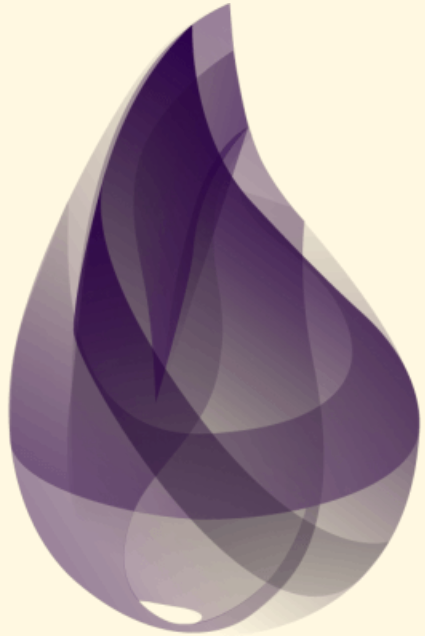
ULID + Ecto

Presented by @andreicek

0x7f



About me

- JavaScript dev for 6 yrs
- Python dev for 2 yrs
- Elixir freelancer for 1 yr
  - "Why was I writing so much code?!" ~ me, circa 2021.

The Plan

1. Remembering UUID
2. What’s ULID
3. Integration
4. Use cases
5. Alternatives

So what's UUID?

- Universally Unique IDentifier (UUID) URN Namespace RFC 4122
- Mostly unique identifiers
- Globally accepted solution for database IDs
- `4a823cf9-b89b-40bb-b823-1d98151fee02`

And what's wrong with that?

```elixir
users = for username ← ~w(andrei dino vlado) do
  %{
    id: Ecto.UUID.bingenerate(),
    username: username
  }
end

# [
#  %{id: "51aa4d08-2962-428c-a677-ee8c3900463c", username: "andrei"},
#  %{id: "4f54731e-0d40-4ab5-9a03-2f4d0fbc44d7", username: "dino"},
#  %{id: "f5721fd7-79bf-40cc-815c-09ce4de2a03b", username: "vlado"}
# ]
```

And what's wrong with that?

```
SELECT username FROM users SORT BY username;

-- vlado, andrei, dino
```

- UUIDs are **not** lexicographically sortable.

Collisions who?

```elixir
# lib/ecto/uuid.ex:180
def bingenerate() do
  <<u0::48, _::4, u1::12, _::2, u2::62>> = :crypto.strong_rand_bytes(16)
  <<u0::48, 4::4, u1::12, 2::2, u2::62>>
end

# lib/ecto/ulid.ex:79
def bingenerate(timestamp \\ System.system_time(:millisecond)) do
  <<timestamp::unsigned-size(48), :crypto.strong_rand_bytes(10)::binary>>
end
```

TBH UUIDs won't really colide all that often :D

OK, sold. Let's integrate.

```
defmodule Migrations.CreateNotes do
  use Ecto.Migration

  def change do
    create table("notes", primary_key: false) do
      add(:id, :binary_id, primary_key: true)
      add(:text, :text)

      timestamps()
    end
  end
end

Ecto.Migrator.up(Repo, 1_653_764_519, Migrations.CreateNotes)
```

OK, sold. Let's integrate.

```
defmodule Notes do
  use Ecto.Schema

  @primary_key {:id, Ecto.ULID, autogenerate: true}

  schema "notes" do
    field(:text, :string)

    timestamps()
  end
end
```

OK, sold. Let's integrate.

```
%Notes{
  text: "My first note."
}
▷ Repo.insert!()

# %Notes{
#   __meta__: #Ecto.Schema.Metadata<:loaded, "notes">,
#   id: "01G619AWHETJVZWFJV5YSXZW8Y",
#   inserted_at: ~N[2022-06-20 19:27:21],
#   text: "My first note.",
#   updated_at: ~N[2022-06-20 19:27:21]
# }
```

Your DB won't even notice

```
ulid = Ecto.ULID.bingenerate()
uuid = Ecto.UUID.cast!(ulid)

# fd1807b2-7eaf-441e-bc20-23c80ec0bf2f
```

With just a bit more code we could easly render ULIDs as UUIDs in our app.

Back to our exapmle

```elixir
users = for username ← ~w(andrei dino vlado) do
  %{
    id: Ecto.ULID.bingenerate(),
    username: username
  }
end

# [
#  %{id: "01G619MYE7BK12WPPN5Y1B4GEH", username: "andrei"},
#  %{id: "01G619MYE71VFEDR6TYVMFS3AM", username: "dino"},
#  %{id: "01G619MYE7QDY2RFDR60HDGM54", username: "vlado"}
# ]
```

Back to our example

```
SELECT username FROM users SORT BY username;

-- andrei, dino, vlado
```

Use cases

- If you're partitioning your database by date, you can use the timestamp embedded in the ULID to select the correct partition.

```
date
▷ DateTime.new!(~T[00:00:00])
▷ DateTime.to_unix(:millisecond)
▷ Ecto.ULID.bingenerate()
```

- You can sort by ULID instead of a separate created_at column if millisecond precision is acceptable.
  Esp. useful if creating an index on created_at is no longer possible (a LOT of records)

Downsides

- If exposing the timestamp is a bad idea for your application, ULIDs may not be the best option.
- The sort by ULID approach may not work if you need sub-millisecond accuracy.
- Not backwards compatible with UUID (you can’t compare them!)

Alternatives

- SnowflakeID
  - Simmilar aproach but it also includes a device identificator (MAC address e.g.) to even more reduce collisions
  - Not 100% sortable, but good enough
  - Defunct :-(
- ???

The end

```
{:ecto_ulid, "~> 0.3.0"}
```

Let's connect

- andrei@0x7f.dev
- 0x7f.dev
- @floppy__h_podge

0x7f