• Learn symmetric migrations.

• Learn about the `VERSION` argument to the `rake db:migrate` command.

Learning Goals

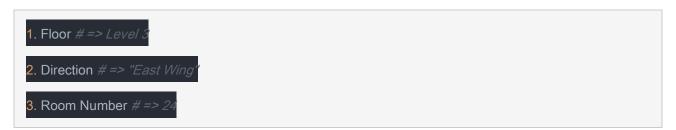
Over time, it is inevitable that your database requirements will change. When they do, it is important that we are able to roll back database schema changes in case something goes wrong.

By the end of this challenge you should know:

The differences between standalone vs symmetric migrationsWhen to use symmetric migrationsHow to write symmetric migrations

Why Symmetric Migrations

For instance, imagine you are building a database for a hospital. For every patient, you store their location in 2 columns:



One day your CTO comes in and requests you to combine the 3 columns into 1 column, e.g.



This seems like a pretty simple request, you just need to

- 1. Add a "location" columns to the table
- 2. Copy information from the "floor", "direction", "room_number" columns to the "location" column
- 3. Delete the 3 columns: "floor", "direction", "room_number"

This is fairly straightforward to do in Active Record, you just need to:

- 1. Write a migration to add 1 columns named "location" with type "string" to the table
- 2. Update each row in the table (in a ruby file / in the console)
- 3. Write a migration to delete the 3 columns from the table

But remember, this is a hospital's database! Each piece of patient-related information is critical and no data loss can be tolerated. What if after running all the migrations, the hospital uses the new DB for 1 month, then decides they want to revert to using the orignal 3 columns to store patient location? You are now faced with some problems:

- 1. You can't just drop the DB and use a backup from 1 month ago, the current DB contains 1 additional month of data
- 2. ActiveRecord can revert the column creation/deletion automatically, however the data in those columns need to be converted manually

To prevent this kind of problem from occuring, you should write your migrations to be symmetric!

What are Symmetric Migrations

Each ActiveRecord Migration has a up and down component. When you write a change method, ActiveRecord tries to infer the up and down components for you.

For example, a change migration file like below:

class CreateUsers < ActiveRecord::Migration[5.0]

def change

```
t.string :name

t.string :email

t.timestamps

end

# add a 'user_id' column to the 'registrations' table

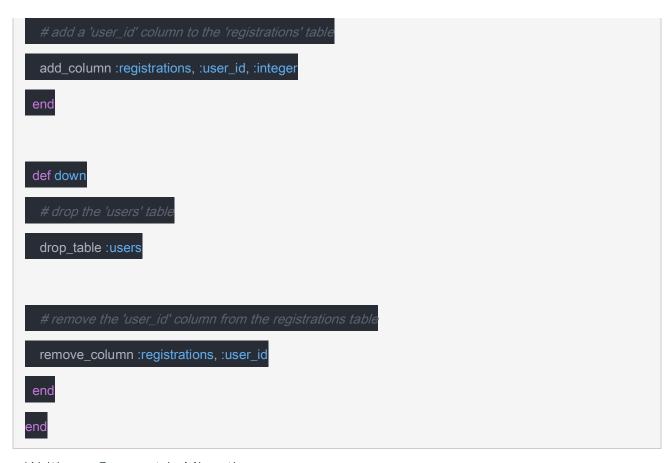
add_column :registrations, :user_id, :integer

end

end
```

Active Record will automatically infer that this change migration should be converted to the following:

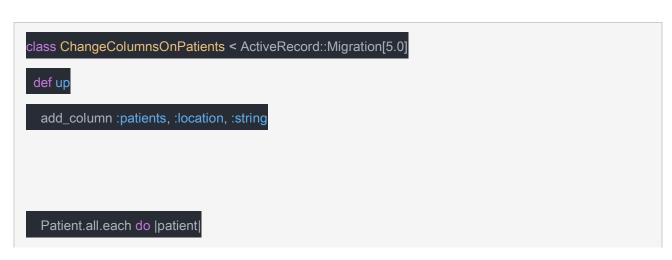




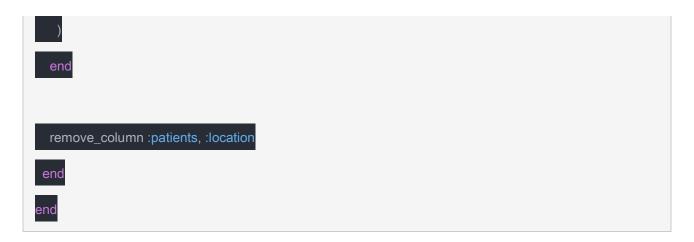
Writing a Symmetric Migration

In the hospital example above, ActiveRecord doesn't know how to handle copying the code from the 1 column to the 3 columns. In this scenario, we will write a symmetric migration by writing the up and down migrations directly.

Let's say this is for a patients table, the more complex symmetric migration might look like below



```
patient.update(location: "#{patient.floor}-#{patient.direction}-#{patient.room_number}")
 end
 remove_column :patients, :floor
 remove_column :patients, :direction
 remove_column :patients, :room_number
end
 add_column :patients, :floor
 add_column :patients, :direction
 add_column :patients, :room_number
 Patient.all.each do |patient|
  location_info = patient.location.split("-")
  floor = location_info[0]
  direction = location_info[1]
  room_number = location_info[2]
  patient.update(
   floor: floor,
   direction: direction,
   room_number: room_number
```



Objectives

Now that you know what are symmetric migrations, lets try implementing it in your arstudent-schema.

A common problem in real life apps is whether to store a person's name as first_name, middle_name, last_name (in 3 columns) or join them into 1 column (e.g. name)

Try to write a symmetric migration in the previous challenge to merge the first_name and last_name for each student into 1 column name. You can reference the hospital example given above.

Special Note

For your ar-student-schema project, you cannot use commands you may have found online, e.g. rails db:rollback, rake db:rollback as this is not a Rails project.

Instead, when you want to test rolling back the DB, please use the following format:



run all migrations

rake db:migrate

roll back second migration

rake db:migrate VERSION=20170421000000 # => version number depends on the timestamp of your first migration