

The SciDB package for R



Contents

1	Introduction	4
2	Connecting to SciDB and Running Queries	4
2.1	Connecting to SciDB	4
2.2	Listing and removing SciDB arrays	5
2.3	Running SciDB queries	5
2.4	Iterating over query results	6
3	SciDB Arrays for R users	7
3.1	The <code>scidbdf</code> data frame-like class	8
3.2	The <code>scidb</code> array class	8
3.3	Arithmetic operations	10
4	Subsetting and indexing SciDB objects	12
5	Data manipulation functions	14
5.1	Apply, sweep and other array functions	14
5.2	Conditional subset operations	15
5.3	Database joins	16
5.4	Aggregation	17
5.5	Binding new variables to an array	18
5.6	Sorting and enumerating factors	18
6	Lazy evaluation, array promises, and garbage collection	19
7	SciDB-specific functions	21

8	Error handling	22
9	Function and method list	22
10	Package installation	23
10.1	Installing the R package from sources on GitHub	23
10.2	Installing the R package from CRAN	23
10.3	Installing the simple HTTP service for SciDB	24
10.3.1	Installation on RHEL/CentOS 6	24
10.3.2	Installation on Ubuntu 12.04	24
10.4	Package options, miscellaneous notes, and software license	25

1 Introduction

SciDB is an open-source database that organizes data in n -dimensional arrays. SciDB features include ACID transactions, parallel processing, distributed storage, efficient sparse array storage, and native linear algebra operations. The `scidb` package for R provides two ways to interact with SciDB from R:

1. By running SciDB queries from R, optionally transferring data through data frames or data frame iterators.
2. Through SciDB array classes for R. The arrays mimic standard R arrays and data frames, but operations on them are performed by the SciDB engine. Data are materialized to R only when requested.

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.

2 Connecting to SciDB and Running Queries

This section outlines the most basic interaction between R and SciDB: running queries and transferring one-dimensional SciDB arrays between R and SciDB through R data frames.

2.1 Connecting to SciDB

The `scidbconnect` function establishes a connection to a simple HTTP network service called shim running on a SciDB coordinator instance (see Section 10.3). 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 user authentication. Users are defined by the operating system users on the SciDB coordinator instance. The shim service can be configured to support either open/unencrypted or encrypted/authenticated ports, or both.

Connect to localhost by default on unencrypted port 8080:

```
> library("scidb")
> scidbconnect()
```

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

```
> scidbconnect(host="localhost", port=8083,
+             username="scidbuser", password="test")
```

We recommend using only encrypted/authenticated sessions when communicating with SciDB over public networks.

2.2 Listing and removing SciDB arrays

The `scidblist` function lists SciDB objects (arrays, instances, etc.), optionally showing detailed schema information for arrays. Returned results may be filtered using regular expression-style syntax.

The `scidbremove` function removes a SciDB array, or optionally a set of SciDB arrays defined by regular expression. 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.

2.3 Running SciDB queries

The `iquery` function executes SciDB queries using either the SciDB array functional language (AFL) or declarative array query language (AQL) syntax. When AFL is used, the `iquery` function optionally returns query results in an R data frame if the argument `return=TRUE` is specified. Returned output is similar to output obtained by the SciDB `iquery` command-line program with the `-olcsv+` option. The `iquery` function does not return anything by default.

Query results returned by the `iquery` function are internally presented to R using a generic CSV format, providing very flexible support for many data types. (The n -dimensional array class described in the next section uses a binary data exchange method between R and SciDB.) Note that, although R and SciDB have a number of common data types, each system contains types not supported by the other. Thus, conversion errors may arise. The `iquery` function is designed to reasonably minimize such issues and simplify basic data transfer between the systems. Data types common to

R and SciDB include double-precision numeric, character string, logical, and 32-bit integers. The `iquery` function supports standard R `read.table` parameter options to facilitate type conversion.

The following code example illustrates basic use of `iquery`.

```
> # Manually build a 1-d SciDB array named "P:"
> iquery("store(build(<x:double>[i=0:99,100,0],asin(1)*i/25),P)")
>
> # Return a query as a data.frame, manually specifying returned column classes:
> S <- iquery("apply(P,y,sin(x))",return=TRUE,
+           colClasses=c("integer","double","complex"))
> head(S, n=3)
```

```
  i      x      y
1 0 0.00000000 0.00000000+0i
2 1 0.06283185 0.06279052+0i
3 2 0.12566371 0.12533323+0i
```

2.4 Iterating over query results

The `iquery` function returns query results into a single R data frame by default. Large results expected to contain lots of rows may be iterated over by setting the `iterative=TRUE` argument. When `iquery` is used with the `iterative=TRUE` setting, it returns an iterator that iterates over chunks of rows of the result data frame. Iterators are defined by the `iterators` package. Their data may be directly accessed with the `nextElem` method, or indirectly with `foreach`. See the `iterators` and `foreach` packages for many examples and further documentation of their use.

```
> # Build a small 1-d SciDB test array:
> iquery("store(build(<x:double>[i=1:10,10,0],i/10.0),A)")
>
> # Return the result of a SciDB apply operator in an R iterator with a
> # block size of at most 7 rows at a time:
> it <- iquery("apply(A,y,sqrt(x))", return=TRUE, iterative=TRUE, n=7)
> nextElem(it)
```

```
  i  x      y
1 1 0.1 0.3162278
```

```
2 2 0.2 0.4472136
3 3 0.3 0.5477226
4 4 0.4 0.6324555
5 5 0.5 0.7071068
6 6 0.6 0.7745967
```

```
> nextElem(it)
```

```
      i      x      y
1  7 0.7 0.8366600
2  8 0.8 0.8944272
3  9 0.9 0.9486833
4 10 1.0 1.0000000
```

3 SciDB Arrays for R users

Data are organized by SciDB in n -dimensional sparse arrays. “Sparse” in SciDB arrays means that array elements may be left undefined, and such array elements are omitted from computations. Note that this interpretation of sparse differs in a subtle way from that used by sparse matrices defined by R’s Matrix package (whose sparse matrix elements are implicitly zero).

The elements of a SciDB array, called *cells*, contain one or more *attributes* (similar to R variables). The number and data types of attributes are uniform across all cells in an array, and SciDB stores data for each attribute separately. Thus, a one-dimensional SciDB array is conceptually very similar to a data frame in R: the SciDB dimension index corresponds to data frame row index, and SciDB attributes to data frame columns. Higher-dimensional arrays in SciDB don’t correspond directly to objects in R; the `scidb` n -dimensional array class described below is limited to working with one attribute at a time to more closely emulate standard R arrays.

The integer coordinate systems used to index SciDB arrays are similar to R, except that SciDB integer indices may be zero or negative, and are represented by 62-bit signed integers (R indices use either unsigned positive 31-bit integer or positive 52-bit integer-valued double values). This means that SciDB arrays can’t always be indexed by R. In practice, R users should avoid creating SciDB arrays with coordinate systems that extend past R’s limits.

SciDB attribute values within a cell may be explicitly marked missing, indicated by a special SciDB missing code also referred to as a NULL code. SciDB internally supports a large number of possible missing codes. All SciDB missing code values are mapped to NA values in R.

The `scidb` package defines two array classes for R with data backed by SciDB arrays: the `scidbdf` class for data frame-like objects, and the `scidb` class for matrix and general array objects.

3.1 The `scidbdf` data frame-like class

The `scidbdf` class defines a data frame-like class backed by one-dimensional SciDB arrays. Like data frames, the columns represent variables of distinct types and the rows represent observations. Each attribute in the backing SciDB array represents a column in the `scidbdf` object. The `scidbdf` object elements are read-only (the backing SciDB array may be manually updated, for example using the `iquery` function).

Use the `as.scidb` function to create new SciDB arrays and corresponding `scidbdf` R objects by copying R data frames into SciDB. The `types` and `nullable` options may be used to explicitly specify the SciDB type and nullability values of each data frame column. See the R help page for `df2scidb` for more information.

The `scidb` function returns an R `scidbdf` or `scidb` object representation of an existing SciDB array.

Objects of class `scidbdf` obey a subset of R indexing operations. Columns may be selected by numeric positional index, variable name, or with the shorthand dollar sign notation similarly to standard data frames. Rows may be selected by numeric ranges—see the following section on array indexing for details.

Subsets of `scidbdf` objects are returned as new `scidbdf` objects of the appropriate size (dimension, number of attributes/columns). The package uses the special empty-bracket notation, `[]`, to indicate that data should be materialized to R as an R data frame. Illustrations are provided in the examples.

3.2 The `scidb` array class

Similarly to the data frame-like class, the package defines the `scidb` array class for R that represents vectors, matrices and general n -dimensional arrays. Array objects defined by the `scidb` class behave in some ways like standard R arrays. But their data reside in SciDB and most operations on them are computed by SciDB.

The `scidb` array class supports working with a single array attribute at a time to conform to R arrays (which generally support a single value per cell). Consider the iris data, represented as a data frame within SciDB as a 1-d array with five attributes. The next example illustrates creating


```
> # Copy the Michelson-Morley experiment data to SciDB,
> # returning a scidbdf object
> X <- as.scidb(morley, name="morely")
> str(X)
```

SciDB array name: morely
 SciDB array schema: array<Expt:int32,Run:int32,Speed:int32> [row=1:100,100,0]
 Attributes:

	attribute	type	nullable
1	Expt	int32	FALSE
2	Run	int32	FALSE
3	Speed	int32	FALSE

Row dimension:

	name	type	start	length	chunk_interval	chunk_overlap	low	high
1	row	int64	1	100	100	0	NA	NA

```
> # Materialize the first four rows of X and the "Run" and "Speed"
> # columns to R (using [] to return results to R):
> X[1:4, c("Run","Speed")][]
```

	Run	Speed
1	1	850
2	2	740
3	3	900
4	4	1070

Example of a scidbdf object

a `scidbdf` data frame-like object in R referring to the iris data and also a 1-d `scidb` array object in R referring to the same data using the `Petal_Width` attribute. Note that, unlike R, SciDB array origin indices are arbitrary and may be zero or negative (matrices and vectors in SciDB are generally zero-indexed). Data from `scidb` array objects are not materialized to R until extracted with the empty indexing function, `[]`. Additional notes and examples about indexing appear in the next section.

```
> # Upload the iris data to SciDB storing it in an array named `iris`
> # and create a data frame-like object `df` that refers to it:
```

```
> df <- as.scidb(iris,name="iris")
> dim(df)

[1] 150    5

> # Now create a vector-like object using the same data in SciDB, but
> # referring to only one of the available variables (Petal_Width):
> x <- scidb("iris", attribute="Petal_Width", data.frame=FALSE)
> length(x)

[1] 150

> x[99:103][]

[1] 1.1 1.3 2.5 1.9 2.1

> # List all available attributes in the SciDB array:
> x@attributes

[1] "Sepal_Length" "Sepal_Width"  "Petal_Length" "Petal_Width"  "Species"
```

3.3 Arithmetic operations

The `scidb` array class supports a few standard linear algebra operations for dense and sparse matrices and vectors.

The example below shows a dense matrix example that compares matrix arithmetic in R and SciDB. Like previous data frame examples, use `as.scidb` to export R matrices and vectors to SciDB arrays.

We generally use `as.scidb` for convenience—it's far from the most efficient way to import data into SciDB. For very large data, use the SciDB bulk data load utility as outlined in the SciDB documentation instead.

```
> v <- as.scidb(matrix(rnorm(25),5))
> str(v)
```

```

SciDB array name:  R_array3ec3b08cf9a11...
SciDB array schema:  array<val:double> [i=0:4,5,0,j=0:4,5,0]
attribute in use:  val
All attributes:  val
Array dimensions:
  name  type start length chunk_interval chunk_overlap low high
1    i int64    0     5             5             0  NA  NA
2    j int64    0     5             5             0  NA  NA

```

```
> crossprod( v[] )           # Compute  $t(v) \% \% v$  using R
```

```

      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 10.438874  2.27018567 -7.1883979  1.04113352  4.274616
[2,]  2.270186  5.49588733 -0.6948195  0.05329931 -5.851780
[3,] -7.188398 -0.69481952  7.5489252 -2.70527379 -4.764550
[4,]  1.041134  0.05329931 -2.7052738  2.58124882  2.002071
[5,]  4.274616 -5.85177991 -4.7645497  2.00207129 12.071290

```

```

> # Now compute using SciDB, and materialize the result to R:
> crossprod(v)[]

```

```

      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 10.438874  2.27018567 -7.1883979  1.04113352  4.274616
[2,]  2.270186  5.49588733 -0.6948195  0.05329931 -5.851780
[3,] -7.188398 -0.69481952  7.5489252 -2.70527379 -4.764550
[4,]  1.041134  0.05329931 -2.7052738  2.58124882  2.002071
[5,]  4.274616 -5.85177991 -4.7645497  2.00207129 12.071290

```

Basic matrix/vector arithmetic operations on SciDB arrays (addition, subtraction, matrix and matrix vector products, scalar products, `crossprod` and `tcrossprod`) use standard R syntax. You can mix R and SciDB matrices and vectors and the `scidb` package will try to do the right thing by assigning R data to temporary SciDB arrays conforming to required database schema. The next example shows an example of computations that mix `scidb` array objects with R vectors.

Expression	Operation	Operands	Output
$A \%*\% B$	Matrix multiplication	A, B Conformable SciDB arrays or R matrices/vectors	SciDB array
$A \pm B$	Matrix summation/difference	A, B SciDB arrays or R matrices/vectors	SciDB array
<code>crossprod(A,B)</code>	Cross product $t(A) \%*\% B$	A, B SciDB arrays or R matrices/vectors	SciDB array
<code>tcrossprod(A,B)</code>	Cross product $A \%*\% t(B)$	A, B SciDB arrays or R matrices/vectors	SciDB array
$A */ B$	Elementwise product/quotient	A, B Conformable SciDB arrays or R matrices/vectors	SciDB array
$\alpha */ A$	Scalar multiplication/division	SciDB array A , scalar α	SciDB array
<code>t(A)</code>	Transpose	SciDB array A	SciDB array
<code>sin(A)</code>	Sine function, also other trig functions	SciDB array A	SciDB array
<code>log(A, base)</code>	Logarithm function	SciDB array A , numeric base	SciDB array
<code>diff(A, lag = 1)</code>	Finite differences	SciDB array A , integer lag	SciDB array
<code>A[range, range, ...]</code>	Subarray	SciDB array A	SciDB Array
<code>A[]</code>	Materialize	SciDB array A	R array
<code>diag(A)</code>	Matrix diagonal	SciDB array or vector A	SciDB arrays
<code>svd(A)</code>	Singular value decomposition	Dense SciDB array A	SciDB arrays
<code>svd(A, nu)</code>	Truncated SVD	Sparse or dense SciDB array A	SciDB arrays

Some `scidb` math operations

4 Subsetting and indexing SciDB objects

The `scidb` and `scidbdf` classes generally follow SciDB database indexing convention, which exhibits some differences with standard R indexing. In particular, note that the starting SciDB integer index is arbitrary, but often zero. The upper-left corner of R arrays is always indexed by `[1,1,...]`. Subarray indexing operations use the SciDB convention. Thus, zero and negative indices are literally interpreted and passed to SciDB. In particular, negative indices do not indicate index omission, unlike standard R arrays.

Important indexing notes include:

- Use empty brackets, `[]`, to materialize data back to R. Otherwise, indexing operations produce new SciDB array objects.
- Use numeric indices in any dimension in the units of the underlying SciDB array coordinate system. Note that SciDB arrays generally are zero-indexed and may even have negative

indices. SciDB data frame-like objects generally use 1-based indexing. When in doubt about the base index, use the `str` function to interrogate the SciDB object for details.

- Numeric indexing may include contiguous ranges or vectors of distinct coordinate values, but repeated coordinate values in a single dimension are not allowed. Examples of valid index ranges include `[1:4, c(3,1,5), -10:15]`, but not `[c(1,3,1)]`.
- The `scidbdf` class represents 1-d SciDB arrays as data frame objects with SciDB array attribute as columns. Use either positional numeric or name-based indexing along columns, either with the dollar-sign notation or string indexing.
- The `scidb` supports labeled dimension indexing using R `rownames`, `colnames`, or `dimnames` settings. Labels assigned in this way must be provided by 1-d SciDB arrays that map the integer coordinates to character label values. See the examples below.
- The `scidb` class supports indexing by other SciDB arrays to achieve the effect of filtering by boolean expressions and similar operations, also illustrated below in the examples.
- Use the utility `between` function to avoid forming large sequences representing huge indexing ranges. For example, use `[between(1,1e9)]` instead of `[1:1e9]`.
- The `diag` function is supported to extract a vector of diagonal elements from a matrix, or to create a sparse diagonal matrix from a vector. Examples of indexing operations follow.

```
> set.seed(1)
> X <- as.scidb( matrix(rnorm(20),nrow=5) )
>
> # SciDB matrix objects like X default to zero-based indexing. It's important
> # that the label array have the same starting index:
> rownames(X) <- as.scidb( data.frame(letters[1:5]), start=0)
> X[c("b","a","d"), ]
```

A reference to a 3x4 SciDB array

Labeled indices

```
> # Determine a subset of the rownames of X (another SciDB array):  
> idx <- rownames(X) > "'b'"  
> # Subset X by the condition on its rows:  
> X[idx, ]
```

A reference to a 3x4 SciDB array

Indexing by a condition on an auxiliary array

5 Data manipulation functions

The package defines a number of common SciDB data manipulation functions for SciDB array objects.

5.1 Apply, sweep and other array functions

R's `apply` function applies a function along margins of an array or matrix. A version of `apply` limited to SciDB aggregation functions is available for `scidb` and `scidbdf` objects. The SciDB aggregation function is supplied as a string and references the array attributes. The next example uses `apply` together with `sweep` to center the columns of a matrix. More

```
> # Create an example matrix and upload to SciDB. Its single numeric array  
> # attribute will be called "val" by default.  
> A <- as.scidb( matrix(rnorm(100), nrow=10) )  
>  
> # Create a centered version of the matrix by computing its column means with  
> # apply and subtracting them with sweep. Note that apply uses a SciDB  
> # expression. We specify `eval=TRUE` below to compute and cache the result of  
> # sweep, improving the efficiency of the subsequent crossprod operation.  
> A0 <- sweep(A, 2, apply(A, 2, "avg(val)"), eval=TRUE)  
>  
> # The covariance matrix of A:  
> crossprod(A0)/(nrow(A) - 1)
```

A reference to a 10x10 SciDB array

Compute a covariance matrix

general aggregation and data manipulation functions are outlined in the next section.

The `diff` function works similarly to R's usual one for computing finite differences along vectors and matrices.

Use the `cumulate` function to compute running operations along data, for example cumulative sums. The operation to be performed must be a valid SciDB aggregation function expressed as a character string. Here is a simple example:

```
> x <- as.scidb(iris)
> y <- cumulate(x, "sum(Petal_Width)")
> print(head(y, n=4))
```

	row	Petal_Width_sum
1	1	0.2
2	2	0.4
3	3	0.6
4	4	0.8

The `count` function applied to a `scidb` array object returns the count of non-empty cells in the backing SciDB array.

The `image` function displays a heatmap of a regrid of a 2-d `scidb` array object, and returns the regridded array to R. The `grid=c(m,n)` function parameter specifies the regrid window sizes in each array dimension, and defaults to the array chunk sizes. The regrid aggregation function may be specified using the `op` function argument, and by default averages the array values over the regrid windows.

5.2 Conditional subset operations

Use the `subset` function to filter array contents by a boolean expression somewhat similarly to the standard R `subset` function. Under the hood, this function uses the SciDB `filter` operator—the function name `subset` more closely matches standard R syntax.

The `subset` function requires two arguments, a SciDB array reference and a valid SciDB logical expression represented as a string. Here is a simple example:

```
> df <- as.scidb(iris)
> subset(df, "Petal_Width > 2.4")[]
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
101	6.3	3.3	6.0	2.5	virginica
110	7.2	3.6	6.1	2.5	virginica
145	6.7	3.3	5.7	2.5	virginica

The `subset` function applied to arrays returns a sparse SciDB array with entries not meeting the filter condition omitted. This behavior differs from the standard R `subset` function, which can only operate on rows and columns. The `scidb` class also supports a short-hand notation for `subset` for infix comparison operations.

```
> # Upload a 5x5 matrix to SciDB. Its single numeric attribute name defaults to
> # 'val' (see help for as.scidb):
> set.seed(1)
> A <- as.scidb( matrix(rnorm(9),nrow=3) )
> # Apply a filter to the SciDB array and return the results to R:
> subset(A, "val>0") []
```

3 x 3 sparse Matrix of class "dgCMatrix"

```
[1,] .          1.5952808 0.4874291
[2,] 0.1836433 0.3295078 0.7383247
[3,] .          .          0.5757814
```

```
> # Alternatively, achieve the same filter with an infix comparison:
> (A > 0) []
```

3 x 3 sparse Matrix of class "dgCMatrix"

```
[1,] .          1.5952808 0.4874291
[2,] 0.1836433 0.3295078 0.7383247
[3,] .          .          0.5757814
```

Example of array value filtering with `subset`

5.3 Database joins

The package supports a limited form of the R `merge` function, enabling a number of database join-like operations on SciDB array objects. See `help("merge", package="scidb")` function for detailed help. Here is an example that performs an inner join on array attributes:


```
> authors <- data.frame(
+   surname = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil"),
+   nationality = c("US", "Australia", "US", "UK", "Australia"),
+   deceased = c("yes", rep("no", 4)),
+   stringsAsFactors=FALSE)
> books <- data.frame(
+   name = c("Tukey", "Venables", "Tierney",
+           "Ripley", "Ripley", "McNeil", "R Core"),
+   title = c("Exploratory Data Analysis",
+            "Modern Applied Statistics ...",
+            "LISP-STAT", "Spatial Statistics", "Stochastic Simulation",
+            "Interactive Data Analysis", "An Introduction to R"),
+   other.author = c(NA, "Ripley", NA, NA, NA, NA, "Venables & Smith"),
+   stringsAsFactors=FALSE)
> a <- as.scidb(authors)
> b <- as.scidb(books)
> merge(a,b,by.x="surname",by.y="name")[,4:8][]
```

	nationality	deceased	name	title	other_author
0	Australia	no	McNeil	Interactive Data Analysis	<NA>
1	UK	no	Ripley	Spatial Statistics	<NA>
2	UK	no	Ripley	Stochastic Simulation	<NA>
3	US	no	Tierney	LISP-STAT	<NA>
4	US	yes	Tukey	Exploratory Data Analysis	<NA>
5	Australia	no	Venables	Modern Applied Statistics ...	Ripley

```
> # cf. The standard R data.frame merge function:
> # merge(authors,books,by.x="surname",by.y="name")
```

The `merge` implementation has some limitations outlined in its man page. Joins on attributes (instead of along coordinate axes) are presently the most limited cases, and most joins are limited to inner joins at the moment.

5.4 Aggregation

The `aggregate` function is overloaded to perform aggregation operations on SciDB array objects. Aggregation may be defined over array dimensions and/or array attributes very

similarly to standard R aggregation syntax, except that aggregation functions must be valid SciDB aggregate expressions, represented as strings.

```
> # Upload the iris data to SciDB and create a data frame-like object
> # that refers to it:
> df <- as.scidb(iris)
> aggregate(df, by="Species", FUN="avg(Petal_Length), stdev(Petal_Width)")[]
```

	Species_index	Petal_Length_avg	Petal_Width_stdev	Species
0	0	1.462	0.1053856	setosa
1	1	4.260	0.1977527	versicolor
2	2	5.552	0.2746501	virginica

5.5 Binding new variables to an array

The SciDB package defines the `bind` function to add variables to arrays similar to the R `cbind` function for data frames. However, `bind` can also operate on higher-dimensional arrays. The example below adds a variable named ‘`prod`’ to the SciDB array `df` defined in the last example.

```
> y <- bind(df, "prod", "Petal_Length * Petal_Width")
> head(y, n=3)
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species	prod
1	5.1	3.5	1.4	0.2	setosa	0.28
2	4.9	3.0	1.4	0.2	setosa	0.28
3	4.7	3.2	1.3	0.2	setosa	0.26

5.6 Sorting and enumerating factors

Use the `sort` function to sort on a subset of dimensions and/or attribute of a SciDB array object, creating a new sorted array.

Use the `unique` function to return a SciDB array that removes duplicate elements of a single-attribute SciDB input array.

Use the `index_lookup` function along with `unique` to bind a new variable that enumerates unique values of a variable similarly to the R `factor` function.

Examples follow. Note that we use the SciDB `project` function in one example. `project` presents an alternative syntax to the functionally equivalent column subset selection of variables using brackets.

```
> x <- as.scidb(iris)
> # Sort x by Petal_Width and Species
> a <- sort(x, attributes=c("Petal_Width", "Species"))
> head(a, n=3)
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
1	4.9	3.1	1.5	0.1	setosa
2	4.9	3.6	1.4	0.1	setosa
3	4.8	3.0	1.4	0.1	setosa

```
> # Find unique values of Species:
> unique(a[, "Species"])[ ]
```

	Species
0	setosa
1	versicolor
2	virginica

```
> # Add a new variable that enumerates factor levels of Species:
> head(index_lookup(a, unique(project(a, "Species")), "Species"))
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species	Species_index
1	4.9	3.1	1.5	0.1	setosa	0
2	4.9	3.6	1.4	0.1	setosa	0
3	4.8	3.0	1.4	0.1	setosa	0
4	4.3	3.0	1.1	0.1	setosa	0
5	5.2	4.1	1.5	0.1	setosa	0
6	4.9	3.0	1.4	0.2	setosa	0

6 Lazy evaluation, array promises, and garbage collection

Most operations on SciDB array objects return array promises—new SciDB array objects that have not been fully evaluated by SciDB yet, but that promise to return values when asked.

SciDB array promises are essentially just SciDB query expressions together with a schema that the resulting output array will have once it's been evaluated and an environment providing context for the evaluation. In such cases, the `name` slot of a `scidb` or `scidbdf` object shows the SciDB query expression.

Sometimes it can be more efficient to explicitly evaluate and cache a SciDB array then to use it as an array promise. This can always be achieved by using the `scidbeval` function, or by using the ``eval`=TRUE` argument in the functions that support that. Dynamically allocated arrays use a naming convention that begins with “R_array” and end with a unique numeric identifier determined by the current SciDB session.

Ephemeral intermediate arrays are by default connected to R's garbage collector and automatically deleted from the SciDB catalog when they are no longer needed by R. Users can disconnect SciDB array objects from R's garbage collector (making stored arrays persistent in SciDB), by using the `gc=FALSE` flag in the `as.scidb` and `scidbeval` functions.

SciDB arrays keep references to other SciB array dependencies in a list within an environment, preventing them from automatic garbage collection so that the promise can be evaluated. The environment slot is named `@gc` in the array objects and the dependency list is `@gc$depend`.

Here is an example:

```
> # Create a 5x4 SciDB matrix named 'test' in the database. We set `gc=TRUE`
> # so that the SciDB array `test` will be deleted by R's garbage collector
> # for us automatically:
> A <- as.scidb(matrix(rnorm(20),nrow=5), name="test", gc=TRUE)
> A@name

[1] "test"

> # Assign the third row of the transpose to a scidb object B:
> B <- t(A)[3,]
>
> # B is a promise object--it's name is an (unevaluated) SciDB query expression
> # that depends on A. But it has a valid output schema, and we also see that
> # it depends ultimately on the SciDB array `test`:
> B@name

[1] "sg(subarray(between(transpose(test),3,0,3,4),3,0,3,4),1,-1)"

> B@schema
```

```
[1] "array<val:double> [j=0:0,4,0,i=0:4,5,0]"

> B@gc$depend[[1]]@gc$depend[[1]]@name

[1] "test"

> # We can use `scidbeval` to force B's evaluation and storage inside SciDB:
> C <- scidbeval(B)
> # C has the same schema as B, but is now evaluated and stored inside SciDB
> # using an automatically-generated name. Unlike B, C does not depend on
> # the `test` array any more.
> C@name

[1] "R_array3ec360a6e1cc1101127319914"

> C@schema

[1] "array<val:double> [j=0:0,4,0,i=0:4,5,0]"
```

7 SciDB-specific functions

This section briefly summarizes a number of utility functions that are simple wrappers of miscellaneous SciDB operators. Except for `count`, which returns a number, the listed functions return references to new SciDB array objects.

- `attribute_rename` Rename one or more attributes of a SciDB array.
- `build` Creates new dense arrays from SciDB expressions and schema.
- `cast` Create a new array by altering the schema of an array.
- `count` Return the number of nonempty cells of an array.
- `cumulate` Compute a cumulative aggregate function, for example a cumulative sum or product.
- `index_lookup` Look up attribute values in a mapping array, adding a new attribute with the index.
- `project` Pick out one or more attributes from an array.
- `redimension` Redimension an array into the provided new schema.

- `regrid` Decimate an array by binning it along one or more dimensions and applying an aggregate function to the bins.
- `rename` Rename a SciDB array.
- `repart` Repartition an array.
- `slice` Slice a SciDB array along one of its axes, returning a lower-dimensional array.
- `sort` Sort the elements of an array.
- `unique` Return a set of unique elements of a sorted array.
- `xgrid` Prolong one or more coordinate axes by replicating cell values.

See the online manual pages for more details and examples.

8 Error handling

SciDB errors are trapped and converted to R errors that can be handled by standard R mechanisms. Some operations might try to return too much data to R, exceeding R's indexing limitations, system memory, or both. The package tries to avoid this kind of error using package options that limit returned data size shown in the package options section below.

9 Function and method list

Here is a summary list of functions and methods defined by the package. See online help for examples and more information on each one.

<code>\$</code>	<code>crossprod</code>	<code>names</code>	<code>scidbrm</code>
<code>%*%</code>	<code>cumulate</code>	<code>names<=</code>	<code>sd</code>
<code>[</code>	<code>df2scidb</code>	<code>ncol</code>	<code>show</code>
<code>acos</code>	<code>diag</code>	<code>nrow</code>	<code>sin</code>
<code>aggregate</code>	<code>diff</code>	<code>print</code>	<code>slice</code>
<code>apply</code>	<code>dim</code>	<code>project</code>	<code>sort</code>
<code>as.scidb</code>	<code>dimnames</code>	<code>redimension</code>	<code>str</code>
<code>asin</code>	<code>dimnames<=</code>	<code>regrid</code>	<code>subset</code>
<code>atan</code>	<code>head</code>	<code>rename</code>	<code>sum</code>
<code>attribute_rename</code>	<code>image</code>	<code>repart</code>	<code>svd</code>

<code>between</code>	<code>index_lookup</code>	<code>reshape</code>	<code>sweep</code>
<code>bind</code>	<code>iquery</code>	<code>rownames</code>	<code>t</code>
<code>build</code>	<code>is.scidb</code>	<code>rownames<-</code>	<code>tail</code>
<code>cast</code>	<code>is.scidbdf</code>	<code>scidb</code>	<code>tan</code>
<code>cbind</code>	<code>length</code>	<code>scidbconnect</code>	<code>tcrossprod</code>
<code>colnames</code>	<code>log</code>	<code>scidbeval</code>	<code>unique</code>
<code>colnames<-</code>	<code>max</code>	<code>scidblist</code>	<code>unpack</code>
<code>cos</code>	<code>merge</code>	<code>scidbls</code>	<code>var</code>
<code>count</code>	<code>min</code>	<code>scidbremove</code>	<code>xgrid</code>

10 Package installation

Installation proceeds in two steps: installing the R package on any computer that has a network connection to a SciDB database, and installing a simple network service on the SciDB database coordinator computer.

10.1 Installing the R package from sources on GitHub

The `scidb` package source is maintained in the SciDBR GitHub repository. That's where the most up-to-date version of the package is available. Released versions of the package posted to CRAN are updated much less frequently, approximately semiannually. A git tag indicates each CRAN release version of the package in the source code repository.

The wonderful `devtools` R package makes installation of source packages from GitHub nearly as simple as installation from CRAN.

```
library("devtools")
install_github("SciDBR", "Paradigm4", quick=TRUE)
```

10.2 Installing the R package from CRAN

The `scidb` package is available on CRAN. Start an R session and run:

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

10.3 Installing the simple HTTP service for SciDB

The SciDB R package requires installation of a simple open-source HTTP network service called `shim` on the computer that SciDB is installed on. The service needs to be installed only on the SciDB coordinator computer, not on client computers that connect to SciDB from R. It's available in packaged binary form for supported SciDB operating systems, and as source code which can be compiled and deployed on any SciDB installation.

Both installation approaches install the `shim` network service on the SciDB coordinator computer. Installing as a service requires root permission. The compiled source code version requires no special permissions to run.

Installation from binary software packages for SciDB-supported operating systems is easiest. Detailed up-to-date information can be found on Paradigm4's laboratory on Paradigm4's Github repository at <https://github.com/Paradigm4/shim/wiki/Installing-shim>. We outline installation for each supported operating system below. See our github page for source code. The open source package author, Bryan Lewis, maintains binary packages for SciDB-supported operating systems. They are tied to specific versions of SciDB. The present version is 13.12 (December, 2013).

10.3.1 Installation on RHEL/CentOS 6

```
# Install with:
wget http://paradigm4.github.io/shim/shim-13.12-1.x86_64.rpm
rpm -i shim-13.12-1.x86_64.rpm

# (Uninstall, if desired, with:)
yum remove shim
```

10.3.2 Installation on Ubuntu 12.04

```
# Install with:
wget http://paradigm4.github.io/shim/shim_13.12_amd64.deb
sudo gdebi shim_13.12_amd64.deb

# (Uninstall, if desired, with:)
apt-get remove shim
```


See the Wiki and web pages at <https://github.com/Paradigm4/shim/> for up to date package information and source code.

The installed `shim` network service exposes SciDB as a very simple HTTP API. It includes a simple browser-based status and query tool. After installing `shim`, point your browser to the I.P. address of the SciDB coordinator machine and port 8080. Note that this API is not official and may change in the future. Help drive those changes by contributing ideas, code and bugfixes to the project on Github, or feel free to discuss the service on the SciDB.org/forum or via Github issues.

10.4 Package options, miscellaneous notes, and software license

The `scidb` package defines several global package options. Package options may be set and retrieved with the R `options` function, and are listed in Table 2.

Option	Default value	Description
<code>scidb.debug</code>	NULL	Set to TRUE to display all queries issued to the SciDB engine and other debugging information.
<code>scidb.index.sequence.limit</code>	100 000 000	Maximum allowed <code>scidb</code> array object sequential indexing limit (for larger ranges, use between)
<code>scidb.max.array.elements</code>	100 000 000	Maximum allowed non-empty elements to return in a subsetting operation of a <code>scidb</code> array object.

Package options

Miscellaneous notes:

- R does not support 64-bit integer types. 64-bit signed and unsigned integers smaller than 2^{53} in magnitude will be represented as double-precision floating point numbers. 64-bit integers outside that range appear as $+/-\text{Inf}$. All other integers (`int8`, `uint8`, `int16`, `uint16`, etc.) are represented in R by 32-bit signed integers. The SciDB `uint32` type is not supported.

- R doesn't support single-precision floating point numbers. `iquery` results convert single-precision numbers within SciDB to double-precision floating-point numbers in R. Single-precision SciDB numbers are not supported by the `scidb` array class.
- SciDB does not natively support complex numbers. Loading complex numbers directly into SciDB from R is not defined.
- The `iquery` function provides the most flexible mechanism for type conversion between the systems, fully under user control using `read.table` options.
- Naming convention and associated restrictions vary between R and SciDB. For example, SciDB does not allow decimal points in attribute names. The package may alter names with character substitution to reconcile names when it is reasonable to do so. A warning is emitted whenever an object is automatically renamed in this way.

Copyright (C) 2008–2013 SciDB, Inc.

The SciDB package for R is free software: you can redistribute it and/or modify it under the terms of the AFFERO GNU General Public License as published by the Free Software Foundation.

The SciDB package for R is distributed "AS-IS" AND WITHOUT ANY WARRANTY OF ANY KIND, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. See the AFFERO GNU General Public License for the complete license terms.

You should have received a copy of the AFFERO GNU General Public License along with the package. If not, see <<http://www.gnu.org/licenses/agpl-3.0.html>>