A database query library for Scala

"select * from person"

or

for( p <- Persons ) yield p

including insert, update, delete, DDL
ORM is a swamp
Slick is to Hibernate and JDBC, what Scala is to Java and Groovy

Slick
• **Easy, Concise, Scalable, Safe, Compositional**

Hibernate
• Complex
• Scalable, if used with caution
• HQL: unsafe, non-compositional
• Criteria Queries: safer, compositional, verbose

JDBC/Anorm
• SQL: unsafe, non-compositional

Relational
SQL
rows
expressions
NULL
...

Functional
comprehensions
tuples / case classes
lambdas
Option
...


Agenda

• Key features
• Live demo
• Detailed query features
• Under the hood
• Upcoming features
Slick key features

• **Easy**
  – access stored data like collections
  – unified session handling

• **Concise**
  – Scala syntax
  – fetching results without pain

• **Scales naturally**
  – stateless
  – explicit control

• **Safe**
  – no SQL-injections
  – compile-time checks (names, types, typos, etc.)

• **Composable**
  – it’s Scala code: abstract and re-use with ease
Easy

- It’s Scala – you already know it
- Access stored data like Scala collections

```scala
for(p <- Persons if p.id === 3) yield p.name
```

```scala
Persons.withFilter(_.id === 3).map(_.name)
```
Unified Session Management

- Unified: URL, DataSource, JNDI
- Transactions

```scala
import org.slick.session._
implicit val session = Database
  .forURL("jdbc:h2:mem:test1", driver="org.h2.Driver")
  .createSession

session.withTransaction {

  // execute queries here

}

session.close()
```
Concise: queries

```
val name = ... // <- e.g. user input

Hibernate
Criteria
Queries

session.createCriteria(Person.getClass)
    .add(Restrictions.and(
        .add(Restrictions.gt("age", 20))
        .add(Restrictions.lt("age", 25))
    ))

Slick

for(p <- Persons if p.age > 20 || p.age < 25) yield p
```
val name = ... // <- e.g. user input

val sql = "select * from person where name = ?"
val st = conn.prepareStatement(sql)
try {
    st.setString(1, name)
    val rs = st.executeQuery()
    try {
        val b = new ListBuffer[(Int, String)]
        while(rs.next)
            b.append((rs.getInt(1), rs.getString(2)))
    } finally rs.close()
} finally st.close()

(Rectangle) for( p <- Persons if p.name === name ) yield p).list
Scales naturally

• Stateless
  – No caches

• Explicit control
  – What is transferred
  – When is it transferred (execution)

```
(p <- Persons if p.name === name ) yield (p.id,p.name)
```

```
Hi, this is your son's school. We're having some computer trouble.

Oh, dear — did he break something? In a way—

Did you really name your son Robert? "DROP TABLE Students;--?"

Oh, yes. Little Bobby Tables, we call him.

Well, we've lost this year's student records. I hope you're happy.

And I hope you've learned to sanitize your database inputs.
Slick is Safe

```scala
val name = ... // <- e.g. user input

Hibernate
HQL
"from Person where name = '" + name + "'

SQL
(JDBC/Anorm)
"select * from person where name = '" + name + "'

Hibernate
Criteria Queries
session.createCriteria(Person.getClass)
.add( Restrictions.eq("name", name) )

Slick
for( p <- Persons if p.name === name ) yield p
```

Fully type-checked: No SQL-injections, no typos, code completion
Type-safe use of stored procedures

```scala
def dayOfWeek(c: Column[Date]) = dayOfWeekDynamic(Seq(c))

for (p <- Persons) yield dayOfWeek(p.birthdate)
```
Composable queries

def personByAge(from:Int, to:Int) =
    Persons.filter(p => p.age >= from && p.age <= to)

// Interests of people between 20 and 25
for(p <- personByAge(20, 25); i <- Interests; if i.personId === p.id)
    yield i.text

// Cars of people between 55 and 65
for(p <- personByAge(55, 65); c <- Cars; if c.personId === p.id)
    yield c.model
val name = ... // <- e.g. user input

Slick

( for( p <- Persons if p.name === name ) yield p ).list

Slick using SQL

val sql = "select * from person where name = ?"
query[String, (Int, String)]( sql )( name ).list

Native SQL fallback
Not type-safe, but still more convenient than JDBC
# Comparison

<table>
<thead>
<tr>
<th></th>
<th>JDBC</th>
<th>Anorm</th>
<th>Slick</th>
<th>SQueryl</th>
<th>HQL</th>
<th>Crit.Q.</th>
</tr>
</thead>
<tbody>
<tr>
<td>API (safe, composable)</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Concise</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scala coll. Syntax</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SQL-Like</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Native SQL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

*Unique Slick features coming up soon*
## Supported DBMS

<table>
<thead>
<tr>
<th></th>
<th>JDBC / Anorm</th>
<th>Slick</th>
<th>Squeryl</th>
<th>Hibernate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DB2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MS SQL Server</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sybase</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MySQL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Derby/JavaDB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>H2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HSQLDB/HyperSQL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MS Access</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SQLite</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

NoSQL coming up in Slick: Summer 2013
Slick in the ecosystem

• Slick will be official database connector in Play / Typesafe Stack
• Successor of ScalaQuery
• Inspired by LINQ
• Currently based on JDBC
• NoSQL coming summer 2013
• Influenced by Scala Integrated Query
Stable Versions

• This talk: Slick 0.11 pre-release for Scala 2.10
  – Slick 1.0 coming during Scala 2.10‘s RC period
  – http://slick.typesafe.com

• Use ScalaQuery 0.10 for Scala 2.9
  – http://scalaquery.org

• License: BSD
Live Demo

• Setup
• Meta data
• Queries
  – insert some data
  – find all people above a certain age with their tasks
• Abstractions

Result at
https://github.com/cvogt/slick-presentation
Persons
  .sortBy(_.name)
  .drop(5).take(10)
Grouping and aggregation

// Number of people per age

Persons
  .groupBy(_.age)
  .map( p => ( p._1, p._2.length ) )
First

// person 3

Persons.filter(_.id === 3).first
Union

Persons.filter(_.age < 18)
unionAll
Persons.filter(_.age > 65)
case class Person(..., age : Option[Int] )

object Persons extends Table[Person]("person"){
  def age = column[Option[Int]]("id")
  ...
}

Persons.insertAll(
  Person( 1, "Chris", Some(22) ),
  Person( 2, "Stefan", None )
)
Outer Joins (left, right, full)

for (Join(p, t) <- Tasks outerJoin Persons on (_.personId === _.id)) yield p.title.? ~ t.name.?
object Persons extends Table[Person]("person"){
    def id = column[Int]("id")
    ...
}

object Tasks extends Table[Task]("task"){
    def id = column[Int]("id")
    ...
    def assignees = for( pt <- PersonsTasksAssociations;
        p <- pt.assignee; if pt.taskId === id ) yield p
}

object PersonsTasksAssociations extends Table[(Int,Int)]("person_task"){
    def personId = column[Int]("person_id")
    def taskId = column[Int]("task_id")
    def assignee = foreignKey( "person_fk", personId, Persons )(_.id)
    ...
}

Assignees of task 1:
for( t <- Tasks; ps <- t.assignees; if t.id === 1 ) yield ps
Column Operators

**Common:** .in(Query), .notIn(Query), .count, .countDistinct, .isNull, .isNotNull, .asColumnOf, .asColumnOfType

**Comparison:** === (.is), !== (.isNot), <, <=, >, >=, .inSet, .inSetBind, .between, .ifNull

**Numeric:** +, -, *, /, %, .abs, .ceil, .floor, .sign, .toDegrees, .toRadians

**Boolean:** &&, ||, .unary_!

**String:** .length, .like, ++, .startsWith, .endsWith, .toUpperCase, .toLowerCase, .ltrim, .rtrim, .trim
Other features (not exhaustive)

- auto-increment
- sub-queries
- CASE
- prepared statements
- custom data types
- foreach-iteration
- ...

UNDER THE HOOD
Under the hood

Your app

Native SQL

Slick API

optimizations

Slick Query Tree

Lifting:
Getting Query trees from Scala code

SQL
How lifting works

for ( p <- Persons if p.name === "Chris" ) yield p.name

Scala desugaring

Column[String]  String (implicitly to Column[String])

Persons.filter(p=>p.name === "Chris").map(p=>p.name)

Projection(
  Filter(
    Table( Person ),
    Equals(
      ColumnRef( Person, "name" ),
      Constant( name )
    )
  ),
  ColumnRef(Person,"name"
)

"select name from person where name = 'Chris'")
UPCOMING FEATURES
/ SLICK MILESTONES
Alternative Frontend

Native SQL

Slick "lifted embedding" API

optimizations

Slick Query Tree

SQL

Slick "direct embedding" API

Scala AST

Scala compiler

Slick macros
Alternative Frontend

• Real Scala (types, methods) using macros instead of emulation using lifting
  – no need to think about differences anymore
  – identical syntax
    • == instead of ===
    • if-else instead of case-when
    • ...
  – identical error messages

• Compile-time optimizations

• More compile-time checks
Type providers using macros

- schema auto-generated from database
- compiler checks queries against real database schema

```
object Persons extends Table( "person" )
```

A macro which connects to the db at compile time to fetch schema
Extensible backend

Native SQL ➔ Slick „lifted embedding“ API ➔ Slick Query Tree ➔ SQL ➔ Other backends, e.g. NoSQL like MondoDB

You can hook in here

optimizations
BEGINNING OF 2014
Scheduling over multiple backends

for( p <- Persons; t <- Tasks if p.id ... && t.id ... ) yield ( p, t )

Coming from datasource 1, e.g. Oracle SQL DB

Coming from datasource 2, e.g. MongoDB or webservice
Nested Results

for( p <- Persons ) yield
  ( p, for( t <- Tasks; if ... ) yield t )
  . list
  : List[ ( Person, List[Task] ) ]

• As demonstrated in
  Scala Integrated Query / Ferry
MAYBE 2013
Comprehensive Comprehensions

• For-comprehension support for
  – Sorting
  – Grouping
  – ...

• We are still thinking about it
Summary

Slick makes database access

• easy, concise, scalable, safe, composable

Upcoming features will make Slick

• easier, extensible, faster, more powerful
Thank you!
Questions?

Slick.typesafe.com
EXTRA SLIDES
Direct Embedding

```scala
macro (Scala 2.10)

`Person.filter(p=>p.name == name).map(p=>p)`

String

String

Macro works on this expression's Scala AST at compile time

`Projection(
  Filter(
    Table( Person ),
    Equals(
      ColumnRef( Person, "name" ),
      Constant( name )
    ),
    "*"
  ),
  ""
)`

Arbitrary compile time checks or optimizations possible

== (no need for ===)