SciDB

SciDB is a free open-source database that organizes data in n-dimensional arrays. SciDB features include ACID transactions, parallel processing, distributed storage, and efficient sparse array storage. SciDB arrays consist of a coordinate system whose coordinates are called *dimensions*, and one or more values at each coordinate called *attributes*.

The "scidb" package helps R interact with SciDB by directly running SciDB queries using the native array functional language (AFL) and optionally returning results as data frames.

This vignette illustrates using SciDB from R by example. For more detailed information on the functions described in this vignette, see the manual pages in the package.

Installing the package

From CRAN (stable, but may lag features on GitHub by several months):

```
install.packages("scidb")
```

From the development repository on GitHub (stable branch):

```
devtools::install_github("Paradigm4/SciDBR")
```

"Stable" means that all CRAN checks and package unit tests pass when tested using the current SciDB release. We try to make sure that the scidb package works with *all* previous versions of SciDB but we only actively test against the current release version of the database.

Connecting to SciDB

The `scidbconnect` function establishes a connection to a simple HTTP network service called shim (https://github.com/Paradigm4/shim) running on a SciDB coordinator instance. The function may be safely called multiple times. The function return value contains the SciDB database connection state in an object of class afl.

The network interface optionally supports SSL encryption and SciDB authentication or HTTP digest user authentication.

Connect to SciDB on the default shim port and localhost

```
library("scidb")
db <- scidbconnect()
```

Connect to shim on an encrypted port 8083 with example SciDB authentication

```
db <- scidbconnect(port=8083, username="root", password="Paradigm4")
```

Use encrypted sessions when communicating with SciDB over public networks. SciDB user authentication is only supported by SciDB versions 15.7 and greater and only works over encrypted connections.

More on SciDB authentication, namespaces and user roles

See the official SciDB documentation for details about users, roles and namespaces https://paradigm4.atlassian.net/wiki/display/ESD/Security.

The global package option `scidb.prefix` is designed to simplify working with SciDB user roles from R. SciDB user roles are provided by the namespaces plugin and may require issuing the query `load_library('namespaces')` before use (usually performed by the SciDB system administrator). For instance, use the `scidb.prefix` package option to assume a special database user role "functionary" shown in the following example:

```
options(scidb.prefix="set_role('functionary')")
```

After setting that, all queries will be prefixed with the role-setting statement. You may use the `scidb.prefix` option to prefix queries with arbitrary AFL statements. Separate multiple statements with the semi-colon character.

Alternatively, use the `scidb_prefix()` function on a per-database connection level. The prefix will be displayed in the connection summary.

```
library(scidb)
db <- scidbconnect()
db <- scidb_prefix(db, "set_role('functionary')")
print(db)
```

Listing SciDB arrays, operators and macros

The `scidbconnect()` function returns a SciDB connection object of class afl. In addition to storing the connection state, the returned object has a few special methods. Printing the object shows a summary of the connection state. Applying the `ls()` function to the object returns a list of SciDB arrays (subject to any potential namespace-setting prefix expression). And the object itself is really a list that contains available SciDB AFL operator and macro functions established upon connection. Apply the `ls.str()` to the object to list all AFL operators and macros, or use the dollar sign accessor function and interactive completion with the TAB key.

```
db <- scidbconnect()
print(db)    # summarize connection
ls(db)       # quick list of arrays
ls.str(db)   # quick list of AFL operators
# db$<TAB>   # interactive completion list of AFL operators
```

Each listed AFL operator is presented as an R function. RStudio users can use the TAB key inside the function body to see suggested AFL operator argument completions. Alternatively, `ls.str(db)` shows the formal AFL operator arguments for each function. These functions can be used to compose AFL expressions from R, discussed in more detail below.

NOTE The list of operators and macros is established at connection time. If the database operators change after establishing the connection, for instance by loading a new SciDB plugin, then those changes will not be shown in the database connection object. New connection objects will show the current list of operators and macros.

Mapping SciDB arrays and query expressions to R variables

The `scidb()` function returns an object of class scidb that contains a reference to a SciDB array or query expression. The returned object is presented with a data frame-like view that lists the SciDB schema components.

```
x <- scidb(db, "build(<v:double>[i=1:2,2,0, j=1:3,1,0], i*j)")
print(x)
```

```
SciDB expression  build(<v:double>[i=1:2,2,0, j=1:3,1...
SciDB schema  <v:double> [i=1:2,2,0,j=1:3,1,0]
  variable dimension   type nullable start end chunk
1        i      TRUE  int64    FALSE     1   2     2
2        j      TRUE  int64    FALSE     1   3     1
3        v     FALSE double    FALSE
```

The R variable `x` is a sort of *SciDB array view*; an un-evaluated SciDB query expression. The value of `x` is evaluated by SciDB either lazily when needed or when explicitly requested. It's an S4 R object; then "name" slot contains the AFL expression corresponding to the object, for instance `x@name` in the above example.

Use the `store()` function to evaluate and materialize views in SciDB. The `store()` function stores the evaluation result into a new named SciDB array in the database, returning a new scidb R object that points to the SciDB array. The array name may be optionally specified or automatically generated and the array may optionally be stored in a SciDB temporary array.

```
y <- store(db, x, temp=TRUE)
print(y)
```

```
SciDB expression  R_array1687f71a392851102980131950
SciDB schema  <v:double> [i=1:2,2,0,j=1:3,1,0]
  variable dimension   type nullable start end chunk
1        i      TRUE  int64    FALSE     1   2     2
2        j      TRUE  int64    FALSE     1   3     1
3        v     FALSE double    FALSE
```

Note that the SciDB expression associated with the `y` R variable is now a named SciDB array (automatically named in this case); compare with the SciDB expression for `x` above.

SciDB array values associated with R variables are tied to R's garbage collector by default (unless the argument `gc=FALSE` is specified). When the R variable's contents are garbage-collected by R, the associated SciDB array is removed.

```
ls(db)   # (observe y's corresponding array in the list)
rm(y)
gc()
ls(db)   # (y's corresponding array will no longer appear)
```

The `schema()` function

Use the `schema()` function to display the SciDB schema of scidb objects verbatim or in parsed detail for attributes and dimensions.

```
schema(x)
# [1] "<v:double> [i=1:2:0:2; j=1:3:0:1]"

schema(x, "attributes")
#   name   type nullable
# 1    v double     TRUE

schema(x, "dimensions")
#   name start end chunk overlap
# 1    i     1   2     2       j
# 2    0     1   3     1       0
```

Uploading R values to SciDB with `as.scidb()` and downloading SciDB values to R with `as.R()`

The package provides limited convenience functions for converting and uploading R values to SciDB. The upload mechanism is much less efficient than available SciDB bulk load methods and should only be used for small to moderate-sized data. The following R objects are supported:

- logical, integer, numeric, and character vectors
- numeric matrices
- data frames with logical, character, integer, and numeric values (variable names may be changed to conform to SciDB naming convention)
- numeric column-compressed sparse matrices (class "dgCMatrix")
- raw objects

Factor values are uploaded as character value, replacing factor levels with their corresponding character strings.

The `as.scidb()` function returns a reference to a SciDB array containing the uploaded data. The following example uploads a data frame to SciDB, warning us that variable names were changed because SciDB does not support dots in names, and then downloads the resulting SciDB object data back into R.

```
x <- as.scidb(db, head(iris))
as.R(x)
```

```
Warning message:
In df2scidb(db, x, name = name, gc = gc, ...) :
  Attribute names have been changed

  i Sepal_Length Sepal_Width Petal_Length Petal_Width Species
1 1          5.1         3.5          1.4         0.2  setosa
2 2          4.9         3.0          1.4         0.2  setosa
3 3          4.7         3.2          1.3         0.2  setosa
4 4          4.6         3.1          1.5         0.2  setosa
5 5          5.0         3.6          1.4         0.2  setosa
6 6          5.4         3.9          1.7         0.4  setosa
```

Note that SciDB dimension indices (`i` above) are appended to the data when downloaded.

Running AFL queries

The scidb package defines two ways to compose and execute SciDB queries:

1. Directly running AFL expressions written as character strings (`iquery()`)
2. Composing AFL expressions from the database connection AFL functions, returning scidb objects.

The `iquery()` convenience function

Use the `iquery` function to directly run arbitrary SciDB AFL queries supplied as character strings or scidb objects, optionally returning results to R as a data frame.

```
iquery(db, "build(<v:double>[i=1:2,2,0, j=1:3,1,0], i*j)", return=TRUE)
```

```
  i j v
1 1 1 1
2 2 1 2
3 1 2 2
4 2 2 4
5 1 3 3
6 2 3 6
```

AFL operators and macros as R functions

A SciDB database connection object returned by `scidbconnect()` contains a list of available AFL operators and macros presented as R functions. The functions return scidb objects, and those objects may be used as function arguments to compose complex AFL query expressions.

```
db <- scidbconnect()
x <- db$build("<v:double>[i=1:2,2,0, j=1:3,1,0]", i * j)
y <- db$apply(x, a, 2 * v, b, "'a character string'")
print(y)
```

```
SciDB expression  apply(build(<v:double>[i=1:2,2,0, j...
SciDB schema  <v:double,a:double,b:string NOT NULL> [i=1:2:0:2; j=1:3:0:1]
  variable dimension            type nullable start end chunk
1        i      TRUE           int64    FALSE     1   0     2
2        j      TRUE           int64    FALSE     1   0     3
3        v     FALSE          double     TRUE
4        a     FALSE          double     TRUE
5        b     FALSE string NOT NULL    FALSE
```

Note that in the above expression `i` and `j` are evaluated as SciDB dimension values, not R values. The expression evaluation procedure is:

1. Arguments that evaluate to SciDB arrays are replaced by their corresponding SciDB expression.
2. Character strings are replaced by their unquoted value.
3. All remaining arguments are passed verbatim into the SciDB query expression.

SciDB schema literal values represent one important exception to the above process. SciDB schemas contain characters like `<`, `:`, `[` that correspond to R infix operators. It's difficult to always overload their evaluation as R operators. Thus, SciDB schemas must simply be supplied as R character strings instead of verbatim text as shown in the following example (also used above).

```
x <- db$build("<v:double>[i=1:2,2,0, j=1:3,1,0]", i * j)
```

Using scalar-valued R expressions within AFL statements

Use the special `R()` function syntax within any portion of an AFL statement to replace the enclosed R expression with its scalar value. The R expression is evaluated in the calling environment (the parent frame of the AFL function). The evaluated R expression must be a scalar character, numeric, or logical value. The next example uses two R values, one for the upper limit of the `j` dimension and another in the build statement.

```
jlim <- 3
z <- pi
x <- db$build("<v:double>[i=1:2,2,0, j=1:R(jlim),1,0]", i * j + R(z))
```

Help on AFL expressions

Use the TAB key for argument lists within an AFL function (RStudio only). The standard R `help()` function shows SciDB doxygen help for AFL function arguments, for instance

```
help(db$aggregate)
```

Alternatively, supply a character string argument naming a SciDB operator to the `aflhelp()` function to show SciDB help for the operator.

```
aflhelp(db, "aggregate")
```

SciDB aliasing in AFL expressions with %as%

Use the special `%as%` R infix operator in place of the usual AFL word "as" in SciDB expressions using aliasing, either to rename a SciDB array, expression, or certain AFL expression arguments often in aggregations.

```
db <- scidbconnect()
x <- as.scidb(db, iris)     # upload the iris data frame to SciDB
as.R(db$grouped_aggregate(x, Species, avg(Petal_Length) %as% avg))
```

```
  instance_id value_no    Species   avg
1           2        0     setosa 1.462
2           3        0 versicolor 4.260
3           4        0  virginica 5.552
```

Fall back to quoting AFL expressions verbatim

If you run into trouble using the AFL methods on a scidb connection object, remember that you can always simply supply a quoted AFL expression string. For instance, the last example above could have equivalently been written:

```
db <- scidbconnect()
x <- as.scidb(db, iris)     # upload the iris data frame to SciDB
iquery(db, paste("grouped_aggregate(", x@name, ", Species, avg(Petal_Length) as avg)"), return=TRUE)
```

Package options

The following package options can be set with R's `options` function. Set `options(scidb.debug=TRUE)` to see interesting debugging information including all the issued SciDB queries. Set `options(scidb.stream=TRUE)` to avoid creating server-side data files which is most useful when you're downloading lots of data from SciDB into R.

```
# Set the scidb.prefix option to prefix queries with arbitrary AFL
# statements (issued in the same connection context as the query).
# This is mostly useful for authentication-related options like set_role
# and in some cases set_namespace.
options(scidb.prefix=NULL)

# Default shim port and host.
options(scidb.default_shim_port=8080L)
options(scidb.default_shim_host="localhost")

# How to download arrays and their coordinates. Set scidb.unpack=FALSE
# to use apply, which can be faster in some cases when used with aio.
options(scidb.unpack=FALSE)

# Disable SSL certificate host name checking by default. This is important mostly
# for Amazon EC2 where hostnames rarely match their DNS names. If you enable this
# then the shim SSL certificate CN entry *must* match the server host name for the
# encrypted session to work. Set this TRUE for stronger security (help avoid MTM)
# in SSL connections.
options(scidb.verifyhost=FALSE)

# Enable or disable verbose debugging messages
options(scidb.debug=FALSE)

# List of special DDL operators
options(scidb.ddl=c("cancel",
                    "create_array",
                    "remove",
                    "rename",
                    "sync",
                    "add_instances",
                    "remove_instances",
                    "unregister_instances",
                    "create_namespace",
                    "drop_namespace",
```

```
"move_array_to_namespace",
"create_role",
"create_user",
"drop_namespace",
"drop_role",
"drop_user",
"add_user_to_role",
"drop_user_from_role",
"set_namespace",
"set_role",
"set_role_permissions",
"verity_user",
"create_with_residency")
```