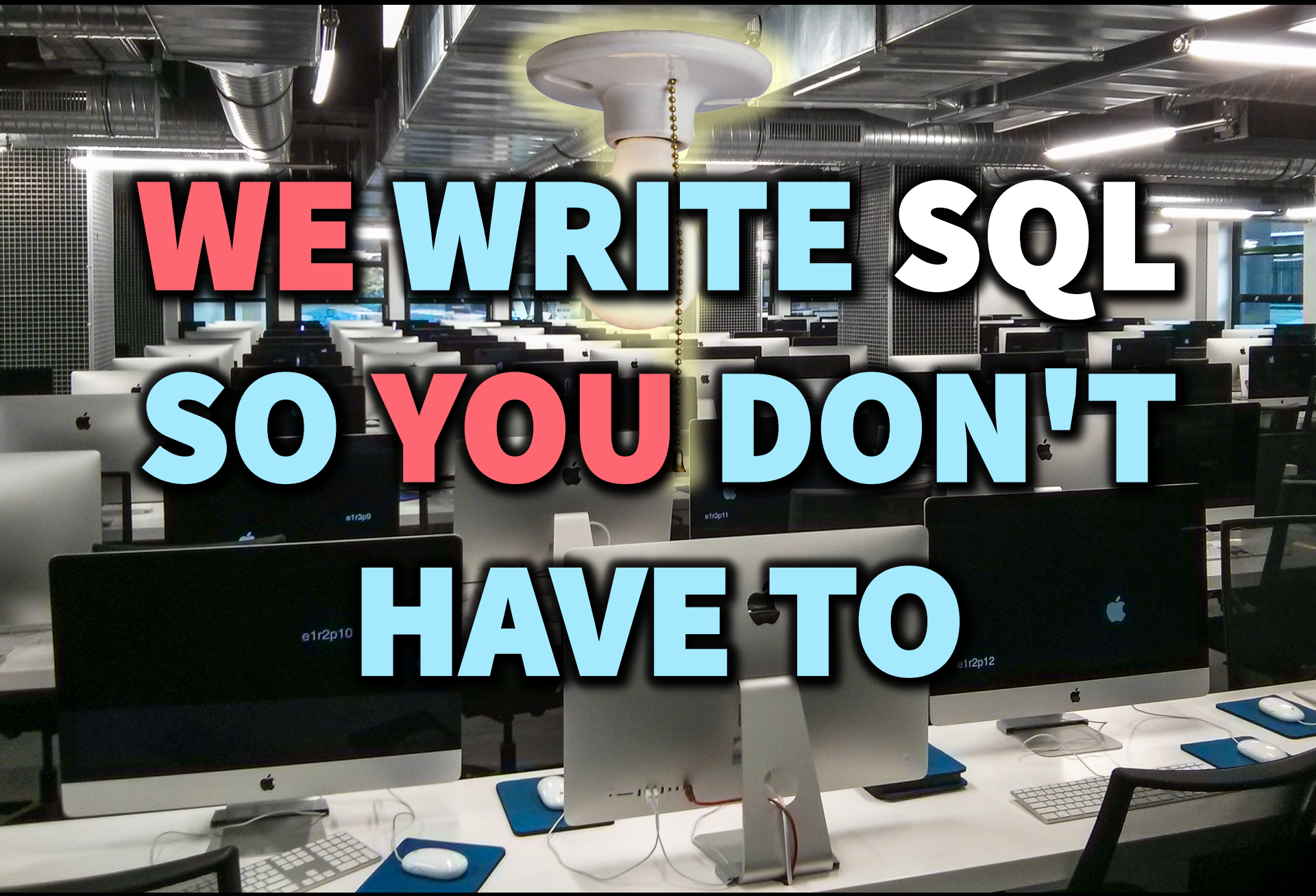
Database Access with Slick

Stefan Zeiger

Introduction

http://toto.lib.unca.edu/findingaids/photo/national_climatic_data_center/NCDC_interior.htm

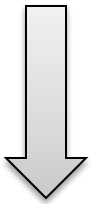
NOAA's National Climatic Data Center is the source of this image and it is used by permission


WE WRITE SQL
SO YOU DON'T
HAVE TO

Write database code in Scala

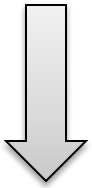
- Instead of SQL, JPQL, Criteria API, etc.

```
for { p <- persons } yield p.name
```



```
select p.NAME from PERSON p
```

```
(for {
      p <- persons.filter(_.age < 20) ++
           persons.filter(_.age >= 50)
           if p.name.startsWith("A")
} yield p).groupBy(_.age).map { case (age, ps) =>
  (age, ps.length)
}
```



```
select x2.x3, count(1) from (
  select * from (
    select x4."NAME" as x5, x4."AGE" as x3
      from "PERSON" x4 where x4."AGE" < 20
    union all select x6."NAME" as x5, x6."AGE" as x3
      from "PERSON" x6 where x6."AGE" >= 50
    ) x7 where x7.x5 like 'A%' escape '^'
  ) x2
group by x2.x3
```

Scala Language Integrated Connection Kit

- Database query and access library for Scala
- Successor of ScalaQuery
- Developed at Typesafe and EPFL
- Open Source

Functional-Relational Mapping

- Embraces the relational model
- No impedance mismatch

```
class Suppliers ... extends
    Table[(Int, String, String)](... "SUPPLIERS")


sup.filter(_.id < 2) ++ sup.filter(_.id > 5)
```

Functional-Relational Mapping

- Composable Queries

```
def f(id1: Int, id2: Int) =
  sup.filter(_.id < id1) ++ sup.filter(_.id > id2)

val q = f(2, 5).map(_.name)
```

Functional-Relational Mapping

- Explicit control over statement execution
- Stateless

```
val result = q.run
```

Supported Databases

- PostgreSQL
- MySQL
- H2
- Hsqldb
- Derby / JavaDB
- SQLite
- Access

Closed-Source *Slick Extensions* (with commercial support by Typesafe):

- Oracle
- DB/2
- SQL Server

Architecture

Components

- **Lifted Embedding**
- Direct Embedding
- **Plain SQL**
- **Session Management**
- Schema Model
- **Code Generator**

Session Management

Unified Session Management

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

val db = Database.forURL("jdbc:h2:mem:test1",
                         driver = "org.h2.Driver")

db withSession { implicit session =>
  doSomethingWithSession
}
```

- forName
- forDataSource

withTransaction

Driver-Independence

```
class MyDAO(driver: JdbcProfile) {
  import driver.simple._
  ...
}
```

BasicProfile
↳ RelationalProfile
↳ SqlProfile
↳ JdbcProfile

➡ MultiDBExample and MultiDBCakeExample in https://github.com/slick/slick-examples

Code Generator

sbt Setup

```
lazy val slick = TaskKey[Seq[File]]("gen-tables")
lazy val slickCodeGenTask =
    (sourceManaged, dependencyClasspath in Compile,
    runner in Compile, streams) map { (dir, cp, r, s) =>
  val outputDir = (dir / "slick").getPath
  val url = "jdbc:h2:~/test"
  val jdbcDriver = "org.h2.Driver"
  val slickDriver = "scala.slick.driver.H2Driver"
  val pkg = "demo"
  toError(r.run(
    "scala.slick.model.codegen.SourceCodeGenerator",
    cp.files, Array(slickDriver, jdbcDriver, url, outputDir,
    pkg), s.log))
    Seq(file(outputDir + "/demo/Tables.scala"))
}
```

Lifted Embedding

Table Definition

```
class Suppliers(tag: Tag) extends
    Table[(Int, String, String)](tag, "SUPPLIERS") {
  def id = column[Int]("SUP_ID",
                       O.PrimaryKey, O.AutoInc)
  def name = column[String]("SUP_NAME")
  def city = column[String]("CITY")
  def * = (id, name, city)
}


val suppliers = TableQuery[Suppliers]
```

Custom Row Types

```
case class Supplier(id: Int, name: String,
  city: String)

class Suppliers(tag: Tag) extends
    Table[ Supplier ](tag, "SUPPLIERS") {
  def id = column[Int]("SUP_ID",
                       O.PrimaryKey, O.AutoInc)
  def name = column[String]("SUP_NAME")
  def city = column[String]("CITY")
  def * = (id, name, city) <>
    (Supplier.tupled, Supplier.unapply)
}

val suppliers = TableQuery[Suppliers]
```

Custom Column Types

```
class SupplierId(val value: Int) extends AnyVal

case class Supplier(id: SupplierId, name: String,
  city: String)

implicit val supplierIdType = MappedColumnType.base
  [SupplierId, Int](_.value, new SupplierId(_))

class Suppliers(tag: Tag) extends
    Table[Supplier](tag, "SUPPLIERS") {
  def id = column[SupplierId]("SUP_ID", ...)
  ...
}
```

Custom Column Types

```
class SupplierId(val value: Int) extends MappedTo[Int]

case class Supplier(id: SupplierId, name: String,
  city: String)

class Suppliers(tag: Tag) extends
    Table[Supplier](tag, "SUPPLIERS") {
  def id = column[SupplierId]("SUP_ID", ...)
  ...
}
```

Foreign Keys

```
class Coffees(tag: Tag) extends Table[
    (String, SupplierId, Double)](tag, "COFFEES") {
  def name = column[String]("NAME", O.PrimaryKey)
  def supID = column[SupplierId]("SUP_ID")
  def price = column[Double]("PRICE")
  def * = (name, supID, price)
  def supplier =
    foreignKey("SUP_FK", supID, suppliers)(_.id)
}

val coffees = TableQuery[Coffees]
```

Creating Tables and Inserting Data

```
val suppliers = new ArrayBuffer[Supplier]
val coffees = new ArrayBuffer[(String, SupplierId, Double)]
```

```
suppliers += Supplier(si1, "Acme, Inc.", "Groundsville")
suppliers += Supplier(si2, "Superior Coffee", "Mendocino")
suppliers += Supplier(si3, "The High Ground", "Meadows")

coffees ++= Seq(
  ("Colombian",         si1, 7.99),
  ("French_Roast",      si2, 8.99),
  ("Espresso",          si3, 9.99),
  ("Colombian_Decaf",   si1, 8.99),
  ("French_Roast_Decaf", si2, 9.99)
)
```

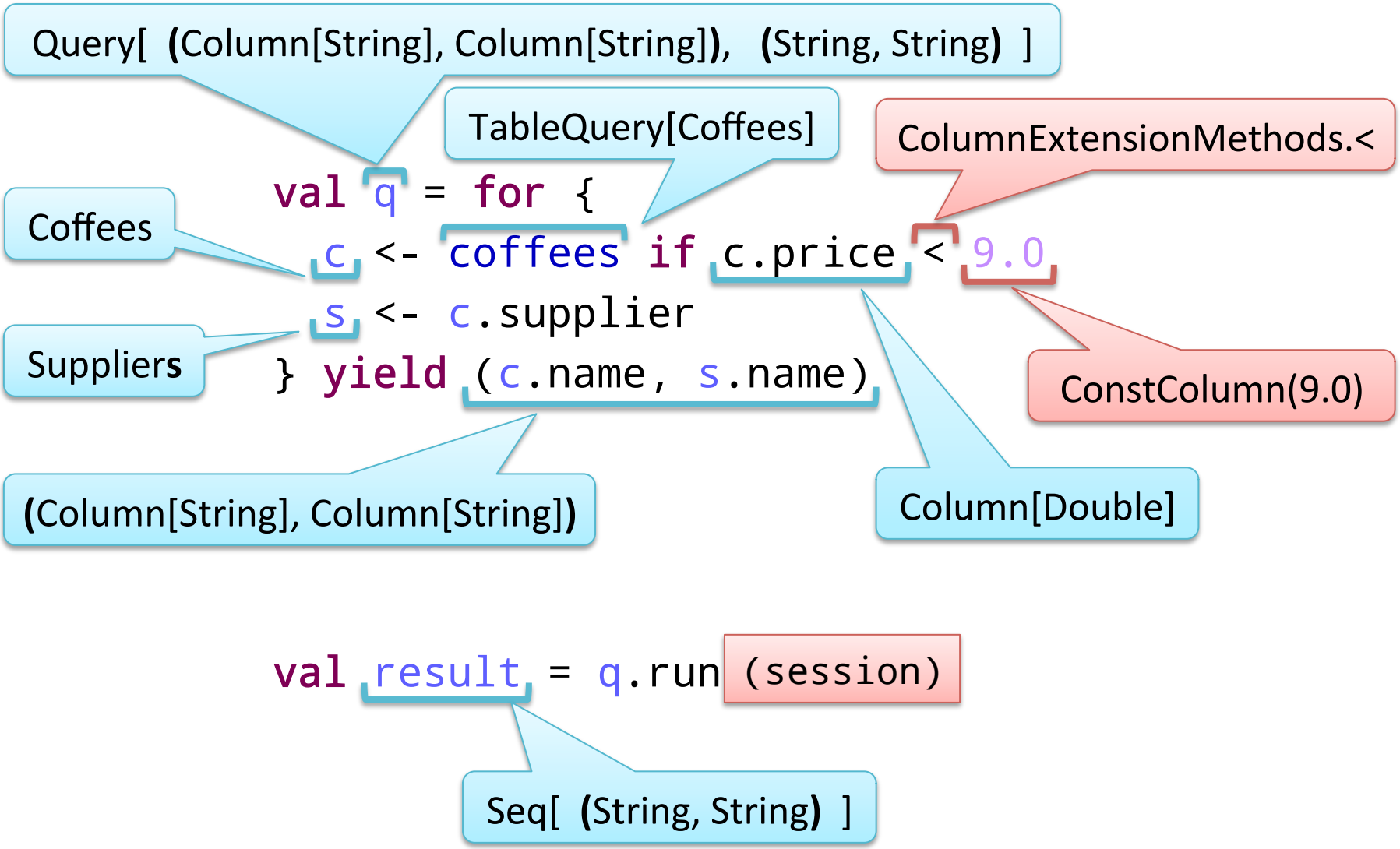
Auto-Generated Keys

```
val ins = suppliers.map(s => (s.name, s.city))
  returning suppliers.map(_.id)

val si1 = ins += ("Acme, Inc.", "Groundsville")
val si2 = ins += ("Superior Coffee", "Mendocino")
val si3 = ins += ("The High Ground", "Meadows")

coffees ++= Seq(
  ("Colombian",          si1, 7.99),
  ("French_Roast",       si2, 8.99),
  ("Espresso",           si3, 9.99),
  ("Colombian_Decaf",    si1, 8.99),
  ("French_Roast_Decaf", si2, 9.99)
)
```

Queries


Query[(Column[String], Column[String]), (String, String)]
TableQuery[Coffees]
ColumnExtensionMethods.<
Coffees
Suppliers
val q = for {
c <- coffees if c.price < 9.0
s <- c.supplier
} yield (c.name, s.name)
ConstColumn(9.0)
(Column[String], Column[String])
Column[Double]
val result = q.run (session)
Seq[(String, String)]

More Queries

```
val q1 = suppliers.filter(_.id === 42)
val q2 = suppliers.filter(_.id =!= 42)

val q4 = (for {
  c <- coffees
  s <- c.supplier
} yield (c, s)).groupBy(_._2.id).map { case (_, q) =>
  (q.map(_._2.name).min.get, q.length)
}
```

Column[**Option[String]**]

Plain SQL

JDBC

```
def personsMatching(pattern: String)(conn: Connection) = {
  val st = conn.prepareStatement(
    "select id, name from person where name like ?")
  try {
    st.setString(1, pattern)
    val rs = st.executeQuery()
    try {
      val b = new ListBuffer[(Int, String)]
      while(rs.next)
        b.append((rs.getInt(1), rs.getString(2)))
      b.toList
    } finally rs.close()
  } finally st.close()
}
```

Slick

```
def personsMatching(pattern: String)(implicit s: Session) =
  sql"select id, name from person where name like $pattern"
    .as[(Int, String)].list
```

Outlook

Slick 2.0 – What's New

- Improved API
- Code Generator
- Query scheduling (experimental)
- New driver and back-end architecture

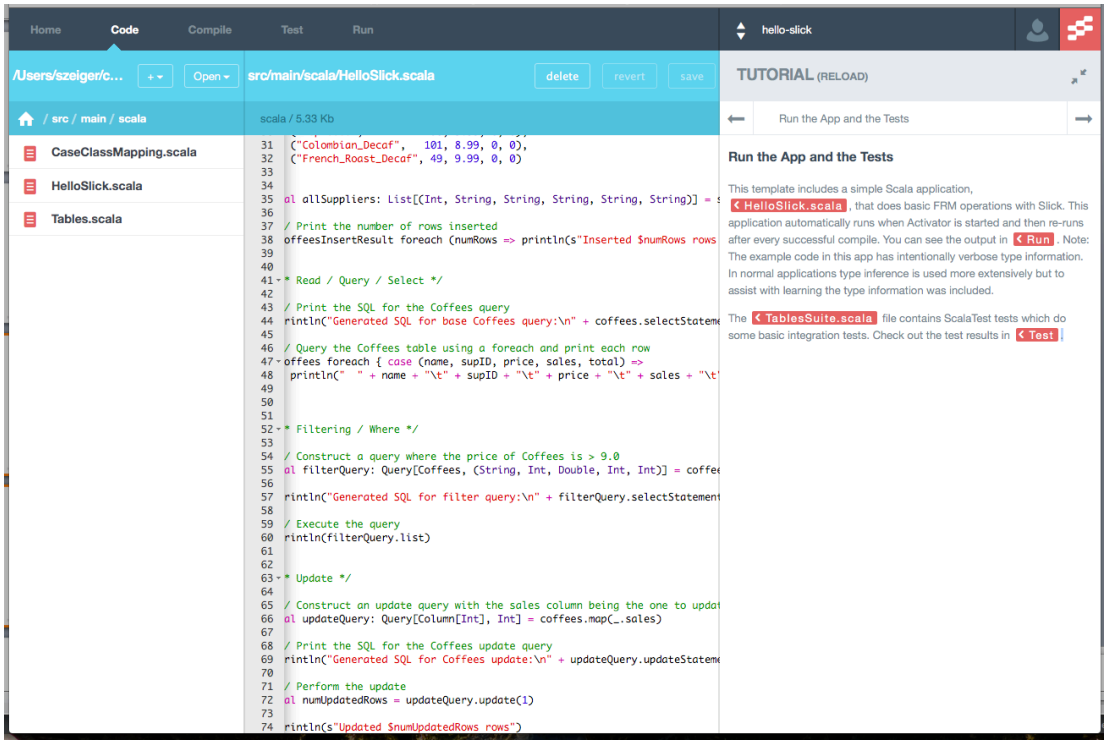
Outlook

- Slick 2.1: Focus on usability (API, docs, semantics, etc.)
- Default database library for Play 2.3
  - as part of the Typesafe Platform
- Macro-based type providers
  - Prototype based on type macros (*`topic/type-providers`*)
  - Macro annotations should be enough
  - Scala 2.12? Dotty?
- Investigating async support and Java API

Getting Started

- Typesafe Activator:

  http://typesafe.com/activator



slick.typesafe.com

@StefanZeiger