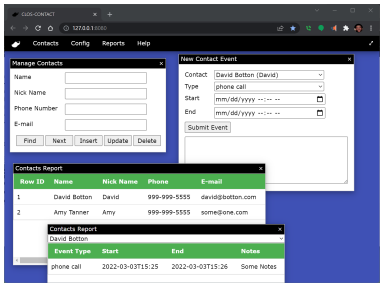
CLOG Builder 1.1

# Tutorial 4 - A Complete Database App

**A Relational Database Web GUI + Basic Reporting - CLOS-CONTACT**



## Introduction

One of my goals with CLOG is to allow for easy relational database programming of websites and local apps. I have many plans for CLOG that will keep advancing in this area and I hope many of you take advantage of this aspect of CLOG and make an amazing living writing apps and sites for people and companies. (**Volunteer Opportunity** - I can always use help with improving tutorials and manuals, etc is a long story why but sorry for the poor spelling and incomplete thoughts etc better poor docs than none. I speak better Lisp, promise ;)

For this example we will create a contact management app we will call CLOS-CONTACT

## Step 1 - Create our model

So let's first look at the big picture:

We need the following tables:

1. Contact Information
2. Contact Event and Notes
3. Event Types

We are going to use SqlLite that automatically creates a pseudocolumn rowid for each table and use that unique id to build our relations.

Now let's write our SQL to create each table (we are going to use a sqlite database so there is limited typing):

CREATE TABLE contact (name varchar, nickname varchar, phone varchar, email varchar)
CREATE TABLE contact_event (contact_id integer, event_type_id integer, start_dtime varchar, end_dtime varchar, notes varchar)
CREATE TABLE event_type (description varchar)

We could certainly create a better schema with constraints and views, etc. but we are here to learn about CLOG.

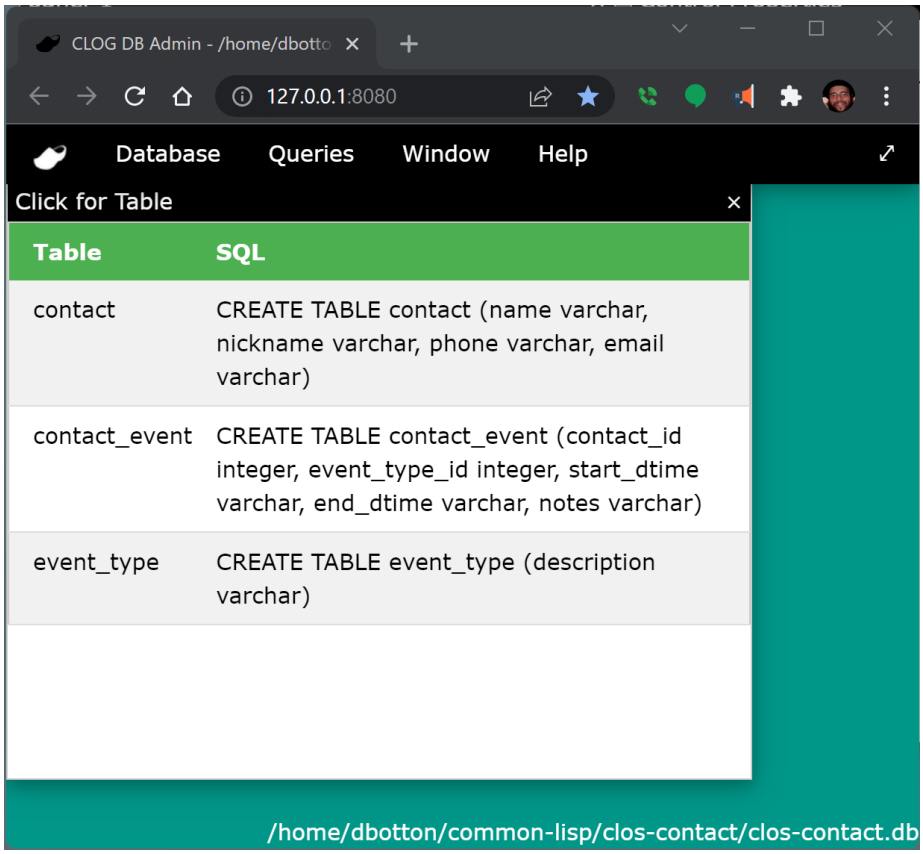
There are various ways to create or manage programmatically schema but we will just use the very basic clog-db-admin for this tutorial. (**Volunteer Opportunity** - Extend the app to use dbi and other types of databases, visual table creator etc :)

Let's get our schema in to a database:

1. In ~/common-lisp install clog from github

```
git clone https://github.com/rabbibotton/clog.git
```

2. load up in emacs **`M-x slime`**
3. run **`(ql:quickload :clog/tools)`**
4. run **`(clog-tools:clog-db-admin)`**
5. In the clog-db-admin tool choose from menu Database -> Open Connection
6. Opening a sqlite file that does not exist will create it.
7. Create - clos-contact.db
8. The color of background will change to green and the path to our database file will be in the bottom right corner of the browser.
9. Now for each of the three sql create statements above copy and paste one at a time after choosing Queries -> Execute Non Query
10. Your schema is now in the database and you can choose Queries -> Tables and should look like the following graphic.




## Step 2 - Create our App

Now that we have our database ready let's create our app.

1. Run **`(clog-tools:clog-builder)`**
2. Choose Builder -> New Application Template -> New CLOG-GUI Project and press Fill Template
3. For new system name enter **`clos-contact`**

4. When the directory dialog comes up just hit OK to place the new project in the ~/common-lisp/ directory

Let's close down the tools and the current instance of CLOG that is running on the clog directory and open our new app.

1. Run (clog:shutdown)
2. Run (ql:quickload :clos-contact)
3. Run (clos-contact:start-app)

Copy your clos-contact.db file into your clos-contact app directory. You can keep the original clos-contact.db as a backup.
You can play around with the app or open the file clos-contact.lisp and change colors, etc.

## Step 3 - Planning our Panels

So similar to the schema, let's first take a look at the big picture.

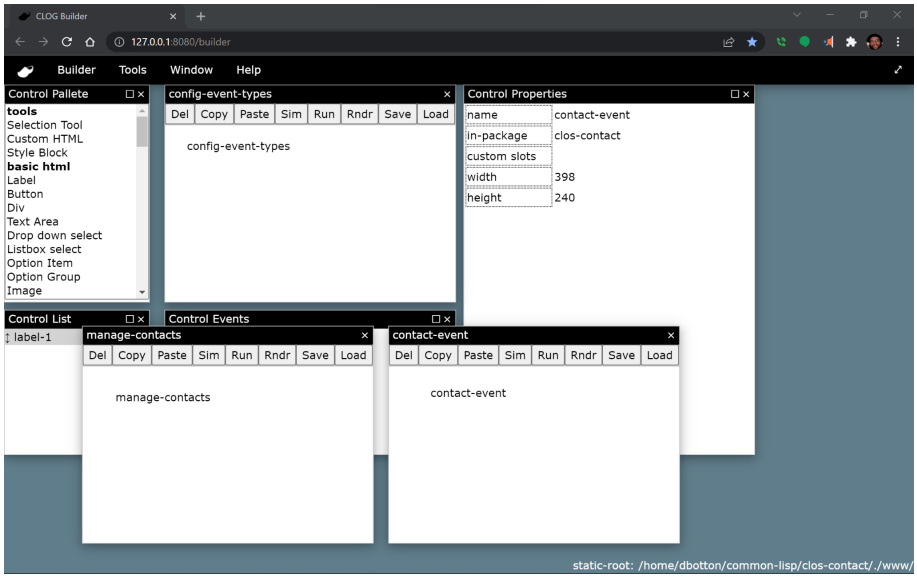
We will need the following panels

1. A panel to configure event_types
2. A panel to manage contacts
3. A panel to enter a new contact_event

We will also add some reports later and we will install those on the menu.

Ideally you have seen the previous tutorials and at least clog-builder tutorial 1. So we will not repeat everything from there but things are fairly intuitive and you should be able to follow.

## Step 4 - Creating the Panels




In clog-builder open a new CLOG-GUI panel if there is not one already there on start up. Set the in-package property to **`clos-contact`** and name the panel **`config-event-types`**

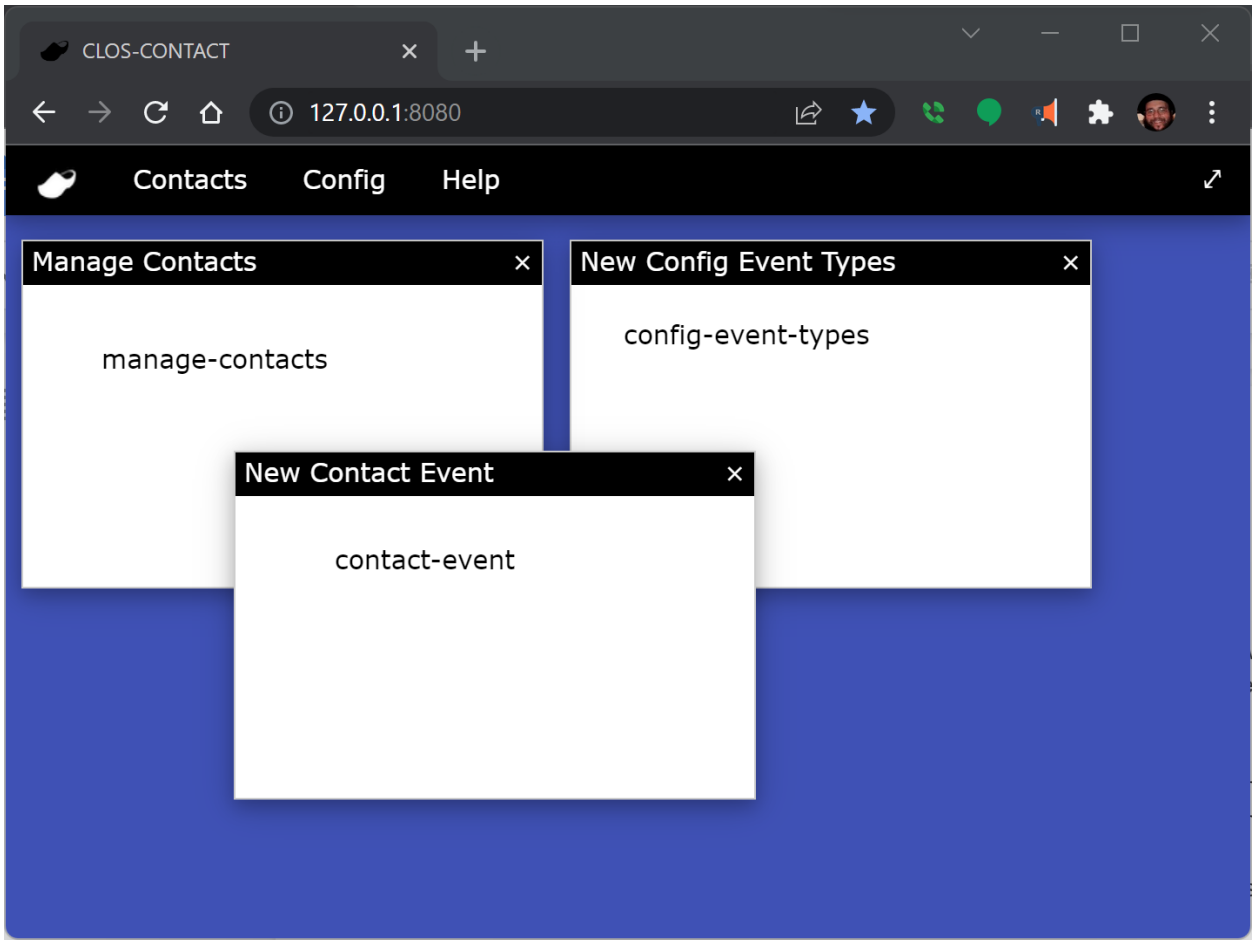
Click on the save button and save the file in our clos-contact directory as config-event-types.clog

Let's open two more Builder -> New CLOG-CUI Panels and give each the same in-package **`clos-contact`**. One we will name **`manage-contacts`** and the other **`contact-event`** and save each in clos-contact directory with the extension .clog

Now for each panel lets just add a label to identify them while we install the three files into our system and our menu.

Now for each of our three panels use the Rndr button to render the panels to Lisp with the same file names and the extension .lisp

## Step 5 - Adding the Panels




First let us add the three panels to our asdf file in the components section after our clos-contact file (since we define the package in clos-contact it needs to come before the others):

```
(asdf:defsystem #:clos-contact
  :description "New CLOG System"
  :author "some@one.com"
  :license  "BSD"
  :version "0.0.0"
  :serial t
  :depends-on (#:clog)
  :components ((:file "clos-contact")
               (:file "config-event-types")
               (:file "manage-contacts")
               (:file "contact-event")))
```

Next let's get all in memory, run **`(ql:quickload :clos-contact)`** again.

Let's add the three panels in clos-contact.lisp. First let's modify the menu:

```
(defun on-new-window (body)
  (let ((app (make-instance 'app-data)))
    (setf (connection-data-item body "app-data") app)
    (setf (title (html-document body)) "CLOS-CONTACT")
    (clog-gui-initialize body)
    (add-class body "w3-indigo")
    (let* ((menu-bar    (create-gui-menu-bar body))
           (icon-item   (create-gui-menu-icon menu-bar :on-click 'on-help-about))
           (cntct-item  (create-gui-menu-drop-down menu-bar :content "Contacts"))
           (contacts    (create-gui-menu-item cntct-item :content "Manage Contacts"
:on-click 'on-manage-contacts))
           (events      (create-gui-menu-item cntct-item :content "New Contact Event"
:on-click 'on-new-event))
           (config-item (create-gui-menu-drop-down menu-bar :content "Config"))
           (event-types (create-gui-menu-item config-item :content "Manage Event
Types" :on-click 'on-config-event-types))
           (help-item   (create-gui-menu-drop-down menu-bar :content "Help"))
           (help-about  (create-gui-menu-item help-item :content "About" :on-click
'on-help-about))
           (full-screen (create-gui-menu-full-screen menu-bar)))
      (declare (ignore icon-item help-about full-screen)))))
```

Do a M-C-x to compile the function and now you can go to the browser window with 127.0.0.1:8080 that opened when we ran (clos-contact:start-app) before and refresh it and our new menus should be there.

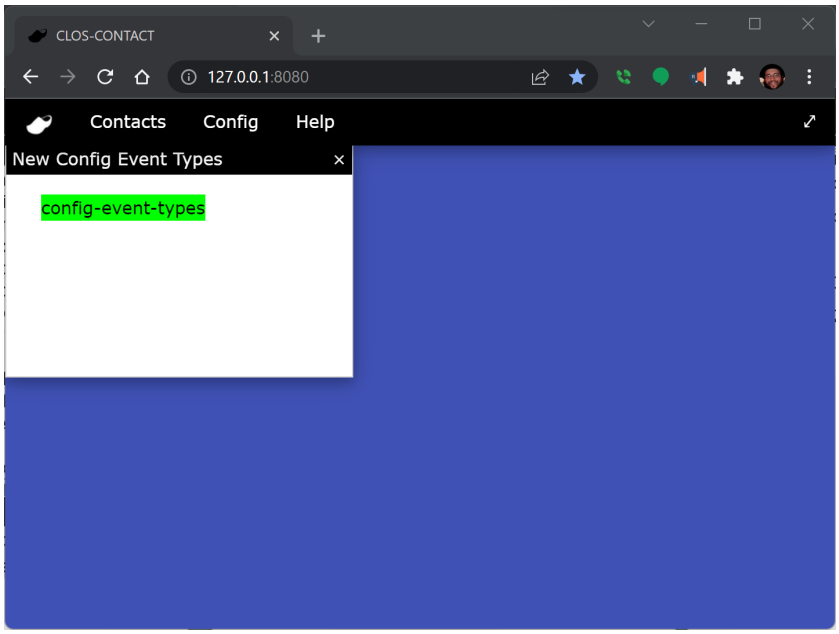
New let's get rid of the on-file-new function and add the following three new ones.

```
(defun on-manage-contacts (obj)
  (let* ((app (connection-data-item obj "app-data"))
         (win (create-gui-window obj :title "Manage Contacts")))
    (declare (ignore app))
    (create-manage-contacts (window-content win))))

(defun on-new-event (obj)
  (let* ((app (connection-data-item obj "app-data"))
         (win (create-gui-window obj :title "New Contact Event")))
    (declare (ignore app))
    (create-contact-event (window-content win))))

(defun on-config-event-types (obj)
  (let* ((app (connection-data-item obj "app-data"))
         (win (create-gui-window obj :title "New Config Event Types")))
    (declare (ignore app))
    (create-config-event-types (window-content win))))
```

After adding to the file (always save ;) compile each function with M-C-x and your apps menus will show our three panels. It is not necessary to refresh the browser as only changes to our on-connect handler (our on-new-window) require that. The power of a REPL and Lisp in development is simply awesome.

## Step 6 - More Awesome

Before we go further, it is worth showing you more of that awesomeness to make development with Lisp faster than any other language. Go back to the clog-builder, drop anything on the config-events-types panel so you can see the change. Now click the run button on the config-events-types panel. When it opens a new window you can close it. Go back to our browser window with our app and choose the menu Config -> Manage Event Types and the change is live in our app.




## Step 7 - The Config-Event-Types Panel

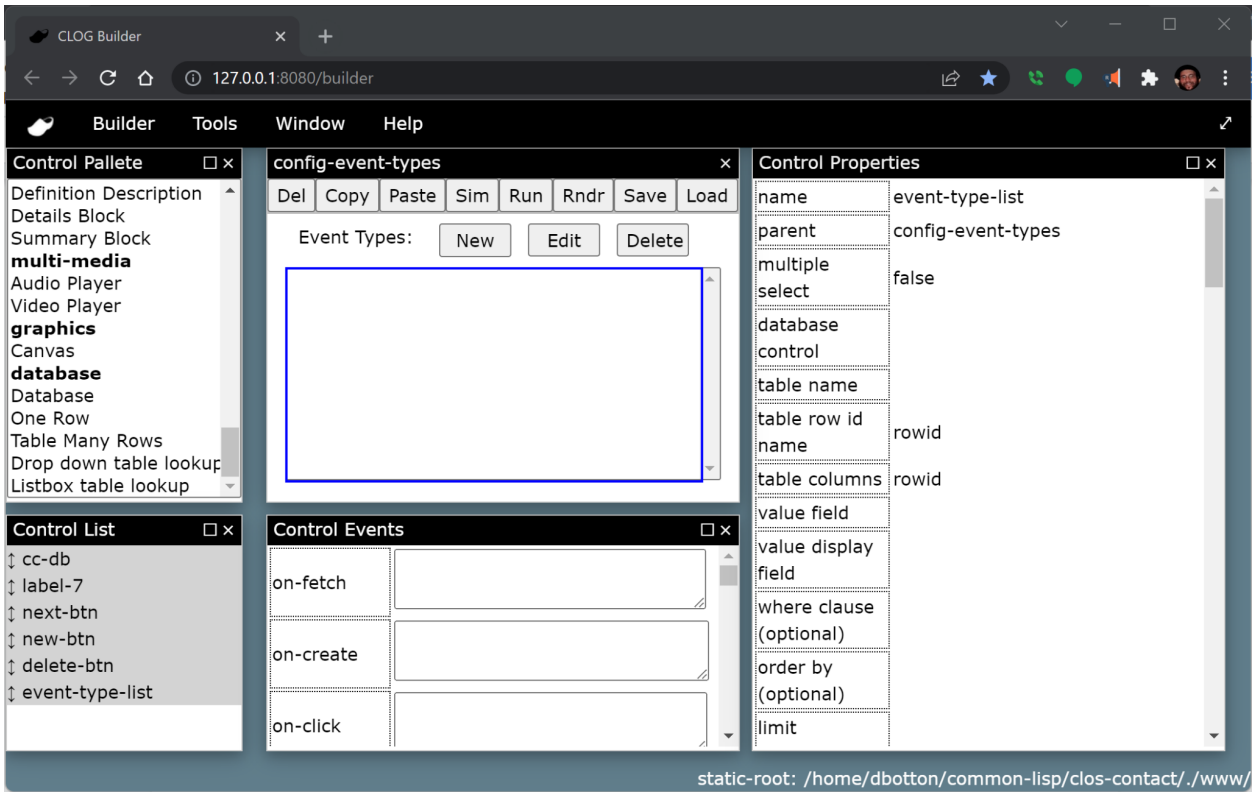
So let's now work on our config-event-types panel.

First let's drop a database control. It needs to come before any other control that depends on it so that the connection is made before they use it. I named it **`cc-db`** and changed positioning to **`static`**. I then set the location for the database in the database name property to /home/dbotton/common-lisp/clos-contact/clos-contact.db for now. Unless, of course, your name is the same as mine, I suggest adjusting the path :)

Now let's add a label "Event Types:", three buttons New Edit and Delete setting their names to **`new-btn`**, **`edit-btn`**, and **`delete-btn`** and a Listbox table control.

This list box table control I set up with Top 45px Left, Right and Bottom at 15px and erased the width and height (they will show up again but the act of erasing them erases it from the html, ie we end up with a listbox that is "pinned" at each of its corners and will adjust in size according to its containing panel).

This is how my panel looks now:




Next we have to connect our database listbox, that I called **`event-type-list`** to our cc-db by setting the database control property to **`cc-db`**. Let's set the table name to **`event_types`**, the value field to **`rowid`** and the value display field to **`description`**.
We need to add one more event, to on-create of the **`event-type-list`** we need to add **`(get-row target panel)`** which tell the listbox to fill from the table. (Don't forget to save! ;)

At this point the Listbox table lookup control is actually functional and if there were entries already in the database and we hit run they would show up.

Now we need to add functionality to our buttons. So let's set three event handlers one for each that will add to our clos-contact.lisp file shortly. So in the on-click event for the new-btn type **`(cet-on-new target panel)`**, for the edit-btn **`(cet-on-edit target panel)`** and for the delete-btn **`(cet-on-delete target panel)`**

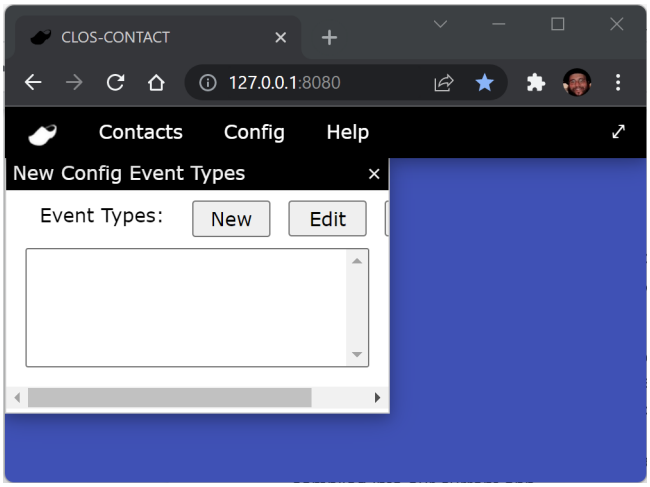
Before going to our editor, let's Rndr the panel to config-event-types.lisp and let's Run it so it is compiled into our current app.

## Step 8 - Coding the events of the Config-Event-Types Panel

So in our config-event-types.lisp file let's add our three events as place fillers.

```
(defun cet-on-new (target panel))
(defun cet-on-edit (target panel))
(defun cet-on-delete (target panel))
```

Now we can see that all is working, but the size of the new window is a bit too small.




To fix that we add :width 400 in the on-config-event-types creation of the window (don't forget to compile with M-C-x after making the change - lisp is awesome!):

```
(defun on-config-event-types (obj)
  (let* ((app (connection-data-item obj "app-data"))
         (win (create-gui-window obj :width 400
                                     :title "New Config Event Types")))
    (declare (ignore app))
    (create-config-event-types (window-content win))))
```

You can go ahead and test if you like, but we are going to implement our three events.

So let's add the code to pop an INPUT-DIALOG and insert a new row:

```
(defun cet-on-new (target panel)
  (input-dialog panel "Enter new event type description"
                (lambda (input)
                  (when input
```

```
                    (dbi:do-sql (database-connection (cc-db panel))
                      "INSERT INTO event_type (description) VALUES (?)" (list input))
                    ;; get-row on one of the table related database controls
                    ;; refreshes the query.
                    (get-row (event-type-list panel) panel)))))
```

We inserted the row using dbi and straight SQL, when we learn about the one-row control next you will see this could have been done with no SQL.

We refreshed the listbox with get-row which for our listbox will then fill itself for each row in the table internally calling next-row till done.

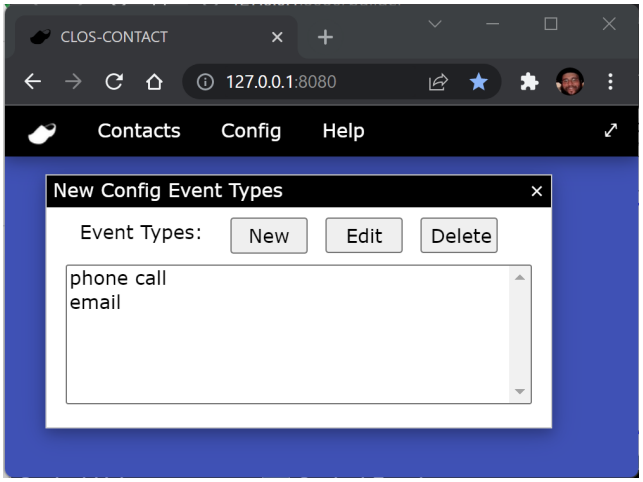
Now let's implement the cet-on-edit event:

```
(defun cet-on-edit (target panel)
  (let ((rowid (value (event-type-list panel))))
    (unless (equal rowid "")
      (input-dialog panel "Update event type description"
                    (lambda (update)
                      (when update
                        (dbi:do-sql (database-connection (cc-db panel))
                          "UPDATE event_type SET description=? WHERE rowid=?"
                          (list update rowid))
                        (get-row (event-type-list panel) panel)))
                    :default-value (select-text (event-type-list panel))))))
```

And the last event cet-on-delete

```
(defun cet-on-delete (target panel)
  (let ((rowid (value (event-type-list panel))))
    (unless (equal rowid "")
      (confirm-dialog panel
                      (format nil "Delete - ~A?"
                              (select-text (event-type-list panel)))
                    (lambda (del)
                      (when del
                        (dbi:do-sql (database-connection (cc-db panel))
                          "DELETE FROM event_type WHERE rowid=?" (list rowid))
                        (get-row (event-type-list panel) panel)))))))
```

With those events we have an entirely working configuration panel.




## Step 9 - The Manage Contacts Panel

So let's now work on our manage-contacts panel.

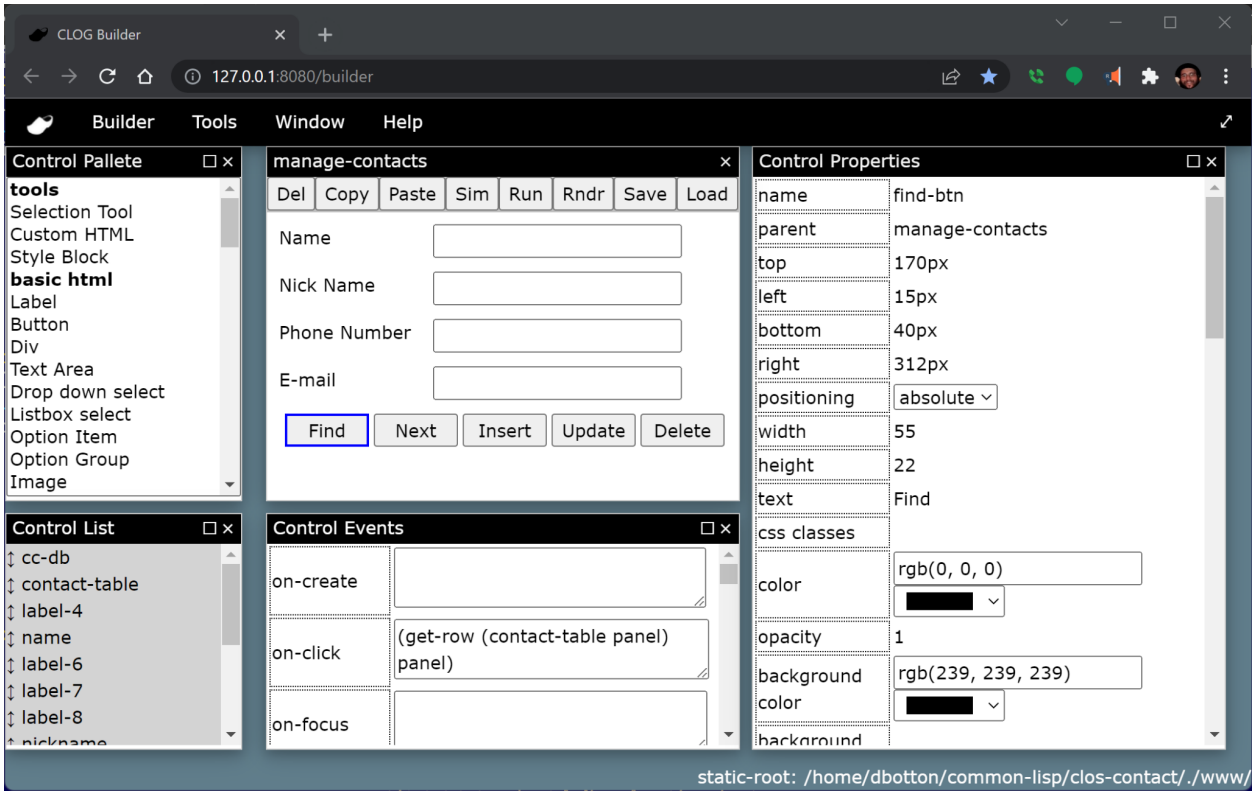
First let's add a database control. I named it cc-db and changed positioning to static. I then set the location for the database in the database name property to /home/dbotton/common-lisp/clos-contact/clos-contact.db (adjust path as needed).

Let's add and configure a one-row control, set the database control to **`cc-db`**, set the table to **contact** and the table columns to **rowid name nickname phone email** and let's name it **contact-table**. Let's also set sort by to **`name`**. The one-row control will automatically match elements on the panel to rows it fetches based on their name being the same as the table column. You can use (table-column as-name) in the table column list and as-name is used instead to match controls. Keep in mind that sql doesn't like dashes in field names.

Let's add labels and form inputs for each row (except rowid) and name them the same as the columns and let's add Find, Next, Insert, Update and Delete buttons each names -btn in lowercase.

Let's modify the on-click for each button with calls to functions of the one-row control contact-table. Later we will create a more complex event handler for find and will add it to clos-contact.lisp

For on-click in find - **`(get-row (contact-table panel) panel)`**
For on-click in next - **`(next-row (contact-table panel) panel)`**
For on-click in insert - **`(insert-row (contact-table panel) panel)`**
For on-click in update - **`(update-row (contact-table panel) panel)`**
For on-click in delete - **`(delete-row (contact-table panel) panel)`**




We can now Save, Rndr, and Run. You can try it out on the already both in the window that opens or go back to our app and try in there.

After you are done trying that out, let's change the on-click for find to -
**`(manage-contacts-find target panel)`**

Then Save, Rndr, and Run again and let's add some code.

## Step 10 - Coding the events of the Manage-Contacts Panel

First let's adjust the size of the Manage Contacts window by adding :height 240 :width 400 to the create-gui-window.

Next let's create the manage-contacts-find handler.

```
(defun manage-contacts-find (target panel)
 (input-dialog panel "Enter search string eg, name like 'david%'"
               (lambda (input)
                 (when input
                   (setf (where-clause (contact-table panel)) input)
                   (get-row (contact-table panel) panel)))))
```

And with that, a little M-C-x magic and we have a completely working Manage Contacts panel.

## Step 11 - The Contact Events Panel

So for the contact events panel let's set up cc-db and as before.

Let's set up a table one row control we will call it contact-event-table, we set the database control to cc-db we set the table name to contact_event and we set the table columns to:

```
rowid contact_id event_type_id start_dtime end_dtime notes
```

Let's add a drop down table look up, we will name it **`contact_id`** to match our relational field in the contact_event table, set the database control to **`cc-db`**, the table name to **`contact`**, Then for table columns use the following:

```
rowid ("name||' ('||nickname||')'" "disp")
```

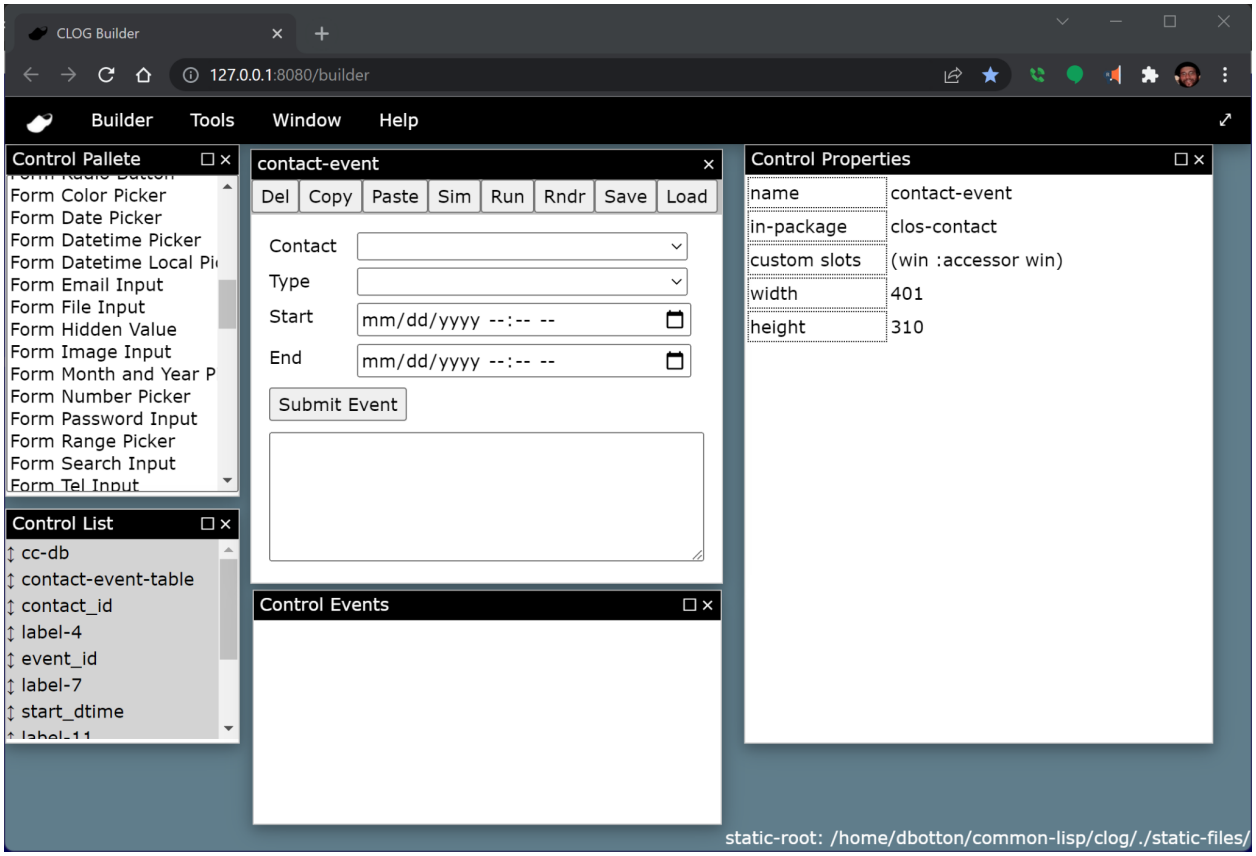
In this case we use the (table-column as-name) construct we mentioned above. In this case we construct using sql ( || is sqlite for concatenate) the column that we will call disp. Then we set the value field to **`rowid`** and the value display field to **`disp`** our calculated field.

Let's add a second drop down table lookup with our event type, we will name it **`event_type_id`** to match the contact_event table field, we set the database control to **`cc-db`** the table name to **`event_type`** the table columns to **`rowid description`** the value field to **`rowid`** and the value display field to **`description`**.

Now let's set the rest up, we need a start and end time that are called **`start_dtime`** and **`end_dtime`** we can use the date time local picker for a nice graphical picker, then we need a text area called **`notes`** and finally our submit event button with and on-click set to our soon to **`(contact-event-submit target panel)`**

We also need a custom slot on the panel (click on the panel not on a control and the panel properties appear) called **`win`** so that we can close the clog-gui window once we submit our event.

We 'Save, Rndr, and Run' ™ and we are ready to go back to the editor.




## Step 12 - Coding the events of the Contact Events Panel

First adjust the size of the contact event window in on-new-event to :height 300 and :width 400 and we need to set the win slot on the new panel:

```
(defun on-new-event (obj)
  (let* ((app (connection-data-item obj "app-data"))
         (win (create-gui-window obj
                                 :height 350
                                 :width 450
                                 :title "New Contact Event")))
    (declare (ignore app))
    (setf (win (create-contact-event (window-content win))) win)))
```

Next we implement our contact-event-submit button:

```
(defun contact-event-submit (target panel)
  (insert-row (contact-event-table panel) panel)
  (window-close (win panel)))
```

Use M-C-x on those two functions and the app is done except for our reports.

## Step 13 - Adding our reports menus

We are going to only add two reports (contact list and contact events) and leave as an exercise creating a panel to pick dates, choose which contacts reports are for etc.

First let's add a menu item for reports and two menu entries for our reports.

```
(defun on-new-window (body)
  (let ((app (make-instance 'app-data)))
    (setf (connection-data-item body "app-data") app)
    (setf (title (html-document body)) "CLOS-CONTACT")
    (clog-gui-initialize body)
    (add-class body "w3-indigo")
    (let* ((menu-bar    (create-gui-menu-bar body))
           (icon-item   (create-gui-menu-icon menu-bar :on-click 'on-help-about))
           (cntct-item  (create-gui-menu-drop-down menu-bar :content "Contacts"))
           (contacts    (create-gui-menu-item cntct-item :content "Manage Contacts"
:on-click 'on-manage-contacts))
           (events      (create-gui-menu-item cntct-item :content "New Contact Event"
:on-click 'on-new-event))
           (config-item (create-gui-menu-drop-down menu-bar :content "Config"))
           (event-types (create-gui-menu-item config-item :content "Manage Event
Types" :on-click 'on-config-event-types))
           (report-item (create-gui-menu-drop-down menu-bar :content "Reports"))
           (contact-rep (create-gui-menu-item report-item :content "Contact List"
:on-click 'on-report-contacts))
           (events-rep  (create-gui-menu-item report-item :content "Contact Event
List" :on-click 'on-report-events))
           (help-item   (create-gui-menu-drop-down menu-bar :content "Help"))
           (help-about  (create-gui-menu-item help-item :content "About" :on-click
'on-help-about))
           (full-screen (create-gui-menu-full-screen menu-bar)))
      (declare (ignore icon-item help-about full-screen)))))
```

Now let's add the two placeholder handlers that open windows:

```
(defun on-report-contacts (obj)
  (let* ((app (connection-data-item obj "app-data"))
         (win (create-gui-window obj :width 400
                                     :title "Contacts Report")))
    (declare (ignore app))))

(defun on-report-events (obj)
  (let* ((app (connection-data-item obj "app-data"))
         (win (create-gui-window obj :width 400
                                     :title "Contacts Report")))
    (declare (ignore app))))
```

## Step 14 - Adding our Contacts report

So let's work on our first report, the contact list. In builder let's start a new CLOG-GUI panel, set the name to **`report-contacts`**, and the in-package to **`clos-contact`**, add our database control with the same settings. We will save it as **`report-contacts.clog`**

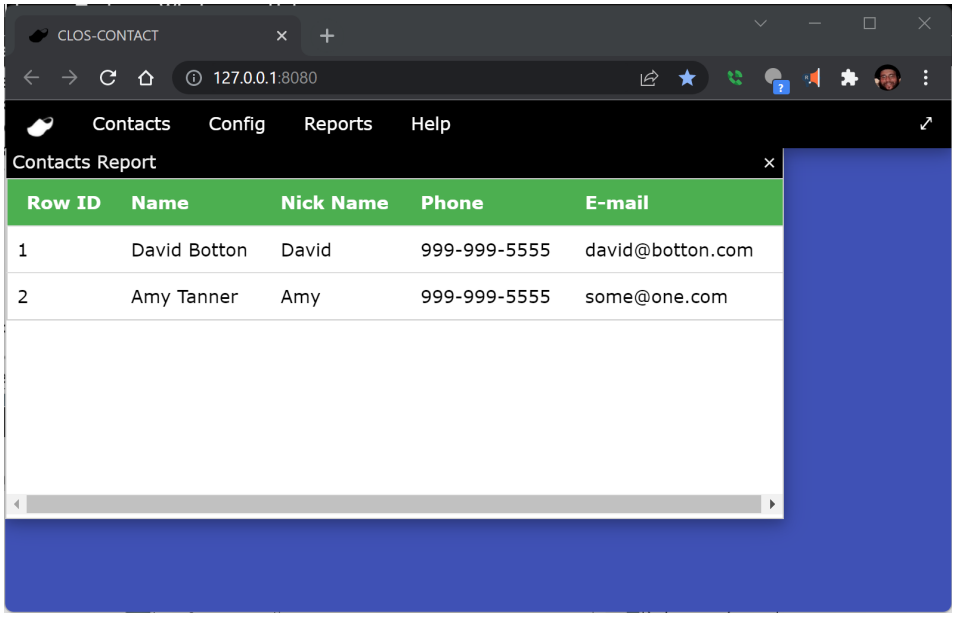
Then we will add with static positioning a Many Rows control. We will name it **`table-contact`**, set the table-name to contact, table columns to **`rowid name nickname phone email`**, set the css classes to **`w3-table-all`** to make it look nicer. Will add to on-create event **`(get-row target panel)`** to load data on creation and for on-header let's add to give it header:

```
(create-child target "<tr class='w3-green'><th>Row
ID</th><th>Name</th><th>Nick
Name</th><th>Phone</th><th>E-mail</th></tr>")
```

Lets Save, Rndr and Run and you will see the report. Let's add it to our clos-contact.asd file at the end of the list and then let's install it in our clos-contact.lisp file in the on-report-contacts handler:

```
(defun on-report-contacts (obj)
  (let* ((app (connection-data-item obj "app-data"))
         (win (create-gui-window obj :width 400
                                     :title "Contacts Report")))
    (declare (ignore app))
    (create-report-contacts target)))
```

Do a M-X-c and should be able to see the report in the app.



## Step 15 - Adding our Event report

For the event report let's make the report a bit more dynamic and allow selecting the contact for the events to see.

In CLOG builder let's start a new CLOG-GUI panel, set the name to **`report-events`**, and the in-package to **`clos-events`**, add our database control with the same settings. We will save it as **`report-events.clog`**

Now let's place a Drop down table lookup. Set it to use positioning **`static`**, width set **`100%`** and call it **`contact-list`**. Set the database control to **`cc-db`**, the table name to **`contact`**, table columns to **`rowid name`**, value field is **`rowid,`** the value display field is **`name`** and the sort by also to **`name.`** For the events we will set on create to **`(get-row target panel)`** to populate the drop down at creation and we will set on-change to handle changes to the drop down to call **`(report-events-run (event-table panel) panel)`** which we will create the event in clos-contact.lisp.

Now we add a Table many rows control. We set the name to **`event-table`** and width to **`100%`**, and for css classes we will use **`w3-table-all`** to make it look snazzy. This time we will only set the database control to **`cc-db`** since we will be doing a custom query in code. In the events set the on-header to:

```
(create-child target "<tr class='w3-green'><th>Event
Type</th><th>Start</th><th>End</th><th>Notes</th></tr>")
```

Lets Save, Rndr and Run and you will see the but only the drop will work as we have not created our respot-event-run event yet. So in clos-contact.lisp lets add our report to the on-report-events and our report-event-run:

```
(defun on-report-events (obj)
  (let* ((app (connection-data-item obj "app-data"))
         (win (create-gui-window obj :width 400
                                     :title "Contacts Report")))
    (declare (ignore app))
    (create-report-events (window-content win))))

(defun report-events-run (target panel)
  (query-row (event-table panel) panel
             (format nil "SELECT e.description, c.start_dtime, c.end_dtime, c.notes ~
                          FROM contact_event c ~
                          INNER JOIN event_type e ~
                          ON c.event_type_id=e.rowid
                          WHERE c.contact_id=~A"
                     (value (contact-list panel)))))
```

At this point you can do M-X-c on the two functions or just recompile it all with (ql:quickload :clos-contact) as we are done :)

## Conclusion - CLOG is Awesome