Generating Data Models with Scalameta

Daniel Oakey

Me

Software engineer in Quantexa's core data team

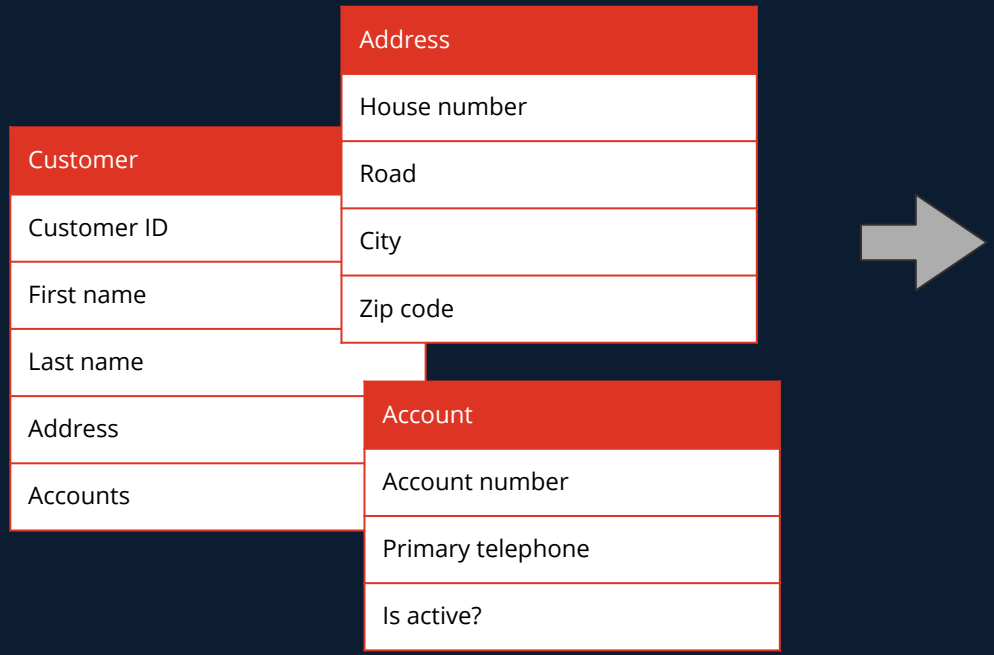
danieloakey@quantexa.com

We're hiring - get in touch!

quantexa

Problem

Users with little or no programming knowledge (but lots of domain knowledge) need to build data models.



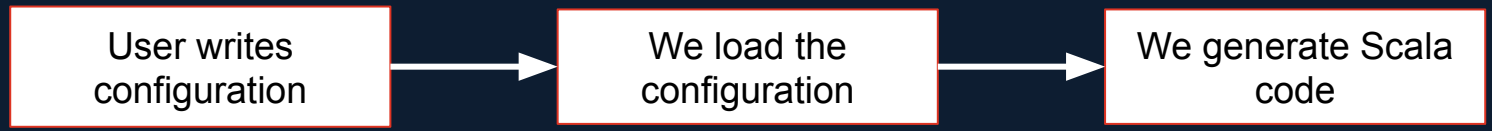

```
case class Customer(
    customerId: Int,
    firstName: Option[String],
    lastName: Option[String],
    address: Address,
    accounts: List[Account]
)
case class Address(
    houseNumber: Option[String],
    road: Option[String],
    city: Option[String],
    zipCode: Option[String]
)
case class Account(
    accountNum: Int,
    primaryTelephone: Option[String],
    isActive: Boolean
)
```

Solution

- Configuration-based models
- User doesn’t need to know Scala
- We can change the model’s representation without the user caring

```
{
    "name": "Customer",
    "fields": [
        {"name": "customerId", "type": "number", "isOptional": false},
        {"name": "fullName", "type": "string", "isOptional": true}
    ]
}
```

Solution


User writes configuration
We load the configuration
We generate Scala code

Configuration

Requirements:

- A model has a name and a list of fields
- A field has a name, a type and a flag indicating whether optional
- A field's type is string, boolean or number

```
{
    "name": "Customer",
    "fields": [
        {"name": "customerId", "type": "number", "isOptional": false},
        {"name": "fullName", "type": "string", "isOptional": true}
    ]
}
```

Configuration

Requirements:

- A model has a name and a list of fields
- A field has a name, a type and a flag indicating whether optional
- A field's type is string, boolean or number

```
case class Model(name: String, fields: List[Field])
case class Field(name: String, tpe: FieldType, isOptional: Boolean)

sealed trait FieldType
case object TString extends FieldType
case object TNumber extends FieldType
case object TBoolean extends FieldType
```

How do we go from data to code?

- Naïve approach: just write out strings
- Complects data with presentation
- Not syntax checked

```
def convertModel(model: Model): String = {
  val fields = model.fields.map(convertField)
  s"case class ${model.name}($fields"
}

def convertField(field: Field): String = // ...
```

```
case class Customer(name: String, favoriteColor: String
```

```
')' expected but '}' found. 😢
```

Introducing Scalameta

- meta
- meta = self-referential
- meta-programming = code that describes code
- Scalameta = using Scala code to write more Scala code 😉
- Code as data

Code as data

```
xyz
```

```
Term.Name("xyz")
```

```
42
```

```
Lit.Int(42)
```

```
xyz()
```

```
Term.Apply(Term.Name("xyz"), List.empty)
```

```
xyz(42, "kitten")
```

```
Term.Apply(
  Term.Name("xyz"),
  List(Lit.Int(42), Lit.String("kitten"))
)
```

Scalameta

```
scala> import scala.meta._
import scala.meta._

scala> q"xyz".structure
val res0: String = Term.Name("xyz")

scala> q"42".structure
val res1: String = Lit.Int(42)

scala> q"xyz()".structure
val res2: String = Term.Apply(Term.Name("xyz"), Nil)

scala> q"""xyz(42, "kitten")""".structure
val res3: String = Term.Apply(Term.Name("xyz"),
List(Lit.Int(42), Lit.String("kitten")))
```

This slide was not shown as this part was live-coded

Scalameta solution

```scala
def convertModel(model: Model): Defn.Class = {
  val name = Type.Name(model.name)
  val fields = model.fields.map(convertField)
  q"case class $name( ..$fields)"
}

def convertField(field: Field): Term.Param = {
  val tpe = convertType(field.tpe, field.isOptional)
  Term.Param(Nil, Name(field.name), Some(tpe), None)
}

def convertType(tpe: FieldType, isOptional: Boolean): Type = {
  val baseType = tpe match {
    case TString  ⇒ Type.Name("String")
    case TNumber  ⇒ Type.Name("Int")
    case TBoolean ⇒ Type.Name("Boolean")
  }
  if (isOptional)
    Type.Apply(Type.Name("Option"), List(baseType))
  else
    baseType
}
```

```scala
val example = Model(
  "Customer",
  List(
    Field("customerId", TNumber, isOptional = false),
    Field("firstName", TString, isOptional = true),
    Field("lastName", TString, isOptional = true),
    Field("isActive", TBoolean, isOptional = false)
  )
)
```

```scala
case class Customer(customerId: Int, firstName:
Option[String], lastName: Option[String],
isActive: Boolean)
```

This slide was not shown as this part was live-coded

Thank you

Slides and code at github.com/danieloakey/scalabase-talk