Database Access
& Making Low-level Libraries Rackety

Ryan Culpepper
PLT, University of Utah
iconoclast n.
1. One who destroys sacred images.
2. One who attacks and seeks to overthrow traditional or popular ideas or institutions. *The American Heritage Dictionary* Wordsmith

Q: What is the shortest lie in computing? A: It works.

On Programming: A bad day writing code in Scheme is better than a good day writing code in C. -- David Stigant

More on Programming: Always code as if the guy who ends up maintaining your code will be violent psychopath who knows where you live. -- John F. Woods

On LISP: In twenty years no one will be programming in anything but LISP. -- Piet Hut, IAS Princeton. Who Got Einstein’s Office. Ed Regis, pg 166. It’s 2001, so there’s hope. -- Shriram Krishnamurthi

On the Quality of Shrink Wrap
## Comp312: Homework Projects

<table>
<thead>
<tr>
<th>Due Date</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 23</td>
<td>Creating Web Content</td>
</tr>
<tr>
<td>Jan 30</td>
<td>Processing XML</td>
</tr>
<tr>
<td>Feb 06</td>
<td>Creating Web Content Dynamically</td>
</tr>
<tr>
<td>Feb 13</td>
<td>Processing Web Content</td>
</tr>
<tr>
<td>Feb 20</td>
<td>Link Checking Web Sites</td>
</tr>
<tr>
<td>Feb 27</td>
<td>A Gas Station Simulator</td>
</tr>
<tr>
<td>Mar 13</td>
<td>Extending the Gas Station Simulator</td>
</tr>
<tr>
<td>Mar 20</td>
<td>A New Look for the Gas Station Simulator</td>
</tr>
<tr>
<td>Apr 03</td>
<td>Correctness Proofs</td>
</tr>
<tr>
<td>Apr 10</td>
<td>Using Tools</td>
</tr>
<tr>
<td>Apr 17</td>
<td>Building a Web Server</td>
</tr>
<tr>
<td>Apr 24</td>
<td>Extending a Web Server</td>
</tr>
</tbody>
</table>

[312 Home](http://www.cs.rice.edu/~matthias/312/Homework/)
Creating Web Contents Dynamically

Your task is to develop a CGI script that helps people order a political commentary of their liking. The dialog consists of three steps:

- a welcome page that asks for the login name, the password, and the current preference;
- a page that displays the last four digits of the credit cards used in the past and that gives the consumer the option of submitting a new credit card number; and
- a confirmation page that summarizes the dialog.

Consider the following example:

welcome                            credit card choice                            goo bye

The color of the page changes as the consumer progresses through the dialogue.

If the consumer chooses to use a new credit card number, the program must know about the new number when the consumer logs in again. You may store the login names, passwords, and credit card numbers in a file. Do not worry about security concerns.

If at any point in the dialog something goes wrong, the program should display a red page with white text:

warning

The warning here explains what the consumer might have done wrong and provides a button for starting over.

Notes:

- You program must be started with a URL that runs to a CGI script.
- You are free to choose the exact words, the exact layout of the page, the colors (except for red), etc.
- Don't overstep your freedom. Your program must conform to the functional specifications stated above.
<?php

// String, String, String --> String
function handle_payment_main($username, $password, $wing) {
    $db = pg_connect("", ",", ",", ",", "comp312");

    $login_sql = <<<END_SQL
        select password from users
        where username = '$username';
END_SQL;

    $login_query = pg_exec($db, $login_sql);
    $login_numrows = pg_numrows($login_query);

    if ($login_numrows == -1) {
        // Error.
        echo "Whoops, error on query <b>$sql</b>";
        exit();
    }

    if ($login_numrows == 0) {
        // The user was not found. We will add them and continue.
        add_user($db, $username, $password);
    } else {
        // Get the (sole) row
        $login_row = pg_fetch_array($login_query, 0);
        $real_password = $login_row['password'];

        if ($password == $real_password) {
            // Okay. Continue with script.

        } else {
            11
        }
    }
}
Q:
Why did you do this in *php*?
Q: Why did you do this in **php**?

A: Because a database was the right tool for the job.
Q:
Were you aware that PLT has database support?
(define henv
    (alloc-handle 'sql-handle-env))
(set-env-attr henv
    'sql-attr-odbc-version
    'sql-ov-odbc3)

(define hdbc
    (alloc-handle 'sql-handle-dbc henv))
(connect hdbc datasource user passwd)
(define hstmt
    (alloc-handle 'sql-handle-stmt))
(exec-direct hstmt query-sql)
(fetch hstmt)
(get-data hstmt 1 buffer indicator)
...
(free-handle hstmt)
(free-handle hdbc)
(define henv
  (alloc-handle 'sql-handle-env))

(set-env-attr henv 'sql-attr-odbc-version 'sql-ov-odbc3)

(define hdbc
  (alloc-handle 'sql-handle-dbc henv))

(connect hdbc datasource user passwd)

(define hstmt
  (alloc-handle 'sql-handle-stmt))

(exec-direct hstmt query-sql)

(fetch hstmt)

(get-data hstmt 1 buffer indicator)

...
spgsql

(planet schematics/spgsql)

(planet ryanc/db)
in defense of SrPersist

based mature interface

existing drivers

PLTA version 103
the db library
A database interface for functional programmers
wire protocol
base connections
(postgresql-connect
   #:database "mydb"
   #:user "ryan"
   #:password "secret")
(postgresql-connect
  #:database "mydb"
  #:user "ryan"
  #:password "secret"
  #:server "db.mysite.com"
  #:port 5432
  #:ssl 'yes
  #:notice-handler (current-output-port))
(postgresql-connect
  #:database "mydb"
  #:user "ryan"
  #:password "secret"
  #:socket
  "'/var/run/postgresql/.s.PGSQL.5432'"
(postgresql-connect
    #:database "mydb"
    #:user "ryan"
    #:password "secret"
    #:socket 'guess)
(mysql-connect #:database "mydb"
  #:user "ryan"
  #:password "secret")

(sqlite3-connect #:database "~/my.db")

(odbc-connect #:dsn "mydsn")
(put-dsn 'pg

  (postgresql-data-source
    #:database "mydb"
    #:user "ryan"
    #:password "secret")

(dsn-connect 'pg)

(dsn-connect 'pg #:ssl 'yes)
create table the_numbers (n integer, d varchar(20))

> (query c "select n, d from the_numbers \
where n < 3")

(recordset
'(((name . "n") ....) ((name . "d") ....))
(list (vector 0 "zero")
  (vector 1 "one")
  (vector 2 "two")))
create table the_numbers (n integer, d varchar(20))

>` (query-exec c "delete from the_numbers \n where n = 0")`

>` (query-rows c "select n, d from the_numbers \n where n < 3")`

(list (vector 1 "one") (vector 2 "two"))
create table the_numbers (n integer, d varchar(20))

> (query-value c "select d from the_numbers \\
where n = 1")
"one"

> (query-list c "select n from the_numbers \\
where n < 3")
'(1 2)

> (query-row c "select n, d from the_numbers \\
where n % 5 = 0")
(vector 5 "five")
create table the_numbers (n integer, d varchar(20))

> (in-query c "select n, d from the_numbers \n    where n <= 3")

#<sequence>

> (for [[(n d)]]
    (in-query c "select n, d from the_numbers \n        where n <= 3")))
    (printf "~a: ~a\n" n d))

1: one
2: two
3: three
create table the_numbers (n integer, d varchar(20))

> (query-value pg-c
   "select d from the_numbers \
     where n = $1"
   7)
"seven"
create table the_numbers (n integer, d varchar(20))

> (define get-d-between
  (prepare my-c
    "select d from the_numbers \n    where n > ? and n < ?")

> (query-list my-c get-d-between 1 6)
  (list "two" "three" "four" "five")

> (query-value my-c get-d-between 0 2)
  "one"
Non-goal:
Abstracting over different SQL dialects (even placeholder syntax)
<table>
<thead>
<tr>
<th>SQL</th>
<th>Racket</th>
</tr>
</thead>
<tbody>
<tr>
<td>varchar, etc</td>
<td>string</td>
</tr>
<tr>
<td>integer, etc</td>
<td>exact integer</td>
</tr>
<tr>
<td>numeric, decimal</td>
<td>exact rational</td>
</tr>
<tr>
<td>real</td>
<td>inexact real</td>
</tr>
<tr>
<td>blob, bytea</td>
<td>bytes</td>
</tr>
</tbody>
</table>
(struct sql-date (year month day))

(struct sql-time (hour minute second nanosecond tz))

(struct sql-timestamp (year month day hour minute second nanosecond tz))

(struct sql-interval (years months days hours minutes seconds nanoseconds))
Other SQL types

- bit vectors
- geometry
A problem: Web servlet database connection lifetimes
- short
- unpredictable
(define pool
  (connection-pool (lambda () ....))))

(let ([c (connection-pool-lease pool)])
  .... do work with c ....
  (disconnect c))
virtual connection

(define c
  (virtual-connection (lambda () ....)))

.... do work with c ....
(define pool (connection-pool (lambda () ....)))

(define c
  (virtual-connection
    (lambda ()
      (connection-pool-lease pool
        (current-thread))))))

..... do work with c .....
(define c
  (virtual-connection
   (connection-pool (lambda () ....)))))

.... do work with c ....
making libraries
rackety
no crashes
manage resources (GC)
thread-safe, etc
no crashes
manage resources (GC)
thread-safe, etc

- continuations, exceptions
- break-thread, kill-thread
- custodian-shutdown-all
eliminate superfluous side-effects

- rackety
- simple
- functional
- safe (managed)
eliminate superfluous concepts
cooperate with Racket features
no documentation = no users
writing documentation shames us into removing little idiocies from our designs
writing documentation shames us into removing little idiocies from our designs

if you can't explain it, redesign it
Questions?
Questions
I have them for you.
cursors
date/time
geometry