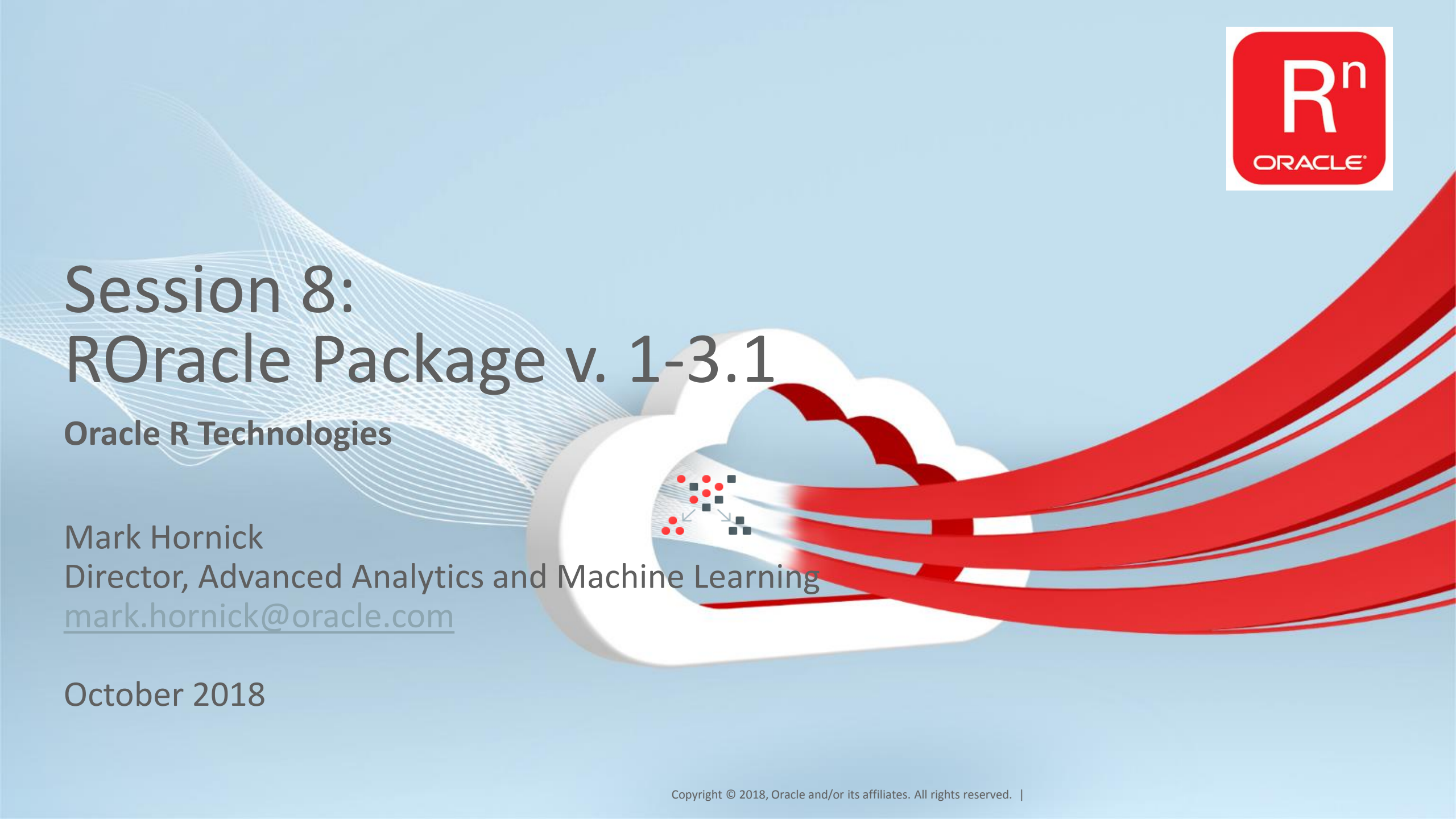


Session 8: ROracle Package v. 1-3.1

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.

Agenda

1. What is ROracle?
2. Using ROracle
3. Summary

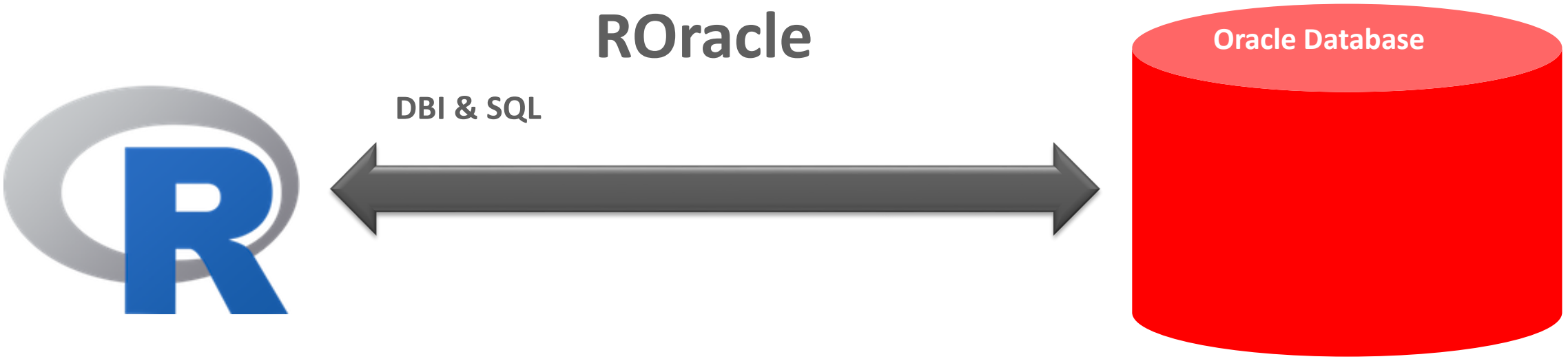
What is ROracle?

ROracle

- R package enabling connectivity to Oracle Database
  - Open source
  - Publically available on CRAN
- Execute SQL statements from R interface
- Oracle Database Interface (DBI) for R
- DBI –compliant Oracle driver based on OCI
- Requirements
  - Oracle Instant Client or Oracle Database Client

Examples from ROracle package documentation http://cran.r-project.org/web/packages/ROracle/ROracle.pdf

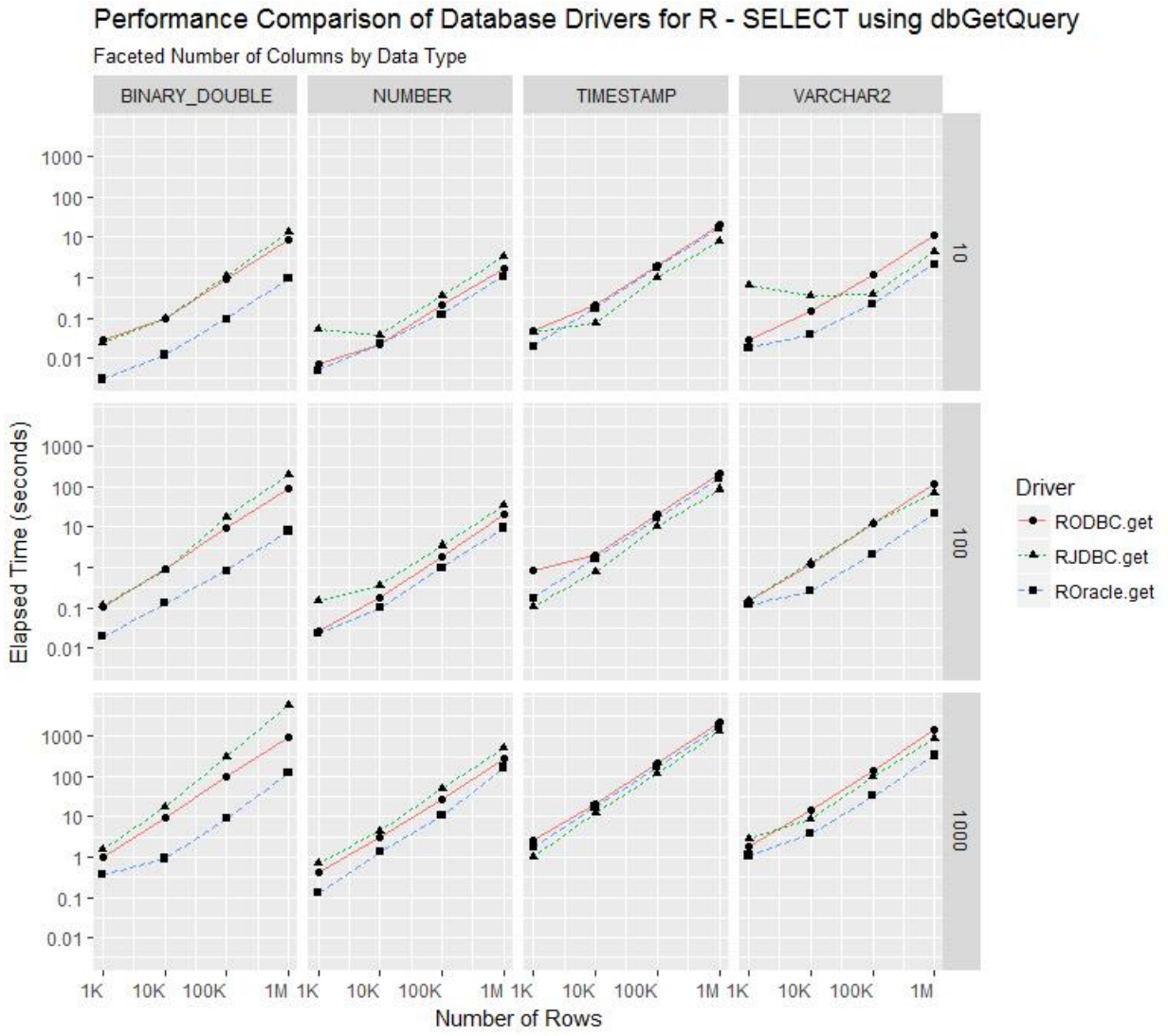
ROracle



- R package enabling connectivity to Oracle Database
  - Open source, publicly available on CRAN, free to R community
- Execute SQL statements from R interface
- Execute PLSQL procedures and functions from R interface
- Oracle Database Interface (DBI) for R based on OCI for high performance
- Supports **Oracle R Enterprise** database connectivity

Comparison reading database table to R data.frame

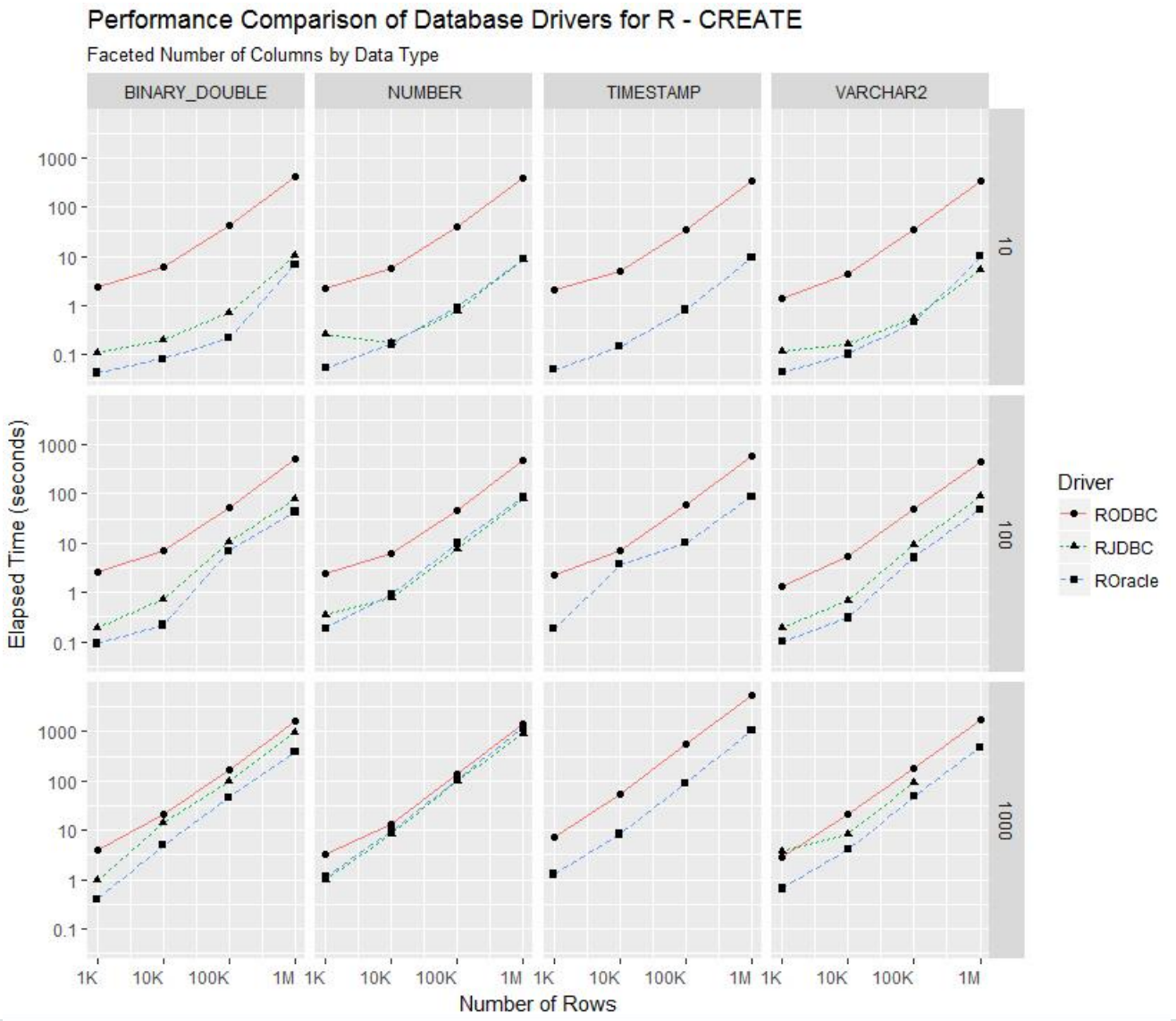
- ROracle
  - Up to 48X faster than RJDBC
  - Up to 11X faster than RODBC
  - Scales across NUMBER, BINARY_DOUBLE, VARCHAR2, TIMESTAMP data types



See https://blogs.oracle.com/R

Comparison writing database table from R data.frame

- ROracle
  - Up to 198X faster than RODBC (mean of 23X and median 9X)
  - Up to 6X faster than RJDBC (mean and median approaching 2X)
  - Scales across the remaining data types



ROracle 1-3.1 Enhancements

- Handles upper and lower cases in ora.type, ora.parameter_mode and ora.encoding attributes
- Support for OCIBindByName added for SQL and PL/SQL calls by specifying ora.parameter_name attribute; bind variables used in SQL or PL/SQL calls must be bound by the name specified in ora.parameter_name
- PL/SQL stored procedure/function and REF cursors supported
- OUT bind parameter must be initialized with 'NA' or 'NA' coerced to any other vector type
- Use OCIArrayDescriptorAlloc instead of OCIDescriptorAlloc for performance improvements with Oracle client release 11.1 or higher to build and run ROracle
- Removed functions (dbSetDataMappings and dbCallProc) that will not be available in 0.3.1 DBI specifications

ROracle 1-2.1 Enhancements

- Support for NATIVE, UTF8 and LATIN1 encoded data in query and results
- enhancement 20603162 - CLOB/BLOB enhancement on ROracle
- bug 15937661 – mapping of dbWriteTable BLOB, CLOB, NCLOB, NCHAR AND NVARCHAR columns. Data frame mapping to Oracle Database type is provided
- bug 16017358 – proper handling of NULL extproc context when passed to in ORE embedded R execution
- bug 16907374 - ROracle creates time stamp column for R Date with dbWriteTable
- ROracle now displays NCHAR, NVARCHAR2 and NCLOB data types defined for columns in the server using dbColumnInfo and dbGetInfo

ROracle 1-1.12 Enhancements

- Add bulk_write parameter to specify number of rows to bind at a time to improve performance for dbWriteTable and DML operations
- Date, Timestamp, Timestamp with time zone and Timestamp with local time zone data are maintained in R and Oracle's session time zone. Oracle session time zone environment variable ORA_SDTZ and R's environment variable TZ must be the same otherwise an error is reported when operating on any of these data types
- Performance improvements surrounding fetch

ROracle 1-1.11 Enhancements

- Performance enhancements for RAW data types and large result sets
- Cache resultset in memory before transferring to R to avoid unnecessary alloc and free using allocVector when result exceeds bulk_read rows
- Added session mode to connect as SYSDBA or using external authentication
- bug 17383542: Enhanced dbWritetable() & dbRemoveTable() to work on global schema

Using ROracle

Example – rolling back transactions

```
drv <- dbDriver("Oracle")

# Create the connection string
host <- "localhost"
port <- 1521
sid <- "orcl"
connect.string <- paste(
  "(DESCRIPTION=",
  "(ADDRESS=(PROTOCOL=tcp)(HOST=", host, ")(PORT=", port, "))",
  "(CONNECT_DATA=(SID=", sid, ")))", sep = "")

con <- dbConnect(drv, username = "scott",
                 password = "tiger",dbname=connect.string)
con
```

Example – rolling back transactions

```
dbReadTable(con, "EMP")

rs <- dbSendQuery(con, "delete from emp where deptno = 10")

dbReadTable(con, "EMP")
if(dbGetInfo(rs, what = "rowsAffected") > 1){
   warning("dubious deletion -- rolling back transaction")
   dbRollback(con)
}

dbReadTable(con, "EMP")
```

Example – username/password authentication

```
## create an Oracle instance and create one connection
drv <- dbDriver("Oracle")

## use username/password authentication - if using local database
con <- dbConnect(drv, username = "scott", password = "tiger")

## run a SQL statement by first creating a resultSet object
rs <- dbSendQuery(con, "select * from emp where deptno = 10")

## fetch records from the resultSet into a data.frame
data <- fetch(rs)
## extract all rows
dim(data)
data
```

Example – connect to an extproc for use within ORE ERE

```
## connect to an extproc (this assumes that driver has already
## been initialized in the embedded code by passing an external
## pointer representing extproc context)
con <- dbConnect(Extproc())

## run a SQL statement by first creating a resultSet object
rs <- dbSendQuery(con, "select * from dual")

## we now fetch records from the resultSet into a data.frame
data <- fetch(rs)   ## extract all rows
dim(data)
```

Example – unload driver

```
# create an Oracle instance
drv <- dbDriver("Oracle")
con <- dbConnect(drv, "scott", "tiger", dbname=connect.string)
res <- dbSendQuery(con, "select * from emp")
fetch(res, n = 5)
fetch(res)

# free resources occupied by result set
dbClearResult(res)
dbUnloadDriver(drv)
```

Example – getInfo methods

```
drv <- dbDriver("Oracle")
con <- dbConnect(drv, "scott", "tiger", dbname=connect.string)
rs <- dbSendQuery(con, "select * from emp")
dbGetStatement(rs)
dbHasCompleted(rs)
dbGetInfo(rs)

# DBIDriver info
names(dbGetInfo(drv))

# DBIConnection info
names(dbGetInfo(con))

# DBIResult info
names(dbGetInfo(rs))
```

Example – getInfo methods

```
drv  <- dbDriver("Oracle")
con1 <- dbConnect(drv, "scott", "tiger", dbname=connect.string)
res1 <- dbSendQuery(con1, "select * from emp where deptno = 10")
res2 <- dbSendQuery(con1, "select * from emp where deptno = 20")
con2 <- dbConnect(drv, "scott", "tiger")
res3 <- dbSendQuery(con2, "select * from dept")

## get all active statements
for(con in dbListConnections(drv))
  for (res in dbListResults(con))
    print(dbGetStatement(res))
```

Example – read/write table methods

```
con <- dbConnect(Oracle(), "scott", "tiger", dbname=connect.string)
if (dbExistsTable(con, "FOO", "SCOTT"))
dbRemoveTable(con, "FOO")
foo <- dbReadTable(con, "EMP")
row.names(foo) <- foo$EMPNO
foo <- foo[,-1]
dbWriteTable(con, "FOO", foo, row.names = TRUE)
dbWriteTable(con, "FOO", foo, row.names = TRUE, overwrite = TRUE)
dbReadTable(con, "FOO", row.names = 1)
dbGetQuery(con, "delete from foo")
dbWriteTable(con, "FOO", foo, row.names = TRUE, append = TRUE)
dbReadTable(con, "FOO", row.names = 1)
dbRemoveTable(con, "FOO")
dbListTables(con)
dbListFields(con, "EMP")
if (dbExistsTable(con, "RORACLE_TEST", "SCOTT"))
dbRemoveTable(con, "RORACLE_TEST")
```

Example – using attributes to map NCHAR, CLOB, BLOB, NCLOB columns correctly with dbWriteTable

```
str1           <- paste(letters, collapse="")
lstr1          <- paste(rep(str1, 200), collapse="")
raw.lst        <- vector("list",1)
lraw.lst       <- vector("list",1)
raw.lst[[1L]]  <- charToRaw(str1)
lraw.lst[[1L]] <- rep(charToRaw(str1), 200)
test.df <-
  data.frame(char=str1,nchar=str1,varchar=str1,
      clob=lstr1,nclob=lstr1, stringsAsFactors=FALSE)
test.df$raw.typ    <- raw.lst
test.df$blob       <- lraw.lst
test.df$char_max   <- str1
test.df$raw_max.typ <- raw.lst
test.df$nvchar     <- str1
test.df$nvchar_max <- str1
a <- as.POSIXct("2014-01-01 14:12:09.0194733")
b <- as.POSIXct("2014-01-01 14:12:09.01947")

# adding attributes
attr(test.df$clob, "ora.type") <- "clob"
attr(test.df$blob, "ora.type") <- "blob"
```

```
attr(test.df$nclob, "ora.type") <- "clob"
attr(test.df$nclob, "ora.encoding") <- "UTF-8"
attr(test.df$char_max, "ora.maxlength") <- 3000
attr(test.df$raw_max.typ, "ora.maxlength") <- 1000
attr(test.df$nvchar, "ora.encoding") <- "UTF-8"
attr(test.df$nvchar_max, "ora.encoding") <- "UTF-8"
attr(test.df$nvchar_max, "ora.maxlength") <- 1500
attr(test.df$char, "ora.type") <- "char"
attr(test_max.df$char, "ora.type") <- "char"
attr(test_max.df$char, "ora.maxlength") <- 0
attr(test.df$date_tz, "ora.type") <- "timestamp with time zone"
attr(test.df$date_ltz, "ora.type") <- "timestamp with local time
  zone"
attr(test.df$nchar, "ora.type") <- "char"
attr(test.df$nchar, "ora.encoding") <- "UTF-8"
attr(test.df$date_tz, "ora.fractional_seconds_precision") <- 9
attr(test_max.df$clob, "ora.type") <- "clob"
attr(test_max.df$blob, "ora.type") <- "blob"

dbWriteTable(con, name="TEST_TAB", value=test.df)
res <- dbReadTable(con, name="TEST_TAB")
```

Example – working with NaN

```
df <- data.frame(A=c(0,1,NaN,4), B=c(NA, 2,3,NaN))
con <- dbConnect(Oracle(), "scott", "tiger")
dbWriteTable(con,"TEST", df, row.names = FALSE,
  ora.number=FALSE)
```

Example – read/write table methods

```
# example of POSIXct usage
# A table is created using:
createTab <- "create table RORACLE_TEST(row_num number, id1 date,
                  id2 timestamp, id3 timestamp with time zone,
                  id4 timestamp with local time zone )"
dbGetQuery(con, createTab)

# Insert statement
insStr <- "insert into RORACLE_TEST values(:1, :2, :3, :4, :5)";

# Select statement
selStr <- "select * from RORACLE_TEST";

# Insert time stamp without time values in POSIXct form
x <- 1;
y <- "2012-06-05";
y <- as.POSIXct(y);
dbGetQuery(con, insStr, data.frame(x, y, y, y, y));
```

Example – read/write table methods

```
# Create a table
createStr <- "create table RORACLE_TEST2(row_num number, col1 varchar2(40))"
dbGetQuery(con, createStr)

# Insert data
insStr <- "insert into RORACLE_TEST2 values(:1, :2)"
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
y <- c("A","B","C","D","E","F","G","H","I")
dbGetQuery(con, insStr, data.frame(x, y))

# View data
selStr <- "select * from RORACLE_TEST2"
dbGetQuery(con, selStr)

# Update statement
updStr <- 'update RORACLE_TEST2 set col1 = :1 where row_num > :2'
dbGetQuery(con, updStr, data.frame(x="NEW",y= 5))
dbGetQuery(con, selStr)
```

Example – read/write table methods

```
# Insert date & times tamp with time values in POSIXct form
x <- 2;
y <- "2012-01-05 07:15:02";
y <- as.POSIXct(y);
z <- "2012-01-05 07:15:03.123";
z <- as.POSIXct(z);
dbGetQuery(con, insStr, data.frame(x, y, z, z, z));
# Insert list of date objects in POSIXct form
x <- c(3, 4, 5, 6);
y <- c('2012-01-05', '2011-01-05', '2018-01-05', '2020-01-05');
y <- as.POSIXct(y);
dbGetQuery(con, insStr, data.frame(x, y, y, y, y));
dbCommit (con)
# Select data and display it
res <- dbGetQuery(con, selStr)
res[,1]
res[,2]
res[,3]
res[,4]
res[,5]
```

Example – send query methods

```
drv <- dbDriver("Oracle")
con <- dbConnect(drv, "scott", "tiger")
res <- dbSendQuery(con, "select * from emp where deptno = :1",
                   data = data.frame(deptno = 10))
data <- fetch(res, n = -1)
res2 <- dbSendQuery(con, "select * from emp where deptno = :1",
                    data = data.frame(deptno = 10), prefetch=TRUE, bulk_read=2L)
data1 <- fetch(res2, n = -1)
res3 <- dbSendQuery(con, "select * from emp where deptno = :1",
                    data = data.frame(deptno = 10), bulk_read=10L)
data2 <- fetch(res3, n = -1)
res4 <- dbSendQuery(con, "select * from emp where ename = :1",
                    data = data.frame(ename = 'SMITH'))
data3 <- fetch(res4, n = -1)
```

Example – Oracle method

```
## create a Oracle instance and create one connection.
ora <- Oracle() ## or dbDriver("Oracle")
con <- dbConnect(ora, username = "scott", password = "tiger", dbname = "inst1")

## if you are connecting to a local database
con <- dbConnect(ora, username = "scott", password = "tiger")

## execute a statement and fetch output in chunks <= 5000 rows at a time
rs <- dbSendQuery(con, "select * from emp where deptno = 10")
while (!dbHasCompleted(rs)) {
  df <- fetch(rs, n = 5000)
  ## process df
}
dbClearResult(rs) ## done with this query
```

Example – Oracle method

```
## execute and fetch a statement with bind data
df <- dbGetQuery(con,
                 "select * from emp where deptno = :1", data = data.frame(depno = 10))

## create a copy of emp table
dbGetQuery(con, "create table foo as select * from emp")

## execute and bind an INSERT statement
my.data   = data.frame(empno = c(8001, 8002), ename = c('MUKHIN', 'ABOYOUN'))
more.data = data.frame(empno = c(8003), ename = c('JAMES'))
rs <- dbSendQuery(con, "insert into foo (empno, ename) values (:1, :2)",data = my.data)

## execute with more data
execute(rs, data = more.data)
df <- dbGetQuery(con, "select * from foo")
dbClearResult(rs) ## done with this query

dbCommit(con)     ## ok, everything looks fine
summary(ora)      ## a concise description of the driver
dbDisconnect(con) ## done with this connection
```

Invoking a stored procedure

```
dbGetQuery(con, 'create or replace
  PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
  BEGIN
     IF x < y THEN
        z:= x;
     ELSE
        z:= y;
     END IF;
  END;')


  a <- 23
  b <- 45
  c <- 10
  df <- data.frame(a, b, c)
  dbGetQuery(con,  'BEGIN findMin(:1, :2, :3); END;', df)
```

NOTE: that the OUT variable z will not be populated. ROracle does not support record or table types and REF cursor. But one can execute stored procedures, compute results, store those results in a table and then query that table

Summary

- R package enabling connectivity to Oracle Database
  - Open source
  - Publically available on CRAN
- Execute SQL statements from R interface
- Oracle Database Interface (DBI) for R
- DBI –compliant Oracle driver based on OCI
- Requirements
  - Oracle Instant Client or Oracle Database Client

To 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®