Polymorphic Variants in OCaml 

FPSyd Meetup

Sydney, April 22nd 2020

fp-syd

Carlos D.

Introduction

About me

I'm Carlos.

I like FP.

This is my lightning talk on Polymorphic Variants.

What is a Polymorphic Variant?

🛂 *Typical* Variant

```
type auth =
  | NoAuth
  | Jwt of string
  | UserInfo of string * string

let auth_debug = function
    | NoAuth  -> print_endline @@ "No auth"
    | Jwt str -> print_endline @@ "Jwt auth: " ^ str
    | UserInfo (user, _pwd) ->
        print_endline @@ "UserAuth user: " ^ user
```

```
val auth_debug : auth -> unit = <fun>
```

Here `auth` is what we call *refined*.

What is a Polymorphic Variant?

🔃 Polymorphic Variants

```
let auth_debug = function
    | `NoAuth  -> print_endline @@ "No auth"
    | `Jwt str -> print_endline @@ "Jwt auth: " ^ str
    | `UserInfo (user, _pwd) ->
        print_endline @@ "Saw user auth: " ^ user
```

```
val auth_debug :
  [< `Jwt of string | `NoAuth | `UserInfo of string * 'a ] -> unit = <fun>
```

Notice:

- The backtick in the name.
- No type defined beforehand.
- The type signature may still be refined.

What is a Polymorphic Variant?

Note that if we wanted, we could still assign polymorphic variants to a type alias:

```
type auth =
  [
  | `NoAuth
  | `Jwt of string
  | `UserInfo of string * string
  ]

let auth_debug = function
    | `NoAuth  -> print_endline @@ "No auth"
    | `Jwt str -> print_endline @@ "Jwt auth: " ^ str
    | `UserInfo (user, _pwd) ->
        print_endline @@ "Saw user auth: " ^ user
```

It would be similar to a *typical* definition.

What is a Polymorphic Variant?

We can *coerce* (`:>`) polymorphic variants, provided some conditions are met:

```
type jwt_auth = [ `Jwt of string ]

let debug_jwt : jwt_auth -> unit = function
  | `Jwt str as jwt ->
    print_endline str;
    auth_debug (jwt :> auth)
```

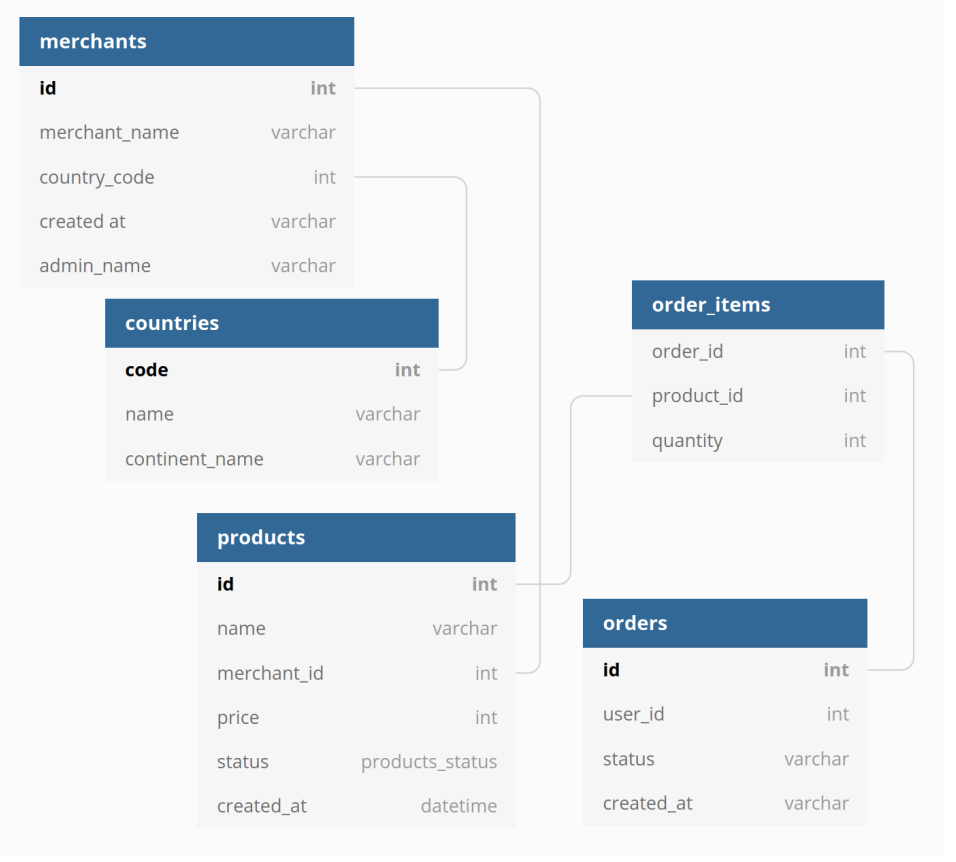
The compiler can tell if the coercion is valid.

Example

http://postgrest.org/

PostgREST turns your DB into a REST API

Turns this



PostgREST turns your DB into a REST API

Into this

Create

```
POST /merchants
```

Read

```
GET /merchants?id=eq.5
```

Update

```
PATCH /merchants?id=eq.5
```

Delete

```
DELETE /merchants?id=eq.5
```

PostgREST turns your DB into a REST API

But more interestingly

Querying

```
GET /merchant?id=lt.10
```

```
GET /merchant?id=in.(1, 2, 3)
```

```
GET /merchant?and=(id.gte.5, id.lte.10)
```

```
GET /merchant?select=merchant_name
```

Many more operators are available:

1. Numeric: `eq`, `gt`, `gte`, `lt`, `lte`, `neq`, `in`
2. Strings: `like`, `ilike`
3. Numeric or Strings: `in`
4. Many more, such as full text search

Demo

I want to generate these queries

So let's look at some of my code

https://github.com/carlosdagos/ocaml-postgrest

Reading Material

- *Ocaml Manual* - Polymorphic Variants
- *Ocaml Manual* - Inheritance and Coercions
- *Real World OCaml* - Polymorphic Variants
- *PostgREST documentation*

Advanced Material

- *OCaml Manual* - Extensible Variants