A (more) declarative approach to accessing collections in Elixir

Mateusz Front

software mansion

Membrane

```elixir
users_list = [
  %User{id: 1, nick: "Bob", age: 20},
  %User{id: 2, nick: "Eve", age: 30},
  ...
]
```

```
users = Map.new(users_list, fn user -> {user.id, user} end)

%{
  1 => %User{id: 1, nick: "Bob", age: 20},
  2 => %User{id: 2, nick: "Eve", age: 30},
  ...
}
```

```
Map.get(users, 1)

%{id: 1, nick: "Bob", age: 20}
```

Will this work?

```elixir
bob = Enum.find(users, fn user -> user.nick == "Bob" end)
```

Nope. This will:

```elixir
{_id, bob} = Enum.find(users, fn {_id, user} -> user.nick == "Bob" end)
```

Speeding it up

```elixir
user_nick_to_id = Map.new(users, fn {id, user} -> {user.nick, id} end)

bob_id = Map.get(user_nick_to_id, "Bob")

bob = Map.get(users, bob_id)
```

Speeding it up

```elixir
user_nick_to_id = Map.new(users, fn {id, user} -> {user.nick, id} end)

bob_id = Map.get(user_nick_to_id, "Bob")

bob = Map.get(users, bob_id)

Map.delete(users, bob_id)

Map.delete(user_nick_to_id, "Bob")
```

Idx

```elixir
Mix.install(idx: "~> 0.1.0")

users = Idx.new(users_list, fn user -> user.id end)

Idx.get(users, 1)

%{id: 1, nick: "Bob", age: 20}
```

Idx

```elixir
bob = Enum.find(users, fn user -> user.nick == "Bob" end)

%{id: 1, nick: "Bob", age: 20}
```

Speeding it up

```elixir
users = Idx.new(users_list, fn user -> user.id end)

users = Idx.create_index(users, :nick, fn user -> user.nick end)

bob = Idx.get(users, 1)

%{id: 1, nick: "Bob", age: 20}

bob = Idx.get(users, Idx.key(:nick, "Bob"))

%{id: 1, nick: "Bob", age: 20}
```

Which is better?

```elixir
bob = Enum.find(users, fn user -> user.nick == "Bob" end)
```

vs

```elixir
users = Idx.create_index(users, :nick, fn user -> user.nick end)

bob = Idx.get(users, Idx.key(:nick, "Bob"))
```

Which is better?

```elixir
bob = Enum.find(users, fn user -> user.nick == "Bob" end)
```

vs

```elixir
users = Idx.create_index(users, :nick, fn user -> user.nick end, lazy?: true)

bob = Idx.get(users, Idx.key(:nick, "Bob"))
```

Grouping our users

Grouping our users

```
users_list = [
  %User{id: 1, nick: "Bob", age: 20, team: :a},
  %User{id: 2, nick: "Eve", age: 30}, team: :b}
  ...
]

teams_list = {
  %Team{id: :a, metadata: "...", users: [1, 5, 30, ...]},
  %Team{id: :b, metadata: "...", users: [2, 8, 14, ...]},
}
```

Grouping our users

```elixir
users_list = [
  %User{id: 1, nick: "Bob", age: 20, team: :a},
  %User{id: 2, nick: "Eve", age: 30, team: :b}
  ...
]

teams_list = {
  %Team{id: :a, metadata: "..."},
  %Team{id: :b, metadata: "..."},
}
```

Multi index

```elixir
users = Idx.new(users_list, fn user -> user.id end)

users = Idx.create_index(users, :team, fn user -> user.team end, multi?: true)

users_from_team_a = Idx.get(users, Idx.key(:team, :a))

[%User{id: 1, nick: "Bob", age: 20, team: :a}, %User{id: 5, nick: ...}, ...]
```

Removing a team

```
teams = Idx.delete(teams, :a)

users = Idx.delete(users, Idx.key(:team, :a))
```

Removing a team

```
teams = Idx.delete(teams, :a)

# users = Idx.delete(users, Idx.key(:team, :a))
```

```elixir
users = Idx.create_index(users, :team, fn user -> user.team end, multi?: true)

users_from_team_a = Idx.get(users, Idx.key(:team, :a))
```

```elixir
users = Idx.create_index(users, :team, fn user -> user.team end, multi?: true)

users_from_team_a = Idx.get(users, Idx.key(:team, :a))
```

```sql
CREATE INDEX team ON Users (Team);

SELECT * FROM Users WHERE Team="a"
```

Questions?

github.com/mat-hek/idx

hexdocs.pm/idx