Time Series Database Interface (TSdbi)

May 9, 2017

1 TSdbi Functions

In R, the functions in this package are made available with

```
> library("TSdbi")
```

The *TSdbi* package provides the common parts of an interface to time series databases. To use this package it is necessary to also have one of the several extension packages which provide provide the interface to the underlying source. A complete vignette, illustrating the general functionality of all *TSdbi* extension packages is provided in the vignette with package *TSdata*. Installing that package requires that most of the *TS** packages are available, which will not be necessary for most users, so it may be easier to get the pdf version of the vignette from CRAN at `http://cran.r-project.org/web/packages/TSdata/index.html`.

2 SQL setup

The documentation below is intended for an administrator who needs to set up an SQL version of a database for local use. For many users this will not be necessary, as extensions other than the SQL ones will not require this setup (and the SQL ones setup a test database automatically).

The *TSdbi* interface works with some databases that are not SQL, but for SQL databases the instructions below provide details for setting up the backend database server tables.

3 Administration: Database Table Setup

The instructions to build SQL tables using R are given in the file CreateTables.TSsql distributed in TSdbi/inst/TSsql/ and for simple examples such as illustrated in the database specific packages it is adequate to simply

source(system.file("TSsql/CreateTables.TSsql", package="TSdbi")).

Below the plain SQL instruction are shown. In a few places MySQL specific commands are used, but the equivalent for other SQL variants should be fairly

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

clear to someone familiar with the SQL variant. The plain SQL instruction below can be executed in a standalone client, such as mysql, which might be convenient when bulk loading data. (Example makefiles for bulk loading data might eventually 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 in, 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.

In addition, there will be tables "vintages" and "panels" if those features are used. 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 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.