Production Readiness: Speeding Up Your Slow App

Time	~ 2 - 3 hours
Learning Goals	• Learn more about ActiveRecord Callbacks. • Gain familiarity with common read / write performance tradeoffs.

Make your app slow

Make sure your app auto generates a short URL for a given URL before creation through an `ActiveRecord::Callback`. This should have been done in the first challenge for your Bitly clone.

If you have not done this, there should be a `before_create :shorten_url` in your `Url` model.

Now, if you have that in place, download this file and seed the long form URLs into your database. Note that there are 500k URLs in the file.We recommend using SQL mass import to prevent the import time to take forever. But if you go with this approach, it will not trigger any `ActiveRecord::Callbacks` so you will need to figure out how to assign a short link or unique short 'key' for each row of Url. For example,

```
values = "('http://example.com/hfu2hiu3nkunkjniu4n3kjnjk'),
  ('http://example.com/jhfwnkjnui4y7hkjnfjknkjnfk'),
  ('http://example.com/uiejorunkuyengof68enf8oe'),
  ('http://example.com/ufopooppenfiienkf7890eh3');"


Url.transaction do
 Url.connection.execute "INSERT INTO urls (long_form) VALUES #{values}"
end
```

There should now be 500k URLs in your URLs table. Try visiting any of the short links available and you might notice that it takes some time for your app to redirect your browser to the long form URL. If it's slow, you're ready to move on to the next step!

Migrations & Database Indexes

Try to speed up your app! You'll need to create a new migration to make any changes to your database. It's a horrible practice to drop your database and make changes to migration files created earlier. It's incredible high risk to do that especially when your app is already in production or you have teammates you collaborate with. Just don't do it at all!

Now, one way to speed up querying a large database is to use indexes at the right places. A database index is an auxiliary data structure which allows for faster retrieval of data stored in the database at the cost of maintaining the consistency of all the indexes. They are keyed off of a specific column or set of columns so that queries like "Give me all people with a last name of 'Smith'" are fast.

Submit It!

Upload any files you've changed to the repo-link and post that as your attempt.

More on Database Indexes

This is purely extra; the below will make more sense later and after you've been playing around with databases for a few more weeks. Don't sweat it if everything below sounds crazy.

In some instances they work like a hash, e.g., you could imagine a hash where the keys were last names and the values were an array of people with those last names. Every time we inserted or updated data in the database we'd have to make sure to do the same to our hash index.

This doesn't work for a wide variety of queries, though, like "Give me all people who have been members of our site for longer than a year" or "Give me all the people whose last name begins with 'R'." For this reason the default data structured used in most databases is something more general than a hash table.

If you're curious, you can read more about what index types Postgres supports. The default index type for most databases is a B-tree because it allows for generally efficient inserting, updating, deleting, and searching using both equality and comparison, i.e., `=`, `<`, `>`, `<=`, and `>=`.