Anorm

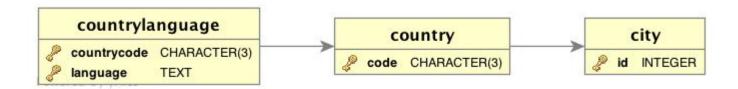
Indy Scala, August 2012

Anorm synopsis

- Originally part of the Play Framework
- Spun off as a standalone project.
- Anorm: JDBC:: Slick: Hibernate
- It's cool. Slick is cool. Right tool for the job.

World Database

- Originally MySQL, but I'm a Postgres bigot
- http://pgfoundry.org/projects/dbsamples/



Getting started

- I had to hunt for the release coordinates.
- Available only in Typesafe Repo. *sigh*

```
libraryDependencies ++=
  "play" %% "anorm" % "2.1-08072012" ::
  "postgresql" %% "postgresql" % "9.1-901.jdbc4" ::
  Nil
```

%% means "for current Scala version"

Connecting

- Most of Anorm works with an implicit JDBC connection.
- The documentation does it with Play, but you don't need to pay for Play.

```
implicit val conn =
   DriverManager.getConnection(
    "jdbc:postgresql:world",
    "ross", "*****")
```

Querying

```
val q = SQL("select count(*) from country")
val rs = q()
val firstRow = rs.head
val countries = firstRow[Long]("count")
```

- Query does not execute until it is applied.
- Applying a query returns a plain old Stream.

A word about Streams

- Streams leak memory if you hold their head.
- You'd be surprised what holds the head:

```
(1 to 10000000).toStream.foldLeft(0)(_+_) // okay (1 to 10000000).toStream.reduce( + ) // FAIL
```

- Tricks to deal with Streams:
 - <u>Tail recursive method that refers to Stream only through an argument</u>.
 - Imperative loops.
 - Convert to iterator.

Params

 Put query params in braces, pass a map to bind them.

```
SQL("""
  select * from country
  where code2 = {code2}
""").on("code2" -> "US")
```

Pattern matching

```
val q = SQL("select code, population, area from country")
val r = q.apply().collect {
  case Row(code: String, population: Int, area: Float) =>
    code -> (population / area)
}
val densities = Map(r : _*)
```

- Caution: types have to be right!
- I'd have done this in the database, but it's a simple calculation.

Nulls

In Scala, we are trained to think "Option" where we used to think null. Good news...

```
val q = SQL("""
  select headofstate from country where code = 'USA'
""")
q().head[Option[String]]("headofstate")
```

Updates

```
SQL("""
  update country
  set headofstate = {president}
  where code = 'USA'
""")
  .on("president" -> "Barack Obama")
  .executeUpdate() // Just like JDBC
```

Conclusions

- Stays out of your way
- If you know JDBC, this is easier.
- Not going to help you much with CRUD.
- Not going to help you at all with refactoring.
- Up and running quickly. I never used it before tonight. It just works.

About Me

Ross A. Baker

Senior Cloud Engineer, CrowdStrike

Scalatra Committer

http://github.com/rossabaker

baker@alumni.indiana.edu