

# sql egg

---

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

**Hans Bulfone**

---

# Table of Contents

<b>1</b>	<b>About this egg</b> .....	<b>1</b>
1.1	Version history .....	1
1.2	Usage .....	1
<b>2</b>	<b>Documentation</b> .....	<b>2</b>
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
<b>3</b>	<b>Examples</b> .....	<b>5</b>
<b>4</b>	<b>License</b> .....	<b>6</b>
	<b>Index</b> .....	<b>7</b>

# 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

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

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

`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.
"SELECT f.data,quux.name FROM foobar AS f,quux WHERE ((f.quux=quux.bla) AND (f.timestamp>=
#;6>

```

## 4 License

Copyright (c) 2006, Hans Bulfone  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- \* Neither the name of the author nor the names of his contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## Index

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