envycasts

Advanced ActiveRecord Cheat Sheet

Loading Large Data Sets

Wrap multiple queries in a transaction

```
User.transaction do
  users_to_import.each do |user|
    User.create({...})
  end
end
```

ActiveRecord Extensions Gem to import lots of data

```
gem install ar-extensions
```

```
require 'ar-extensions'
fields = [:first_name, :last_name]
User.import fields, users, :validate => false
```

Use the Faker gem to generate fake data when you need to benchmark

```
gem install faker
```

```
[Faker::Name.first_name, Faker::Internet.email, Faker::Address.city]
```

Don't Forget Indexes

```
add_index :posts, :user_id
add_index :subscriptions, [:user_id, :magazine_id], :unique => true
add_index :orders, [:user_id, :store_id], :name => 'by_user_store'
```

Safely Using :include

```
Guide.find(:all, :include => :user)
Guide.find(:all, :include => [:user, :questions])
Guide.find(:all, :include => [:user, {:questions => [:user, {:answers => :user}]}])
Guide.find(:all, :include => [:user, {:questions => [:user, {:answers => :user}]}],
           :conditions => "answers.user_id = 639")
```

Don't use conditions with lots of includes

Using Counter Caches

Users Table

```
create_table :users do |t|
  t.string :first_name
  t.string :last_name
  t.integer :posts_count,
            :default => 0
end
```

Posts Table

```
create_table :posts do |t|
  t.string :title
  t.text :body
  t.integer :user_id
end
add_index :posts, :user_id
```

	without cache	with cache
u.posts.**size**	count query	no query look at cache
u.posts.**length**	pull all records & calls .length	pull all records & calls .length
u.posts.**count**	count query	count query

user.rb

```
class User < ActiveRecord::Base
  has_many :posts
end
```

post.rb

```
class Post < ActiveRecord::Base
  belongs_to :user, :counter_cache => true
end
```

Named Scope

```
class Post < ActiveRecord::Base
  named_scope :published, :conditions => { :published => true }, :order => "created_at"
  named_scope :in_last_day, lambda { { :conditions => ["created_at > ?", 1.day.ago]} }
  named_scope :within,  lambda { |date| { :conditions => ["created_at > ?", date] } }
  named_scope :between, lambda { |date, older_date|
          { :conditions => ["created_at > ? and created_at < ?", date, older_date] } }
end
```

Polymorphic Associations

envycasts

Table Before

```
create_table :comments do |t|
  t.string :body
  t.integer :post_id
  t.integer :company_id
  t.integer :project_id
  t.integer :task_id
end
```

Table After

```
create_table :comments do |t|
  t.string :body
  t.integer :commentable_id
  t.string :commentable_type
end

add_index :comments, [:commentable_id,:commentable_type]
```

Model Before

```
class Comment < ActiveRecord::Base
  belongs_to :post
  belongs_to :company
  belongs_to :project
  belongs_to :task
end
```

Model After

```
class Comment < ActiveRecord::Base
  belongs_to :commentable, :polymorphic => true
end
```

Other side of the relationship

```
class Project < ActiveRecord::Base
  has_many :comments, :as => :commentable
end
```

Remember, "commentable" is a keyword and can be set to any string you want.

Single Table Inheritance

Tables before

```
create_table :songs do |t|
  t.string  :name
  t.integer :user_id
  t.string  :file_path
end
```

```
create_table :videos do |t|
  t.string  :name
  t.integer :user_id
  t.string  :file_path
end
```

```
create_table :photos do |t|
  t.string  :name
  t.integer :user_id
  t.string  :file_path
end
```

Table after

```
create_table :media_files do |t|
  t.string  :name
  t.integer :user_id
  t.string  :file_path
  t.string  :type
end
```

media_file.rb

```
class MediaFile < ActiveRecord::Base
  belongs_to :user
end
```

song.rb

```
class Song < MediaFile
  # Song specific methods go here
end
```

Rails 2.1 Enhancements

```
User.find(:all)   → User.all
User.find(:first) → User.first
                    User.last
```

Partial Update Exceptions

```
u = User.first
u.login_will_change!
u.login.downcase!
u.first_name_will_change!
u.first_name << " meister"
u.save
```

Dirty Object Methods

```
p = Post.first
p.changed?       # false
p.title          # "Old Title"
p.title = "Cha-Cha"
p.changed?       # true
p.title_changed  # true
p.title_was      # "Old Title"
p.title_change   # ["Old Title", "Cha-Cha"]
p.changed        # ["title"]
p.changes
  # { 'title' => ["Old Title", "Cha-Cha"] }
```