Addison Wesley Data & Analytics Series

R for Everyone

Advanced **Analytics** and **Graphics**

SECOND EDITION

JARED P. LANDER

R for Everyone

Second Edition

regular data.frames, so it will take getting used to, which is probably the primary reason it has not seen near-universal adoption.

The secret to the speed is that data.tables have an index like databases. This enables faster value accessing, group by operations and joins.

Creating data.tables is just like creating data.frames, and the two are very similar.

```
> library(data.table)
> # create a regular data.frame
> theDF <- data.frame(A=1:10,
+                     B=letters[1:10],
+                     C=LETTERS[11:20],
+                     D=rep(c("One", "Two", "Three"), length.out=10))
> # create a data.table
> theDT <- data.table(A=1:10,
+                     B=letters[1:10],
+                     C=LETTERS[11:20],
+                     D=rep(c("One", "Two", "Three"), length.out=10))
> # print them and compare
> theDF

    A B C     D
1   1 a K   One
2   2 b L   Two
3   3 c M Three
4   4 d N   One
5   5 e O   Two
6   6 f P Three
7   7 g Q   One
8   8 h R   Two
9   9 i S Three
10 10 j T   One

> theDT

     A B C     D
 1:  1 a K   One
 2:  2 b L   Two
 3:  3 c M Three
 4:  4 d N   One
 5:  5 e O   Two
 6:  6 f P Three
 7:  7 g Q   One
 8:  8 h R   Two
 9:  9 i S Three
10: 10 j T   One

> # notice by default data.frame turns character data into factors
> # while data.table does not
> class(theDF$B)
```

```
[1] "factor"
```

```
> class(theDT$B)
```

```
[1] "character"
```

The data is identical—except that **data.frame** turned `B` into a `factor` while **data.table** did not—and only the way it was printed looks different.

It is also possible to create a `data.table` out of an existing `data.frame`.

```
> diamondsDT <- data.table(diamonds)
> diamondsDT
```

```
       carat       cut color clarity depth table price    x    y    z
    1:  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
    2:  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
    3:  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
    4:  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
    5:  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
   ---
53936:  0.72     Ideal     D     SI1  60.8    57  2757 5.75 5.76 3.50
53937:  0.72      Good     D     SI1  63.1    55  2757 5.69 5.75 3.61
53938:  0.70 Very Good     D     SI1  62.8    60  2757 5.66 5.68 3.56
53939:  0.86   Premium     H     SI2  61.0    58  2757 6.15 6.12 3.74
53940:  0.75     Ideal     D     SI2  62.2    55  2757 5.83 5.87 3.64
```

Notice that printing the `diamonds` data would try to print out all the data but `data.table` intelligently just prints the first five and last five rows.

Accessing rows can be done similarly to accessing rows in a `data.frame`.

```
> theDT[1:2, ]
```

```
   A B C   D
1: 1 a K One
2: 2 b L Two
```

```
> theDT[theDT$A >= 7, ]
```

```
    A B C     D
1:  7 g Q   One
2:  8 h R   Two
3:  9 i S Three
4: 10 j T   One
```

```
> theDT[A >= 7, ]
```

```
    A B C     D
1:  7 g Q   One
2:  8 h R   Two
3:  9 i S Three
4: 10 j T   One
```

While the second line in the preceding code is valid syntax, it is not necessarily efficient syntax. That line creates a vector of length `nrow(theDT)` = 10 consisting of `TRUE` or `FALSE` entries, which is a vector scan. After we create a key for the `data.table` we can use different syntax to pick rows through a binary search, which will be much faster and is covered in Section 11.4.1. The third line computes the same result as the second and is nicely without the dollar sign notation. This is because the **data.table** function knows to find the column `A` within the `theDT` `data.table`.

Accessing individual columns must be done a little differently than accessing columns in `data.frames`. In Section 5.1 we show that multiple columns in a `data.frame` should be specified as a `character` `vector`. With `data.tables` the columns should be specified as a `list` of the actual names, not as `characters`.

```
> theDT[, list(A, C)]

     A C
 1:  1 K
 2:  2 L
 3:  3 M
 4:  4 N
 5:  5 O
 6:  6 P
 7:  7 Q
 8:  8 R
 9:  9 S
10: 10 T

> # just one column
> theDT[, B]

 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

> # one column while maintaining data.table structure
> theDT[, list(B)]

    B
 1: a
 2: b
 3: c
 4: d
 5: e
 6: f
 7: g
 8: h
 9: i
10: j
```

If we must specify the column names as `characters` (perhaps because they were passed as arguments to a function), the `with` argument should be set to `FALSE`.

```
> theDT[, "B", with=FALSE]

    B
 1: a
 2: b
 3: c
 4: d
 5: e
 6: f
 7: g
 8: h
 9: i
10: j

> theDT[, c("A", "C"), with=FALSE]

     A C
 1:  1 K
 2:  2 L
 3:  3 M
 4:  4 N
 5:  5 O
 6:  6 P
 7:  7 Q
 8:  8 R
 9:  9 S
10: 10 T

> theCols <- c("A", "C")
> theDT[, theCols, with=FALSE]

     A C
 1:  1 K
 2:  2 L
 3:  3 M
 4:  4 N
 5:  5 O
 6:  6 P
 7:  7 Q
 8:  8 R
 9:  9 S
10: 10 T
```

This time we used a `vector` to hold the column names instead of a `list`. These nuances are important to proper functions of `data.tables` but can lead to a great deal of frustration.

11.4.1 Keys

Now that we have a few data.tables in memory, we might be interested in seeing some information about them.

```
> # show tables
> tables()

     NAME         NROW NCOL MB
[1,] diamondsDT 53,940   10  4
[2,] theDT          10    4  1
[3,] tomato3        16   11  1
     COLS
[1,] carat,cut,color,clarity,depth,table,price,x,y,z
[2,] A,B,C,D
[3,] Round,Tomato,Price,Source,Sweet,Acid,Color,Texture,Overall,Avg of Totals,Total o
     KEY
[1,]
[2,]
[3,]
Total: 6MB
```

This shows, for each data.table in memory, the name, the number of rows, the size in megabytes, the column names and the key. We have not assigned keys for any of the tables so that column is blank. The key is used to index the data.table and will provide the extra speed.

We start by adding a key to theDT. We will use the D column to index the data.table. This is done using **setkey**, which takes the name of the data.table as its first argument and the name of the desired column (without quotes, as is consistent with column selection) as the second argument.

```
> # set the key
> setkey(theDT, D)
> # show the data.table again
> theDT

     A B C     D
 1:  1 a K   One
 2:  4 d N   One
 3:  7 g Q   One
 4: 10 j T   One
 5:  3 c M Three
 6:  6 f P Three
 7:  9 i S Three
 8:  2 b L   Two
 9:  5 e O   Two
10:  8 h R   Two
```

The data have been reordered according to column D, which is sorted alphabetically. We can confirm the key was set with **key**.

```
> key(theDT)

[1] "D"
```

Or **tables**.

```
> tables()

     NAME         NROW NCOL MB
[1,] diamondsDT 53,940   10  4
[2,] theDT          10    4  1
[3,] tomato3        16   11  1
     COLS
[1,] carat,cut,color,clarity,depth,table,price,x,y,z
[2,] A,B,C,D
[3,] Round,Tomato,Price,Source,Sweet,Acid,Color,Texture,Overall,Avg of Totals,Total o
     KEY
[1,]
[2,] D
[3,]
Total: 6MB
```

This adds some new functionality to selecting rows from `data.tables`. In addition to selecting rows by the row number or by some expression that evaluates to `TRUE` or `FALSE`, a value of the key column can be specified.

```
> theDT["One", ]

    A B C   D
1:  1 a K One
2:  4 d N One
3:  7 g Q One
4: 10 j T One

> theDT[c("One", "Two"), ]

    A B C   D
1:  1 a K One
2:  4 d N One
3:  7 g Q One
4: 10 j T One
5:  2 b L Two
6:  5 e O Two
7:  8 h R Two
```

More than one column can be set as the key.

```
> # set the key
> setkey(diamondsDT, cut, color)
```

To access rows according to both keys, there is a special function named **J**. It takes multiple arguments, each of which is a `vector` of values to select.