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 provides two ways to interact with SciDB from R:

- Directly running SciDB queries using the native array functional language (AFL), optionally returning results as data frames
- Using the `scidb` class and related methods.

The SciDB array classes facilitate programming large-scale SciDB computation in R using natural R syntax. 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. Once a connection is established, connection information is maintained until a different connection is established or the R session ends.

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")
scidbconnect()
```

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

```
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.

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 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.

Use the `namespace` option with the `scidblist()` or `scidbls()` functions to list arrays in the specified namespace. Use the period character '.' as a path separator to prefix SciDB array names with their namespace location. The following example creates a namespace called 'cazart', uploads the `iris` data frame to an array named 'iris' in the 'cazart' namespace, and lists the arrays in that namespace.

```
iquery("create_namespace('cazart')")
scidbls(type="namespaces")
x = as.scidb(iris, name="cazart.iris")
scidbls(namespace="cazart")
```

You can remove arrays in namespaces (with sufficient privelege) by referring to the R object or fully qualified SciDB array name with path:

```
scidbrm(x, force=TRUE)

# or

scidbrm("cazart.iris", force=TRUE)
```

(we use the `force=TRUE` option because the name is protected from easy removal by the R package as it's not a temporary array name).

Listing and removing SciDB arrays

The `scidblist` function lists SciDB objects and features including aggregates, arrays, datastores, functions, instances, libraries, macros, namespaces, operators, queries, roles, types, and users, optionally showing detailed schema information for arrays. Returned results can be filtered using regular expression-style syntax.

The `scidbremove` function removes a SciDB array, or optionally a set of SciDB arrays. The function accepts a vector of array names, resulting in the removal of all the specified arrays. Combine this feature with the regular expression filtering output of `scidblist` to remove sets of arrays matching the filter. Array names not associated with a specific R session are protected from easy removal. Specify `force=TRUE` as indicated in the warning message to remove those arrays.

Running AFL queries

Use the `iquery` function to run arbitrary SciDB AFL queries, optionally returning results to R as data frames.

```
iquery("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
```

The scidb data frame-like class

The `scidb` function returns representations of SciDB arrays as data frame-like R objects with class `scidb`. The objects display the SciDB coordinate dimensions and attributes as columns of a data frame. The `scidb` function accepts arbitrary SciDB AFL expressions or named SciDB arrays.

```
x <- scidb("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 variable `x` is a sort of *SciDB array view*; an un-evaluated SciDB query expression. The value of `x` is evaluated by SciDB lazily when needed or when explicitly requested. One may force evaluation and return values from `x` to R in data frame form using the `iquery` function that we used above:

```
iquery(x, 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
```

Alternatively, use a special empty-indexing operator `[]` on any SciDB array object to return its values to R:

```
x[]
```

```
  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
```

Retrieve SciDB attribute values without corresponding SciDB array dimension coordinates using the `drop=TRUE` argument:

```
x[, drop=TRUE]
```

```
[1] 1 2 2 4 3 6
```

Use the `scidbeval` function to evaluate and materialize `scidb` views in SciDB, returning a new `scidb` R variable that points to the evaulated result in SciDB. The `scidbeval` function stores the evaluation result into new named SciDB arrays in the database, including optionally temporary arrays.

```
y <- scidbeval(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 by default tied to R's garbage collector. When the R variable's contents are garbage-collected by R, the associated SciDB array is removed:

```
yname <- y@name
scidbls(yname)
```

```
[1] "R_array1687f71a392851102980131950"
```

```
rm(y)
gc()
scidbls(yname)
```

```
character(0)
```

Note! that we use the `@name` slot of the `scidb` object directly here to interrogate the underlying SciDB expression associated with the R variable. See the help page for `scidbeval` for more on this topic, including disabling automatic garbage collection of SciDB arrays.

Data upload from R to SciDB

Upload local R variables into named SciDB arrays with the `as.scidb` function. It's not as efficient as native SciDB bulk-uploading options, but works well to transfer small amounts of data from R into SciDB.

Note that variable names may be changed to reflect naming conventions in SciDB.

Supported R objects for upload include data frames, matrices and vectors, and numeric sparse matrices from the Matrix package. **Note**! Data frames are *uploaded* to SciDB using a TSV-delimited text format but matrices and vectors are uploaded using a binary format. (All data are *downloaded* from SciDB to R in binary format by default.)

```
i <- as.scidb(iris)
print(i)
```

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

SciDB expression  R_array2cd623f34ac91470467539269804...
SciDB schema  <Sepal_Length:double,Sepal_Width:double,Petal_Length:double,
               Petal_Width:double,Species:string>
              [tuple_no=0:*,100000,0,dst_instance_id=0:7,1,0,
               src_instance_id=0:7,1,0]
         variable dimension   type nullable start end  chunk
1        tuple_no      TRUE  int64    FALSE     0   * 100000
2 dst_instance_id      TRUE  int64    FALSE     0   7      1
3 src_instance_id      TRUE  int64    FALSE     0   7      1
4    Sepal_Length     FALSE double     TRUE
5     Sepal_Width     FALSE double     TRUE
6    Petal_Length     FALSE double     TRUE
7     Petal_Width     FALSE double     TRUE
8         Species     FALSE string     TRUE
```

The dimension of the uploaded result shows eight *columns* (combined SciDB dimensions and attributes). The output of the upload process for data frames into SciDB uncludes a dimension without an explicit upper bound (the asterisk above), so the total number of elements is not known.

```
dim(i)
```

```
[1] Inf 8
```

The `count` function always obtains an explicit count of elements of a SciDB array:

```
count(i)
```

```
[1] 150
```

Data frames may be uploaded by one of several available SciDB upload plugins including the SciDB load_tools, prototype_load_tools, or aio_tools plugins depending on what the R package finds available. The dimensions of the resulting SciDB array may vary depending on the upload tool used (aio tools in the above example).

Type conversion is applied as required. Some SciDB types (various non-32 bit integer sizes) are not available in R. Some R types (complex, factors, etc.) are not available in SciDB. Missing values in R are translated to

SciDB NULL values and vice-versa. (SciDB has a notion of multiple NULL codes, but all codes are translated to R missing values.)

The above example shows that some data upload operators in SciDB return 2- or 3-d arrays with variable dimension names depending on the SciDB upload tool used. In order to simplify later examples, we force the uploaded iris array into a 1-d SciDB array with a single dimension named "row". The operations used here are described below.

```
i <- scidbeval(
       project(dimension_rename(unpack(as.scidb(iris)), new="row"),
               gsub("\\.", "_", names(iris))),
       temp=TRUE)
print(i)
```

```
SciDB expression  project(cast(unpack(R_array312a169b...
SciDB schema  <Sepal_Length:double,Sepal_Width:double,Petal_Length:double,
               Petal_Width:double,Species:string> [row=0:*,1000000,0]
      variable dimension   type nullable start end   chunk
1          row      TRUE  int64    FALSE     0   * 1000000
2 Sepal_Length     FALSE double     TRUE
3  Sepal_Width     FALSE double     TRUE
4 Petal_Length     FALSE double     TRUE
5  Petal_Width     FALSE double     TRUE
6      Species     FALSE string     TRUE
```

Important operations on scidb values

This section covers a few of the most important operations on scidb array values in R. With few exceptions, operations on SciDB array objects produce new SciDB array objects. Since objects are effectively SciDB array views, compositions of R operations build up increasingly complex SciDB queries.

Project and slice

Use project, or equivalently the data frame single column subset notation $ to project the array object onto a subset of its SciDB attributes. The bracket [can also perform column projection, for instance i[, c("Petal_Length", "Species")].

```
i <- as.scidb(iris)
project(i, "Species")
```

```
SciDB expression  project(R_array1687f572177bd1102980...
SciDB schema  <Species:string NULL DEFAULT null> [row=0:*,100000,0]
  variable dimension   type nullable start end chunk
1      row      TRUE  int64    FALSE     1  *  100000
2  Species     FALSE string     TRUE
```

```
i$Species   # or, equivalently, i[, "Species"]
```

```
SciDB expression  project(project(cast(unpack(R_array...
SciDB schema  <Species:string> [row=0:*,1000000,0]
```

```
  variable dimension   type nullable start end   chunk
1      row      TRUE  int64    FALSE     0   * 1000000
2  Species     FALSE string     TRUE
```

Use the `slice` function to slice out one or more dimensions of a SciDB array at specific coordinate values.

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

```
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
```

```
slice(x, "j", 2)
```

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

Subset

Filter SciDB array values with R's `subset` function in one of two forms:

- Explicit AFL filter expressions
- Implicit queries generated from R expressions

The implicit form can sometimes generate more efficient queries and allows you to mix simple R scalar values in the SciDB expression. However, the implicit form uses non-standard R evaluation which can be confusing to use in some cases and might not be well-suited for non-interactive use; see for example the warnings in the documentation for the base R `transform` function. The following examples illustrate each form.

1. Explicit AFL filter expression (note the quoted AFL expression)

```
y <- subset(i, "Species = 'setosa' and row > 40")
y@name
```

```
[1] "filter(R_array1687f572177bd1102980, Species = 'setosa' and row > 40)"
```

2. Implicit query from R expression (note the unquoted R expression and use of R scalar variable)

```
rownum <- 40
z <- subset(i, Species == 'setosa' & row > rownum)
z@name
```

```
[1] "filter(between(R_array1687f572177bd1102980, 41, null), Species = 'setosa' )"
```

See `?subset.scidb` for more details.

Transform

R's `transform` function acts as a wrapper for the SciDB "apply" operator. With it, you can add new attributes (variables) to SciDB arrays or replace existing ones with new values.

The syntax of `transform` is

```
transform(array, new_value1="quoted AFL expression 1",
                 new_value2="quoted AFL expr 2", ...)
```

Let's try it:

```
i <- scidbeval(
       project(dimension_rename(unpack(as.scidb(iris)), new="row"),
               gsub("\\.", "_", names(iris))),
       temp=TRUE)
x <- transform(i,
       Species_index="iif(Species='virginica', 1, iif(Species='versicolor', 2, 3))")
head(x)
```

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

See `?transform.scidb` for many more details.

Redimension

Aggregations and other operations are most efficient along SciDB coordinate axes. The SciDB "redimension" operator is used to set those up from among existing dimension and attribute values in a SciDB array or array expression.

Let's illustrate by making the "Species_index" attribute in the last example a dimension, repeating the steps in the last example for completeness.

```
i <- scidbeval(
       project(dimension_rename(unpack(as.scidb(iris)), new="row"),
               gsub("\\.", "_", names(iris))),
       temp=TRUE)
x <- transform(i,
       Species_index="iif(Species='virginica', 1, iif(Species='versicolor', 2, 3))")
y <- redimension(x, dim=c("row", "Species_index"))
print(y)
```

```
SciDB expression  redimension(apply(R_array312a65ae74...
SciDB schema  <Sepal_Length:double,Sepal_Width:double,Petal_Length:double,
               Petal_Width:double,Species:string>
               [row=0:*,1000000,0,Species_index=0:*,1,0]
```

```
       variable dimension   type nullable start end   chunk
1           row      TRUE  int64    FALSE     0   * 1000000
2 Species_index      TRUE  int64    FALSE     0   *       1
3  Sepal_Length     FALSE double     TRUE
4   Sepal_Width     FALSE double     TRUE
5  Petal_Length     FALSE double     TRUE
6   Petal_Width     FALSE double     TRUE
7       Species     FALSE string     TRUE
```

Aggregate

SciDB can very rapidly aggregate array data along the array coordinate system. R's standard `aggregate` function is overloaded for `scidb` array objects with some SciDB-specific extensions.

Let's extend the last few examples to compute the average petal length grouped by Species_index. We include the original "Species" variable to help identify the index. We use the special empty indexing operator to return the result to R.

```
i <- scidbeval(
       project(dimension_rename(unpack(as.scidb(iris)), new="row"),
               gsub("\\.", "_", names(iris))),
       temp=TRUE)
x <- transform(i,
       Species_index="iif(Species='virginica', 1, iif(Species='versicolor', 2, 3))")
y <- redimension(x, dim=c("row", "Species_index"))
aggregate(y, by="Species_index", FUN="avg(Petal_Length), max(Species) as Species")[]
```

```
  Species_index Petal_Length_avg    Species
1             1            5.552  virginica
2             2            4.260 versicolor
3             3            1.462     setosa
```

SciDB users equipped with the grouped aggregate plugin available from https://github.com/Paradigm4/grouped_aggregate can also directly aggregate by attributes and/or SciDB dimensions.

```
aggregate(i, by="Species", FUN="avg(Petal_Length)")[]
```

```
  instance_id value_no    Species Petal_Length_avg
1           2        0 versicolor            4.260
2           2        1     setosa            1.462
3           7        0  virginica            5.552
```

Both of the above queries are equivalent to the R aggregation:

```
aggregate(iris$Petal.Length, by=list(iris$Species), FUN=mean)
```

```
##      Group.1     x
## 1     setosa 1.462
## 2 versicolor 4.260
## 3  virginica 5.552
```

SciDB's version of `aggregate` includes many extras like windowed aggregates. SciDB cumulative aggregation is supported by the R `cumulate` function. See `?regrid` and `?xgrid` for help with high-performance SciDB coordinate-based aggregation and interpolation (to perform fast spatial or temporal decimation, for instance). See `?image` for a matrix image display function that uses `regrid` to visualize potentially very large sparse or dense matrices.

See `?'aggregate,scidb-method'` for help and more examples.

Merge and join

R's `merge` function implements SciDB `join`, `cross_join`, `equi_join` and `merge` operators. The following example illustrates the difference between join and merge.

```
i <- scidbeval(
       project(dimension_rename(unpack(as.scidb(iris)), new="row"),
               gsub("\\.", "_", names(iris))),
       temp=TRUE)
a <- i$Petal_Length
b <- i$Petal_Width
merge(a, b)                # SciDB (inner) join
```

```
SciDB expression  join(project(R_array312a4ca70b21146...
SciDB schema  <Petal_Length:double,Petal_Width:double> [row=0:*,1000000,0]
      variable dimension   type nullable start end   chunk
1          row      TRUE  int64    FALSE     0   * 1000000
2 Petal_Length     FALSE double     TRUE
3  Petal_Width     FALSE double     TRUE
```

```
merge(a, b, merge=TRUE)   # SciDB "merge"
```

```
SciDB expression  merge(project(R_array312a4ca70b2114...
SciDB schema  <Petal_Length:double> [row=0:*,1000000,0]
      variable dimension   type nullable start end   chunk
1          row      TRUE  int64    FALSE     0   * 1000000
2 Petal_Length     FALSE double     TRUE
```

Here is an example of a SciDB "cross join" along one common axis between two arrays with different dimensions.

```
a <- i$Petal_Length
x <- transform(i,
       Species_index="iif(Species='virginica', 1, iif(Species='versicolor', 2, 3))")
y <- redimension(x, dim=c("row", "Species_index"))$Petal_Width
print(a)
```

```
SciDB expression  project(R_array312a4ca70b2114695912...
SciDB schema  <Petal_Length:double> [row=0:*,1000000,0]
      variable dimension   type nullable start end   chunk
1          row      TRUE  int64    FALSE     0   * 1000000
2 Petal_Length     FALSE double     TRUE
```

```
print(y)
```

```
SciDB expression  project(redimension(apply(R_array31...
SciDB schema  <Petal_Width:double> [row=0:*,1000000,0,Species_index=0:*,1,0]
       variable dimension   type nullable start end   chunk
1           row      TRUE  int64    FALSE     0   * 1000000
2 Species_index      TRUE  int64    FALSE     0   *       1
3   Petal_Width     FALSE double     TRUE
```

```
merge(a, y)
```

```
SciDB expression  cross_join(project(redimension(appl...
SciDB schema  <Petal_Width:double,Petal_Length:double>
              [row=0:*,1000000,0,Species_index=0:*,1,0]
       variable dimension   type nullable start end   chunk
1           row      TRUE  int64    FALSE     0   * 1000000
2 Species_index      TRUE  int64    FALSE     0   *       1
3   Petal_Width     FALSE double     TRUE
4  Petal_Length     FALSE double     TRUE
```

Many additional options are available, see `?merge.scidb` for examples.

Antijon

Use the `antijoin` function to return a logical-valued mask of array coordinates that *do not* join between two arrays with the same coordinate schema.

Other SciDB operations

The `scidb` package defines many additional SciDB-related functions besides the most important few outlined above.

SciDB schema-related functions

The `cast` function works like the SciDB "cast" operator; use it to rename any part of a SciDB schema, or use the slightly more convenient `dimension_rename` and `attribute_rename` functions.

The following functions return scalar or vector values to R (not SciDB array objects). The `dimensions` and `scidb_attribute` functions return the names of the SciDB array dimension and attribute variables, respectively. The `scidb_nullable`, `scidb_types`, `scidb_coordinate_bounds`, and `schema` functions return information about SciDB attributes and array schema.

SciDB schema/shape-altering operations

We've already seen the very important `redimension` function. Use the `repart` function similarly to the SciDB "repart" operator to alter the SciDB array coordinate partitions. Use the `reshape` function to change the dimensionality of a SciDB array, and similarly the `unpack` function to flatten a SciDB array into a table.

The `subarray` function is a wrapper for the SciDB "subarray" operator that filters an array along its coordinates and resets the top-left coordinate to the origin.

Filling in missing values

Use the `replaceNA` function to replace missing values in SciDB arrays with a constant, similarly to the SciDB "substitute" operator (the name "substitute" already had a substantially different meaning in R, so we picked a different function name).

SciDB catalog operations

Use `scidbremove` or, equivalently, `scidbrm` to remove SciDB arrays from the database (permanently!). Related to this topic, use `scidbeval` to control the connection of R's garbage collector to SciDB arrays associated with R variables. The `persist` function can be used to recursively modify the garbage-collection association for any R `scidb` array object, including all its dependencies.

The `scidbls` function can be used to list arrays and many other SciDB database objects.

Use the `remove_old_versions` function to remove all the old versions in the SciDB catalog of a named array, leaving only the last version.

Use `rename` to rename a stored SciDB array in the database.

The `count` function counts the number of elements in the specified SciDB array or expression just like the SciDB "op_count" operator/macro.

Math operations

The scidb package defines a number of mathematical functions that operate on SciDB arrays.

- `gemm` Compute a dense matrix product of two dense SciDB matrices
- `gesvd` Compute the full singular value decomposition of a dense SciDB matrix
- `cov,cor` Covariance and correlation matrices
- `glm,glm.fit,predict` Compute various generalized linear models similarly to R's `glm` function see `?"glm,ANY,ANY,scidb-method"` for help and examples of generalized linear models
- `t` Form a SciDB matrix transpose
- `image` Optionally decimate, dowload and display a 2-d matrix image representation of a dense or sparse SciDB matrix
- `phyper,qhyper` Distribution and quantile functions for the hypergeometric distribution
- `scidb_fisher.test` Fisher's exact test for count data including conditional odds estimate computed by maximum likelihood

See online R help for the above functions for details and examples. Other mathematical operations are available through direct use of SciDB's native AFL query language.

Package options

The following package options can be set with R's `options` function. The most interesting ones are the last two! 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)

# The scidb.version option is set during scidbconnect(). However, users
# may carefully override it to enable certain bug fixes specific to older
# versions of SciDB.
  options(scidb.version=15.7)

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

# Make it harder to remove arrays. When this option is TRUE, users
# have to specify scidbrm(array, force=TRUE) to remove arrays that do not
# begin with "R_array".
  options(scidb.safe_remove=TRUE)

# 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)

# Set to TRUE to enable debugging that shows SciDB queries as they run.
  options(scidb.debug=FALSE)
```

Special note on the `curl` package dependency

Newer versions of the `scidb` package depend on Jeroen Ooms' `curl` package, replacing the older `RCurl` dependency and fixing a number of memory, performance and stability issues.

As of this writing, the `curl` package has one significant problem. Users are not able to cancel HTTP requests until data transfers begin. In the context of SciDB interaction, that means that users can't cancel a SciDB query until data starts flowing back to R.

There exists an experimental version of the `curl` package that does add this capability, but at the cost of some additional CPU overhead. If you want to try that out, you can install it directly from GitHub using the `devtools` package with:

```
devtools::install_github("jeroenooms/curl", ref="interrupt")
```