

Forum:[Hack Forums](#)

Thread: [\[RUBY\] Database Backend Written In Ruby - Create, Delete, Update, Insert, Fetch](#)

Toph Beifong - February 6, 2016, 12:40 pm

For anyone interested this is my basic database backend that i've been working on - <https://github.com/tophbeifong/SimpleRubyDatabase/blob/master/SimpleRubyDatabase.rb>

nearly 500 lines, very buggy but works(will update and fix)

Features:

- Create Table
- Update
- Insert
- Delete
- Fetch

I've posted my github repo but here is the code... (Dont flame if code is ugly in places, this was my first time using Ruby in a couple of months XD)

Code =begin

Table Management Database

=end

require "error-handling"

class Table

\$empty_table = []

\$data_to_return = Hash.new { |h,k| h[k] = [] }

\$errors = ErrorHandler.new

def create(table_number_rows , table_number_columns)

#first array(they start at zero) is the column names...

#loop through to create the rows

for row_count in 0..table_number_rows

#create new table row

\$empty_table.push(Array.new())

#populate the columns with "NULL", can be accessed later

for column_count in 0..table_number_columns.length

if row_count == 0

#populate the array with the column name

\$empty_table[row_count].push(table_number_columns[column_count])

```

else

  #if the column_name is "id" it will auto increment the row
  if table_number_columns[column_count] == "id"

    $empty_table[row_count].push("#{row_count}")

  else

    $empty_table[row_count].push("NULL")

  end

end

end

end

end

def fetch(row, column_name, where_clause)

  #split the where clause into an array so i can access the column name and the value
  where_clause = where_clause.split(" ")

  puts("#{column_name}")

  #make sure the column_name(s) exist
  for column in column_name

    #making sure the column exists
    if !$empty_table[0].include?(column)

      $errors.create("Syntax Error", "Column `#{column}` does not exist.")
      $errors.display("last")
      $errors.export("SRD_errors.log")

    else

      #get the array column name index number
      hash = Hash[$empty_table[0].map.with_index.to_a]

      #if the where clause is set create the column_id
      if where_clause[1]

        #set the column_id
        column_id = hash[where_clause[0]].to_i

      end

    end

  end

end

```

```
#if the row integer is higher than 0 run code to get the data...
if row > 0
```

```
  for loop_counter in 1..row
```

```
    #make sure we dont go over the size of the table
    if loop_counter >= $empty_table.length
```

```
      break
```

```
    end
```

```
    #get the data and return it, array
    if !column_id
```

```
      #put the selected data into the hash
      $data_to_return[column] << $empty_table[loop_counter][hash[column].to_i]
```

```
    else
```

```
      #check if the column data is equal to the where clause
      if $empty_table[loop_counter][column_id] == where_clause[2]
```

```
        #put data into the hash
        $data_to_return[column] << $empty_table[loop_counter][hash[column].to_i]
```

```
      end
```

```
    end
```

```
  end
```

```
end
```

```
end
```

```
end
```

```
#empty variable
return $data_to_return
```

```
end
```

```
def update(column_name, where_clause, text_to_change)
```

```
  if where_clause[2]
```

```
    #process where clause and change data
    where_clause = where_clause.split(" ")
```

```
    #column to update
    data_column_get = where_clause[0]
```

```

#data to compare it against
data_to_compare = where_clause[2]

#get the column_id
hash = Hash[$empty_table[0].map.with_index.to_a]

#set the column_id var
column_id = hash[data_column_get].to_i

#find the table_length
table_length = $empty_table.length

#loop through the rows
for counter in 0..table_length-1

  #if the table data == to the data_to_get
  if $empty_table[counter][column_id] == data_to_compare

    #get the column_name id
    get_column_to_change_id = hash[column_name].to_i

    #make the update
    $empty_table[counter][get_column_to_change_id] = text_to_change

  end

end

else

  #get the table rows count
  get_table_rows_count = $empty_table.length

  #loop through rows
  for row_counter in 1..get_table_rows_count - 1

    #loop through columns
    for column_counter in 0..$empty_table[0].length - 1

      #compare the column name with the database columns
      if $empty_table[0][column_counter] == column_name

        #make the update
        $empty_table[row_counter][column_counter] = text_to_change

      end

    end

  end

end

end

end

```

```

end

def insert(data_to_insert, columns_to_insert_to)

  column_ids = []
  next_row_id = $empty_table.length

  #check if columns_to_insert_to is set, if so it will insert the data into the correct fields
  if columns_to_insert_to != "NULL"

    #make sure the same number of columns as data to insert
    if data_to_insert.length != columns_to_insert_to.length

      $errors.create("Bad Parameters", "The data passed is not equal to the number of columns
passed")
      $errors.display("last")
      $errors.export("SRD_errors.log")
      return

    end

    $empty_table.push(Array.new)

    $empty_table[next_row_id].shift(next_row_id)

    #check if i need to auto increment
    if $empty_table[0][0] == "id"

      #map the column ids...
      hash = Hash[$empty_table[0].map.with_index.to_a]

      for column_counter in 0..columns_to_insert_to.length - 1

        #make sure the column to insert to exists else the insert fails
        if $empty_table[0].include?(columns_to_insert_to[column_counter])

          column_ids.push(hash[columns_to_insert_to[column_counter]].to_i)

        else

          $errors.create("Syntax Error", "Column `#{columns_to_insert_to[column_counter]}` does not
exist.")
          $errors.display("last")
          $errors.export("SRD_errors.log")
          return

        end

      end

    end

    #insert the data in the columns to insert
    for column_id in 0..column_ids.length - 1

```

```

    $empty_table[next_row_id][column_ids[column_id]] = data_to_insert[column_id]

end

#loop through and replace any nil elements as "NULL"
for column in 1..$empty_table[next_row_id].length

    $empty_table[next_row_id].insert(column, "NULL") if $empty_table[next_row_id][column] ==
nil

end

$empty_table[next_row_id][0] = "#{next_row_id}"

else

    #map the column ids...
    hash = Hash[$empty_table[0].map.with_index.to_a]

    for column_counter in 0..columns_to_insert_to.length - 1

        #make sure the column to insert to exists else the insert fails
        if $empty_table[0].include?(columns_to_insert_to[column_counter])

            column_ids.push(hash[columns_to_insert_to[column_counter]].to_i)

        else

            $errors.create("Syntax Error", "Column `#{columns_to_insert_to[column_counter]}` does not
exist.")
            $errors.display("last")
            $errors.export("SRD_errors.log")
            return

        end

    end

end

#insert the data in the columns to insert
for column_id in 0..column_ids.length - 1

    $empty_table[next_row_id][column_ids[column_id]] = data_to_insert[column_id]

end

#loop through and replace any nil elements as "NULL"
for column in 0..$empty_table[next_row_id].length

    if $empty_table[next_row_id][column] == nil

        $empty_table[next_row_id].insert(column, "NULL")

    end

end

```

```

    end

end

else

    next_row_id = $empty_table.length

    #make sure the data wont overrun
    if data_to_insert.length > $empty_table[0].length

        $errors.create("Bad Parameters", "The data passed is not equal to the number of columns
passed")
        $errors.display("last")
        $errors.export("SRD_errors.log")
        return

    end

    #create the new row
    $empty_table.push(Array.new)

    #loop through the data_to_insert
    for data_count in 0..$empty_table[0].length

        if data_to_insert[data_count]

            #add the data
            $empty_table[next_row_id][data_count] = data_to_insert[data_count]

        else

            #add null if the data is not long enough
            $empty_table[next_row_id][data_count] = "NULL"

        end

    end

end

end

end

def delete(where_clause)

    #split the where clause to get the data and the column name
    where_clause = where_clause.split(" ")

    #make sure the column name passed is valid
    if !$empty_table[0].include?(where_clause[0])

        #if not produce error

```

```

$errors.create("Syntax Error", "Column `${where_clause[0]}` does not exist.")
$errors.display("last")
$errors.export("SRD_errors.log")
return

else

  #get the array index for the rows to delete
  hash = Hash[$empty_table[0].map.with_index.to_a]
  rows_to_clear = []

  #loop through the rows
  for row_counter in 0..$empty_table.length - 1

    #check the data is the same, if so push the row id to the rows_to_clear array
    if $empty_table[row_counter][hash[where_clause[0]].to_i] == where_clause[2]

      rows_to_clear.push(row_counter)

    end

  end

  #check if the table uses auto incrementing id's
  if $empty_table[0][0] == "id"

    #loop through rows and column setting the data to NULL
    for row_number in rows_to_clear

      for column in 1..$empty_table[0].length - 1

        #set the data to NULL
        $empty_table[row_number][column] = "NULL"

      end

    end

  end

else

  for row_number in rows_to_clear

    for column in 0..$empty_table[0].length - 1

      $empty_table[row_number][column] = "NULL"

    end

  end

end

end

end

```

end

def export(data_export_name)

 #make sure 1 parameter is passed.

 if !data_export_name

 \$errors.create("Data Export Failed", "Export file name must be supplied.")

 \$errors.display("last")

 \$errors.export("SRD_errors.log")

 return

 end

 #create database table file

 File.open("tables/" + data_export_name, "w") do |file|

 #loop through the data and put it in the file

 for i in 0..\$empty_table.length - 1

 data_string = ""

 for column_data in \$empty_table[i]

 data_string = data_string + "<d%>" + column_data.to_s

 end

 #put the data in the database file

 file.puts(data_string)

 end

 end

end

def import(data_table_file)

 #varify the file does inface exist

 if File.file?("tables/" + data_table_file)

 #open error file and read each line.

 File.open("tables/" + data_table_file, "r").each do |file_line|

 #split the items to push individually into the errors array

 data_split = file_line.split("<d%>")

 data_split.shift

 data_length = data_split.length - 1

```
data_split[data_length] = data_split[data_length].chop

data_split.delete_at(data_length)

$empty_table.push(data_split)

    end

else

$errors.create("Data Import Failed", "Import file name must be supplied.")
$errors.display("last")
$errors.export("SRD_errors.log")
return

end

end

end
```

severos - February 6, 2016, 11:19 pm

Nice share, but there's already stuff like Rake and ActiveRecord to deal with databases, maybe it'll be better if you read their code and try to improve/fix it?

I haven't read it, just scrolling around, but noticed you use too many global variables (\$variable) are you sure they all need to be global?

Toph Beifong - February 7, 2016, 1:57 am

tbh there arent that many, just the 3... 1 is the main table which i made global so all the methods can access and manipulate the data. The second is just to return data but again is used across methods.

The third is for my error handling gem which although doesnt need to be global makes life easier(in my view anyway). And tbh i know rake etc exist i just wanted to try and make a functional database