

# Ruby on Rails



Austin User Group  
<http://austinonrails.org>

# ActiveRecord Migrations

Overview and Strategies

Eric Stewart

<http://www.eric-stewart.com>

# Ruby on Rails

==

web application development that's  
easy, agile, and fun

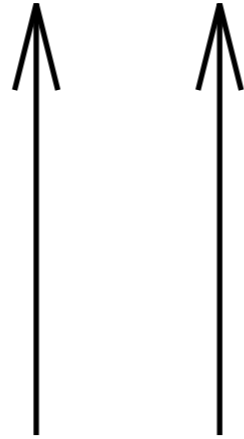
Even for database backed apps!

ActiveRecord

Database schemas evolve...

...along with your code.

New(Schema)



Old(Schema)

# ActiveRecord::Migration



A Ruby API for managing the evolution of a schema

What can you do with  
Migrations?

# Available Transformations

<code>create_table</code>	<code>change_column</code>
<code>drop_table</code>	<code>remove_column</code>
<code>add_column</code>	<code>add_index</code>
<code>rename_column</code>	<code>remove_index</code>

```
1 class AddTags < ActiveRecord::Migration
2   def self.up
3     create_table :tags do |t|
4       t.column :name, :string
5       t.column :created_at, :datetime
6       t.column :updated_at, :datetime
7     end
8
9     create_table :articles_tags, :id => false do |t|
10      t.column :article_id, :integer
11      t.column :tag_id, :integer
12    end
13  end
14
15  def self.down
16    drop_table :tags
17    drop_table :articles_tags
18  end
19 end
```

or...

any kind of SQL or ActiveRecord  
code that you like.

It's just Ruby

# Changes not Supported by ActiveRecord

```
11_make_join_unique.rb
1 class MakeJoinUnique < ActiveRecord::Migration
2   def self.up
3     execute "ALTER TABLE `pages_linked_pages` ADD UNIQUE
4             `page_id_linked_page_id` (`page_id`,`linked_page_id`)"
5   end
6
7   def self.down
8     execute "ALTER TABLE `pages_linked_pages` DROP INDEX
9             `page_id_linked_page_id`"
10  end
11 end
```

Line: 11 Column: 5 Ruby on Rails Soft Tabs: 2 self.down

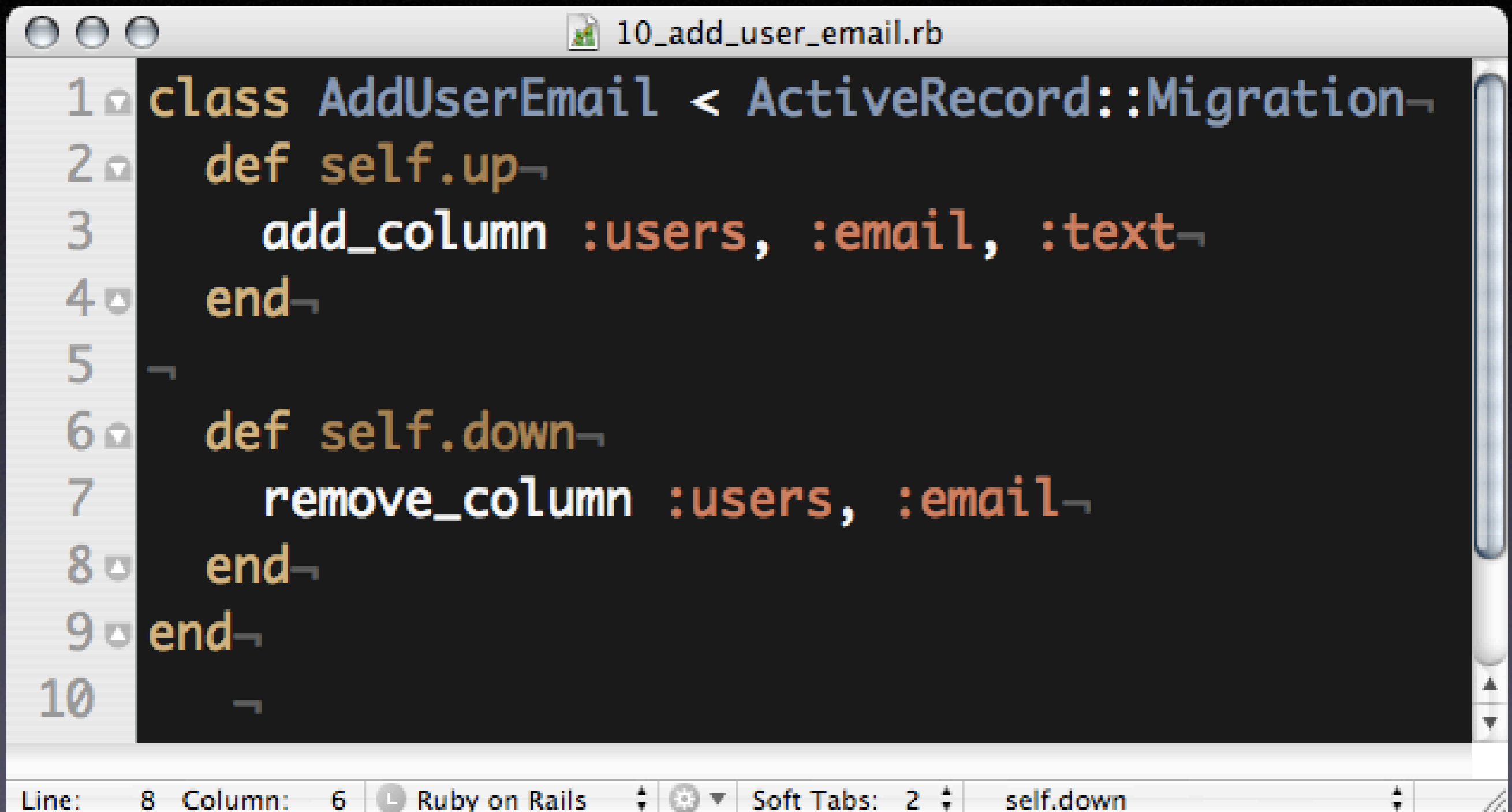
# Creating and Applying Migrations

# Create the migration source

```
me:~$ script/generate migration AddUserEmail
```

This creates `db/migrate/2_add_user_email.rb`

# Edit the migration file



```
1 class AddUserEmail < ActiveRecord::Migration
2   def self.up
3     add_column :users, :email, :text
4   end
5
6   def self.down
7     remove_column :users, :email
8   end
9 end
10
```

Line: 8 Column: 6 Ruby on Rails Soft Tabs: 2 self.down

# Apply the Migration

```
me:~$ rake migrate
```

```
me: ~$ rake migrate VERSION=VersionNumber
```

Check in your migration(s)

Along with the code that uses  
those changes

ActiveRecord::Schema

## Manage schema defined in code

```
me:~$ rake db_schema_import
```

Executes db/schema.rb against current db

```
me:~$ rake db_schema_dump
```

Writes current schema to db/schema.rb

# db/schema.rb

```
1 ActiveRecord::Schema.define(:version => 15) do
2   create_table :authors do |t|
3     t.column :name, :string, :null => false
4   end
5
6   add_index :authors, :name, :unique
7
8   create_table :posts do |t|
9     t.column :author_id, :integer, :null => false
10    t.column :subject, :string
11    t.column :body, :text
12    t.column :private, :boolean, :default => false
13  end
14
15  add_index :posts, :author_id
16 end
```

# Why Use Migrations?

- Database agnostic schema management
- Transformations are data preserving
- Easy support multiple database back-ends
- Stop worrying about SQL syntax
- Manage incremental schema transforms along with related code changes
- Supports iterative development
- Easier to use deployment tools (i.e. SwitchTower)

# Why Not?

- You may need to integrate tightly with a particular database engine, and use non-portable SQL in your code
- Still have to worry about ActiveRecord support/bugs for various DBs
- Migration risks increase with size of user base
- Your DBA “Doesn’t get it”

# Supported Databases

- MySQL
- PostgreSQL
- SQLite
- SQL Server
- Oracle
- DB2 (Not supported, though ActiveRecord supports)

Gotchas

# Migrate in Two Dimensions

Migrations need to be upgraded  
with code changes

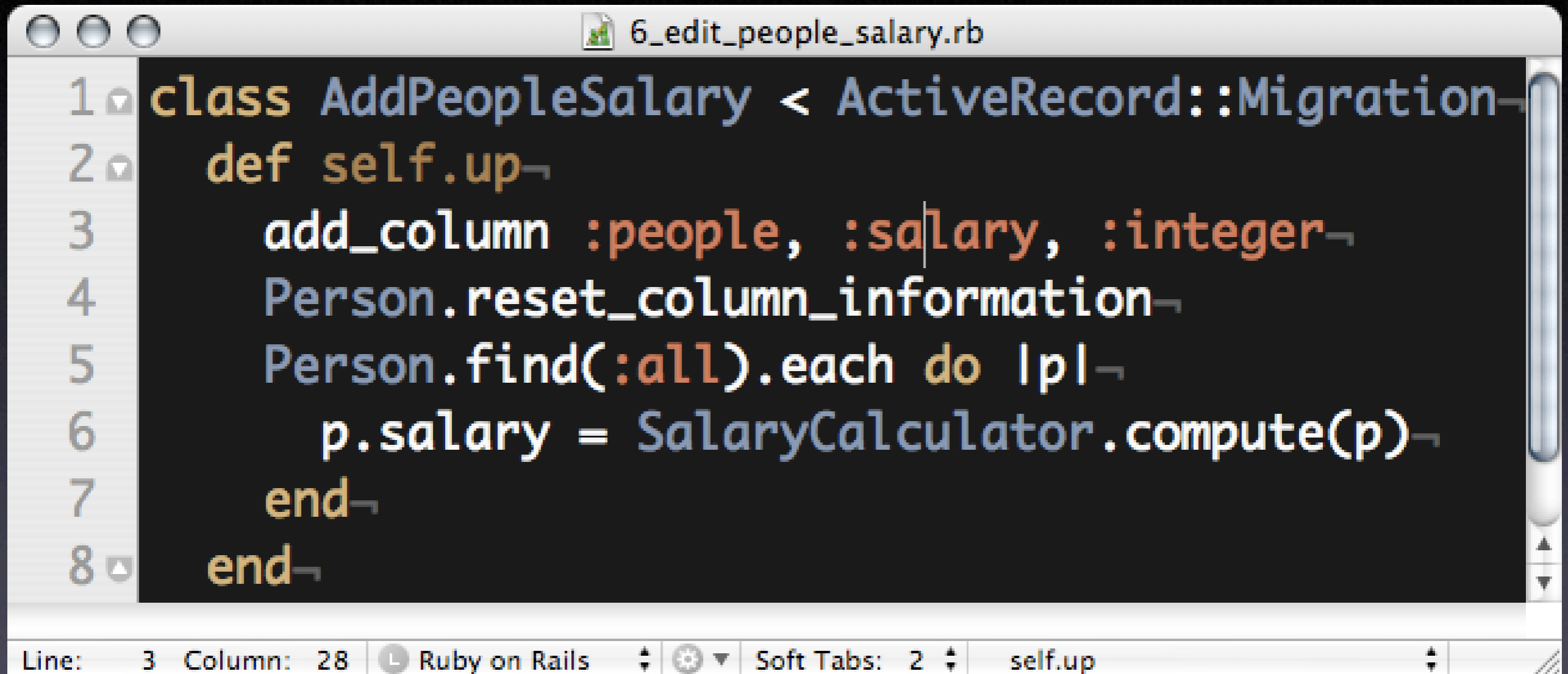
# Some Migrations are Irreversible

7\_remove\_empty\_tags.rb

```
1 class RemoveEmptyTags < ActiveRecord::Migration
2   def self.up
3     Tag.find(:all).each { |tag|
4       tag.destroy if tag.pages.empty?
5     }
6   end
7
8   def self.down
9     # not much we can do to restore deleted data
10    raise IrreversibleMigration
11  end
12 end
```

Line: 12 Column: 4 Ruby on Rails Soft Tabs: 2 self.down

# Using Models after Transformation



```
1 class AddPeopleSalary < ActiveRecord::Migration
2   def self.up
3     add_column :people, :salary, :integer
4     Person.reset_column_information
5     Person.find(:all).each do |p|
6       p.salary = SalaryCalculator.compute(p)
7     end
8   end
end
```

Line: 3 Column: 28 Ruby on Rails Soft Tabs: 2 self.up

# Strategies

# Manage Initial Schema

- `db/schema.rb`
- Initial schema migration
- Rails Schema Generator

# Migrating Legacy Databases

- Might work if you want to preserve existing data, but change the DB to only work with Rails
- Strong advantage to adoption of ActiveRecord conventions through refactoring
- Point Rails to the database and write migrations to modify schema to be more Rails friendly while transforming/preserving data

I've done this quite a few times now. Most often, it's a migration from an older MySQL db to PostgreSQL. In other situations, I would build an AR::Migration to mimick the current version of the db and then work from there on adding new features.

— Robby Russell on migrating legacy DBs

# Resources

- <http://rails.rubyonrails.com/classes/ActiveRecord/Schema.html>
- <http://rails.rubyonrails.com/classes/ActiveRecord/Migration.html>
- Schema Generator - <http://wiki.rubyonrails.com/rails/pages/SchemaGenerator>
- Typo - <http://typo.leetsoft.com> - Open source blogging packing in Rails that uses migrations and has a lot of examples