ActiveRecord$^{\text{Beta}}$

Your Challenge

Today you’ll be creating a Ruby class that makes it effortless to read & write from a Postgres database.

This challenge won’t be easy, but it will provide you with deep understand of how exactly the ActiveRecord library *does what it does*.

ActiveRecord

This library abstracts (hides) the details of SQL so instead of writing queries like:

```
INSERT INTO students (first_name, last_name, dob)

VALUES (‘James’, ‘Cube’, ‘1973-4-9’);
```

ActiveRecord

You can write Ruby like this:

```
student = Student.new(first_name: “James”,
                      last_name: “Cube”,
                      dob: Date.new(1973, 4, 9))

student.save
```

ActiveRecord

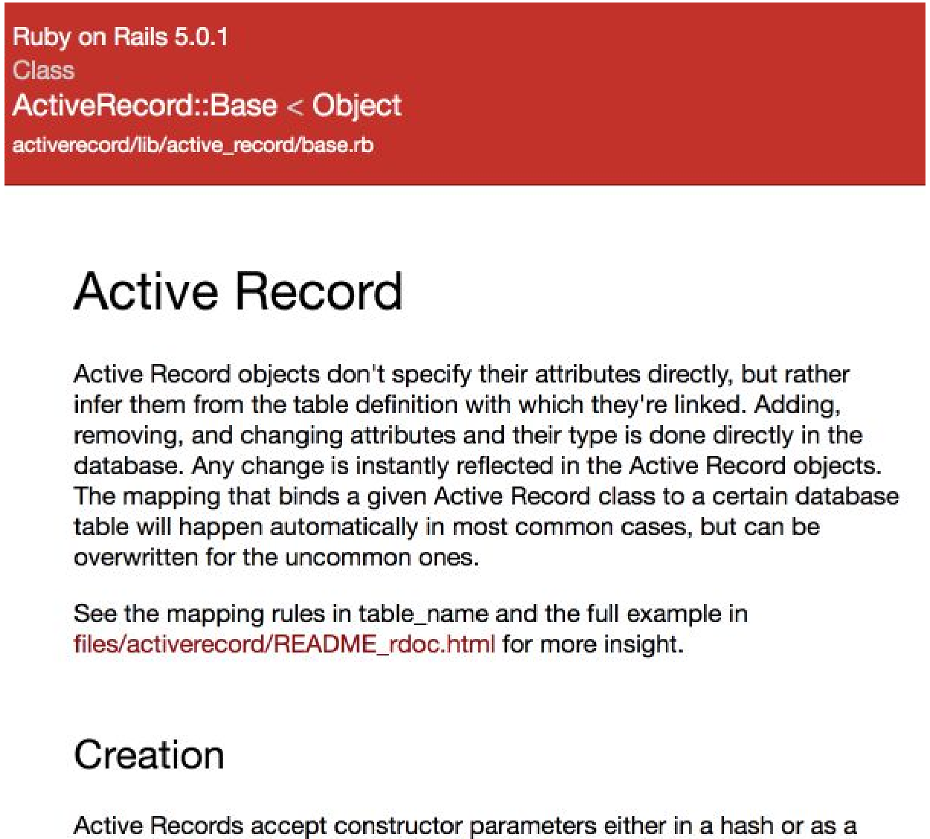
What do you think happens right after `student.save`?

```
INSERT INTO students (first_name, last_name, dob)

VALUES (‘James’, ‘Cube’, ‘1973-4-9’);
```

Remember, it’s just an abstraction. There is *no magic here.*

ActiveRecord



Ruby on Rails 5.0.1
Class
ActiveRecord::Base < Object
activerecord/lib/active_record/base.rb

Active Record

Active Record objects don't specify their attributes directly, but rather infer them from the table definition with which they're linked. Adding, removing, and changing attributes and their type is done directly in the database. Any change is instantly reflected in the Active Record objects. The mapping that binds a given Active Record class to a certain database table will happen automatically in most common cases, but can be overwritten for the uncommon ones.

See the mapping rules in table_name and the full example in files/activerecord/README_rdoc.html for more insight.

Creation

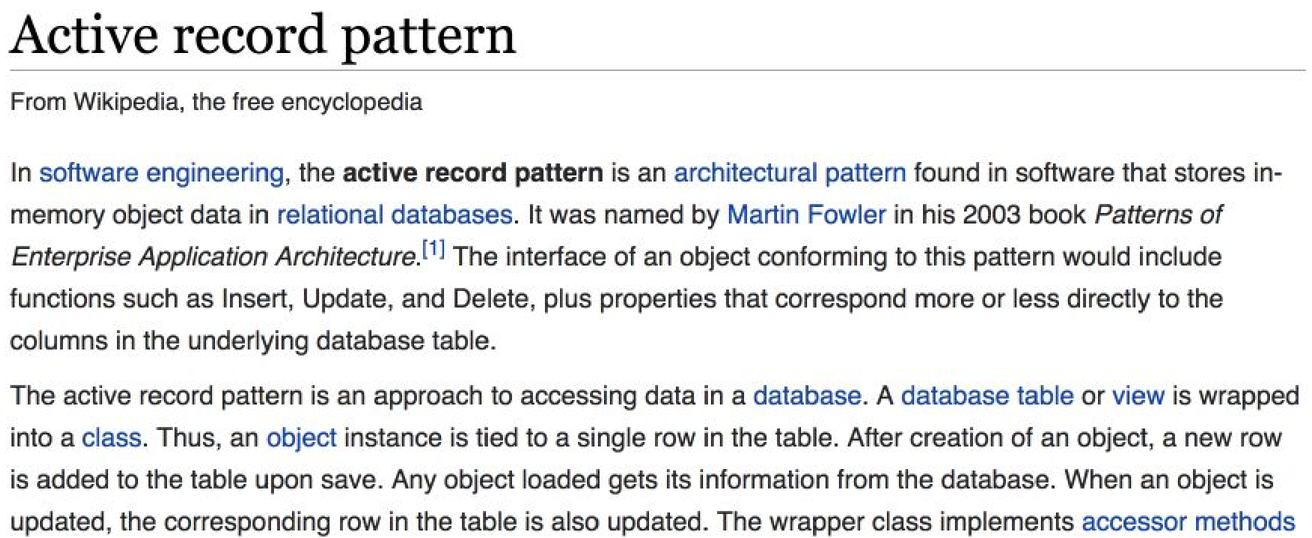
Active Records accept constructor parameters either in a hash or as a

← This is the library.

ActiveRecord

Before using the ActiveRecord library….

You’ll be implementing the Active Record Pattern.



Active record pattern

From Wikipedia, the free encyclopedia

In software engineering, the **active record pattern** is an architectural pattern found in software that stores in-memory object data in relational databases. It was named by Martin Fowler in his 2003 book *Patterns of Enterprise Application Architecture*.[1] The interface of an object conforming to this pattern would include functions such as Insert, Update, and Delete, plus properties that correspond more or less directly to the columns in the underlying database table.

The active record pattern is an approach to accessing data in a database. A database table or view is wrapped into a class. Thus, an object instance is tied to a single row in the table. After creation of an object, a new row is added to the table upon save. Any object loaded gets its information from the database. When an object is updated, the corresponding row in the table is also updated. The wrapper class implements accessor methods

Motivations

- ActiveRecord is a huge library with a ton of functionality.
- While it’s not all that difficult to get started, a shallow understanding of how AR works will make debugging frustrating.
- In general you’re bound to regret using any piece of software without understanding at least *some* of how it works. Get in the habit of peeling back a few layers of anything you commit to using.
- This is a great chance to try of some Ruby’s dynamic programming abilities.

Active Record Pattern

Database	**Object Oriented Programming**
Table `persons, orders`	Class `Person, Order`

Columns
 	
`first_name, last_name`
	Instance variables (getter/setters)
	
	`person.first_name`
	`person.last_name = “Green”`

Queries like `SELECT, INSERT, UPDATE, DELETE` `SELECT * FROM students WHERE id = 2;`	Instance or class methods. `student = Student.find_by(id: 2)`

Boundaries

As you build your own implementation of the Active Record pattern it will help if you stay aware of the boundaries between systems.

Failing to keep these boundaries in mind will lead to hand-waving explanations and muddy your understanding.

Keep in mind:

- When does the Ruby end and the SQL begin?
- Given the limitations of each system (Ruby & an RDBMS) what realistically can and cannot be done?

Debugging

- Use the best debugging tools you can. Give `byebug` a try if you haven’t yet.
- Keep a database console open to review the effects of running your code.
- Print out any SQL statements you execute.
- Be alert of any syntax issues when generating SQL (e.g. not quoting a string).
- Let the challenge’s tests guide you.
- Always have a prediction: what do you expect to happen?