sql egg

A small scheme library for constructing SQL queries.
Extension for Chicken Scheme
Version 1.0

Hans Bulfone

Table of Contents

1 About this egg 1

1.1 Version history 1

1.2 Usage 1

2 Documentation 2

2.1 Introduction 2

2.2 Notes 2

2.3 NULL object 2

2.4 Quoting strings 2

2.5 Transforming S-expressions to SQL 2

2.6 Utility functions for generating common SQL queries 4

3 Examples 5

4 License 6

Index 7

1 About this egg

1.1 Version history

`1.0`	Initial release
`1.1`	Documentation, some fixes/features

1.2 Usage

Load this egg like so:

```
(require-extension sql)
```

2 Documentation

2.1 Introduction

This extension provides procedures for constructing SQL queries from S-expressions.

No support for actually accessing a database is provided so this extension is meant to be used together with other extensions like `postgresql` or `mysql`.

2.2 Notes

- sql.egg is incomplete. It mostly contains what I've needed so far :)
- sql.egg has only been used with PostgreSQL so far.

2.3 NULL object

`sql:null` [procedure]

```
(sql:null)
```

Returns the NULL object

`sql:null?` [procedure]

```
(sql:null? X)
```

Returns `#t` if `X` is the NULL object, `#f` otherwise.

These functions can be redefined to recognize the NULL object of the database that is in use, for example:

```
(require-extension postgresql sql)
(define (sql:null) pg:sql-null-object)
(define sql:null? pg:sql-null-object?)
```

`'()` is also recognized as NULL by the sql egg, no matter how `sql:null?` is defined.

2.4 Quoting strings

`sql:quote` [procedure]

```
(sql:quote S)
```

Returns a copy of `S` with `'` replaced by `''` and \ replaced by \\.

2.5 Transforming S-expressions to SQL

`sql:transform` [procedure]

```
(sql:transform EXPR)
```

Returns the S-expression `EXPR` converted to SQL syntax as a list. The following transformation rules exist:

```
(and)                    ->                    TRUE
```

`(and x ...)`	`->`	`(x AND ...)`
`(or)`	`->`	`FALSE`
`(or x ...)`	`->`	`(x OR ...)`
`(not x)`	`->`	`(NOT x)`
`(null? x)`	`->`	`(x IS NULL)`
`(= x` *`(sql:null)`*`)`	`->`	`(x IS NULL)`
`(`*`binary-operator`*` x1 x2)`	`->`	`(x1 `*`op`*` x2)`
`(`*`n-ary-operator`*` x ...)`	`->`	`(x `*`op`*` ...)`
`(-> type x)`	`->`	`(x::type)`
`(as x alias)`	`->`	`x AS alias`
`(asc x)`	`->`	`x ASC`
`(desc x)`	`->`	`x DESC`
`(extract f s)`	`->`	`EXTRACT(f FROM s)`
`(substring s start count)`	`->`	`SUBSTRING(s FROM start FOR count)`
`(string-append x ...)`	`->`	`(x || ...)`
`(bitwise-and x ...)`	`->`	`(x & ...)`
`(bitwise-ior x ...)`	`->`	`(x | ...)`
`(bitwise-xor x ...)`	`->`	`(x # ...)`
`(bitwise-not x)`	`->`	`(~x)`
`(join/on type a b on)`	`->`	`(a type JOIN b ON on)`
`(join/using type a b (using1 u2 ...))`	`->`	`(a type JOIN b USING (using1,u2,...))`
`(join/natural type a b)`	`->`	`(a NATURAL type JOIN b)`

`(func x ...)`	`->`	`func(x,...)`
string	`->`	`'(quoted-string)'`
`#t`	`->`	`TRUE`
`#f`	`->`	`FALSE`
`(sql:null)`	`->`	`NULL`
`()`	`->`	`NULL`

`sql:list->string` [procedure]

`(sql:list->string L)`

Returns the elements of `L` concatenated as a string.

`sql:binary-operators` [parameter]

A list of binary operators recognized by `sql:transform`, by default `'(< > <= >= = <> != << >>)`.

`sql:n-ary-operators` [parameter]

A list of n-ary operators recognized by `sql:transform`, by default `'(+ - * /)`.

2.6 Utility functions for generating common SQL queries

`sql:select` [procedure]

`(sql:select WHAT FROM WHERE #!optional (ORDER-BY #f))`

Returns a SQL SELECT-query as a string. `WHAT` and `FROM` are lists of S-expressions. `WHERE` is an S-expression or `#f`. `ORDER-BY`, if given, is a list of S-expressions.

`sql:insert` [procedure]

`(sql:insert TABLE VALUES)`

Returns a SQL INSERT-query as a string. `TABLE` is a string or symbol naming the table to receive the data. `VALUES` is an alist mapping column names (symbols) to values (S-expressions)

`sql:delete` [procedure]

`(sql:delete TABLE WHERE)`

Returns a SQL DELETE-query as a string. `TABLE` is a string or symbol naming the table to modify. `WHERE` is either an S-expression specifying the rows to delete or `#f`.

`sql:update` [procedure]

`(sql:update TABLE UPDATES WHERE)`

Returns a SQL UPDATE-query as a string. `TABLE` is a string or symbol naming the table to modify. `UPDATES` is an alist mapping column names (symbols) to values (S-expressions), `WHERE` is either an S-expression specifying the rows to modify or `#f`.

3 Examples

```
$ csi
 )   ___
(__/_____) /)   ,    /)
  /       (/      _ (/_   _ __
 /        / )__(_(__/(___(/_/ (_
(______)
Version 2, Build 3 - linux-unix-gnu-x86 - [ libffi dload ptables ]
(c)2000-2005 Felix L. Winkelmann
#;1> (use sql)
; loading /usr/lib/chicken/sql.so ...
#;2> (define (get-some-data) "some-data")
#;3> (sql:insert "foobar" `((timestamp . (now)) (data . ,(get-some-data)) (quux . 5)))
"INSERT INTO foobar(timestamp,data,quux) VALUES(now(),'some-data',5)"
#;4> (sql:update "foobar" `((timestamp . (+ timestamp 5))) `(< quux 10))
"UPDATE foobar SET timestamp=(timestamp+5) WHERE (quux<10)"
#;5> (sql:select '(f.data quux.name) '((as foobar f) quux) '(and (= f.quux quux.bla) (>= f.t
"SELECT f.data,quux.name FROM foobar AS f,quux WHERE ((f.quux=quux.bla) AND (f.timestamp>=(n
#;6>
```

4 License

Index

`sql:binary-operators` 4
`sql:delete` 4
`sql:insert` 4
`sql:list->string` 4
`sql:n-ary-operators` 4
`sql:null` 2
`sql:null?` 2
`sql:quote` 2
`sql:select` 4
`sql:transform` 2
`sql:update` 4