Time Series Database Interface: R SQLite (TSSQLite)

May 7, 2008

1 Introduction

The code from the vignette that generates this guide can be loaded into an editor with *edit(vignette("TSSQLite"))*. This uses the default editor, which can be changed using *options()*. It should be possible to view the pdf version of the guide for this package with *print(vignette("TSSQLite"))*.

WARNING: running these example will overwrite tables in the SQLite "test" database on the server.

In SQLite there does not seem to be any need to set user or password information, and examples here all use the localhost.

Once R is started, the functions in this package are made available with

```
> library("TSSQLite")
```

This will also load required packages *TSdbi*, *DBI*, *RSQLite*, *methods*, and *tframe*. Some examples below also require *zoo*, and *tseries*.

The next small section of code is necessary to setup database tables. It needs to be done only once for a database and might typically be done by an administrator rather than an end user. A more detailed description of the instructions is given in the last section of this guide.

```
> m <- dbDriver("SQLite")
> con <- dbConnect(m, dbname = "test")
> source(system.file("TSsql/CreateTables.TSsql", package = "TSdbi"))
> dbDisconnect(con)
```

2 Using the Database - TSdbi Functions

This section gives several simple examples of putting series on and reading them from the database. (If a large number of series are to be loaded into a database, one would typically do this with a batch process using the database program's utilities for loading data.) The first thing to do is to establish a connection to the database:

```
> m <- dbDriver("SQLite")
> con <- TSconnect(m, dbname = "test")
```

TSconnect uses *dbConnect* from the *DBI* package, but checks that the database has expected tables, and checks for additional features. (It cannot be used before the tables are created, as done in the previous section.)

This puts a series called *vec* on the database and then reads is back

```
> z <- ts(rnorm(10), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- "vec"
> if (TSexists("vec", con)) TSdelete("vec", con)
> TSput(z, con)
> z <- TSget("vec", con)
```

If the series is printed it is seen to be a "ts" time series with some extra attributes.

TSput fails if the series already exists on the *con*, so the above example checks and deletes the series if it already exists. *TSreplace* does not fail if the series does not yet exist, so examples below use it instead. Several plots below show original data and the data retrieved after it is written to the database. One is added to the original data so that both lines are visible.

And now more examples:

```
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)

[1] TRUE

> TSget("matc1", con)

Time Series:
Start = 1990
End = 1999
Frequency = 1
         1          2          3          4          5          6          7
-0.2314490  0.8417985 -1.8754379 -1.4759967  0.2200534 -0.3295805  0.4239013
         8          9         10
-0.9176695 -1.3072813 -0.4221239
attr(,"seriesNames")
[1] matc1
attr(,"TSmeta")
<S4 Type Object>
attr(,"serIDs")
[1] "matc1"
attr(,"dbname")
[1] "test"
attr(,"con")
```

```
[1] "TSSQLiteConnection"
attr(,"con")attr(,"package")
[1] "TSSQLite"
attr(,"ExtractionDate")
[1] NA
attr(,"TSdescription")
[1] ""
attr(,"TSdoc")
[1] ""
attr(,"class")
[1] "TSmeta"
attr(,"class")attr(,"package")
[1] "TSdbi"
```

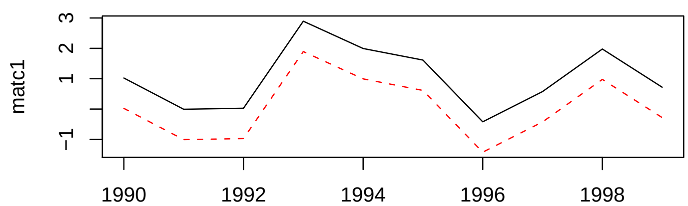
```
> TSget("matc2", con)
```

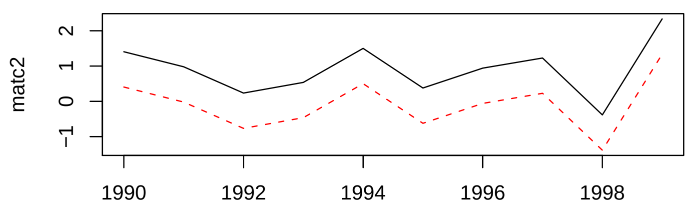
```
Time Series:
Start = 1990
End = 1999
Frequency = 1
          1           2           3           4           5           6 
 0.02185015  1.24501507  0.12831757  0.57926033  0.31459968  0.78007230 
          7           8           9          10 
 0.56418185 -0.95821180 -1.40464453 -0.04522594 
attr(,"seriesNames")
[1] matc2
attr(,"TSmeta")
<S4 Type Object>
attr(,"serIDs")
[1] "matc2"
attr(,"dbname")
[1] "test"
attr(,"con")
[1] "TSSQLiteConnection"
attr(,"con")attr(,"package")
[1] "TSSQLite"
attr(,"ExtractionDate")
[1] NA
attr(,"TSdescription")
[1] ""
attr(,"TSdoc")
[1] ""
attr(,"class")
[1] "TSmeta"
attr(,"class")attr(,"package")
[1] "TSdbi"
```

```
> TSget(c("matc1", "matc2"), con)

Time Series:
Start = 1990
End = 1999
Frequency = 1
          matc1       matc2
1990 -0.2314490  0.02185015
1991  0.8417985  1.24501507
1992 -1.8754379  0.12831757
1993 -1.4759967  0.57926033
1994  0.2200534  0.31459968
1995 -0.3295805  0.78007230
1996  0.4239013  0.56418185
1997 -0.9176695 -0.95821180
1998 -1.3072813 -1.40464453
1999 -0.4221239 -0.04522594
attr(,"seriesNames")
[1] matc1 matc2
attr(,"TSmeta")
<S4 Type Object>
attr(,"serIDs")
[1] "matc1" "matc2"
attr(,"dbname")
[1] "test"
attr(,"con")
[1] "TSSQLiteConnection"
attr(,"con")attr(,"package")
[1] "TSSQLite"
attr(,"ExtractionDate")
[1] NA
attr(,"TSdescription")
[1] ""
attr(,"TSdoc")
[1] ""
attr(,"class")
[1] "TSmeta"
attr(,"class")attr(,"package")
[1] "TSdbi"

> tfplot(z + 1, TSget(c("matc1", "matc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```






```
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 4)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)
```

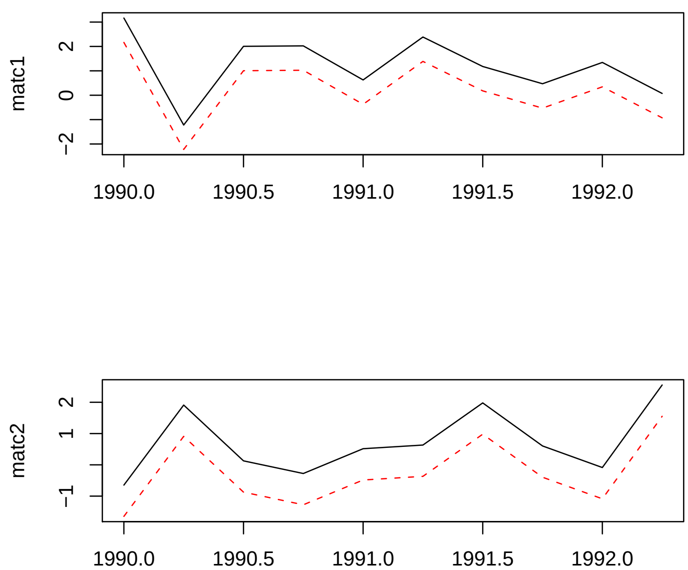
```
[1] TRUE
```

```
> TSget(c("matc1", "matc2"), con)
```

```
              matc1      matc2
1990 Q1  0.83964258  0.9803267
1990 Q2 -1.95732333 -0.5905428
1990 Q3 -0.05614891  0.5189691
1990 Q4 -0.84199105  0.4176482
1991 Q1  0.21767972 -0.8092528
1991 Q2  2.83235877 -1.1400437
1991 Q3 -0.78726026  0.7787594
1991 Q4 -0.21891313  1.3110374
1992 Q1  0.60959455  1.2810009
1992 Q2  0.69211773 -1.1654346
attr(,"seriesNames")
[1] matc1 matc2
attr(,"TSmeta")
<S4 Type Object>
```

```
attr(,"serIDs")
[1] "matc1" "matc2"
attr(,"dbname")
[1] "test"
attr(,"con")
[1] "TSSQLiteConnection"
attr(,"con")attr(,"package")
[1] "TSSQLite"
attr(,"ExtractionDate")
[1] NA
attr(,"TSdescription")
[1] ""
attr(,"TSdoc")
[1] ""
attr(,"class")
[1] "TSmeta"
attr(,"class")attr(,"package")
[1] "TSdbi"
```

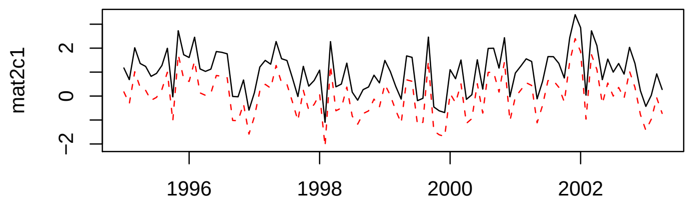
```
> tfplot(z + 1, TSget(c("matc1", "matc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```

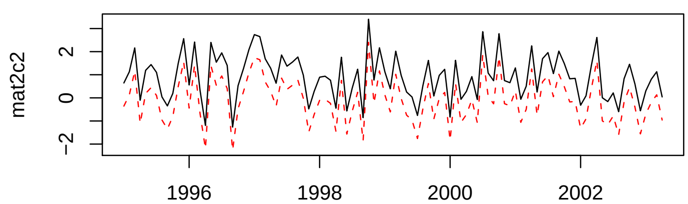


```
> z <- ts(matrix(rnorm(200), 100, 2), start = c(1995, 1), frequency = 12)
> seriesNames(z) <- c("mat2c1", "mat2c2")
> TSreplace(z, con)

[1] TRUE

> tfplot(z + 1, TSget(c("mat2c1", "mat2c2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```





The following extract information about the series from the database, although not much information has been added for these examples.

```
> TSmeta("mat2c1", con)
> TSmeta("vec", con)
> TSdates("vec", con)
> TSdescription("vec", con)
> TSdoc("vec", con)
```

Below are exampoles that make more use of *TSdescription* and codeTSdoc. Often it is convenient to set the default connection:

```
> options(TSconnection = con)
```

and then the *con* specification can be omitted from the function calls unless another connection is needed. The *con* can still be specified, and some examples below do specify it, just to illustrate the alternative syntax.

```
> z <- TSget("mat2c1")
> TSmeta("mat2c1")
```

```
serIDs:  mat2c1  from dbname:  test
description:
documentaion:
```

Data documentation can be in two forms, a description specified by *TSdescription* or longer documentation specified by *TSdoc*. These can be added to the time series object, in which case they will be written to the database when *TSput* or *TSreplace* is used to put the series on the database. Alternatively, they can be specified as arguments to *TSput* or *TSreplace*. The description or documentation will be retrieved as part of the series object with *TSget* only if this is specified with the logical arguments *TSdescription* and *TSdoc*. They can also be retrieved directly from the database with the functions *TSdescription* and *TSdoc*.

```
> z <- ts(matrix(rnorm(10), 10, 1), start = c(1990, 1), frequency = 1)
> TSreplace(z, serIDs = "Series1", con)
```

```
[1] TRUE
```

```
> zz <- TSget("Series1", con)
> TSreplace(z, serIDs = "Series1", con, TSdescription = "short rnorm series",
      TSdoc = "Series created as an example in the vignette.")
```

```
[1] TRUE
```

```
> zz <- TSget("Series1", con, TSdescription = TRUE, TSdoc = TRUE)
> start(zz)
```

```
[1] 1990    1
```

```
> end(zz)
```

```
[1] 1999    1
```

```
> TSdescription(zz)
```

```
[1] "short rnorm series"
```

```
> TSdoc(zz)
```

```
[1] "Series created as an example in the vignette."
```

```
> TSdescription("Series1", con)
```

```
[1] "short rnorm series"
```

```
> TSdoc("Series1", con)
```

```
[1] "Series created as an example in the vignette."
```

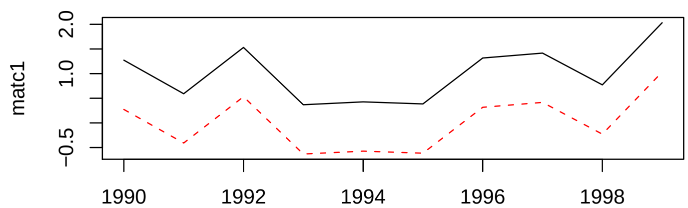
```
> z <- ts(rnorm(10), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- "vec"
> TSreplace(z, con)
```

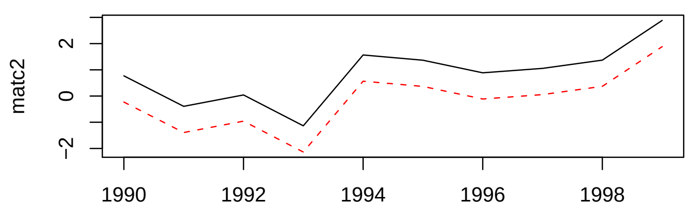
```
[1] TRUE
```

```
> zz <- TSget("vec", con)
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)
```

```
[1] TRUE
```

```
> tfplot(z + 1, TSget(c("matc1", "matc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```

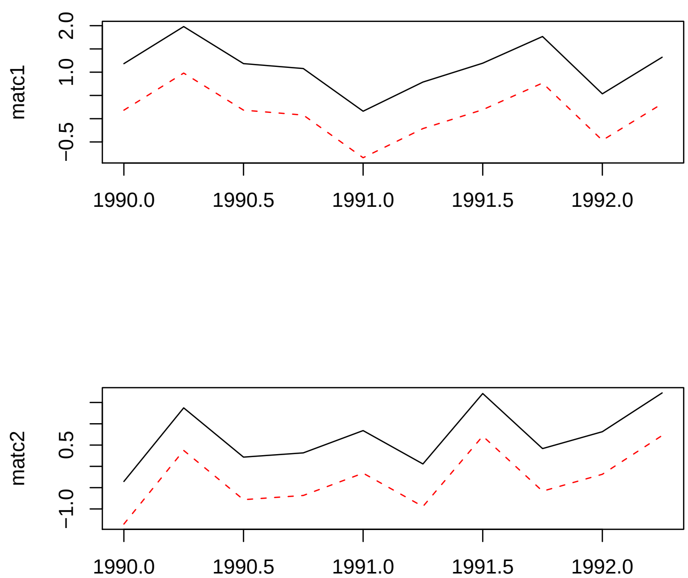





```
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 4)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)
```

```
[1] TRUE
```

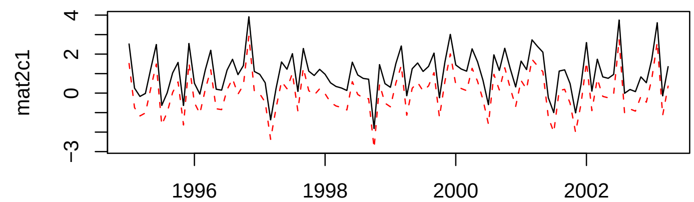
```
> tfplot(z + 1, TSget(c("matc1", "matc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```

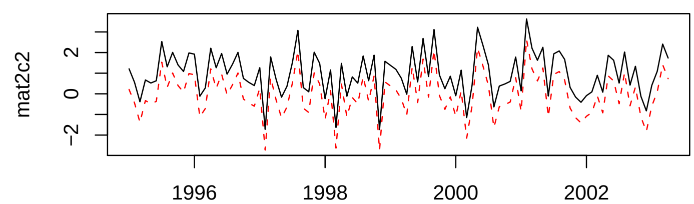



```
> z <- ts(matrix(rnorm(200), 100, 2), start = c(1995, 1), frequency = 12)
> seriesNames(z) <- c("mat2c1", "mat2c2")
> TSreplace(z, con)

[1] TRUE

> tfplot(z + 1, TSget(c("mat2c1", "mat2c2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```



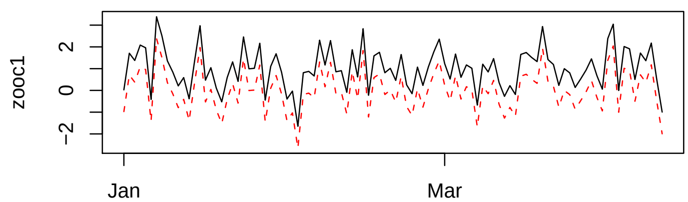


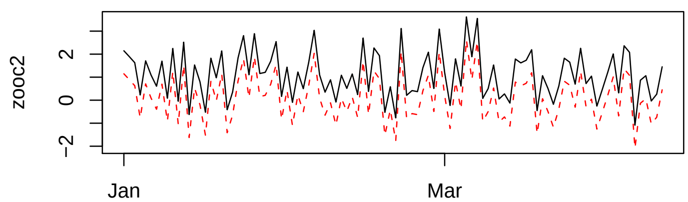
The following examples use dates and times which are not handled by *ts*, so the *zoo* time representation is used.

```
> require("zoo")
> z <- zoo(matrix(rnorm(200), 100, 2), as.Date("1990-01-01") +
      0:99)
> seriesNames(z) <- c("zooc1", "zooc2")
> TSreplace(z, con, Table = "D")

[1] TRUE

> tfplot(z + 1, TSget(c("zooc1", "zooc2"), con), lty = c("solid",
      "dashed"), col = c("black", "red"))
```

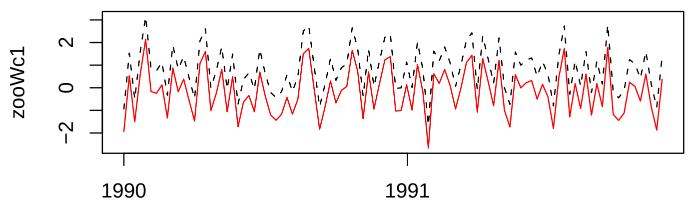


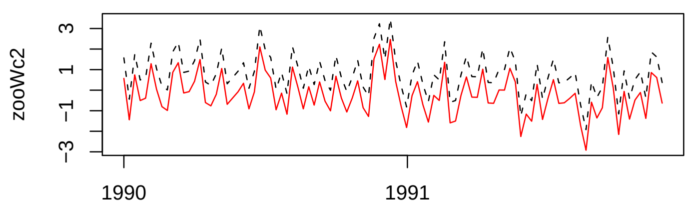



```
> z <- zoo(matrix(rnorm(200), 100, 2), as.Date("1990-01-01") +
      0:99 * 7)
> seriesNames(z) <- c("zooWc1", "zooWc2")
> TSreplace(z, con, Table = "W")

[1] TRUE

> tfplot(z + 1, TSget(c("zooWc1", "zooWc2"), con), col = c("black",
      "red"), lty = c("dashed", "solid"))
```






```
> dbDisconnect(con)
```

2.1 Examples Using TSdbi with ets

The database called "ets" is available at the Bank of Canada. These examples are illustrated in the *TSMySQL* and *TSpadi* packages, but ets is not yet implemented under *TSSQLite*.

3 Examples Using get.hist.quote

This section illustrates fetching data from elsewhere and loading it into the database. This would be a very slow way to load a database, but provides examples of different kinds of time series data.

The fetches are wrapped in *try()* and a flag *quote.ok* set because the fetch attempt may fail due to lack of an Interenet connection or delays.

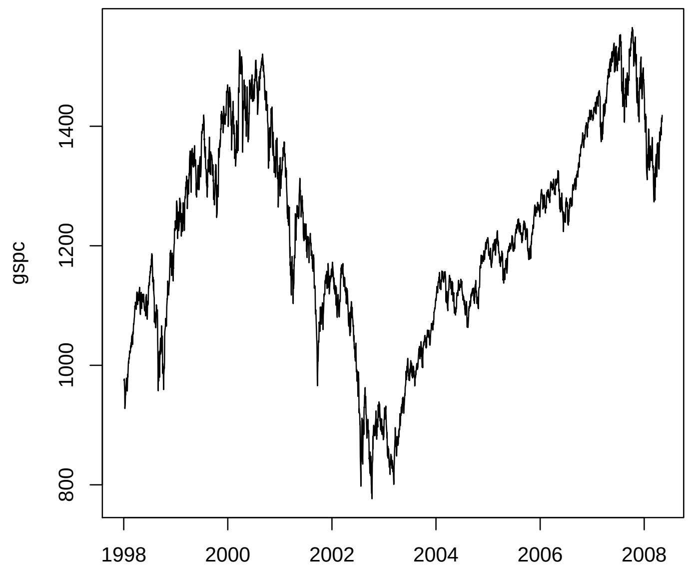
```
> con <- TSconnect(dbDriver("SQLite"), dbname = "test")
> options(TSconnection = con)
```

```
> require("tseries")
> quote.ok <- !inherits(try(x <- get.hist.quote(instrument = "^gspc",
      start = "1998-01-01", quote = "Close"), silent = TRUE), "try-error")
```

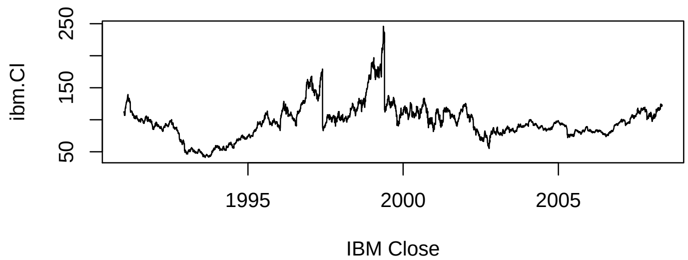
```
time series starts 1998-01-02
```

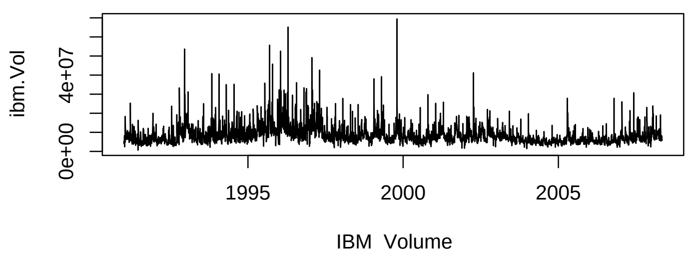
```
> if (quote.ok) plot(x)

> if (quote.ok) {
      TSrefPeriod(x) <- "Close"
      TSreplace(x, serIDs = "gspc", Table = "B", TSdescription. = "gspc  Close",
          TSdoc. = paste("gspc  Close retrieved with get.hist.quote on ",
              Sys.Date()))
      tfplot(TSget(serIDs = "gspc"))
  }
```

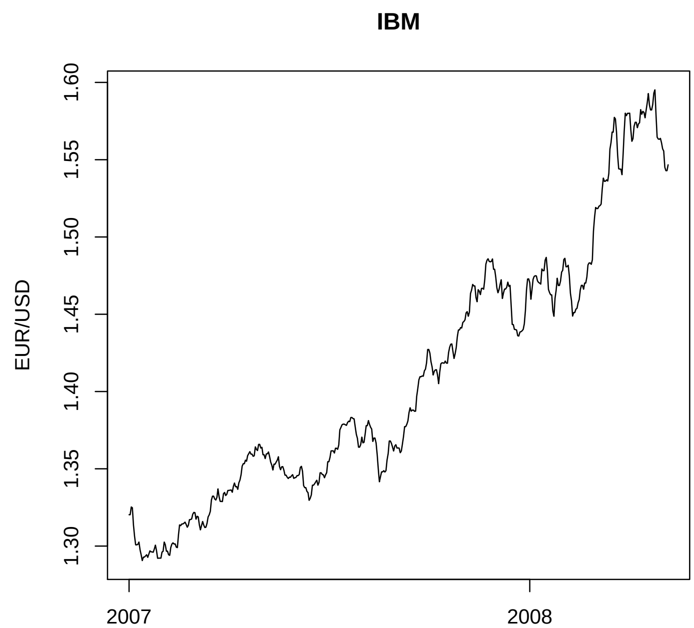



```
> quote.ok <- !inherits(try(x <- get.hist.quote(instrument = "ibm",
      quote = c("Cl", "Vol"))), "try-error")
> if (quote.ok) {
      TSreplace(x, serIDs = c("ibm.Cl", "ibm.Vol"), Table = "B",
          TSdescription. = c("IBM Close", "IBM  Volume"), TSdoc. = paste(c("IBM  Close retri
              "IBM  Volume retrieved with get.hist.quote on "),
              Sys.Date()))
      z <- TSget(serIDs = c("ibm.Cl", "ibm.Vol"), TSdescription = TRUE)
      tfplot(z, xlab = TSdescription(z))
  }
```

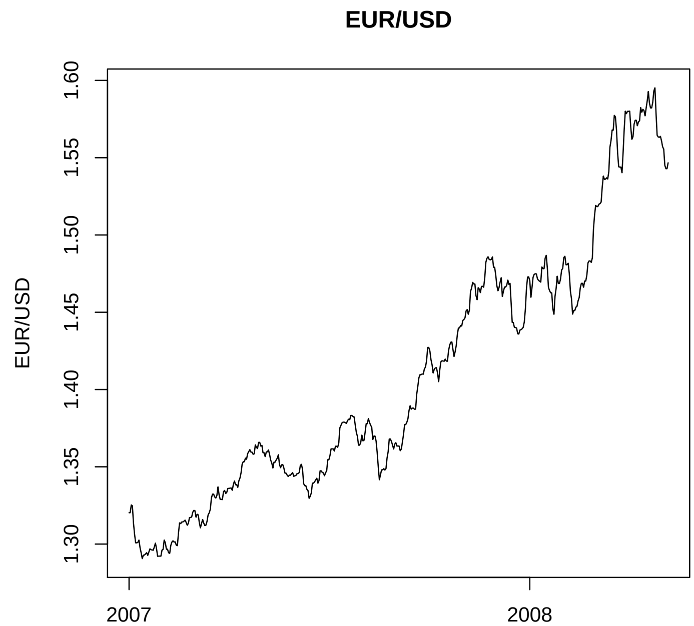




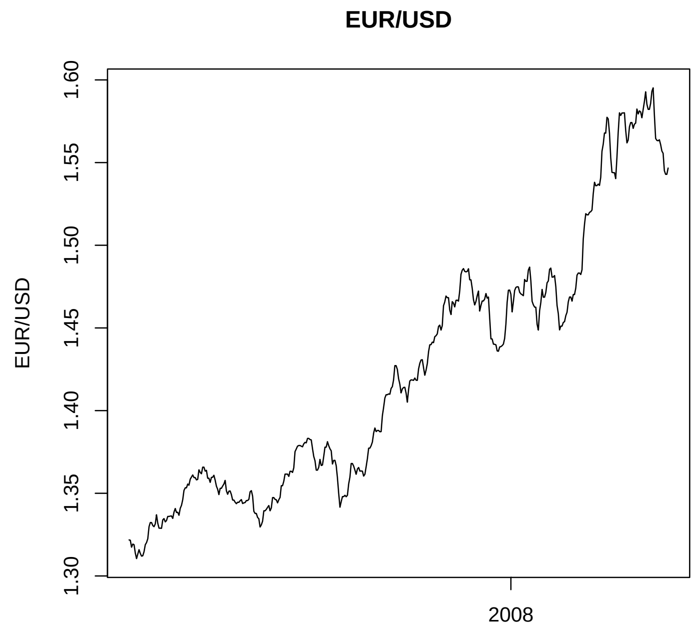

```
> if (quote.ok) {
      tfplot(z, Title = "IBM", start = "2007-01-01")
  }
> quote.ok <- !inherits(try(x <- get.hist.quote(instrument = "EUR/USD",
      provider = "oanda", start = "2004-01-01")), "try-error")
> if (quote.ok) {
      TSreplace(x, serIDs = "EUR/USD", Table = "D")
      z <- TSget(serIDs = "EUR/USD")
      tfplot(z, Title = "EUR/USD")
  }
```



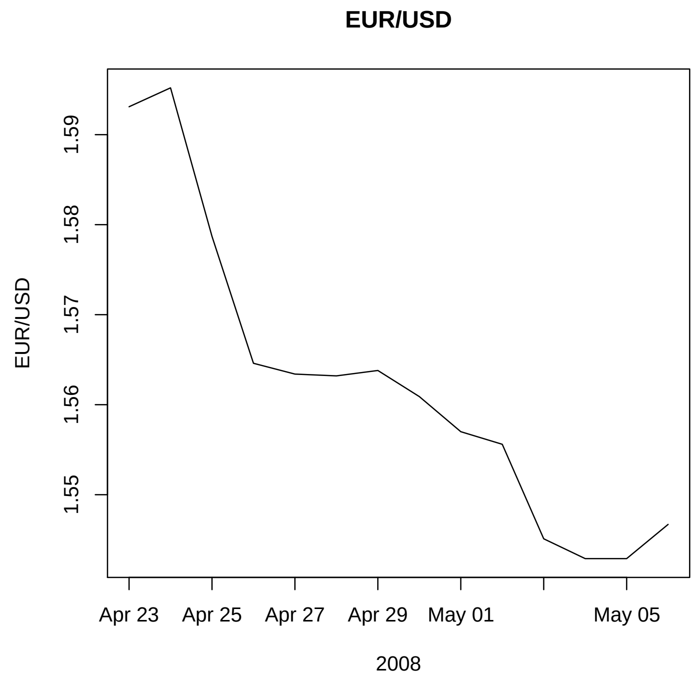

```
> if (quote.ok) {
      tfplot(z, Title = "EUR/USD", start = "2007-01-01")
  }
```




```
> if (quote.ok) {
      tfplot(z, Title = "EUR/USD", start = "2007-03-01")
  }
```




```
> if (quote.ok) {
      tfplot(z, Title = "EUR/USD", start = Sys.Date() - 14, end = Sys.Date(),
          xlab = format(Sys.Date(), "%Y"))
  }
```




```
> dbDisconnect(options()$TSconnection)
> options(TSconnection = NULL)
```

4 Examples Using DBI and direct SQL Queries

The following examples are queries using the underlying "DBI" functions. They should not often be needed to access time series, but may be useful to get at more detailed information, or formulate special queries.

```
> m <- dbDriver("SQLite")
> con <- TSconnect(m, dbname = "test")
> options(TSconnection = con)
```

```
> dbListTables(con)
```

```
 [1] "A"    "B"    "D"    "I"    "M"    "Meta" "Q"    "S"    "T"    "U"
[11] "W"
```

If schema queries are supported then table information can be obtained in a (almost) generic SQL way. On some systems this will fail because users do not have read priveleges on the INFORMATION_SCHEMA table. This does not seem to be an issue in SQLite, but I have not figured out the SQLite implementation so the following are wrapped in *try()*.

Table 1: Data Tables

Table	Contents
Meta	meta data and index to series data tables
A	annual data
Q	quarterly data
M	monthly data
S	semiannual data
W	weekly data
D	daily data
B	business data
U	minutely data
I	irregular data with a date
T	irregular data with a date and time

```
> try(dbGetQuery(con, paste("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.Columns ",
      " WHERE TABLE_SCHEMA='test' AND table_name='A' ;")))
> try(dbGetQuery(con, paste("SELECT COLUMN_NAME, COLUMN_DEFAULT, COLLATION_NAME, DATA_TYPE,"
      "CHARACTER_SET_NAME, CHARACTER_MAXIMUM_LENGTH, NUMERIC_PRECISION",
      "FROM INFORMATION_SCHEMA.Columns WHERE TABLE_SCHEMA='test' AND table_name='A' ;")))
> try(dbGetQuery(con, paste("SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, NUMERI
      "FROM INFORMATION_SCHEMA.Columns WHERE TABLE_SCHEMA='test' AND table_name='M';")))
```

Finally, to disconnect gracefully, one should

```
> dbDisconnect(options()$TSconnection)
> options(TSconnection = NULL)
```

5 Administration: Database Table Setup

The instructions in this section can be done in R using instructions in the file CreateTables.TSsql in the *TSdbi* package (distributed in TSdbi/inst/TSsql/). A simple way to do this was illustrated in the Introduction. Below the plain SQL instruction are shown. These could be executed in the mysql standalone client. This might be convenient when bulk loading data. (Example makefiles might sometime be available from the author.)

The database tables are shown in the Data Tables table. The *Meta* table is used for storing meta data about series, such as a description and longer documentation, and also includes an indication of what table the series data is stored in. To retrieve series it is not necessary to know which table the series is on, since this can be found on the *Meta* table. Putting data on the database may require specifying the table, if it cannot be determined from the R representation of the series.

The tables can be set up with the following commands. (Please note that this documentation is not automatically maintained, and could become out-of-date. The instructions in the file TSsql/CreateTables.TSsql are tested automatically, and thus guaranteed to be current.)

```
DROP TABLE IF EXISTS Meta;

create table Meta (
   id          VARCHAR(40) NOT NULL,
   tbl         CHAR(1),
   refPeriod   VARCHAR(10) default NULL,
   description   TEXT,
   documentation     TEXT,
   PRIMARY KEY (id)
   );

DROP TABLE IF EXISTS A;

create table A (
   id       VARCHAR(40),
   year       INT,
   v    double DEFAULT  NULL
   );

DROP TABLE IF EXISTS B;

create table B (
   id       VARCHAR(40),
   date       DATE,
   period     INT,
   v    double DEFAULT  NULL
   );

DROP TABLE IF EXISTS D;

create table D (
   id       VARCHAR(40),
   date       DATE,
   period     INT,
   v    double DEFAULT  NULL
   );

DROP TABLE IF EXISTS M;

create table M (
   id       VARCHAR(40),
```

```
   year       INT,
   period     INT,
   v    double DEFAULT  NULL
   );

DROP TABLE IF EXISTS U;

create table U (
   id       VARCHAR(40),
   date       DATETIME,
   tz         VARCHAR(4),  #not tested
   period     INT,
   v    double DEFAULT  NULL
   );

DROP TABLE IF EXISTS Q;

create table Q (
   id       VARCHAR(40),
   year       INT,
   period     INT,
   v    double DEFAULT  NULL
   );

DROP TABLE IF EXISTS S;

create table S (
   id       VARCHAR(40),
   year       INT,
   period     INT,
   v    double DEFAULT  NULL
   );

DROP TABLE IF EXISTS W;

create table W (
   id       VARCHAR(40),
   date       DATE,
   period     INT,
   v    double DEFAULT  NULL
   );

DROP TABLE IF EXISTS I;

create table I (
   id       VARCHAR(40),
```

```
   date       DATE,
   v    double DEFAULT  NULL
   );

DROP TABLE IF EXISTS T;

create table T (
   id    VARCHAR(40),
   date  DATETIME,
   v     double DEFAULT  NULL
   );
```

Indexes can be generated as follows. (It may be quicker to load data before generating indices.)

```
 CREATE INDEX Metaindex_tbl ON Meta (tbl);

 CREATE INDEX Aindex_id     ON A (id);
 CREATE INDEX Aindex_year   ON A (year);
 CREATE INDEX Bindex_id     ON B (id);
 CREATE INDEX Bindex_date   ON B (date);
 CREATE INDEX Bindex_period ON B (period);
 CREATE INDEX Dindex_id     ON D (id);
 CREATE INDEX Dindex_date   ON D (date);
 CREATE INDEX Dindex_period ON D (period);
 CREATE INDEX Mindex_id     ON M (id);
 CREATE INDEX Mindex_year   ON M (year);
 CREATE INDEX Mindex_period ON M (period);
 CREATE INDEX Uindex_id     ON U (id);
 CREATE INDEX Uindex_date   ON U (date);
 CREATE INDEX Uindex_period ON U (period);
 CREATE INDEX Qindex_id     ON Q (id);
 CREATE INDEX Qindex_year   ON Q (year);
 CREATE INDEX Qindex_period ON Q (period);
 CREATE INDEX Sindex_id     ON S (id);
 CREATE INDEX Sindex_year   ON S (year);
 CREATE INDEX Sindex_period ON S (period);
 CREATE INDEX Windex_id     ON W (id);
 CREATE INDEX Windex_date   ON W (date);
 CREATE INDEX Windex_period ON W (period);
 CREATE INDEX Iindex_id     ON I (id);
 CREATE INDEX Iindex_date   ON I (date);

 CREATE INDEX Tindex_id   ON T (id);
 CREATE INDEX Tindex_date ON T (date);
```

In SQLite you can check table information (eg. table A) with

```
describe A;
```

In sqlite3 data might typically be loaded into a table with command like

```
LOAD DATA LOCAL INFILE 'A.csv' INTO TABLE A FIELDS TERMINATED BY ',';
.import 'A.csv'  A
```

Of course, the corresponding Meta table entries also need to be made.