

# SQL Yoga

<b>1</b>	<b>Introduction</b>	
1.1	About SQL Yoga and This Manual	6
<b>2</b>	<b>SQL Yoga Primer</b>	
2.1	The Primer Stack	8
2.2	Connect to a Database By Creating Database and Connection Objects	11
2.3	Getting Data From the Database and Displaying It in the UI	13
<b>3</b>	<b>Notes on Error Reporting</b>	
3.1	How SQL Yoga Reports Errors	17
3.2	Errors Thrown From Password Protected Stacks	19
3.3	Revolution Message Box	23
<b>4</b>	<b>General Concepts</b>	
4.1	How To Use the sql_yoga.rev Stack File In the IDE	26
4.2	Unlocking The SQL Yoga Library	30
4.3	How SQL Yoga Represents Objects	31
<b>5</b>	<b>Database Objects</b>	
5.1	Introduction to the Database Object and Database Schema	33
5.2	Creating A Database Object	36
5.3	Saving A Database Object Across Sessions	38
5.4	Loading The Database Object When Your Stack Opens	45
5.5	How To Work With Multiple Database Objects	49
<b>6</b>	<b>Database Connections</b>	
6.1	Introduction to Connection Objects	51

6.2	Connecting to a Database Using a Connection Object	52
6.3	Telling SQL Yoga When Your Database Schema Changes	56
6.4	How To Work With Multiple Connection Objects	57
6.5	My Database Schema Has Columns/Tables That Are Reserved Words? What Can I Do?	58
<b>7</b>	<b>SQL Query Objects</b>	
7.1	Introduction to SQL Query Objects	60
7.2	Creating A SQL Query Object	61
7.3	How Do I Create Records In a Database Table Using a SQL Query Object?	62
7.4	How Do I Delete Records From the Database Table Using a SQL Query Object?	63
7.5	How To Use Cursors With a SQL Query Object	64
7.6	How Can I Convert a User Search String Into An AND/OR Search When Generating a Query?	66
<b>8</b>	<b>Record Objects</b>	
8.1	Introduction to SQL Record Objects	71
8.2	Create Records in the Database Using SQL Record Objects	73
8.3	Retrieve Records from the Database Using SQL Record Objects	74
8.4	Update Records in the Database Using SQL Record Objects	75
8.5	Delete Records from the Database Using SQL Record Objects	76
<b>9</b>	<b>Table Objects</b>	
9.1	Introduction to Table Objects	78
9.2	Familiarizing Yourself With The "table objects behavior" Property	80
9.3	Creating a Table Object	81
9.4	How Do I Add Custom Properties To Tables?	82



# Introduction

## About SQL Yoga and This Manual

---

This manual is the how-to manual for SQL Yoga. SQL Yoga extends the simplicity of Revolution to databases by allowing you to treat your database like an object. Stop wrestling with SQL and see how easy database integration can be:

- Set properties rather than writing SQL
- Define searches using english words rather than arcane wildcard symbols
- Manipulate arrays, not cursors
- Easily generate searches from complex search UIs
- Tap into database table relationships to simplify code

To get the latest information, API documentation, learning materials and SQL Yoga library visit the SQL Yoga web page:

<http://www.bluemangolearning.com/revolution/software/libraries/sql-yoga/>

**IMPORTANT: SQL Yoga Is Built On Top of RevDB And the Valentina (V4REV) External**

SQL Yoga is built on top of RevDB and the Valentina external (V4REV). In order to use SQL Yoga you must have Revolution 3.5 or higher. Your version of Revolution must also support database access **OR** your must own a copy of the [V4REV database external](#).

# SQL Yoga Primer

## The Primer Stack

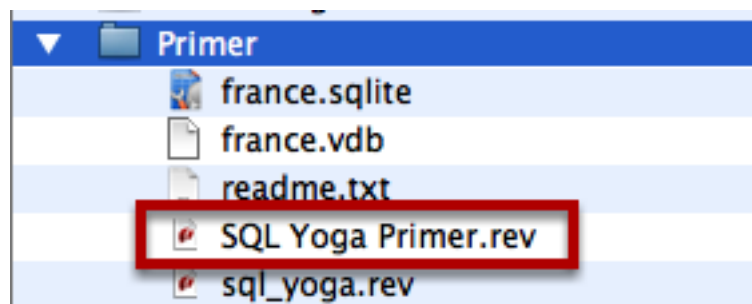
---

The SQL Yoga primer stack is intended to provide a bare bones example stack that shows you how the SQL Yoga objects work together to connect to a database and display data. The next few lessons show you how to use the stack but do not go into too much detail. It is expected that you can use this stack to introduce yourself to the basics of the library. You can then customize it and use it to experiment with as you go through this manual.

You can download the example stack from the following url:

[http://www.bluemangolearning.com/download/revolution/sql\\_yoga/sql\\_yoga\\_primer.zip](http://www.bluemangolearning.com/download/revolution/sql_yoga/sql_yoga_primer.zip)

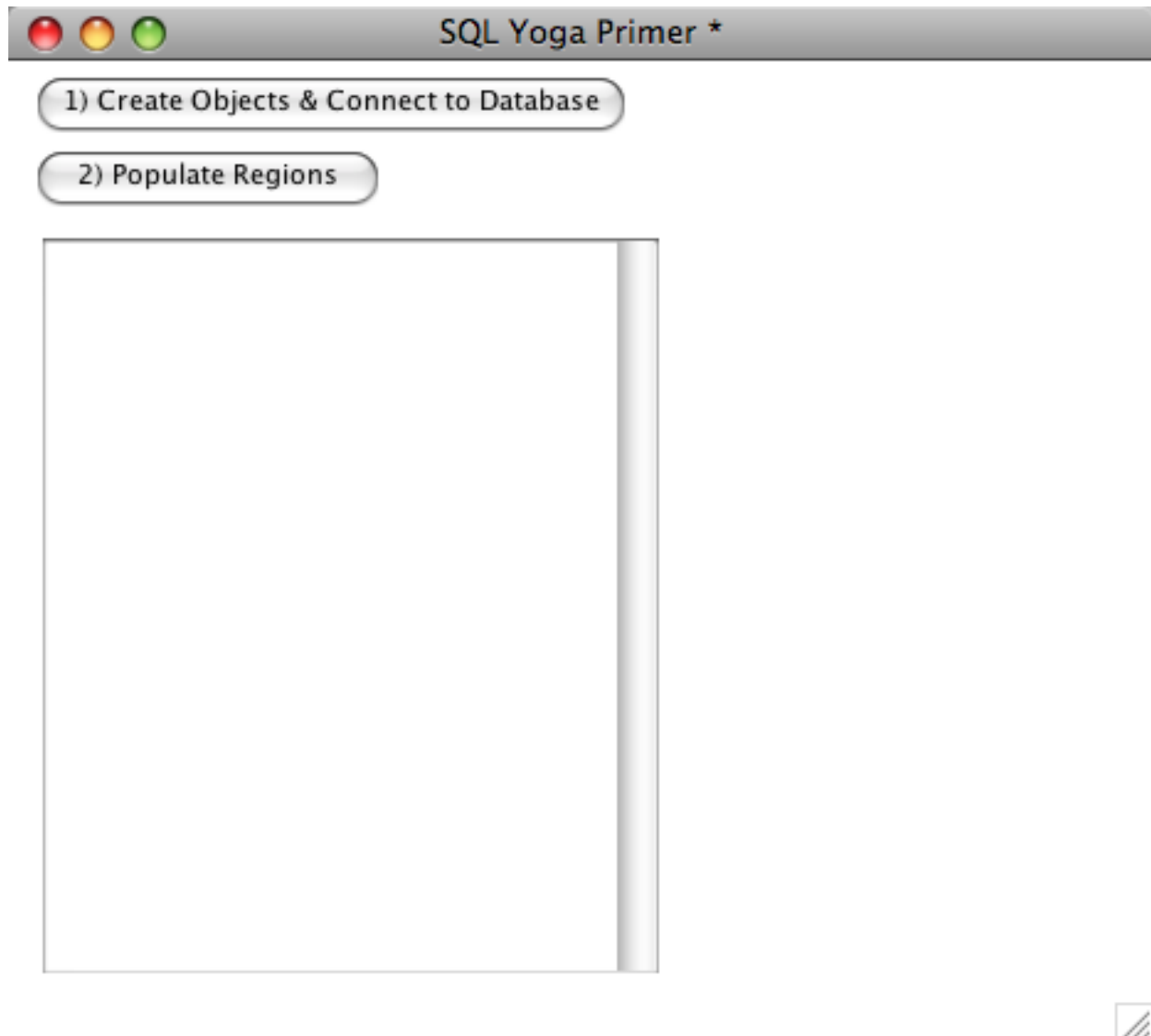
### The Primer Files



After downloading and unzipping the primer ZIP archive you will see these files. Open the **SQL Yoga Primer.rev** file in the Revolution IDE. This stack will open the **sql\_yoga.rev** stack and start using it as a library for you automatically.

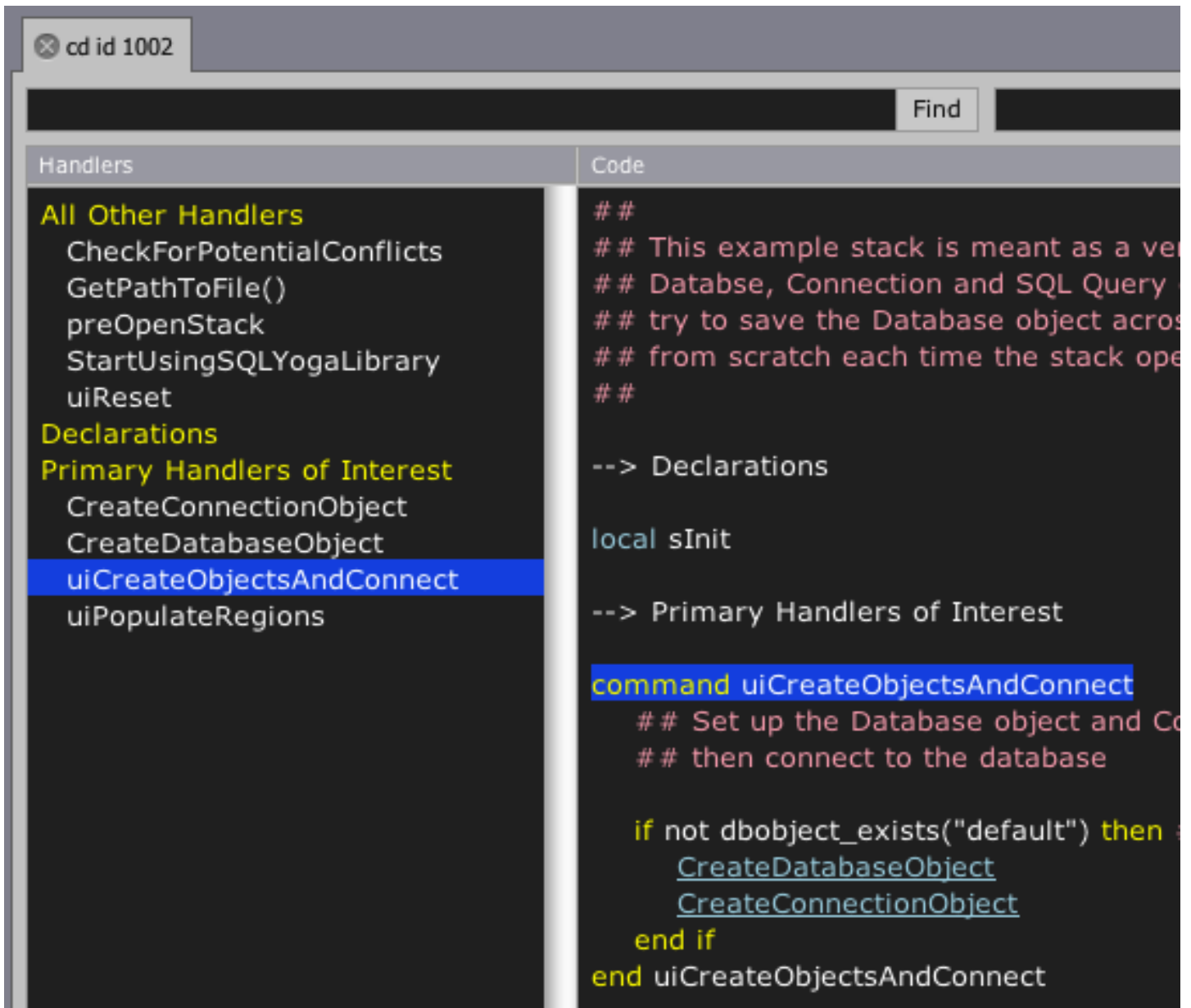


## The Primer Stack



This is what the primer stack looks like in the IDE.

## The Card Script



The screenshot shows a software interface with a tab labeled 'cd id 1002'. At the top right is a 'Find' button. Below it are two panels: 'Handlers' on the left and 'Code' on the right.

**Handlers Panel:**

- All Other Handlers**
  - CheckForPotentialConflicts
  - GetPathToFile()
  - preOpenStack
  - StartUsingSQLYogaLibrary
  - uiReset
- Declarations**
- Primary Handlers of Interest**
  - CreateConnectionObject
  - CreateDatabaseObject
  - uiCreateObjectsAndConnect** (highlighted in blue)
  - uiPopulateRegions

**Code Panel:**

```
##
## This example stack is meant as a ve
## Database, Connection and SQL Query
## try to save the Database object across
## from scratch each time the stack ope
##

--> Declarations

local sInit

--> Primary Handlers of Interest

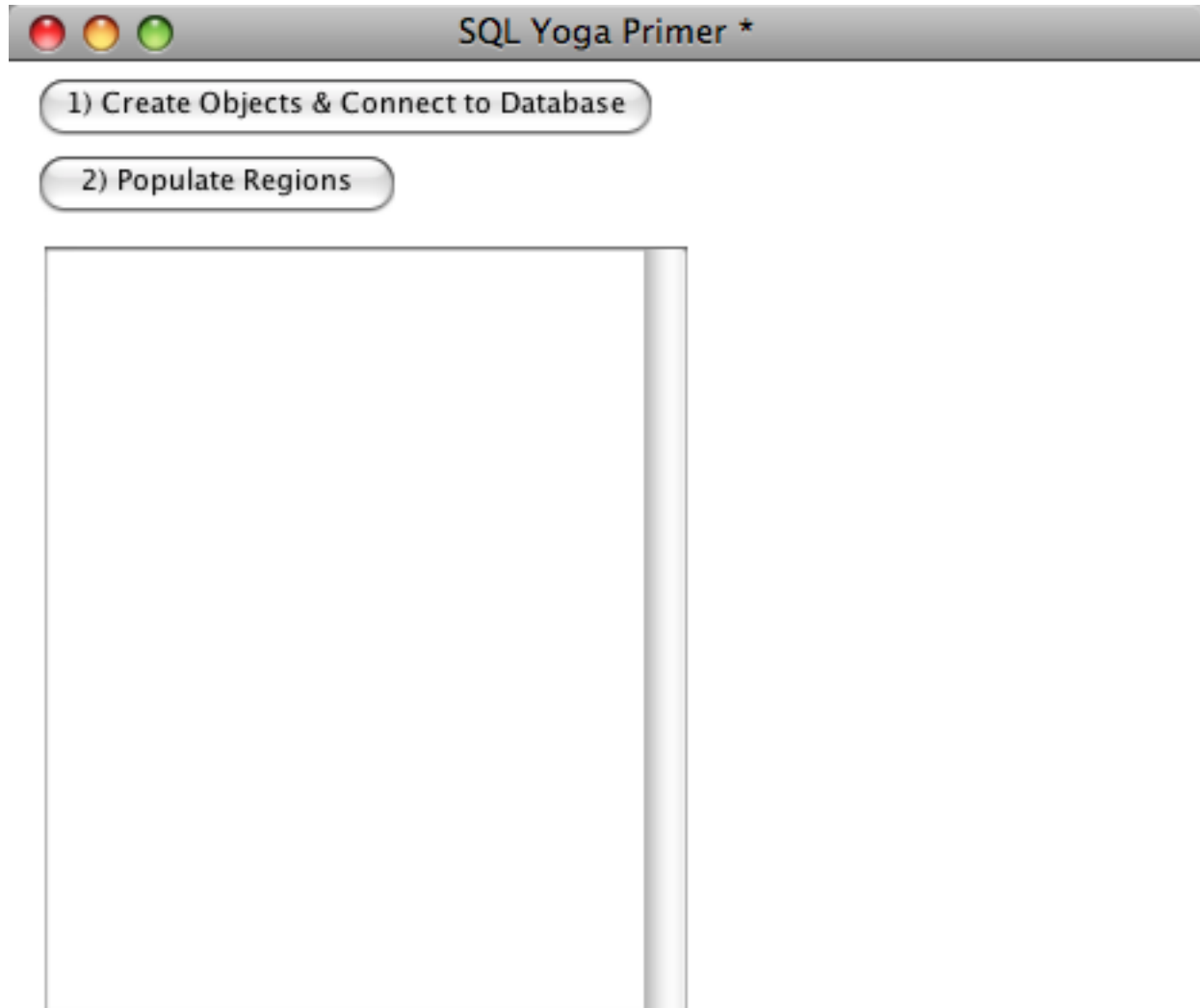
command uiCreateObjectsAndConnect
    ## Set up the Database object and Co
    ## then connect to the database

    if not dbobject_exists("default") then
        CreateDatabaseObject
        CreateConnectionObject
    end if
end uiCreateObjectsAndConnect
```

All of the logic for the primer stack is included in the card script of the **SQL Yoga Primer** stack. Edit the card script to see how the various objects are created and used.

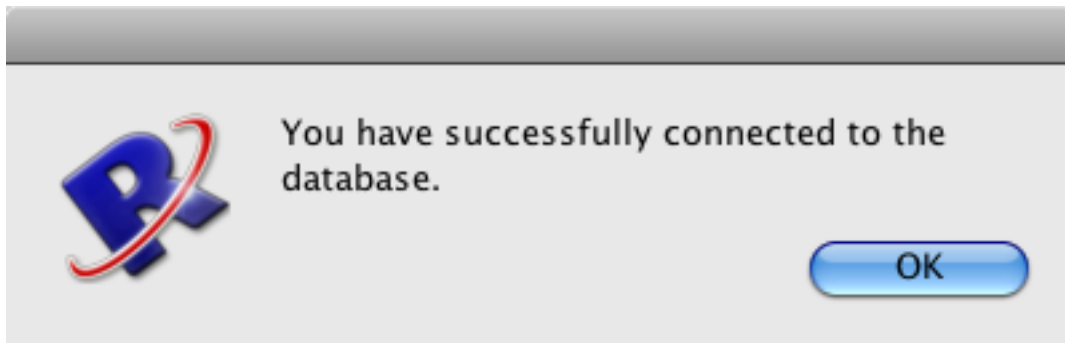
## Connect to a Database By Creating Database and Connection Objects

### Create Objects & Connect to Database



As a developer you interact with the SQL Yoga library by creating objects and setting their properties. The root object in the SQL Yoga library is the **Database** object. A Database object can contain one or more **Connection** objects.

Click the **Create Objects & Connect to Database** button to create a Database and Connection object.



If the connection is successful then you will see this dialog.

```
command CreateConnectionObject
## Create a connection object so that we can connect to the
## database file. The Database object that was created in
## is the default Database object for all SQL Yoga API calls
## will be attached to it.

## Because no other Connection objects exist yet this new
## Connection object will be set as the default connection
## the Database object.

put "french regions" into theConnectionName

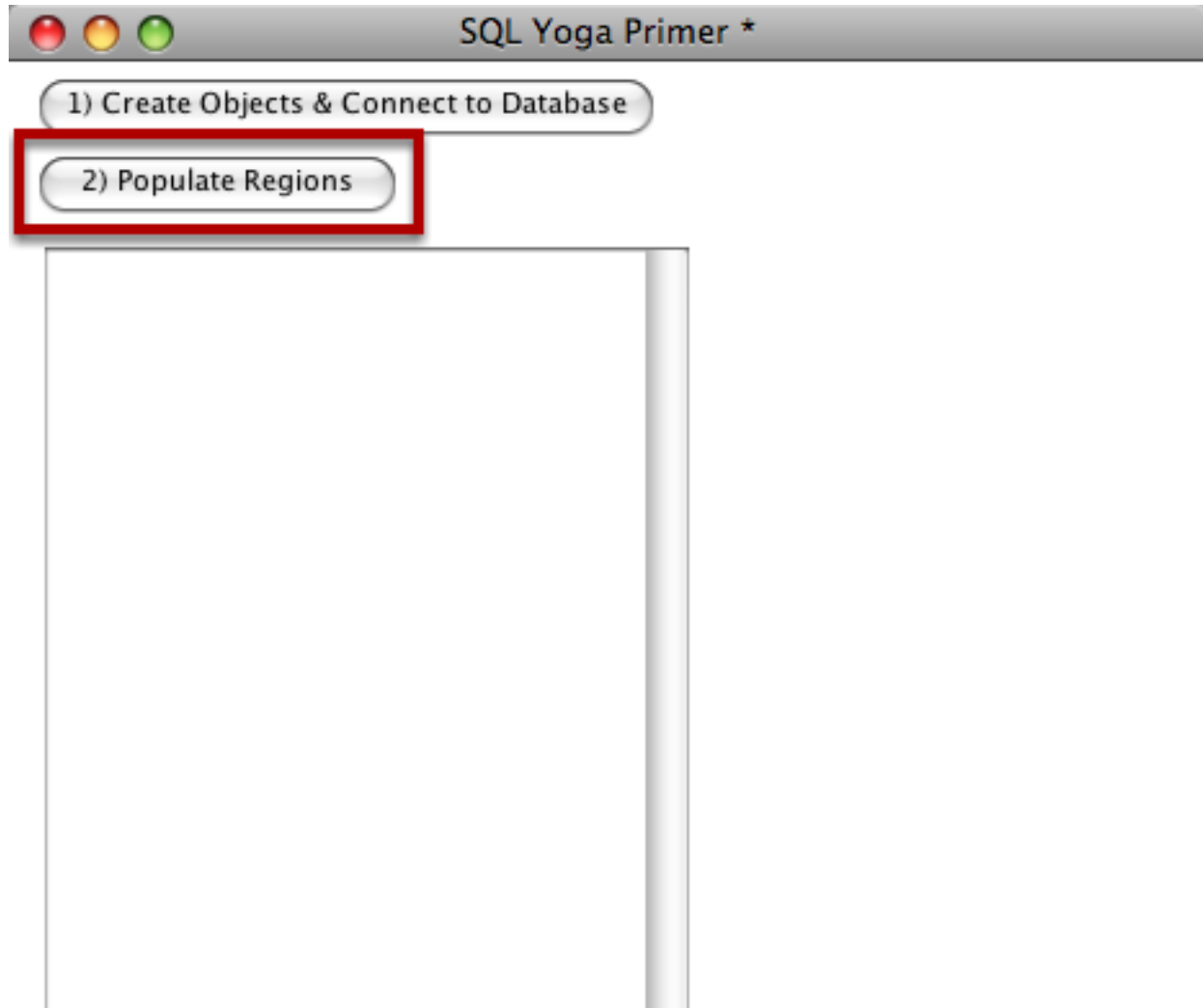
##
## SQLite
1 ##
dbconn_createObject theConnectionName, "sqlite"
put GetPathToFile("france.sqlite") into theDatabaseFile

##
## Valentina
2 ##
-- dbconn_createObject theConnectionName, "valentina"
-- put GetPathToFile("france.vdb") into theDatabaseFile
```

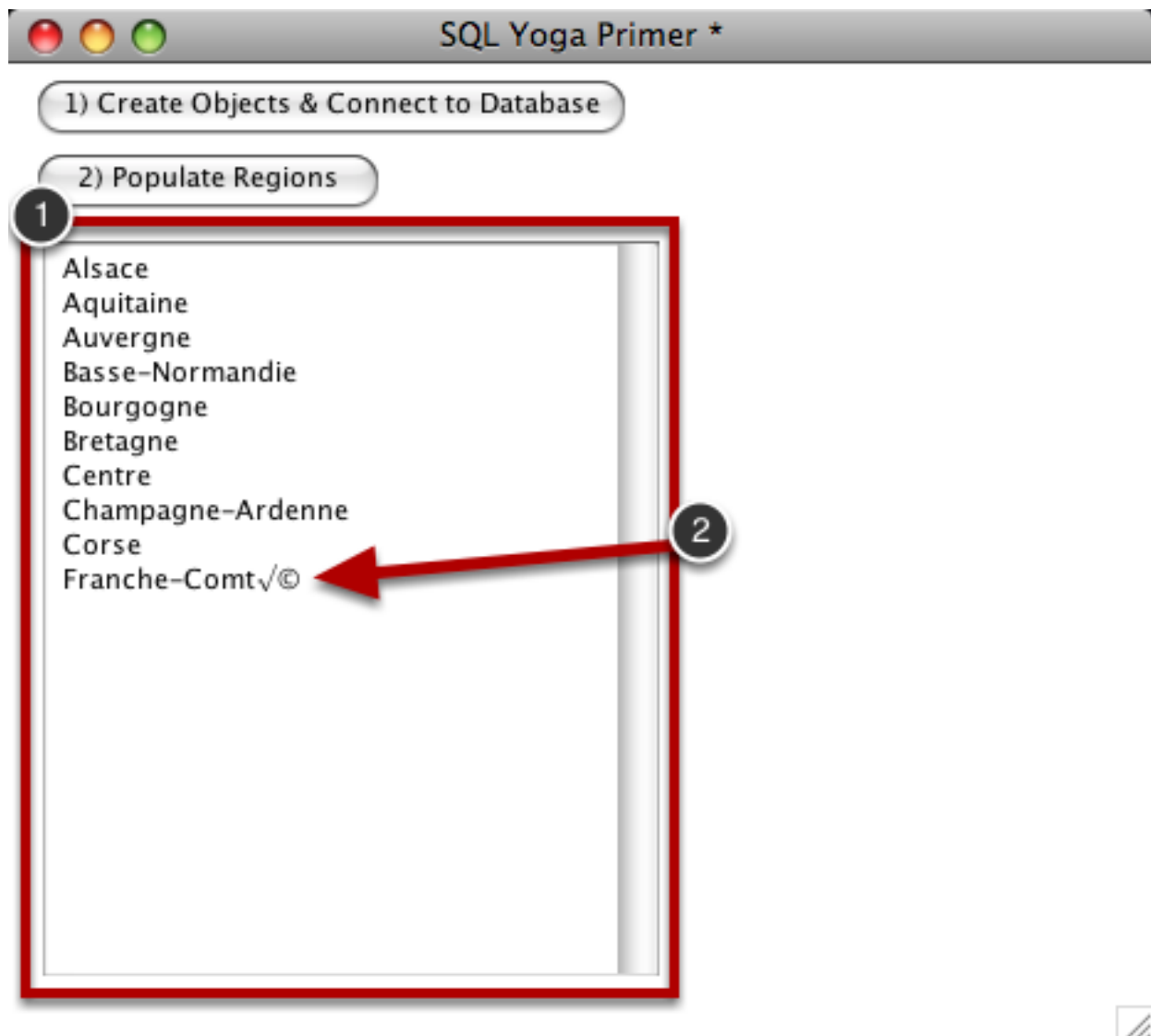
The primer stack will connect to the france.sqlite database file by default. A Valentina database (france.vdb) is also included in the distribution files. If you would like to test with Valentina then comment out the SQLite lines (1) and uncomment the Valentina lines in the **CreateConnectionObject** command of the card script.

## Getting Data From the Database and Displaying It in the UI

### Create SQL Query Object And Retrieve Data From Database



With SQL Yoga the primary means of getting data from a database is to use a SQL Query object. Create a SQL Query object and retrieve data from the database by clicking on the **Populate Regions** button.



After clicking on the button you will see the list field populate with data (1).

**Note:** You may see some encoding issues as the data is stored as UTF-8 (2) and the primer stack does not try to deal with converting the data to a format that can display in a field.

## The uiPopulateRegions Handler

```
command uiPopulateRegions
  local theArrayA

  ## Create a Query object and set properties. The Qu
  ## use the default Database object.
  ## Note that unlike Database and Connetion objects,
  ## are not persistent. They are stored in a variable
  ## and disappear when the variable no longer exists
  put sqlquery_createObject("regions") into theQueryA
  sqlquery_set theQueryA, "order by", "name"

  ## Get results from database and convert to a nume
  ## indexed multi-dimensional array.
  sqlquery_retrieveAsArray theQueryA, theArrayA
  put the result into theError

  if theError is empty then
    ## loop from 1 to number of elements returned in
    repeat with i = 1 to item 2 of line 1 of the extents
      put theArrayA[i]["name"] & cr after theData
    end repeat
    delete the last char of theData

    set the text of field "Regions" to theData
  end if
end uiPopulateRegions
```

You can find the code that the button calls in the **uiPopulateRegions** command of the card script.

# Notes on Error Reporting



## How SQL Yoga Reports Errors

---

Before we continue on to the various SQL Yoga objects and how to use them we are going to mention a few things about how SQL Yoga reports errors. Knowing how to properly track down errors will save you a lot of time and headaches.

### Connection and SQL Query Errors

SQL Yoga will throw an error if something goes wrong when connecting to the database or when executing a SQL query.

If an error occurs while trying to connect to the database then an error prefixed with **sqlyoga\_connection\_err**, will be thrown. Having the error thrown allows your application to handle lost database connections globally. For example, you can add an errorDialog handler to the message path that alerts the user if a connection to the database cannot be made or disappears during a session.

```
on errorDialog pError
  if pError begins with "sqlyoga_connection_err" then
    answer "The database connection has gone away. Try reconnecting later."
  else
    pass errorDialog
  end if
end errorDialog
```

If an error occurs while trying to execute a SQL query then an error prefixed with **sqlyoga\_executesql\_err**, will be thrown. Invalid queries represent a programming mistake that needs to be addressed.

### Invalid Values

SQL Yoga will throw errors if you do one of the following:

- 1) Pass in an invalid object reference. For example, passing in the name of a table that does not exist to `sqlquery_createObject`.
- 2) Try to set an object property to an invalid value.
- 3) Try to set a non-existent property of a SQL Yoga object.

## Other Errors

Other types of errors are returned in **the result**. For example, if you try to disconnect from a Database object and RevDB encounters an error then **the result** would contain the error message.

dbconn\_disconnect

put the result into theError

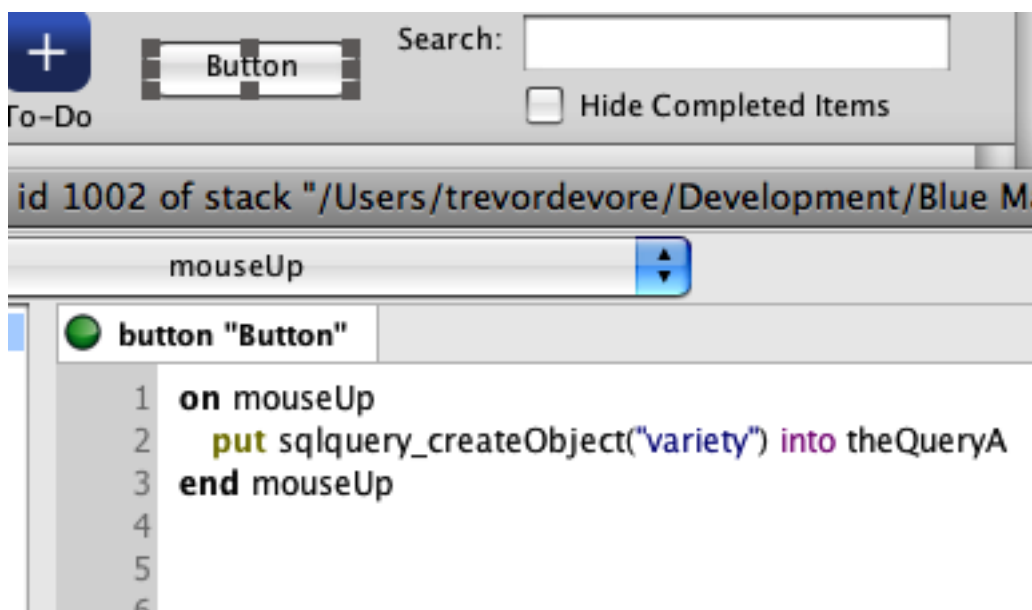
**theError** now contains an error message if an error occurred.

## Errors Thrown From Password Protected Stacks

As of Revolution 4.0 the Script Editor has issues when trying to report an error that was thrown from a password protected stack. Since SQL Yoga is distributed as a password protected stack this information is relevant to developers using SQL Yoga.

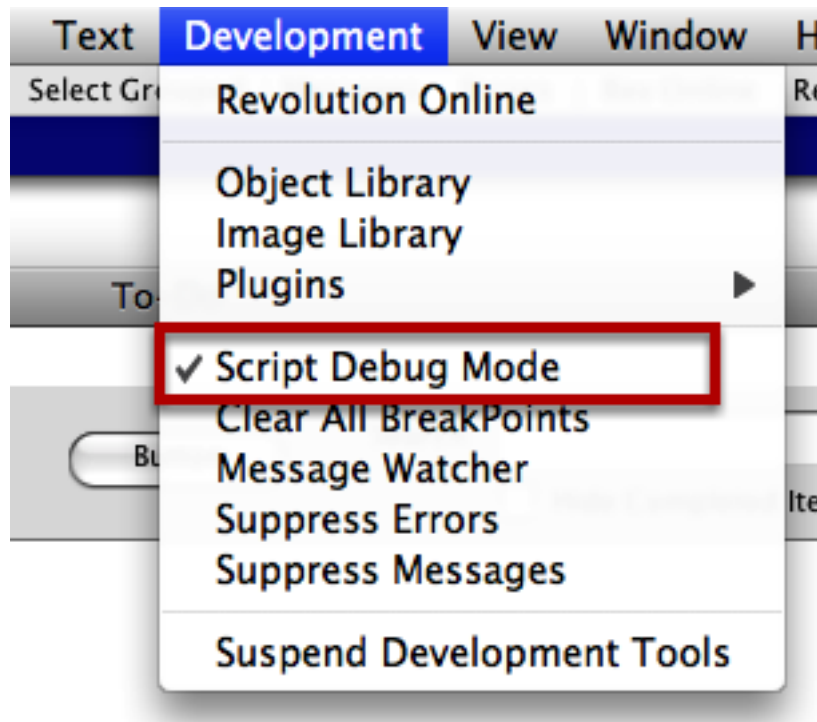
This lesson demonstrates the issue and discusses a workaround.

### The Issue

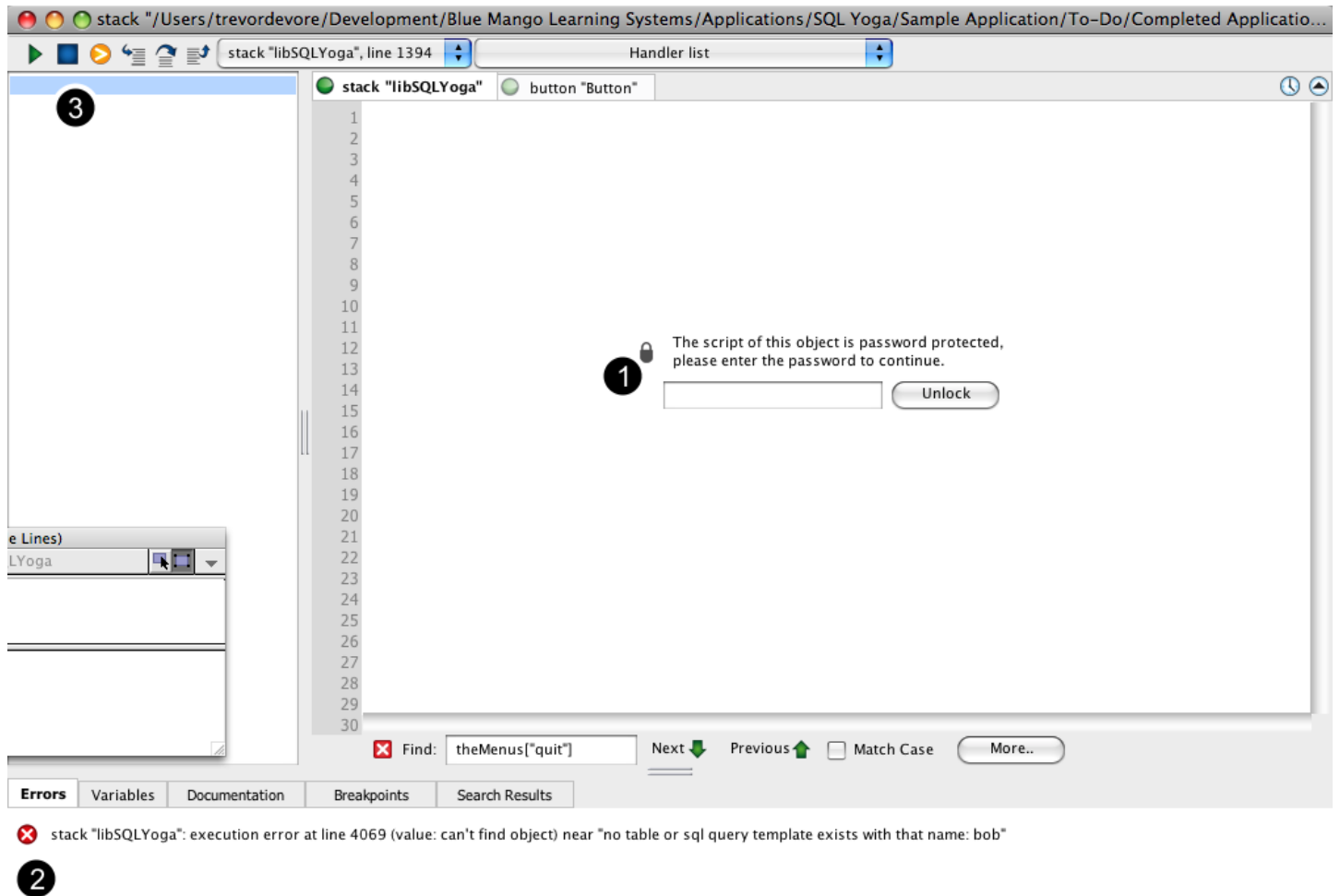


Assume you have some code that calls the SQL Yoga command **sqlquery\_createObject**. The code passes an invalid table *variety*. This causes SQL Yoga to throw an error. This can cause problems for the Revolution script editor.

## When Script Debug Mode Is On



If **Script Debug Mode** is on...

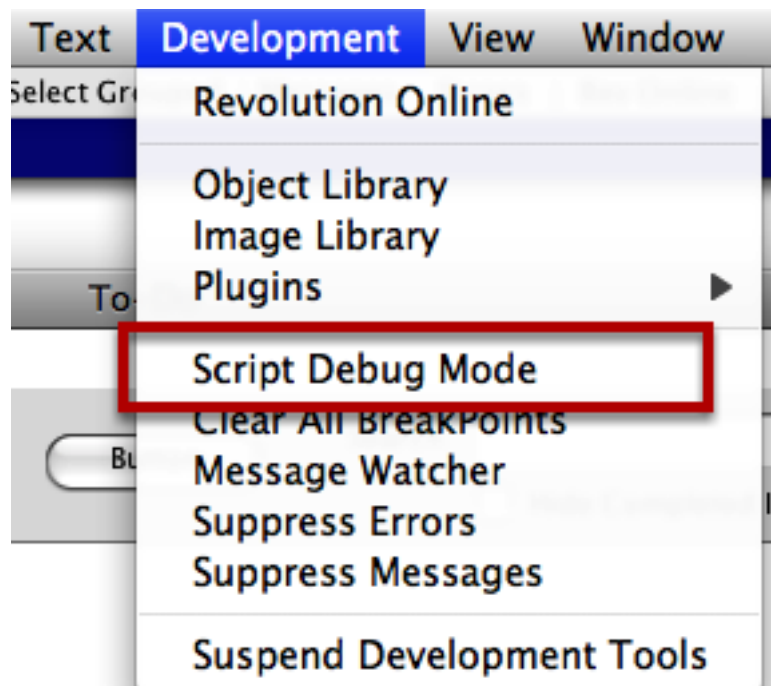


the Revolution script editor appears, asking you for the password of the SQL Yoga stack (1). If Revolution recognizes the error then it is reported in the Errors pane (2). This error is what you should be concerned with. You cannot unlock the SQL Yoga stack script.

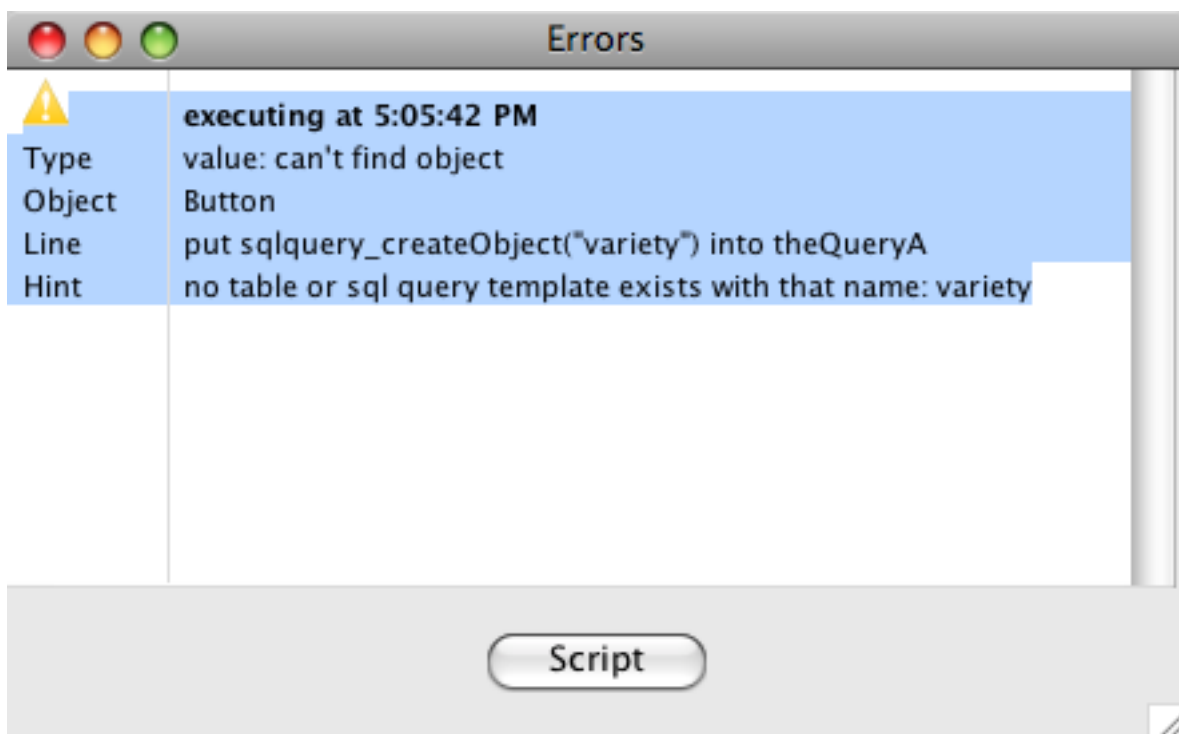
You should click the **Stop** button (3) in the debugger controls.

Note that if Revolution does not recognize the error, i.e. the error starts with *sqllyoga\_connection\_err*, then Revolution shows nothing in the Errors pane and you have no way of knowing the error message.

## When Script Debug Mode is Off



If **Script Debug Mode** is off...

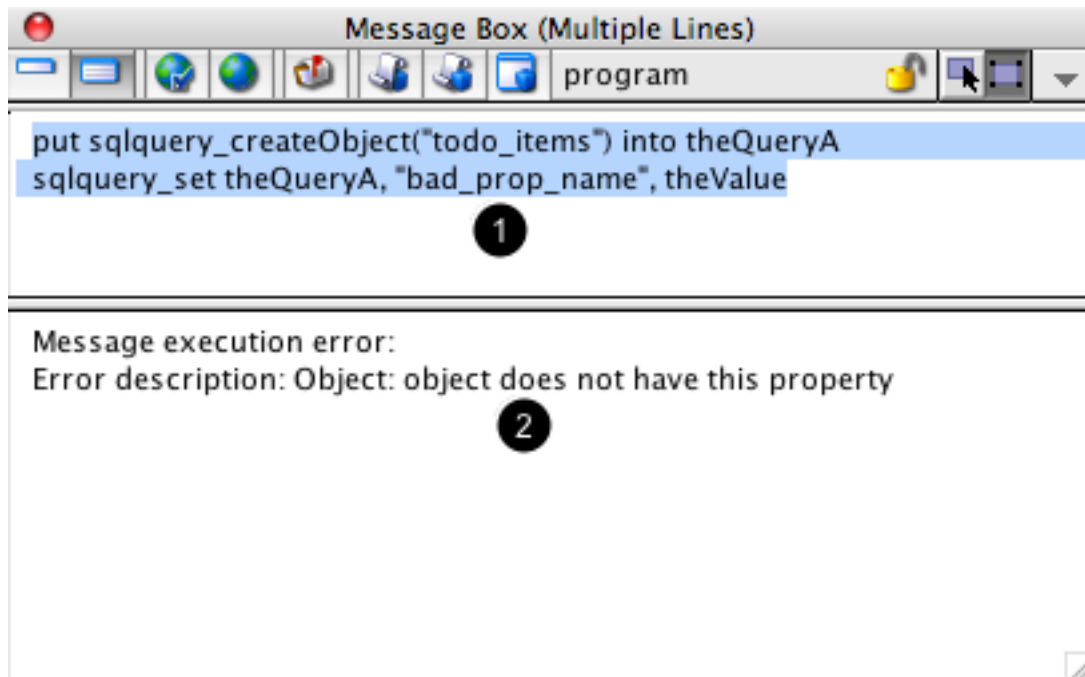


The error will be displayed under all circumstances. If you have an error that Revolution isn't reporting properly in the Script Editor window then try turning off Script Debug Mode.

## Revolution Message Box

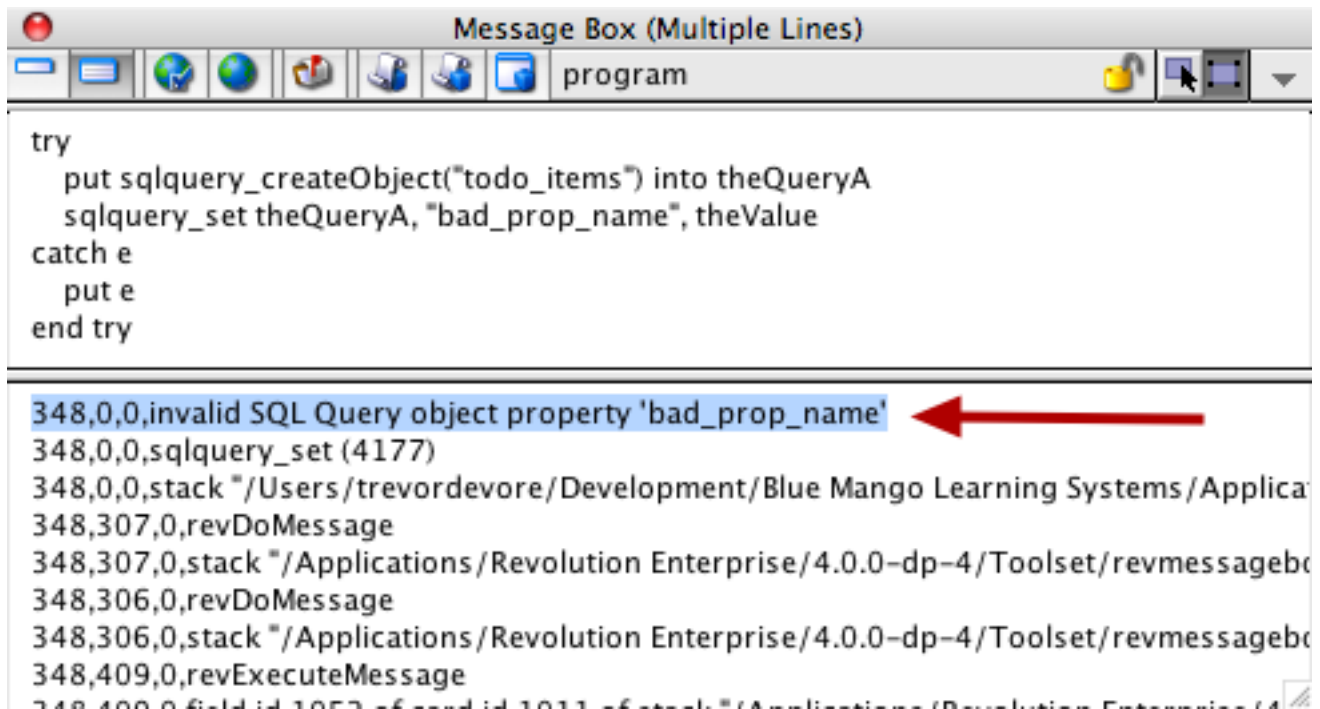
As of Revolution 4.0 the Message Box does not print out hints when reporting errors. You should be aware of this when executing RevTalk that makes calls to SQL Yoga.

### Default Message Box Behavior



In this example I am passing an invalid property name to `sqlquery_set` (1). The Message Box merely reports that *object does not have this property* (2).

## Printing Additional Information



```
try
  put sqlquery_createObject("todo_items") into theQueryA
  sqlquery_set theQueryA, "bad_prop_name", theValue
catch e
  put e
end try
```

```
348,0,0,invalid SQL Query object property 'bad_prop_name'
348,0,0,sqlquery_set (4177)
348,0,0,stack "/Users/trevordevore/Development/Blue Mango Learning Systems/Applica
348,307,0,revDoMessage
348,307,0,stack "/Applications/Revolution Enterprise/4.0.0-dp-4/Toolset/revmessagebo
348,306,0,revDoMessage
348,306,0,stack "/Applications/Revolution Enterprise/4.0.0-dp-4/Toolset/revmessagebo
348,409,0,revExecuteMessage
348,409,0,stack "/Applications/Revolution Enterprise/4.0.0-dp-4/Toolset/revmessagebo
```

One workaround for this behavior is to wrap your code in a try/catch block. This will print out the entire error, including the hint. Notice how I can now see the hint *invalid SQL Query object property 'bad\_prop\_name'*.

This makes troubleshooting easier when working with the Message Box.

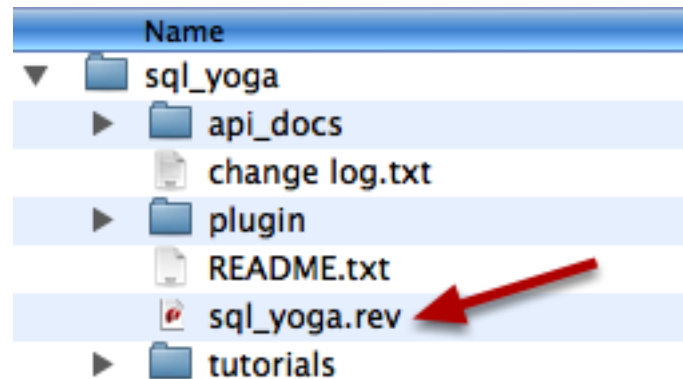


# General Concepts

## How To Use the sql\_yoga.rev Stack File In the IDE

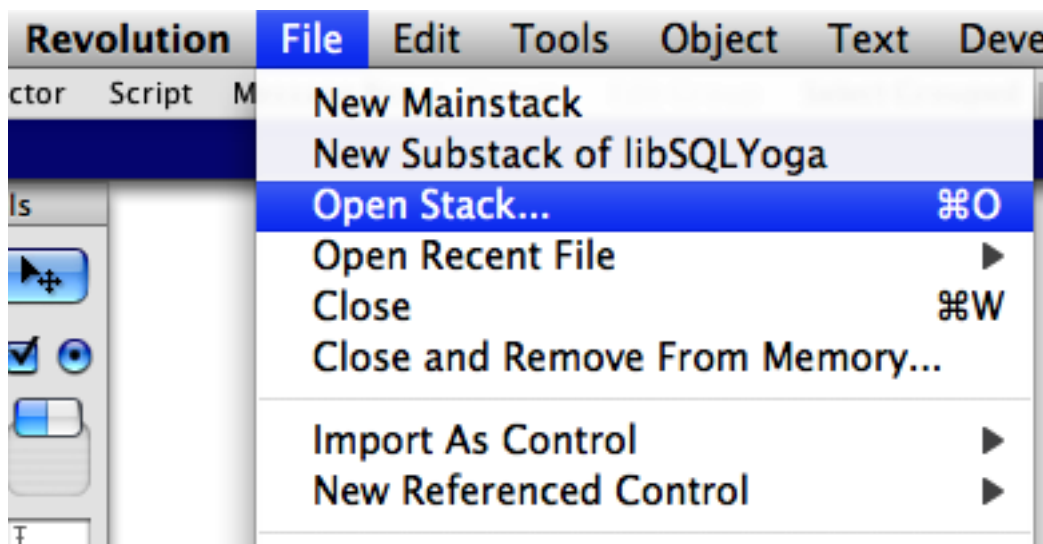
This lesson will discuss how to use the SQL Yoga stack file (sql\_yoga.rev) with Revolution.

### The sql\_yoga.rev Stack File

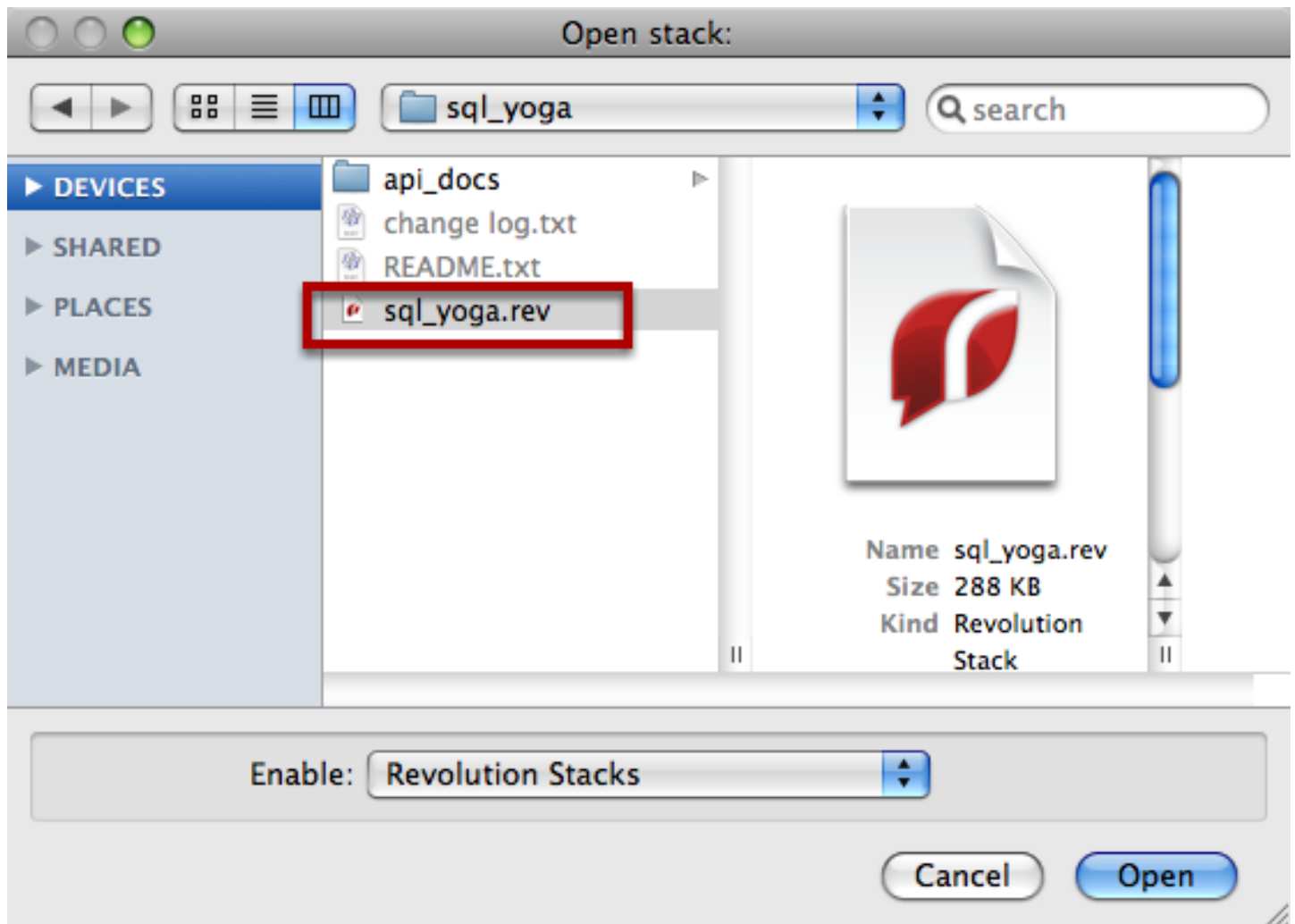


When you download the SQL Yoga distribution from the website you will have a folder resembling this one. **sql\_yoga.rev** is the file that you use for development. This stack file contains all of the SQL Yoga library code that you will use in your projects.

### Open sql\_yoga.rev

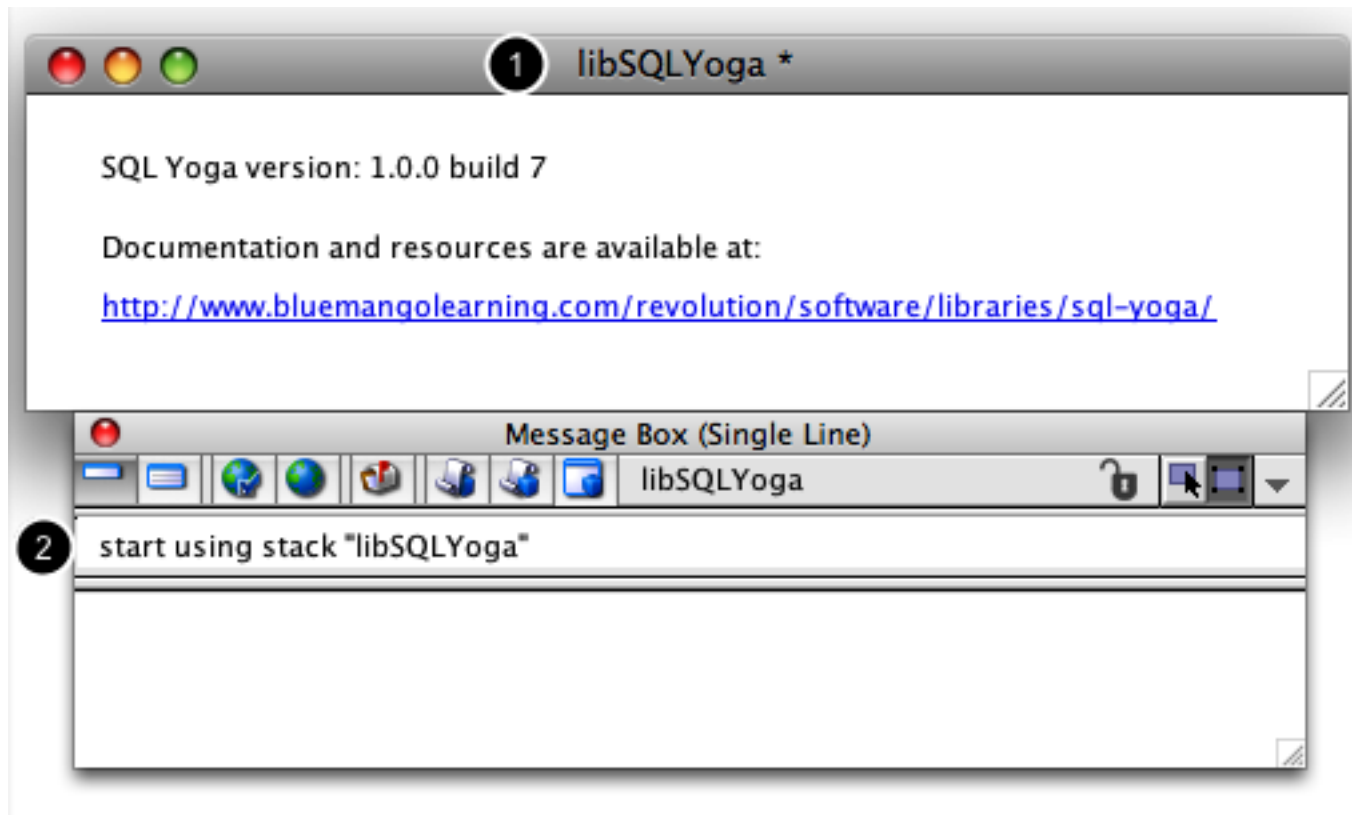


Choose the **File > Open Stack...** menu item.



Select **sql\_yoga.rev**.

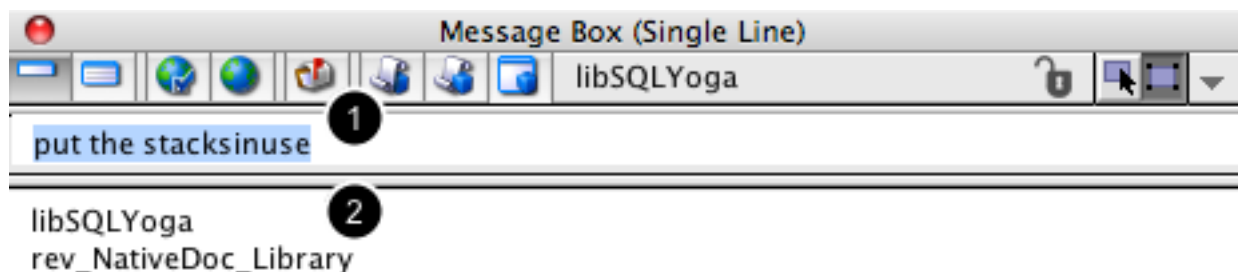
## The libSQLYoga Stack



The sql\_yoga.rev stack file has one mainstack named **libSQLYoga** (1). The script of this stack contains all of the SQL Yoga library handlers.

libSQLYoga is intended to be used as a library stack. To use libSQLYoga as a library and put the stack script into the message path, issue the **start using** command (2) in the Message Box.

## Verify Result



To verify that **libSQLYoga** is now a library stack and is available to all scripts, execute the command **put the stacksInUse** in the Message Box (1). libSQLYoga should appear among the lines of the results that are displayed (2).

You are now ready to start calling the SQL Yoga library handlers. You can close the libSQLYoga stack window if you would like. The window does not need to be open while working with SQL Yoga.

## Unlocking The SQL Yoga Library

---

When you load the **libSQLYoga** library stack it defaults to demo mode. In demo mode a maximum of 10 records will be returned from the database and a dialog will be displayed every 10 minutes.

To unlock SQL Yoga for the current session call **sqlyoga\_register**:

```
sqlyoga_register "myemail@myemail.com", \  
    "MY_REGISTRATION_KEY"
```

put the result into theError

If you have a registration key then you can add this RevTalk code to your application code. For example, you might execute this code at the beginning of a preOpenCard handler before you make any other SQL Yoga library calls.

## How SQL Yoga Represents Objects

---

As of Revolution 4.0 the Revolution engine has no means of creating custom objects. Rather a developer has to rely on one of the existing objects such as stacks, cards, buttons, fields, groups, etc.

While relying on existing Revolution objects works okay for custom GUI objects it is not an appropriate solution for faceless objects (no UI required) such as those used with SQL Yoga.

### Representing Objects with Arrays

SQL Yoga addresses the issue of not being to create objects by using Revolution arrays to represent objects. Arrays in Revolution are easy to create and work with and can be multi-dimensional. While not perfect, arrays provide enough flexibility to be an adequate stand-in for real objects.

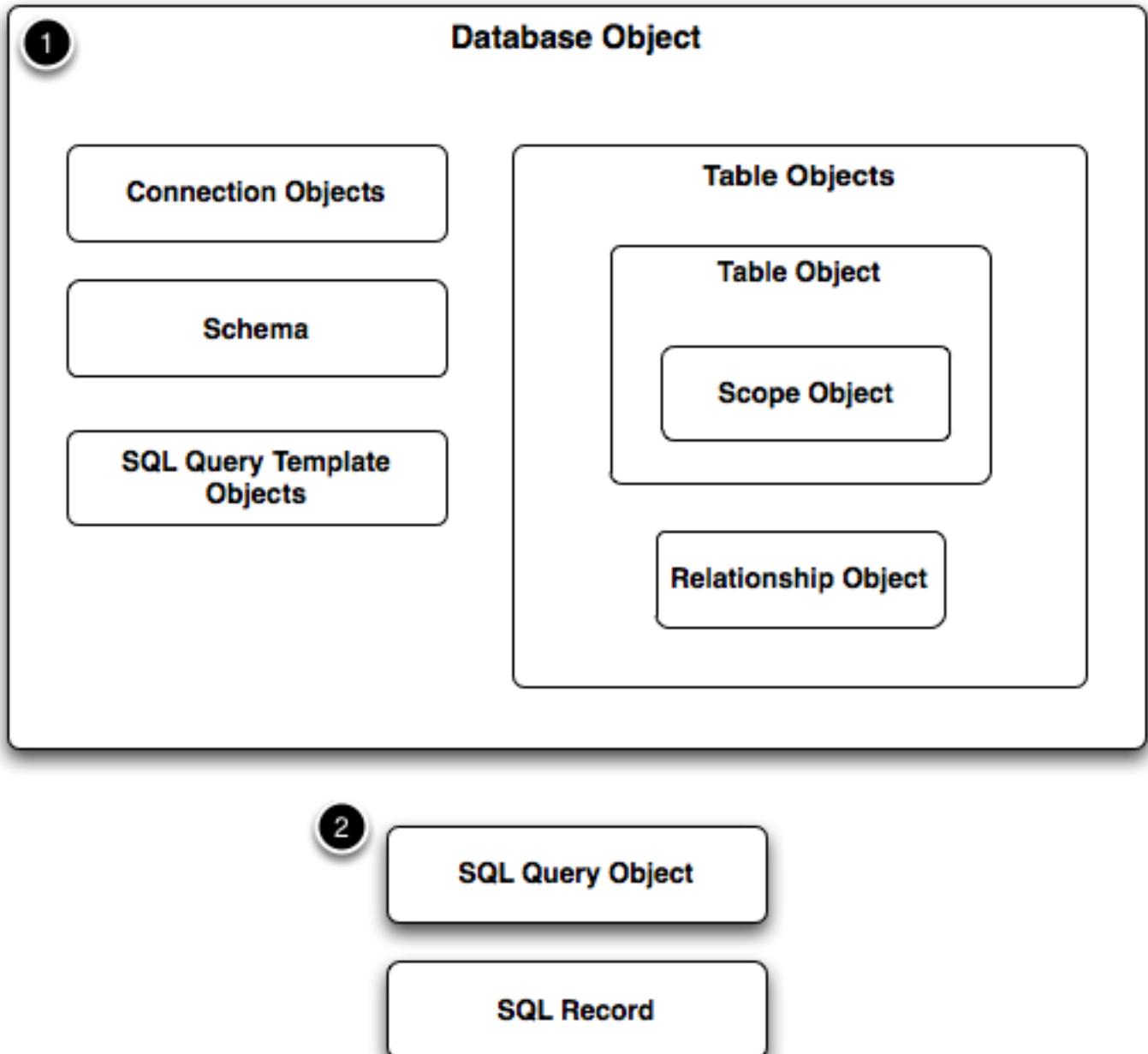
If you inspect some of the object arrays that SQL Yoga creates you will notice that some keys are prefixed with the @ symbol. The @ symbol designates a special property that the SQL Yoga library uses in order to store certain information about the object.

# Database Objects



## Introduction to the Database Object and Database Schema

### The Database Object



The Database object is the base SQL Yoga object. Each Database object represents a one database schema but can connect to numerous databases that have that schema. If you have two different database schemas you will need two Database objects.

All other SQL Yoga objects are:

1) Stored within the Database object, or

2) Derived from information stored in the Database object.

Any objects that are stored within the Database object are saved when calling `dbobject_save`. Objects not stored within the Database object, such as a SQL Query object, will not be.

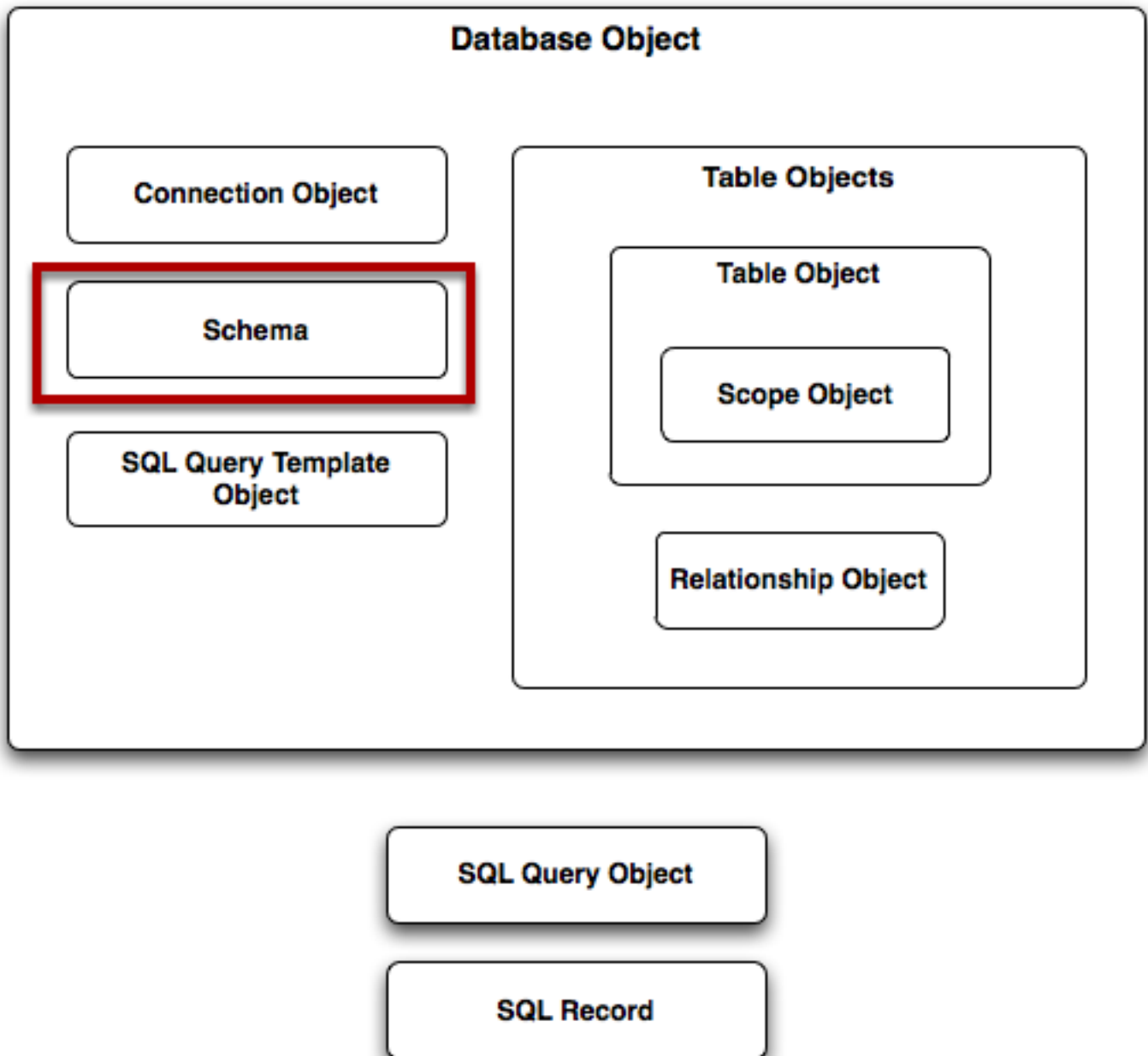
### How A Database Object is Represented in Memory and Storing Them

Like all SQL Yoga objects, the Database object is stored as an array. Each Database object is stored in memory as part of a script local variable in the SQL Yoga library.

In order to permanently save the Database object array SQL Yoga allows you to specify a file or Revolution button where you can store the array. For more information please see the **storage file** and **storage object** properties in the [dbobject\\_set API documentation](#).

When you need to load a Database object into memory from a file or button you can use the [dbobject\\_createFromFile](#) or [dbobject\\_createFromObject](#) commands.

## The Database Schema



A Database object stores the schema of the database in an internal Schema object. The information about the database tables and fields stored in the Schema object is what enables many of SQL Yoga's features.

The Schema object is created the first time you connect to a database. If the database schema changes at any time you will need to tell SQL Yoga to refresh the Schema object. This can be done by calling the command `dbobject_reloadSchema`.

## Creating A Database Object

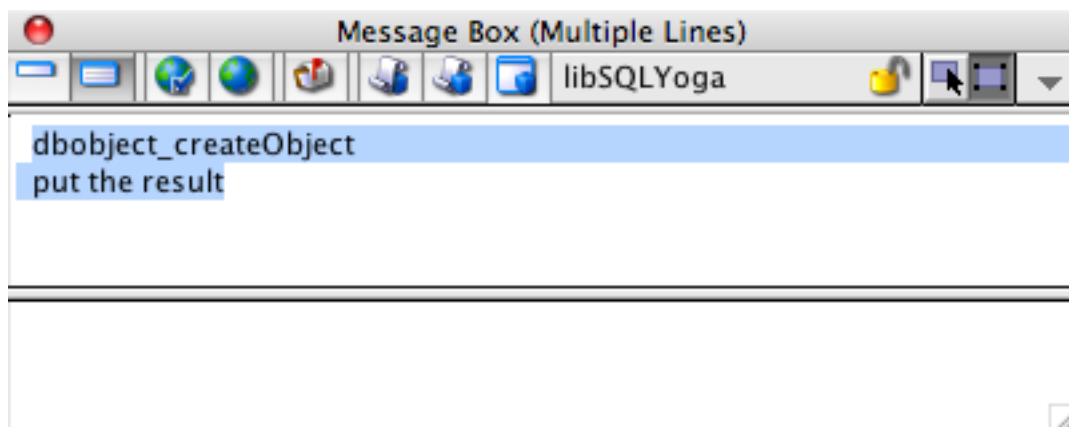
A Database object is where SQL Yoga stores all of the information about your database that is used to automate the creation of SQL queries for you.

Let's look at how to create a Database object.

**Note:** The SQL Yoga distribution includes a Revolution IDE plugin that is helpful during development. To learn more about the plugin please visit the [plugin manual](#).

**Note:** The examples used in these tutorials will use the default values for many of the parameters passed into the SQL Yoga handlers. In cases where you are only using one Database object (most cases) there is no need to pass parameters to a command like `dbobject_createObject`. If you plan on using multiple Database objects make sure you read the [API documentation](#) so you understand what the parameters are.

### 1) Create a Database Object



You can use **`dbobject_createObject`** to create a new database object in memory.

This code only needs to be run once so you can execute it in the message box if you would like.

#### ***Copy & Paste***

```
dbobject_createObject  
put the result
```

Now a new Database object exists in memory. The name of the Database object is **default**.

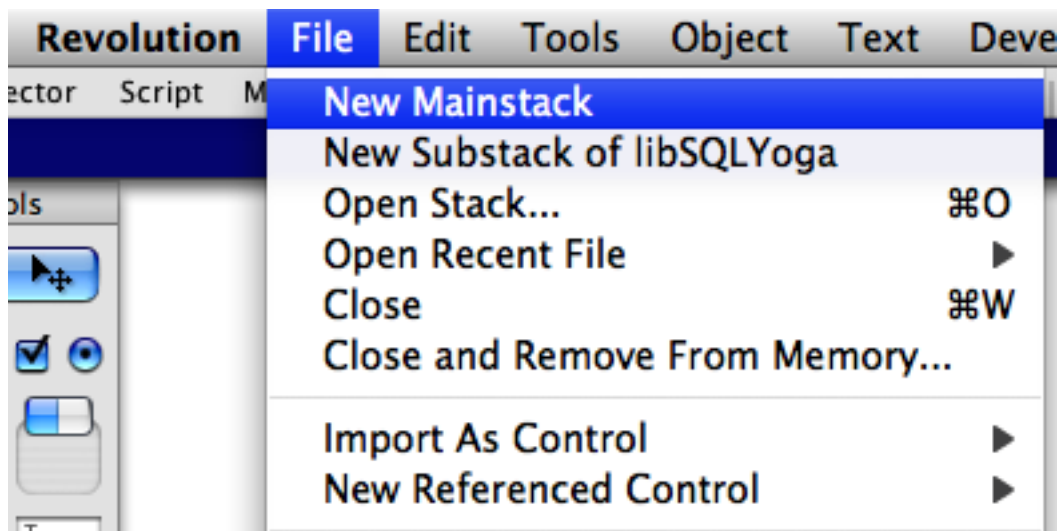
**Note:** SQL Yoga will throw an error if a Database object named **default** already exists.

## Saving A Database Object Across Sessions

When you create a Database object it only exists in memory. If you were to quit Revolution the Database object would disappear forever. SQL Yoga allows you to store a Database object across sessions using a Revolution control, such as a button, or in a file.

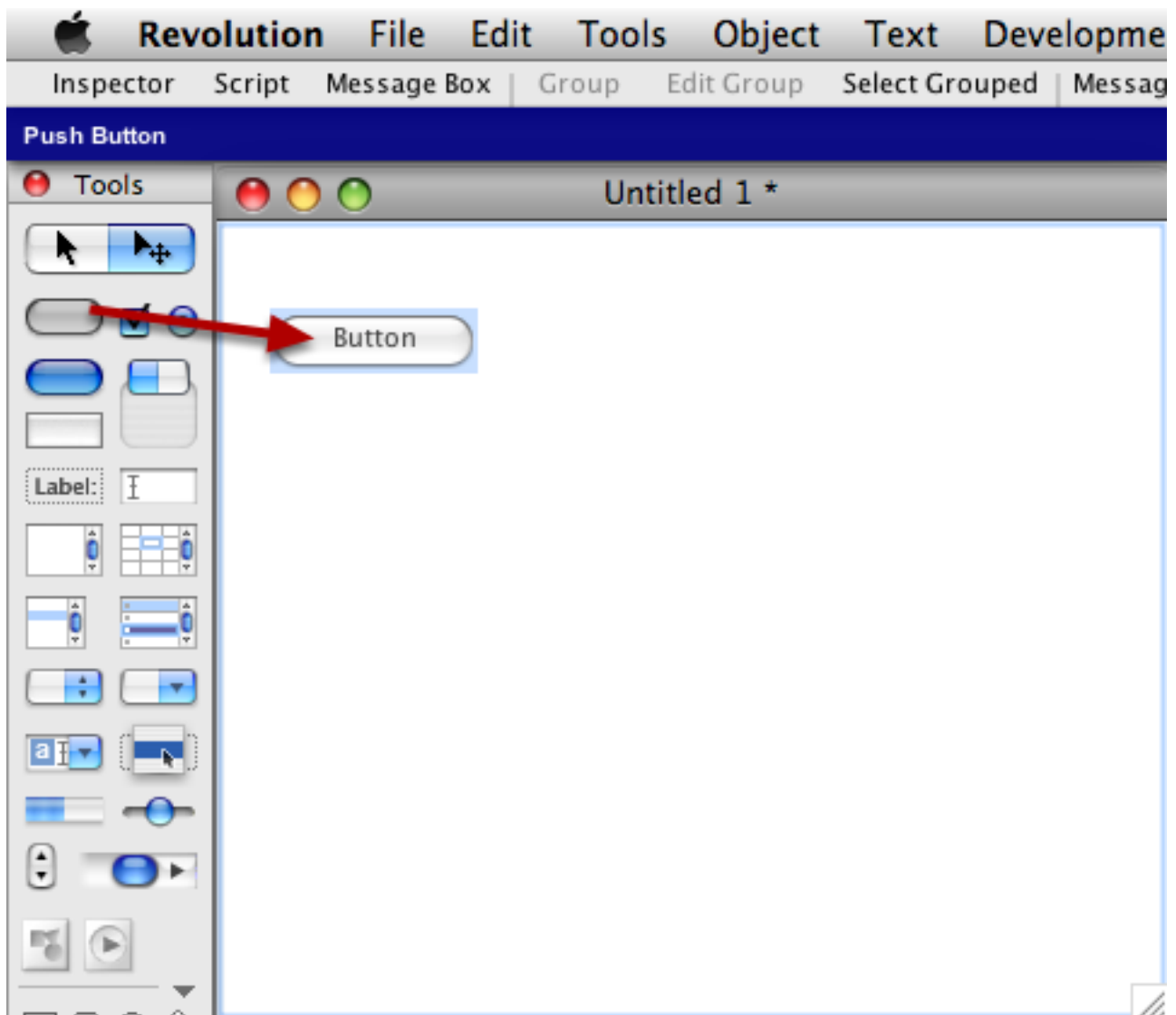
This lesson will show you how to configure SQL Yoga to store the Database object in a Revolution button.

### Create a Stack



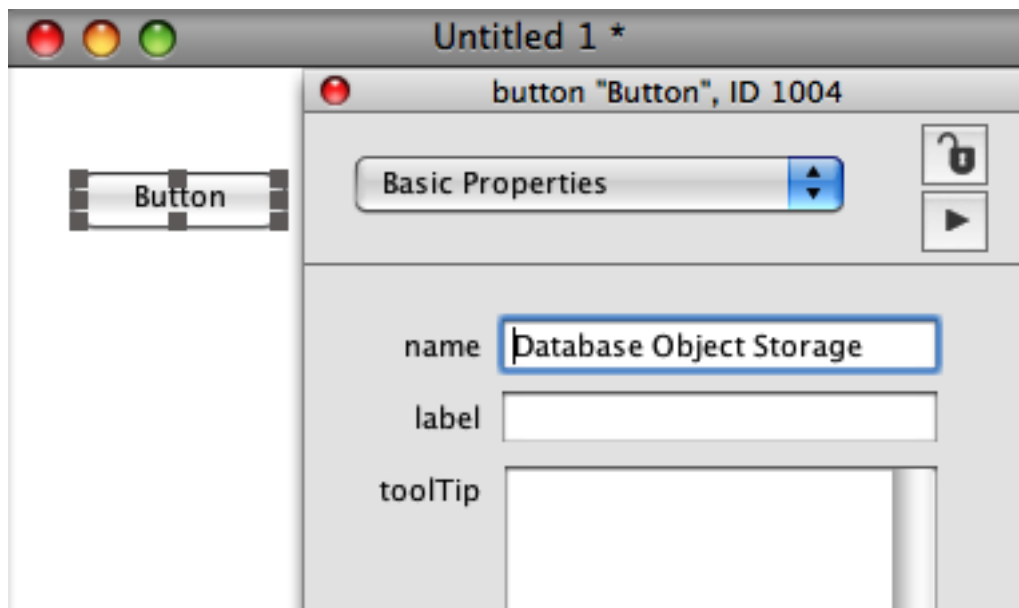
We need to create a button to store the Database object in. In order to do that we need to create a stack. Choose the **File > New Mainstack** menu item to create a new stack.

## Add A Button



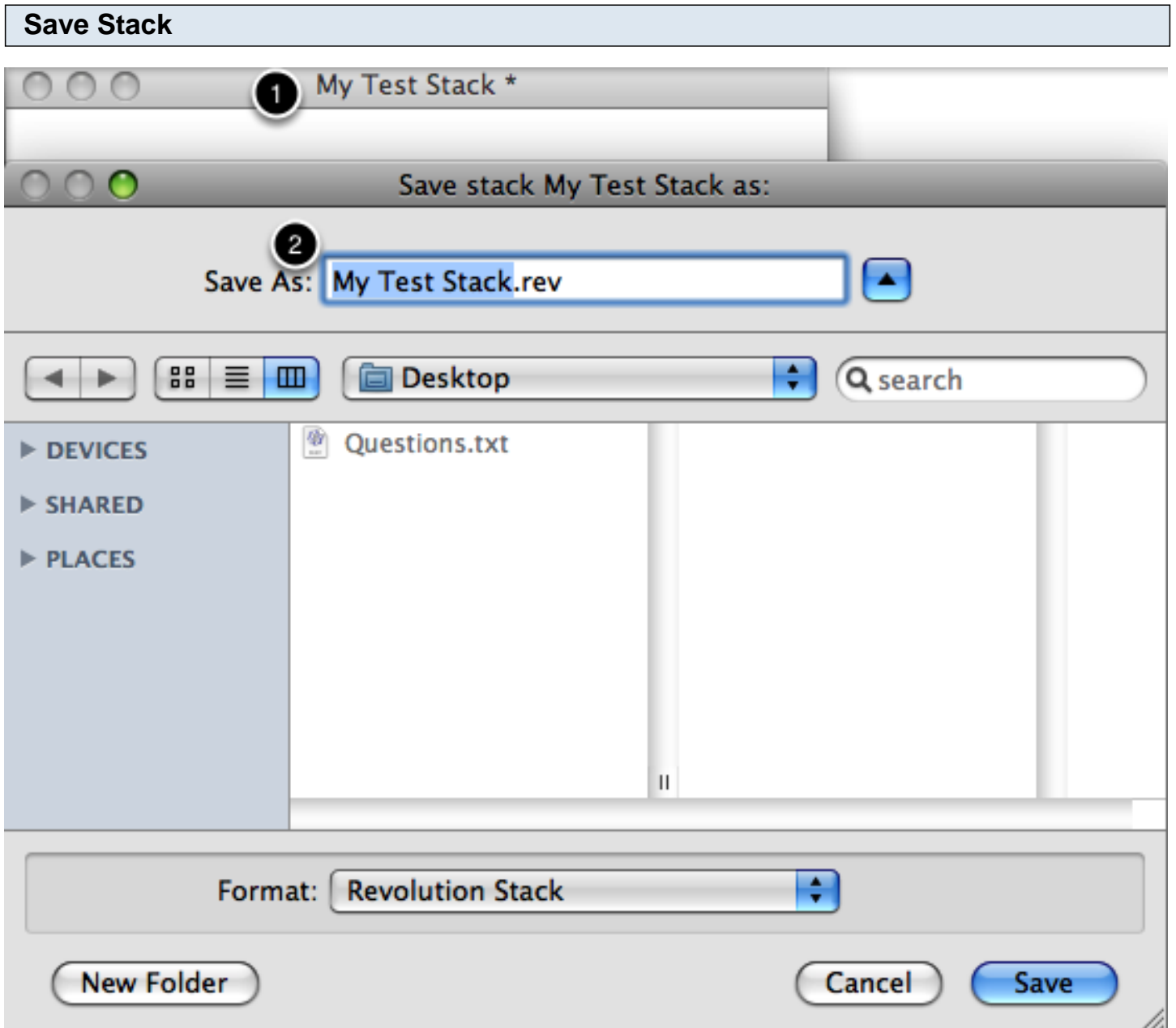
Drag and drop a button from the Tools Palette onto the new stack.

## Name the Button



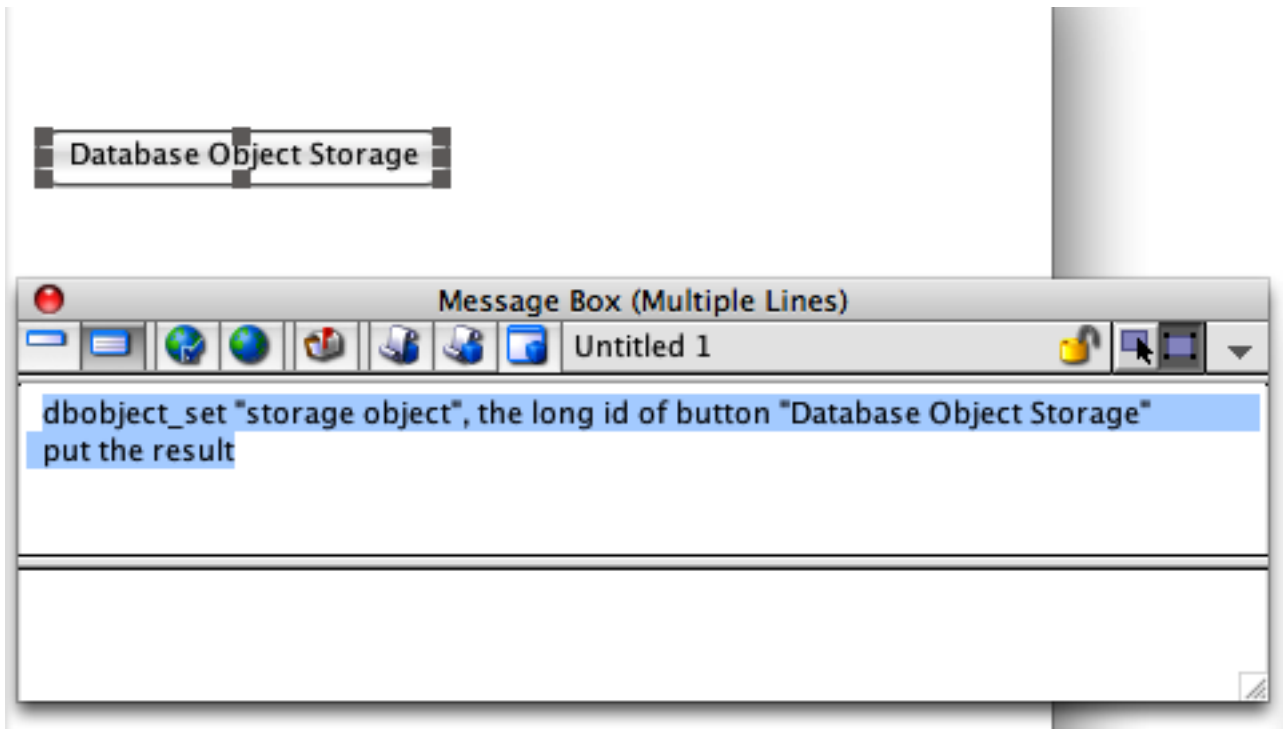
In the Object Inspector name the button **Database Object Storage**.





Before going on you need to save your stack to disk. I named my stack **My Test Stack** (1) using the Object Inspector and I'm saving my stack as **My Test Stack.rev** on disk (2).

## Tell SQL Yoga Where To Save The Database Object



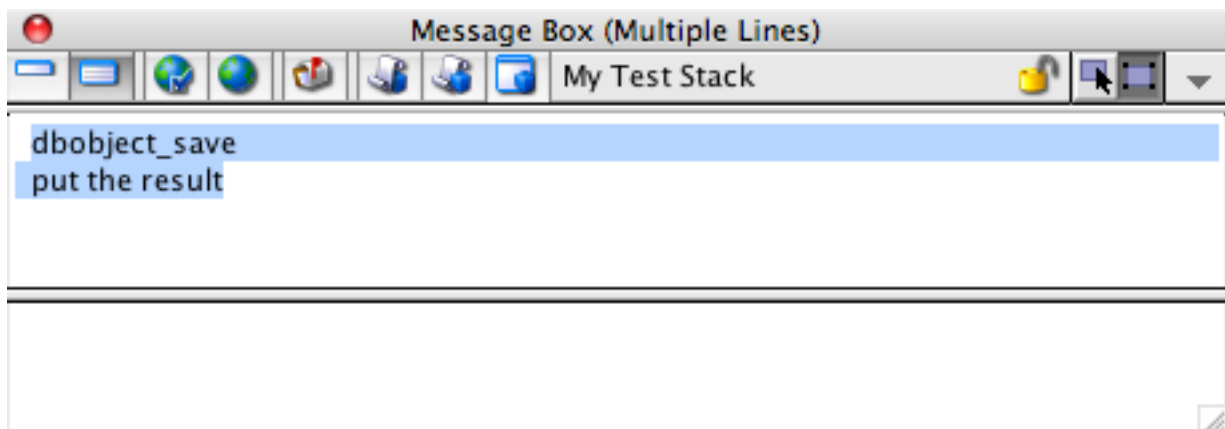
Now that you have a button to store the Database Object in you can set the **storage object** property of the Database object.

This code only needs to be run once so you can execute it in the message box if you would like.

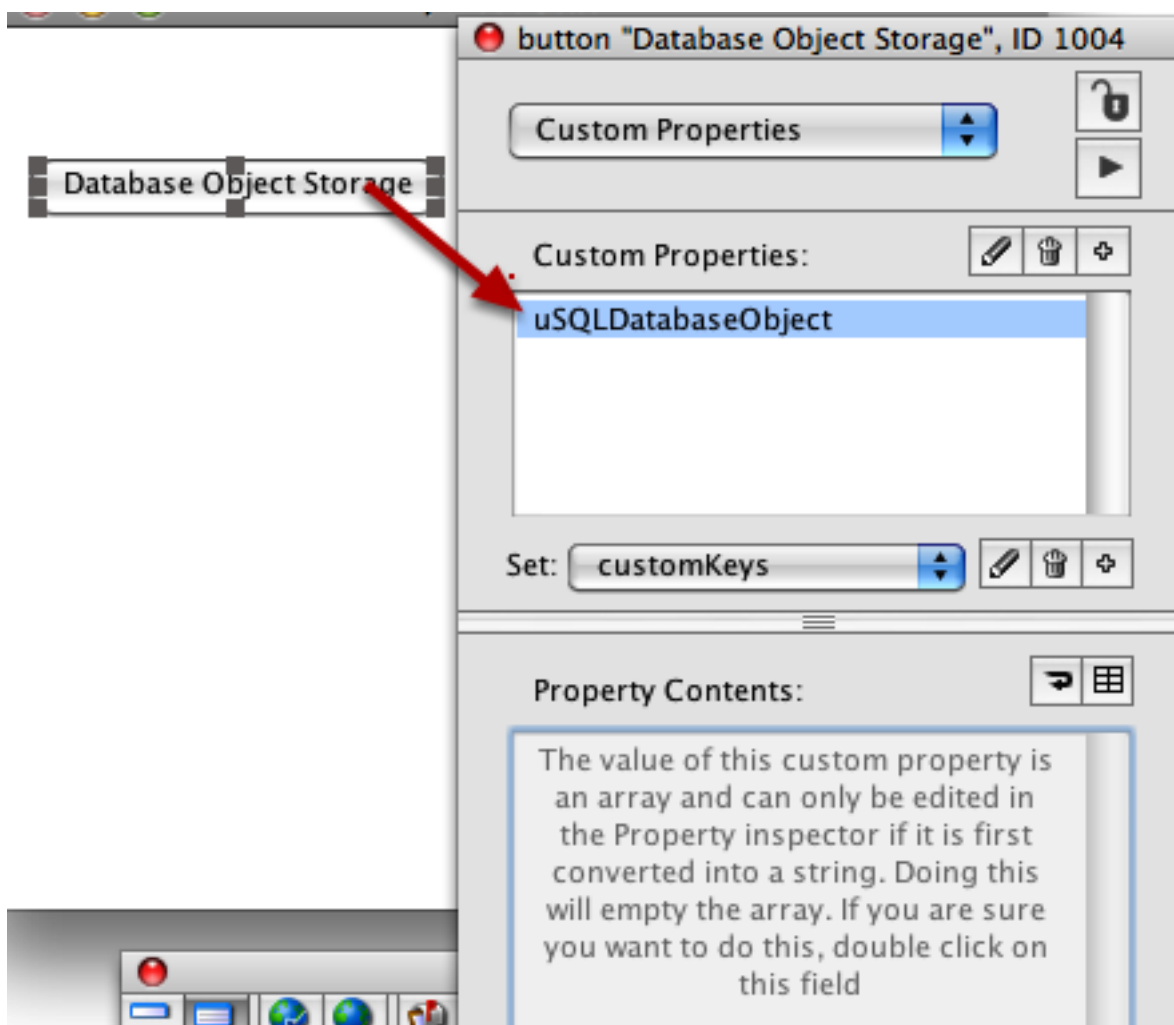
### **Copy & Paste**

```
dbobject_set "storage object", the long id of button "Database Object Storage"  
put the result
```

## Save The Database Object To The Button



Whenever you want to save the current state of the Database object to the **Database Object Storage** button you call **dbobject\_save**. This will store the Database object in the **uSQLDatabaseObject** custom property of the button.



**IMPORTANT!!!**

**dbobject\_save** only stores the Database object in the custom property of the button. You **MUST** save the stack that the button is on in order to permanently save the information.

## Loading The Database Object When Your Stack Opens

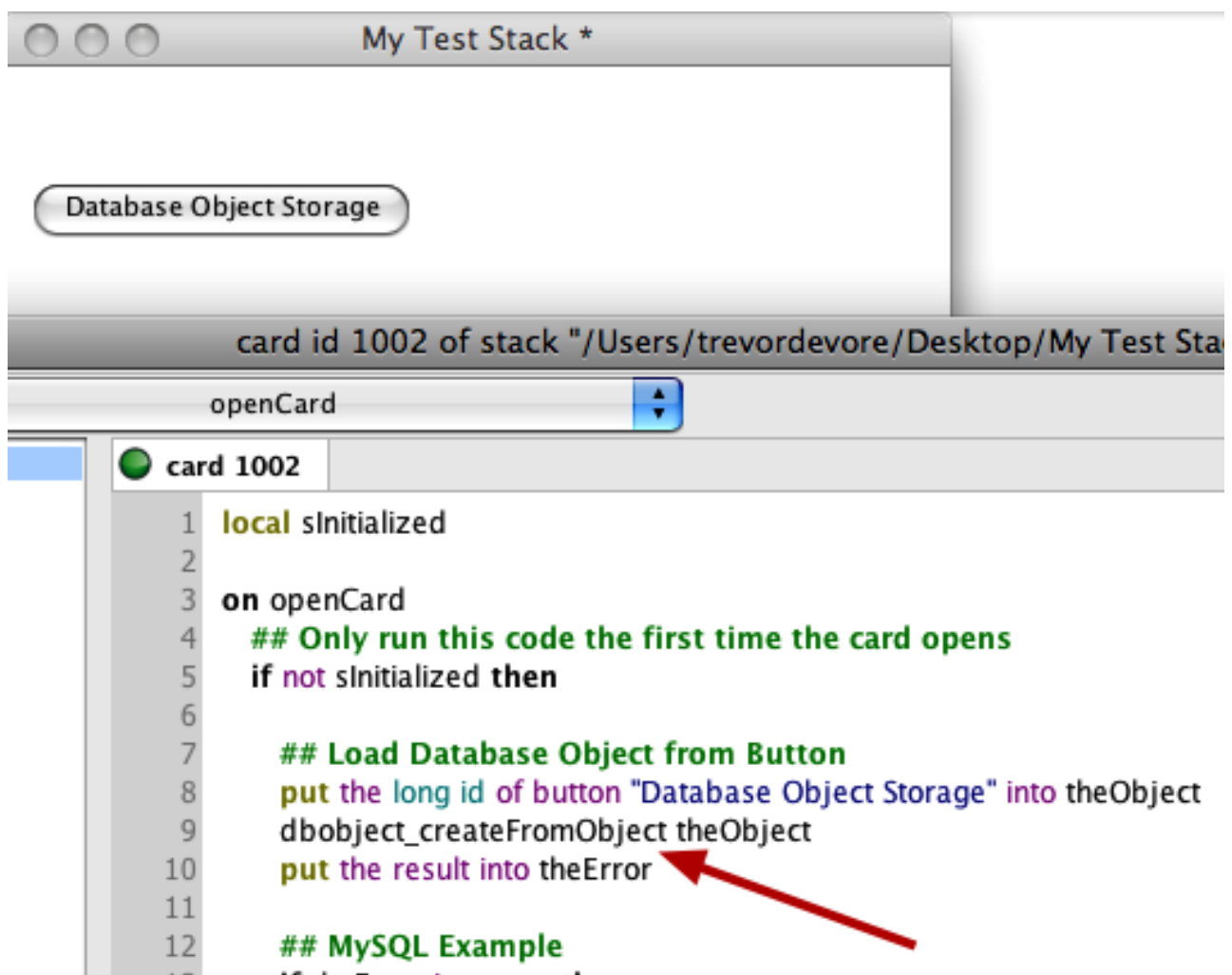
If you are storing your Database object in a Revolution button across sessions then you will need to tell SQL Yoga to load the Database object each time your project is opened.

This lesson will show you how to load the Database object from the button that you have saved it in.

### dbobject\_createFromObject

SQL Yoga provides a command that will create a Database object from the control that you have saved it in. The command is called **dbobject\_createFromObject** and you pass a reference to the control that your Database object is stored in.

### Example Script



In this example the **dbobject\_createFromObject** command is called when a card opens and before any SQL Yoga calls are made that interact with the Database object.

```

1  local sInitialized
2
3  on openCard
4    ## Only run this code the first time the card opens
5    if not sInitialized then ← 1
6
7    ## Load Database Object from Button
8    put the long id of button "Database Object Storage" into theObject
9    dbobject_createFromObject theObject
10   put the result into theError
11
12   ## MySQL Example
13   if theError is empty then ← 2
14     dbconn_set "host", "localhost"
15     dbconn_set "username", "root"
16     dbconn_set "password", empty
17     dbconn_set "database name", "sql_yoga_test"
18
19     try
20       ## If a connection fails then an error is thrown
21       ## Later on this will help you handle this error
22       ## globally in your application by adding your
23       ## own errorDialog handler. For now just wrap
24       ## the dbconn_connect call in try/catch.
25       dbconn_connect
26     catch theError
27       answer "An error occurred while connecting to the database:" && theError
28     end try
29   end if
30
31   if theError is empty then
32     ## Set flag so code doesn't run again
33     put true into sInitialized
34   end if
35
36   end if ## end of initialization code
37 end openCard
38

```

Here is an example of what a finished **openCard** script might look like. Some things to note:

- 1) A variable is being stored that ensures the code only runs the first time the card is open.

2) Error checking has been added.

=====

***Copy & Paste (remember to customize connection settings)***

=====

**local** sInitialized

**on** openCard

**## Only run this code the first time the card opens**

**if not** sInitialized **then**

**## Load Database Object from Button**

**put** the long id of button "Database Object Storage" into theObject

dbobject\_createFromObject theObject

**put** the result into theError

**## MySQL Example**

**if** theError is empty **then**

dbconn\_set "host", "localhost"

dbconn\_set "username", "root"

dbconn\_set "password", empty

dbconn\_set "database name", "sql\_yoga\_test"

**try**

**## If a connection fails then an error is thrown**

**## Later on this will help you handle this error**

**## globally in your application by adding your**

**## own errorHandler handler. For now just wrap**

**## the dbconn\_connect call in try/catch.**

dbconn\_connect

**catch** theError

**answer** "An error occurred while connecting to the database:" && theError

**end try**

**end if**

**if** theError is empty **then**

**## Set flag so code doesn't run again**

**put true** into sInitialized

**end if**

```
end if ## end of initialization code  
end openCard
```



## How To Work With Multiple Database Objects

---

When working with SQL Yoga you can create any number of Database objects by calling [dbobject\\_createObject](#). This lesson will show you how you can work with multiple Database objects in your code.

In order to make writing SQL Yoga code easier, all API calls in the library assume you are targeting the default Database object. The name of the default Database object is the value returned from [sqlyoga\\_getDefaultDatabase\(\)](#).

For example, if you look at the [dbconn\\_createObject command](#) you will notice a 4th parameter that is optional: *pDBKey*. If this parameter is empty then the Database object name returned by [sqlyoga\\_getDefaultDatabase\(\)](#) will be used.

If you want to work with more than one Database Object you have two options.

1) Pass in the name of the Database object to any API calls that accept the *pDBKey* parameter.

```
dbconn_createObject "my connection", "sqlite",, "my database object name"
```

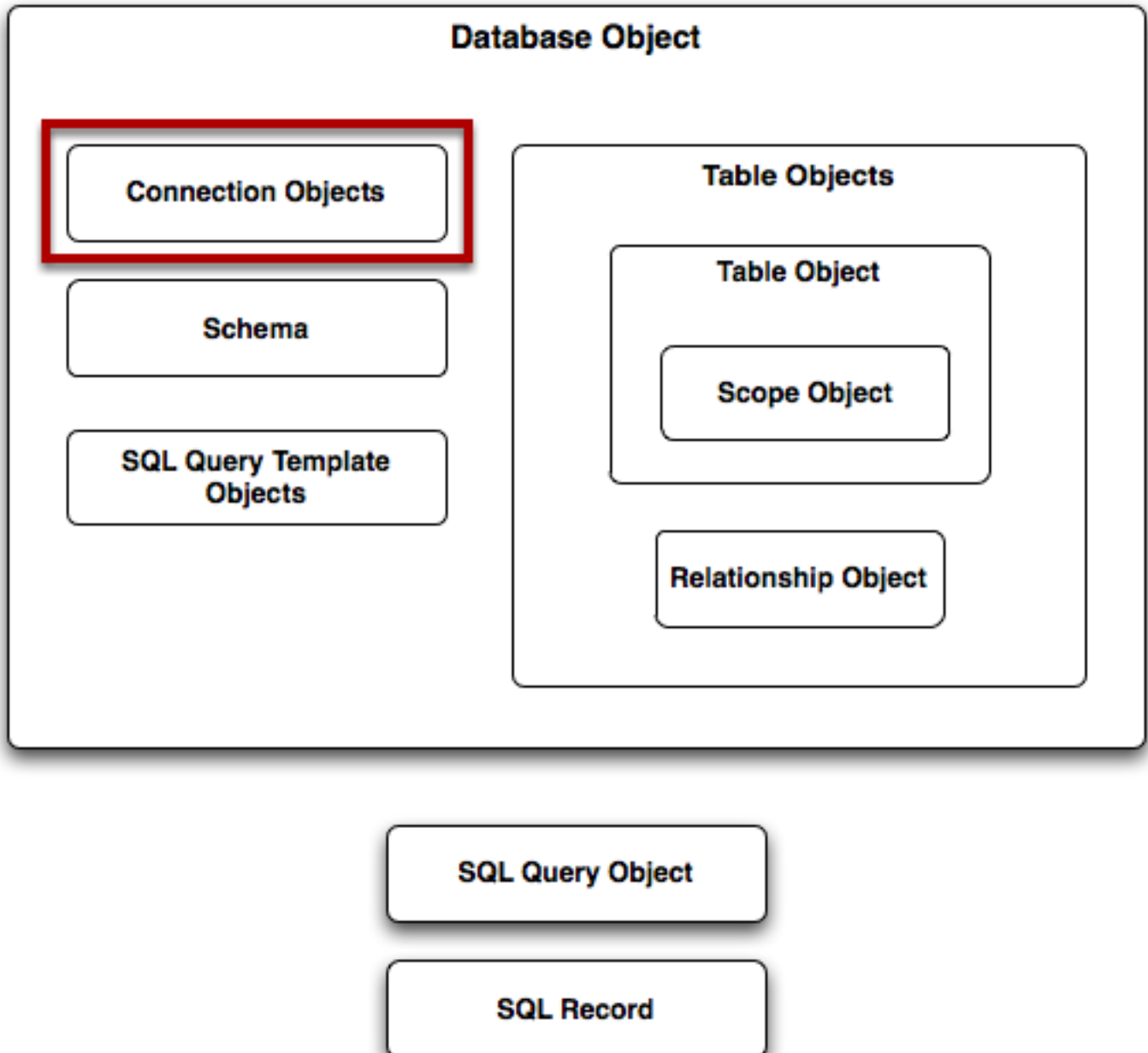
2) Change the name of the default Database object by calling [sqlyoga\\_setDefaultDatabase](#).

```
sqlyoga_setDefaultDatabase "my database object name"
```

# Database Connections

## Introduction to Connection Objects

### The Connection Object



Connection objects are used by SQL Yoga to connect to databases that have the same schema.

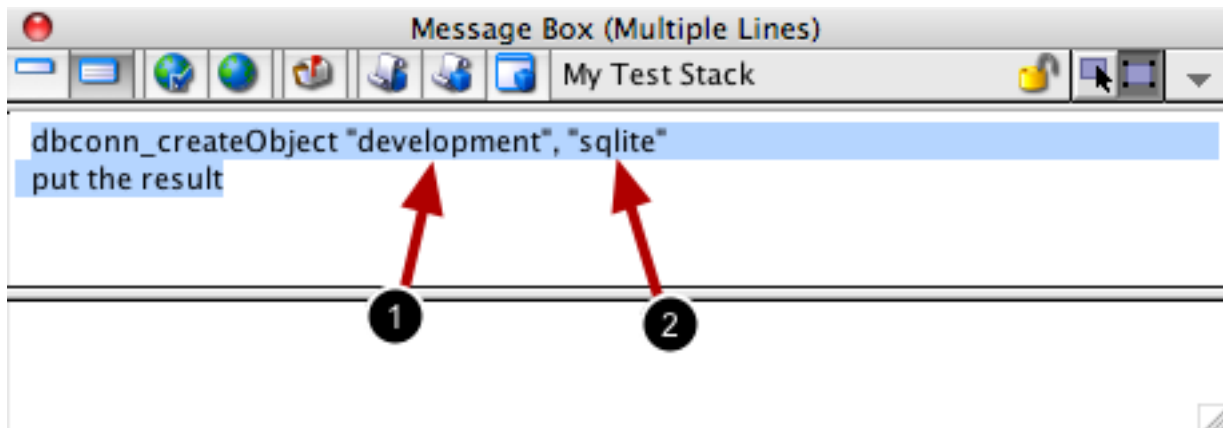
A Database object can have multiple connections. For example, you can define one connection that connects to a database used for testing and another that connects to the live database.

The default connection that SQL Query and SQL Record objects use can be changed at any time by setting the **default connection** property (see [dbobject\\_set](#)) of the Database object.

## Connecting to a Database Using a Connection Object

A Connection object is used by SQL Yoga to connect to a database. You set properties on this object and then call `dbconn_connect` to open a connection. Let's look at how this is done.

### Create a Database Connection Object



To create a Connection object you call `dbconn_createObject`. You pass in a name that you would like to give to the connection as the 1st parameter (1) and the type of database you are connecting to as the 2nd parameter (2).

You only need to create a Connection object once so you can execute it in the message box if you would like.

=====

### Copy & Paste

=====

```
dbconn_createObject "development", "sqlite"  
put the result
```

### Save Database Object

A Connection object is stored inside of a Database object much like a control can be stored inside of a group. Since you have added a new object to the Database object you will need to save it so that the new object is not lost when you quit Revolution. Just call **dbobject\_save** and then save the stack file to disk.

## Define Connection Properties

```

1  on openCard
2    ## MySQL Example
3    dbconn_set "host", "localhost"
4    dbconn_set "username", "root"
5    dbconn_set "password", empty
6    dbconn_set "database name", "sql_yoga_test"
7
8    try
9      ## If a connection fails then an error is thrown
10     ## Later on this will help you handle this error
11     ## globally in your application by adding your
12     ## own errorDialog handler. For now just wrap
13     ## the dbconn_connect call in try/catch.
14     dbconn_connect
15   catch e
16     answer "An error occurred while connecting to the database:" && e
17   end try
18 end openCard
19

```

Now that a Connection object exists we can configure it and then connect to the database. You need to configure most of the properties of a Connection object at runtime. SQL Yoga does not store information such as the username and password inside the object.

Edit the card script of the stack you are working on and add an **openCard** handler.

=====

**Copy & Paste**

=====

```

on openCard
  ## MySQL Example
  dbconn_set "host", "localhost"
  dbconn_set "username", "root"
  dbconn_set "password", empty
  dbconn_set "database name", "sql_yoga_test"

  try

```

```
## If a connection fails then an error is thrown
## Later on this will help you handle this error
## globally in your application by adding your
## own errorDialog handler. For now just wrap
## the dbconn_connect call in try/catch.
```

```
dbconn_connect
```

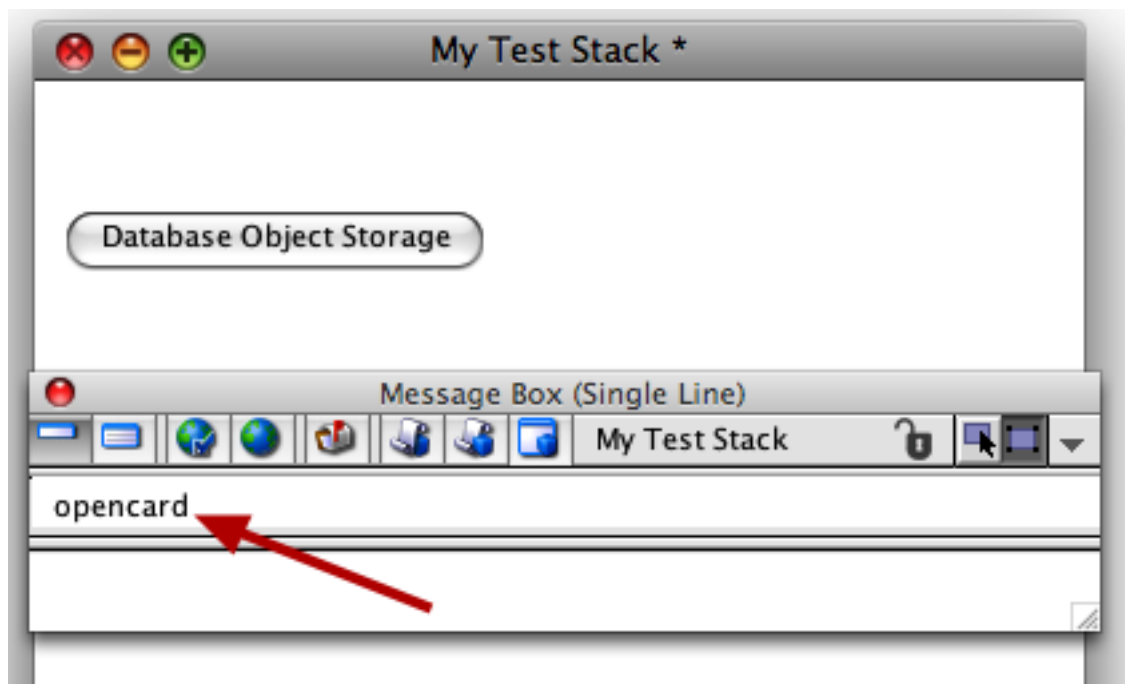
```
catch e
```

```
  answer "An error occurred while connecting to the database:" && e
```

```
end try
```

```
end openCard
```

### Call openCard Handler



After you finish customizing the connection settings in the **openCard** handler you can trigger openCard from the Message Box. This will configure your Connection object and connect to the database.

### Some Examples

Here are some examples of properties you set when connecting to different types of databases.

#### ## SQLite Example

```
dbconn_set "file", theFullPathToTheDatabaseFile
```

#### ## MySQL Example

```
dbconn_set "host", "localhost"  
dbconn_set "username", "root"  
dbconn_set "password", empty  
dbconn_set "database name", "sql_yoga_test"
```

### ## ODBC Example

```
-- Note: When calling dbconn_createObject you pass in 3 parameters  
--       The 2nd is "odbc" and the 3rd is the type of database you are  
--       connection too  
-- dbconn_createObject "development", "odbc", "sql server"  
dbconn_set "dsn", "sqlserver_odbc"  
dbconn_set "username", "odbc_user"  
dbconn_set "password", empty
```

### ## Local Valentina Example

```
dbconn_set "file", theFullPathToTheDatabaseFile  
dbconn_setVendor "valentina", "mac serial number", MyValMacSerialNumber  
dbconn_setVendor "valentina", "win serial number", MyValWinSerialNumber  
dbconn_setVendor "valentina", "unix serial number", MyValUnixSerialNumber  
dbconn_setVendor "valentina", "encryption key", MyEncryptionKey
```

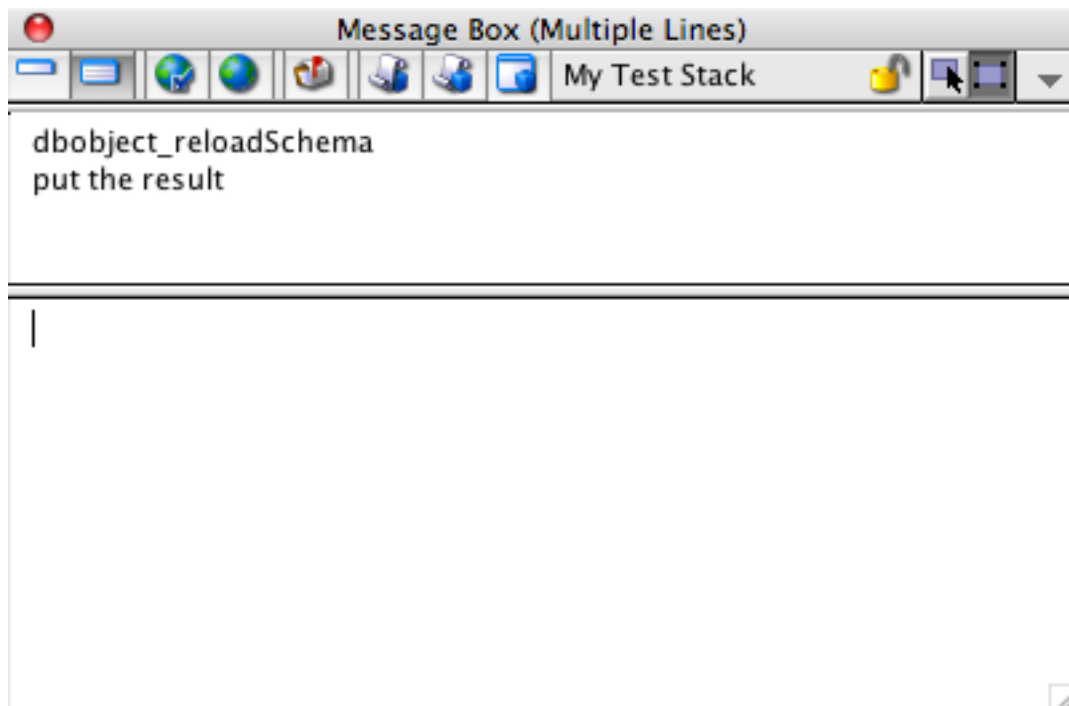
### ## Valentina Client Example

```
dbconn_set "host", "localhost"  
dbconn_set "username", "sa"  
dbconn_set "password", "sa"  
dbconn_set "use ssl", true  
dbconn_set "database name", "my database.vdb"
```

## Telling SQL Yoga When Your Database Schema Changes

Now we have covered everything you need to do to get SQL Yoga up and running in a stack. You may recall that SQL Yoga provides much of the functionality that it does because it stores information about the tables and columns in your database. Because of this, it is important that you let SQL Yoga know if you add or delete tables and columns from your database.

### Telling SQL Yoga to Reload Database Schema



Execute the following handler in the message box to update the schema stored in the database object:

```
dbobject_reloadSchema  
put the result
```

### Save Your Work

After reloading the database schema you should save your work by calling **dbobject\_save** in the message box and then saving your stack.



## How To Work With Multiple Connection Objects

---

When working with a SQL Yoga Database object you can create any number of Connection objects by calling [dbconn\\_createObject](#). This lesson will show you how you can work with multiple Connection objects in your code.

In order to make writing SQL Yoga code easier, all API calls in the library assume you are targeting the default Connection Object of the default Database object. The name of the default Connection object is a property of a Database object and can be retrieved by calling [dbobject\\_get\(\)](#):

```
put dbobject_get("default connection")
```

For example, you look at the [sqlquery\\_createObject function](#) you will notice a 2nd and 3rd parameters are optional: *pConnection* and *pDBKey*. If the 4th parameter is empty then the Database object name returned by `sqlyoga_getDefaultDatabase()` will be used. If the 3rd parameter is empty then the default Connection object of the Database object will be used.

If you want to work with more than one Connection Object you have two options.

The first option is to pass in the name of the Connection object to any API calls that accept the *pConnection* parameter.

```
put sqlquery_createObject("lessons", "my connection name") into theQueryA
```

The second option is to change the name of the default Connection object by setting the "default connection" property for the Database Object using [dbobject\\_set](#):

```
dbobject_set "default connection", "my connection name"
```

## My Database Schema Has Columns/Tables That Are Reserved Words? What Can I Do?

---

If your database schema has tables or columns that are named after reserved words then you need to turn on the **quote identifiers** property for the Database Connection object.

```
dbconn_set 'quote identifiers', true
```

Once set to true, SQL Yoga will quote all table/column names that it generates in a query. Note that it will not quote any names that you hard code into a query.

# SQL Query Objects

## Introduction to SQL Query Objects

---

SQL Yoga provides a SQL Query object in order to facilitate querying the database. With SQL Query objects you set properties rather than build SQL strings. When the time comes to query the database SQL Yoga will generate a SQL query based on the properties of the SQL Query object.

SQL Query objects are represented as an array and are meant to be temporary. When you create a SQL Query you store it as a variable (local or global). When you create a SQL Query object you can base it off of a SQL Query Template object. This allows you to create reusable templates that have many of the properties you need already set.

## Creating A SQL Query Object

---

SQL Query objects are arrays that represent a pseudo object. This object defines what a database query will do. You create a SQL Query object by passing the name of a table to `sqlquery_create`. SQL Queries are not stored anywhere but are stored in temporary variables.

### Create A SQL Query Object

```
put sqlquery_createObject("lessons") into theQueryA
```

### Set Search Conditions Using User Search Input

A SQL Query object provides a special helper that will parse a user search term and turn it into the WHERE clause of a SQL statement. This keeps you from having to parse search strings like "Basketball OR Volleyball". The library handles it for you.

```
put "lessons.title contains :1 OR steps.title contains :1" into theConditions  
put the text of field "SearchCriteria" into theUserSearchString  
sqlquery_setConditionsFromUserSearchString theQueryA, theUserSearchString, theConditions
```

### Set Sorting Field

```
sqlquery_set theQueryA "order by", "lessons.title"
```

### Get Data From A SQL Query Object

Use any of the following to retrieve data from a SQL Query object. You can get data as an array, raw data or as records. The second parameter to each command is passed by reference and will contain the data from your database after the call has completed.

If you don't know what records are yet don't worry. That will be covered later.

```
sqlquery_retrieveAsArray theQueryA, theArrayA  
put the result into theError
```

```
sqlquery_retrieveAsData theQueryA, theData  
put the result into theError
```

```
sqlquery_retrieveAsRecords theQueryA, theRecordsA  
put the result into theError
```

## How Do I Create Records In a Database Table Using a SQL Query Object?

---

A SQL Query object provides a quick way of creating data in a database. This lesson will show you how to use `sqlquery_create` to do so.

**Note:** When using `sqlquery_create` none of the validations or callbacks you've defined for Table objects will be triggered.

### Create An Array

The first step is to fill in an array variable with the values you want to insert into the database. The array keys should be the names of the columns in your database table.

**## Create array with column values for new database record.**

```
put "My Dog's Car" into theRowA["title"]  
put false into theRowA["draft"]
```

### Create The Record

Once you have created the array you just pass it as a parameter to `sqlquery_create` along with the name of the table you want to create the record in. SQL Yoga will generate the query for you.

```
sqlquery_create "lessons", theValuesA  
put the result into theError  
put it into theAffectedRows
```

## How Do I Delete Records From the Database Table Using a SQL Query Object?

A SQL Query object provides a quick way of deleting data from a database. This lesson will show you how to use `sqlquery_delete` to do so.

**Note:** When using `sqlquery_delete` none of the validations or callbacks you've defined for Table objects will be triggered.

### Create An Object

The first step is to create a SQL Query object that can be passed to `sqlquery_delete`.

```
sqlquery_createObject("lessons") into theQueryA
```

### Define Conditions

When calling `sqlquery_delete`, the **where clause** property of the SQL Query object will be used to perform the delete. You define this property by setting the **conditions** property using `sqlquery_set`.

**## Specify that lesson with an id of 12 should be deleted.**

```
sqlquery_set theQueryA, "conditions", "id is :1", 12
```

### Delete

**## Perform the delete.**

```
sqlquery_delete theQueryA
```

```
put the result into theError
```

```
put it into theAffectedRows
```

## How To Use Cursors With a SQL Query Object

---

After setting properties for a SQL Query object you can use `sqlquery_retrieveAsArray`, `sqlquery_retrieveAsData` and `sqlquery_retrieveAsRecords` to retrieve your data without having to fuss with a database cursor. But what do you do if you need to access the database cursor for a query? This lesson will show you.

### `sqlquery_retrieve`

A SQL Query object can open a database cursor using **`sqlquery_retrieve`**. This command generates a SQL query using the SQL Query object properties and then opens the database cursor. You are then responsible for closing the database cursor using **`sqlquery_close`**.

You can access the RevDB cursor id through the "cursor" property:

```
put sqlquery_get(theQueryA, "cursor") into theCursor
put the result into theError
```

### **## Do something with the cursor**

```
sqlquery_close theQueryA
```

### Navigating the Cursor

Once you have opened a database cursor there are a couple of SQL Query object handlers that help navigate the cursor. They are:

- `sqlquery_moveToFirstRecord`
- `sqlquery_moveToLastRecord`
- `sqlquery_moveToNextRecord`
- `sqlquery_moveToPreviousRecord`
- `sqlquery_moveToRecord`

Here is an example:

```
put sqlquery_createObject("lessons") into theQueryA
sqlquery_set theQueryA, "conditions", "id < 4"
sqlquery_retrieve theQueryA
put the result into theError
```



```
if theError is empty then
    sqlquery_moveToNextRecord theQueryA
    if the result is true then
        ## You are now on the second record in the result set.
    end if
end if
```

## How Can I Convert a User Search String Into An AND/OR Search When Generating a Query?

---

Allowing a user to search a database for certain strings is a common operation. Creating a SQL query based on a string entered by the user can be a bit of work, however. SQL Yoga comes to the rescue with some helper commands that can parse strings for you automatically and convert them into search conditions for a SQL Query object.

### **sqlquery\_setConditionsFromUserSearchString**

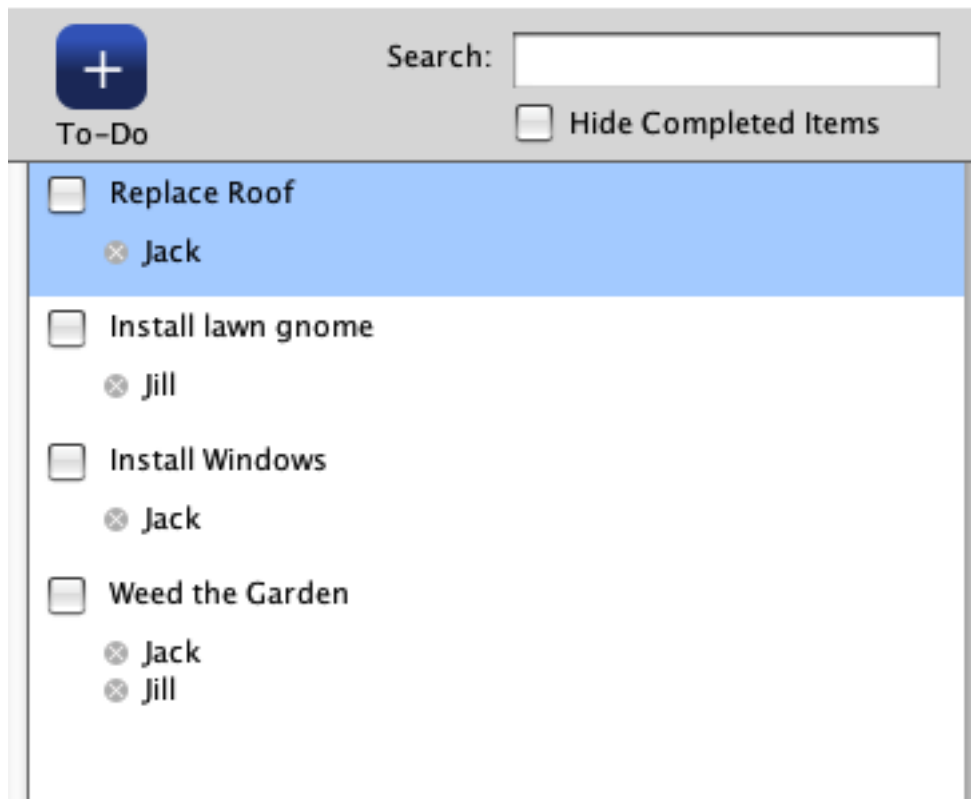
When working with SQL Query objects you can use [sqlquery\\_setConditionsFromUserSearchString](#) to set the "conditions" property of a SQL Query based on text a user enters.

In the following examples you will see results based on various search strings that were entered. The code to perform the search looks like this:

```
put sqlquery_createObject("todo_items") into theQueryA
put the text of field "Search" into theSearchString
put "todo_items.name contains :1" into theSearchCondition
sqlquery_setConditionsFromUserSearchString theQueryA, theSearchString, theSearchCondition
```

Basically the text that the user enters in the "Search" field will be parsed by SQL Yoga and inserted into the search condition that has been defined.

## Results With No Search Term

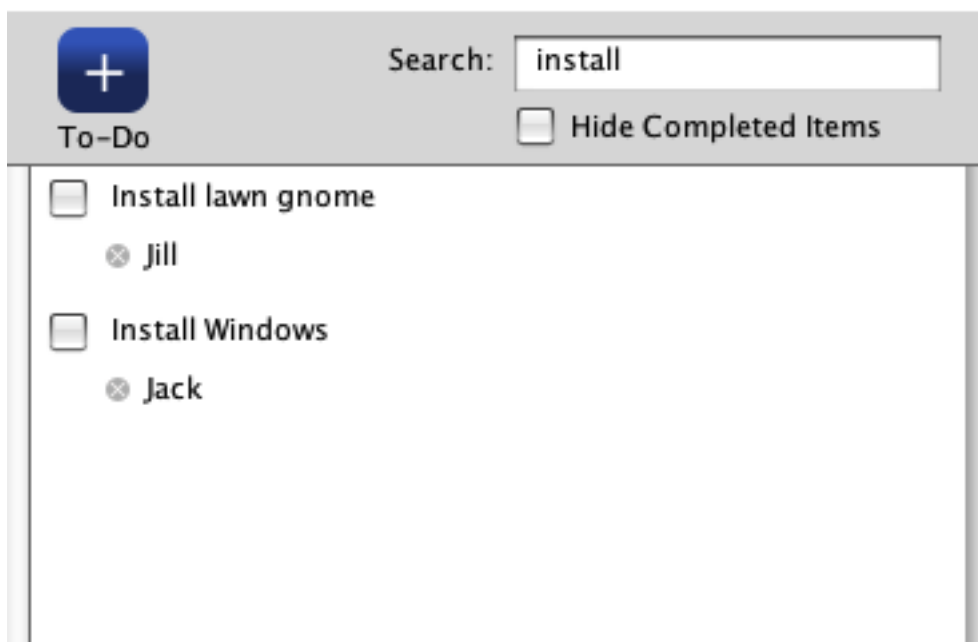


The screenshot shows a 'To-Do' application interface. At the top left is a blue square button with a white plus sign. To its right is a 'Search:' label followed by an empty text input field. Below the plus button is the text 'To-Do'. To the right of the search field is a checkbox labeled 'Hide Completed Items'. The list of tasks below is as follows:

- ☐ Replace Roof
  - ⊗ Jack
- ☐ Install lawn gnome
  - ⊗ Jill
- ☐ Install Windows
  - ⊗ Jack
- ☐ Weed the Garden
  - ⊗ Jack
  - ⊗ Jill

Here is what the results look like when no search term has been entered in the Search field.

## Single Word Search

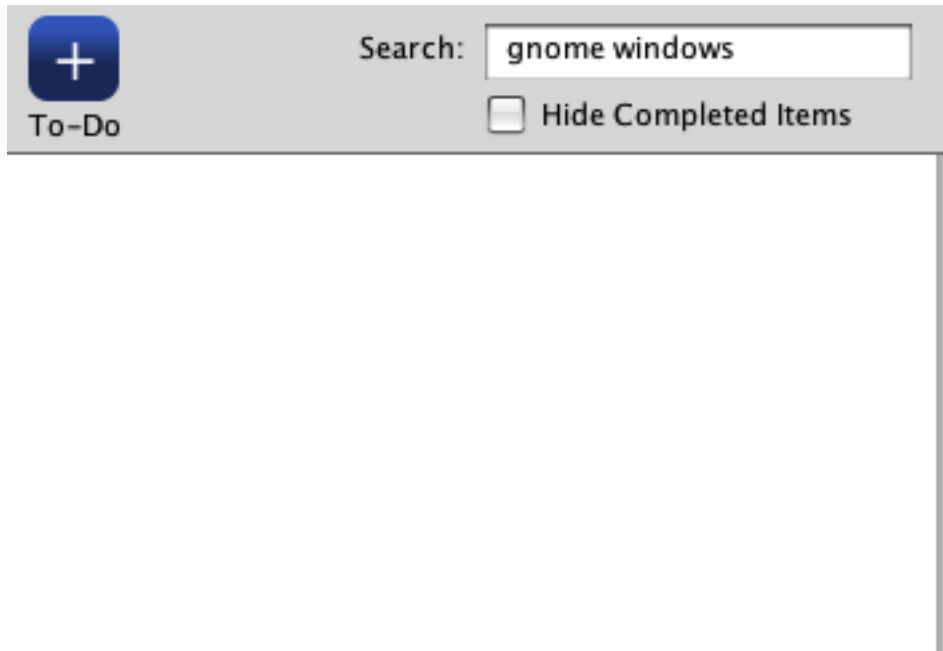


The screenshot shows the same 'To-Do' application interface, but with the search field containing the word 'install'. The results are filtered to show only tasks containing that word:

- ☐ Install lawn gnome
  - ⊗ Jill
- ☐ Install Windows
  - ⊗ Jack

If a single word is entered then any record that contains that word will be returned. Here you can see that two to-do items contain the word **install**.

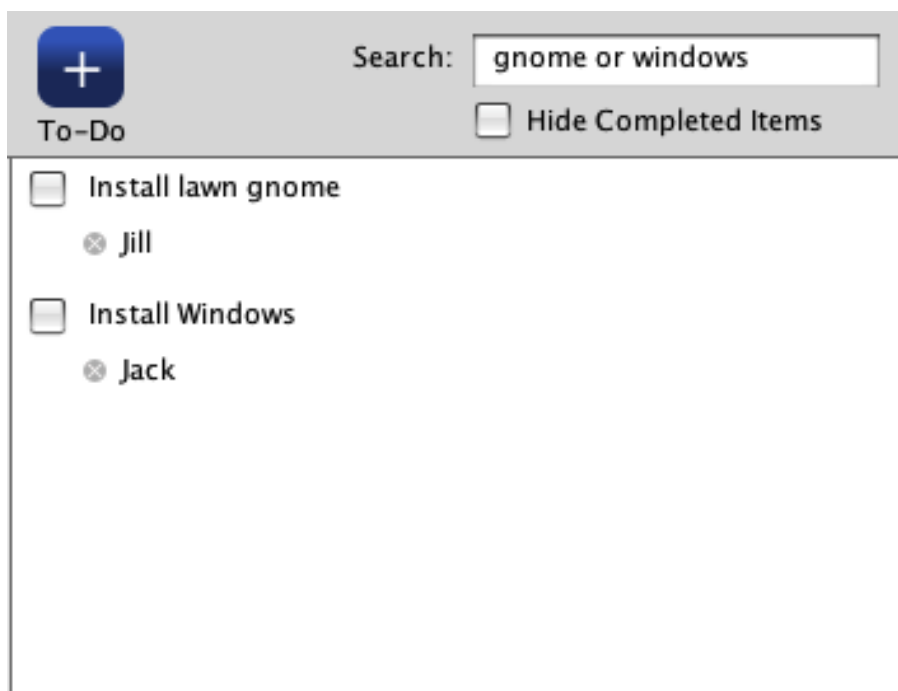
## Multiple Word Search



The screenshot shows a search interface for a To-Do list. At the top left is a blue square button with a white plus sign and the text 'To-Do' below it. To the right is a search bar with the text 'gnome windows'. Below the search bar is a checkbox labeled 'Hide Completed Items' which is currently unchecked. The main area below the search bar is empty, indicating no results were found for the search term 'gnome windows'.

If more than one word is used then the words are split up and results that contain all of the words are returned. Here you can see that no records contain both the word **gnome** AND the word **windows**.

## User Defined Boolean Searches

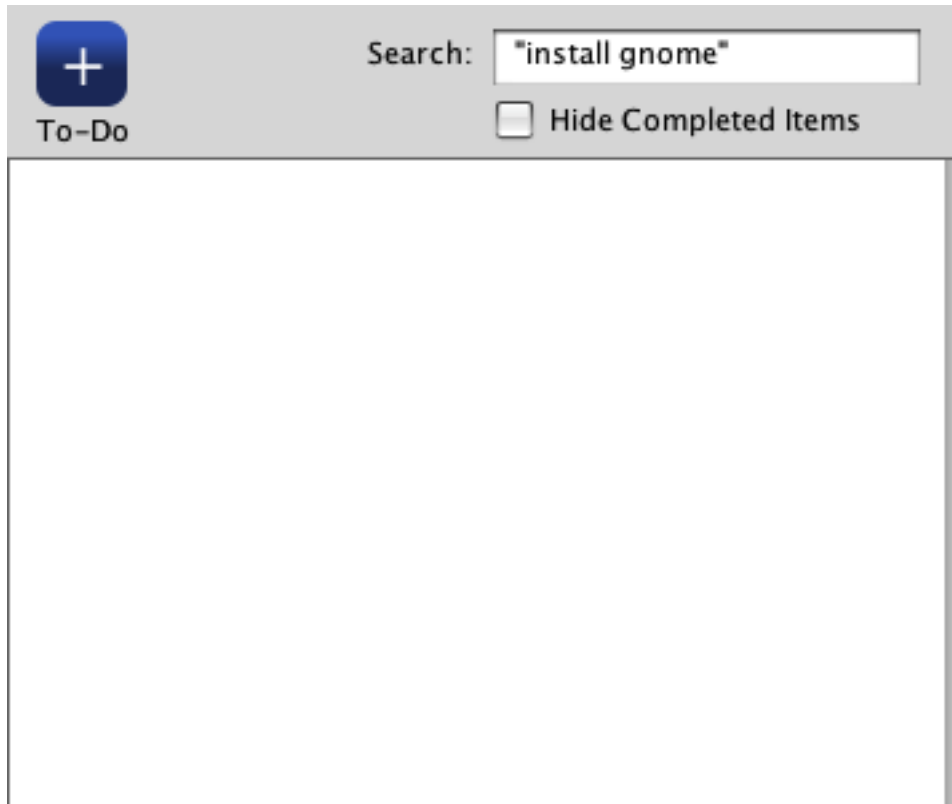


The screenshot shows the same search interface as before, but the search bar now contains 'gnome or windows'. The 'Hide Completed Items' checkbox remains unchecked. The search results area now displays two items, each with a checkbox and a user name:

- ☐ Install lawn gnome  
⊗ Jill
- ☐ Install Windows  
⊗ Jack

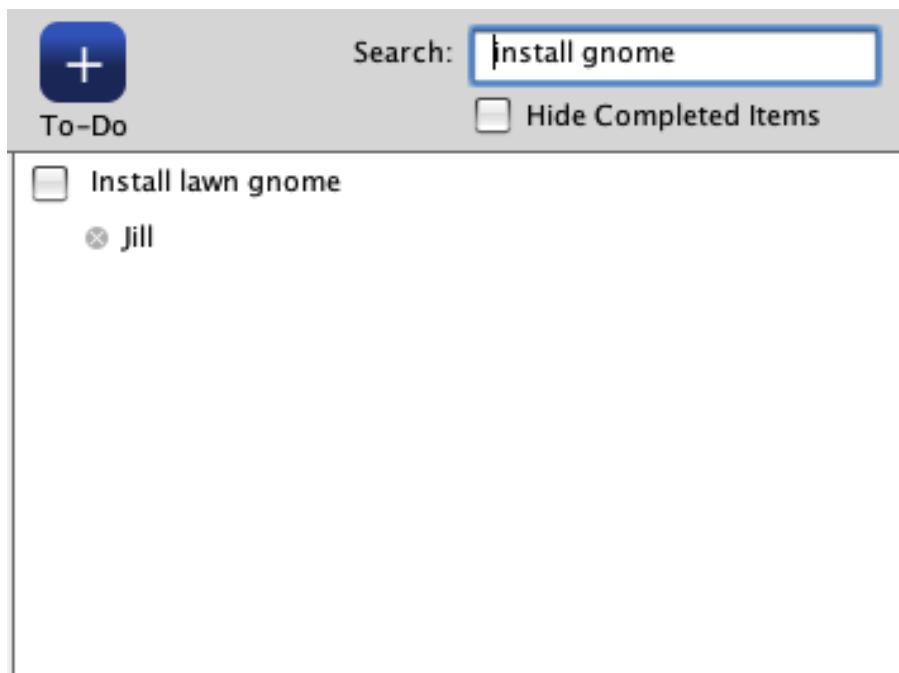
If the search term enters the words AND or OR then the search takes that into account. Here you can see that records that contain the word **gnome** OR the word **windows** are returned.

## Exact Matches



The screenshot shows a 'To-Do' application window. At the top left is a blue square button with a white plus sign and the text 'To-Do' below it. To the right is a search bar with the text 'Search: "install gnome"'. Below the search bar is a checkbox labeled 'Hide Completed Items'. The main area of the application is empty, indicating no results were found for the exact search term.

If the search term is wrapped in quotes then an exact match is searched for rather than both words. Here you can see that there are no records that contain the phrase **install gnome**.



The screenshot shows the same 'To-Do' application window. The search bar now contains the text 'Search: install gnome' without quotes. Below the search bar, a single result is displayed: a checkbox followed by the text 'Install lawn gnome'. Below this, there is a small 'x' icon followed by the name 'Jill'. The 'Hide Completed Items' checkbox remains unchecked.

If the quotes are removed then a result is returned as there is one record that contains the word **install** and the word **gnome**.

# Record Objects

## Introduction to SQL Record Objects

---

SQL Yoga has a feature called SQL Record objects. Using SQL Record objects SQL Yoga can create, retrieve, update and delete records from a database for you very easily. SQL Yoga can also create hierarchal SQL Record object arrays that represent the relationships in your database.

A SQL Record object is a simple Revolution array with special keys prefixed with @. These keys help SQL Yoga locate the table in the database that the record represents among other things.

### Single Record

A SQL Record object that represents a single record looks something like this:

```
theRecordA["@table"] = lessons
theRecordA["@connection"] = development
theRecordA["@database"] = default
theRecordA["id"] = 5
theRecordA["name"] = lesson title
theRecordA["description"] = lesson description
```

### Multiple Records

A SQL Record object can also represent many records in the database using multi-dimensional arrays. Here is what a SQL Record object with 3 records might look like:

```
theRecordA[1]["@table"] = lessons
theRecordA[1]["@connection"] = development
theRecordA[1]["@database"] = default
theRecordA[1]["id"] = 5
theRecordA[1]["name"] = lesson 1 title
theRecordA[1]["description"] = lesson 1 description
```

```
theRecordA[2]["@table"] = lessons
theRecordA[2]["@connection"] = development
theRecordA[2]["@database"] = default
theRecordA[2]["id"] = 7
theRecordA[2]["name"] = lesson 2 title
theRecordA[2]["description"] = lesson 2 description
```

```
theRecordA[3]["@table"] = lessons
```

```
theRecordA[3]["@connection"] = development  
theRecordA[3]["@database"] = default  
theRecordA[3]["id"] = 8  
theRecordA[3]["name"] = lesson 3 title  
theRecordA[3]["description"] = lesson 3 description
```



## Create Records in the Database Using SQL Record Objects

---

You can create records in your database using a SQL Record object. This lesson will show you how using `sqlrecord_create`.

### Create SQL Record Object

The first step is to create a SQL Record object.

```
put sqlrecord_createObject("lessons") into theRecordA
```

**## theRecordA is now an array that includes all fields in the 'lessons' table.**

### Fill In Values For Table

Now that you have a SQL Record object you can set the column values for the new record. In this example the "title" column and "description" column are being assigned values.

```
sqlrecord_set theRecordA, "title", "My Lesson"  
sqlrecord_set theRecordA, "description", "A short lesson I made."
```

### Create Record in Database

Once you are done assigning values to the columns you can create the record in the database.

```
sqlrecord_create theRecordA  
put the result into theError  
put it into theAffectedRows
```

**## In addition to the other column values, theRecordA now contains the database value for the 'id' (primary key) column.**

## Retrieve Records from the Database Using SQL Record Objects

---

You can use SQL Record objects to quickly find records in the database by using [sqlrecord\\_find](#).

### Create SQL Record Object By Searching (Example 1)

`sqlrecord_find` allows you to find records in a particular table in the database. The 2nd parameter is what defines the search conditions. You can pass in an integer in which case the primary key field of the table is searched and a single record is returned.

**## Look for a lessons record that has an 'id' of 1 (id is the primary key field).**

**## Store the found record in theRecordA.**

**## If no records are returned then the keys of theRecordA will be empty.**

```
sqlrecord_find "lessons", 1, theRecordA
```

### Create SQL Record Object By Searching (Example 2)

For more advanced searches you can pass in an array that contains any properties that you can set on a SQL Query object.

**## Advanced search using a parameters array**

```
put "id is in :1" into theParamsA["conditions"]
```

```
put "1,2" into theParamsA["condition bindings"][1]
```

```
put "id ASC" into theParamsA["order by"]
```

```
sqlrecord_find "lessons", theParamsA, theRecordsA
```

## Update Records in the Database Using SQL Record Objects

---

### Create SQL Record Object

**## Create SQL Record object by getting record from database.**

**## Store the found record in theRecordA**

sqlrecord\_find "lessons", 1, theRecordA

### Fill In Values For Table

sqlrecord\_set theRecordA, "title", "My New Lesson"

sqlrecord\_set theRecordA, "description", "A short lesson I made and modified."

### Update Record in Database

sqlrecord\_update theRecordA

**put** the result into theError

**put** it into theAffectedRows

## Delete Records from the Database Using SQL Record Objects

---

### Create Record Array

```
put sqlrecord_createObject("lessons") into theRecordA
```

### Fill In Primary Key Fields For Table

```
sqlrecord_set theRecordA, "id", 12 ## You must set the primary key column value(s)
```

### Delete Record in Database

```
sqlrecord_delete theRecordA
```

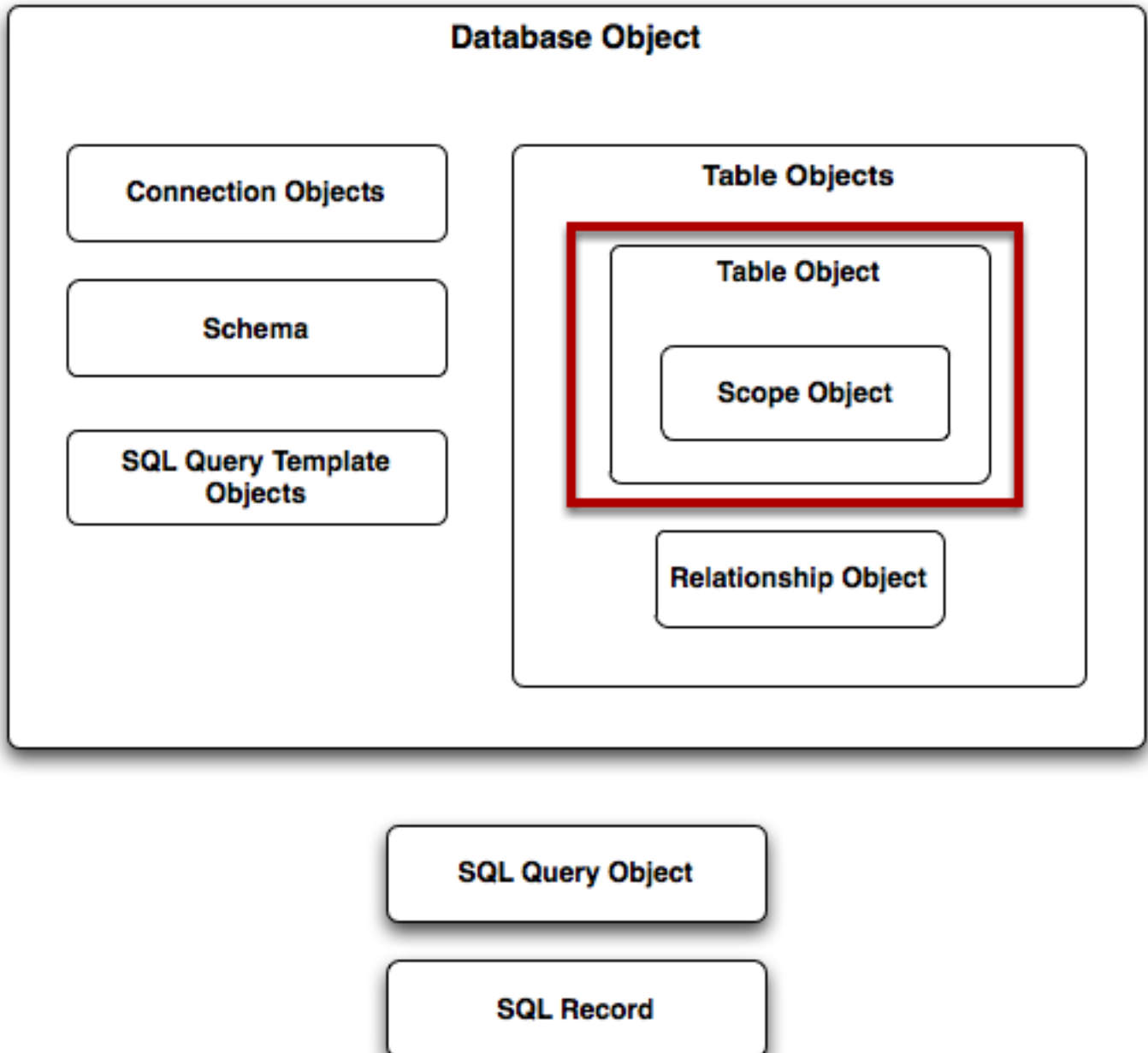
```
put the result into theError
```

```
## The lesson record where id = 12 was deleted from the database
```

# Table Objects

## Introduction to Table Objects

### The Table Object



SQL Yoga is able to automate a number of tasks because it imports information about the database tables and table fields. SQL Query and Record objects can be used out of the box without doing anything more than setting up a Connection to the database.

Some of SQL Yoga's features require some configuration on your part however:

- Adding custom properties to database tables.

- Setting search conditions using Scope objects
- Defining validation routines that are run before data is inserted into a table.
- Defining callbacks when records are created, deleted, retrieved or updated from a table.
- Automatically linking and unlinking records in two related tables.
- Turning SQL record sets into hierarchal arrays based on relationships between tables.
- Retrieving the related records from a table that is related to a SQL Record object's table.

Table objects unlock these features.

## Familiarizing Yourself With The "table objects behavior" Property

Each Database object in SQL Yoga has a [table objects behavior](#) property. The **table objects behavior** is a central location where you store the code to create various SQL Yoga objects such as Tables, Scopes and Relationships. It is also a location where you add validation routines and other callbacks.

Before you can create table objects you should assign the **table objects behavior** property of the Database object to a button in your application. The script of the button will be used to create objects and define certain behaviors.

### The dbobject.createTables Message

```

1  /**
2  * You can trigger this message for the database object by
3  * calling tableobjects_reload. Anytime you want to change the defined
4  * objects for the Database object modify this handler and call tableobjects_reload.
5  * You should then call dbobject_save.
6  */
7  on dbobject.createTables
8      ## Create Table Objects in order to define
9      ## relationships
10     tableobj_createObject "projects"
11     tableobj_createObject "todo_items"
12     tableobj_createObject "people"
13
14     ## Now that table objects exist create relationships
15     _CreateRelationships
16
17     _CreateScopes
18 end dbobject.createTables
19

```

The only message your behavior script must handle is the **dbobject.createTables** message. This message is sent whenever SQL Yoga is processing a request to create the table, relationship and scope objects for a Database object. All existing objects are deleted before this message is sent so you are creating the objects from scratch each time.

This message is triggered when a) you set the **table objects behavior** property of the Database object or b) you call [tableobjects\\_reload](#).



## Creating a Table Object

---

Table Objects are created in the **dbobject.createTables** handler of the button object script that is assigned to the [table objects behavior](#) property of the Database object.

### Create A Table Object

Here is an example how to create a table object for some tables named "projects", "todo\_items" and "people".

```
on dbobject.createTables
  ## Create Table Objects
  tableobj_createObject "projects"
  tableobj_createObject "todo_items"
  tableobj_createObject "people"
end dbobject.createTables
```

## How Do I Add Custom Properties To Tables?

---

Once you have created a Table object you can define your own properties for the table. For example, if a table has a *FirstName* and *LastName* field you might define a custom property called *Name* that combined those two fields in order to return the full name.

Table properties that you define can be accessed when working with SQL Record objects as a SQL Record object is an instance of a record in a table.

### Defining A New Table Property

Table object properties are defined in the button script assigned to the [table objects behavior](#) property of the Database object. You define a property for a Table object by creating a function with the following naming convention:

```
tableobj.get.[TABLE_NAME].[PROPERTY_NAME_WITHOUT_SPACES]
```

The function will be passed a SQL Record object array as the 1st parameter.

For example, to create add a **name** property to a **people** table you would add the following function to the behavior script:

```
function tableobj.get.people.name pRecordA  
    return pRecordA["firstname"] && pRecordA["lastname"]  
end tableobj.get.people.name
```

### Accessing a Table Object Property

You can access a Table object property when you are working with SQL Record objects by calling the [sqlrecord\\_get](#) function.

The following example retrieves a person from the database and then accesses the **name** property of the resulting SQL Record object. When accessing the property the `tableobj.get.people.name` function is called.

```
sqlrecord_find "people", 12, thePersonA  
put sqlrecord_get(thePersonA, "name") into theFullName
```

## Properties With Spaces

You can define properties that have spaces. Before calling the `tableobj.get` function all spaces are removed from the property name.

For example, if you tried to access the property **short bio** property like this:

```
put sqlrecord_get(thePersonA, "short bio") into theBio
```

then the following function would need to be defined in the behavior script:

```
function tableobj.get.people.shortbio pRecordA  
...  
end tableobj.get.people.shortbio
```