Typed-SQL

Вадим Челышов

- github.com/dos65
- hydrosphere.io
- Scalalaz podcast
- ScalaNews

План

- Дилемма выбора SQL библиотеки
- Состояние экосистемы
- Typed-SQL
- Что внутри?

Дилемма выбора

- SQL
- Scala

Дилемма выбора

- SQL - runtime ошибки
- Scala - "Компилируется значит работает"

Чего хочется?

- Оставаться в рамках одного языка
- Проверка запросов на этапе компиляции
- Поменьше бойлерплейта
- Простая синтаксис - желательно SQL

Дилемма выбора

- DSL - безопасно, но сложно
- Сырые строки - просто, но небезопасно

Ecosystem tour

Ecosystem tour

Ecosystem tour

Slick (“Scala Language-Integrated Connection Kit”) is Lightbend’s Functional Relational Mapping library for Scala that makes it easy to work with relational databases.

Slick

FirstExample.scala

```scala
// Definition of the SUPPLIERS table
class Suppliers(tag: Tag) extends Table[(Int, String, String, String, String, String)](
  def id = column[Int]("SUP_ID", O.PrimaryKey) // This is the primary key column
  def name = column[String]("SUP_NAME")
  def street = column[String]("STREET")
  def city = column[String]("CITY")
  def state = column[String]("STATE")
  def zip = column[String]("ZIP")
  // Every table needs a * projection with the same type as the table's type parameter
  def * = (id, name, street, city, state, zip)
}
val suppliers = TableQuery[Suppliers]
```

Slick

```
// Perform a join to retrieve coffee names and supplier names for
// all coffees costing less than $9.00
val q2 = for {
  c <- coffees if c.price < 9.0
  s <- suppliers if s.id === c.supID
} yield (c.name, s.name)
// Equivalent SQL code:
// select c.COF_NAME, s.SUP_NAME from COFFEES c, SUPPLIERS s where c.PRICE < 9.0 and s.
```

- *map*
- *flatMap*

SQL query is ... ?

- *map*
- *flatMap*

SQL query is just a monoid in the category ...

SQL → Slick → SQL

slick-codegen + *Flyway*

SQL Tables → Slick Tables → SQL → Slick → SQL

- + Типизированный DSL
- + Оптимизации
- - ЕЩЕ ОДИН СИНТАКСИС

ROC

github.com/finagle/roc

finagle / roc

Watch 2 | Star 48 | Fork 6

Code | Issues 13 | Pull requests 2 | Projects 0 | Insights

A Modern Finagle-Postgresql Client http://finagle.github.io/roc/docs/

116 commits | 2 branches | 5 releases | 1 contributor | BSD-3-Clause

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

ROC

github.com/finagle/roc

```
val client = Postgresql.client
  .withUserAndPasswd("username", "password")
  .withDatabase("database")
  .newRichClient("inet!localhost:5432")
val req = new Request("SELECT * FROM STATES;")
val result = Await.result(client.query(req))
```

ROC

github.com/finagle/roc

Roc is published to Maven Central, so for the latest stable version add the following to your build:

```
libraryDependencies ++= Seq(
  "com.github.finagle" %% "roc-core"  % "0.0.4",
  "com.github.finagle" %% "roc-types" % "0.0.4"
)
```

ROC

github.com/finagle/roc

High fives for everyone!

ROC

github.com/finagle/roc

finagle / roc

Watch 2 | Star 48 | Fork 6

Code | Issues 13 | Pull requests 2 | Projects 0 | Insights

Requests returning results from previous queries #74

New issue

Open **enginoid** opened this issue on Jun 29, 2016 · 0 comments

enginoid commented on Jun 29, 2016 • edited

We've been experiencing some occasional "Could not find element X in row" exceptions when mapping

Assignees

penland365

- Будьте внимателены
- Не ведитесь на картинки

Compile-time Language Integrated Query for Scala


```
import io.getquill._

case class Person(name: String, age: Int)
```

flatMap

```
val q = quote {
  query[Person].filter(p => p.age > 18).flatMap(p => query[Contact].filter(c => c.pers
}

ctx.run(q)
// SELECT c.personId, c.phone FROM Person p, Contact c WHERE (p.age > 18) AND (c.perso
```

Quill vs Slick

- Меньше бойлерплейта
- QDSL + Compile Time - Dualistic Quotations in Quill

Pure functional JDBC layer for Scala


```scala
case class Country(code: String, name: String, population: Long)

def find(n: String): ConnectionIO[Option[Country]] =
  sql"select code, name, population from country where name = $n".query[Country].option
```

Doobie


enumeratum
ecosystem
playframework
effect
scalafx
doobie
library
refined
spark
slick
lagom
better
actor
cat
js
fs2
breeze
scala
akka
sbt
native
aws
mill
scala.js
kafka
circe
play
http
utest
quill
important
ensime
lib
lift
scalaj
http4s
stream
flink
spire
file
apache
typelevel
framework
kamon
ammonite
monocle
macwire
fastparse
json4s

Doobie

- + Чистый SQL
- + Cats-effect
- - Не типо-безопасный

Typed-SQL

github.com/Hydrospheredata/typed-sql

Hydrospheredata / **typed-sql**

Unwatch 24 | Star 55 | Fork 1

Code | Issues 4 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

No description, website, or topics provided.

Edit

Manage topics

45 commits | **1** branch | **1** release | **1** contributor | Apache-2.0

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

Typed-SQL

Typed-SQL

- Typesafe DSL *for Doobie*

Typed-SQL

- Typesafe DSL *for Doobie*
- v0.1.0

Typed-SQL

- Typesafe DSL *for Doobie*
- v0.1.0 - **не для продакшена**

Typed-SQL

- Typesafe DSL *for Doobie*
- v0.1.0 - **не для продакшена**
- Концепт шикарен!

Typed-SQL

SQL syntax == eDSL

Typed-SQL

```scala
case class Row(
  a: Int,
  b: String,
  c: String
)

val table = Table.of[Row].name('test)
// or if `a` column is a primary key and has serial type
val table = Table.of[Row].autoColumn('a).name('test)

val a = table.col('a)
val b = table.col('b)
val c = table.col('c)
```

Typed-SQL

```
insert.into(table).values(1, "b", "c")
// or if `a` column is a primary key and has serial type
insert.into(table).values("b", "c")

select(*).from(table)
select(*).from(table).where(a === 1)
select(a, b).from(table)

update(table).set(b := "Upd B").where(a === 1)

delete.from(table).where(a === 1)
```

Typed-SQL

```
val q0: Query0[Row] = select(*).from(table).toQuery
// the same for update and insert
val u0: Update0 = delete.from(table).where(a === 1).toUpdate
```

Typed-SQL

- + SQL
- + Typesafe

Похожие проекты

- Scalalike JDBC
- Frameless

Что внутри?

Что внутри?

Shapeless!

Shapeless features

- Singletons
- FieldType/LabeledGeneric

Singleton

```
scala> import syntax.singleton._
import syntax.singleton._

scala> 'foo          // non-singleton type
res0: Symbol = 'foo

scala> 'foo.narrow   // singleton type
res1: Symbol with shapeless.tag.Tagged[String("foo")] = 'foo
```

Records

```
case class Row(a: Int, b: String)

val row = ("a" ->> 42) :: ("b"  ->> "value") :: HNil
```

Records

```
scala> val rec = ("a" ->> 42) :: ("b"  ->> "value") :: HNil
rec: Int with KeyTag[String("a"),Int] ::
      String with KeyTag[String("b"),String]::
      shapeless.HNil = 42 :: value :: HNil

scala> rec("a")
res9: Int = 42

scala> rec("c") // compile-time error
error: No field String("c") in record ...
```

Типобезопасность - SQL

- Принадженость колонки таблице
- Проверка операций (like for String)

Механика доказательств

```
select(a, b, c).from(table)
```

```
def select[In](in: In): SelectionPrefix[In]

class SelectionPrefix[In](in: In) {

  def from[A](t: Table[A])(implicit
    proof: SelectionInfer[A, In, Out]
  ): Selection[A, In, Out]
}
```

Механика доказательств

```
select(a, b, c).from(table)
```

Дано - *select(a, b, c)*

From: Есть ли такие колонки? - *from(table)*

Механика доказательств

```
select(a, b, c).from(table).where(a === 42)
```

Дано - *select(a, b, c).from(table)*

Where: a это Int? - *a === 42*

Where: есть ли колонка a? - *where(a === 42)*

Heterogeneous table shape

```
select(a).from(table)
```

```
sealed trait Shape
// A - RecordType:
//   FieldType[Int with "a"] ::
//   FieldType[String with "b"] ::
//   HNil
case class From[A](t: Table[A]) extends Shape
```

Join

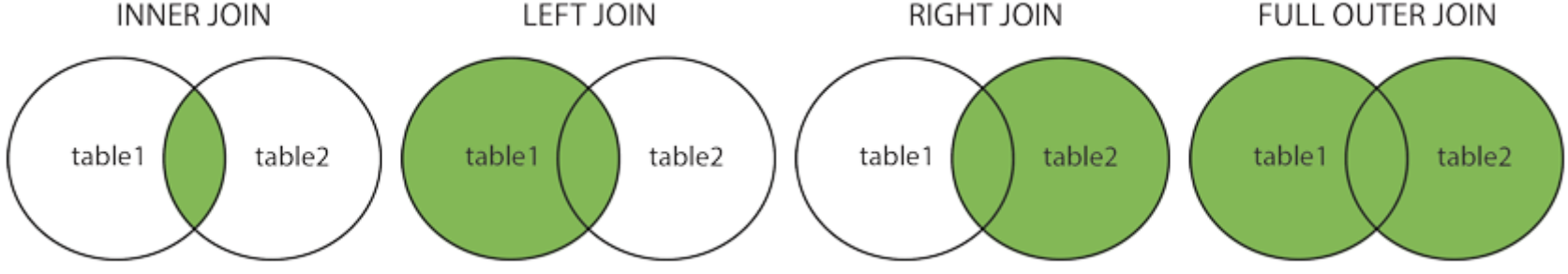
```scala
sealed trait Shape
case class From[A](t: Table(a)) extends Shape

case class InnerJoin[T1, T2, JoinCond](
  t1: Table[T1],
  t2: Table[T2],
  cond: JoinCond,
) extends Shape
```

```scala
// InnerJoin[
//   From[RecordType1],
//   From[RecordType2],
//   Eq["a1", "a2"]
// ]
val joined = table1.innerJoin(table2).on(a1 <==> a2)
select(*).from(joined)
```

Joins

```
sealed trait Shape
case class From[A](t: Table(a)) extends Shape
case class InnerJoin[T1, T2, JoinCond] ... extends Shape
case class LeftJoin[T1, T2, JoinCond] ... extends Shape
case class FullJoin[T1, T2, JoinCond] ... extends Shape
```

Joins


INNER JOIN
table1
table2
LEFT JOIN
table1
table2
RIGHT JOIN
table1
table2
FULL OUTER JOIN
table1
table2

Joins

```
// Query0[(Row1, Row2)]
select(*).from(table.innerJoin(table2).on(a1 <==> a2))
// Query0[(Row1, Option[Row2])]
select(*).from(table.leftJoin(table2).on(a1 <==> a2))
// Query0[(Option[Row1], Row2)]
select(*).from(table.rightJoin(table2).on(a1 <==> a2))
// Query0[(Option[Row1], Option[Row2])]
select(*).from(table.fullJoin(table2).on(a1 <==> a2))
```

Typed-SQL

github.com/Hydrospheredata/typed-sql

- + Выглядит почти как SQL
- + Typesafe
- - Выглядит почти как SQL
- - Невменяемые имплиситы/некрасивые ошибки