ActiveRecord

Refresher

Why ActiveRecord

1. SQL is rigid, limited and easy to make syntax errors. Ruby is less so.
2. Same Ruby commands for different databases. (SQLite isn't the only one there!)
3. More versatility and options with Ruby DataTypes.

Plain Ruby

```
class User

end



User.where(id:1) => #no method error



ActiveRecord Jr



class User < Database::Model

end



User.where('id=?', 1) => #User Object
```

ActiveRecord

```
class User < ActiveRecord::Base

end



User.where('id=?', 1) => #Argument error
User.where(id:1) => #User Object
```

Comparison

```
Ruby Land                      |              SQL
                     |
User.all => List of users           |  SELECT * FROM users
                     |
User.where(id:1)                 |  SELECT * FROM users WHERE users.id = 1
                     |
User.first.purchases.where(price:300) |  SELECT * FROM purchases WHERE purchases.price = 300
```

ActiveRecord 101 : How it all begins

ActiveRecord is … a way to simplify coding by defining a standardised way to execute commands to change your database.

Database Essentials: What do we start with?

Tables and columns To interact: Use SQL language

e.g

```
CREATE TABLE users
ALTER TABLE users ADD COLUMN first_name string
DROP TABLE users
```

Weaknesses of using SQL:

1. It’s all strings! I can’t standardised them easily.
2. Easy to make syntax errors.
3. Batch commands (Everything is a single command ended with ;)
4. How do I undo my previous changes?

Database phases in business e.g :

1. Grocery store has products and daily transactions.
2. Grocery store opens up new branches, products need to be tracked b location
3. Grocery store set up online service, need to keep track of deliveries and online transaction

   phase_one CREATE TABLE products CREATE TABLE transactions phase_two CREATE TABLE branches

```
ALTER TABLE products ADD COLUMN branch_id

ALTER TABLE transactions ADD COLUMN branch_id
```

   phase_three CREATE TABLE deliveries CREATE TABLE online_transactions

The SOLUTION = ActiveRecord Migrations

Example

```
class CreateStudents < ActiveRecord::Migration
  def change
    create_table :students do |t|
      t.string :first_name
      t.string :last_name
      t.string :email
      t.string :phone_number
      t.date   :birthday
      t.timestamps
    end

    add_column :teachers, :subject, :string
  end
end
```

ActiveRecord Migrations is how you set up your database tables and structures.

It's like creating the blueprint for your database, or in other words, your schema.

How it works:

1. Write your migration file. (Create blueprint)
2. Start your migration! (Submit blueprint for execution)
3. ActiveRecord checks which migration needs to be run! (Check blueprint)
4. AcitveRecord runs the migrations! (Execute blueprint)
5. Rollback to previous versions if you need to. (Remove changes)

ActiveRecord 102 : How to write your migration

1. Write your migration file. (Create blueprint)
2. ActiveRecord checks which migration needs to be run! (Check blueprint)

The typical migration file:

location : `db/migrate/<migration_file_name>` migration_file_name : `20170126103050_create_students.rb`

```
20170126103050 => YYYYMMDDhhmmss
create_students => file_name


YYYYMMDDhhmmss_file_name.rb


class CreateStudents < ActiveRecord::Migration
end
```

My db/migrate folder:

```
20170126103050_phase_one.rb
20170126113030_phase_two.rb
20170126124030_phase_three.rb
```

```
20170126125040_final_phase.rb
```

My db/migrate folder:

```
20170126103050_phase_one.rb


    class PhaseOne < ActiveRecord::Migration
    end


20170126113030_phase_two.rb


    class PhaseTwo < ActiveRecord::Migration
    end


20170126124030_phase_three.rb


    class PhaseThree < ActiveRecord::Migration
    end


20170126125040_final_phase.rb


    class FinalPhase < ActiveRecord::Migration
    end
```

ActiveRecord::Migration Convention

1. Timestamp in filename determines the version.

    - Used to determine whether to execute or not.
    - A past version will not be executed
    - Or to rollback to a specific time
2. File_name has to correspond to the class name

ActiveRecord 103: Executing your migration

1. Start your migration! (Submit blueprint for execution)

Q: How to trigger? A: rake db:migrate

But, how does your app know what to do?

```
rake -T
```

Rakefile The location where all your rake tasks are stored

```
desc "create the database"
task "db:create" do
  touch 'db/ar-students.sqlite3'
end


desc "drop the database"
task "db:drop" do
  rm_f 'db/ar-students.sqlite3'
end
```

But everything is stored in different places! How do I know what to run?

```
ActiveRecord Jr => app.rb

ActiveRecord ar-student-schema =>
 db/config.rb
 Rakefile
```

ActiveRecord 104: Symmetric Migrations OR Rolling Back

Remember this?

1. Grocery store has products and daily transactions.
2. Grocery store opens up new branches, products need to be tracked b location
3. Grocery store set up online service, need to keep track of deliveries and online transaction

```
phase_one
 CREATE TABLE products
 CREATE TABLE transactions
phase_two
 CREATE TABLE branches
 ALTER TABLE products ADD COLUMN branch_id
 ALTER TABLE transactions ADD COLUMN branch_id
phase_three
 CREATE TABLE deliveries
 CREATE TABLE online_transactions
```

I run `rake db:version = 20170126120000_phase_three.rb`

BUT IT'S NOT WORKING! MY CODE IS BROKEN!

How do I go back to phase_one?

in SQL:

```
DROP TABLE deliveries

DROP TABLE online_transactions


ALTER TABLE products DROP COLUMN branch_id

ALTER TABLE transactions DROP COLUMN branch_id

DROP TABLE branches
```

in ActiveRecord: `rake db:migrate version= 20170126113020_phase_two.rb`

Symmetric Migration:

If by running my migration I can execute changes to my database, I can revert the changes my migration made by running the reverse of my migration.

```
def up

create_table :students do |t|

...

end

end


def down

drop_table :students

end
```

Some Git Fu

```
git add .                    => Stages/ Prepares my files for commit and pushing.

git commit -m 'my_message'    => Commits the changes under a message for clarity

git push                     => Pushes my changes to the remote repository
git clone <url>              => Clones the file and generates a git init file for
my git configuration, i.e where my repository is.

If you didn't clone, you have no git init and you willl have to set up your remote
repository location manually.

git init
git remote add origin git@github.com:<username>/<filename>

Helpful commands to have an idea where you are:

git status      => Shows inforation about your current git status(are you up to
   date? What files have changed? What branch am I on?)
git remote -v   => What's my repository?
git help        => HELP!
```