In this lecture, we will discuss…

- Migrations
- Incentives for migrations
- How migrations work

Migrations – Agility Incentive

- Agility inside the applications is a given!
  - *"The only constant thing about software requirements is that they are always changing"*
- But how do we monitor and undo changes to the DB?
- There is no easy way – manually applying and undoing changes is messy and error-prone.

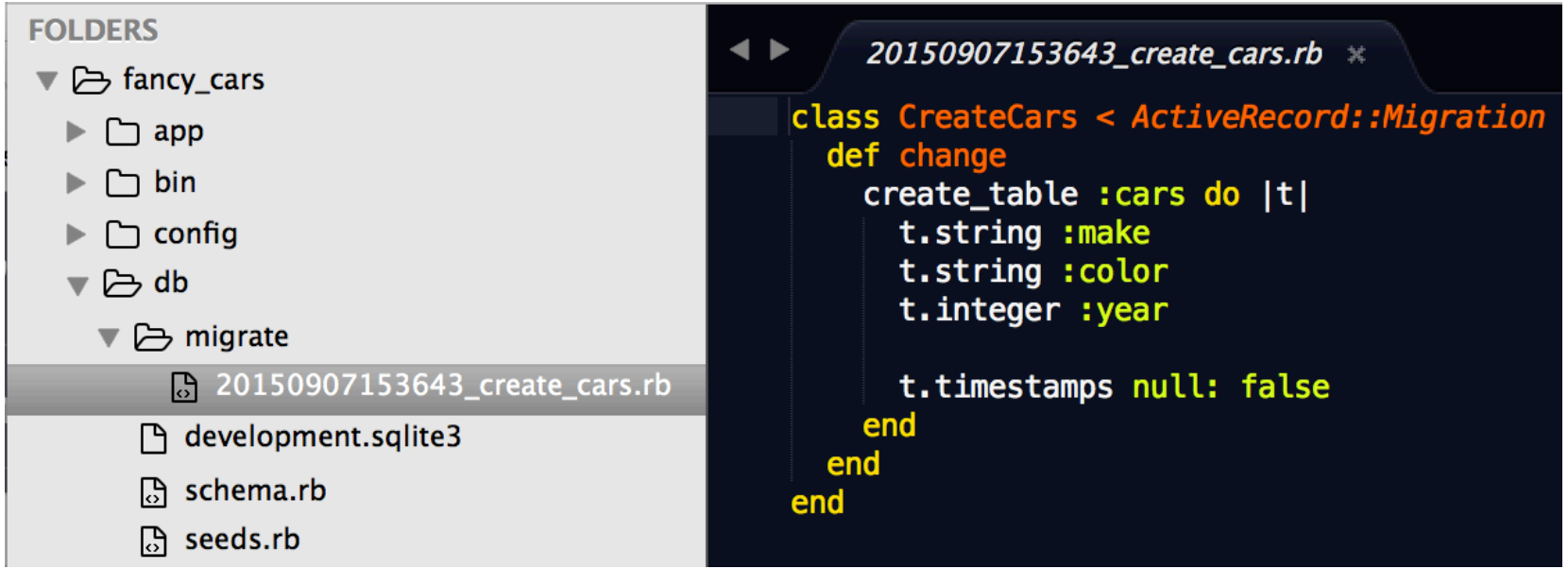
Migrations: Cross Database Incentive

- Typically, SQL (or more specifically DDL) is used to create and modify tables for a particular relational database
- This feels intuitive, but what if you have to switch databases in the middle?
  - For example, develop on SQLite and deploy to Postgres

Enter Migrations

- Ruby `classes` that extend `ActiveRecord::Migration`
- File name needs to start with a timestamp (year/month/date/hour/minute/second) and be followed by some name, which becomes the name of the `class`
- This timestamp defines the sequence of how the migrations are applied and acts as a database version of sorts or snapshot in time

Migrations


FOLDERS
fancy_cars
app
bin
config
db
migrate
20150907153643_create_cars.rb
development.sqlite3
schema.rb
seeds.rb
20150907153643_create_cars.rb
class CreateCars < ActiveRecord::Migration
def change
create_table :cars do |t|
t.string :make
t.string :color
t.integer :year
t.timestamps null: false
end
end
end

Creating Migrations

- You can create migrations by hand, but it's obviously much less error-prone to use a generator
- We already saw that scaffold generator creates a migration (unless passed `--no-migration` flag)
- There is also an explicit migration generator

Applying Migrations

- Once the migration is created (either manually or through a generator) it needs to be applied to a database in order to “migrate” the database to its new state
- No two migrations can have the same `class` name
- You run `rake db:migrate` to apply all migrations in `db/migrate` folder in (timestamp) order

How Do Migrations Work?

- Migration code maintains a table called `schema_migrations` table with one column called `version`
- Once the migration is applied – its version (timestamp) goes into the `schema_migrations` table
- It therefore follows that running `db:migrate` (on the same set of migrations) multiple times will have no effect

Anatomy of Migration

- So, what actually goes inside the `ActiveRecord::Migration` subclass?
- Either
  - `def up`
    - Generate db schema changes
  - `def down`
    - Undo the changes introduced by the `up` method
- Or, just `change` method when Rails can guess how to undo changes (most of the time)

Database Independence

- Instead of specifying database-specific types for particular DB type – migrations let you specify logical database types
- The ruby database adapter you are using does the translation to the actual DB type.
  - So, for MySQL – it will end up being one type, Postgres another and so on.

Actual Type Mapping

Migration type	Sqlite3	Oracle	Postgres
:binary	blob	blob	bytea
:boolean	tinyint(1)	number(1)	boolean
:date	date	date	date
:datetime	datetime	date	timestamp
:decimal	decimal	decimal	decimal
:float	float	number	float
:integer	Integer	number(38)	integer
:string	varchar(255)	varchar2(255)	character varying
:text	text	clob	text
:time	datetime	date	time

Extra Column Options

- Besides specifying logical types, you can specify up to three more options (when the underlying DB supports it)
- `null: true` or `false`
  - When `false` – a `not null` constraint is added
- `limit: size`
  - Sets a limit on the size of the field
- `default: value`
  - Default value for the column (Calculated once!)

Decimal Column Options

- Decimal columns (optionally) take two more options
- `precision: value`
  - Total number of digits stored
- `scale: value`
  - Where to put the decimal point
    - For example, precision 5 and scale 2 can store the range -999.99 to 999.99

Summary

- ✧ Migrations are just Ruby classes that get translated into DB speak
- ✧ Table in the DB keeps track of which migration was applied last

What's Next?

- ✧ Creating and altering tables and columns