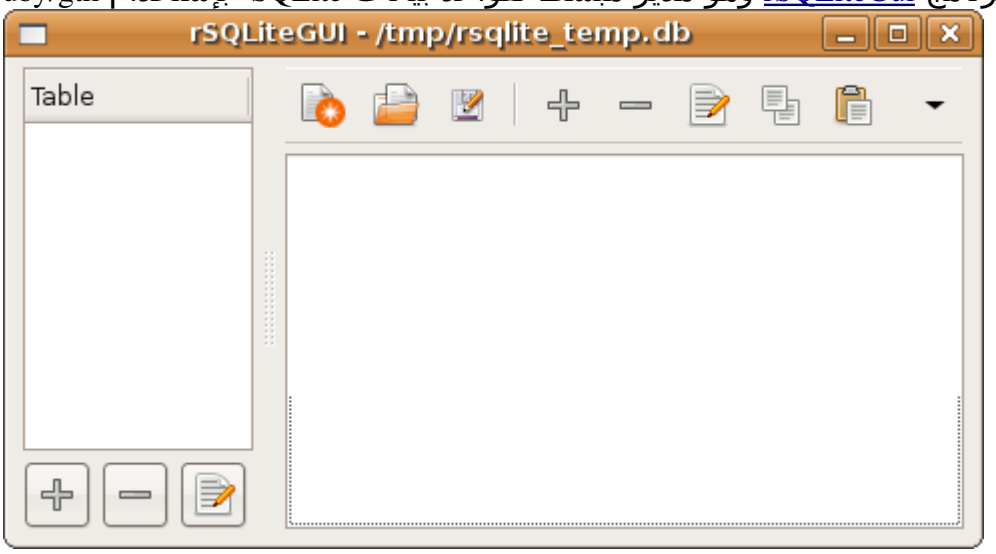
rSQLiteGUI Case Study

هنتكلم عن برنامج rSQLiteGui وهو مدير مبسط لقواعد بيانات SQLite بإستخدام ruby/gtk

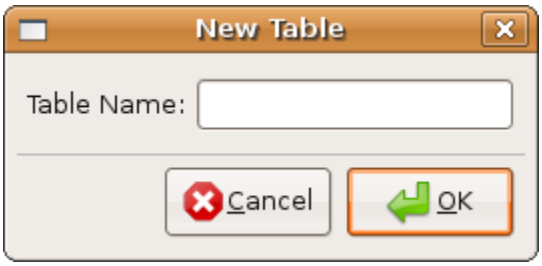


لاحظ الشاشة مقسومة لنصين افقى
على اليمين يوجد toolbar فيه new, open, save, add_row, del_row, edit_row, copy_row, paste_row, execute SQL

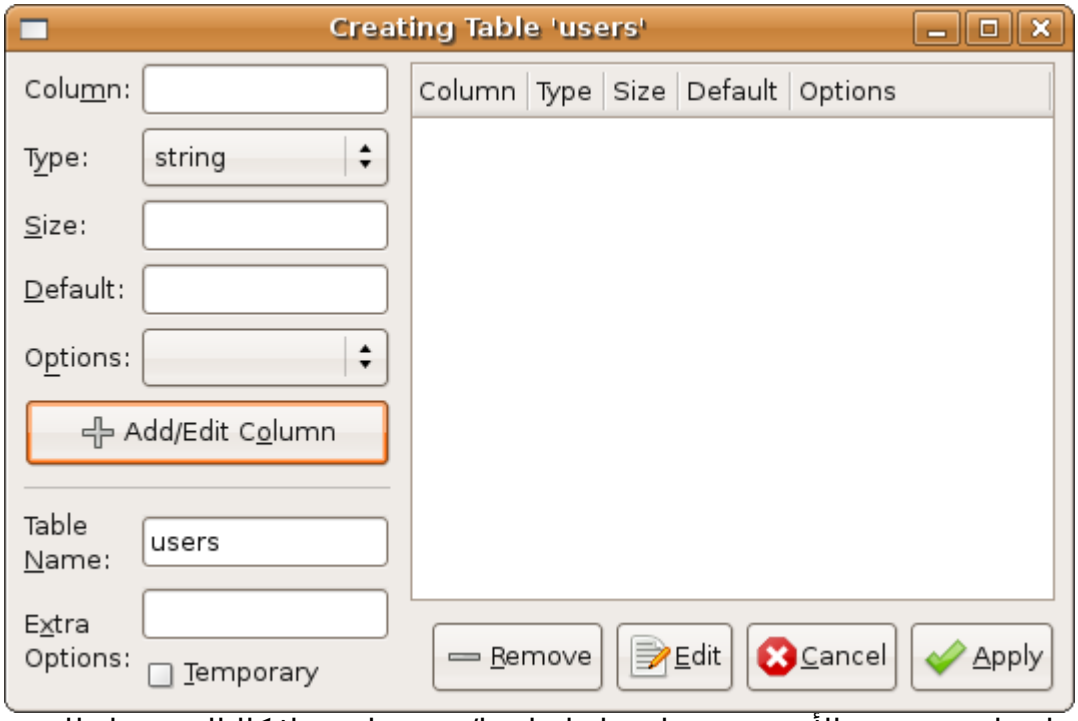
new لإنشاء قاعدة بيانات (ملف واحد بما اننا بنتكلم على sqlite)
open لفتح قاعدة بيانات موجودة بالفعل
save للحفظ
add_row لإضافة صف جديد فى جدول
del_row لحذف السطر المختار من جدول
edit_row لتحرير قيم الصف
copy_row لتخزين القيم الموجودة بالصف المختار لإستخدامها فى عملية لصق
paste_row للصق البيانات افتراضيا فى نافذة الإضافة

اسفل ال toolbar يوجد ScrolledWindow وتحتوى على TreeView لتعبر عن صفوف الجدول

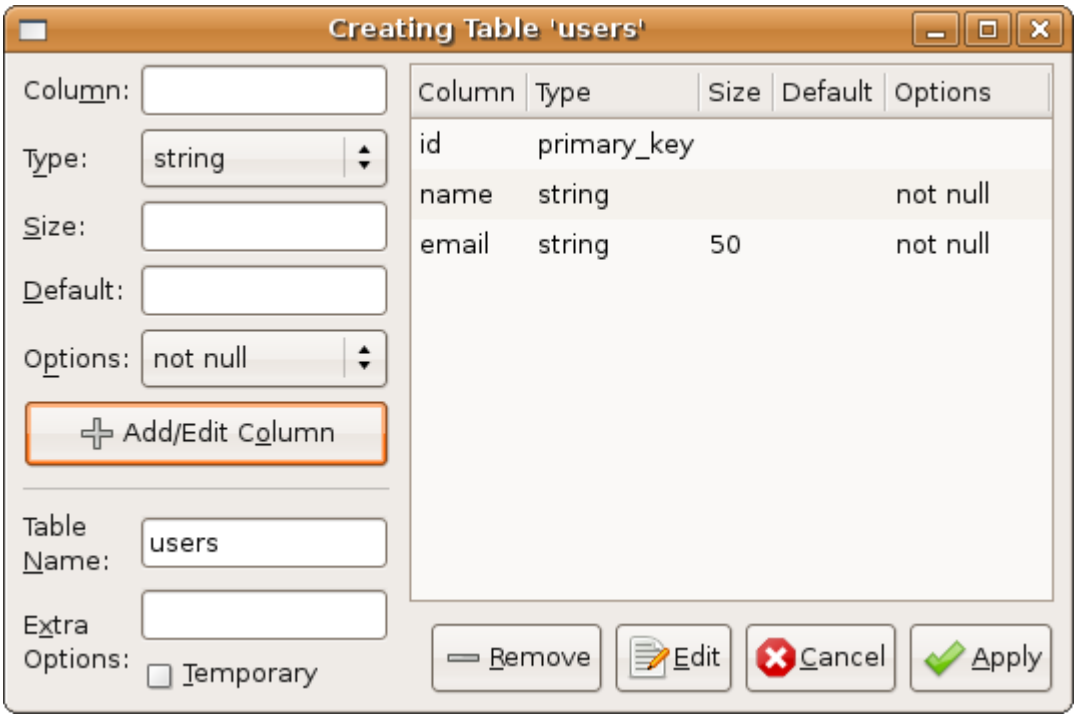
فى القسم على اليسار هنجد tables_container ودى بتشمل
ScrolledWindow بتشمل TreeView ليتم اضافة اسماء الجداول فيها
وتحتها يوجد Hbox بيشمل العمليات الاساسية(اضافة ، حذف ، تحرير)
لإضافة جدول جديد



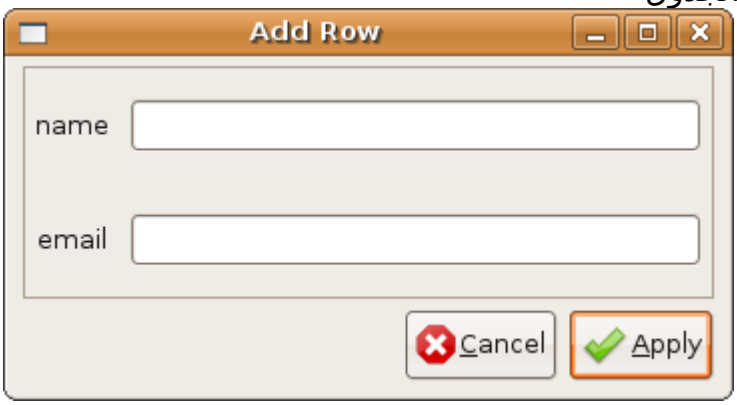
dialog بيتم فيه تحديد اسم الجدول



لإضافة جدول ما بيتم تحديد الأعمدة وخصائصها واضافتها/تحريرها .. مثلا كالتالى جدول للمستخدمين ب id, name, email

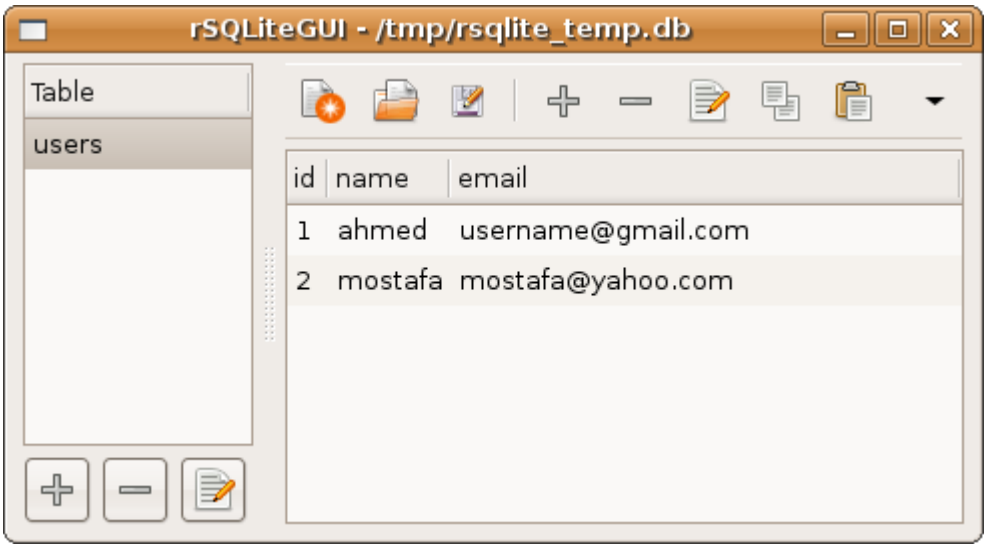


لإضافة صف جديد فى الجدول



dialog ديناميكى بيتم تجهيز حقوله بناء على اسماء الأعمدة بالجدول لإضافة الصفوف..

بعد ادخال بعض القيم المفروض يكون مشابه للتالى ..



كدا خدنا نبذة عن البرنامج وبيعمل ايه .. ناقص نكتب الكود
كدا قشطة.. تصميم الواجهة موجود فى ملف بإمتداد .glade اكيد استخدم ديزينر بدل الكتابة باليد..

اولا المتطلبات
1- active record
2- sqlite3
3- gtk
4- libglade (للتعامل مع الملف .glade "ملف xml" ممثل فيه الواجهة اللى اتكتبت ب glade/glade-like)
5- fileutils للتعامل مع الملفات

```
require 'gtk2'
require 'libglade2'
require 'fileutils'
begin require 'rubygems'; rescue LoadError; end
begin
```

```
  require 'active_record'
rescue LoadError
  puts "ActiveRecord is not installed."
  exit
end
```

تمام ننشئ class بإسم RSQLite

```
class RSQLite
  def initialize(adapter='sqlite3')
    @glade = GladeXML.new( File.join( File::dirname(__FILE__), 'rsqlitegui.glade' ) ) { |handler|
method(handler) }
    @tables   = @glade['tables']
    @db_table = @glade['table']
    @columns  = @glade['columns']
    @adapter  = adapter
    create_tables_view
    new_database
  end
```

لاحظ ال constructor حددنا فيه ال adapter او نوع الداتابيز من خلال named parameter "يعنى ليه قيمة افتراضية" وهى sqlite3 فى حال عدم تحديده

@glade هو GladeXML object بيمثل ملف ال glade. "اللى فيه وصف التصميم"

@tables متغير بيعبر عن الجداول الموجودة فى قاعدة البيانات "تكلمنا عنها فى وصف التصميم"

@db_table متغير بيعبر عن الجدول الحالى

@columns متغير بيعبر عن ال TreeView اللى هيظهر عليها الأعمدة

create_tables_view هى ميثود بتجهز ال tables TreeView من حيث المودل والريندرر وهكذا

```
  def create_tables_view
    renderer = Gtk::CellRendererText.new
    @tables.append_column( Gtk::TreeViewColumn.new('Table', renderer, :text => 0) )
    @tables.get_column(0).set_property('resizable', true)
    @tables.model = Gtk::ListStore.new(String)
  end
```

new_database هى ميثود لإنشاء قاعدة بيانات فى ال tmp dir عند بداية البرنامج..

```
  def new_database(widget=nil)
    temp_file = PLATFORM =~ /linux|bsd/ ? '/tmp/rsqlite_temp.db' : ENV['TEMP'] + 'rsqlite_temp.db'
    File.open(temp_file, 'w') { " }
    open_database(temp_file)
  end
```

open_database هى ميثود لفتح ملف قاعدة بيانات ما

```
  def open_database(dbfile)
    begin
      ActiveRecord::Base.establish_connection( { :adapter => @adapter, :dbfile => dbfile } )
      @conn = ActiveRecord::Base.connection
    rescue ActiveRecord::AdapterNotFound
      display_error_dialog("The #{@adapter} adapter is not installed.")
      return
```

```
  end
  tables = @conn.tables or Array.new # temp db will have no tables to begin with
  populate_tables_view(tables)
  @tables.set_cursor( Gtk::TreePath.new('0'), nil, false )
  @dbfile = dbfile
  @glade['rsqlite'].title = "rSQLiteGUI - #{@dbfile}"
end
```

(اكيد هنستخدم مزايا ال active_record) زى ماتعرضنا فى جزئية ال active_record هننشئ اتصال ونحدد الأدبتر (فى حال عدم وجوده هيكون فى exception AdapterNotFound) وملف قاعدة البيانات ونحصل على connection object هنا @conn

وفى حال انشاء الإتصال هنحصل على الجداول بإستخدام tables method. الموجودة بال connection object او فى حال عدم وجود جداول (ملف فاضى) نعيد مصفوفة فارغة ونجهز قسم الجداول ب المصفوفة العائدة سواء كانت تحوى قيم او لأ من خلال ال populate_tables_view

نخلى المسار قاعدة البيانات متغير داخلى من خلال اسناده ل @dbfile ..ونعدل عنوان النافذة إلى مسار الملف

```
@glade['rsqlite'].title = "rSQLiteGUI - #{@dbfile}"
```

لاحظ اننا بإستدعائنا ل @glade['rsqlite'] بنحصل على اوبجكت خاص بالنافذة اللى بيحوى title method لتحديد العنوان (راجع جزئية ال gtk)

```
def open_database(dbfile)
  begin
    ActiveRecord::Base.establish_connection( { :adapter => @adapter, :dbfile => dbfile } )
    @conn = ActiveRecord::Base.connection
  rescue ActiveRecord::AdapterNotFound
    display_error_dialog("The #{@adapter} adapter is not installed.")
    return
  end
  tables = @conn.tables or Array.new # temp db will have no tables to begin with
  populate_tables_view(tables)
  @tables.set_cursor( Gtk::TreePath.new('0'), nil, false )
  @dbfile = dbfile
  @glade['rsqlite'].title = "rSQLiteGUI - #{@dbfile}"
end
```

بنشئ عمود بإسم tables وبرندرر نصى والمودل من قسم واحد

```
def create_tables_view
  renderer = Gtk::CellRendererText.new
  @tables.append_column( Gtk::TreeViewColumn.new('Table', renderer, :text => 0) )
  @tables.get_column(0).set_property('resizable', true)
  @tables.model = Gtk::ListStore.new(String)
end
```

populate_tables_view بتقوم بملء ال model بالمصفوفة التى تشمل العناصر

```
def populate_tables_view(tables)
  @tables.model.clear
  tables.each do |table|
    iter = @tables.model.append
```

```
    iter.set_value(0, table)
  end
end
```

save_database_as ميثود ل حفظ قاعدة البيانات

```
def save_database_as(widget)
  dialog = Gtk::FileChooserDialog.new( "Save Database", nil,
          Gtk::FileChooser::ACTION_SAVE, nil,
          [Gtk::Stock::CANCEL, Gtk::Dialog::RESPONSE_CANCEL],
          [Gtk::Stock::OPEN,   Gtk::Dialog::RESPONSE_ACCEPT] )
  if dialog.run == Gtk::Dialog::RESPONSE_ACCEPT
    begin
      FileUtils.copy(@dbfile, dialog.filename)
      @dbfile = dialog.filename
      @glade['rsqlite'].title = "rSQLiteGUI - #{@dbfile}"
      ActiveRecord::Base.establish_connection( { :adapter => @adapter, :dbfile => @dbfile } )
    rescue RuntimeError => error
      display_error_dialog(error)
    end
  end
  dialog.destroy
end
```

بنعرض dialog للحفظ

```
  dialog = Gtk::FileChooserDialog.new( "Save Database", nil,
          Gtk::FileChooser::ACTION_SAVE, nil,
          [Gtk::Stock::CANCEL, Gtk::Dialog::RESPONSE_CANCEL],
          [Gtk::Stock::OPEN,   Gtk::Dialog::RESPONSE_ACCEPT] )
  if dialog.run == Gtk::Dialog::RESPONSE_ACCEPT
```

وفى حال ضغط المستخدم للحفظ هيتم عمل نسخة من قاعدة البيانات "الملف" ونسخها للمسار اللى تم اختياره.. ويلى ذلك تعديل المتغيرات الداخلية مثل مسار الملف و عنوان النافذة وإنشاء اتصال جديد بالمسار الجديد

```
FileUtils.copy(@dbfile, dialog.filename)
      @dbfile = dialog.filename
      @glade['rsqlite'].title = "rSQLiteGUI - #{@dbfile}"
      ActiveRecord::Base.establish_connection( { :adapter => @adapter, :dbfile => @dbfile } )
```

quit عند الخروج من البرنامج يجب حذف قاعدة البيانات المؤقته

```
def quit(widget)
  temp_file = PLATFORM =~ /linux|bsd/ ? '/tmp/rsqlite_temp.db' : ENV['TEMP'] + '\rsqlite_temp.db'
  FileUtils.remove(temp_file) if FileTest.exist?(temp_file)
  Gtk.main_quit
end
```

load_table ميثود لقراءة جدول ما بالإسم فيتم الحصول على الأعمدة والصفوف به وتمثيلهم للمستخدم من خلال create_columns و create_model و populate_model

```
def load_table(table)
  begin
    columns = @conn.columns(table)
```

```
    rows    = @conn.select_all("select * from #{table}")
    create_columns(columns)
    create_model(columns.size)
    populate_model(rows, columns)
  rescue ActiveRecord::StatementInvalid => error
    display_error_dialog(error)
  end
end
```

create_columns

لإنشاء ال headers على ال treeview (الخاصة بتمثيل الجدول)

```
def create_columns(columns)
  @db_table.columns.each { |column| @db_table.remove_column(column) }
  renderer = Gtk::CellRendererText.new
  columns.each_with_index do |column, i|
    @db_table.append_column( Gtk::TreeViewColumn.new(column.name, renderer, :text => i) )
    @db_table.get_column(i).set_property('resizable', true)
  end
end
```

create_model

```
def create_model(num_columns)
  column_types = [String] * num_columns
  @db_table.model = Gtk::ListStore.new(*column_types)
end
```

بنحدد عن اقسام ال model "الصف الواحد" (عدد ال headers) للجدول

populate_model

```
def populate_model(rows, columns)
  columns.map! { |column| column.name }
  rows.each do |row|
    iter = @db_table.model.append
    row.each { |key, value| iter.set_value( columns.index(key), value ) }
  end
end
```

نتكلم عن ال helpers dialogs اللى تعرضنا لوصفهم فى جزئية التصميم

```
# ----- Dialogs ---- #
```

display_error_dialog

ميثود لعرض الإيرور بصورة dialog بدلا عرضها على ال stdout

```
def display_error_dialog(error)
  md = Gtk::MessageDialog; modal = Gtk::Dialog::Flags::MODAL
  dialog = md.new(nil, modal, md::ERROR, md::BUTTONS_OK, error)
  dialog.run; dialog.destroy
end
```

display_confirm_dialog

لعرض رسالة تأكيد على المستخدم

```
def display_confirm_dialog(message)
  md = Gtk::MessageDialog; modal = Gtk::Dialog::Flags::MODAL
  dialog = md.new(nil, modal, md::QUESTION, md::BUTTONS_YES_NO, message)
  dialog.signal_connect('key_press_event') do |widget, event|
    dialog.response(md::RESPONSE_NO) \
      if event.keyval == Gdk::Keyval::GDK_Escape
  end
  response = dialog.run; dialog.destroy
  return response == md::RESPONSE_YES
end
```

لأخد ناتج من المستخدم

```
def get_input(title, label, default=nil)
  dialog = Gtk::Dialog.new( title, nil, Gtk::Dialog::MODAL,
          [Gtk::Stock::CANCEL, Gtk::Dialog::RESPONSE_REJECT],
          [Gtk::Stock::OK,     Gtk::Dialog::RESPONSE_ACCEPT] )
  dialog.default_response = Gtk::Dialog::RESPONSE_ACCEPT
  dialog.signal_connect('key_press_event') do |widget, event|
    dialog.response(Gtk::Dialog::RESPONSE_REJECT) \
      if event.keyval == Gdk::Keyval::GDK_Escape
  end

  label = Gtk::Label.new(label).show
  entry = Gtk::Entry.new.show
  entry.text = default unless default.nil?
  entry.activates_default = true

  hbox = Gtk::HBox.new(false, 5).show
  hbox.border_width = 5
  hbox.add(label); hbox.add(entry)
  dialog.vbox.add(hbox)

  response = dialog.run
  input = entry.text
  dialog.destroy

  return nil if response != Gtk::Dialog::RESPONSE_ACCEPT
  return input.strip.empty? ? nil : input
end
```

ينشئ label, textentry وبياخد ناتج ادخال المستخدم (طبعا السؤال بيتم تحديده فى ال method paramters)

display_sql_dialog

بياخد استعلام ويقوم بتنفيذه

```
def display_sql_dialog(sql_text=nil)
  dialog = Gtk::Dialog.new( 'SQL', nil, Gtk::Dialog::MODAL,
          [Gtk::Stock::CANCEL, Gtk::Dialog::RESPONSE_REJECT],
          [Gtk::Stock::OK,     Gtk::Dialog::RESPONSE_ACCEPT] )
  dialog.default_response = Gtk::Dialog::RESPONSE_ACCEPT
  dialog.signal_connect('key_press_event') do |widget, event|
    dialog.response(Gtk::Dialog::RESPONSE_REJECT) \
      if event.keyval == Gdk::Keyval::GDK_Escape
  end

  container = Gtk::ScrolledWindow.new.show
  container.set_policy(Gtk::POLICY_AUTOMATIC, Gtk::POLICY_AUTOMATIC)
  entry = Gtk::TextView.new.show
  unless sql_text.nil?
    entry.buffer.text = sql_text
    entry.editable = false
  end
  container.add(entry)

  hbox = Gtk::HBox.new(false, 5).show
  hbox.border_width = 5
  hbox.add(container)
  dialog.vbox.add(hbox)
  dialog.set_size_request(320, 240)

  response = dialog.run
  input = entry.buffer.text
  dialog.destroy

  return nil if response == Gtk::Dialog::RESPONSE_REJECT
  return ( input.strip.empty? ) ? nil : input
end
```

لعرض الحقول المطلوبة ماعدا ال PK بالتأكيد (على افتراض انه serial)

```
def display_fields_dialog(fields, title='rSQLiteGUI', values=nil, create=nil)
  create = values.nil? if create.nil?
  window = Gtk::Window.new(title)
  window.modal = true
  window.set_border_width(5)
  window.signal_connect('key_press_event') do |widget, event|
    window.destroy if event.keyval == Gdk::Keyval::GDK_Escape
  end

  container = Gtk::ScrolledWindow.new
  container.set_policy(Gtk::POLICY_AUTOMATIC, Gtk::POLICY_AUTOMATIC)
```

```
  table = Gtk::Table.new(fields.size, 2, false)
  entries = Array.new
  primary_key = @conn.primary_key(current_table)
  fields.each_with_index do |field, i|
    next if field.name == primary_key and create # don't fill in id when creating
    label = Gtk::Label.new(field.name)
    entry = Gtk::Entry.new
    entry.text = values[i] or '' unless values.nil?
    entry.activates_default = true
    table.attach(label, 0, 1, i, i + 1, Gtk::FILL, Gtk::FILL, 5, 5)
    table.attach(entry, 1, 2, i, i + 1, Gtk::FILL | Gtk::EXPAND, Gtk::FILL | Gtk::EXPAND, 5, 5)
    entries << entry
  end
  container.add_with_viewport(table)

  apply = Gtk::Button.new(Gtk::Stock::APPLY)
  apply.signal_connect('clicked') { row_action(entries, create); window.destroy }
  cancel = Gtk::Button.new(Gtk::Stock::CANCEL)
  cancel.signal_connect('clicked') { window.destroy }

  hbox = Gtk::HBox.new(false, 5)
  hbox.pack_start(Gtk::Label.new, true, false, 0)
  hbox.pack_start(cancel, false, false, 0)
  hbox.pack_start(apply, false, false, 0)

  vbox = Gtk::VBox.new(false, 5)
  vbox.pack_start(container, true, true, 0)
  vbox.pack_start(hbox, false, false, 0)

  window.add(vbox)
  height = ( 40 * fields.size > 480 ) ? 480 : 40 * fields.size + 40
  window.set_size_request(320, height)
  window.show_all
  apply.has_default = apply.can_default = true
end

def row_action(entries, create=true)
  columns = @conn.columns(current_table).map { |column| column.name }
  entries.map! { |entry| "\"#{entry.text.gsub(/'/, "'")}\"" }
  primary_key = @conn.primary_key(current_table)
  if create
    columns.delete(primary_key)
    sql = "insert into %s (%s) values(%s)" %
          [ current_table, columns.join(','), entries.join(',') ]
    begin
      @conn.insert(sql)
```

```
    rescue ActiveRecord::StatementInvalid => error
      display_error_dialog(error)
    end
  else
    primary_index = columns.index(primary_key)
    columns.each_with_index do |column, i|
      next if i == primary_index
      sql = "update %s set %s = %s where %s = %s" %
            [ current_table, column, entries[i], primary_key, entries[primary_index] ]
      begin
        @conn.update(sql)
      rescue ActiveRecord::StatementInvalid => error
        display_error_dialog(error)
      end
    end
  end
  load_table(current_table)
end

end
```

التعامل مع الجدول

add_table لإضافة جدول (الحصول على اسمه من خلال get_input helper) ويتم الإضافة

```
def add_table(widget)
  if name = get_input("New Table", "Table Name:")
    unless @conn.tables.index(name)
      init_table_view(name)
      @glade['table_view'].show
    else
      display_error_dialog("Table '#{name}' already exists.")
    end
  end
end
```

drop_table لحذف جدول بإستخدام drop_table method الموجودة بال connection object

```
def drop_table(widget)
  return unless current_table
  message = "Are you sure you want to drop the table '#{current_table}'?"
  if display_confirm_dialog(message)
    @conn.drop_table(current_table)
    @tables.model.remove( @tables.model.get_iter( @tables.cursor[0] ) )
    @tables.set_cursor( Gtk::TreePath.new('0'), nil, false )
  end
end
```

لإعادة تسمية الجدول بيتم استخدام rename_table method الموجودة بال connection object بعد

استدعاء اسم الجدول الجديد(من ال get_input helper)

```
def rename_table(widget)
  if current_table and new_name = get_input("Rename Table", "New Name:")
    @conn.rename_table(current_table, new_name)
    @current_table = new_name
    iter = @tables.model.get_iter( @tables.cursor[0] )
    iter.set_value(0, new_name)
    @tables.grab_focus
  end
end
```

عند اختيار جدول ما يتم استدعاء load_table عليه (cursor_changed)

```
def select_table(widget)
  path = widget.cursor[0]
  if path
    iter = widget.model.get_iter(path)
    @current_table = iter.get_value(0)
    load_table(current_table)
  else
    @current_table = nil
    @db_table.model = nil
  end
end
```

current_table هى getter للحصول على الجدول الحالى

```
def current_table
  @current_table
end
```

العمليات على الجدول

```
# ----- Table Actions ----- #
```

execute_sql هى ميثود بتقوم بتنفيذ استعلام مين هيدخله المستخدم عن طريق display_sql_dialog helper ويتم تنفيذ الإستعلام عن طريق ال execute ميثود الخاصة بال connection object...

```
def execute_sql(widget)
  if sql = display_sql_dialog
    begin
      out = @conn.execute(sql)
    rescue ActiveRecord::StatementInvalid => error
      display_error_dialog(error)
      return
    end
    formatted = String.new
    case out
    when String: formatted = out
    when Array
      out.each do |item|
        item.delete_if { |k, v| k.is_a? Integer }
```

```
      formatted += item.inspect + "\n"
    end
  end
  display_sql_dialog(formatted) unless formatted.empty?
  cursor = @tables.cursor[0]
  populate_tables_view(@conn.tables)
  @tables.set_cursor( cursor.nil? ? Gtk::TreePath.new('0') : cursor, nil, false )
 end
end
```

لإضافة صف جديد (للحصول على الأعمدة بيتم استخدم ال columns method الخاصة بال connection object) ودا طبعا فى حال وجود جدول حالى..

```
def add_row(widget)
 display_fields_dialog( @conn.columns(current_table), 'Add Row' ) if current_table
end
```

تحرير صف ما، نفس السابقة مع عرض ال dialog مجهز بالقيم السابقة التى سيتم تحريرها..

```
def edit_row(widget, path=nil, column=nil)
 path = @db_table.cursor[0] if path.nil?
 return if path.nil?
 iter = @db_table.model.get_iter(path)
 columns = @conn.columns(current_table)
 values = Array.new
 columns.size.times { |i| values << iter.get_value(i) }
 display_fields_dialog(columns, 'Edit Row', values)
end
```

نسخ قيم صف معين .. لاحظ ال \001 هيستخدم ك joiner بين القيم

```
def copy_row(widget, path=nil, column=nil)
 path = @db_table.cursor[0] if path.nil?
 return if path.nil?
 iter = @db_table.model.get_iter(path)
 columns = @conn.columns(current_table)
 values = Array.new
 columns.size.times { |i| values << iter.get_value(i) }
 clipboard = Gtk::Clipboard.get(Gdk::Selection::CLIPBOARD)
 clipboard.text = values.join("\001")
end
```

لصق صف معين.. وللحصول على القيم يتم عمل split للقيم عن طريق ال \001

```
def paste_row(widget)
 return unless current_table
 columns = @conn.columns(current_table)
 clipboard = Gtk::Clipboard.get(Gdk::Selection::CLIPBOARD)
 values_text = clipboard.wait_for_text or ''
 values = values_text.split("\001")
 # NOTE: is this necessary? (submitted by Benjamin Bock)
 # try to be intelligent: don't expect an id if there are fewer values than columns
 values.unshift '' if columns.size > values.size
 display_fields_dialog(columns, 'Paste Row', values, true)
```

```
end
```

حذف صف ما.. يتم الحصول على ال PK الخاص به والحصول على ال index تبعه وتنفيذ جملة delete

```
def remove_row(widget)
  path = @db_table.cursor[0]
  return if path.nil?
  iter = @db_table.model.get_iter(path)
  columns = @conn.columns(current_table).map { |column| column.name }
  primary_key = @conn.primary_key(current_table)
  primary_index = columns.index(primary_key)
  sql = "delete from %s where %s = %s" %
        [ current_table, primary_key, iter.get_value(primary_index) ]
  begin
    @conn.delete(sql)
    load_table(current_table)
  rescue ActiveRecord::StatementInvalid => error
    display_error_dialog(error)
  end
end
```

التعامل مع ال table_view بيتم فيها انشاء الجداول وتحديد الأعمدة وخصائصها الخ الخ

```
# ----- Table View ----- #

def init_table_view(name, editing = false)
  clear_column_details
  @glade['table_view'].title = "#{editing ? 'Editing' : 'Creating'} Table '#{name}'"
  @glade['table_name'].text = name
  @glade['table_name'].editable = editing ? false : true
  @glade['column_type'].active = 1
  @glade['column_options'].active = 0
  @columns.columns.each { |column| @columns.remove_column(column) }
  renderer = Gtk::CellRendererText.new
  %w{Column Type Size Default Options}.each_with_index do |column, i|
    @columns.append_column( Gtk::TreeViewColumn.new(column, renderer, :text => i) )
    @columns.get_column(i).set_property('resizable', true)
  end
  @columns.model = Gtk::ListStore.new(String, String, String, String, String)
  @editing_table = editing
  @glade['column_name'].grab_focus
end
```

لإلغاء تفاصيل العمود

```
def clear_column_details
  @glade['column_name'].text = ''
  @glade['column_size'].text = ''
  @glade['column_default'].text = ''
end
```

إضافة عمود (اسم ونوع ومساحة وقيمة افتراضية وخياراته)

```
def add_column(widget)
  column_data = Array.new
  column_data[0] = @glade['column_name'].text.strip
  column_data[1] = @glade['column_type'].active_text.strip
  column_data[2] = @glade['column_size'].text.strip
  column_data[3] = @glade['column_default'].text.strip
  column_data[4] = @glade['column_options'].active_text.strip
  unless column_data[0].empty?
    iter = @columns.model.append
    5.times { |i| iter.set_value( i, column_data[i] ) }
    clear_column_details
  else
    display_error_dialog("The column must have a name.")
  end
end
```

حذف عمود ما

```
def remove_column(widget)
  path = @columns.cursor[0]
  @columns.model.remove( @columns.model.get_iter(path) ) if path
end
```

تحرير عمود معين

```
def edit_column(widget)
  types = Hash[ *[:primary_key, :string, :text, :integer, :float, :datetime, :timestamp,
                :time, :date, :binary, :boolean].zip( (0..10).collect ).flatten ]
  options = { '' => 0, 'not null' => 1 }
  path = @columns.cursor[0]
  iter = @columns.model.get_iter(path)
  @glade['column_name'].text = iter.get_value(0)
  @glade['column_type'].active = types[ iter.get_value(1).to_sym ]
  @glade['column_size'].text = iter.get_value(2)
  @glade['column_default'].text = iter.get_value(3)
  @glade['column_options'].active = options[ iter.get_value(4) ]
  @columns.model.remove(iter)
  @glade['column_name'].grab_focus
end
```

حفظ التعديلات النهائية

```
def apply_changes(widget)
  if @editing_table
    # TODO:
    # to find db changes, iterate through new columns and see if
    # they match with the old columns
  else
    columns = Array.new
    @columns.model.each do |model, path, iter|
      column = Array.new
```

```
      5.times { |i| column << iter.get_value(i).strip.gsub(' ', '_') }
      columns << Hash[ *[:name, :type, :size, :default, :options].zip(column).flatten ]
    end
    primary_key = nil
    columns.each do |column|
      if column[:type] == 'primary_key'
        primary_key = column[:name]
        columns.delete(column)
        if columns.size == 0
          display_error_dialog("Please specifiy an additional column.")
          return
        end
        break
      end
    end
    table_name = @glade['table_name'].text.strip.gsub(' ', '_')
    extra_table_opts = @glade['extra_table_options'].text.strip
    options = Hash.new
    options[:primary_key] = primary_key if primary_key
    options[:temporary]   = true if @glade['temporary_table'].active?
    options[:options]     = extra_table_opts unless extra_table_opts
    begin
      @conn.create_table( table_name, options ) do |table|
        columns.each do |column|
          column[:name].gsub!(' ', '_')
          options = Hash.new
          options[:limit]   = column[:size].to_i unless column[:size].empty?
          options[:default] = column[:default] unless column[:default].empty? or column[:default] ==
'null'
          options[:null]    = false if column[:options] == 'not null'
          table.column( column[:name].to_sym, column[:type].to_sym, options )
        end
      end
      hide_table_view(nil)
      iter = @tables.model.append
      iter.set_value(0, table_name)
      @tables.set_cursor(iter.path, nil, false)
    rescue ActiveRecord::StatementInvalid => error
      display_error_dialog(error)
    end
  end
end

def hide_table_view(widget, event=nil) @glade['table_view'].hide_on_delete end
```

اخيرا التنفيذ النهائى

```
BEGIN {
  $adapter = 'sqlite3'
  case ARGV.first
  when '-h', '--help'
    load File.join( File::dirname(__FILE__), "help.rb" )
    exit
  when '-s2', '--sqlite2'
    $adapter = 'sqlite'
    ARGV.shift
  end
}

if __FILE__ == $0
  Gtk.init
  app = RSQLite.new($adapter)
  app.open_database(ARGV.first) if ARGV.first
  Gtk.main
end
```

By *Programming-Fr34ks[dot]NET*