Intro into

Ecto

Agenda

- What is ecto
- Basic usage
- Protips

Agenda

- What is ecto
- Basic usage
- Protips

Ecto **is a domain specific language for writing queries and interacting with databases in Elixir.**

Supports various databases

- PostgreSQL
- MySQL
- MSSQL
- SQLite3
- MongoDB

Ecto's guts

- Repo
- Query
- Migrations
- Changesets
- Adapters

Agenda

- What is ecto
- **Basic usage**
- Protips

Repo

- Defines a repository used by the app.

- Connects the adapter + config for the database.

- Interface for managing data in the database that delegates to `Repo.Queryable`, `Repo.Query`, and a few others.

Repo setup

```
config :lost_legends, LostLegends.Repo,
  adapter: Ecto.Adapters.Postgres,
  url: "postgres://garrett:garrett@localhost/lost_legends_dev",
  pool_size: 10

defmodule LostLegends.Repo do
  use Ecto.Repo, otp_app: :lost_legends
end
```

What do we get?

```
mix ecto.create             # Create the storage for the repo
mix ecto.drop               # Drop the storage for the repo
mix ecto.gen.migration      # Generate a new migration for the repo
mix ecto.gen.repo           # Generate a new repository
mix ecto.migrate            # Run migrations up on a repo
mix ecto.rollback           # Rollback migrations from a repo
```

Migrations

```
$ mix ecto.gen.migration create_monsters
* creating priv/repo/migrations/20160112072054_create_monsters.exs

defmodule LostLegends.Repo.Migrations.CreateMonsters do
  use Ecto.Migration

  def change do
    create table(:monsters) do
      add :name, :string
      add :desc, :text
      add :health, :integer

      timestamps
    end
  end
end
```

Schema

```elixir
defmodule LostLegends.Monster do
  use Ecto.Schema

  schema "monsters" do
    field :name, :string
    field :desc, :string
    field :health, :integer

    timestamps
  end
end
```

Lets use this sucker

```
iex> alias LostLegends.{Repo, Monster}
iex> Repo.all Monster
[debug] SELECT m0."id", m0."name", m0."desc", m0."health", m0."inserted_at", m0."updated_at"
        FROM "monsters" AS m0 [] OK query=2.7ms
[
  %LostLegends.Monster{__meta__: #Ecto.Schema.Metadata<:loaded>,
    desc: "Drugs are bad mkay.", health: 10, id: 1,
    inserted_at: #Ecto.DateTime<2016-01-07T11:02:43Z>, name: "Crackie Monster",
    updated_at: #Ecto.DateTime<2016-01-07T11:20:54Z>},

   %LostLegends.Monster{__meta__: #Ecto.Schema.Metadata<:loaded>,
    desc: "Today totally isn't Sunday, trust me.", health: 3000, id: 2,
    inserted_at: #Ecto.DateTime<2016-01-07T11:02:43Z>, name: "That 12th beer",
    updated_at: #Ecto.DateTime<2016-01-07T11:02:43Z>}
]
```

Such Data

```
Repo.all(Model)
```

- Repo is the interface for the database
- No magic functions on Model

Repo Interface

```
Repo.all
Repo.one
Repo.get
Repo.insert
Repo.update(_all)
Repo.delete(_all)
Repo.transaction
Repo.preload
# and more..!
```

Query

The thing **Repo** takes

Query

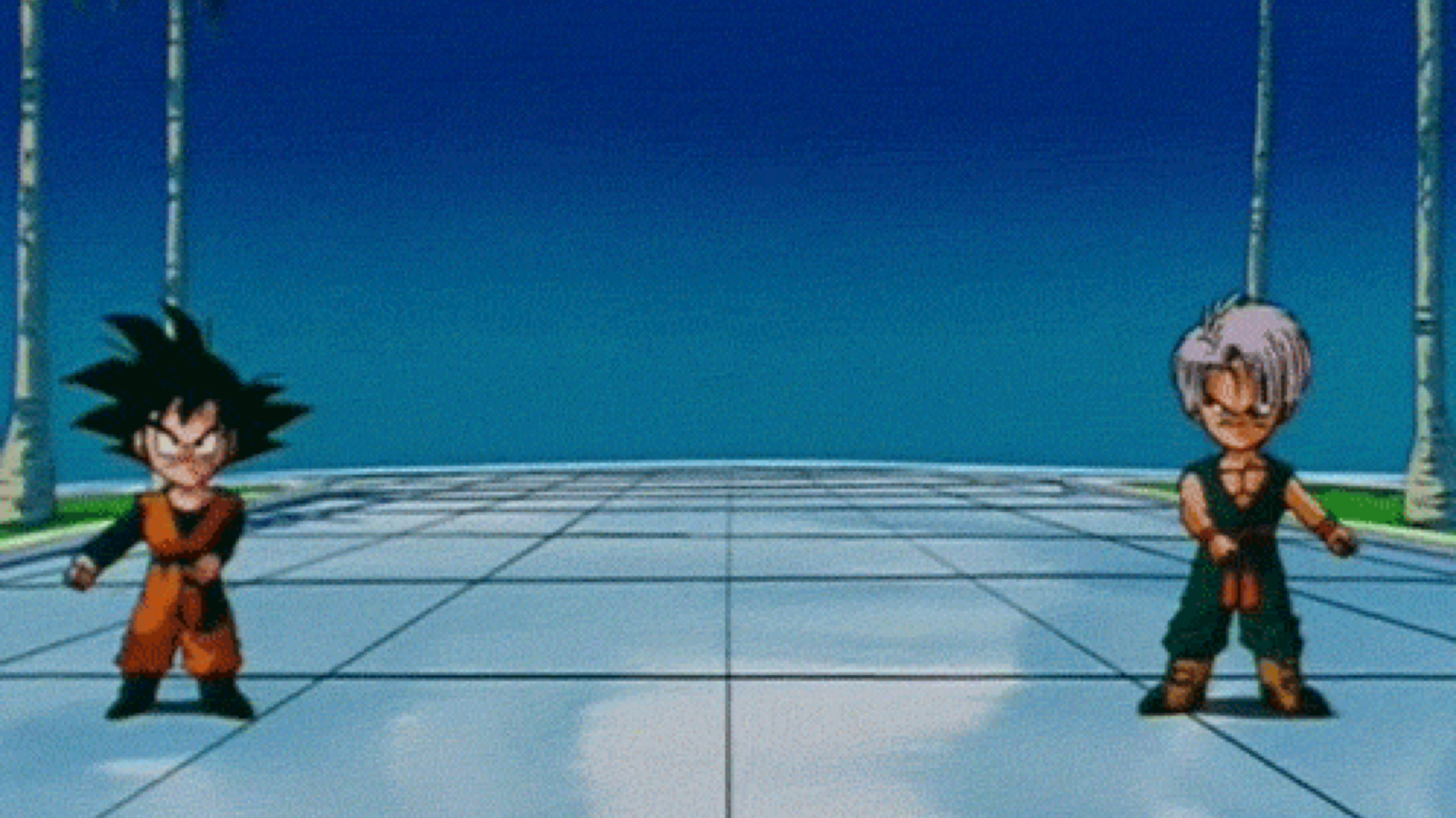
```
import Ecto.Query, only: [from: 2]

query = from m in Monster, where: m.health > 10

Repo.all(query)
```

Composition

When two become one



Composition

```
import Ecto.Query, only: [from: 2]

such_health_query = from m in Monster, where: m.health > 10

recent_ms_query = from m in such_health_query, # notice me
                  where: m.inserted_at > datetime_add(^Ecto.DateTime.utc, -1, "month")

Repo.all(recent_ms_query)
```

Real example

```elixir
def required_assignments(date, remote_id) do
  filter_by_status(required_status)
  |> for_student(remote_id)
  |> current(date)
  |> Dogfood.Repo.all
end

defp required_status, do: "required"
defp filter_by_status(status) do
  from assignment in Dogfood.Assignment,
    where: assignment.status == ^status
end

defp for_student(query, remote_id) do
  from assignment in query,
    where: assignment.student_id == ^remote_id # RE: student_id == MS: remote_id
end

defp current(query, %Ecto.DateTime{} = date) do
  from assignment in query,
    where: assignment.start_date <= ^date and ^date < assignment.end_date
end
```

Preloading Records

Preventing your looping problems since 2013

Preloading Records

```
iex> battle = Repo.get Battle, 1
[debug] SELECT bo."id", bo."monster_id", bo."inserted_at", bo."updated_at"
        FROM "battles" AS bo WHERE (bo."id" = $1) [1] OK query=0.7ms
%LostLegends.Battle{__meta__: #Ecto.Schema.Metadata<:loaded>, id: 1,
 inserted_at: #Ecto.DateTime<2016-01-07T11:02:43Z>,
 monster: #Ecto.Association.NotLoaded<association :monster is not loaded>,
 monster_id: 1, updated_at: #Ecto.DateTime<2016-01-07T11:02:43Z>}
iex> battle.monster
#Ecto.Association.NotLoaded<association :monster is not loaded>
```

Preloading Records

```
iex> battle = Repo.get(Battle, 1) |> Repo.preload(:monster)

[debug] SELECT b0."id", b0."monster_id", b0."inserted_at", b0."updated_at"
        FROM "battles" AS b0 WHERE (b0."id" = $1) [1] OK query=0.7ms
[debug] SELECT m0."id", m0."name", m0."desc", m0."health", m0."inserted_at", m0."updated_at"
        FROM "monsters" AS m0 WHERE (m0."id" IN ($1)) [1] OK query=0.5ms
%LostLegends.Battle{__meta__: #Ecto.Schema.Metadata<:loaded>, id: 1,
 inserted_at: #Ecto.DateTime<2016-01-07T11:02:43Z>,
 monster: %LostLegends.Monster{__meta__: #Ecto.Schema.Metadata<:loaded>,
  desc: "Drugs are bad mkay.", health: -9, id: 1,
  inserted_at: #Ecto.DateTime<2016-01-07T11:02:43Z>, name: "Crackie Monster",
  updated_at: #Ecto.DateTime<2016-01-07T11:20:54Z>}, monster_id: 1,
 updated_at: #Ecto.DateTime<2016-01-07T11:02:43Z>}

iex> battle.monster
%LostLegends.Monster{__meta__: #Ecto.Schema.Metadata<:loaded>,
 desc: "Drugs are bad mkay.", health: 10, id: 1,
 inserted_at: #Ecto.DateTime<2016-01-07T11:02:43Z>, name: "Crackie Monster",
 updated_at: #Ecto.DateTime<2016-01-07T11:20:54Z>}
```

```
iex> battle = Repo.get(Battle, 1) |> Repo.preload(monster: :battles)

[debug] SELECT bo."id", bo."monster_id", bo."inserted_at", bo."updated_at"
        FROM "battles" AS bo WHERE (bo."id" = $1) [1] OK query=0.6ms
[debug] SELECT mo."id", mo."name", mo."desc", mo."health", mo."inserted_at", mo."updated_at"
        FROM "monsters" AS mo WHERE (mo."id" IN ($1)) [1] OK query=0.3ms queue=0.1ms
[debug] SELECT bo."id", bo."monster_id", bo."inserted_at", bo."updated_at"
        FROM "battles" AS bo WHERE (bo."monster_id" IN ($1)) ORDER BY bo."monster_id" [1] OK query=0.4ms
%LostLegends.Battle{__meta__: #Ecto.Schema.Metadata<:loaded>, id: 1,
 monster: %LostLegends.Monster{__meta__: #Ecto.Schema.Metadata<:loaded>,
  battles: [%LostLegends.Battle{__meta__: #Ecto.Schema.Metadata<:loaded>, id: 1,
    monster: #Ecto.Association.NotLoaded<association :monster is not loaded>,
    monster_id: 1, updated_at: #Ecto.DateTime<2016-01-07T11:02:43Z>},
   %LostLegends.Battle{__meta__: #Ecto.Schema.Metadata<:loaded>, id: 2,
    monster: #Ecto.Association.NotLoaded<association :monster is not loaded>,
    monster_id: 1, updated_at: #Ecto.DateTime<2016-01-07T11:02:43Z>}],
  desc: "Drugs are bad mkay.", health: -9, id: 1,
  inserted_at: #Ecto.DateTime<2016-01-07T11:02:43Z>, name: "Crackie Monster",
  updated_at: #Ecto.DateTime<2016-01-07T11:20:54Z>}, monster_id: 1,
 updated_at: #Ecto.DateTime<2016-01-07T11:02:43Z>}

iex> battle.monster.battles
[%LostLegends.Battle{__meta__: #Ecto.Schema.Metadata<:loaded>, id: 1,
  monster: #Ecto.Association.NotLoaded<association :monster is not loaded>,
  monster_id: 1, updated_at: #Ecto.DateTime<2016-01-07T11:02:43Z>},
 %LostLegends.Battle{__meta__: #Ecto.Schema.Metadata<:loaded>, id: 2,
  monster: #Ecto.Association.NotLoaded<association :monster is not loaded>,
  monster_id: 1, updated_at: #Ecto.DateTime<2016-01-07T11:02:43Z>}]
```

Agenda

- What is ecto
- Basic usage
- Protips

Protips

Fragments

```
def unpublished_by_title(title) do
  from p in Post,
    where: is_nil(p.published_at) and
           fragment("downcase(?)", p.title) == ^title
end
```

Protips

Piping

```
# no worky
from u in User, where: u.admin == true
|> Repo.all

# worky
query = from u in User, where: u.admin == true
query |> Repo.all
```

Protips

no HABTM, but HMT works fine

```
schema "recurrings" do
  field :frequency, :string
  has_many :recurring_tags, BudgetApi.RecurringTag
  has_many :tags, through: [:recurring_tags, :tag]
end

schema "transactions" do
  field :amount, :float, default: 0.0
  has_many :transaction_tags, BudgetApi.TransactionTag
  has_many :tags, through: [:transaction_tags, :tag]
end

schema "recurrings_tags" do
  belongs_to :recurring, BudgetApi.Recurring, references: :id
  belongs_to :tag, BudgetApi.Tag, references: :id
end

schema "transactions_tags" do
  belongs_to :transaction, BudgetApi.Transaction, references: :id
  belongs_to :tag, BudgetApi.Tag, references: :id
end

schema "tags" do
  field :tag, :string
  has_many :recurring_tags, BudgetApi.RecurringTag
  has_many :recurrings, through: [:recurring_tags, :recurring]
  has_many :transaction_tags, BudgetApi.TransactionTag
  has_many :transactions, through: [:transaction_tags, :transaction]
end
```

Protips

"`Repo.first` and `Repo.last` have been historically confusing on Rails because many databases do not give a guarantee of ordering. [...]

That's why Ecto provides `Repo.one` that does not give any idea of ordering."
— José Valim

```
def first(query, opts \\ []) do
  Repo.one(from(q in query, order_by: [asc: q.id], limit: 1), opts)
end

def last(query, opts \\ []) do
  Repo.one(from(q in query, order_by: [desc: q.id], limit: 1), opts)
end
```

Protips

jsonb columns

```
create table(:assignments) do
  add :assignment_details, :map
end
schema "assignments" do
  field :assignment_details, :map
end

iex> {:ok, assignment} = Repo.insert(%Assignment{assignment_details: %{awesome: "elixir"}})
iex> assignment.assignment_details["awesome"] # "elixir"
```

Thanks

@gogogarrett