A high level interface to SQLite

Federico de Ceballos
Universidad de Cantabria

The SQLite interpreter

```
SQLite version 3.6.1
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> begin;
sqlite> create table episodes (id integer primary key,
   ...>                         season int,
   ...>                         name text );
sqlite> insert into episodes values(1, 1, 'male unbonding');
sqlite> insert into episodes values(2, 1, 'the stake out');
sqlite> create table foods (id integer primary key,
   ...>                     type_id integer,
   ...>                     name text );
sqlite> insert into foods values(1, 1, 'bagels');
sqlite> insert into foods values(2, 2, 'bavarian cream
pie');
sqlite> drop table foods;
sqlite> commit;
```

Some sample SQL code

```
begin;

create table episodes (id integer primary key,
                       season int,
                       name text);
insert into episodes values(1, 1, 'male unbonding');
insert into episodes values(2, 1, 'the stake out');

create table foods (id integer primary key,
                    type_id integer,
                    name text);
insert into foods values(1, 1, 'bagels');
insert into foods values(2, 2, 'bavarian cream pie');

drop table foods;

commit;
```

Types of words

- Some words are not used in Forth: **commit insert**
- Others are: **begin create drop**
- A few words may appear by themselves, without additional parameters, so the closing semicolon could be attached: **begin; commit;**

Getting results

After a normal query, we expect to receive a result set. This is usually printed on the screen.

Some words allow the user to choose the format used.

+headers	A header is produced
-headers	A header is not produced
mode-csv	Columns are separated by a string
mode-column	Columns are of a given width
mode-line	Each column is given in its own line
set-separator	Sets the string used as separator
set-null	Sets the string used for null values
set-widths	Sets the widths to be used for columns

Other possibilities

`mode-user`	A user function is called for each row, this function has to get the column values
`mode-stack`	A user function is called for each row with the column values already on the stack

```
: sample ( ) cr 1 get-text type ;

/sql

' sample mode-user

select * from my_table;

sql/
```

Using parameters

```
insert into foo values(?,?,?)";p

   [ 1                   1 int]
   [ s" pi"              2 text]
   [ 1e fatan 4e f* f.   3 float] ;p

   [ 2                   1 int]
   [ s" e"               2 text]
   [ 1e fexp             3 float] ;
```

The [word is used to “pop” out of SQL mode and into Forth mode, in a similar way as you are able to temporally leave compilation state to go to interpretation state.

Defining user functions

```
: sample ( ) 0 get-int 1 get-int + result-int ;
: sum    ( ) 0  #args 0 ?do  i get-int +  loop  result-int ;

/sql

' sample  2 def-function my_function
' sum    -1 def-function sum

select my_function(1,2);
select sum(),sum(1),sum(1,2);

sql/
```

Using code inside definitions

```
s" insert into episodes (id) values (?)" prepare
   30 1 bind-int
continue
   35 1 bind-int
conclude

:noname ( ) cr 1 get-int . ; is row

s" select * from episodes" process

s" insert into episodes (id) values (" >sq (.) +sq s" )" +sq
sq@ process
```

This is normal Forth code that can be used anywhere.

Future work

Test the code, complete the binding and make it public

Expand the system by a new set of functions

Move files into a database

Use the program as part of a course ?