Slick SQL Interaction in Scala

Jan Christopher Vogt, EPFL
Slick Team
Slick (vs. ORM)

- Functional-Relational Mapper
- natural fit (no impedance mismatch)
- declarative
- embraces relational
- stateless
- Slick is to ORM what Scala is to Java
8 Reasons for using Slick
1

Scala collection-like API
Scala collection-like API

```scala
for ( d <- Devices;
    if d.price > 1000.0
  ) yield d.acquisition
```

<table>
<thead>
<tr>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: Long</td>
</tr>
<tr>
<td>price: Double</td>
</tr>
<tr>
<td>acquisition: Date</td>
</tr>
</tbody>
</table>

Devices

```scala
Devices
  .filter(_.price > 1000.0)
  .map(_.acquisition)
```
2
Predictable SQL structure
Predictable SQL structure

Devices
  .filter(_.price > 1000.0)
  .map(_.acquisition)
  .selectStatement

select x2."ACQUISITION" from "DEVICE"
x2 where x2."PRICE" > 1000.0
3

Type-safety
Compile-Time Safety

- Spelling mistake in column name?
- Wrong column type?
- Query doesn’t match the result type?

scalac sees it all!
Caution: Error messages can be bad

Piotr Buda @piotrbuda

...and the 'Most Informative Stack Trace Award goes to...' evernote.com/shard/s28/sh/5... #slick #scala

12 hours ago
Enforce schema consistency

- Generate DDL from table classes
- Slick 2.0: Generate table classes and mapped classes from database
4
Small configuration using Scala code
class Devices(tag: Tag) extends Table[((Long, Double, Date))(tag,"DEVICE") { 
  def id          = column[Long]  ("ID", O.PrimaryKey)
  def price       = column[Double]("PRICE")
  def acquisition = column[Date]  ("ACQUISITION")
  def * = (id, price, acquisition)
}
def Devices = TableQuery[Devices]

### Table description

can be auto-generated in Slick 2.0
Connect

```scala
import scala.slick.driver.H2Driver.simple._

val db = Database.forURL("jdbc:h2:mem:testdb", "org.h2.Driver")

db.withTransaction { implicit session =>
  // <- run queries here
}
```
Explicit control over execution and transfer
val query = for {
    d <- Devices
    if d.price > 1000.0
} yield d.acquisition

db.withTransaction { implicit session =>

    val acquisitionDates = query.run(session)

    no unexpected behavior,
    no loading strategy configuration,
    just write code
Loosely-coupled, flexible mapping
Table description

class Devices(tag: Tag)
    extends Table[(Long, Double, Date)](tag,"DEVICE") {
    def id          = column[Long]  ("ID", 0.PrimaryKey)
    def price       = column[Double]("PRICE")
    def acquisition = column[Date]  ("ACQUISITION")
    def * = (id, price, acquisition)
}
val Devices = TableQuery[Devices]
case class Device(id: Long, price: Double, acquisition: Date)

class Devices(tag: Tag)
    extends Table[Device](tag, "DEVICE") {
    def id          = column[Long]  ("ID", O.PrimaryKey)
    def price       = column[Double]("PRICE")
    def acquisition = column[Date]  ("ACQUISITION")

    def * = (id, price, acquisition) <>
            (Device.tupled, Device.unapply)
}

val Devices = TableQuery[Devices]
Custom mapping

def construct : ((Long,Double,Date)) => CustomType
def extract: CustomType => Option[((Long,Double,Date)]

class Devices(tag: Tag)
    extends Table[CustomType](tag,"DEVICE") {
        def id = column[Long] ("ID", O.PrimaryKey)
        def price = column[Double]("PRICE")
        def acquisition = column[Date] ("ACQUISITION")
        def * = (id, price, acquisition) <>
            (construct,extract)
    }

val Devices = TableQuery[Devices]
Plain SQL support
Plain SQL support

```scala
import scala.slick.jdbc.{GetResult, StaticQuery}
import StaticQuery.interpolation

implicit val getDeviceResult =
  GetResult(r => Device(r.<<, r.<<, r.<<))

val price = 1000.0

val expensiveDevices: List[Device] =
  sql"select * from DEVICE where PRICE > $price"
  .as[Device].list
```
8 composable / re-usable queries
Composable, re-usable queries

def deviceLocations
    (companies: Query[Companies, Company])
    : Query[Column[String], String] = {
        companies.computers.devices.sites.map(_.location)
    }

val apples = Companies.filter(_.name iLike "%apple%")
val locations : Seq[String] = {
    deviceLocations(apples)
        .filter(_.inAmerica: Column[String] => Column[Boolean])
        .run
}

execute exactly one, precise query

re-use queries

re-use joins

re-use user-defined operators
Live Demo
Slick app design
Mental paradigm shift

Non-composable executor APIs (DAOs)

DevicesDAO
    .inPriceRange( 500.0, 2000.0 )
      : List[Device]

Composable query libraries

devices
    .inPriceRange( 500.0, 2000.0 )
      : Query[_,Device]
Suggested Slick app architecture

View → Controller → Database Session

Table classes:
- Companies
- Computers
- Devices
- Sites

Composable Query Library:
- `def byId( Column[Long] ) : Query[...],Computers]`
- `def withComputers : Query[...,(...,Computers)]`
- `def iLike( Column[String] ) : Column[Boolean]`

Non-composable Executor API / DAO:
- `def byId( id:Long ) : Device`
- `def withComputers : Map[...],Seq[Computer]]`
- `...`
Relationships / Associations

• Via composable queries using foreign keys!

```plaintext
companies.withComputers : Query[...(Company,Computer)]
```

• Not object references within query results
• Not executor APIs
Auto joins
(only in play-slick sample app)

```scala
implicit def autojoin1 = joinCondition[Sites, Devices]
  (_.id === _.siteId)

implicit def autojoin2 = joinCondition[Devices, Computers]
  (_.computerId === _.id)
```

```
sites.autoJoin(devices).further(computers)
  : Query[_, (Site, Computer)]

sites.autoJoin(devices).autoJoinVia(computers)(_._2)
  : Query[_, ((Site, Device), Computer)]
```
Other features
Other features

• inserts += +=, updates query.update(…)
• user defined column types, e.g. type-safe ids
• user defined database functions
• …
Outlook
2.0 is around the corner

- code-generation based type providers
- hlists and custom shapes (no 22-col limit, easy integration with shapeless, etc.)
- distributed queries (over multiple dbs)
- improved pre-compiled queries
Current experiments

• improved macro-based api (simpler types)
• macro-based type providers
• schema manipulation api
• migration/version management tool
• extended for-comprehensions (order, group)

Thanks to @amirsh @clhodapp @nafg
Thank you

slick.typesafe.com

@cvogt
@StefanZeiger

http://slick.typesafe.com/talks/
https://github.com/cvogt/play-slick/