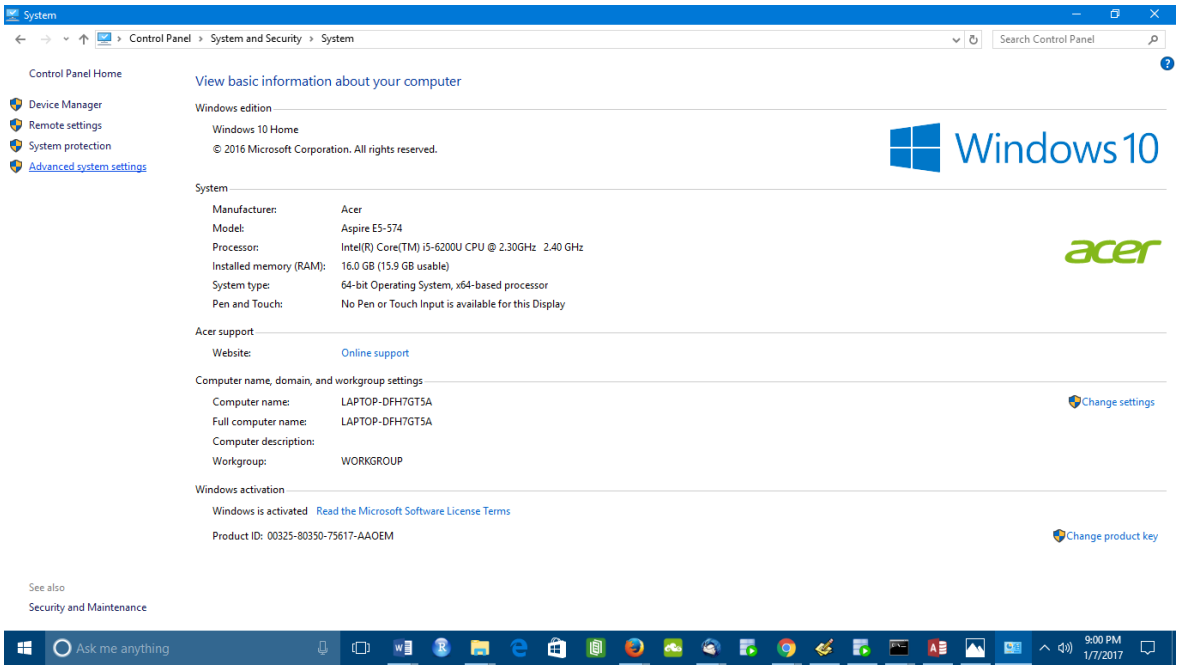
I will be using an **Oracle 12c (12.1.0.2)** database for this test. It has six pluggable databases on it with various datasets we will try to connect to **DATA16PR** (**CDB$ROOT**) and **PDT16TST** which has many data sets.

The image below is of a **SQL*Plus** windows prompt with the **show pdbs** command's output showing this is where we will execute many of our SQL scripts.

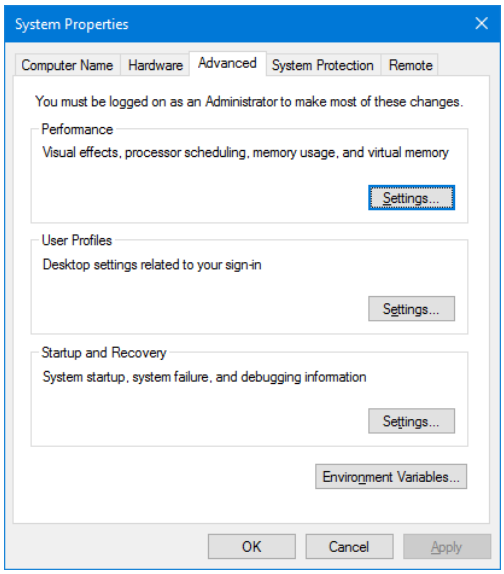


The **PATH** should be set to include the path to the Oracle Client libraries. Oracle Universal Installer should have set the **PATH** and **ORACLE_HOME** environment variables in the registry database.

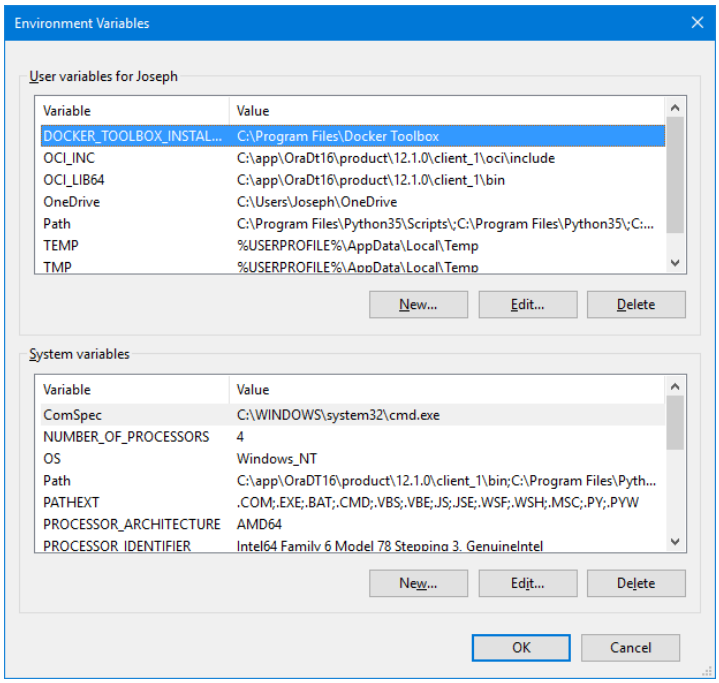
Search for '**Control Panel**' and you will see the screen listed below. Then select the '**Advance System Settings**' will get you to the Environmental Variables.

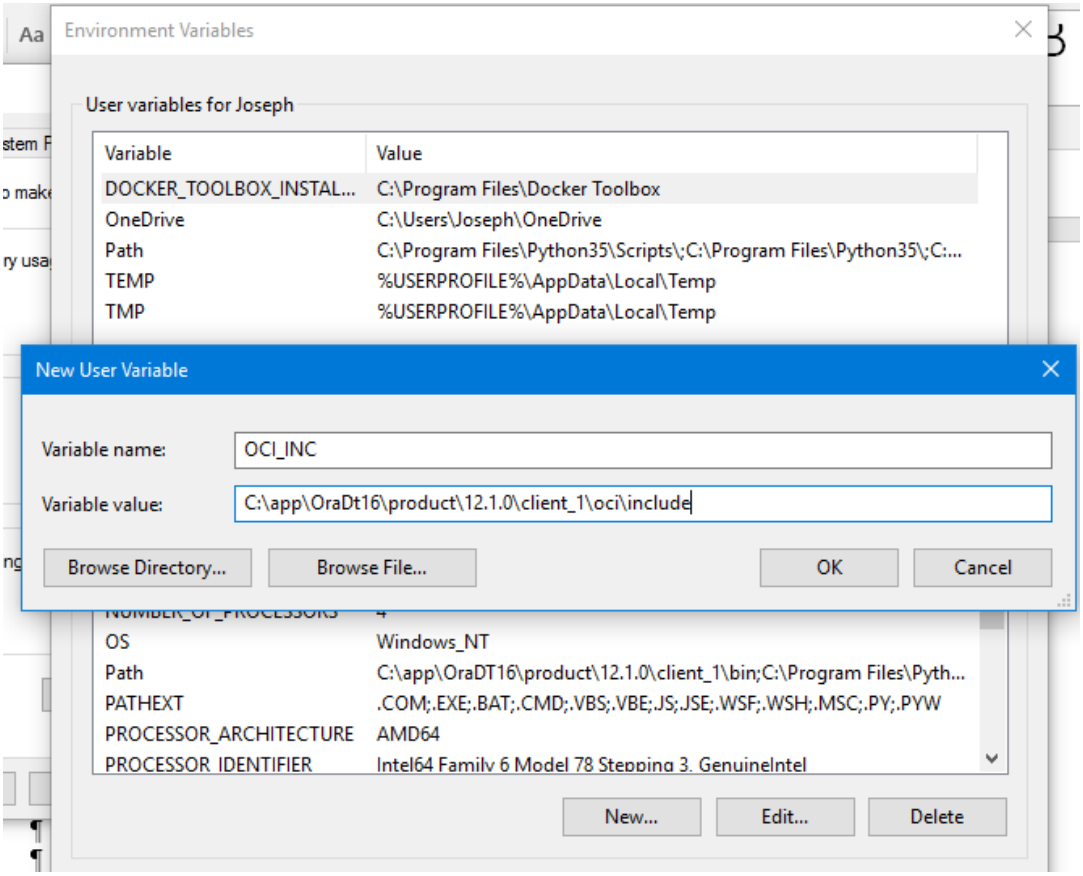


Select '**Environment Variables**' and here we must create two new ones to locate the Oracle OCI libraries under the Oracle 12c Client install directories.



Click the **New** button for '**User variables for Joseph**' Joseph is the current user's name.

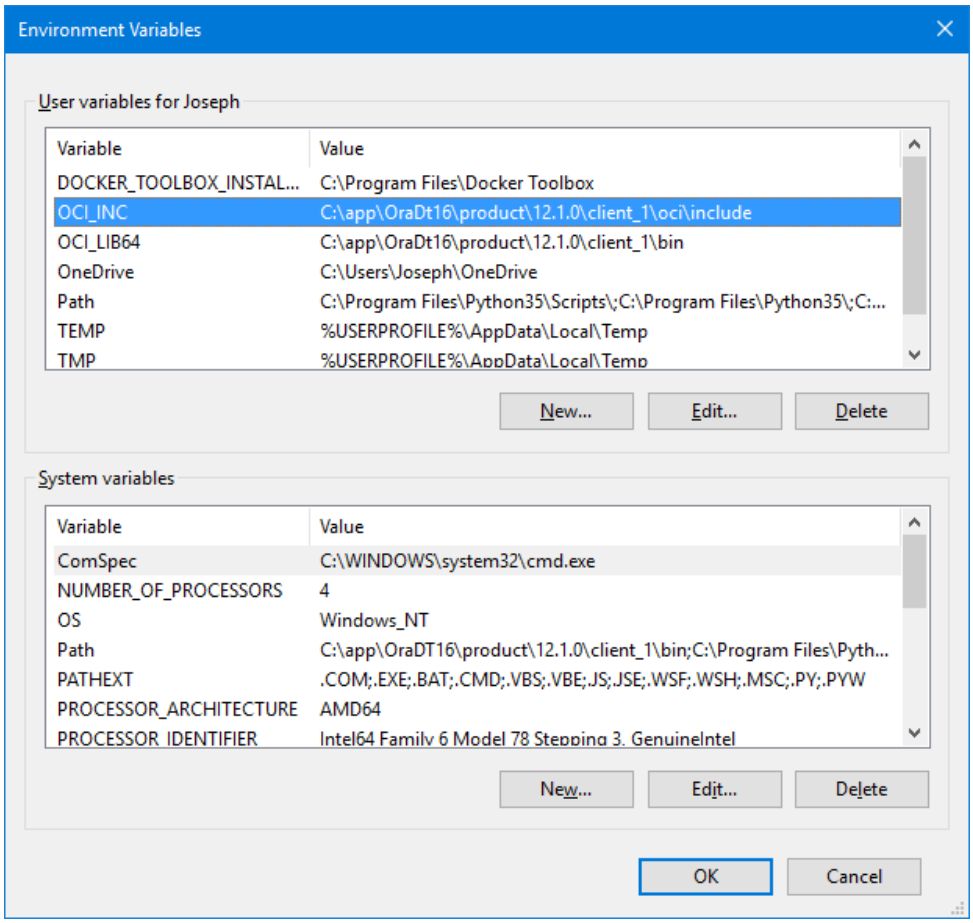

Environment Variables
User variables for Joseph
Variable
Value
DOCKER_TOOLBOX_INSTAL...
C:\Program Files\Docker Toolbox
OCI_INC
C:\app\OraDt16\product\12.1.0\client_1\oci\include
OCI_LIB64
C:\app\OraDt16\product\12.1.0\client_1\bin
OneDrive
C:\Users\Joseph\OneDrive
Path
C:\Program Files\Python35\Scripts\;C:\Program Files\Python35\;C:...
TEMP
%USERPROFILE%\AppData\Local\Temp
TMP
%USERPROFILE%\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
ComSpec
C:\WINDOWS\system32\cmd.exe
NUMBER_OF_PROCESSORS
4
OS
Windows_NT
Path
C:\app\OraDT16\product\12.1.0\client_1\bin;C:\Program Files\Pyth...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW
PROCESSOR_ARCHITECTURE
AMD64
PROCESSOR_IDENTIFIER
Intel64 Family 6 Model 78 Stepping 3, GenuineIntel
New...
Edit...
Delete
OK
Cancel


Environment Variables
User variables for Joseph
Variable
Value
DOCKER_TOOLBOX_INSTAL...
C:\Program Files\Docker Toolbox
OneDrive
C:\Users\Joseph\OneDrive
Path
C:\Program Files\Python35\Scripts\;C:\Program Files\Python35\;C:...
TEMP
%USERPROFILE%\AppData\Local\Temp
TMP
%USERPROFILE%\AppData\Local\Temp
New User Variable
Variable name:
OCI_INC
Variable value:
C:\app\OraDt16\product\12.1.0\client_1\oci\include
Browse Directory...
Browse File...
OK
Cancel
OS
Windows_NT
Path
C:\app\OraDT16\product\12.1.0\client_1\bin;C:\Program Files\Pyth...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW
PROCESSOR_ARCHITECTURE
AMD64
PROCESSOR_IDENTIFIER
Intel64 Family 6 Model 78 Stepping 3, GenuineIntel
New...
Edit...
Delete

Here we will create two variables they are **OCI_INC** and **OCI_LIB64**. When the '**New User Variable**' dialog comes up enter both the '**Variable Name**' and '**Variable Value**' then click the OK button. The two values are listed below.

set OCI_INC=C:\app\OraDt16\product\12.1.0\client_1\oci\include
set OCI_LIB64= C:\app\OraDt16\product\12.1.0\client_1\bin

The two variables are listed below, when finished click OK.

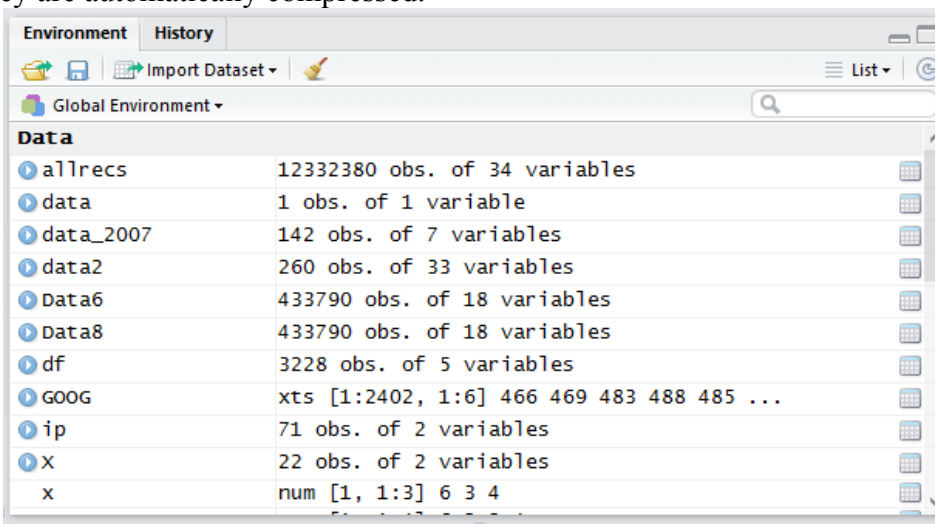


Load the library and use the package; you will have to change **DBNAME** to one of aliases listed in your **tnsnames.ora** file, as in the following example I am using the database container **CDB$ROOT** for my dbname of **data16pr**:

Workspaces and the .RData file

The **.RData** binary file, what is it and how does it work in the R environment? This file holds all the R objects that you create while you're working in your R environment. The workspace is your current R working environment and includes any user-defined objects (vectors, matrices, data frames, lists, functions) that you have created. At the end of a R session, the user can either save an image or not of the current workspace when prompted. This image is automatically reloaded the next time R is started

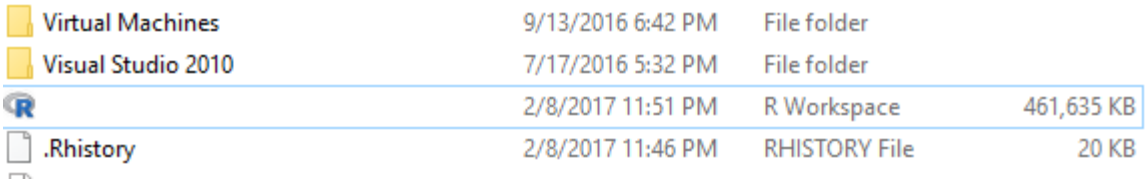
The **.RData** file is an unusually interesting aspect of the R processing environment, because it holds the workspace of the R environment and any objects created during your session are stored there, but they are automatically compressed.



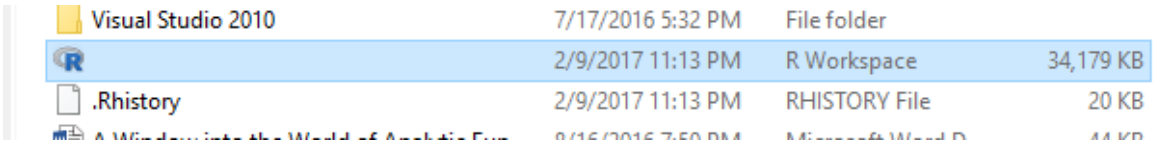
The image above lists the objects in the 'Environment' tab of RStudio, this lists all of the data frames, vectors, matixes and variables defined in my R environment. There is one copy of the **yellow_taxi_trip_june_bk** (allrecs) table that was copied from Oracle, two copies **RESTAURANT** (Data6, Data8) also from Oracle and an assortment of other objects, which makes up our workspace in R.

The chart below uses as an example the **yellow_taxi_trip_june_bk** table that was copied from Oracle to the .RData file this increased the file from 34998296 bytes to 437715551bytes in size.

	Size	Rows	Read Time	Filename	Compression
Unzip'd	2.3976 GB	12332379		yellow_taxi_trip_june_bk.csv	0
Zip'd	338.798 MB	12332379	16.51 sec	yellow_taxi_trip_june_bk.7z	85.869%
.RData	417.4381 MB	12332379		yellow_taxi_trip_june_bk	82.589%
Oracle	3649MB	12332379	30.222 sec	yellow_taxi_trip_june_bk	0



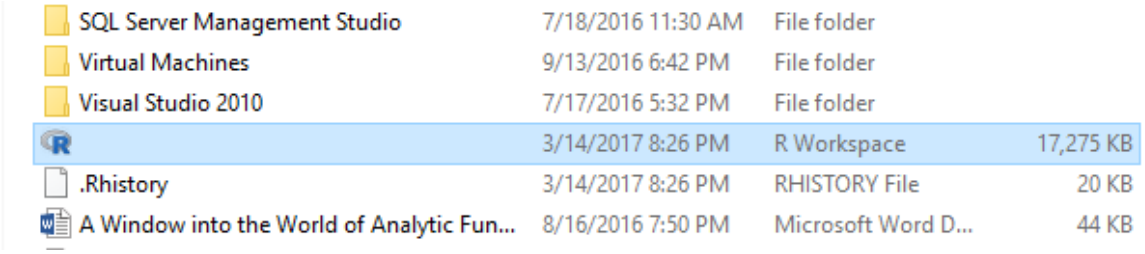
When we remove the **yellow_taxi_trip_june_bk** tables rows from R by equating the **allrecs** variable to 1 the rows are dropped and when you exit RStudio the .RData file returns to the 34MB size, see below.



The .RData file compresses any data that is written to it as you can see from the above chart. This compression is very similar to that done by 7zip or gzip. Next we will choose another object, this time the RESTAURANT table from our Oracle database and the results are the same it is compressed.

	Size	Rows	Read Time	Filename	Compression
Unzip'd	163,841,355 Bytes	433790		RESTAURANT_V7.csv	0
Zip'd	7,856,080 Bytes	433790	27 sec	RESTAURANT_V7.7z	95.205%
.RData	16903.59 KB	433790		RESTAURANT	89.44 %
Oracle	376MB	433790	Sec	RESTAURANT	

Let's look at what happens when we drop the records in the Data6 data.frame in the .RData file, see below.



Now you can see that the .RData file plays a very important role in R and that it can do even more. If you want to extract large data sets from Oracle or any other database then storing it in the .RData file for processing and modeling this is the way to go.

Mvbutils package

The **mvbutils** package has tools like **cd ()** function which allows you to set up and move through a **hierarchically-organized** set of R **workspaces**, each corresponding to a **directory**. While working at any level of the hierarchy, all higher levels are attached on the **search path**. You can easily switch between workspaces in the same session, you can move objects around in the hierarchy, and you can do several hierarchy-wide things such as searching, even on parts of the hierarchy that aren't currently attached.

R workspaces can become cluttered, so that it becomes very difficult to keep track of what's in them. If you work on several different projects, it can be difficult to work out where to put things or to remember where things are. If you just want to test out a bit of code without leaving permanent clutter, but while still being able to "see" your important objects, how do you do it? **Cd** helps with all such problems, by letting you organize all your projects into a **single tree structure**, regardless of where they are stored on disk. Each workspace is referred to (for historical reasons) as a "**task**".

There are two basic choices when you work with R, you keep everything you write in a text file which you access every time you start; or you store all the objects in the R workspace as a binary image in a **.RData** file. Some people prefer the text-based approach, but others including me prefer the binary image approach; my reasons are that binary images let me organize my work across tasks more systematically, and that repeated text-sourcing is much too slow when lengthy analyses or data extractions are involved.

The cd system is really geared to the binary image model and, before cd moves to a new task, either up or down the hierarchy, the current workspace is automatically saved to a binary image. Nevertheless, I don't think cd is incompatible with other ways of working, as long as the ".RData" file (actually the tasks object) is not destroyed from session to session.

At any rate, some people who work by sourcing large code files still seem to find cd useful; it's even possible to use the **.First.task** feature to auto-load a task's source files into a text editor when you cd to that task. With the **.RData** only approach, it is highly advisable to have some way of keeping separate text backups, at least of function code. The **fixr** editing system is geared up to this, and I presume other systems such as **ESS** are too.

To use the cd system, you will need to start R in the same workspace every time. This will become your ROOT or home task, from which all other tasks stem. There need not be much in this workspace except for an object called tasks (see below), though you can use it for shared functions that you don't want to organize into a package. From the ROOT task, your first action in a new R session will normally be to use cd to switch to a real task. The cd command is used both to switch between existing tasks, and to create new ones.

Performance Test of the Database connection to the Oracle server

Now we will be exploring the **DBI** and **ROracle** interface by issuing a series of **ROracle** commands which will submit **SQL** commands to the Oracle database. We will be benchmarking the connection between **RStudio** and the **Oracle 12c** database by submitting the same query from **RStudio** using **ROracle** commands and submitting the same **SQL** commands using **SQL Developer** and **SQL*Plus**. On the **SQL Developer** and **SQL*Plus** we will use the following commands:

1. **set echo on;**
2. **set timing on;**

On **RStudio** I will issue similar commands issuing the **proc.time()** function call before and after my code block to get elapsed time to execute the statements.

Our test Data Sets

We will be using several data sets **Restaurant inspection**, **Companies doing business in NYC**, and the **Yellow Taxi trip data** which are part of the **NYC Open Data initiative** see the URL's and descriptions below:

1) Yellow Taxi trip data 2015

We will be using **Yellow Taxi trip data** which is a GIS data set for the year 2015. This data set has data for 2015 for NYC from January through June of that year. The table has over 78 million records and is 11 GB in size before it was up loaded into Oracle. This took 2 hours and 24 minutes.

URL: https://data.cityofnewyork.us/Transportation/2015-Yellow-Taxi-Trip-Data/ba8s-jw6u

This dataset includes trip records from all trips completed by yellow taxis from in NYC from January to June in 2015. Records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. The data used in the attached datasets were collected and provided to the **NYC Taxi and Limousine Commission** (TLC) by technology providers authorized under the Taxicab Passenger Enhancement Program (TPEP). The trip data was not created by the TLC, and TLC makes no representations as to the accuracy of these data.

2) DOHMH New York City Restaurant Inspection Results Health

This dataset provides restaurant inspections, violations, grades and adjudication information

URL: https://data.cityofnewyork.us/browse?category=Health

The dataset contains every sustained or not yet adjudicated violation citation from every full or special program inspection conducted up to three years prior to the most recent inspection for restaurants and college cafeterias in an active status on the **RECORD DATE** (date of the data pull). When an inspection results in more than one violation, values for associated fields are repeated for each additional violation record. Establishments are uniquely identified by their **CAMIS** (record ID) number. Keep in mind that restaurants go in and out of business; only restaurants in an active status are included in the dataset.

Records are also included for each restaurant that has applied for a permit but has not yet been inspected and for inspections resulting in no violations. Establishments with inspection date of **1/1/1900** are new establishments that have not yet received a full inspection. Restaurants that received no violations are represented by a single row and coded as having no violations using the **ACTION** field.

Programming Logic

We will be benchmarking both R and SQL programs which will be performing the same queries on the data set tables which we have loaded into Oracle from the **NYC Open Data initiative**. I will be timing each programs and checking the run times for comparison. In addition, we will be testing joins and rollups of the data from other tables. Also we will use analytic functions and all queries will be run with and without indexes.

With no indexes:

1) Our first test is to select one months' worth of data from the **YELLOW_TAXI_TRIP** table and fetch it into the **RStudio** side as a **data.frame** and then create an Oracle table were you can write it back into a table using only R. We also do the same using SQL and **SQL*Plus** and benchmark both experiments.
2) Next we will perform several joins

This is for testing the database connection for tables with **no indexes**. For the first two tests, we are connecting to the container database so there is no change in the connect string from how you would connect to an Oracle 11g database, see connect string below.

```
library('ROracle')
drv <- dbDriver("Oracle")

#sample connect command
##  con <- dbConnect(drv, "USERNAME", "PASSWORD", dbname='DBNAME')

# real connect commad
    con <- dbConnect(drv, "system ", "PASSWORD", dbname='data16pr')
```

The dbConnect function call has spaces in both the ID and the password which gives you an error see below.

```
> con <- dbConnect(drv, " system ", " PASSWORD", dbname=' data16pr')
Error in .oci.Connect(.oci.drv(), username = username, password = password,  :
  ORA-01017: invalid username/password; logon denied
>
```

```
>
> library('ROracle')
> drv <- dbDriver("Oracle")
>
> con <- dbConnect(drv, "system ", " PASSWORD", dbname='data16pr')
>
```

To test the connection to the Oracle 12c database by reading a Oracle table:

Test 1: In our test we will use the **dbReadTable()** function, which can be used to read whatever is in the table **DUAL**.

```
dbReadTable(con, 'DUAL')
```

```
>
> library('ROracle')
> drv <- dbDriver("Oracle")
>
> con <- dbConnect(drv, "system", "PASSWORD", dbname='data16pr')
>
> dbReadTable(con, 'DUAL')
  DUMMY
1     X
>
>
```

Lets use the **dbReadTable()** function to read what is in the view **V$PDBS** which is a more interesting output.

dbReadTable(con, 'V$PDBS')

```
dbReadTable(con, 'V$PDBS')
  CON_ID       DBID    CON_UID
1      2  369848507  369848507
2      3 1769860667 1769860667
3      4 2647541362 2052914599
4      5 1132693728 1132693728
5      6 2975992775 2975992775
6      7 2938944833 2938944833
                                                          GUID      NAME
1 4f, 24, fe, dc, 9e, 82, 4f, 44, 95, 1d, 1e, a4, 9c, 42, ff, 4e  PDB$SEED
2 72, 19, 85, 7a, 52, ab, 4c, 13, ac, 78, ad, 59, bc, 58, f3, 6e  PHGSDATA
3 ca, d5, c4, e6, 39, a5, 49, 1f, 83, b0, 4b, a6, 67, 11, 40, 38  PDT16TST
4 7d, eb, c2, 1e, ce, 15, 45, 49, bb, 39, 31, 3d, 18, 0c, 96, 7f  ECOMDATA
5 0f, a5, f4, b5, e4, 24, 45, c9, 9c, 86, c1, f6, b6, 7a, a4, e2  MGENPICS
6 fe, 04, 38, b2, f7, f0, 49, 14, a2, 7a, cc, 10, 7d, 51, 2b, dd PDT16TST2
   OPEN_MODE RESTRICTED           OPEN_TIME CREATE_SCN TOTAL_SIZE BLOCK_SIZE
1  READ ONLY         NO 2016-12-25 21:24:49    2233966  796917760       8192
2    MOUNTED       <NA>                <NA>    3434346          0       8192
3 READ WRITE         NO 2016-12-27 21:43:26    3443508 8113487872       8192
4 READ WRITE         NO 2016-12-27 21:44:33    3460730 1390411776       8192
5    MOUNTED       <NA>                <NA>    3538655          0       8192
6    MOUNTED       <NA>                <NA>    5891656          0       8192
  RECOVERY_STATUS SNAPSHOT_PARENT_CON_ID
1         ENABLED                      0
2         ENABLED                      0
3         ENABLED                      0
4         ENABLED                      0
5         ENABLED                      0
6         ENABLED                      0
```

Connecting to a Pluggable database

Connecting to a pluggable database is different than connecting to an Oracle 12c container or to an Oracle 11g database for one the **dbname** connect string is different see below. For one the host name the port and the **pluggable database name** must be provided, in our case it's '**pdt16tst**'. The host is '**localhost'** since the database is on the same PC as the **RStudio** and port is the default port '**1521**'. We also have some other parameters as follows:

- **bulk_read** is the size of the read cash the default is 1000, the
- **bulk_write** is the size of the write cash it has a default value 1000,
- **stmt_cache** is set to 0 so no statements will be kept in memory.
- **externa_credentials** if your logging in from a remote database.
- **sysdba** is if you are logging in with the **SYSDBA** role.

library('ROracle')

drv <- dbDriver("Oracle")

This is the full connect for **PDB 'pdt16tst**' as you can see it's different

ptm <- proc.time()

con <- dbConnect(drv, "searstgi_admin", "5933tgi", dbname='**//localhost:1521/pdt16tst**', prefetch = FALSE, bulk_read = 1000L, bulk_write = 1000L, stmt_cache = 0L,external_credentials = FALSE, sysdba = FALSE)

proc.time() - ptm

To check how much time our tests use, we will be using the **proc.time()** function, this function determines how much real **CPU** time in seconds the currently running **R** process has already taken.

The **proc.time** function **returns five elements** for backwards compatibility, but its print method **prints** a named vector of **length 3**. The first two entries are the **total user** and **system CPU** times of the current R process and any child processes on which it has waited, and the third entry is the '**real**' elapsed time since the process was started.

Test 2: Agregate the record count by month for the **YELLOW_TAXI_TRIP** .

In **RStudio**

```
###   2/9/17
>
>
> ptm <- proc.time()
> rs <- dbSendQuery(con, "SELECT EXTRACT(MONTH FROM TPEP_PICKUP_DATETIME ) AS TMONTH,
OUNT(*) FROM SEARSTGI_ADMIN.YELLOW_TAXI_TRIP GROUP BY EXTRACT(MONTH FROM TPEP_PICKUP_D
TETIME )")
> proc.time() - ptm
   user  system elapsed
   0.01    0.02    0.04
>
>
>
>
> ptm <- proc.time()
>
> fetch(rs)
  TMONTH COUNT(*)
1      1 12741017
2      6 12332380
3      2 12442388
4      4 13063760
5      5 13158079
6      3 13342951
```

```
>
> proc.time() - ptm
   user  system elapsed
   0.02    0.00  198.38
>
>
```

2/8/17

```
>
> ptm <- proc.time()
>
> rs <- dbSendQuery(con, "SELECT EXTRACT(MONTH FROM TPEP_PICKUP_DATETIME ) AS TMONTH,
UNT(*) FROM SEARSTGI_ADMIN.YELLOW_TAXI_TRIP GROUP BY EXTRACT(MONTH FROM TPEP_PICKUP_DA
TIME )")
>
> proc.time() - ptm
   user  system elapsed
      0       0       0
>
> ptm <- proc.time()
>
> fetch(rs)
> proc.time() - ptm

  TMONTH COUNT(*)
1      1 12741017
2      6 12332380
3      2 12442388
4      4 13063760
5      5 13158079
6      3 13342951
> proc.time() - ptm
   user  system elapsed
   0.00    0.00  389.82
>
>
```

In SQL Developer 4.15 we will execute the same query, see below:

2/9/17

```
set echo on;
set timing on;

SELECT EXTRACT(MONTH FROM TPEP_PICKUP_DATETIME ) AS TMONTH,
COUNT(*) FROM SEARSTGI_ADMIN.YELLOW_TAXI_TRIP GROUP BY
EXTRACT(MONTH FROM TPEP_PICKUP_DATETIME );
```

All Rows Fetched: 6 in 220.094 seconds

1 12741017
6 12332380
2 12442388
4 13063760
5 13158079
3 13342951

2/8/17

set echo on;
set timing on;

SELECT EXTRACT(MONTH FROM TPEP_PICKUP_DATETIME) AS TMONTH,
COUNT(*) FROM SEARSTGI_ADMIN.YELLOW_TAXI_TRIP GROUP BY
EXTRACT(MONTH FROM TPEP_PICKUP_DATETIME);

All Rows Fetched: 6 in 262.229 seconds

1 12741017
6 12332380
2 12442388
4 13063760
5 13158079
3 13342951

-- ##
-- 2/6/2017 9:21pm
Test 3: This query counts how many records are in the **YELLOW_TAXI_JUNE** table.

In SQL Developer.

set echo on;
set timing on;

SELECT COUNT(*) FROM SEARSTGI_ADMIN.YELLOW_TAXI_JUNE;
All Rows Fetched: 1 in 56.187 seconds
24664760

In RStudio.

ptm <- proc.time()

rs <- dbSendQuery(con, "SELECT COUNT(*) FROM
SEARSTGI_ADMIN.YELLOW_TAXI_JUNE")

```
 ## We now fetch records from the resultSet into a data.frame.
proc.time() - ptm

ptm <- proc.time()

 data <- fetch(rs)    ## extract all rows

 dim(data)
proc.time() – ptm
```

```
> ptm <- proc.time()
>
> rs <- dbSendQuery(con, "SELECT COUNT(*) FROM SEARSTGI_ADMIN.YELLOW_TAXI_JUN
E")
>
> ## We now fetch records from the resultSet into a data.frame.

> proc.time() - ptm
   user  system elapsed
   0.00    0.00    0.01
>
> ptm <- proc.time()
>
> data <- fetch(rs)    ## extract all rows
>
> dim(data)
[1] 1 1
> proc.time() - ptm
>
> dim(data)
[1] 1 1
> proc.time() - ptm
   user  system elapsed
   0.00    0.00   53.48
> data
  COUNT(*)
1 24664760
```

Test 4: This test is in SQL Developer to read from the database and create a copy of a table using a **CTAS**.
For **RStudio** the code reads the same table in Oracle and writes it to the '**.RData**' file and in every case is faster or the same.
This result is very interesting since it means that the '**.RData**' is faster than the database on writes. This is a consistent result when no matter what query I run it's faster.

```
set echo on;
set timing on;
```

```
CREATE TABLE RESTAURANT_CP3
AS
SELECT * FROM SEARSTGI_ADMIN.RESTAURANT;
```

Task completed in 60.545 seconds

```
SQL> set echo on
SQL> set timing on
SQL> CREATE TABLE RESTAURANT_CP3
AS
SELECT * FROM SEARSTGI_ADMIN.RESTAURANT;

Table RESTAURANT_CP3 created.
```

Elapsed: 00:00:50.915

```
 ptm <- proc.time()
 rs <- dbSendQuery(con, "select * from searstgi_admin.restaurant")

 ## We now fetch records from the resultSet into a data.frame.

 Data8 <- fetch(rs) ## extract all rows

 dim(data)
 proc.time() - ptm
```

2/9/17

```
> Data8 <- fetch(rs) ## extract all rows
>
> dim(data)
[1] 1 1
> proc.time() - ptm
   user  system elapsed
   2.14    0.30    4.20
>
```

2/8/17

```
>
> ptm <- proc.time()
> rs <- dbSendQuery(con, "select * from searstgi_admin.restaurant")
>
> ## We now fetch records from the resultSet into a data.frame.
>
> Data6 <- fetch(rs) ## extract all rows
>
> dim(data)
[1] 1 1
> proc.time() - ptm
   user  system elapsed
```

```
   2.15    0.25    6.77
>
```

Test 5: When writing to the database there is no significant difference between **RStudio** and **SQL Developer** the results are consistent.

CREATE TABLE SEARSTGI_ADMIN.YELLOW_TAXI_JUNE
AS
SELECT * FROM SEARSTGI_ADMIN.YELLOW_TAXI_TRIP WHERE
EXTRACT(MONTH FROM TPEP_PICKUP_DATETIME) = 6;

Table SEARSTGI_ADMIN.YELLOW_TAXI_JUNE created.

815.164 seconds

This error occurred because two fields in the "**YELLOW_TAXI_JUNE**" are formatted as timestamps. Which creates this error when the data is written back to the database.

```
> ptm <- proc.time()
>
> ##dbWriteTable(conn, "ORACLE_DB_TABLE", r_data_table, overwrite = F, append
= T, row.names = F)
> dbWriteTable(con, "YELLOW_TAXI_JUNE", allrecs, overwrite = FALSE, append=TR
UE, row.names = F, schema="SEARSTGI_ADMIN")
Error in .oci.WriteTable(conn, name, value, row.names = row.names, overwrite
= overwrite,  :
  Error in .oci.ValidateZoneInEnv(FALSE) :
  environment variable 'ORA_SDTZ()' must be set to the same time zone region
as the the environment variable 'TZ()'
>
> proc.time() - ptm
   user  system elapsed
   0.01    0.07    0.14
>
>
```

This error is caused by having timestamp fields in the data. By changing the TZ and the ORS_SDTZ values to "GMT" this issue can be resolved, see below.

```
>
>
> ptm <- proc.time()
>
> Sys.setenv(TZ = "GMT")
> Sys.setenv(ORA_SDTZ = "GMT")
>
> proc.time() - ptm
   user  system elapsed
   0.00    0.00    0.03
>
> ptm <- proc.time()
```

```
>
> ##dbWriteTable(conn, "ORACLE_DB_TABLE", r_data_table, overwrite = F, append = T, row
ames = F)
> dbWriteTable(con, "YELLOW_TAXI_JUNE", allrecs, overwrite = FALSE, append=TRUE, row.n
es = F, schema="SEARSTGI_ADMIN")
[1] TRUE
>
> proc.time() - ptm
   user  system elapsed
  51.35    9.53  816.10
>
>
```

set echo on;

set timing on;

SELECT COUNT(*) FROM SEARSTGI_ADMIN.YELLOW_TAXI_JUNE;

All Rows Fetched: 1 in 86.187 seconds

24664760

Details

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

This implementation allows you to connect to multiple host servers and run multiple connections on each server simultaneously. When interruptible is set to **TRUE**, it allows for interrupting long-running queries on the server by executing the query in a thread. Main thread checks for **Ctrl-C** and issues **OCIBreak/OCIReset** to cancel the operation on the server. By default, interruptible is **FALSE**. When **unicode_as_utf8** is set to **FALSE**, **NCHAR**, **NVARCHAR** and **NCLOB** data is fetched using the character set specified by the **NLS_LANG** setting. By default, **unicode_as_utf8** is set to **TRUE**. When **ora.attributes** is set to **TRUE**, the result set from **dbGetQuery** and fetch contains DBMS-specific attributes like **ora.encoding**, **ora.type**, and **ora.maxlength** for the corresponding column.

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

dbDriver An object **OraDriver** or **ExtDriver** whose class extends **DBIDriver**. This object is used to create connections, using the function **dbConnect**, to one or more Oracle Database engines.

dbUnloadDriver Free all resources occupied by the driver object.

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

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