Database Access with Slick

Stefan Zeiger, Typesafe
WE WRITE SQL
SO YOU DON'T HAVE TO

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
Write database code in Scala

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

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

```sql
select p.NAME from PERSON p
```
```sql
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
)
INTRODUCTION
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
- Composable Queries
- Explicit control over statement execution
- Stateless
Supported Databases

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

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

• Oracle
• DB/2
• SQL Server
Components

• Lifted Embedding
• Direct Embedding
• Plain SQL
• Session Management
• Schema Model
import scala.slick.driver.H2Driver.simple._

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

• forName
• forDataSource

db withSession { implicit session =>
  doSomethingWithSession
}

  withTransaction
Driver-Independence

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

MultiDBExample and MultiDBCakeExample in https://github.com/slick/slick-examples
LIFTED EMBEDDING
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]
case class Supplier(id: Int, name: String, city: String)

class Suppliers(tag: Tag) extends Table[Supplier](tag, "SUPPLIERS") {
  def id = column[Int]("SUP_ID", 0.PrimaryKey, 0.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 id: Int) extends AnyVal

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

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

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

```scala
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

```scala
val ins = suppliers.map(s => (s.name, s.city))

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)
)
```

returning suppliers.map(_.id)

val si1 =
val si2 =
val si3 =
val q = for {
  c <- coffees if c.price < 9.0
  s <- c.supplier
} yield (c.name, s.name)

val result = q.run(session)
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
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()
}
def personsMatching(pattern: String)(implicit session: Session) =
  sql"select id, name from person where name like $pattern"
  .as[(Int, String)].list
OUTLOOK
Slick 2.0

• Coming Q3 / 2013
• Query scheduling
• API Improvements
• New driver and backend architecture
• Generate Slick code from database schemas

• This presentation is based on 2.0.0-M2
Outlook

• MongoDB (scheduled for Q1/2014)
• Asynchronous, non-blocking API
• Macro-based type providers (Scala 2.12?)
• Default database library for Play