

Tables and Queries With **aRT**

Pedro Ribeiro de Andrade Neto
Paulo Justiniano Ribeiro Júnior

February 6, 2006

Contents

1	Introduction	1
2	Static tables	3
2.1	External tables	7
2.2	Media tables	8
3	Attribute queries	9
3.1	Join tables	9
3.2	Attribute restrictions	9
3.3	Spatial queries	9
4	Temporal Tables	9
4.1	Event tables	10
4.2	Dynamic attribute tables	12
4.3	Dynamic geometry tables	12
4.4	Fully dynamic tables	12
5	Temporal queries	12
5.1	Joining tables	13
5.2	Temporal slicing	13

1 Introduction

In **aRT** there are seven types of table, implementing the **TerraLib** models. They are:

- static,
- external,
- media,
- event,

- dynatt,
- dynggeom, and
- alldyn.

These tables are encapsulated in the class `aRTtable`, and we can query them using `aRTtheme` and `aRTquerier` objects. The three classes are discussed in this document, and we present here how to execute database queries, selecting and joining attributes.

```
> library(aRT)

Loading required package: sp

-----
R-TERRALIB API
aRT version 0.4-15 (2005-12-20) is now loaded
-----

> conn = openConn()

Trying to connect ... yes

> if (any(showDbs(conn) == "tabletest")) deleteDb(conn, "tabletest",
+       force = T)

Checking for database 'tabletest' ... yes
Deleting database 'tabletest' ... yes

> db = createDb(conn, db = "tabletest")

Connecting with database 'tabletest' ... no
Creating database 'tabletest' ... yes
Creating conceptual model of database 'tabletest' ... yes
Loading layer set of database 'tabletest' ... yes
Loading view set of database 'tabletest' ... yes

We will create a new layer, populate with some geometries, and, for each
type of table, we discuss how to populate the table, and how to make temporal
and attributes queries, using the other aRT classes. Note that spatial querier
can be done directly using aRTlayer objects.

> quantity = 10
> xc = round(runif(quantity), 2)
> yc = round(runif(quantity), 2)
> xy = cbind(xc, yc)
> xy.sp = SpatialPoints(xy)
> xy.spdf = SpatialPointsDataFrame(xy, data.frame(ID = paste(1:quantity)))
```

```

> lstatic = createLayer(db, l = "static")

Building projection to layer 'static' ... yes
Creating layer 'static' ... yes

> addPoints(lstatic, xy.spdf)

Converting points to TerraLib format ... yes
Adding 10 points to layer 'static' ... yes
Reloading tables of layer 'static' ... yes

```

All non-geometric data in **aRT** is stored in tables, and the way to exchange this type of information with the database is using **aRTtable** objects. **aRTtable** objects are created and opened from **aRTlayer** objects, using **createTable** and **openTable**, respectively. Table data is read from and written to databases using **data.frame**. **row.names** are not used to represent IDs in **aRT**, because IDs can be unsufficient to describe temporal data.

2 Static tables

The most basic type of table in **aRT** is *static*. Static tables store attributes with no variation in the time, for example the object **df** below:

```

> id = getID(xy.spdf)
> norm = unlist(lapply(id, function(x) rnorm(1, 20, 10)))
> unif = unlist(lapply(id, function(x) runif(1, 1, 100)))
> df = data.frame(id, norm, unif)
> df

```

	id	norm	unif
1	1	11.984301	97.89574
2	2	19.713813	31.62978
3	3	15.566071	89.62430
4	4	21.704297	56.84160
5	5	6.500174	38.60381
6	6	6.332662	36.27251
7	7	11.063396	63.53696
8	8	32.873980	23.49168
9	9	34.684389	11.19028
10	10	7.358129	91.07037

To create a static table in a layer we use **createTable()**:

```

> tstatic = createTable(lstatic, "tstatic")

Creating static table 'tstatic' on layer 'static' ... yes
Creating link ids ... yes

```

```
> tstatic
```

Object of class aRTtable

Table: "tstatic"

Type: static

Layer: "static"

Rows: 10

Attributes:

id: character[16] (key)

Note that when we create a table it already has 10 rows with the unique ids from the geometries. To add this data to the table we use `updateColumns`, and it also create new columns using `colnames(df)`.

```
> updateColumns(tstatic, df)
```

Checking for column 'id' in table 'tstatic' ... yes

Checking for column 'norm' in table 'tstatic' ... no

Creating column 'norm' in table 'tstatic' ... yes

Checking for column 'unif' in table 'tstatic' ... no

Creating column 'unif' in table 'tstatic' ... yes

Converting 3 attributes to TerraLib format ... yes

Converting 10 rows to TerraLib format ... yes

Updating columns of table 'tstatic' ... yes

```
> tstatic
```

Object of class aRTtable

Table: "tstatic"

Type: static

Layer: "static"

Rows: 10

Attributes:

id: character[16] (key)

norm: numeric

unif: numeric

And finally, `getData` is used for reading the data from the table:

```
> getData(tstatic)
```

	id	norm	unif
1	1	11.984301	97.89574
2	10	7.358129	91.07037
3	2	19.713813	31.62978
4	3	15.566071	89.62430

```

5  4 21.704297 56.84160
6  5  6.500174 38.60381
7  6  6.332662 36.27251
8  7 11.063396 63.53696
9  8 32.873980 23.49168
10 9 34.684389 11.19028

```

```

> df2 = data.frame(id, norm = unlist(lapply(id, function(x) rnorm(1,
+      20, 10)))), unif = unif)
> updateColumns(tstatic, df2)

```

```

Checking for column 'id' in table 'tstatic' ... yes
Checking for column 'norm' in table 'tstatic' ... yes
Checking for column 'uniff' in table 'tstatic' ... no
Creating column 'uniff' in table 'tstatic' ... yes
Converting 3 attributes to TerraLib format ... yes
Converting 10 rows to TerraLib format ... yes
Updating columns of table 'tstatic' ... yes

```

```

> getData(tstatic)

```

	id	norm	unif	uniff
1	1	10.982032	97.89574	97.89574
2	10	49.948164	91.07037	91.07037
3	2	5.710605	31.62978	31.62978
4	3	15.653211	89.62430	89.62430
5	4	36.561501	56.84160	56.84160
6	5	19.498305	38.60381	38.60381
7	6	18.246118	36.27251	36.27251
8	7	4.481289	63.53696	63.53696
9	8	10.616478	23.49168	23.49168
10	9	30.935900	11.19028	11.19028

Note that, as aRT automatically have created the rows, the order of the rows in the result is not the same of the df.

To avoid it, we can create an empty table, and populate it manually. First, we need to use `gen=FALSE`, to avoid generate the rows of the table:

```

> tstatic2 = createTable(lstatic, "tstatic2", gen = FALSE)

```

```

Creating static table 'tstatic2' on layer 'static' ... yes

```

```

> tstatic2

```

```

Object of class aRTtable

```

```

Table: "tstatic2"

```

```

Type: static

```

```

Layer: "static"
Rows: 0
Attributes:
  id: character[16] (key)

```

The argument `gen` indicates that the function must create one row for each spatial object, and fill it with the id of the spatial object. Now we need to create two columns, one of integer type and other of real type, and then we add some rows to the table.

```

> createColumn(tstatic2, "norm", type = "integer")

Checking for column 'norm' in table 'tstatic2' ... no
Creating column 'norm' in table 'tstatic2' ... yes

> createColumn(tstatic2, "unif", type = "numeric")

Checking for column 'unif' in table 'tstatic2' ... no
Creating column 'unif' in table 'tstatic2' ... yes

> addRows(tstatic2, df[1:5, ])

Converting 3 attributes to TerraLib format ... yes
Converting 5 rows to TerraLib format ... yes
Inserting data into table 'tstatic2' ... yes

> getData(tstatic2)

```

	id	norm	unif
1	1	12	97.89574
2	2	20	31.62978
3	3	16	89.62430
4	4	22	56.84160
5	5	7	38.60381

```

> addRows(tstatic2, df[6:10, ])

Converting 3 attributes to TerraLib format ... yes
Converting 5 rows to TerraLib format ... yes
Inserting data into table 'tstatic2' ... yes

> getData(tstatic2)

```

	id	norm	unif
1	1	12	97.89574
2	2	20	31.62978
3	3	16	89.62430
4	4	22	56.84160
5	5	7	38.60381

```

6  6    6 36.27251
7  7   11 63.53696
8  8   33 23.49168
9  9   35 11.19028
10 10    7 91.07037

```

But `addRows` only creates new elements in the table, it cannot change the old elements. For example

```
> addRows(tstatic2, data.frame(id = "1", norm = 2.1, unif = 0.3))
```

```

Converting 3 attributes to TerraLib format ... yes
Converting 1 rows to TerraLib format ... yes
Inserting data into table 'tstatic2' ... yes

```

```
> getData(tstatic2)[1:3, ]
```

```

      id norm      unif
1  1    12 97.89574
2  2    20 31.62978
3  3    16 89.62430

```

We can also create columns of string type, and set the maximum size of the string, as:

```
> createColumn(tstatic2, "charcol", type = "character", length = 5)
```

```

Checking for column 'charcol' in table 'tstatic2' ... no
Creating column 'charcol' in table 'tstatic2' ... yes

```

```
> tstatic2
```

Object of class `aRTtable`

```

Table: "tstatic2"
Type: static
Layer: "static"
Rows: 10
Attributes:
  id: character[16] (key)
  norm: integer
  unif: numeric
  charcol: character[5]

```

`updateColumns()` already calculates the type and the size of the data, before creating the columns.

2.1 External tables

External tables are tables with no geometry associated. Therefore they are created directly from the database, and we can not use `genid`. We create external tables also using `createTable`, as in the next example:

```
> textexternal = createTable(db, "textexternal", id = "myid", length = 5)
```

```
Checking for table 'textexternal' ... no
```

```
Creating external table 'textexternal' ... yes
```

```
> textexternal
```

```
Object of class aRTtable
```

```
Table: "textexternal"
```

```
Type: external
```

```
Rows: 0
```

```
Attributes:
```

```
  myid: character[5] (key)
```

Note that here we define the name of the key and also its length. It can be defined when creating tables from layers too. As `textexternal` is an object of class `aRTtable` we can use the same functions as described for tables from layers.

2.2 Media tables

Media tables are useful when building databases that will be used in `TerraView`, or another `TerraLib`-based GIS. It associates a web page to a double-click in a drawn geometry. This type of table can be created using `type="media"`:

```
> mediatable = createTable(lstatic, type = "media")
```

```
Creating media table to layer 'static' ... yes
```

A layer can have one, and only one, media table, and a media table does not have a name. Also, each media table has two, and only two, attributes: `object_id`, the link to geometries, and `media_name`, a web address.

```
> mediatable
```

```
Object of class aRTtable
```

```
Type: media
```

```
Layer: static
```

```
Rows: 0
```

```
Attributes:
```

```
  object_id: character[50] (key)
```

```
  media_name: character[255] (key)
```


A media table can be manipulated as all the other tables, but new columns can not be created. In the next code we associate web pages to each geometry of the layer, and we use `addRows()` to fill the table.

```
> id = getID(xy.spdf)
> url = "http://www.est.ufpr.br/~pedro/media/media"
> name = lapply(id, function(x) sprintf("%s%s.html", url, x))
> name = unlist(name)
> df = data.frame(object_id = id, media_name = name)
> addRows(mediatable, df)
```

Adding 10 urls ... yes

To check if it is correct, we can use `getData()`:

```
> getData(mediatable)[1:5, ]

  object_id                               media_name
1         1 http://www.est.ufpr.br/~pedro/media/media1.html
2         2 http://www.est.ufpr.br/~pedro/media/media2.html
3         3 http://www.est.ufpr.br/~pedro/media/media3.html
4         4 http://www.est.ufpr.br/~pedro/media/media4.html
5         5 http://www.est.ufpr.br/~pedro/media/media5.html
```

```
> lstatic
```

Object of class `aRTlayer`

```
Layer: "static"
Database: "tabletest"
Number of polygons: 0
Number of lines: 0
Number of points: 10
Layer does not have raster data
Projection Name: "NoProjection"
Projection Datum: "Spherical"
Projection Longitude: 0
Projection Latitude: 0
Tables:
  "tstatic": static
  "tstatic2": static
  "media_layer_1": media
```

3 Attribute queries

We can get all the data of a table using `getData()`. But some operations are useful, for example selecting values that follows a condition, and it can be an

attribute, or spatial, or temporal condition. In the case of spatial queries, here we only use the result to get spatial/attribute queries. If you want to see how spatial queries work, see *Spatial Queries*.

To execute database queries, we need to create **aRTtheme** objects.

3.1 Join tables

3.2 Attribute restrictions

3.3 Spatial queries

4 Temporal Tables

Temporal tables in **aRT** work as static tables, noting that there are three identifiers, instead of only one in static tables. The two others are the initial and the final time.

These attributes are strings, but they follow the TerraLib model of dates. To convert temporal dates to **aRT** format we will use `toDate()`. This function gets as arguments integer variables `year = 0`, `month = 1`, `day = 1`, `hour = 0`, `minute = 0` and `second = 0` and returns a string describing the date. It is a bit different from `ISOdate`.

```
> toDate(year = 2005, month = 8, day = 7, hour = 6, sec = 5)
```

```
[1] "2005-08-07 06:00:05"
```

```
> ISOdate(year = 2005, month = 8, day = 7, hour = 6, sec = 5)
```

```
[1] "2005-08-07 06:00:05 GMT"
```

4.1 Event tables

An event table represents a temporal table which each element has a static geometry and attributes, but it occurs in a time interval. When we are using an event table, we do not need static tables because each event is unique, and therefore we can put all attributes in the same table. We will use the same layer to create an event table.

```
> lpoints = lstatic
```

To create an event table, we need to set `type="event"` at `createTable()`. The default value of this argument is `"static"` when creating from layers and `"external"` when creating directly from databases.

```
> tevent = createTable(lpoints, "events", type = "event")
```

```
Creating event table 'events' on layer 'static' ... yes
```

```
> tevent
```

Object of class aRTtable

```
Table: "events"
Type: event
Layer: "static"
Rows: 0
Attributes:
  id: character[16] (key)
  itime: date (key)
  ftime: date (key)
```

When an event table is created, it already contains three attributes: `id`, `itime` and `ftime`, and they are keys. We recommend not to generate ids (`gen=FALSE`), because it would also generate `itime` and `ftime`, and put zero in all time values (0000-00-00 00:00:00).

To fill the event table we will generate a random attribute `value` with duration of 59 minutes, all in the same day:

```
> id = getID(xy.spdf)
> hours = unlist(lapply(id, function(x) round(runif(1, 0, 10),
+      0))))
> itime = unlist(lapply(hours, function(x) toDate(y = 2005,
+      month = 3, day = 30, hour = x)))
> ftime = unlist(lapply(hours, function(x) toDate(y = 2005,
+      month = 3, day = 30, hour = x, minute = 59)))
> value = unlist(lapply(id, function(x) runif(1, 1, 100)))
> df = data.frame(id, value, itime, ftime)
> df[1:7, ]
```

	id	value	itime	ftime
1	1	37.292322	2005-03-30 04:00:00	2005-03-30 04:59:00
2	2	13.109769	2005-03-30 05:00:00	2005-03-30 05:59:00
3	3	98.482764	2005-03-30 01:00:00	2005-03-30 01:59:00
4	4	32.542697	2005-03-30 03:00:00	2005-03-30 03:59:00
5	5	58.776804	2005-03-30 04:00:00	2005-03-30 04:59:00
6	6	80.230024	2005-03-30 02:00:00	2005-03-30 02:59:00
7	7	4.076095	2005-03-30 01:00:00	2005-03-30 01:59:00

As the table already has three attributes, we need only to create the column `value`, and then we can add the rows:

```
> createColumn(tevent, "value", type = "i")
```

```
Checking for column 'value' in table 'events' ... no
Creating column 'value' in table 'events' ... yes
```

```

> addRows(tevent, df)

Converting 4 attributes to TerraLib format ... yes
Converting 10 rows to TerraLib format ... yes
Inserting data into table 'events' ... yes

> tevent

Object of class aRTtable

Table: "events"
Type: event
Layer: "static"
Rows: 10
Attributes:
  id: character[16] (key)
  itime: date (key)
  ftime: date (key)
  value: integer

> getData(tevent)

   id      itime      ftime value
1  1 2005-03-30 04:00:00 2005-03-30 04:59:00    37
2  2 2005-03-30 05:00:00 2005-03-30 05:59:00    13
3  3 2005-03-30 01:00:00 2005-03-30 01:59:00    98
4  4 2005-03-30 03:00:00 2005-03-30 03:59:00    33
5  5 2005-03-30 04:00:00 2005-03-30 04:59:00    59
6  6 2005-03-30 02:00:00 2005-03-30 02:59:00    80
7  7 2005-03-30 01:00:00 2005-03-30 01:59:00     4
8  8 2005-03-30 09:00:00 2005-03-30 09:59:00    87
9  9 2005-03-30 06:00:00 2005-03-30 06:59:00    54
10 10 2005-03-30 01:00:00 2005-03-30 01:59:00    40

```

4.2 Dynamic attribute tables

Dynamic attribute tables work with geometries where one or more attributes changes with the time. It works such as event tables, with the conceptual difference that the identifier may repeat.

```

> tdynatt = createTable(lpoints, "dynatt", type = "dynatt")

Creating dynamic table 'dynatt' on layer 'static' ... yes

> tdynatt

Object of class aRTtable

```

```

Table: "dynatt"
Type: dynattr
Layer: "static"
Rows: 0
Attributes:
  id: character[16] (key)
  itime: date (key)
  ftime: date (key)

```

4.3 Dynamic geometry tables

(not implemented yet)

4.4 Fully dynamic tables

(not implemented yet)

5 Temporal queries

We can get all table data with `getData()`, but if it is a temporal table, we get it sliced. To do it, we need first to create an `aRTtheme` object.

```
> theme = createTheme(lpoints, "events", table = "events")
```

```

Checking for theme 'events' in layer 'tabletest' ... no
Creating theme 'events' on layer 'static' ... yes
Checking for view 'events' in database 'tabletest' ... no
Creating view 'events' ... yes
Inserting view 'events' in database 'tabletest' ... yes
Checking tables of theme 'events' ... yes
Saving theme 'events' ... yes
Building collection of theme 'events' ... yes

```

```
> theme
```

Object of class `aRTtheme`

```

Theme: "events"
Layer: "static"
View: "events"
Tables:
  "events"
  Attributes: "id", "itime", "ftime", "value"

```

```
Attribute Restriction: ""
```

Note that the theme has two tables ("static" and "events"), and `getData()` returns the join of them.

5.1 Joining tables

5.2 Temporal slicing

To slice the theme data, we need to create an `aRTquerier`, with `openQuerier()`. This function takes as argument `chronon`, representing the type of slides to be produced. It can be `"second"`, `"month"`, `"season"`, `"year"`, `"weekofyear"` and others, and the default is `"nochronon"`. To exemplify using `aRTquerier`, we implement an algorithm to calculate the number of occurrences in each hour, and the sums of `value`. Therefore we need an `aRTquerier` sliced by hour.

```
> querier = openQuerier(theme, chronon = "hour")
> querier
```

Object of class `aRTquerier`

```
Theme: "events"
Retrieves: geometry
Frames: 0/9
Elements: 0/0
```

To get data from the querier there are two functions. `nextSlide()` loads the next slide, returning the number of elements of it, and `getData()` returns one of the elements of the slide, read from the database. Both functions do not take any argument.

```
> qtde = summary(querier)$slides
```

References

Chambers, J.M., 1998, Programming with data, a guide to the S language. Springer, New York.