第 7 章

チェンジセット

Phoenix ではデータベーステーブルにレコードを挿入したり、既存レコードの値を変更する際に「チェンジセット」と呼ばれる構造体が重要な役割を果たします。本章では、NanoPlanner の開発からいったん離れて、この概念について学習します。

7.1 スキーマ構造体

『Elixir/Phoenix 初級②』第 5 章で、「モデルモジュールで定義された構造体のことをモデル構造体と呼ぶ」と書きました。しかし、Phoenix 1.3 からは「モデル」という用語・概念が使われなくなりましたので、この呼称も変えなければなりません。本巻ではこれを**スキーマ構造体**と呼びます。ただし、これは本書独自の呼び方ではなく、Ecto（Phoenix が使用しているデータベース操作ライブラリ）のドキュメントで使われている正式の用語です。

名前は変わったものの、この構造体の役割が変化したわけではありません。しかし、少し復習をしてから次に進むことにしましょう。

まず、マップ（『初級①』第 14 章）と構造体（『初級②』第 4 章）の関係について記憶を呼び覚ましてください。マップは、他のプログラミング言語で「連想配列」とか「ハッシュ」と呼ばれるデータ型に対応しています。そして、構造体はマップに「あらかじめ列挙されたアトムしかキーとして使えない」という制限を加え、名前を与えたものです。

次の例をご覧ください。

```
m = %{name: "foo", email: "foo@example.com"}
u = %User{name: "foo", email: "foo@example.com"}
```

変数 m にセットされた値はマップであり、変数 u にセットされた値は構造体です。ただし、このコードをコンパイルするためには、あらかじめ次のように User モジュールを定義しておく必要があります。

```
defmodule User do
  defstruct [:name, :email]
end
```

スキーマ構造体は、データベーステーブルのレコードを表現するための構造体です。PlanItem モジュールのソースコードをご覧ください。

lib/nano_planner/schedule/plan_item.ex

```
:
  schema "plan_items" do
    field(:name, :string)
    field(:description, :string)
    field(:starts_at, Timex.Ecto.DateTime)
    field(:ends_at, Timex.Ecto.DateTime)

    timestamps()
  end
:
```

schema ... end の間でスキーマ構造体が定義されています。

次のように書くと、このスキーマ構造体を型とする値が変数 p にセットされます。

```
p = %NanoPlanner.Schedule.PlanItem{name: "Test", description: ""}
```

alias NanoPlanner.Schedule.PlanItem と書いてエイリアスを定義すれば、上記のコードは次のように短縮できます。

```
p = %PlanItem{name: "Test", description: ""}
```

7.2 チェンジセット

この節では「チェンジセット」という概念について学ぶために、短い実験スクリプトを作成して実行し、結果を観察します。

実験スクリプト cast.exs の作成と実行

ディレクトリ priv の下に実験スクリプトを置くためのサブディレクトリ exp を作成してください（“exp”は“experimental”の略）。

```
$ mkdir -p priv/exp
```

そして、priv/exp ディレクトリの下に新規ファイル cast.exs を作成し、次の内容を書き込んでください。

priv/exp/cast.exs (New)

```
1  alias NanoPlanner.Schedule.PlanItem
2
3  item = %PlanItem{}
4  params = %{
5    "name" => "Test",
6    "description" => "Experiment",
7    "starts_at" => "2018-04-01T12:00:00",
8    "ends_at" => "2018-04-01T13:00:00"
9  }
10 fields = [:name, :description, :starts_at, :ends_at]
11 cs = Ecto.Changeset.cast(item, params, fields)
12
13 IO.inspect(Map.keys(cs))
14 IO.inspect(cs.data == item)
15 IO.inspect(cs.params == params)
16 IO.inspect(cs.changes)
```

そして、ターミナルで次のコマンドを実行します。

```
$ mix run priv/exp/cast.exs
```

すると、次のような結果が出力されます。

```
[:__struct__, :action, :changes, :constraints, :data, :empty_values, :errors,
 :filters, :params, :prepare, :repo, :repo_opts, :required, :types, :valid?,
 :validations]
true
true
%{description: "Experiment", ends_at: #DateTime<2018-04-01 13:00:00Z>,
  name: "Test", starts_at: #DateTime<2018-04-01 12:00:00Z>}
```

実験スクリプト cast.exs の解説 (1)

では、実験スクリプトのソースコードと出力結果について細かく見ていきましょう。まず、ソースコードの 1〜3 行目をご覧ください。

```
alias NanoPlanner.Schedule.PlanItem

item = %PlanItem{}
```

モジュール NanoPlanner.Schedule.PlanItem のエイリアスを定義した後、そのエイリアスを用いてスキーマ構造体を作り、それを変数 item にセットしました。

続いて、ソースコードの 4〜9 行目をご覧ください。

```
params = %{
  "name" => "Test",
  "description" => "Experiment",
  "starts_at" => "2018-04-01T12:00:00",
  "ends_at" => "2018-04-01T13:00:00"
}
```

変数 params にマップをセットしています。ブラウザから送信されてきたフォームデータであると考えてください。

ソースコードの 10〜11 行がこの実験スクリプトの肝となる部分です。

```
fields = [:name, :description, :starts_at, :ends_at]
cs = Ecto.Changeset.cast(item, params, fields)
```

変数 fields には 4 個のアトムから構成されるリストをセットしています。そして、すでに値がセットされた変数 item、params、fields を引数として関数 Ecto.Changeset.cast/4 を呼び出しました（第 4 引数は省略可）。この関数からの戻り値は、Ecto.Changeset という名前を持つ構造体です。この構造体のことを「**チェンジセット**（changeset）」と呼びます。

実験スクリプト cast.exs の解説 (2)

実験スクリプト cast.exs の 13 行目をご覧ください。

```
IO.inspect(Map.keys(cs))
```

関数 IO.inspect/2 は、第 1 引数に取った値の中身を読みやすい形式に直して出力します（第 2 引数は省略可）。関数 Map.keys/1 は、引数に取ったマップのすべてのキーのリストを返します。13 行目のコードからは、次のような結果が出力されます。

```
[:__struct__, :action, :changes, :constraints, :data, :empty_values, :errors,
 :filters, :params, :prepare, :repo, :repo_opts, :required, :types, :valid?,
 :validations]
```

構造体はマップの一種であることを思い出してください。チェンジセットと呼ばれる構造体には、action 以下 15 個のフィールドがあります。これらのフィールドのうち、本巻の内容と直接の関連を持つのは、data と params と changes です。

実験スクリプトの 14〜16 行目では、これらのフィールドにどんな値がセットされているかを調べていて、次のような結果が出力されています。

```
true
true
%{description: "Experiment", ends_at: #DateTime<2018-04-01 13:00:00Z>,
  name: "Test", starts_at: #DateTime<2018-04-01 12:00:00Z>}
```

この結果から、基本的な事実として以下の 2 点が観察できます。

1. チェンジセットの data フィールドには、変数 item の値（PlanItem 構造体）がセットされる。
2. チェンジセットの params フィールドには、関数 Ecto.Changeset.cast/4 の第 2 引数として与えたマップ（フォームデータ）がセットされる。

さて、チェンジセットの changes フィールドに関しては、注意深く見る必要があります。一見すると params フィールドの値と同じマップのようですが、そうではありません。第 1 に、キーが異なります。params フィールドのマップではキーがすべて文字列ですが、changes フィールドの場合はアトムです。

第 2 に、マップの値が部分的に異なります。例えば、params フィールドのマップでは "starts_at" キーの値が "2018-04-01T12:00:00" という文字列であるのに対し、changes フィールドのマップでは :starts_at キーに対して DateTime 構造体がセットされています。

7.3 チェンジセットとは何か

関数 Ecto.Changeset.cast/4

実験スクリプト cast.exs の 11 行目を再度ご覧ください。

```
cs = Ecto.Changeset.cast(item, params, fields)
```

関数 Ecto.Changeset.cast/4 は、次に示す 4 個の引数を取り、チェンジセットと呼ばれる構造体を返します。

1. データ（スキーマ構造体）
2. パラメータ（マップ）
3. スキーマ構造体のフィールドの集合（リスト）
4. オプション（キーワードリスト）

> 第 1 引数にはスキーマ構造体の代わりにチェンジセットまたはタプルも指定できますが、本巻ではこの呼び出し方を扱いません。第 4 引数は省略可能です。

関数の名前として使われている“cast”という英語の動詞は、プログラミングの文脈では「〜のデータ型を変換する」という意味を持ちます。変換の対象となるのは、第 2 引数として与えられたマップの値です。

この ケース では "Hello"、"Experiment"、"2018-04-01T12:00:00"、"2018-04-01T13:00:00" という 4 個の文字列です。これらの文字列が、対応するフィールドの型に変換されます。PlanItem 構造体の name フィールドと description フィールドは文字列型なので、そのまま維持されます。他方、starts_at フィールドと ends_at フィールドは日付・時刻型なので、DateTime 構造体へと変換されます。

チェンジセットの役割

この辺りで、本章のテーマ「チェンジセットとは何か」に向き合うことにしましょう。チェンジセットとは、与えられたスキーマ構造体に関して、それをどのように変化させたいかという「意図」を表現する構造体です。

関数 Ecto.Changeset.cast/4 の第 1 引数が、その「与えられたスキーマ構造体」です。これはそのままチェンジセットの data フィールドにセットされます。第 2 引数のマップは、例えばブラウザから送られてきたフォームデータです。これがチェンジセットの params フィールドにセットされます。

フォームデータには余分なキーが含まれている可能性があるので、選別が必要です。そのために関数 Ecto.Changeset.cast/4 は、第 3 引数にフィールド名のリストを取ります。また、フォームデータの値は文字列なので、一般的にはデータ型の変換が必要となります。この型変換と選別が行われた結果が、チェンジセットの changes フィールドの値になります。

実験スクリプト cast.exs からの出力結果の末尾は、次のようになっていました。

```
%{description: "Experiment", ends_at: #DateTime<2018-04-01 13:00:00Z>,
  name: "Test", starts_at: #DateTime<2018-04-01 12:00:00Z>}
```

すなわち、変数 item の description フィールドの値を "Experiment" に、ends_at フィールドの値を「2018 年 4 月 1 日 13 時」に、name フィールドの値を "Test" に、starts_at フィールドの値を「2018 年 4 月 1 日 12 時」に変化させたいという意図がここに表現されているわけです。

注意していただきたいのは、この時点では「変化の意図」が表現されているだけだ、ということです。この意図を実現する方法、つまりデータベーステーブルにレコードを挿入したり、既存のレコードを更新したりする方法については、後述します。

フォームデータの選別

関数 Ecto.Changeset.cast/4 の振る舞いをさらに調べるため、実験スクリプト cast.exs を次のように書き換えてください。

priv/exp/cast.exs

```
  :
10 - fields = [:name, :description, :starts_at, :ends_at]
10 + fields = [:name, :description]
  :
```

そして、ターミナルで次のコマンドを実行します。

```
$ mix run priv/exp/cast.exs
```

すると、次のように出力結果が変化します。

```
...
true
true
%{description: "Experiment", name: "Test"}
```

このことから、第 3 引数のリストに含まれないフィールドはフォームデータから除去されることがわかります。

では、実験スクリプト cast.exs をさらに次のように書き換えてください。

priv/exp/cast.exs

```
  :
10 - fields = [:name, :description]
10 + fields = [:name, :description, :foo]
  :
```

そして、mix run priv/exp/cast.exs コマンドを実行すると、次のようなエ

ラーメッセージがターミナルに出力されます。

```
** (ArgumentError) unknown field `foo`. Only fields, embeds and >
associations (except :through ones) are supported in changesets
    (ecto) lib/ecto/changeset.ex:489: Ecto.Changeset.type!/2
…
```

スキーマ構造体で定義されていないフィールド名が第 3 引数のリストに含まれていると、例外 ArgumentError が発生します。例外の発生を確認したら、コードを元に戻してから次に進んでください。

関数 Ecto.Changeset.cast/4 がフォームデータを選別する仕組みは、Ruby on Rails のストロング・パラメータ（strong parameters）によく似ています。

7.4 チェンジセットを用いたレコードの挿入と更新

チェンジセットは主にふたつの用途で使われます。ひとつは、HTML フォームの生成です。もうひとつは、データベーステーブルの変更（レコードの挿入、更新）です。前者については次章以降でじっくりと解説します。この節では後者の用途について基本を学びます。

レコードの挿入

チェンジセットを利用してデータベーステーブルにレコードを挿入する方法を学ぶため、この章の前半で作成した実験スクリプト priv/exp/cast.exs をコピーして create.exs を作成します。

```
$ cp priv/exp/cast.exs priv/exp/create.exs
```

そして、create.exs を次のように書き換えます。

priv/exp/create.exs

```
1   alias NanoPlanner.Schedule.PlanItem
2 + alias NanoPlanner.Repo
```

```
  3
  :
```

同ファイルをさらに次のように書き換えます。

priv/exp/create.exs

```
  :
 11   fields = [:name, :description, :starts_at, :ends_at]
 12   cs = Ecto.Changeset.cast(item, params, fields)
 13
 14 - IO.inspect(Map.keys(cs))
 15 - IO.inspect(cs.data == item)
 16 - IO.inspect(cs.params == params)
 17 - IO.inspect(cs.changes)
 14 + Repo.insert!(cs)
```

そして、ターミナルで次のコマンドを実行します。

```
$ mix run priv/exp/create.exs
```

すると、次のような結果が出力されます。

```
[debug] QUERY OK db=10.5ms
INSERT INTO "plan_items" ("description","ends_at","name","starts_at"," >
inserted_at","updated_at") VALUES ($1,$2,$3,$4,$5,$6) RETURNING "id" >
["Experiment", {{2018, 4, 1}, {13, 0, 0, 0}}, "Test", {{2018, 4, 1}, >
{12, 0, 0, 0}}, {{2017, 12, 21}, {15, 33, 54, 0}}, {{2017, 12, 21}, >
{15, 33, 54, 0}}]
```

SQL の INSERT 文が発行されていることがわかります。

create.exs の 14 行目をご覧ください。

```
Repo.insert!(cs)
```

私たちはすでにシードデータ投入スクリプト（priv/repo/seeds.ex）の中で関数 NanoPlanner.Repo.insert!/2 を使用しています。しかし、そこでは引数にスキーマ構造体を指定してこの関数を呼び出しています。実は、この関数はチェンジセットも第 1 引数に取ることができます。その場合、チェンジセットの data

フィールドのスキーマ構造体と changes フィールドのマップをマージし、その結果を用いてデータベーステーブルへのレコード挿入を行います。

レコードの更新

次に、既存のレコードを更新する例を見てみましょう。

priv/exp ディレクトリの下に新規ファイル update.exs を作成し、次の内容を書き込んでください。

priv/exp/update.exs (New)

```
1  alias NanoPlanner.Schedule.PlanItem
2  alias NanoPlanner.Repo
3
4  item = PlanItem |> Ecto.Query.first(:id) |> Repo.one()
5  params = %{
6    "name" => "Foo",
7    "description" => "Bar"
8  }
9  fields = [:name, :description]
10 cs = Ecto.Changeset.cast(item, params, fields)
11
12 Repo.update!(cs)
```

そして、ターミナルで次のコマンドを実行します。

```
$ mix run priv/exp/update.exs
```

すると、次のような結果が出力されます。

```
[debug] QUERY OK source="plan_items" db=1.6ms decode=4.4ms
SELECT p0."id", p0."name", p0."description", p0."starts_at", p0."ends_at", >
p0."inserted_at", p0."updated_at" FROM "plan_items" AS p0 ORDER BY >
p0."id" LIMIT 1 []
[debug] QUERY OK db=3.8ms
UPDATE "plan_items" SET "description" = $1, "name" = $2, "updated_at" = $3 >
WHERE "id" = $4 ["Bar", "Foo", {{2017, 12, 21}, {15, 55, 46, 0}}, 1]
```

SQL の UPDATE 文が発行されていることがわかります。

update.exs の 4 行目をご覧ください。

```
item = PlanItem |> Ecto.Query.first(:id) |> Repo.one()
```

plan_items テーブルに最初に挿入されたレコードを表現するスキーマ構造体を作り、変数 item にセットしています。

この式の意味がわからなくなってしまった方は、『初級②』第 15 章を復習してください。

次に、update.exs の 5〜10 行目をご覧ください。

```
params = %{
  "name" => "Foo",
  "description" => "Bar"
}
fields = [:name, :description]
cs = Ecto.Changeset.cast(item, params, fields)
```

変数 item にセットされたスキーマ構造体の name フィールドと description フィールドを変化させるためのチェンジセットを作って、変数 cs にセットしています。

最後に、update.exs の 12 行目をご覧ください。

```
Repo.update!(cs)
```

チェンジセット cs を用いて plan_items テーブルのあるレコードを更新しています。関数 NanoPlanner.Repo.insert!/2 の場合と異なり、関数 NanoPlanner.Repo.update!/2 の第 1 引数には必ずチェンジセットを指定する必要があります。この関数はチェンジセットの changes フィールドを用いて SQL の UPDATE 文を組み立てます。