

Single Table Inheritance Tricks..

Contributed by Nick Griffith
 Tuesday, 30 January 2007
 Last Updated Tuesday, 30 January 2007

Single table inheritance is an extremely powerful and extremely easy thing to do in rails. If you wanted to do this in java you would probably shoot yourself just designing it, but all that pain is relieved inside the wonderful rails framework.

Single table inheritance is an extremely powerful and extremely easy thing to do in rails. If you wanted to do this in java you would probably shoot yourself just designing it, but all that pain is relieved inside the wonderful rails framework.

The Scenario

You have the need for users to log into your system with different roles and each of these different roles can have different permissions. But these roles also can have things in common and share permissions. You could add a column into the table called 'role' and then do `if(role=="admin")` to determine what permissions you should do. But there is an easier solution. Single Table Inheritance.

The Table

First you would create a table called User and put in your common columns eg.

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL auto_increment,
  `manager_id` int(11) default NULL,
  `first_name` varchar(255) default NULL,
  `last_name` varchar(255) default NULL,
  `type` varchar(255) default NULL,
  `user_name` varchar(255) default NULL,
  `hash_password` varchar(255) default NULL,
  `email` varchar(255) default NULL,
  `parent_id` int(11) default NULL,
  `created_on` date default NULL,
  PRIMARY KEY (`id`)
)
```

The important column in this Single Table Inheritance is the type column. This column must be a string because what Rails does is it fills this column with the class name.

The Model

The model that goes along with this table is the User model. Now we want to create an Admin model. Admins have the same characteristics of Users, but have different permissions and abilities. To do this all you have to do is extend the User class instead of the ActiveRecord::Base class.

```
class Admin < User
end
```

Now you have the power to do calls like `Admin.find_all` and it will only grab the Admin users. But if you did `User.find_all` it would include admins in the query. Now you can add methods in the Admin class that are specific for that class only. You can build your permissions into those methods instead of having to dig permissions out of the base User object. You can also take this further. Say you have 3 classes, a User, a Manager, and an Admin. The user has no permissions, the manager has some permissions, and the Admin has both the managers permissions and its own. The way you would do this is to have managers extend user, and Admins extend Manager. All of this is kept in the same table so there is no need to change anything in the database.

Tricks with Extending many classes

there are a few things you can do to help this process out. In the previous scenario if i wanted to use the same 'create' page, I would have to determine what class of user I wanted to create before I saved it. This is a helpful method to put in your User class to help.

```
def self.factory(type,params)
  type ||= 'User'
  class_name = type
  if defined? class_name.constantize
    return class_name.constantize.new(params)
  else
    User.new(params) # or do what you will...
  end
end
```

Now when you want to create a user you would do put a role paramter in your page so that the controller can determine what you want:

```
@user = User.factory(params[:user][:role], params[:user])
@user.save
```

Another important trick is if you wanted to change that users class or want to know what kind of class he is for display purposes. You would do this by putting these getter and setter methods inside of your User model.

```
def class_type=(value)
  self[:type] = value
end

def class_type
  return self[:type]
end
```

So if you wanted to make a manager into an admin all you would do is `@manager.class_type="Admin"` and he would magically become an admin.

Hope these tips helped. Leave comments if you have any other suggestions or find a flaw with this tutorial.