Session 3:
Oracle R Enterprise 1.5.1
Embedded R Execution – R Interface

Oracle R Technologies

Mark Hornick
Director, Advanced Analytics and Machine Learning
mark.hornick@oracle.com

October 2018

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Topics

- Introduction to Embedded R Execution: What and Why?
- Embedded R Scripts
  - Execution through the R interface
- Select Features
  - Working with connections and auto-connect
  - Generating image streams
  - ORE-defined graphics function examples
- Example of ORE Workflow for Model Building and Scoring
- Summary

Embedded R Execution

- Execute R code on the database server machine
- Have Oracle Database control and manage spawning of R engines
- Eliminate loading data to user's client R engine and result write-back to Oracle Database
- Execute user-defined R functions using data- and task-parallelism
- Invoke R from SQL and return results in Oracle tables
- Use open source CRAN packages at the database server
- Store and manage user-defined R functions in the database
- Schedule user-defined R functions for automatic execution

Motivation – why embedded R execution?

- Facilitate application use of R script results
  - Develop/test user-defined R functions interactively with R interface
  - Invoke user-defined R functions directly from SQL for production applications
  - User-defined R functions – *scripts* – stored in Oracle Database
- Improved performance and throughput
  - Oracle Database-enabled data- and task-parallelism
  - Memory and compute resources of database server, e.g., Exadata
  - More efficient read/write of data between Oracle Database and R Engine
  - Parallel simulations
- Image/plot generation at database server
- Rich XML for structured and image (PNG) data

Embedded Script Execution – R Interface

Execute R scripts at the database server

R Interface function	Purpose
ore.doEval()	Invoke stand-alone R script
ore.tableApply()	Invoke R script with ore.frame as input
ore.rowApply()	Invoke R script on one row at a time, or multiple rows in chunks from ore.frame
ore.groupApply()	Invoke R script on data partitioned by grouping column of an ore.frame
ore.indexApply()	Invoke R script N times
ore.scriptCreate()	Create an R script in the database repository
ore.scriptList()	List the R scripts in the repository
ore.scriptLoad()	Load an R script by name from the repository
ore.scriptDrop()	Drop an R script in the database repository
ore.grant()	Grant access to an R script to a user
ore.revoke()	Revoke access to an R script to a user

Embedded Script Execution – R Interface

ORE function	Signature
ore.doEval	ore.doEval(FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL)
ore.tableApply	ore.tableApply(X, FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL)
ore.rowApply	ore.rowApply(X, FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL, rows = 1, parallel = getOption("ore.parallel", NULL))
ore.groupApply	ore.groupApply(X, INDEX, FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL, parallel = getOption("ore.parallel", NULL))
ore.indexApply	ore.indexApply(times, FUN, ..., FUN.VALUE = NULL, FUN.NAME = NULL, parallel = getOption("ore.parallel", NULL))
ore.scriptCreate	ore.scriptCreate(name, FUN, overwrite=FALSE, type)
ore.scriptList	ore.scriptList(name, pattern, type)
ore.scriptLoad	ore.scriptLoad(name, owner = NULL, newname = NULL, envir = parent.frame())
ore.scriptDrop	ore.scriptDrop(name, type)
ore.grant	ore.grant(name, type='rqscript', user)
ore.revoke	ore.revoke(name, type='rqscript', user)

ORE function	Input data	FUN.VALUE	Arguments	Function	Special
ore.doEval()	• None • Generated within R function • Load via ore.pull • Transparency layer • ROracle data load • Flat file data load	NULL (returns ore.object) or data.frame or ore.frame used as a template for the return value (returns ore.frame)	… arguments to function can be NULL or of the form <argument> = <value> Optional control arguments	FUN.NAME= name of function stored in R script repository or FUN = function NOTE: For table/row/groupApply, first argument corresponds to input data as data.frame object. For indexApply, first argument corresponds to index number.	Not applicable
ore.tableApply()	X = ore.frame				Not applicable
ore.rowApply()					• ***rows*** >= 1, the maximum number of rows in each chunk • parallel=T/F or n
ore.groupApply()					• ***INDEX*** = list or ore.frame object referencing ore.factor objects/columns with same length as X • ***parallel***=T/F or *n*
ore.indexApply()	• None • Generated within R function • Load via ore.pull • Transparency layer • ROracle data load • Flat file data load				• ***times*** = number of times to execute the function • ***parallel***=T/F or n

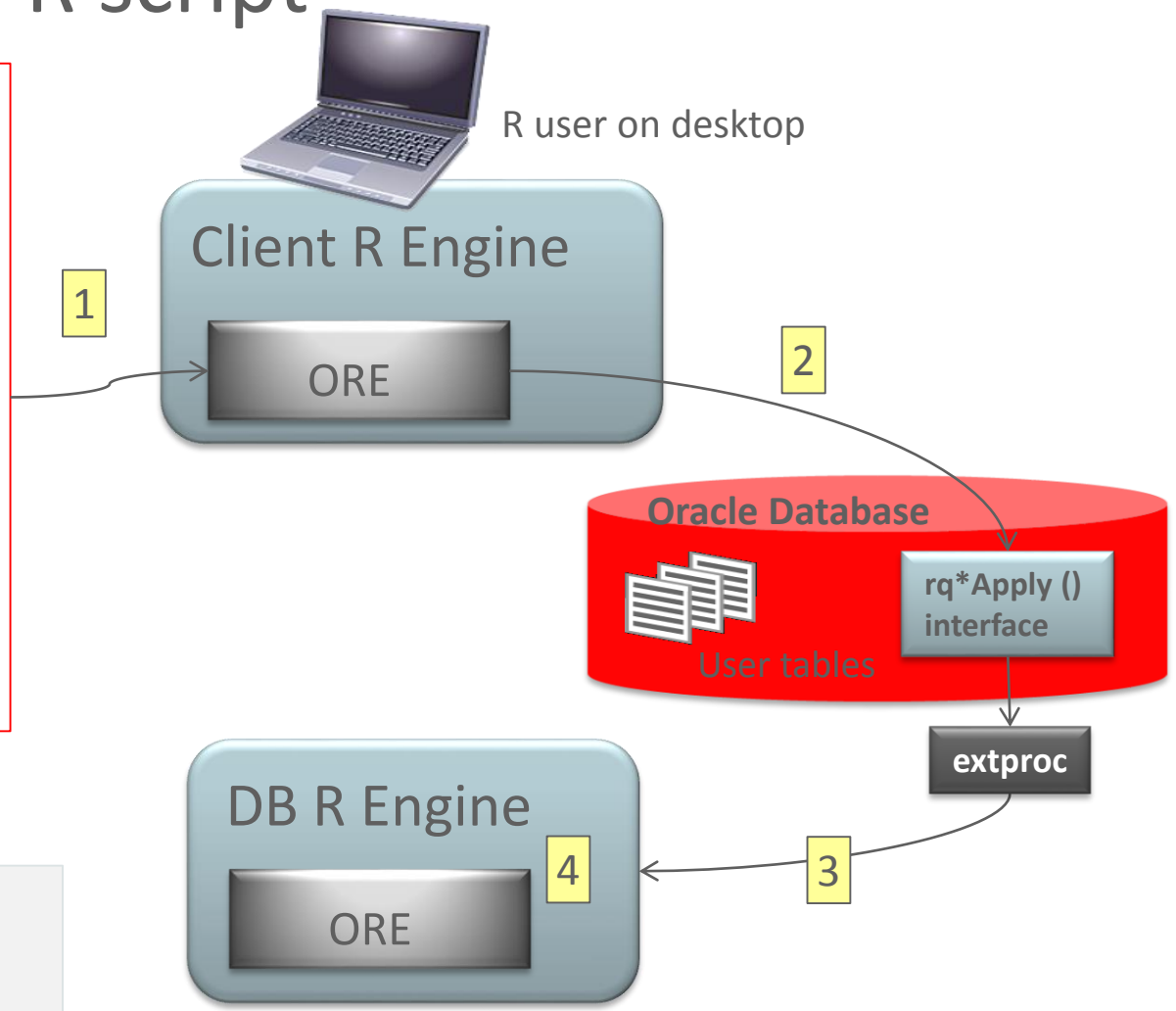
Using the R Script Repository

```
ore.scriptCreate("MyLM",
                 function(data, formula, ...) lm(formula, data, ...))
ore.scriptList(pattern="My")
ore.scriptLoad(name="MyLM", newname="MyNewLM")
ore.scriptDrop("MyLM")
ore.scriptCreate("MyNewLM",MyNewLM)


ore.grant(name="MyNewLM", type="rqscript")
ore.scriptList(type="grant")
ore.revoke(name="MyNewLM", type="rqscript")
ore.scriptList(type="grant")
ore.scriptDrop("MyNewLM")
```

```
> ore.scriptCreate("MyLM",
+                  function(data, formula, ...) lm(formula, data, ...))
> ore.scriptList(pattern="My")
  NAME                                                          SCRIPT
1 MyLM function (data, formula, ...) \nlm(formula, data, ...)
> ore.scriptLoad(name="MyLM", newname="MyNewLM")
> ore.scriptDrop("MyLM")
> ore.scriptCreate("MyNewLM",MyNewLM)
>
> ore.grant(name="MyNewLM", type="rqscript")
> ore.scriptList(type="grant")
     NAME GRANTEE
1 MyNewLM  PUBLIC
> ore.revoke(name="MyNewLM", type="rqscript")
> ore.scriptList(type="grant")
[1] NAME    GRANTEE
<0 rows> (or 0-length row.names)
> ore.scriptDrop("MyNewLM")
```

ore.doEval – invoking a simple R script

```
myFun <- function (num = 10, scale = 100) {
          ID <- seq(num)
          data.frame(ID = ID, RES = ID / scale)
          }
options(ore.warn.order=FALSE)
res <- ore.doEval(myFun)

class(res)
res

local_res <- ore.pull(res)
class(local_res)
local_res
```



Goal: scale the first n integers by value provided
Result: a serialized R data.frame as an ore.object, which remains at the database server until retrieved by the client

Results

```
> myFun <- function (num = 10, scale = 100) {
+     ID <- seq(num)
+     data.frame(ID = ID, RES = ID / scale)
+ }
>
> res <- ore.doEval(myFun)
>
> class(res)
[1] "ore.object"
attr(,"package")
[1] "OREembed"
> res
   ID  RES
1   1 0.01
2   2 0.02
3   3 0.03
4   4 0.04
5   5 0.05
6   6 0.06
7   7 0.07
8   8 0.08
9   9 0.09
10 10 0.10
```

```
> local_res <- ore.pull(res)
> class(local_res)
[1] "data.frame"
> local_res
   ID  RES
1   1 0.01
2   2 0.02
3   3 0.03
4   4 0.04
5   5 0.05
6   6 0.06
7   7 0.07
8   8 0.08
9   9 0.09
10 10 0.10
```

ore.doEval – specifying return value as ore.frame

```
myFun2 <- function (num = 10, scale = 100) {
              ID <- seq(num)
              data.frame(ID = ID, RES = ID / scale)
              }

res <- ore.doEval(myFun2,
             FUN.VALUE = data.frame(ID = 1,
                                    RES = 1))
class(res)
res
```

```
> myFun2 <- function (num = 10, scale = 100) {
+     ID <- seq(num)
+     data.frame(ID = ID, RES = ID / scale)
+ }
>
> res <- ore.doEval(myFun2,
+                   FUN.VALUE = data.frame(ID = 1, RES = 1))
> class(res)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
> res
   ID  RES
1   1 0.01
2   2 0.02
3   3 0.03
4   4 0.04
5   5 0.05
6   6 0.06
7   7 0.07
8   8 0.08
9   9 0.09
10 10 0.10
```

ore.doEval – specifying parameters

```
res <-
   ore.doEval(function (num = 10, scale = 100) {
                ID <- seq(num)
                data.frame(ID = ID,
                           RES = ID / scale)
                },
           num = 20, scale = 1000)
class(res)
res
```

```
> res <-
+     ore.doEval(function (num = 10, scale = 100) {
+         ID <- seq(num)
+         data.frame(ID = ID, RES = ID / scale)
+     },
+     num = 20, scale = 1000)
> class(res)
[1] "ore.object"
attr(,"package")
[1] "OREembed"
> res
   ID   RES
1   1 0.001
2   2 0.002
3   3 0.003
4   4 0.004
5   5 0.005
6   6 0.006
7   7 0.007
8   8 0.008
```

ore.doEval – using the R script repository

```
ore.scriptDrop("SimpleScript1")

ore.scriptCreate("SimpleScript1",
          function (num = 10, scale = 100) {
            ID <- seq(num)
            data.frame(ID = ID, RES = ID / scale)
            })


res <- ore.doEval(FUN.NAME="SimpleScript1",
                  num = 20, scale = 1000)
```

```
> ore.scriptDrop("SimpleScript1")
> ore.scriptCreate("SimpleScript1",
+                  function (num = 10, scale = 100) {
+                      ID <- seq(num)
+                      data.frame(ID = ID, RES = ID / scale)
+                  })
>
>
> res <- ore.doEval(FUN.NAME="SimpleScript1",
+                   num = 20, scale = 1000)
>
> res
   ID   RES
1   1 0.001
2   2 0.002
3   3 0.003
4   4 0.004
5   5 0.005
6   6 0.006
7   7 0.007
8   8 0.008
```

ore.doEval – with other data types

```
res <- ore.doEval(function (num = 10, scale = 100) {
            ID <- seq(num)
            data.frame(ID = ID, RES = ID / scale, CHAR="x")
            },
       FUN.VALUE = data.frame(ID = 1, RES = 1, CHAR="a"))
class(res)
res
```

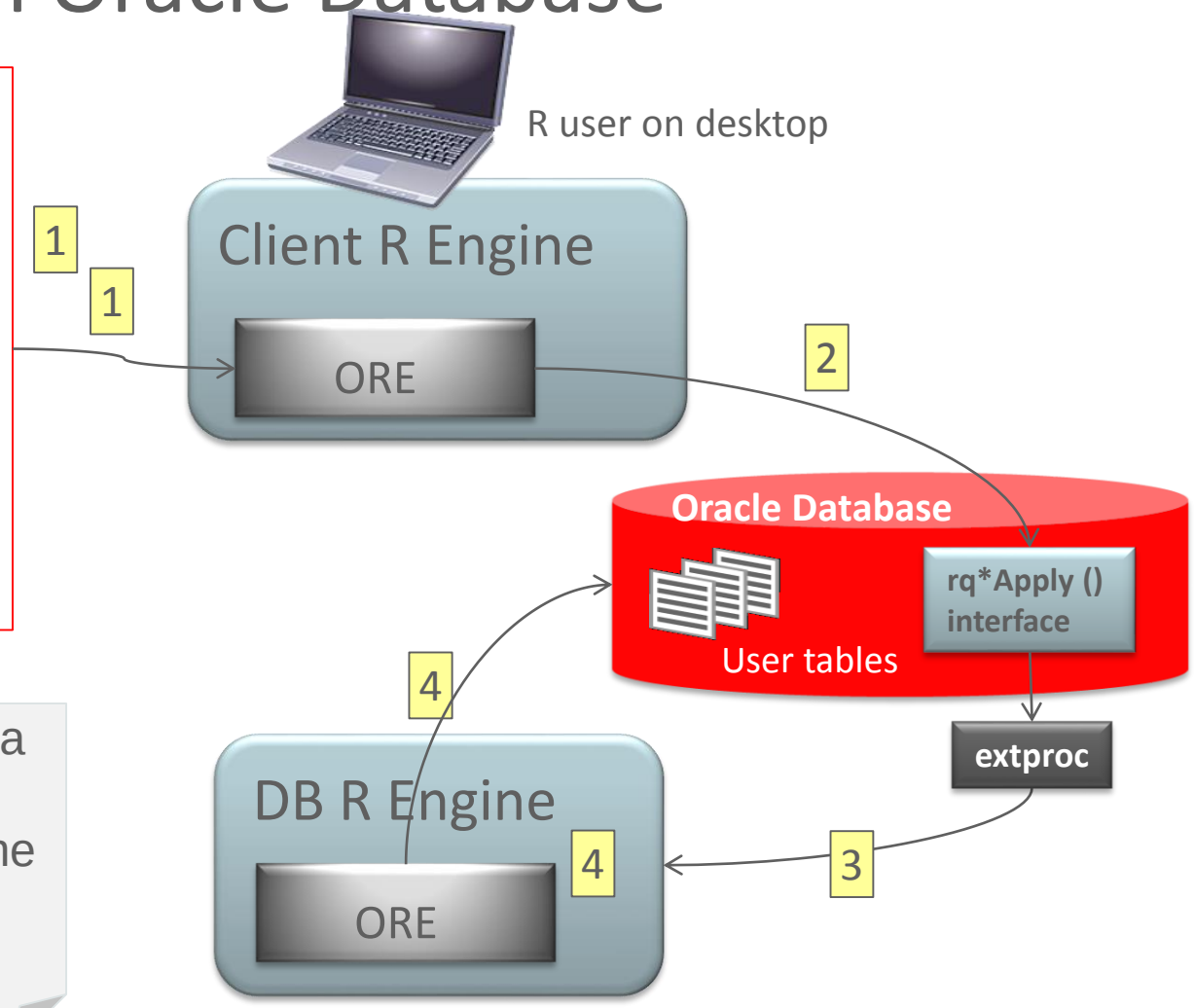
```
res <- ore.doEval(function (num = 10, scale = 100) {
       ID <- seq(num)
       d <- data.frame(ID = ID, RES = ID / scale, CHAR="x")
       d$BOOL <- d$RES < 0.04
       d
      },
      FUN.VALUE = data.frame(ID = 1, RES = 1,
                             CHAR="a", BOOL=TRUE))
class(res)
res
```

```
R> res <- ore.doEval(function (num = 10, scale = 100) {
+               ID <- seq(num)
+               data.frame(ID = ID, RES = ID / scale, CHAR="x")
+               },
+        FUN.VALUE = data.frame(ID = 1, RES = 1, CHAR="a"))
R> class(res)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> res
   ID  RES CHAR
1   1 0.01    x
2   2 0.02    x
3   3 0.03    x
4   4 0.04    x
5   5 0.05    x
6   6 0.06    x
7   7 0.07    x
8   8 0.08    x
9   9 0.09    x
10 10 0.10    x
```

```
R> res <- ore.doEval(function (num = 10, scale = 100) {
+        ID <- seq(num)
+        d <- data.frame(ID = ID, RES = ID / scale, CHAR="x")
+        d$BOOL <- d$RES < 0.04
+        d
+       },
+       FUN.VALUE = data.frame(ID = 1, RES = 1,
+                              CHAR="a",BOOL=TRUE))
R> class(res)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> res
   ID  RES CHAR  BOOL
1   1 0.01    x  TRUE
2   2 0.02    x  TRUE
3   3 0.03    x  TRUE
4   4 0.04    x FALSE
5   5 0.05    x FALSE
6   6 0.06    x FALSE
7   7 0.07    x FALSE
8   8 0.08    x FALSE
9   9 0.09    x FALSE
10 10 0.10    x FALSE
```

ore.doEval – pulling data from Oracle Database

```
mod <- ore.doEval(
   function() {
     ore.sync(table="ONTIME_S")
     dat <- ore.pull(ore.get("ONTIME_S"))
     lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
   },
   ore.connect = TRUE)
class(mod)
mod_local <- ore.pull(mod)
class(mod_local)
summary(mod_local)
```



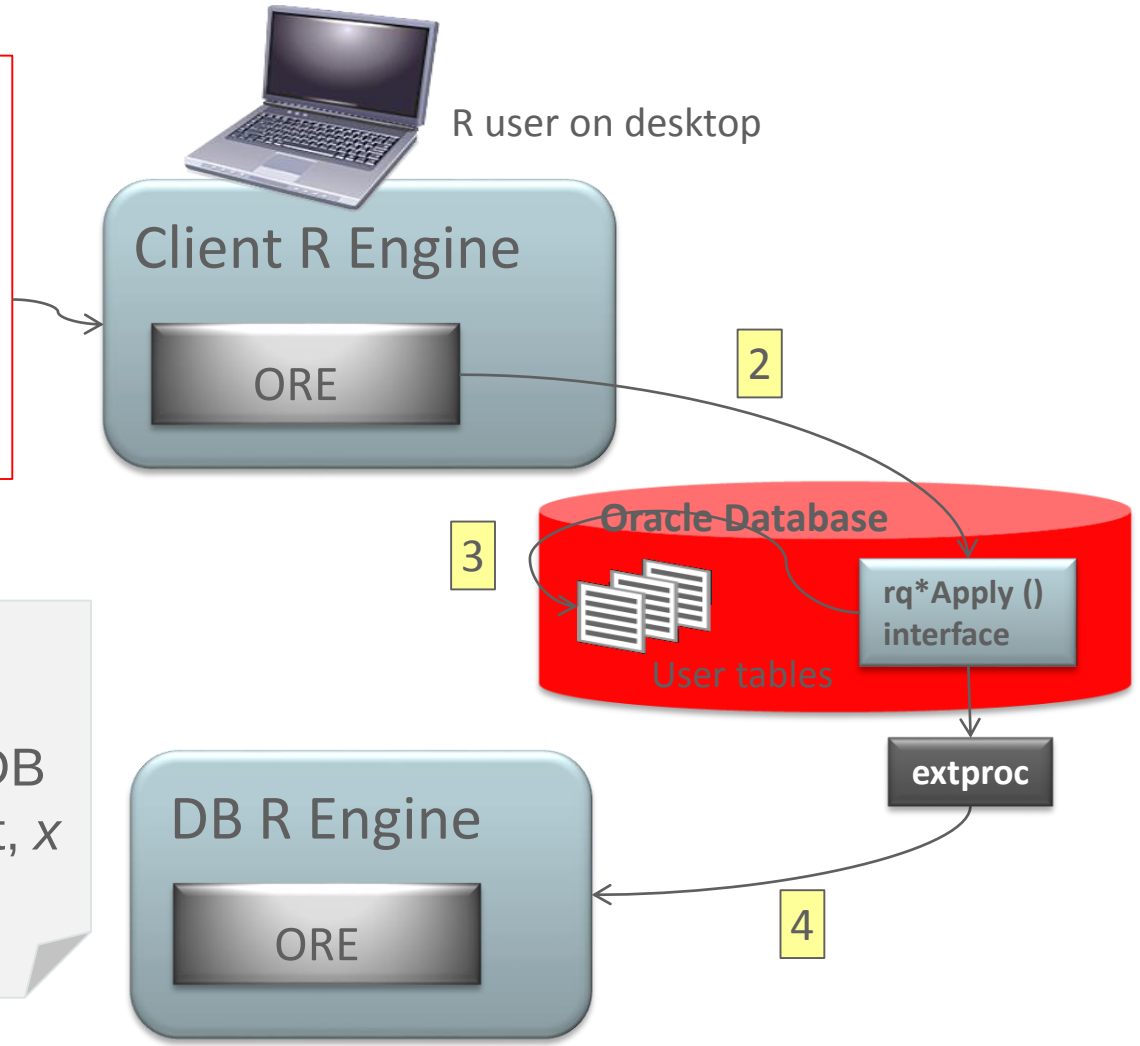
- Goal: Build a single regression model retrieving data using Transparency Layer
- Data explicitly loaded into R memory at DB R Engine using ore.pull()
- Result “mod” returned as an ore.object

Results

```
R> mod <- ore.doEval(
+    function() {
+      ore.sync(table="ONTIME_S")
+      dat <- ore.pull(ore.get("ONTIME_S"))
+      lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
+    },
+    ore.connect = TRUE);
R> class(mod)
[1] "ore.object"
attr(,"package")
[1] "OREembed"
R> mod_local <- ore.pull(mod)
R> class(mod_local)
[1] "lm"
R> summary(mod_local)

Call:
lm(formula = ARRDELAY ~ DISTANCE + DEPDELAY, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-1462.45    -6.97    -1.36     5.07   925.08

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.254e-01  5.197e-02   4.336 1.45e-05 ***
DISTANCE    -1.218e-03  5.803e-05 -20.979  < 2e-16 ***
DEPDELAY     9.625e-01  1.151e-03 836.289  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14.73 on 215144 degrees of freedom
  (4785 observations deleted due to missingness)
Multiple R-squared:  0.7647,    Adjusted R-squared:  0.7647
F-statistic: 3.497e+05 on 2 and 215144 DF,  p-value: < 2.2e-16
```

ore.tableApply – with parameter passing

```
modCoef <- ore.tableApply(
  ONTIME_S[,c("ARRDELAY","DISTANCE","DEPDELAY")],
  function(dat, family) {
    mod <- glm(ARRDELAY ~ DISTANCE + DEPDELAY,
               data=dat, family=family)
    coef(mod)
    }, family=gaussian());
modCoef
```



- Goal: Build model on data from input cursor with parameter family = gaussian()
- Data set loaded into R memory as data.frame at DB R Engine and passed to function as first argument, *x*
- Result coefficient(mod) returned as R object

Results

```
R> modCoef <- ore.tableApply(
+   ONTIME_S[,c("ARRDELAY","DISTANCE","DEPDELAY")],
+   function(dat, family) {
+     mod <- glm(ARRDELAY ~ DISTANCE + DEPDELAY,
+                data=dat, family=family)
+     coef(mod)
+     }, family=gaussian());
R> modCoef
 (Intercept)     DISTANCE     DEPDELAY
 0.225378249 -0.001217511  0.962528054
```

ore.tableApply – using CRAN package

```
library(e1071)
mod <- ore.tableApply(
  ore.push(iris),
  function(dat) {
    library(e1071)
    dat$Species <- as.factor(dat$Species)
    naiveBayes(Species ~ ., dat)
 })
class(mod)
mod
```

- Goal: Build model on data from input cursor
- Package e1071 loaded at DB R Engine
- Data set pushed to database and then loaded into R memory at DB R Engine and passed to function
- Result “mod” returned as serialized object

```
R> library(e1071)
R> mod <- ore.tableApply(
+   ore.push(iris),
+   function(dat) {
+     library(e1071)
+     dat$Species <- factor(dat$Species)
+     naiveBayes(Species ~ ., dat)
+ })
R> class(mod)
[1] "ore.object"
attr(,"package")
[1] "OREbase"
R> mod

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333

Conditional probabilities:
            Sepal.Length
Y             [,1]      [,2]
  setosa     5.006 0.3524897
  versicolor 5.936 0.5161711
  virginica  6.588 0.6358796

            Sepal.Width
Y             [,1]      [,2]
  setosa     3.428 0.3790644
  versicolor 2.770 0.3137983
```

ore.tableApply – batch scoring returning ore.frame

```
IRIS <- ore.push(iris)
IRIS_PRED <- IRIS
IRIS_PRED$PRED <- "A"
res <- ore.tableApply(
   IRIS,
   function(dat, mod) {
     library(e1071)
     dat$PRED <- predict(mod, newdata = dat)
     dat
   },
   mod = ore.pull(mod),
   FUN.VALUE = IRIS_PRED)
class(res)
head(res)
```

```
R> IRIS <- ore.push(iris)
R> IRIS_PRED <- IRIS
R> IRIS_PRED$PRED <- "A"
R> res <- ore.tableApply(
+    IRIS,    function(dat, mod) {
+      library(e1071)
+      dat$PRED <- predict(mod, newdata = dat)
+      dat
+    },
+    mod = ore.pull(mod),
+    FUN.VALUE = IRIS_PRED)
R> class(res)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> head(res)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species   PRED
1          5.1         3.5          1.4         0.2  setosa setosa
2          4.9         3.0          1.4         0.2  setosa setosa
3          4.7         3.2          1.3         0.2  setosa setosa
4          4.6         3.1          1.5         0.2  setosa setosa
5          5.0         3.6          1.4         0.2  setosa setosa
6          5.4         3.9          1.7         0.4  setosa setosa
Warning messages:
1: ORE object has no unique key - using random order
2: ORE object has no unique key - using random order
```

- Goal: Score data using model with data from ore.frame
- Return value specified using IRIS_PRED as *example* representation
- Result returned as ore.frame

ore.rowApply – data parallel scoring

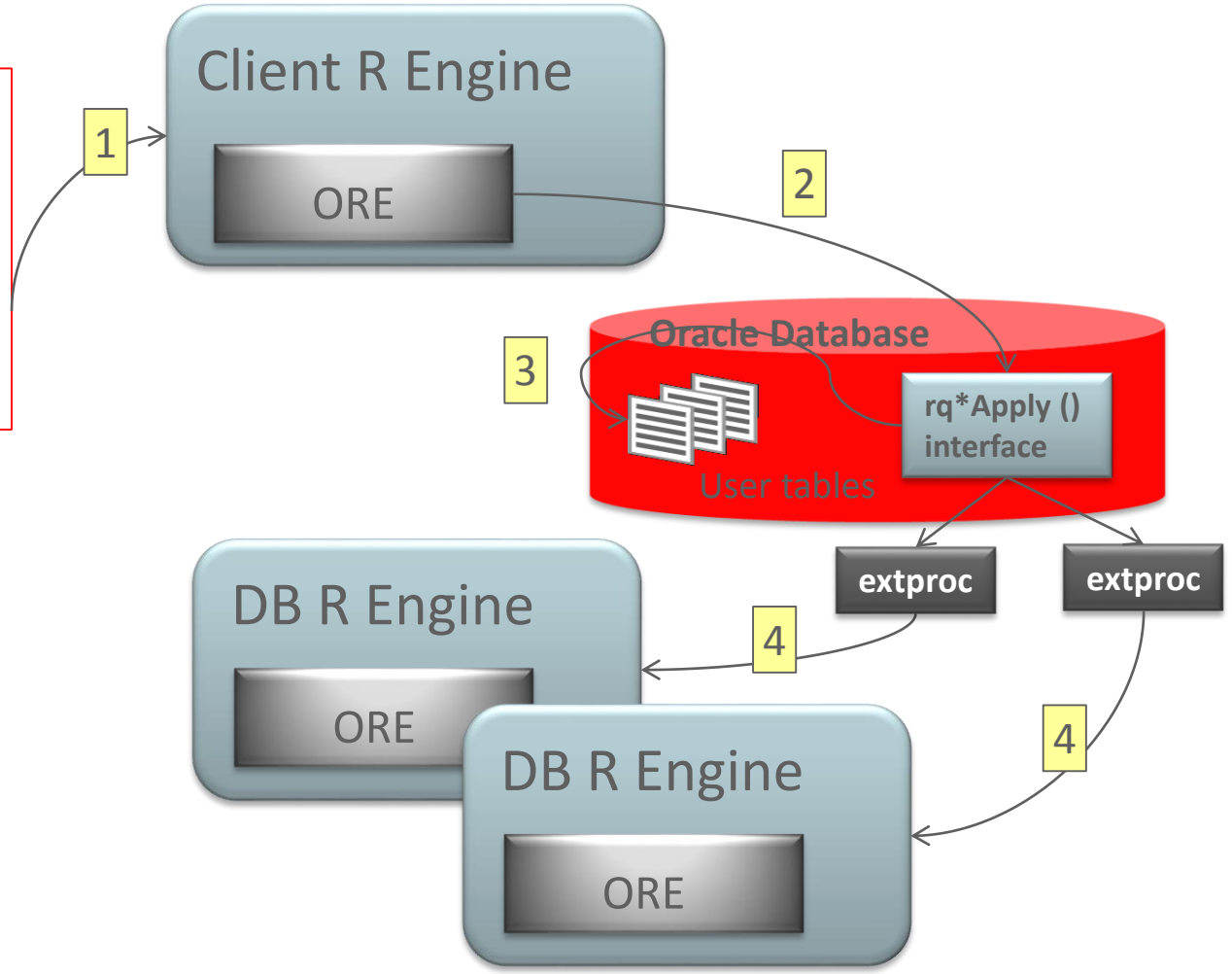
```
scoreNBmodel <- function(dat, mod) {
     library(e1071)
     dat$PRED <- predict(mod, newdata = dat)
     dat
   }

IRIS <- ore.push(iris)
IRIS_PRED <- IRIS
IRIS_PRED$PRED <- "A"

res <- ore.rowApply(
   IRIS ,
   scoreNBmodel,
   mod = ore.pull(mod),
   FUN.VALUE = IRIS_PRED,
   rows=10)

class(res)
table(res$Species, res$PRED)
```

```
R> IRIS <- ore.push(iris)
R> IRIS_PRED$PRED <- "A"
R> res <- ore.rowApply(
+    IRIS ,
+    function(dat, mod) {
+      library(e1071)
+      dat$Species <- as.factor(dat$Species)
+      dat$PRED <- predict(mod, newdata = dat)
+      dat
+    },
+    mod = ore.pull(mod),
+    FUN.VALUE = IRIS_PRED,
+    rows=10)
R> class(res)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> table(res$Species, res$PRED)

            setosa versicolor virginica
  setosa        50          0         0
  versicolor     0         47         3
  virginica      0          3        47
```

- Goal: Score data in batch (rows=10) using data from input ore.frame
- Data loaded into R memory at DB R Engine and passed to function
- Return value specified using IRIS_PRED as *example* representation
- Result returned as ore.frame

ore.groupApply – partitioned data flow

```
modList <- ore.groupApply(
    X=ONTIME_S,
    INDEX=ONTIME_S$DEST,
    function(dat) {
        lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
    })
summary(modList$BOS) ## return model for BOS
```



ore.groupApply – returning a single data.frame

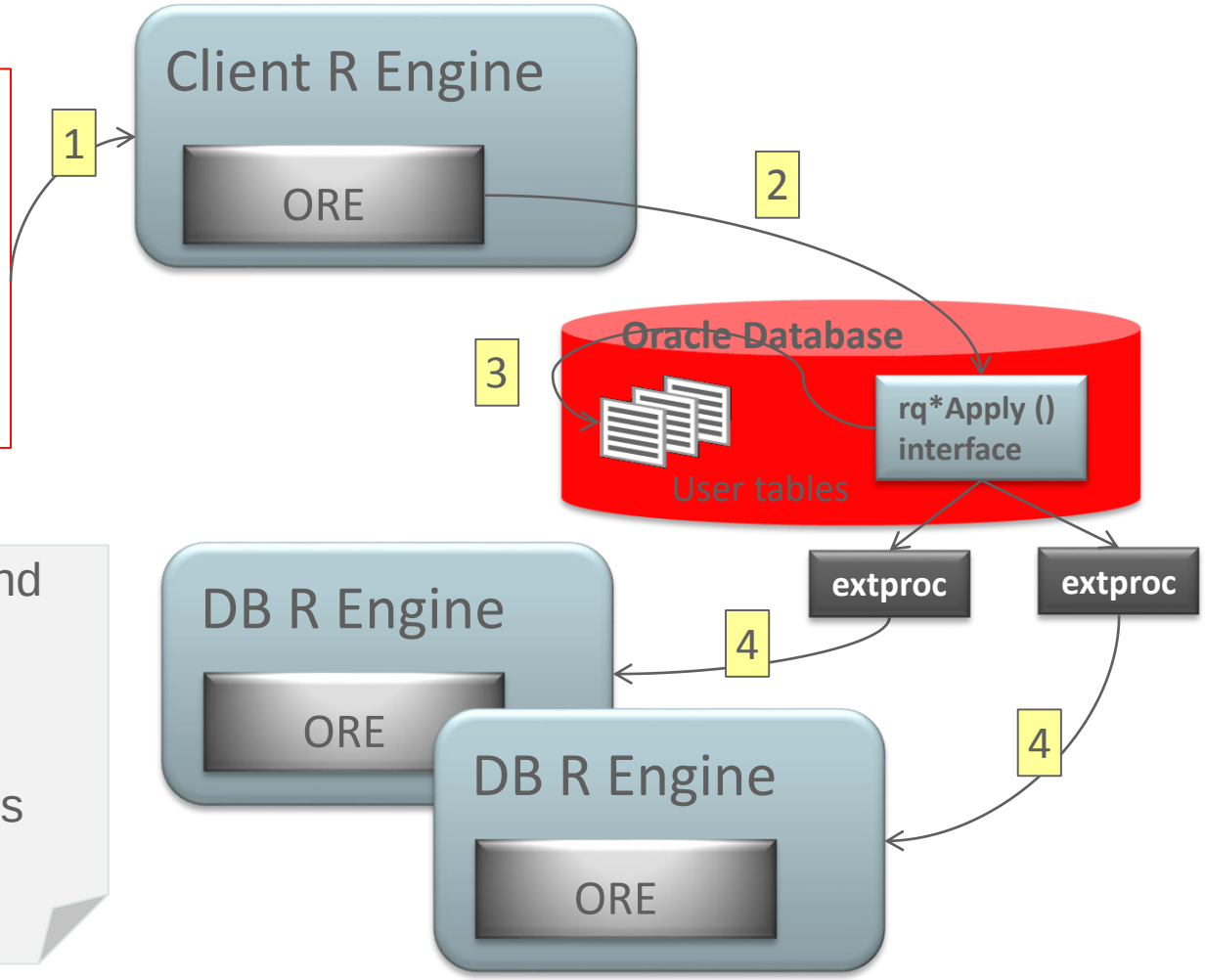
```
scoreReturningDF <- function(dat) {
                    species <- as.character(dat$Species)
                    mod <- lm(Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width, dat)
                    prd <- predict(mod, newdata=dat)
                    prd[as.integer(rownames(prd))] <- prd
                    data.frame(Species = species, Sepal.Length = dat$Sepal.Length,
                                PRED= prd, stringsAsFactors = FALSE)
                  }

IRIS <- ore.push(iris)
test  <- ore.groupApply(IRIS, IRIS$Species,
                scoreReturningDF,
                FUN.VALUE = data.frame(Species = character(), Sepal.Length = numeric(0),
                                       PRED = numeric(),
                                       stringsAsFactors = FALSE),
                parallel = TRUE)
# save results in database table TEST
ore.drop("TEST")
ore.create(test, "TEST")
```

‘parallel’ argument

- Preferred degree of parallelism to use in an embedded R job
- Supported by…
  - ore.groupApply
  - ore.rowApply
  - ore.indexApply
- Values
  - positive integer >= 2 for a specific degree of parallelism
  - 'FALSE' or 1 for no parallelism
  - 'TRUE' takes on the 'data’ argument's default parallelism
  - 'NULL' for the database default for the operation

ore.groupApply – multi-column INDEX

```
modList <- ore.groupApply(
   X=ONTIME_S,
   INDEX=ONTIME_S[,c("DEST","UNIQUECARRIER")],
   function(dat) {
      mod <- NULL
      try(mod <- lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat))
      mod
   }, parallel=16)
summary(modList$BOSAA) ## return model for BOS & AA
```



- Goal: Compute a linear model for each destination and airline (unique carrier) combination in parallel with requested 16 R engines
- View the model for Boston and AA
- Note: Some combinations have no data, so the try() is used

When does processing actually occur?

- Case 1: Using data.frame for FUN.VALUE parameter
  - ore.groupApply returns ore.frame promptly, which contains the underlying rqGroupEval call query
  - The query execution is deferred to the point when ore.frame is pulled and the return of the query is relational data (there is no serialization/deserialization process taking place on the query result)
- Case 2: No FUN.VALUE parameter (default to NULL)
  - ore.groupApply returns ore.list, which contains rqGroupEval query execution result serialized into a temp table
  - Query execution at the time ore.groupApply is called
  - ore.list will go through deserialization to the R object when ore.pull is called (showing the result at R client)

- For ore.groupApply, adding a FUN.VALUE parameter does two things
  - Format the result to be a single ore.frame
  - Changes when the processing occurs *from* time of ore.groupApply invocation *to* time of result ore.frame read
- When the result from ore.groupApply is large, Option 1 could be faster than Option 2
  - Option 1 does not involve (de)serialize process on the output

ore.indexApply – task-parallel execution

```
illustrateIndexApply <-
   function(index,a,b,c) {
     x <- "Hi"
     paste(x,index,a,b,c,sep=":")
   }

ore.indexApply(2,
               illustrateIndexApply,
               a=42, b="xyz", c=TRUE,
               parallel=TRUE)
```

- Goal: illustrate using index as input to vary behavior of function
- Return ore.list, one element per index for 2 indexes

```
R> illustrateIndexApply <-
+    function(index,a,b,c) {
+      x <- "Hi"
+      paste(x,index,a,b,c,sep=":")
+    }
R>
R> ore.indexApply(2,
+                 illustrateIndexApply,
+                 a=42, b="xyz",c=TRUE,
+                 parallel=TRUE)
$`1`
[1] "Hi:1:42:xyz:1"

$`2`
[1] "Hi:2:42:xyz:1"
```

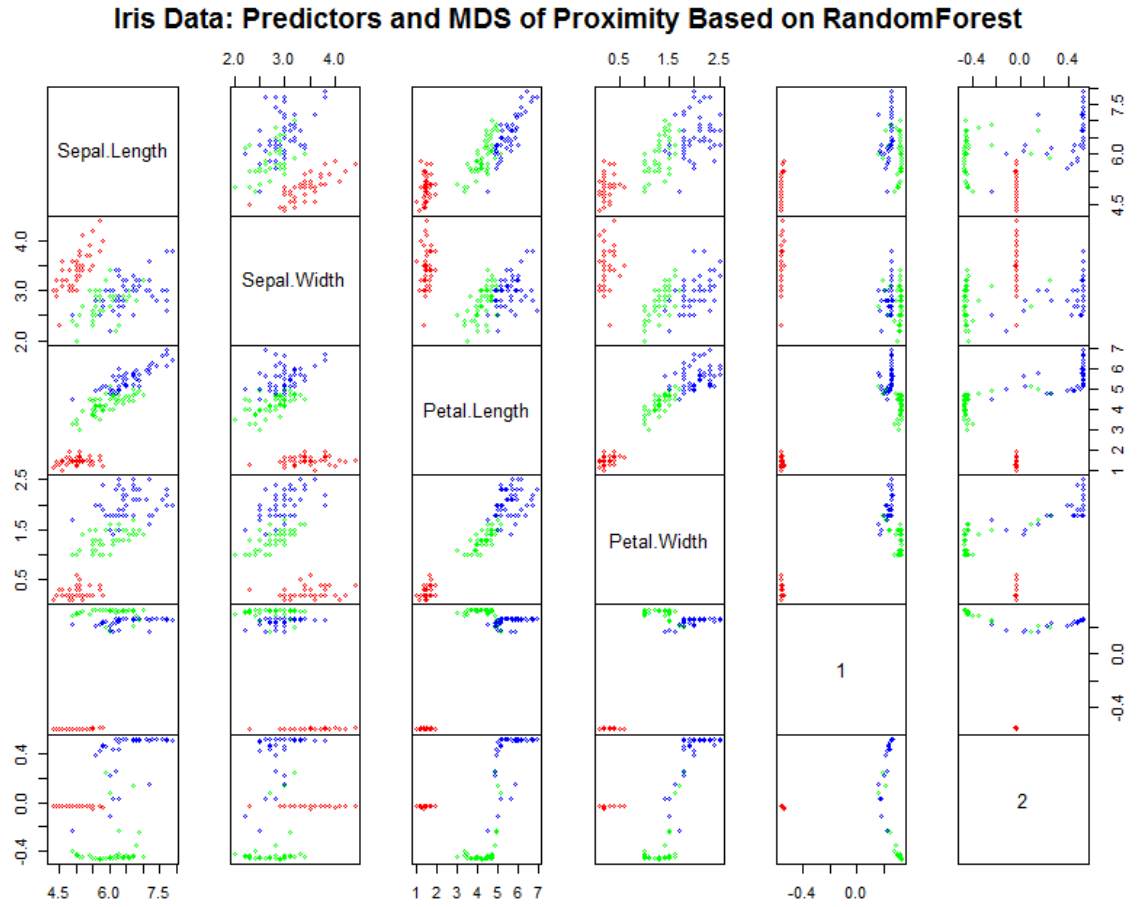
Viewing database server-generated graphics in client

```
ore.doEval(function (){
  set.seed(71)
  library(randomForest)
  iris.rf <- randomForest(Species ~ ., data=iris, importance=TRUE, proximity=TRUE)

  imp <- round(importance(iris.rf), 2) # Look at variable importance

  iris.mds <- cmdscale(1 - iris.rf$proximity, eig=TRUE) # Do MDS on 1 - proximity
  op <- par(pty="s")
  pairs(cbind(iris[,1:4], iris.mds$points), cex=0.6, gap=0,
        col=c("red", "green", "blue")[as.numeric(iris$Species)],
        main="Iris Data: Predictors and MDS of Proximity Based on RandomForest")
  par(op)
  list(importance = imp, GOF = iris.mds$GOF)
})
```

- Goal: generate graph at database server, view on client and return importance from R randomForest model

Results

```
...
R> ore.doEval(function (){
+   set.seed(71)
+   iris.rf <- randomForest(Species ~ ., data=iris, importance=TRUE,
+                           proximity=TRUE)
+   ## Look at variable importance:
+   imp <- round(importance(iris.rf), 2)
+   ## Do MDS on 1 - proximity:
+   iris.mds <- cmdscale(1 - iris.rf$proximity, eig=TRUE)
+   op <- par(pty="s")
+   pairs(cbind(iris[,1:4], iris.mds$points), cex=0.6, gap=0,
+         col=c("red", "green", "blue")[as.numeric(iris$Species)],
+         main="Iris Data: Predictors and MDS of Proximity Based on RandomForest")
+   par(op)
+   list(importance = imp, GOF = iris.mds$GOF)
+ })
$importance
             setosa versicolor virginica MeanDecreaseAccuracy MeanDecreaseGini
Sepal.Length   1.40       1.76      1.77                 1.38             8.77
Sepal.Width    0.99       0.25      1.25                 0.71             2.19
Petal.Length   3.73       4.37      4.26                 2.50            42.54
Petal.Width    3.86       4.42      4.35                 2.55            45.77

$GOF
[1] 0.7842697 0.8183542
```




```
ore.doEval(function (){
   …
}, ore.graphics=TRUE, ore.png.height=700, ore.png.width=500)
```

Parameterizing server-generated graphics in client

```
ore.doEval(function (rounding = 2, colorVec= c("red", "green", "blue")){
  set.seed(71)
  library(randomForest)
  iris.rf <- randomForest(Species ~ ., data=iris, importance=TRUE, proximity=TRUE)
  ## Look at variable importance:
  imp <- round(importance(iris.rf), rounding)
  ## Do MDS on 1 - proximity:
  iris.mds <- cmdscale(1 - iris.rf$proximity, eig=TRUE)
  op <- par(pty="s")
  pairs(cbind(iris[,1:4], iris.mds$points), cex=0.6, gap=0,
        col=colorVec[as.numeric(iris$Species)],
        main="Iris Data: Predictors and MDS of Proximity Based on RandomForest")
  par(op)
  list(importance = imp, GOF = iris.mds$GOF)
  },
  rounding = 3, colorVec = c("purple","black","pink"))
```

Control Arguments Summary

- Arguments starting with 'ore.' are special control arguments
  - Not passed to the function specified by 'FUN' or 'FUN.NAME' arguments
  - Controls what happens before or after the execution of the function
- Supported control arguments include:
  - **ore.drop** - controls the input data. If TRUE, a one column input data.frame will be converted to a vector (default: TRUE)
  - **ore.na.omit –** controls missing value handling. If TRUE, rows or vectors with no data will be removed from processing
  - **ore.connect** - controls whether to automatically connect to ORE inside the closure. This is equivalent to doing an ore.connect call with the same credentials as the client session. (default: FALSE)
  - **ore.graphics** - controls whether to start a graphical driver and look for images (default: TRUE)
  - **ore.envAsEmptyevn** - controls whether referenced environments in an object should be replaced with an empty environment during serialization
  - **ore.png.*** - if ore.graphics=TRUE, provides additional parameters for png graphics device driver. Use “ore.png.” prefix to arguments of png function. E.g., if ore.png.height is supplied, argument “height” will be passed to the png function. If not set, the standard default values for the png function are used. See ?png for details

```
png(filename = "Rplot%03d.png", width = 480, height = 480, units = "px", pointsize = 12,
    bg = "white", res = NA, ..., type = c("cairo", "cairo-png", "Xlib", "quartz"), antialias)
```

Viewing R Script Repository Contents

```
ore.scriptList(name="Example1")
ore.scriptList(pattern="Ex")
ore.scriptList(type="user")
```

type: A scalar character string specifying the type of R script to list.

- 'user' (default) lists scripts created by current session user.
- 'grant' lists scripts with the read privilege granted to other users.
- 'granted' lists scripts the current session user is granted
- 'global' lists all global R scripts
- 'all' lists all scripts to which the current session user has read access to

```
# Alternatively, access these views directly
ore.sync(query = c(USER_RQ_SCRIPTS="select * from USER_RQ_SCRIPTS"))
row.names(USER_RQ_SCRIPTS) <- USER_RQ_SCRIPTS$NAME
USER_RQ_SCRIPTS$NAME  # List all scripts in SYS schema

ore.sync(query = c(ALL_RQ_SCRIPTS="select * from ALL_RQ_SCRIPTS"))
row.names(ALL_RQ_SCRIPTS) <- ALL_RQ_SCRIPTS$NAME
ALL_RQ_SCRIPTS$NAME  # List all scripts in SYS schema
```

Working with Connections

Connecting to databases from an embedded R function

- Enable embedded R function executing in database to access and manipulate database tables using SQL (CRUD operations) without requiring explicit login
- Scenario 1: Connect to the same database in which embedded R execution originated
  - Login credentials are already available from the current active database session
  - Steps: Obtain ROracle connection object. Use connection to execute queries. Disconnect
  - Example

```
con = dbConnect(Extproc())
...
dbGetQuery(con, 'query')
dbDisconnect(con)
```

- Scenario 2: Connect to other databases or more than 1 database
  - Login credentials not available since desired connection is to a different schema or different database instance
  - Steps: Obtain connection object via explicit login, Use connection to execute queries, Disconnect when done
  - Example

```
con = dbConnect(Oracle(), "login credentials/connect string")
      # OR con = dbConnect(Oracle(), "WALLET")
dbGetQuery(con, 'query');
dbDisconnect(con)
```

A few examples…

```
ore.doEval(function(){
  ore.is.connected()}    # returns FALSE
)

ore.doEval(function(){
  ore.is.connected()},   # returns TRUE
  ore.connect = TRUE
)

ore.doEval(function(){
  library(ORE)
  ore.connect("rquser", password = "rquser", conn_string = "inst1")
  ore.is.connected()     # returns TRUE
})
```

More examples…

```
ore.doEval(function() {
  ore.sync(table = "NARROW")
  NARROW <- ore.get("NARROW")
  head(ore.pull(NARROW))
  },
  ore.connect = TRUE)

ore.doEval(function() {
  ore.sync(table = "NARROW")
  ore.attach()
  head(ore.pull(NARROW))
  },
  ore.connect = TRUE)
```

```
R> ore.doEval(function() {
+   ore.sync(table = "NARROW")
+   NARROW <- ore.get("NARROW")
+   head(ore.pull(NARROW))
+   },
+   ore.connect = TRUE)
      ID GENDER AGE MARITAL_STATUS                COUNTRY EDUCATION OCCUPATION YRS_RESIDENCE CLASS
1 101501   <NA>  41         NeverM United States of America   Masters      Prof.             4     0
2 101502   <NA>  27         NeverM United States of America     Bach.      Sales             3     0
3 101503   <NA>  20         NeverM United States of America   HS-grad    Cleric.             2     0
4 101504   <NA>  45        Married United States of America     Bach.      Exec.             5     1
5 101505   <NA>  34         NeverM United States of America   Masters      Sales             5     1
6 101506   <NA>  38        Married United States of America   HS-grad      Other             4     0
R>
R> ore.doEval(function() {
+   ore.sync(table = "NARROW")
+   ore.attach()
+   head(ore.pull(NARROW))
+   },
+   ore.connect = TRUE)
      ID GENDER AGE MARITAL_STATUS                COUNTRY EDUCATION OCCUPATION YRS_RESIDENCE CLASS
1 101501   <NA>  41         NeverM United States of America   Masters      Prof.             4     0
2 101502   <NA>  27         NeverM United States of America     Bach.      Sales             3     0
3 101503   <NA>  20         NeverM United States of America   HS-grad    Cleric.             2     0
4 101504   <NA>  45        Married United States of America     Bach.      Exec.             5     1
5 101505   <NA>  34         NeverM United States of America   Masters      Sales             5     1
6 101506   <NA>  38        Married United States of America   HS-grad      Other             4     0
```

Another example…

```
ff <- function () {
  con = dbConnect(Extproc())
  dbGetQuery(con, "select * from NARROW where rownum < 3")
}

ore.doEval(ff)
```

```
R> ff <- function () {
+   con = dbConnect(Extproc())
+   dbGetQuery(con, "select * from NARROW where rownum < 3")
+ }
R>
R> ore.doEval(ff)
      ID GENDER AGE MARITAL_STATUS                  COUNTRY EDUCATION OCCUPATION YRS_RESIDENCE CLASS
1 101501   <NA>  41         NeverM United States of America   Masters      Prof.             4     0
2 101502   <NA>  27         NeverM United States of America     Bach.      Sales             3     0
```

Enabling multiple Package Versions

- Support different users needing different versions of an R library with embedded R execution
- Example
  - **user1** needs to use SLAM slam_0.1-30.tar.gz because a more recent version may break their code
  - **user2** wants to use a more recent SLAM (say slam_0.1-32.tar.gz)
  - Requires that both versions of SLAM work with the ORD version in use on the database server machine
  - Requires the newer SLAM is the "default" for the installation.
- Approach:
  - Maintain two different library paths
  - First install the packages to the desired paths

- At the OS shell:

```
export R_LIBS="/your/path1"
R CMD INSTALL -l /your/path1 slam_0.1-30.tar.gz

export R_LIBS="/your/path2"
R CMD INSTALL -l /your/path2 slam_0.1-32.tar.gz
```

- Then, in R:

```
library(slam, lib.loc="/your/path1")    # loads
slam version 0.1-30
library(slam, lib.loc="/your/path2")   # loads
slam version 0.1-32
```

- Within R:

```
install.packages("slam", lib="/your/path",
                 repos="http://cran.r-project.org")
```

note this will only install the latest version of the package slam

Summary – ORE Embedded R Execution

- Easily invoke user-defined R functions at the database server machine
- Control and secure R code that runs in Oracle Database
- Use data- and task-parallelism for user-defined R functions
  - Parallelism using multiple database managed and controlled R engines
  - Control degree of parallelism from R API **`parallel`** argument
  - Parallel simulations
- Product graphs at database server and return to R client

Learn More about Oracle's R Technologies…

http://oracle.com/goto/R

R Technologies from Oracle

Bringing the Power of R to the Enterprise


ORACLE®