Time Series Database Interface: R MySQL (TSMySQL)

May 6, 2008

1 Introduction

The code from the vignette that generates this guide can be loaded into an editor with *edit(vignette("TSMySQL"))*. 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("TSMySQL"))*.

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

The MySQL user, password, and hostname should be set in MySQL information file (.my.cnf) before starting R. Once R is started, the functions in this package are made available with

```
> library("TSMySQL")
```

This will also load required packages *TSdbi*, *DBI*, *RMySQL*, *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("MySQL")
> 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("MySQL")
> 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:

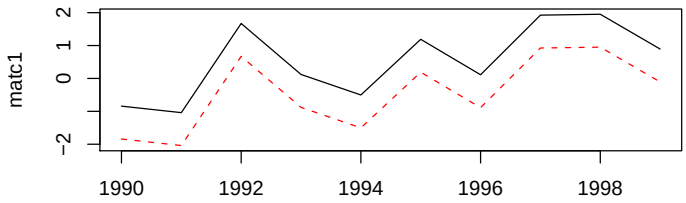
```
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)

[1] TRUE

> TSget("matc1", con)

Time Series:
Start = 1990
End = 1999
Frequency = 1
          1           2           3           4           5           6
 0.16062553  1.14802330 -0.81095699 -0.13845228  0.40714373 -1.69504982
          7           8           9          10
-0.73670465  1.12687156 -0.03235103  0.88615390
attr(,"seriesNames")
[1] matc1
attr(,"TSmeta")
<S4 Type Object>
attr(,"serIDs")
[1] "matc1"
attr(,"dbname")
[1] "test"
attr(,"con")
```

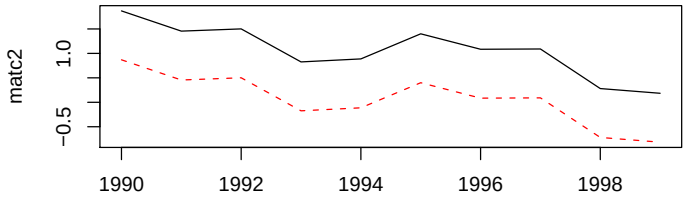
```
[1] "TSMySQLConnection"
attr(,"con")attr(,"package")
[1] "TSMySQL"
attr(,"ExtractionDate")
[1] NA
attr(,"TSdescription")
[1] ""
attr(,"TSdoc")
[1] ""
attr(,"class")
[1] "TSmeta"
attr(,"class")attr(,"package")
[1] "TSdbi"
```

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

```
Time Series:
Start = 1990
End = 1999
Frequency = 1
         1          2          3          4          5          6          7
-0.3085767  1.2415725  1.0322558  0.5488627 -0.8331084 -1.7596716 -1.3454376
         8          9         10
 0.3137934 -0.6600799  1.2910805
attr(,"seriesNames")
[1] matc2
attr(,"TSmeta")
<S4 Type Object>
attr(,"serIDs")
[1] "matc2"
attr(,"dbname")
[1] "test"
attr(,"con")
[1] "TSMySQLConnection"
attr(,"con")attr(,"package")
[1] "TSMySQL"
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  0.16062553 -0.3085767
1991  1.14802330  1.2415725
1992 -0.81095699  1.0322558
1993 -0.13845228  0.5488627
1994  0.40714373 -0.8331084
1995 -1.69504982 -1.7596716
1996 -0.73670465 -1.3454376
1997  1.12687156  0.3137934
1998 -0.03235103 -0.6600799
1999  0.88615390  1.2910805
attr(,"seriesNames")
[1] matc1 matc2
attr(,"TSmeta")
<S4 Type Object>
attr(,"serIDs")
[1] "matc1" "matc2"
attr(,"dbname")
[1] "test"
attr(,"con")
[1] "TSMySQLConnection"
attr(,"con")attr(,"package")
[1] "TSMySQL"
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"))
```

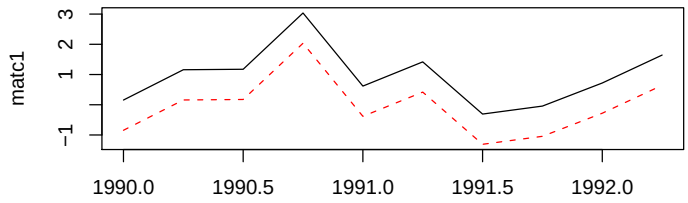


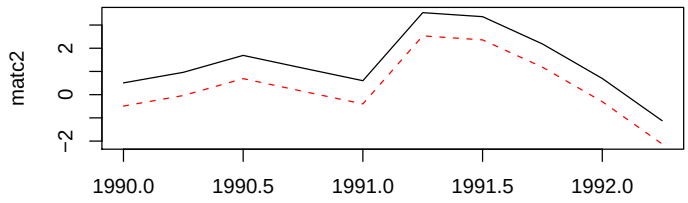



```
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 4)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)

[1] TRUE

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

              matc1      matc2
1990 Q1 -0.50259984  1.2108342
1990 Q2  0.08709998 -0.5601805
1990 Q3 -1.46044617  0.7505638
1990 Q4 -1.49648056 -2.1009415
1991 Q1 -0.45508241 -0.1910283
1991 Q2  0.36392303 -0.2352080
1991 Q3 -0.81303626  0.4901770
1991 Q4 -0.09937131 -0.9063318
1992 Q1  1.10022950  0.4355147
1992 Q2  1.03396799  2.5042559
attr(,"seriesNames")
[1] matc1 matc2
attr(,"TSmeta")
<S4 Type Object>
```

```
attr(,"serIDs")
[1] "matc1" "matc2"
attr(,"dbname")
[1] "test"
attr(,"con")
[1] "TSMySQLConnection"
attr(,"con")attr(,"package")
[1] "TSMySQL"
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"))
```

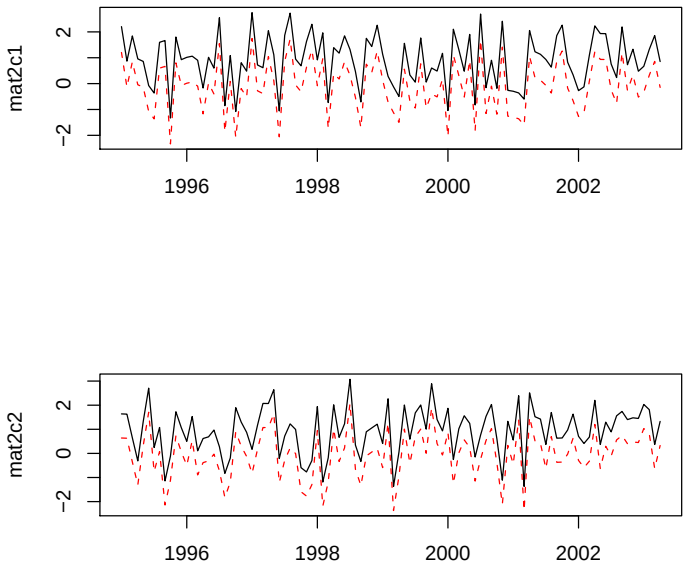




```
> z <- ts(matrix(rnorm(200), 100, 2), start = c(1995, 1), frequency = 12)
> seriesNames(z) <- c("mat2c1", "mat2c2")
> TSreplace(z, con)
```

```
[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."
```

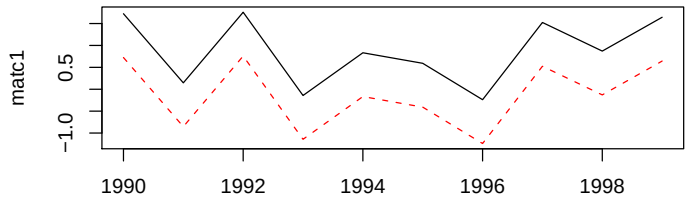
```
> z <- ts(rnorm(10), start = c(1990, 1), frequency = 1)
> seriesNames(z) <- "vec"
> TSreplace(z, con)
```

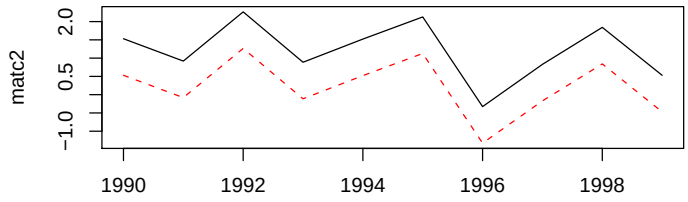
```
[1] TRUE
```

```
> 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"))
```

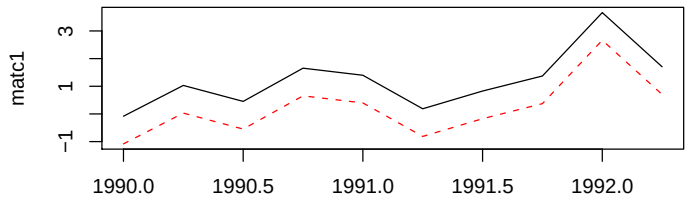


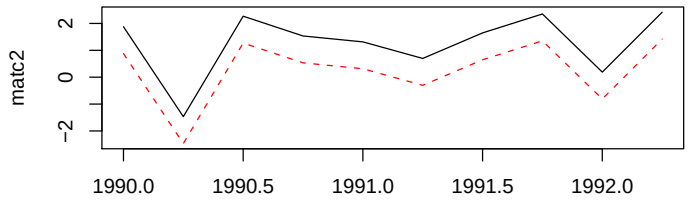



```
> z <- ts(matrix(rnorm(20), 10, 2), start = c(1990, 1), frequency = 4)
> seriesNames(z) <- c("matc1", "matc2")
> TSreplace(z, con)
```

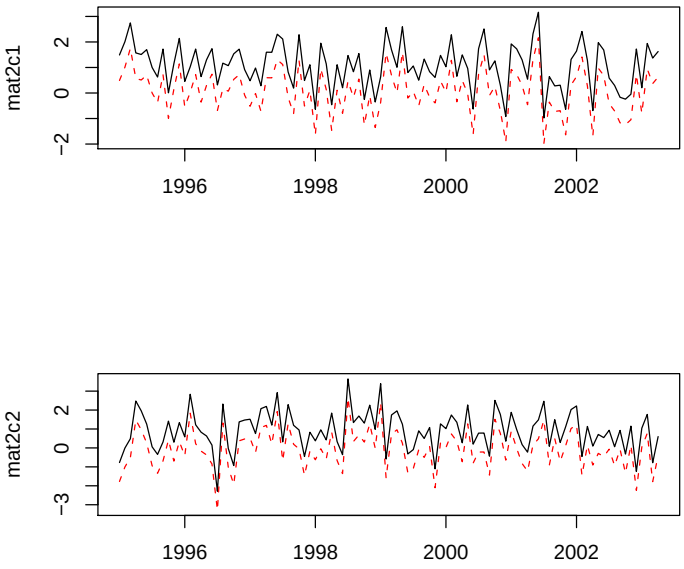
```
[1] TRUE
```

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



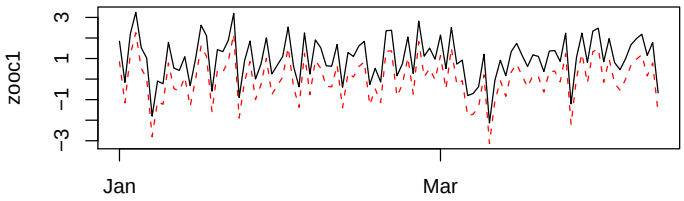


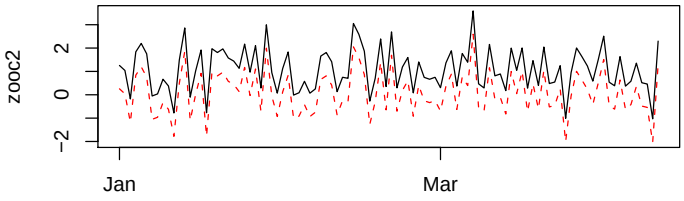

```
> z <- ts(matrix(rnorm(200), 100, 2), start = c(1995, 1), frequency = 12)
> seriesNames(z) <- c("mat2c1", "mat2c2")
> TSreplace(z, con)

[1] TRUE

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



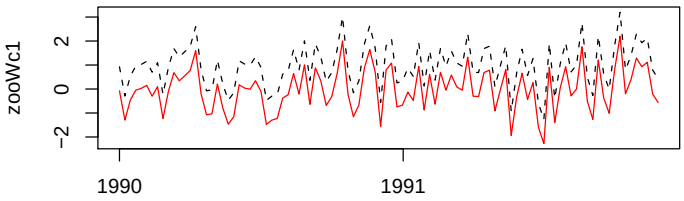
The following examples use dates and times which are not handled by *ts*, so the *zoo* time representation is used.

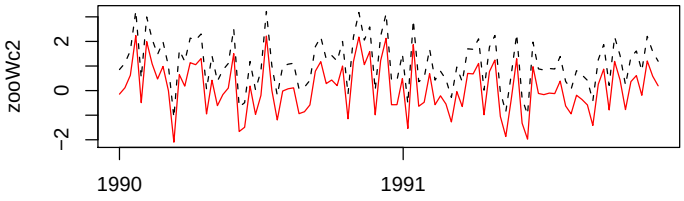
```
> require("zoo")
> z <- zoo(matrix(rnorm(200), 100, 2), as.Date("1990-01-01") +
      0:99)
> seriesNames(z) <- c("zooc1", "zooc2")
> TSreplace(z, con, Table = "D")

[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

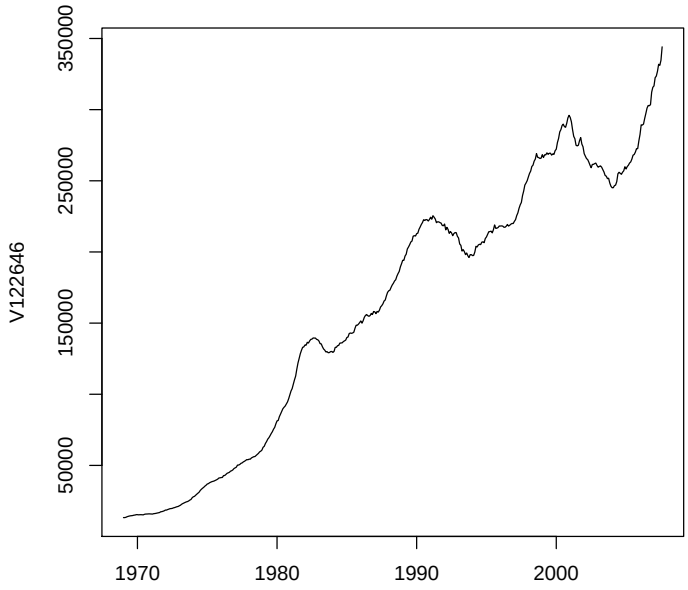
These examples use a database called "ets" which is available at the Bank of Canada. This set of examples illustrates how the programs might be used if a larger database is available. Typically a large database would be installed using database scripts directly rather than from R with *TSput* or *TSreplace*.

The following are wrapped in *if (!inherits(conets, "try-error"))* so that the vignette will build even when the database is not available. This seems to require an explicit call to *print()*, but that is not usually needed to display results below. Another artifact of this is that results printed in the if block do not display until the end of the block.

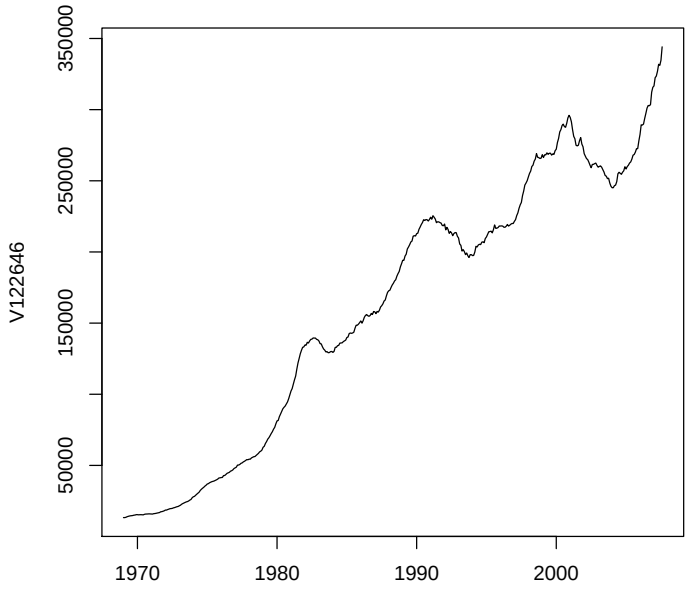
```
> m <- dbDriver("MySQL")
> conets <- try(TSconnect(m, dbname = "ets"))
> if (!inherits(conets, "try-error")) {
      options(TSconnection = conets)
      print(TSmeta("M.SDR.CCUSMA02.ST"))
      EXCH.IDs <- t(matrix(c("M.SDR.CCUSMA02.ST", "SDR/USD exchange rate",
          "M.CAN.CCUSMA02.ST", "CAN/USD exchange rate", "M.MEX.CCUSMA02.ST",
          "MEX/USD exchange rate", "M.JPN.CCUSMA02.ST", "JPN/USD exchange rate",
```

```
        "M.EMU.CCUSMA02.ST", "Euro/USD exchange rate", "M.OTO.CCUSMA02.ST",
        "OECD /USD exchange rate", "M.G7M.CCUSMA02.ST", "G7   /USD exchange rate",
        "M.E15.CCUSMA02.ST", "Euro 15. /USD exchange rate"),
        2, 8))
    print(TSdates(EXCH.IDs[, 1]))
    z <- TSdates(EXCH.IDs[, 1])
    print(start(z))
    print(end(z))
    tfplot(TSget(serIDs = "V122646", conets))
}

serIDs:  M.SDR.CCUSMA02.ST  from dbname  ets
description:  Special Drawing Right---Currency Conversions/US$ exchange rate/Average of dail
documentaion:  Special Drawing Right---Currency Conversions/US$ exchange rate/Average of dai
     [,1]
[1,] "M.SDR.CCUSMA02.ST from 1960 1 to 2007 9 M   NA   "
[2,] "M.CAN.CCUSMA02.ST from 1960 1 to 2007 9 M   NA   "
[3,] "M.MEX.CCUSMA02.ST from 1963 1 to 2007 9 M   NA   "
[4,] "M.JPN.CCUSMA02.ST from 1960 1 to 2007 9 M   NA   "
[5,] "M.EMU.CCUSMA02.ST from 1979 1 to 2007 9 M   NA   "
[6,] "M.OTO.CCUSMA02.ST  not available"
[7,] "M.G7M.CCUSMA02.ST  not available"
[8,] "M.E15.CCUSMA02.ST  not available"
[[1]]
[1] 1960    1

[[2]]
[1] 1960    1

[[3]]
[1] 1963    1

[[4]]
[1] 1960    1

[[5]]
[1] 1979    1

[[6]]
[1] NA

[[7]]
[1] NA

[[8]]
[1] NA
```

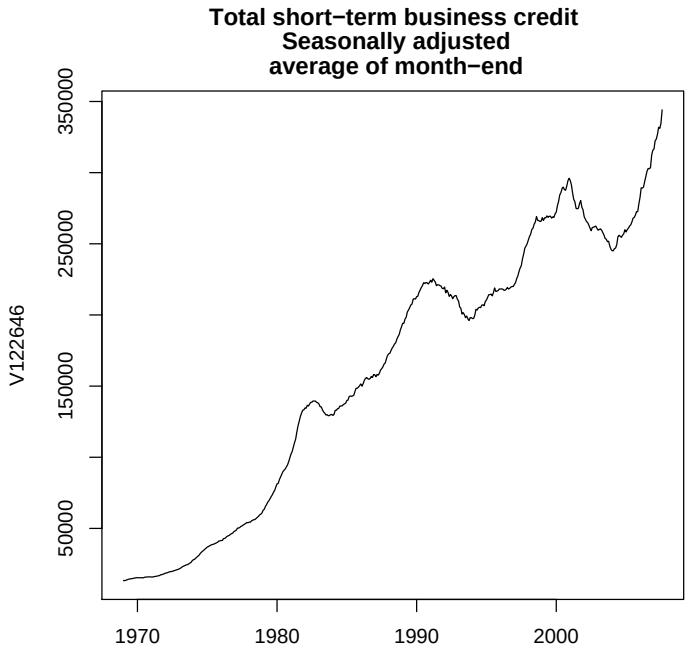
```
[[1]]
[1] 2007    9

[[2]]
[1] 2007    9

[[3]]
[1] 2007    9

[[4]]
[1] 2007    9

[[5]]
[1] 2007    9

[[6]]
[1] NA

[[7]]
[1] NA

[[8]]
[1] NA
```



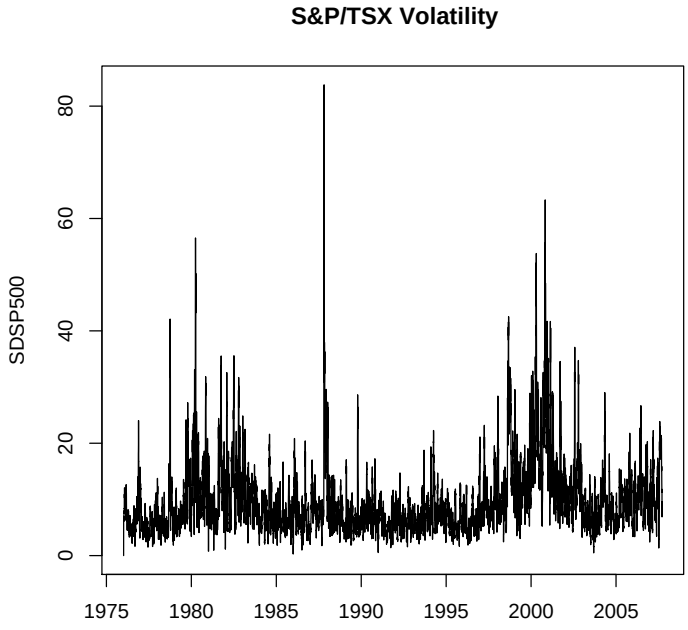

```
> if (!inherits(conets, "try-error")) {
      print(TSdescription(TSget("V122646", TSdescription = TRUE)))
      print(TSdescription("V122646"))
      print(TSdoc(TSget("V122646", TSdoc = TRUE)))
      print(TSdoc("V122646"))
      tfplot(TSget("V122646", names = "V122646", conets))
  }

[1] "Total short-term business credit, Seasonally adjusted, average of month-end"
[1] "Total short-term business credit, Seasonally adjusted, average of month-end"
[1] "Same as B171"
[1] "Same as B171"
```



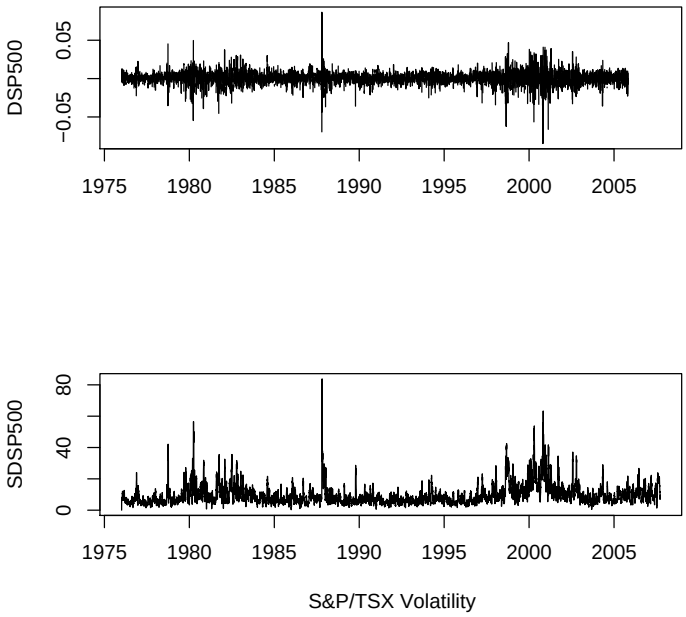

```
> if (!inherits(conets, "try-error")) {
      z <- TSget("V122646", TSdescription = TRUE)
      tfplot(z, Title = strsplit(TSdescription(z), ","))
  }
```



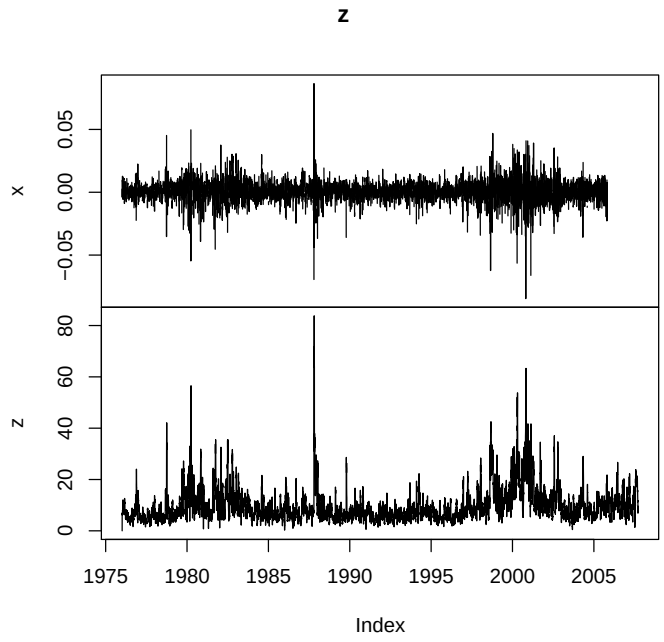

```
> if (!inherits(conets, "try-error")) {
      z <- TSget("SDSP500", TSdescription = TRUE)
      tfplot(z, Title = TSdescription(z))
      plot(z)
  }
```



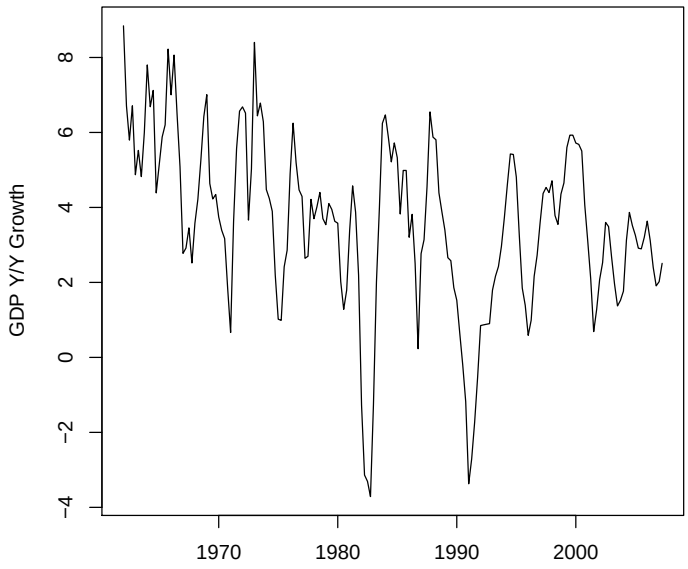

```
> if (!inherits(conets, "try-error")) {
      z <- TSget(c("DSP500", "SDSP500"), TSdescription = TRUE)
      tfplot(z, xlab = TSdescription(z))
  }
```




```
> if (!inherits(conets, "try-error")) {
      plot(z)
  }
```




```
> if (!inherits(conets, "try-error")) {
      ETSgdp <- annualizedGrowth(aggregate(TSget("V1992067"), nfrequency = 4,
          FUN = mean), lag = 4, names = "GDP Y/Y Growth")
      tfplot(ETSgdp)
  }
```




```
> if (!inherits(conets, "try-error")) {
      dbDisconnect(options()$TSconnection)
      options(TSconnection = NULL)
  }
```

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.

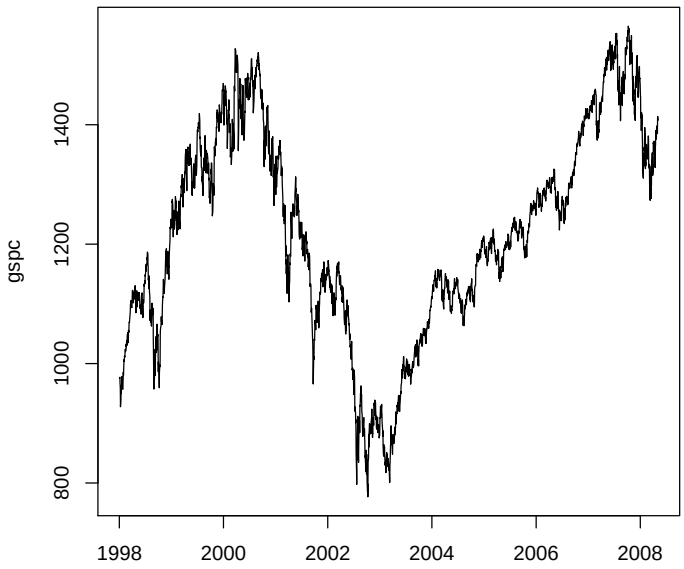
```
> con <- TSconnect(dbDriver("MySQL"), dbname = "test")
> options(TSconnection = con)
```

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

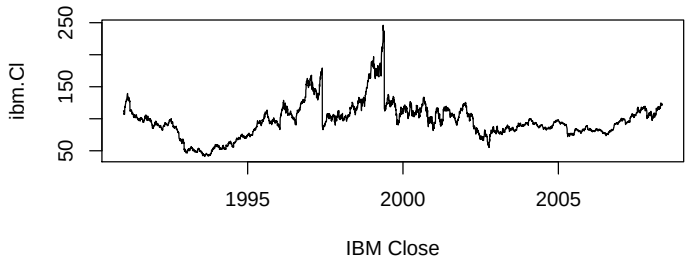
```
time series starts 1998-01-02
```

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

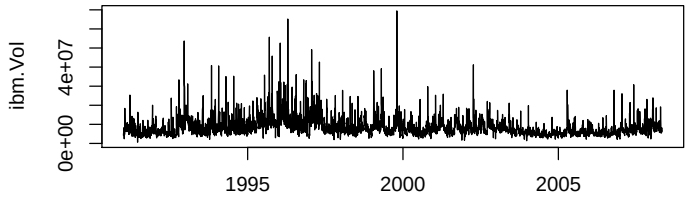
```
> if (quote.ok) {
      TSrefPeriod(x) <- "Close"
      TSreplace(x, serIDs = "gspc", Table = "B", TSdescription. = "gspc  Close",
          TSdoc. = paste("gspc  Close retrieved with get.hist.quote on ",
              Sys.Date()))
      tfplot(TSget(serIDs = "gspc"))
  }
```




```
> 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))
  }
```

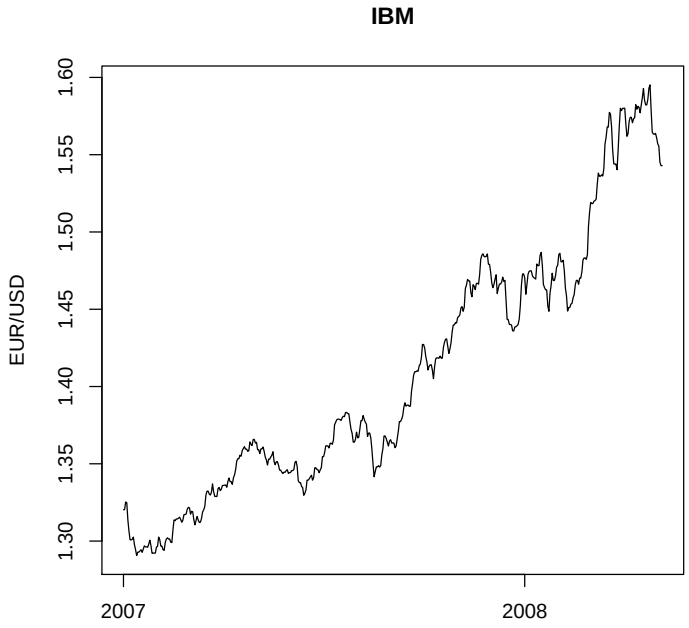


IBM Close

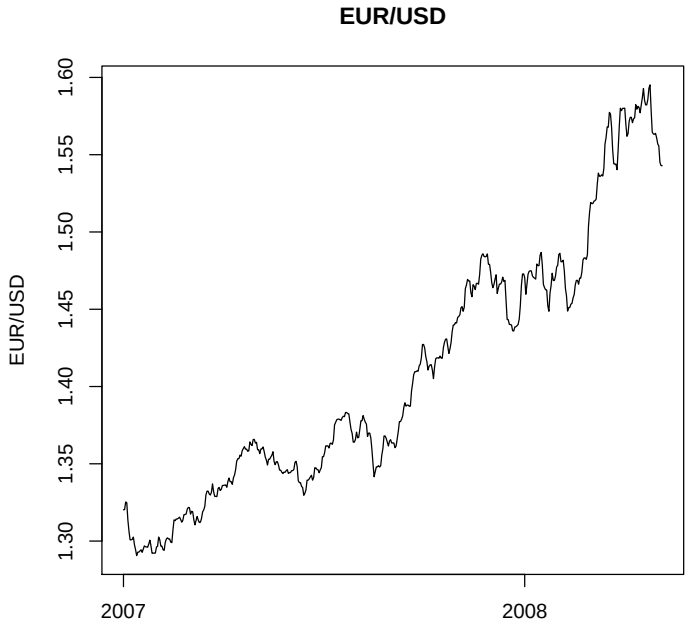


IBM Volume

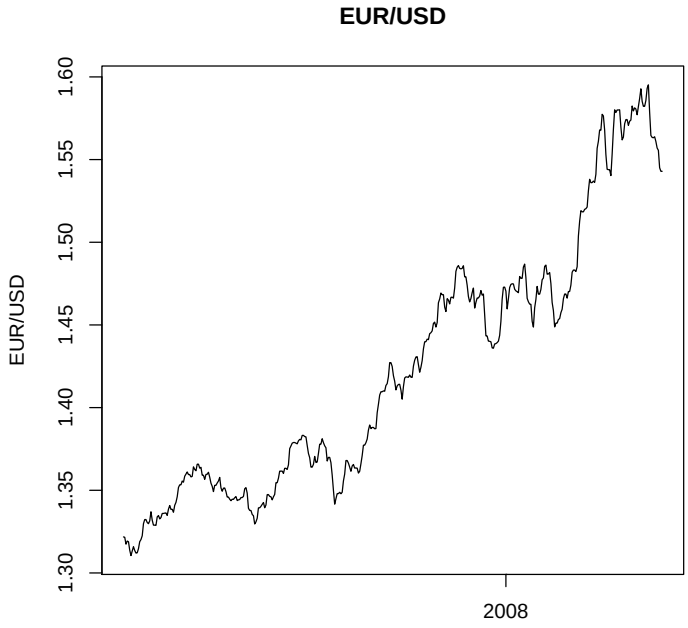
```
> if (quote.ok) {
      tfplot(z, Title = "IBM", start = "2007-01-01")
  }
> quote.ok <- !inherits(try(x <- get.hist.quote(instrument = "EUR/USD",
      provider = "oanda", start = "2004-01-01")), "try-error")
> if (quote.ok) {
      TSreplace(x, serIDs = "EUR/USD", Table = "D")
      z <- TSget(serIDs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



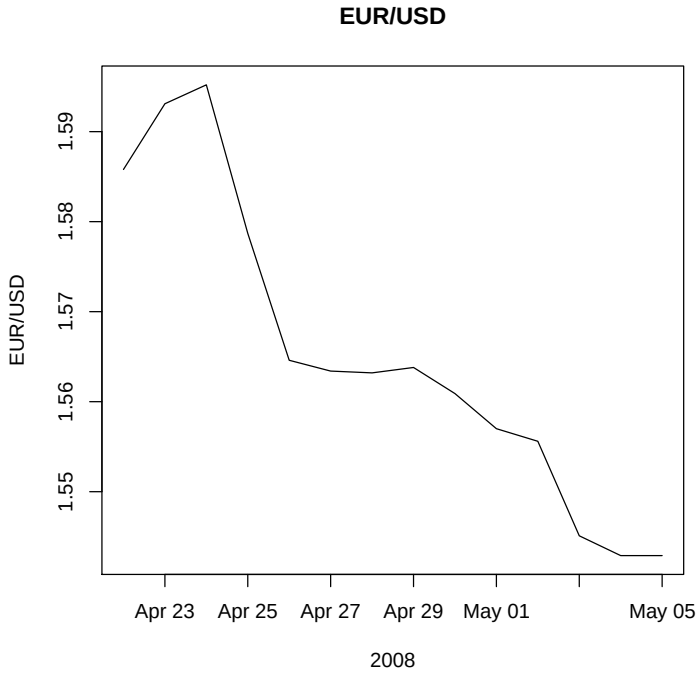

```
> if (quote.ok) {
      tfplot(z, Title = "EUR/USD", start = "2007-01-01")
  }
```




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

> dbListTables(con)

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

This is Mysql specific. Below is a generic sql way to do this.

```
> dbGetQuery(con, "show tables;")
```

```
   Tables_in_test
1               A
2               B
3               D
4               I
5               M
6            Meta
7               Q
8               S
9               T
10              U
11              W
```

```
> dbGetQuery(con, "describe A;")
```

```
  Field        Type Null Key Default Extra
1    id varchar(40)  YES MUL    <NA>
2  year     int(11)  YES MUL    <NA>
3     v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe B;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES MUL    <NA>
2   date        date  YES MUL    <NA>
3 period     int(11)  YES MUL    <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe D;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES MUL    <NA>
2   date        date  YES MUL    <NA>
3 period     int(11)  YES MUL    <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe M;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES MUL    <NA>
2   year     int(11)  YES MUL    <NA>
3 period     int(11)  YES MUL    <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe Meta;")
```

```
          Field        Type Null Key Default Extra
1            id varchar(40)   NO PRI
```

```
2           tbl     char(1)  YES MUL    <NA>
3     refPeriod varchar(10)  YES        <NA>
4   description        text  YES        <NA>
5 documentation        text  YES        <NA>
```

```
> dbGetQuery(con, "describe U;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES MUL    <NA>
2   date    datetime  YES MUL    <NA>
3     tz  varchar(4)  YES        <NA>
4 period     int(11)  YES MUL    <NA>
5      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe Q;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES MUL    <NA>
2   year     int(11)  YES MUL    <NA>
3 period     int(11)  YES MUL    <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe S;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES MUL    <NA>
2   year     int(11)  YES MUL    <NA>
3 period     int(11)  YES MUL    <NA>
4      v      double  YES        <NA>
```

```
> dbGetQuery(con, "describe W;")
```

```
   Field        Type Null Key Default Extra
1     id varchar(40)  YES MUL    <NA>
2   date        date  YES MUL    <NA>
3 period     int(11)  YES MUL    <NA>
4      v      double  YES        <NA>
```

If schema queries are supported then the above can be done in a generic SQL way, but on some systems this will fail because users do not have read priveleges on the INFORMATION_SCHEMA table, so the following are wrapped in *try()*. (SQLite does not seem to support this at all.)

```
> try(dbGetQuery(con, paste("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.Columns ",
      " WHERE TABLE_SCHEMA='test' AND table_name='A' ;")))
```

```
  COLUMN_NAME
1          id
2        year
3           v
```

```
> 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' ;")))

  COLUMN_NAME COLUMN_DEFAULT  COLLATION_NAME DATA_TYPE CHARACTER_SET_NAME
1          id           <NA> utf8_general_ci   varchar              utf8
2        year           <NA>            <NA>       int              <NA>
3           v           <NA>            <NA>    double              <NA>
  CHARACTER_MAXIMUM_LENGTH NUMERIC_PRECISION
1                       40                NA
2                       NA                10
3                       NA                22

> 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';")))

  COLUMN_NAME DATA_TYPE CHARACTER_MAXIMUM_LENGTH NUMERIC_PRECISION
1          id   varchar                       40                NA
2        year       int                       NA                10
3      period       int                       NA                10
4           v    double                       NA                22
```

Finally, to disconnect gracefully, one should

```
> dbDisconnect(con)
> 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 Table below. 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.)

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

```
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 MySQL you can check table information (eg. table A) with

```
describe A;
```

This is generic sql way to get table information but it requires read privileges on INFORMATION_SCHEMA.Columns which the user may not have. (And SQLite does not seem to support this at all.)

```
SELECT COLUMN_NAME, COLUMN_DEFAULT, COLLATION_NAME, DATA_TYPE,
       CHARACTER_SET_NAME, CHARACTER_MAXIMUM_LENGTH, NUMERIC_PRECISION
          FROM INFORMATION_SCHEMA.Columns WHERE table_name='A' ;
```

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

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

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