

How to run migrations?

- `$ rails db:migrate [VERSION=202208221622]`

VERSION -> you can say the timestamp, so you only do a specific migration.

Adding references

- Adding the `belongs_to`, which is the table that should have the id

```
oskarhinojosa@Oskars-MacBook-Pro example % rails g migration AddUserRefToProducts user:references
invoke active_record
create db/migrate/20220816184125_add_user_ref_to_products.rb
oskarhinojosa@Oskars-MacBook-Pro example %
```

- Please note the conventions

```
class AddUserRefToProducts < ActiveRecord::Migration[7.0]
  def change
    add_reference :products, :user, null: false, foreign_key: true
  end
end
```

1. Table on which we want to add the foreign key
2. the referenced table
3. `foreign_key: true` -> add the constraint

`index: false` -> really bad performance so try to avoid (sometimes will be necessary).

Creating JOIN tables

- A product that may have multiple customers and a product that may have multiple products

```
oskarhinojosa@Oskars-MacBook-Pro example % rails g migration CreateJoinTableCustomerProduct customer product
invoke active_record
create db/migrate/20220816184358_create_join_table_customer_product.rb
oskarhinojosa@Oskars-MacBook-Pro example %
```

Join Tables: an explanation

- A product may have multiple customers and a customer may have multiple products

```
class CreateJoinTableCustomerProduct < ActiveRecord::Migration[7.0]
  def change
    create_join_table :customers, :products do |t|
      t.index [:customer_id, :product_id]
      t.index [:product_id, :customer_id]
    end
  end
end
```

They are Intermediary tables on many to many relationships.

A word of knowledge

- Using rails g migration would mean you would have to create your own models
- To avoid this, simply use the model generator
 - The model generator will create the relevant table and model, it is an aid for helping scaffolding (creating multiple files at once)
 - Please know it would also create fixtures (tests)



scaffolding -> generate multiple with one command.

Creating using generators depends on the team, since it can create things that you don't actually will use.

How to specify your own index?



```
class CreateUsers < ActiveRecord::Migration[7.0]
  def change
    create_table :users do |t|
      t.string :name, index: true
      t.string :email, index: { unique: true, name: 'unique_emails' }
      # in the last command, please note that you are also specifying the table
      # column name
    end
  end
end
```

Modifying tables



- Note that in the image, two columns were removed

```
change_table :products do |t|
  t.remove :description, :name
  t.string :part_number
  t.index :part_number
  t.rename :upccode, :upc_code
end
```

All these modifying commands are not necessary for us to memorize them.
You can have multiple actions on one line using a comma.

Using modifiers

```
t.string :first_name, limit: 40  
t.string :last_name, limit: 40
```



- `comment` Adds a comment for the column.
- `collation` Specifies the collation for a `string` or `text` column.
- `default` Allows to set a default value on the column. Note that if you are using a dynamic value (such as a date), the default will only be calculated the first time (i.e. on the date the migration is applied). Use `nil` for `NULL`.
- `limit` Sets the maximum number of characters for a `string` column and the maximum number of bytes for `text/binary/integer` columns.
- `null` Allows or disallows `NULL` values in the column.
- `precision` Specifies the precision for `decimal/numeric/datetime/time` columns.
- `scale` Specifies the scale for the `decimal` and `numeric` columns, representing the number of digits after the decimal point.

Default and limit are the most used ones.

Precision and scale -> make it loose information on rollback.

Add and remove reference



- Add reference creates an index by default
 - Avoid this with index: false
- To remove a reference, please also specify all the information it had
 - Consider helping a rails rollback

```
remove_reference :products, :user, foreign_key: true, index:  
false
```

Copy

Reference is not the same as foreign key. The reference key is the column that the foreign key from one table is using to connect to the other table.

Adding foreign keys

- Just like in the last command, `add_foreign_key` first mentions the table in which the change will occur
 - In this case, `articles` is referencing to `authors`

```
add_foreign_key :articles, :authors
```

```
add_foreign_key :articles, :authors, column: :reviewer,  
primary_key: :email
```

Removing foreign keys

```
# let Active Record figure out the column name  
remove_foreign_key :accounts, :branches  
  
# remove foreign key for a specific column  
remove_foreign_key :accounts, column: :owner_id
```

Executing a database statement

```
Product.connection.execute("UPDATE products SET price =  
'free' WHERE 1=1")
```

Not a good practice at least it is stated by enterprise rules.

The up and down methods

- Allows you to do reversible migrations efficiently
 - It specifically tells ActiveRecord::Migration what to do with each statement



```
class ExampleMigration < ActiveRecord::Migration[7.0]
  def up
    create_table :distributors do |t|
      t.string :zipcode
    end

    # add a CHECK constraint
    execute <<-SQL
    ALTER TABLE distributors
    ADD CONSTRAINT zipchk
    CHECK (char_length(zipcode) = 5);
    SQL

    add_column :users, :home_page_url, :string
    rename_column :users, :email, :email_address
  end

  def down
    rename_column :users, :email_address, :email
    remove_column :users, :home_page_url

    execute <<-SQL
    ALTER TABLE distributors
    DROP CONSTRAINT zipchk
    SQL

    drop_table :distributors
  end
end
```

Copy

It is not a reversible migration since we used an sql statement.

Reverting a migration

- In another migration

```
require_relative "20121212123456_example_migration"

class FixupExampleMigration < ActiveRecord::Migration[7.0]
  def change
    revert ExampleMigration
    ⚡
    create_table(:apples) do |t|
      t.string :variety
    end
  end
end
```

Copy

Another way to do a reverse, using a relative, if you want to do a reverse on another migration you can use that code.

It depends if it is a bad practice to do a rollback, sometimes it is easier to rollback than creating a whole new migration.

Using explicit SQL statements



```
class DontUseConstraintForZipcodeValidationMigration <
  ActiveRecord::Migration[7.0]
  def change
    revert do
      # copy-pasted code from ExampleMigration
      reversible do |dir|
        dir.up do
          # add a CHECK constraint
          execute <<-SQL
            ALTER TABLE distributors
              ADD CONSTRAINT zipchk
                CHECK (char_length(zipcode) = 5);
          SQL
        end
        dir.down do
          execute <<-SQL
            ALTER TABLE distributors
              DROP CONSTRAINT zipchk
          SQL
        end
      end
    end

    # The rest of the migration was ok
  end
end
```

Using explicit SQL statements

```
class DontUseConstraintForZipcodeValidationMigration <
  ActiveRecord::Migration[7.0]
  def change
    revert do
      # copy-pasted code from ExampleMigration
      reversible do |dir|
        dir.up do
          # add a CHECK constraint
          execute <<-SQL
            ALTER TABLE distributors
              ADD CONSTRAINT zipchk
                CHECK (char_length(zipcode) = 5);
          SQL
        end
        dir.down do
          execute <<-SQL
            ALTER TABLE distributors
              DROP CONSTRAINT zipchk
          SQL
        end
      end
    end

    # The rest of the migration was ok
  end
end
```

With explicit sql statements always try to use up and down.

```
> migrate > 20220830202920_create_reversible_migration.rb
1 class CreateReversibleMigration < ActiveRecord::Migration[6.0]
2   def change
3     revert do
4       reversible do |dir|
5         dir.up do
6           execute <<-SQL
7             ALTER TABLE users
8               ADD CONSTRAINT valid_email
9                 CHECK ( position('@' in email ) > 0 );
10          SQL
11        end
12      end
13      dir.down do
14        execute <<-SQL
15          ALTER TABLE users
16            DROP CONSTRAINT valid_email;
17        SQL
18      end
19    end
20  end
21 end
22 end
```



```

db > migrate > 20220830202920_create_reversible_migration.rb
1  class CreateReversibleMigration < ActiveRecord::Migration[6.0]
2    def change
3      revert do
4        reversible do |dir|
5          dir.up do
6            execute <<-SQL
7              ALTER TABLE users
8                ADD CONSTRAINT valid_email
9                  CHECK ( position('@' in email ) > 0 );
10           SQL
11         end
12
13         dir.down do
14           execute <<-SQL
15             ALTER TABLE users
16               DROP CONSTRAINT valid_email;
17           SQL
18         end
19       end
20     end
21   end
22 end
23
24
25 def change
26
27   revert do
28     reversible do |dir|
29       dir.up do
30         end
31
32       dir.down do
33         end
34     end
35   end
36
37 end
38
39
40 def up
41   execute <<-SQL
42     ALTER TABLE users
43       ADD CONSTRAINT valid_email
44         CHECK ( position('@' in email ) > 0 );
45   SQL
46 end
47
48
49 def down
50   execute <<-SQL
51     ALTER TABLE users
52       DROP CONSTRAINT valid_email;
53   SQL
54 end

```

```

oskarhinojosa@Oskars-MacBook-Pro ecommerce % rails db:rollback STEP=2
warning ../package.json: No license field

```

Step -> it is the number of migrations to revert.

Notes on the CLI



- You can run specific migrations by using the timestamp at the end of the created file
 - Rails `db:migrate VERSION=202208160000`
- You can rollback the last migration or the last three migrations (using three, please note that three can be changed to anything)
 - Rails `db:rollback [STEP=#]`
- You can rollback and migrate again in a simple step
 - Rails `db:migrate:redo [STEP=#]`

Irreversible action

```
> migrate > 20220831165015_edit_computer_columns.rb
1 class EditComputerColumns < ActiveRecord::Migration[6.1]
2   def change
3     rename_column :computers, :storage, :storage_device
4     change_column :computers, :name_of_user, :string, null: false
5     change_column :computers, :storage_device, :string, limit: 5
6   end
7 end
```

Caused by:
ActiveRecord::IrreversibleMigration:

This migration uses change_column, which is not automatically reversible.

To make the migration reversible you can either:

1. Define #up and #down methods in place of the #change method.
2. Use the #reversible method to define reversible behavior.

```
/Users/luisroberto/.rvm/gems/ruby-3.0.4/gems/activerecord-6.1.6.1/lib/active_record/migration/command_recorder.rb:100:in `inverse_of'
/Users/luisroberto/.rvm/gems/ruby-3.0.4/gems/activerecord-6.1.6.1/lib/active_record/migration/command_recorder.rb:80:in `record'
/Users/luisroberto/.rvm/gems/ruby-3.0.4/gems/activerecord-6.1.6.1/lib/active_record/migration/command_recorder.rb:112:in `change_column'
/Users/luisroberto/.rvm/gems/ruby-3.0.4/gems/activerecord-6.1.6.1/lib/active_record/migration.rb:929:in `block in method_missing'
/Users/luisroberto/.rvm/gems/ruby-3.0.4/gems/activerecord-6.1.6.1/lib/active_record/migration.rb:897:in `block in say_with_time'
/Users/luisroberto/.rvm/gems/ruby-3.0.4/gems/activerecord-6.1.6.1/lib/active_record/migration.rb:897:in `say_with_time'
/Users/luisroberto/.rvm/gems/ruby-3.0.4/gems/activerecord-6.1.6.1/lib/active_record/migration.rb:918:in `method_missing'
/Users/luisroberto/Documents/ENROUTE/RoR-Workshop/rails-exercises/migration-exercise/db/migrate/20220831165015_edit_computer_columns.rb:4:in `change'
```

What should have been done

```
class EditComputerColumns < ActiveRecord::Migration[6.1]
  def up
    rename_column :computers, :storage, :storage_device
    change_column :computers, :name_of_user, :string, null: false
    change_column :computers, :storage_device, :string, limit: 5
  end

  def down
    rename_column :computers, :storage_device, :storage
    change_column :computers, :name_of_user, :string, null: true
    change_column :computers, :storage_device, :string, limit: 255
  end
end
```

URLs

<https://millarian.com/rails/precision-and-scale-for-ruby-on-rails-migrations/>

<https://www.gapintelligence.com/blog/up-and-down-a-rails-migration/>

<https://medium.com/@kevinkarma55/rails-active-record-migration-change-method-vs-up-down-method-eaba011de3e9>