

# **Gnucobol Html MariaDB**

## Index.

<b>Gnucobol Html MariaDB.....</b>	<b>1</b>
1.- Overall notes and considerations.....	3
2.- Database content. ....	5
3.- COBOL Programs included.....	8
4.- Pages and subpages, HTML design notes and rules.....	11
5.- The Javascript (User browser side execution). ....	13
6.- How it works. ....	14

This project with GNUCobol is an intent to use html web pages to emulate IBM CICS ® and also using Mariadb ® (Mysql ®) as database server like using IBM DB2 ®. So is fully based on free software.

For better end user usability, we use bootstrap 5.

Pages will use also Javascript to perform their actions.

NOTE: I will use Pseudo Conversational transaction model. So no BEGIN TRANSACTION / END TRANSACTION or SELECT FOR UPDATE may be used.

As other on-line systems and applications, we use the "dirty reads" for all table access, except when a user try get a row for update o delete it then we generate an annotation on table sysrowlock, previously we verify that row of the table is not locked (owned) by other user.

[https://en.wikipedia.org/wiki/Pseudoconversational\\_transaction](https://en.wikipedia.org/wiki/Pseudoconversational_transaction).

As far I test only works with Apache server.

I test in windows 10 ® and Fedora Linux 40.

NOTE: I apologize because Some words and content are done in my fluent Spanish, I'm not very fluent in English.

This is a ongoing work and some changes may be done.

## 1.- Overall notes and considerations.

This is not a secure environment; some cryptography must be added on the URL to care de normal usage and prevent the URL manipulation.

I not find a better way to pass program to program information as using a CICS® "commarea". So we use URL parameters to send and relieve arguments to call, load and pass data for programs. The COBOL ACCEPT FROM COMMAND-LINE, is the action we perform to get the arguments passed (our commarea).

Because the ACCEPT only may read all arguments as a single chunk of data and to avoid the normal URL encode/decode, we will change all the spaces with underscore characters.

So, de usual program call will be like:

```
COBLOGIN.EXE?CLOGIN_____00000000000YCOBLOGIN.EXE__usern_____  
_____U2N91E1UHGPYLSQ0W1NJBTYNVA628F1T
```

Where the we have:

```
05 SYSCOMM.  
  
07 ACCTN                PIC X.  
  
07 PAGEX                PIC X(16) .  
  
07 IDET                 PIC X(11) .  
  
07 PAGEACTIVE           PIC X.  
  
07 PROGID               PIC X(16) .  
  
07 USERN                PIC X(32) .  
  
07 SESSION              PIC X(32) .
```

ACCTN, is the usual "CRUDL", actions (Create, Read, Update, Delete, List).

PAGEX, is de name of the page to be used (Send and Retrieve).

IDET, is only used for a single row identifier, action will be done at that row.

PAGEACTIVE, means de page was send to the user with data or empty, with value 'N', value 'Y' means de page data is back from de end user and we must process it. So, for the program 'N' is the first run and the page in PAGEX, must be

composed and send. The 'Y' value means we must get the data and process it (second program run).

PROGID, is the program name (we use .EXE as file extension for compatibility between Windows and Linux).

USERN, is the user's name logged in. (Also, as USERNAME cookie).

SESSION is the session cookie carried from page data.

The "SYSCOMM" usually is included in a COBOL EXTERNAL reference, with that structure we can call libraries (.dll or .so) with that reference.

```
01  SYSCOMMALL                EXTERNAL .  
  
    05  SYSCOMM.  
  
        07  ACCTN                PIC X.  
  
        07  PAGEX                PIC X(16) .  
  
        07  IDETX                PIC X(11) .  
  
        07  PAGEACTIVE          PIC X.  
  
        07  PROGID              PIC X(16) .  
  
        07  USERN               PIC X(32) .  
  
        07  SESSIONX            PIC X(32) .  
  
    05  LIMITE                  PIC 9(10) .  
  
    05  SALTAR                  PIC 9(10) .  
  
    05  BUSCAR                  PIC X(32) .  
  
    05  PAGETOSEND              PIC X(63535) .
```

LIMITE, is maximum rows by page. (Only used in list forms explained later).

SALTAR, is the number of rows to skip from the beginning of the table. (Only used in list forms explained later, see COBUSRLST program).


BUSCAR, is the string to search in the table. . (Only used in list forms explained later, see COBUSRLST program).

PAGETOSEND, the page to be send build by "SENDPAGE" module (.so or .dll).


## 2.- Database content.

The following tables are used.


**Sysfileds:** (See READPAGE program for usage description.

#	Nombre	Tipo	
1	id 	int(11)	Unique Row Id of the table (Primary, index).
2	session	varchar(32)	User session.
3	user	varchar(32)	User name logged in.
4	mapname	varchar(16)	Map or Page used.
5	campo	varchar(32)	Field name read from user browser.
6	lenght	int(11)	Lenth or zero.
7	comment	varchar(128)	Comment if exists.
8	valor	varchar(128)	Data value for that field (Campo).
9	stamp	varchar(20)	Timestamp where row was created.


**Sysmappage**, used by SENDPAGE to build de page to be send (PAGETOSEND).

#	Nombre	Tipo	
1	id 	int(11)	Table unique Row ID. (Primary, index).
2	name	varchar(32)	Page Name (PAGEX in SYSCOMM).
3	ord	tinyint(99)	Loading order.
4	syspages	varchar(32)	Subpage or page part to by loade.
5	fillprog	varchar(16)	Program (.so or .dll) to load part of the page (I.E. Rows in a list page.


**Sysmaps**, first reference to pages with title description.

#	Nombre	Tipo	id	name	title
1	id 	int(11)	2	USERS	USER DETAIL
2	name	varchar(16)	3	USRLST	LIST USERS
3	title	varchar(32)	4	LOGIN	LOGIN
			5	NWPWD	NEW PASSWORD
			6	MENU	COBOL APP MENU

**Syspages**, Parts or subpages to build overall HTML page as described in sysmappage.



#	Nombre	Tipo	
1	id 	int(11)	Unique Row ID.
2	page	varchar(16)	Page name
3	cont	text	Page part or subpage.

**Sysrowlock**, Row locking control.

#	Nombre	Tipo	
1	id 	int(11)	Unique Row ID.
2	tabla	varchar(32)	Table to lock.
3	rowid	int(11)	Locked row in the table.
4	user	varchar(32)	User who owns de row.
5	session	varchar(32)	Session used when locking.
6	stamp	varchar(20)	Timestamp when the lock is activated.

**Sysusers**, Users table.

NOTE we add some fields to show data, time, select check boxes, radio buttons, etc., inside the HTML generated page.

#	Nombre	Tipo	
1	id 	int(11)	Unique Row ID.
2	name	varchar(64)	User Name.
3	email	varchar(64)	User email.
4	username 	varchar(32)	User name to log in.
5	pwd	varchar(32)	Password.
6	datedmy	varchar(10)	Date in European format (dmy).
7	timehms	varchar(8)	Time (hms)
8	type	varchar(16)	Type of job (From HTML selec).
9	age	varchar(16)	Age (From radio buttons).
10	agree	char(3)	Agree to from (Check box).
11	comment	varchar(128)	Comments.

### 3.- COBOL Programs included.

#### **COBLOGIN.SQB**

Main program to the browser call after initial page.

Note the initial page must create a session cookie.

Page to be used as LOGIN in sysmapage and syspages (LOGIN and LOGINJAVA).

Uses SENDPAGE (.dll or .so) to build de page to send.

Uses table sysfields to retrieve the data form the page form.

Uses Javascript to call other programs when the user clicks buttons.

#### **COBFILLRW.SQB**

Simply program to build the inside html table with the rows of sysusers to be show depending on LIMITE and SALTAR (SYSCOMMALL EXTERNAL).

This a .so or .dll loaded from SENDPAGE (also a .so or .dll) as is indicated on sysmappage,

#### **COBNEWPWD.SQB**

To be called when the user forgot the password, allows to create a new one.

Page to be used as NWPWD in sysmapage and syspages (NWPWD and NWPWDJAVA).

Uses SENDPAGE (.dll or .so) to build de page to send.

Uses table sysfields to retrieve the data form the page form.

Uses Javascript to call other programs when the user clicks buttons.

User headers subpages and a special footer subpage 0-footernwpwd (there is not user logged in, son 'nouser' will be passed trough SYSCOMM).

#### **COBOLMENU.SQB**

Sample menu.

Uses SENDPAGE (.dll or .so) to build de page to send (MENU).

Uses table sysfields to retrieve the clicked buttons.

Uses Javascript to call other programs when the user clicks buttons MENUJAVA.

User headers (0-header and 1-header) subpages and footer (o-footer) subpage.



Sysmappage content for PAGEX MENU:

id	name	ord	▲ 1	syspages	fillprog
25	MENU	10		0-header	
24	MENU	20		1-header	
26	MENU	30		MENU	
27	MENU	40		0-footer	
29	MENU	50		MENUJAVA	

**SENDPAGE.SQB**

This is a .so or .ddl object read the sysmappage for the PAGEX in SYSCOMM and bluids all the final page to send (PAGETOSEND in the external field).

Also load de content generated by the indicated subprogram (.so or .dll) in the main page. (##inner\_html\_table##).

id	name	ord	▲ 1	syspages	fillprog
1	USRLST	10		0-header	
2	USRLST	20		1-header	
6	USRLST	30		USRLST	
3	USRLST	40		##inner_html_table##	COBFILLRW
4	USRLST	50		USRLST2	
5	USRLST	60		0-footer	
23	USRLST	70		USRLSTJAVA	

**READPAGE.SQB**

Main program called by javascript to get all the data chunk of data from javascript XMLHttpRequest. The data fields are stored in sysfields.

id	▲ 1	session	user	mapname	campo	lenght	comment	valor	stamp
178		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	session_UID	0	FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK	jordi	28-12-2024 13:17:57
179		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	PROGID	0	hidden	USERLST	28-12-2024 13:17:57
180		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	COOKIE	0	hidden	jordi	28-12-2024 13:17:57
181		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	limite	0	text	14	28-12-2024 13:17:57
182		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	saltar	0	text	1	28-12-2024 13:17:57
183		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	back	0	button	Page Back	28-12-2024 13:17:57
184		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	forw	0	button	Page Forward	28-12-2024 13:17:57
185		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	search	0	button	Search	28-12-2024 13:17:57
186		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	busca	0	text	newusr	28-12-2024 13:17:57
187		FNY8U1CP5JEEEXP0PU34B7FCGDNCXKWHK		USRLST	buttonck	0	hidden	search	28-12-2024 13:17:57

When PAGEACTIVE (SYSCOM) is 'Y', the main program read this table to get the result of the user browser interaction.

See the pages notes and rules to know the hidden fields to care about.

### **COBUSRLST.SQB**

Sample list program.

Uses SENDPAGE (.dll or .so) to build de page to send (USRLST). SENDPAGE calls also COBFILLRW to fill the table rows with users.

Uses table sysfields to retrieve the clicked buttons and data.

Uses Javascript to call other programs when the user clicks buttons USRLSTJAVA.

User headers (0-header and 1-header) subpages and footer (o-footer) subpage.

### **COBNEWUSR.SQB**

Sample CRUD program for sysusers table.

Uses SENDPAGE (.dll or .so) to build de page to send (USERS).

Uses table sysfields to retrieve the clicked buttons and data.

Uses Javascript to call other programs when the user clicks buttons USERSJAVA.

User headers (0-header and 1-header) subpages and footer (o-footer) subpage.

### **IMPORTANT NOTE:**

*Because Javascript call the READPAGE and the main program in the same time we need to wait until READPAGE ends to fill all the sysfields content, so before reading the sysfields rows by the main program we **code a sleep of half second.***

*You may need to adjust the sleep time depending on database and overall system performance.*

#### 4.- Pages and subpages, HTML design notes and rules.

The page build by SENDPAGE carries all the subpages in a single html page, also there are some "## valuetofill ##" like "## title ##" where SENDPAGE fills the page title.

As overall design we use:

- A common header subpage with all the initial html, including ".css" files and initialization steps.
- A second common and optional header page with the real header to be displayed and used when needed.
- The main subpage with the specific content for a program.
  - o This subpage may be in various parts to allow the call of a filler program (.so or .dll). I.e. Table content for a list or html select content.
- The footer page loading all the JavaScript files and also the JavaScript page coding needs.
- A final end page with the JavaScript to control the buttons and calling programs for the user actions.

Designing rules:

- We use "F1" as the form name (html form), if changed remember to change the JavaScript end page.
- The "##some content##" means this may be filled or changed before to send the page.
- Fields need to have id and name.
  - o `<input type="text" class="form-control" id="name" name="name" value="##name##" size="64" maxlength="64">`
- Some hidden fields are used at the beginning of the main subpage:
  - o `<form name="f1" id="f1">`
  - o `<input type="hidden" id="PROGID" name="PROGID" value="MENU">`
  - o `<input type="hidden" id="COOKIE" name="COOKIE" value="">`
- One hidden field needs to be allocated at the end of the form. Also a "result" allocation may be used for message display purposes.
  - o `<input name="buttonck" id="buttonck" type="hidden">`
  - o `<div id="result"> <h4>Message:</h4> </div>`
- Buttons must have, name, id, onclick and "##some content##".
  - o `<input value="Update User" id="update" name="update" type="button" class="btn btn-primary btn-round" role="button" onclick='javascript:clicked("update")' ##upda##>`

- Onclick JavaScript, carry the clicked name to buttonck field, so the program may know with button clicks the end user.
- The "##somme content##" is used to the filling main program to enable or disable the button when running.

## 5.- The Javascript (User browser side execution).

In the JavaScript end page we only need to code:

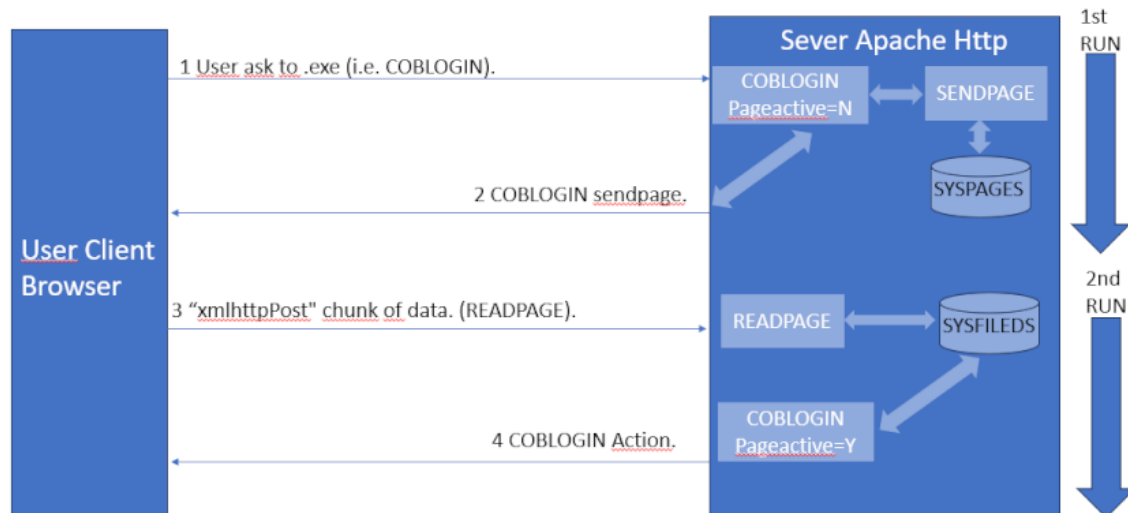
- if (btn == 'menu') {
- var prog = 'COBOLMENU';
- var page = 'MENU';
- var action = 'C';
- var active = 'N';
- }
- if (btn == 'save') {
- var prog = 'COBNEWUSR';
- var page = 'USERS';
- var action = 'C';
- var active = 'Y';
- }

Where we code the main program to be called, the page to be used, the action like to use and the active or not status (remember first or second run).

## 6.- How it works.

In the real on-line world, we may care about two sides of the same work, at one we have a user with a keyboard and a dangerous mouse doing crazy thinks. We must expect for unexpected thinks.

In the other side we have our system and programs.



User loads a page, click a button or ask for some action, that is send to the server where we have out http, this reacts load the desired program (first run) calls de SENDPAGE and send the page to the user and ends.

When the user reacts again the page fields content is sent to the READPAGE and the program is called again (second run), get the field data and act with their logic calls the other program/page and ends again. So the total amount of time and memory used by a program is very low and results free when finish.