



Session 4: Oracle R Enterprise 1.5.1 Embedded R Execution – SQL 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 SQL interface
- ORE-defined graphics function examples
- Returning R statistical results as a database table
- 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 R Scripts – SQL Interface

Embedded Script Execution – SQL Interface

| SQL Interface function | Purpose |
|------------------------|--|
| rqEval() | Invoke stand-alone R script |
| rqTableEval() | Invoke R script with full table as input |
| rqRowEval() | Invoke R script on one row at a time, or multiple rows in chunks |
| “rqGroupEval()” | Invoke R script on data partitioned by grouping column |
| | |
| sys.rqScriptCreate | Create named R script |
| sys.rqScriptDrop | Drop named R script |
| rqGrant | Grant access to an R script |
| rqRevoke | Revoke access to an R script |

rq*Eval() Table Functions

rqEval, rqTableEval, "rqGroupEval", rqRowEval

```
rq*Eval (  
  cursor(select * from <table-1>),  
  cursor(select * from <table-2>),  
  'select <column list> from <table-3> t',  
  <grouping col-name from table-1  
  or num rows>,  
  '<R-script-name>')
```

- Input cursor – Depending on the function, input passed as a whole table, group, or one row at a time to the R closure (not for rqEval)
- Parameters cursor – Parameters are specified through a select statement, scalars only – single row
- Output table definition – a query specifying the format of the result
If NULL, output is a serialized BLOB
If 'PNG', images only as BLOB column
If 'XML', XML string of images and return values
- Group name (optional) – Name of the grouping column
- Number of rows (optional) – number of rows to provide to function at one time
- Name of R function in repository to execute

Embedded Script Execution – SQL Interface

| ORE function | Input data | FUN.VALUE | Arguments | R Script | Special |
|--------------------|--|--|--|---------------|--------------------|
| rqEval | None Generated within R function Load via ore.pull Transparency layer ROracle data load Flat file data load | NULL (returns chunked blob) table signature (returns table) XML PNG | NULL or Cursor with single row select statement with scalar values | R script name | Not applicable |
| rqTableEval | | | | | Not applicable |
| rqRowEval | table cursor | | | | Integer >= 1 |
| “rqGroupEval” | | | | | Single column name |
| sys.rqScriptCreate | Not applicable | Not applicable | R script name | R function | Not applicable |
| sys.rqScriptDrop | | | Not applicable | R script name | Not applicable |

Passing parameters

- Directly pass **scalar numeric and string** values as R parameters via parameter cursor

```
select count(*)
from table(rqTableEval(
  cursor ( select x as "x", y as "y", parameter_value as "z"
           from geological_model_grid),
  cursor( select 30 as "levels", '/oracle/image.png' as "filename",
           1 "ore.connect" from dual),
  NULL,
  'Example5'));
```

- To pass non-scalar R parameter (e.g., a model or list)
 - use a *datastore* object – *preferred*
 - construct string that can be parsed inside user-defined function

rqEval – invoking a simple R script

```
begin
  --sys.rqScriptDrop('Example1')
  sys.rqScriptCreate('Example1',
'function() {
  ID <- 1:10
  res <- data.frame(ID = ID, RES = ID / 100)
  res}') ;
end;
/

select *
  from table(rqEval(NULL,
  'select 1 id, 1 res from dual',
  'Example1')) ;
```

```
SQL> begin
  sys.rqScriptCreate('Example1',
'function() {
  ID <- 1:10
  res <- data.frame(ID = ID, RES = ID / 100)
  res}') ;
end;
/
select *
  from table(rqEval(NULL,
  'select 1 id, 1 res from dual',
  'Example1')) ;
 2  3  4  5  6  7  8
PL/SQL procedure successfully completed.
```

```
SQL>  2  3  4
      ID  RES
-----
      1  .01
      2  .02
      3  .03
      4  .04
      5  .05
      6  .06
      7  .07
      8  .08
      9  .09
     10  .1
```

10 rows selected.

Embedded R Execution – SQL Interface

For model build and batch scoring

```
begin
  sys.rqScriptDrop('Example2');
  sys.rqScriptCreate('Example2',
'function(dat,datastore_name) {
  mod <- lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
  ore.save(mod,name=datastore_name, overwrite=TRUE)
  TRUE
}');
end;
/

select *
  from table(rqTableEval(
    cursor(select ARRDELAY,
                DISTANCE,
                DEPDELAY
           from   ontime_s),
    cursor(select 1 as "ore.connect",
            'myDatastore' as "datastore_name"
           from dual),
    'XML',
    'Example2' ));
```

```
begin
  sys.rqScriptDrop('Example3');
  sys.rqScriptCreate('Example3',
'function(dat, datastore_name) {
  ore.load(datastore_name)
  prd <- predict(mod, newdata=dat)
  prd[as.integer(rownames(prd))] <- prd
  res <- cbind(dat, PRED = prd)
  res}');
end;
/

select *
  from table(rqTableEval(
    cursor(select ARRDELAY, DISTANCE, DEPDELAY
           from   ontime_s
           where  year = 2003
           and    month = 5
           and    dayofmonth = 2),
    cursor(select 1 as "ore.connect",
            'myDatastore' as "datastore_name" from dual),
    'select ARRDELAY, DISTANCE, DEPDELAY, 1 PRED from ontime_s',
    'Example3'))
 order by 1, 2, 3;
```

Results

```
SQL> begin
  sys.rqScriptDrop('Example2');
  sys.rqScriptCreate('Example2',
'function(dat,datastore_name) {
  mod <- lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
  ore.save(mod,name=datastore_name, overwrite=TRUE)
}');
end;
/

select *
  from table(rqTableEval(
    cursor(select ARRDELAY,
                DISTANCE,
                DEPDELAY
           from   ontime_s),
    cursor(select 1 "ore.connect",
           'myDatastore' as "datastore_name"
           from dual),
    'XML',
    'Examp 2 3 4 5 6 7 8 9 le2' ));
```

PL/SQL procedure successfully completed.

```
SQL> SQL> 2 3 4 5 6 7 8 9 10 11
```

| NAME |
|-------|
| ----- |
| VALUE |
| ----- |

<root></root>

```
select *
  from table(rqTableEval(
    cursor(select ARRDELAY, DISTANCE, DEPDELAY
           from   ontime_s
           where  year = 2003
           and    month = 5
           and    dayofmonth = 2),
    cursor(select 1 "ore.connect",
           2 3 4 5 6 7 8 9 10 11
           'myDatastore' as "data
store_name" from dual),
    'select ARRDELAY, DISTANCE, DEPDELAY, 1 PRED from ontime_s',
    'Example3'))
 order by 1, 2, 3;
```

PL/SQL procedure successfully completed.

```
SQL> 2 3 4 5 6 7 8 9 10 11 12
```

| ARRDELAY | DISTANCE | DEPDELAY | PRED |
|----------|----------|----------|------------|
| ----- | ----- | ----- | ----- |
| -24 | 1190 | -2 | -3.1485154 |
| -20 | 185 | -9 | -8.6626137 |
| -16 | 697 | -9 | -9.2859791 |
| -15 | 859 | -8 | -8.5206878 |
| -15 | 2300 | -4 | -6.4250082 |
| -10 | 358 | 0 | -.21049053 |
| -10 | 719 | -8 | -8.3502363 |
| -8 | 307 | -2 | -2.0734536 |
| -4 | 1050 | -5 | -5.8656481 |
| -3 | 150 | 5 | 4.85539194 |
| -2 | 140 | -5 | -4.7577135 |

| ARRDELAY | DISTANCE | DEPDELAY | PRED |
|----------|----------|----------|------------|
| ----- | ----- | ----- | ----- |
| -2 | 543 | -2 | -2.3607861 |
| -2 | 1530 | -5 | -6.4500532 |

rqTableEval – singleton / real-time scoring

Pass cursor argument with values “selected” from dual

```
select *
from   table(rqTableEval(
        cursor(select 23 ARRDELAY, 3210 DISTANCE, 45 DEPDELAY
               from   dual),
        cursor(select 'myDatastore' "datastore_name",
                   1 "ore.connect" from dual),
        'select ARRDELAY, DISTANCE, DEPDELAY, 1 PRED from ontime_s',
        'Example3'));
```

rq*Eval functions: XML and PNG Image generation

Motivation

XML Generation

- R script output is often dynamic – not conforming to pre-defined structure
 - XML is very flexible and expressive
- R applications generate heterogeneous data
 - Statistics, new data, graphics, complex objects
 - Applications R results may often need these results
- Web-based applications typically can consume XML output
- Database-based applications need ready integration of R executed via SQL

PNG Image Generation

- Database-based applications can consume images directly from tables
- R scripts can generate multiple images
 - Enable returning image stream from R script
 - Images directly returned as a table consisting of identifier and BLOB columns
- Such results can be directly integrated with OBIEE for direct image access in dashboards

rqEval – “Hello World!” XML Example

```
set long 20000
set pages 1000
begin
  sys.rqScriptCreate('Example4',
    'function() {"Hello World!"}');
end;
/
select name, value
  from table(rqEval(
    NULL,
    'XML',
    'Example4'));
```

```
--
SQL> set long 20000
set pages 1000
begin
  sys.rqScriptCreate('Example4',
    'function() {"Hello World!"}');
end;
/
select name, value
  from table(rqEval(
    NULL,
    'XML',
    'Example4'));
SQL> SQL> 2 3 4 5
PL/SQL procedure successfully completed.

SQL> 2 3 4 5
NAME
-----
VALUE
-----

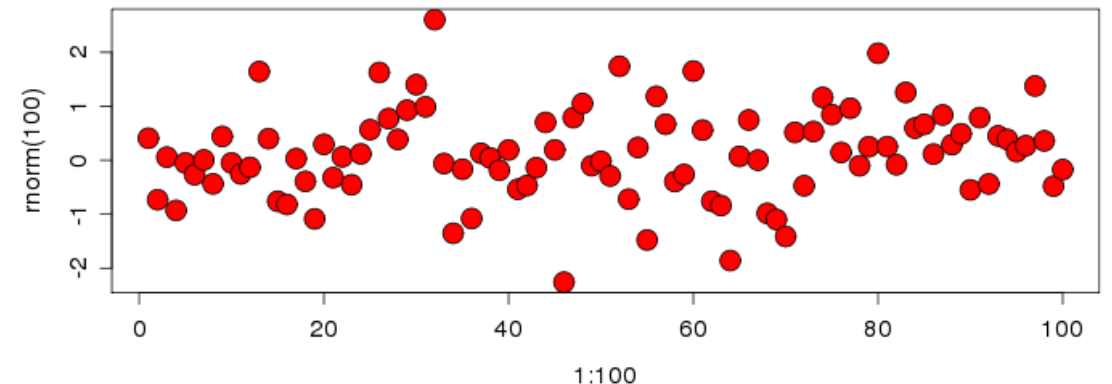
<root><vector_obj><ROW-vector_obj><value>Hello World!</value></ROW-vector_obj></
vector_obj></root>
```


rqEval – generate XML string for image output

```
set long 20000
set pages 1000
begin
  sys.rqScriptDrop('Example5');
  sys.rqScriptCreate('Example5',
'function() {
    res <- 1:10
    plot( 1:100, rnorm(100), pch = 21,
          bg = "red", cex = 2 )

    res
  }');
end;
/
select      value
from        table(rqEval( NULL, 'XML', 'Example5'));
```

- Executes function that plots 100 random numbers
- Returns a vector with values 1 to 10
- No parameters are specified
- Return the results as XML
- View the XML VALUE returned, which can be consumed by BI Publisher



Results

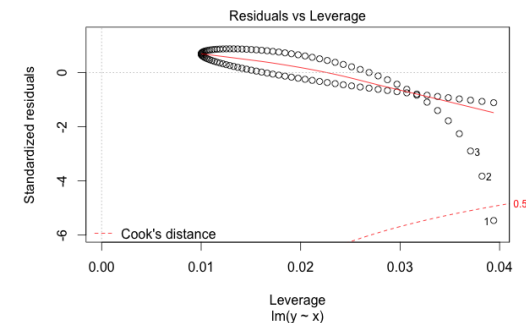
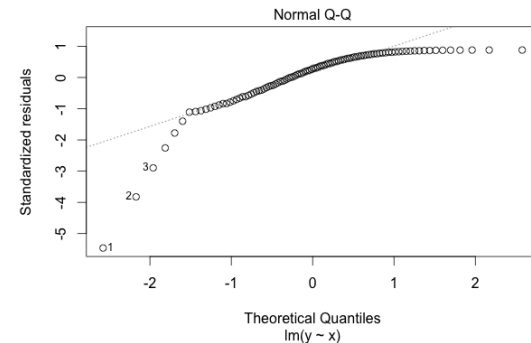
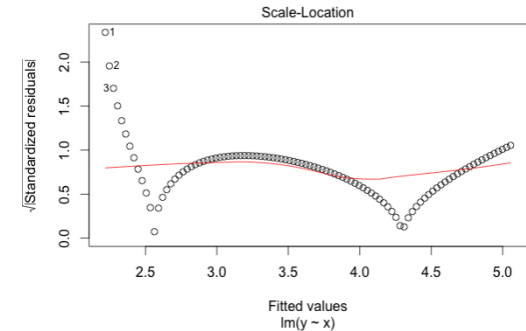
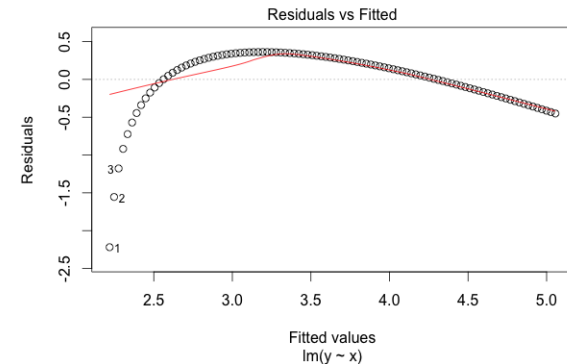
```
SQL> begin
  sys.rqScriptDrop('Example5');
  sys.rqScriptCreate('Example5',
'function(){
    res <- 1:10
    plot( 1:100, rnorm(100), pch = 21,
          bg = "red", cex = 2 )
    res
  }');
end;
/
 2  3  4  5  6  7  8  9 10 11
PL/SQL procedure successfully completed.

SQL> select  value
from      table(rqEval( NULL,'XML','Example5'));
 2
VALUE
-----
<root><R-data><vector_obj><ROW-vector_obj><value>1</value></ROW-vector_obj><ROW-
vector_obj><value>2</value></ROW-vector_obj><ROW-vector_obj><value>3</value></RO
W-vector_obj><ROW-vector_obj><value>4</value></ROW-vector_obj><ROW-vector_obj><v
alue>5</value></ROW-vector_obj><ROW-vector_obj><value>6</value></ROW-vector_obj>
<ROW-vector_obj><value>7</value></ROW-vector_obj><ROW-vector_obj><value>8</value
></ROW-vector_obj><ROW-vector_obj><value>9</value></ROW-vector_obj><ROW-vector_o
bj><value>10</value></ROW-vector_obj></vector_obj></R-data><images><image><img s
rc="data:image/png;base64"><![CDATA[ iVBORw0KGgoAAAANSUHEUgAAeAAAAH0CAYAAABj1L6VA
AAgAE1EQVR4n0zdd3iT1fvH8XfSnaSDWSizg0wle8sGQZaiCA6UoSCCIDJEEGQvBd1D+DoQUYagIMgSE
BCRIU0mgCDKkkLbdI/790fRH2JH2iZN2t6v68r1ZXN4zudpk9x5xjnhICKCUkoppbKU0dk.B1FJKqdxIC
3PCC: 1P5 411746604 111 5 83 386 446 11510 41174 111 11 5 305 4510: 1 54
```

rqEval – generate PNG image stream

```
begin
  sys.rqScriptDrop('Example6');
  sys.rqScriptCreate('Example6',
'function() {
    dat <- data.frame(y=log(1:100),
                      x = 1:100)

    plot(lm(y ~ x, dat))
  }');
end;
/
select  name, id, image
from    table(rqEval( NULL, 'PNG', 'Example6'));
-- best viewed from SQL Developer
```



| NAME | ID | IMAGE |
|----------|----------|-------|
| 1 (null) | 1 (BLOB) | |
| 2 (null) | 2 (BLOB) | |
| 3 (null) | 3 (BLOB) | |
| 4 (null) | 4 (BLOB) | |

Achieving “rqGroupEval” functionality

```
CREATE OR REPLACE PACKAGE ontimePkg AS
  TYPE cur IS REF CURSOR RETURN ontime_s%ROWTYPE;
END ontimePkg;
/
CREATE OR REPLACE FUNCTION ontimeGroupEval (
  inp_cur ontimePkg.cur,
  par_cur SYS_REFCURSOR,
  out_qry VARCHAR2,
  grp_col VARCHAR2,
  exp_txt CLOB)
RETURN SYS.AnyDataSet
PIPELINED PARALLEL_ENABLE (PARTITION inp_cur BY HASH (month))
CLUSTER inp_cur BY (month)
USING rqGroupEvalImpl;
/
```

- Create SQL package for type of input cursor
- Create function with PIPELINED PARALLEL_ENABLE, partition by “group by” variable

“rqGroupEval” build and rqRowEval score

```
begin
  --sys.rqScriptDrop('Example7');
  sys.rqScriptCreate('Example7',
'function(dat) {
  mod <- lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
  name <- paste("mod",dat$MONTH[1],sep="")
  assign(name,mod)
  try(ore.save(list=name,
               name="mydatastore",append=TRUE))
  TRUE}');
end;
/
select * from table(ontimeGroupEval(
  cursor(select /*+ parallel(t, 4) */ * from
ontime_s t),
  cursor(select 1 as "ore.connect" from dual),
  'XML', 'MONTH', 'Example7'));
```

```
begin
  --sys.rqScriptDrop('Example8');
  sys.rqScriptCreate('Example8',
'function(dat) {
  name <- paste("mod",dat$MONTH[1],sep="")
  ore.load("mydatastore",list=name)
  mod <- get(name)
  prd <- predict(mod, newdata=dat)
  prd[as.integer(rownames(prd))] <- prd
  cbind(dat, PRED = prd)}');
end;
/
select * from table(rqRowEval(
  cursor(select /*+ parallel(t, 4) */ MONTH, ARRDELAY,
DISTANCE, DEPDELAY
  from ontime_s t
  where year = 2003 and month in (5, 6)
  and dayofmonth = 2),
  cursor(select 1 as "ore.connect" from dual),
  'select MONTH, ARRDELAY, DISTANCE, DEPDELAY, 1 PRED from
ontime_s',
  1, 'Example8'));
```

“rqGroupEval” – multi-column

```
CREATE OR REPLACE PACKAGE ontimePkg AS
  TYPE cur IS REF CURSOR RETURN ontime_s%ROWTYPE;
END ontimePkg;
/
CREATE OR REPLACE FUNCTION ontimeGroupEval(
  inp_cur  ontimePkg.cur,
  par_cur  SYS_REFCURSOR,
  out_qry  VARCHAR2,
  grp_col  VARCHAR2,
  exp_txt  CLOB)
RETURN SYS.AnyDataSet
PIPELINED PARALLEL_ENABLE (PARTITION inp_cur BY HASH (year, month))
CLUSTER inp_cur BY (year, month)
USING rqGroupEvalImpl;
/
```

- Specify multiple column in HASH and BY

“rqGroupEval” build and rqRowEval score – multi-column

```
begin
  --sys.rqScriptDrop('Example7a');
  sys.rqScriptCreate('Example7a',
'function(dat) {
  mod <- lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
  name <- paste("mod", dat$YEAR[1],
                dat$MONTH[1], sep="")
  assign(name, mod)
  try(ore.save(list=name,
               name="mydatastore", append=TRUE))
  TRUE}');
end;
/
select * from table(ontimeGroupEval(
  cursor(select /*+ parallel(t, 4) */ * from
ontime_s t),
  cursor(select 1 as "ore.connect" from dual),
  'XML', 'YEAR,MONTH', 'Example7a'));
```

```
begin
  --sys.rqScriptDrop('Example8a');
  sys.rqScriptCreate('Example8a',
'function(dat) {
  name <- paste("mod", dat$YEAR[1],
                dat$MONTH[1], sep="")
  ore.load("mydatastore", list=name)
  mod <- get(name)
  prd <- predict(mod, newdata=dat)
  prd[as.integer(rownames(prd))] <- prd
  cbind(dat, PRED = prd)}');
end;
/
select * from table(rqRowEval(
  cursor(select /*+ parallel(t, 4) */ YEAR, MONTH, ARRDELAY,
DISTANCE, DEPDELAY
  from ontime_s t
  where year > 2003 and month in (5, 6)
  and dayofmonth = 2),
  cursor(select 1 as "ore.connect" from dual),
  'select YEAR, MONTH, ARRDELAY, DISTANCE, DEPDELAY, 1 PRED
from ontime_s',
  1, 'Example8a'));
```

“rqGroupEval” build – with minimal data

```
CREATE OR REPLACE PACKAGE ontimePkg AS
  TYPE rec IS RECORD (MONTH      NUMBER(38),
                     ARRDELAY   NUMBER(38),
                     DISTANCE    NUMBER(38),
                     DEPDELAY    NUMBER(38));

  TYPE cur IS REF CURSOR RETURN rec;
END ontimePkg;
/

CREATE OR REPLACE FUNCTION ontimeGroupEval(
  inp_cur  ontimePkg.cur,
  par_cur  SYS_REFCURSOR,
  out_qry  VARCHAR2,
  grp_col  VARCHAR2,
  exp_txt  CLOB)
RETURN SYS.AnyDataSet
PIPELINED PARALLEL_ENABLE (PARTITION inp_cur BY
  HASH (month))
CLUSTER inp_cur BY (month)
USING rqGroupEvalImpl;
/
```

```
begin
  --sys.rqScriptDrop('Example9');
  sys.rqScriptCreate('Example9',
'function(dat) {
  mod <- lm(ARRDELAY ~ DISTANCE + DEPDELAY, dat)
  name <- paste("mod",dat$MONTH[1],sep="")
  assign(name,mod)
  try(ore.save(list=name,name="mydatastore",append=TRUE)
  TRUE}');
end;
/

select *
  from table(ontimeGroupEval(
    cursor(select MONTH, ARRDELAY,
                DISTANCE, DEPDELAY
            from ontime_s),
    cursor(select 1 as "ore.connect" from dual),
    'XML', 'MONTH', 'Example9'));
```


rqRowEval – score

```
begin
  --sys.rqScriptDrop('Example10');
  sys.rqScriptCreate('Example10',
'function(dat) {
    name <- paste("mod",dat$MONTH,sep="")
    ore.load("mydatastore",list=name)
    mod <- get(name)
    prd <- predict(mod, newdata=dat)
    prd[as.integer(rownames(prd))] <- prd
    cbind(dat, PRED = prd)}');
end;
/
select * from table(rqRowEval(
  cursor(select /*+ parallel(t, 4) */ MONTH, ARRDELAY, DISTANCE, DEPDELAY
    from ontime_s t
    where year = 2003 and month in (5, 6)
    and dayofmonth = 2),
  cursor(select 1 as "ore.connect" from dual),
  'select MONTH, ARRDELAY, DISTANCE, DEPDELAY, 1 PRED from ontime_s', 1, 'Example10'));
```

rqEval – with other data types

```
begin
  --sys.rqScriptDrop('Example11');
  sys.rqScriptCreate('Example11',
    'function (num = 10, scale = 100) {
      ID <- seq(num)
      data.frame(ID = ID, RES = ID / scale, CHAR="x")
    }');
end;
/
select * from table(rqEval(NULL,
  'select 1 "ID", 1 "RES", 'a' "CHAR" from dual',
  'Example11'));
```

```
SQL> begin
  --sys.rqScriptDrop('Example11');
  sys.rqScriptCreate('Example11',
    'function (num = 10, scale = 100) {
      ID <- seq(num)
      data.frame(ID = ID, RES = ID / scale, CHAR="x")
    }');
end;
/
select * from table(rqEval(NULL,
  'select 1 "ID", 1 "RES", 'a' "CHAR" from dual',
  'Example11'));
 2  3  4  5  6  7  8  9
PL/SQL procedure successfully completed.

SQL>
  2  3
-----
   ID      RES C
-----
    1      .01 x
    2      .02 x
    3      .03 x
    4      .04 x
    5      .05 x
    6      .06 x
    7      .07 x
    8      .08 x
    9      .09 x
   10      .1 x

10 rows selected.
```

rqEval – with other data types

```
begin
  --sys.rqScriptDrop('Example12');
  sys.rqScriptCreate('Example12',
    '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
    }');
end;
/
select * from table(rqEval(NULL,
  'select 1 "ID", 1 "RES", ''a'' "CHAR1", 1 "BOOL" from dual',
  'Example12'));
```

```
SQL> begin
  --sys.rqScriptDrop('Example12');
  sys.rqScriptCreate('Example12',
    '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
    }');
end;
/
select * from table(rqEval(NULL,
  'select 1 "ID", 1 "RES", ''a'' "CHAR1", 1 "BOOL" from dual',
  'Example12'));
 2  3  4  5  6  7  8  9 10 11
PL/SQL procedure successfully completed.

SQL>
 2  3  4  5  6  7  8  9 10 11
  ID RES C  BOOL
-----
 1   .01 x    1
 2   .02 x    1
 3   .03 x    1
 4   .04 x    0
 5   .05 x    0
 6   .06 x    0
 7   .07 x    0
 8   .08 x    0
 9   .09 x    0
10   .1 x     0

10 rows selected.
```

rq*Eval Output Specification Summary

| Output Type Parameter Value | Data Returned |
|--|--|
| SQL table specification string e.g., "select 1 ID, 'aaa' VAL from dual" | Table – streamed structured data Image stream is discarded |
| 'XML' | XML string May contain both data and image data Images represented as base 64 encoding of PNG |
| 'PNG' | Structured output data ignored Table with 1 image per row NAME varchar2(4000) ID number IMAGE blob |

Embedded R Execution – Privileges

| Database Roles | R Interface | SQL Interface |
|----------------|--|--|
| RQADMIN | Execute ore.doEval and ore.*Apply functions Use FUN argument to dynamically create R scripts Execute ore.scriptCreate and ore.scriptDrop functions Access USER_RQ_SCRIPTS or ALL_RQ_SCRIPTS views Execute ore.grant and ore.drop functions | Execute rq*Eval functions Execute sys.rqScriptCreate and sys.rqScriptDrop functions Access USER_RQ_SCRIPTS or ALL_RQ_SCRIPTS views Execute rqGrant and rqDrop functions Execute rqConfigSet function |

```
grant RQADMIN to <USER>;
```

Note that any user can invoke embedded R scripts that are public in the R Script Repository

Embedded Graphic Function Examples

Why use embedded R graph functions?

- Same reasons for embedded R in general
 - More powerful database server
 - More efficient transfer of data between database and R engine
 - Execute scripts from SQL

ORE-defined graph function examples

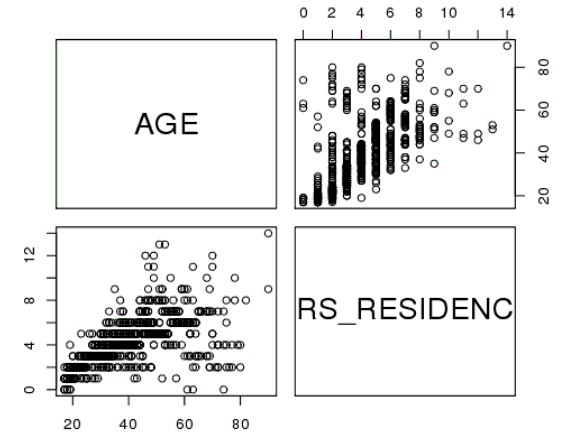
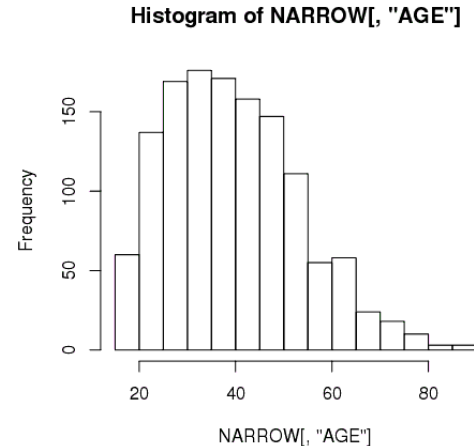
- ORE-defined scripts with a reserved name prefix: 'RQG\$'
- Prefix followed by a function name from 'graphics' package that the script wraps
- Depending on function, takes either the first, the first and second, or all columns of the input 'data.frame'
- For use with
 - ore.tableApply, rqTableEval
 - ore.groupApply, "rqGroupEval"
 - ore.rowApply, rqRowEval
- Each function allows '...' parameters to enable passing graphics function parameters to the wrapped function

ORE-defined graphics function examples

| ORE Embedded R Function | Wraps R Function | Performs function on ... of input ore.frame object |
|-------------------------|------------------|---|
| RQG\$plot1d | plot | first column |
| RQG\$plot2d | plot | first two columns |
| RQG\$hist | hist | first column |
| RQG\$boxplot | boxplot | first column |
| RQG\$smoothScatter | smoothScatter | first two columns |
| RQG\$cdplot | cdplot | first two columns |
| RQG\$pairs | pairs | all columns |
| RQG\$matplot | matplot | all columns |

rqEval – invoking a simple R script

```
hist(NARROW[, "AGE"])  
pairs(NARROW[, c("AGE", "YRS_RESIDENCE")])
```



```
select *  
  from table(rqTableEval(cursor(select AGE from NARROW), NULL,  
                              'PNG',  
                              'RQG$hist')));  
  
select *  
  from table(rqTableEval(cursor(select AGE, YRS_RESIDENCE from NARROW),  
                              NULL,  
                              'PNG',  
                              'RQG$pairs')));  
  
-- best viewed from SQL Developer
```

ORE-defined package and version function examples

| ORE Embedded R Function | Wraps R Function | Arguments | Return Values |
|-------------------------|--------------------|--------------|---|
| RQ\$R.Version | R.Version | None | R version-relevant info |
| RQ\$getRversion | getRversion | None | R version number |
| RQ\$installed.packages | Installed.packages | None | Package name, version number, package installation location, installed packages |
| RQ\$packageVersion | packageVersion | Package name | Package version number |

Invoking a simple R script from R and SQL

```
select * from table(rqEval(NULL,  
  'select cast(''a'' as varchar2(14)) "name", cast(''a''  
  as varchar2(50)) "value" from dual',  
  'RQ$R.Version'));
```

```
ore.doEval(FUN.NAME="RQ$R.Version")  
ore.doEval(FUN.NAME="RQ$getRversion")  
ore.doEval(FUN.NAME="RQ$installed.packages")  
ore.doEval(FUN.NAME="RQ$packageVersion",pkg="OREbase")  
  
ore.scriptList(name="RQ$packageVersion",type="global")
```

```
R> ore.doEval(FUN.NAME="RQ$R.Version")  
   name                                     value  
1  platform                               x86_64-unknown-linux-gnu  
2   arch                                   x86_64  
3   os                                     linux-gnu  
4  system                               x86_64, linux-gnu  
5  status                                <NA>  
6  major                                  3  
7  minor                                  0.1  
8   year                                  <NA>  
9  month                                  <NA>  
10 day                                   <NA>  
11 svn rev                                -99  
12 language                               R  
13 version.string Oracle Distribution of R version 3.0.1 (--)  
14 nickname                                Good Sport
```

Viewing contents of the R Script Repository

```
# List all user scripts
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
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

ore.sync(table = "RQ_SCRIPTS", schema = "SYS")
ore.attach(schema = "SYS")
row.names(RQ_SCRIPTS) <- RQ_SCRIPTS$NAME
RQ_SCRIPTS$NAME # List all scripts in SYS schema
RQ_SCRIPTS["RQ$packageVersion",] # View script body
```

Returning R statistical results as a DB table

A step-by-step example

Step 1: invoke from command line, understand results

```
mod <- princomp(USArrests, cor = TRUE)
class(mod)
mod
dat <- ore.push(USArrests) # create ore.frame
```

```
R> mod <- princomp(USArrests, cor = TRUE)
R> class(mod)
[1] "princomp"
R> mod
Call:
princomp(x = USArrests, cor = TRUE)

Standard deviations:
      Comp.1      Comp.2      Comp.3      Comp.4
1.5748783 0.9948694 0.5971291 0.4164494

4 variables and 50 observations.
R> dat <- ore.push(USArrests)
```

Step 2: wrap in function, invoke from ore.tableApply

```
res <- ore.tableApply(dat,  
  function(dat) {  
    princomp(dat, cor=TRUE)  
  })  
class(res)  
res.local <- ore.pull(res)  
class(res.local)  
str(res.local)  
res.local  
res
```

```
R> res <- ore.tableApply(dat,  
+ function(dat) {  
+ princomp(dat, cor=TRUE)  
+ })  
R> class(res)  
[1] "ore.object"  
attr(,"package")  
[1] "OREembed"  
R> res.local <- ore.pull(res)  
R> class(res.local)  
[1] "princomp"
```


Step 2: ...more results

```
R> str(res.local)
List of 7
 $ sdev : Named num [1:4] 1.575 0.995 0.597 0.416
 ..- attr(*, "names")= chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
 $ loadings: loadings [1:4, 1:4] -0.536 -0.583 -0.278 -0.543 0.418 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : chr [1:4] "Murder" "Assault" "UrbanPop" "Rape"
 .. ..$ : chr [1:4] "Comp.1" "Comap.2" "Comp.3" "Comp.4"
 $ center : Named num [1:4] 7.79 170.76 65.54 21.23
 ..- attr(*, "names")= chr [1:4] "Murder" "Assault" "UrbanPop" "Rape"
 $ scale : Named num [1:4] 4.31 82.5 14.33 9.27
 ..- attr(*, "names")= chr [1:4] "Murder" "Assault" "UrbanPop" "Rape"
 $ n.obs : int 50
 $ scores : num [1:50, 1:4] -0.986 -1.95 -1.763 0.141 -2.524 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : chr [1:50] "1" "2" "3" "4" ...
 .. ..$ : chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
 $ call : language princomp(x = dat, cor = TRUE)
 - attr(*, "class")= chr "princomp"
```

```
R> res.local
Call:
princomp(x = dat, cor = TRUE)

Standard deviations:
      Comp.1      Comp.2      Comp.3      Comp.4
1.5748783 0.9948694 0.5971291 0.4164494

4 variables and 50 observations.
R> res
Call:
princomp(x = dat, cor = TRUE)

Standard deviations:
      Comp.1      Comp.2      Comp.3      Comp.4
1.5748783 0.9948694 0.5971291 0.4164494

4 variables and 50 observations.
```

Step 3: determine what results we really need

```
res <- ore.tableApply(dat,  
  function(dat) {  
    mod <- princomp(dat, cor=TRUE)  
    dd <- dim(mod$loadings)  
    ldgs <-  
as.data.frame(mod$loadings[1:dd[1],1:dd[2]])  
    ldgs$variables <- row.names(ldgs)  
    ldgs  
  })  
class(res)  
res
```

```
ore.create(USArrests, table="USARRESTS")
```

```
R> res <- ore.tableApply(dat,  
+ function(dat) {  
+ mod <- princomp(dat, cor=TRUE)  
+ dd <- dim(mod$loadings)  
+ ldgs <- as.data.frame(mod$loadings[1:dd[1],1:dd[2]])  
+ ldgs$variables <- row.names(ldgs)  
+ ldgs  
+ })  
R> class(res)  
[1] "ore.object"  
attr(,"package")  
[1] "OREembed"  
R> res  


|           | Comp.1     | Comp.2     | Comp.3     | Comp.4      |         |
|-----------|------------|------------|------------|-------------|---------|
| variables |            |            |            |             |         |
| Murder    | -0.5358995 | 0.4181809  | -0.3412327 | 0.64922780  | Murder  |
| Assault   | -0.5831836 | 0.1879856  | -0.2681484 | -0.74340748 | Assault |
| UrbanPop  | -0.2781909 | -0.8728062 | -0.3780158 | 0.13387773  |         |
| UrbanPop  |            |            |            |             |         |
| Rape      | -0.5434321 | -0.1673186 | 0.8177779  | 0.08902432  | Rape    |


```

Step 4: create script in repository using SQL

(could create from R API as well)

```
begin
sys.rqScriptDrop('princomp.loadings');
sys.rqScriptCreate('princomp.loadings',
  'function(dat) {
    mod <- princomp(dat, cor=TRUE)
    dd <- dim(mod$loadings)
    ldgs <- as.data.frame(mod$loadings[1:dd[1],1:dd[2]])
    ldgs$variables <- row.names(ldgs)
    ldgs
  }');
end;
/
```

Step 5: invoke from SQL select statement

```
select *
from table(rqTableEval( cursor(select * from USARRESTS),
                             NULL,
                             'select 1 as "Comp.1", 1 as "Comp.2", 1 as "Comp.3", 1 as
"Comp.4", cast(''a'' as varchar2(12)) "variables" from dual',
                             'RUSER.princomp.loadings')));
```

IMPORTANT: As of ORE 1.5, if a script name contains a `'` you will need to prefix the name with the user schema name where it was created. Otherwise it will not be found. Do not add this name upon creation, however.

To avoid this, simply do not use `'` in script names.

```
SQL> select *
from table(rqTableEval( cursor(select * from USARRESTS),
                             NULL,
                             'select 1 as "Comp.1", 1 as
"Comp.2", 1 as "Comp.3", 1 as "Comp.4", cast(''a'' as
varchar2(12)) "variables" from dual',
                             'RUSER.princomp.loadings')));
```

| 2 | 3 | 4 | 5 | | |
|------------|------------|------------|------------|-----------|--|
| Comp.1 | Comp.2 | Comp.3 | Comp.4 | variables | |
| -.53589947 | .418180865 | -.34123273 | .649227804 | Murder | |
| -.58318363 | .187985604 | -.26814843 | -.74340748 | Assault | |
| -.27819087 | -.87280619 | -.37801579 | .133877731 | UrbanPop | |
| -.54343209 | -.16731864 | .817777908 | .089024323 | Rape | |

Summary

- Execute user-defined R functions from SQL for applications and operational systems
- Control and secure R code that runs in Oracle Database
- Use data- and task-parallelism for user-defined R functions
 - Interface function enable parallelism using multiple database R engines
 - Parallel simulations
- Rq*Eval enables:
 - Rich XML and PNG image output for integration with applications, dashboards and any tool or system that can consume such data
 - R data.frame results returned as query rowsets, i.e., tables

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®