Bitly: Extending Our Link Shortener

Time	~ 2 - 3 hours
Learning Goals	• Understand validations • Understand exception handling • Understand error handling

Objectives

Counter

Add a `click_count` field to your `urls` table, which keeps track of how many times someone has visited the shortened `URL`. Add code to the appropriate place in your controller code so that at any time someone hits a short URL, the counter for the appropriate `Url` is incremented by 1.

Add Validations

Add a validation to your `Url` model so that only `Urls` with valid URLs get saved to the database. Read up on ActiveRecord validations.

What constitutes a "valid URL" is up to you. It's a sliding scale, from validations that would permit lots of invalid URLs to validations that might reject lots of valid URLs. When you get into it you will see that expressing the fact "x is a valid URL" in Ruby Land or SQL Land is never perfect.

For example, a valid URL could range across:

- Any non-empty string
- Any non-empty string that starts with "http://" or "https://"
- Any string that the Ruby URI module says is valid
- Any URL-looking thing which responds to a HTTP request, i.e., we actually check to see if the URL is accessible via HTTP

Some of these are easily expressible in SQL Land. Some of these are hard to express in SQL Land, but ActiveRecord comes with pre-built validation helpers that make it easy to express in Ruby Land. Others require custom validations that express logic unique to our application domain.

The rule of thumb is that where we can, we want to always express constraints in Ruby Land and also express them in SQL Land where feasible.

Add Exception and Error Handling

When you try to save (create or update) an ActiveRecord object that has invalid data, ActiveRecord will fail. Some methods like `#create!` and `#save!` throw an exception. Others

Bitly: Extending Our Link Shortener

like create (without the ! bang) return the resulting object whether the object was saved successfully to the database or not, while #save will return false if perform_validation is true and any validations fail. See create and save for more information.

You can also call #valid? or #invalid? on an ActiveRecord object to see whether its data is valid or invalid.

Use this and the errors method to display a helpful error message if a user enters an invalid URL, giving them the opportunity to correct their error.

Extra reading: More on Validations, Constraints, and Database Consistency

We often want to put constraints on what sort of data can go into our database. This way we can guarantee that all data in the database conforms to certain standards, e.g., there are no users missing an email address. Guarantees of this kind - ensuring that the data in our database is never confusing or contradictory or partially changed or otherwise invalid - are called consistency .

If we think of this as a fact from Fact Land, these constraints look like:

- A user must have a first_name
- A user must have an email
- Two users cannot have the same email address, or equivalently, each user's email must be unique
- A Craigslist post's URL must be a valid URL, for some reasonable definition of valid

These facts can be recorded in both SQL Land and in Ruby Land, like this:

Fact Land	SQL Land	Ruby Land
A user must have an email address	NOT NULL constraint on email field	validates :em
A user must have a first name	NOT NULL constraint on first_namefield	validates :firs
A user's email address must be unique	UNIQUE INDEX on email field	validates :em