# rogue:
# a scala dsl for mongodb

Scalathon 2011 - 7/16/2011

Jason Liszka (@jliszka)

Jorge Ortiz (@jorgeortiz85)

# what is mongodb?

- fast, schema-less document store
- indexes & rich queries on any field
- sharding, auto-balancing
- replication
- geo-indexes

# mongodb: bson

- json++
- binary wire format
- more built-in types:
  - objectid
  - date
  - regex

# mongodb: bson example

```
{
  “_id” :
    { “oid” : “4ddbd19468614811 0d5c1ccc” },
  “venuename” : “Starbucks”,
  “mayorid” : 464,
  “tags” : [“coffee”, “wifi”, “snacks”],
  “latlng” : [39.0, -74.0]
}
```

# mongodb: query example

```
{
  “mayorid” : { “$lte” : 100 },
  “venuename” : “Starbucks”,
  “tags” : “wifi”,
  “latlng” : { “$near” : [39.0, -74.0] }
}

{ “_id” : -1 }
```

# mongodb: query example

```
val query =
  (BasicDBOBjectBuilder
    .start
    .push(“mayorid”)
      .add(“$lte”, 100)
    .pop
    .add(“veneuname”, “Starbucks”)
    ...
    .get)
```

# rogue: a scala dsl for mongo

- type-safe
- all mongo query features
- logging & validation hooks
- pagination
- index-aware
- cursors

http://github.com/foursquare/rogue

# rogue: schema example

```
class Venue extends MongoRecord[Venue] {
  object _id extends ObjectIdField(this)
  object venuename extends StringField(this)
  object mayorid extends LongField(this)
  object tags extends ListField[String](this)
  object latlng extends LatLngField(this)
}

object Venue extends Venue
             with MongoMetaRecord[Venue]
```

# rogue: code example

```
val vs: List[Venue] =
  (Venue where (_.mayorid <= 100)
          and (_.venuename eqs “Starbucks”)
          and (_.tags contains “wifi”)
          and (_.latlng near
                (39.0, -74.0, Degrees(0.2))
    orderDesc (_._id)
       fetch (5))
```

# rogue: BaseQuery

```
class BaseQuery[M <: MongoRecord[M], R,
                Ord, Sel, Lim, Sk](...) {
  def where[F](clause: M => QueryClause[F]): ...

  ...
}
```

# rogue: QueryField

```
class QueryField[V, M <: MongoRecord[M]]
                (val field: Field[V, M]) {
  def eqs(v: V) = new EqClause(...)
  def neqs(v: V) = ...
  def in[L <% List[V]](vs: L) = ...
  def nin[L <% List[V]](vs: L) = ...
  def exists(b: Boolean) = ...
}
```

# rogue: implicits

```
Field[F, M] => QueryField[F, M]
Field[LatLong, M] => GeoQueryField[M]
Field[List[F], M] => ListQueryField[F, M]
Field[String, M] => StringQueryField[M]
Field[Int, M] => NumericQueryField[Int, M]
Field[Double, M] => NumericQueryField[Double, M]
...
```

# rogue: select

```
val vs: List[(Long, Int)] =
  (Venue where (_.venuename eqs “Starbucks”)
        select (_.mayorid, _.mayorCheckins)
         fetch ())
```

# rogue: select

```
class BaseQuery[M <: MongoRecord[M], R,
                Ord, Sel, Lim, Sk](...) {

  def select[F1, F2](
      f1: M => Field[F1, M],
      f2: M => Field[F2, M]):
        BaseQuery[M, (F1, F2), ...] = ...

  ...
}
```

# rogue: select problems

```
val vs: List[???] =
  (Venue where (_.venuename eqs “Starbucks”)
        select (_.mayorid, _.mayorCheckins)
        select (_.venuename)
         fetch ())
```

# rogue: limit problems

```
val vs: List[Venue] =
  (Venue where (_.venuename eqs “Starbucks”)
        limit (5)
        limit (100)   // (???)
        fetch ())
```

# rogue: more limit problems

```
(Venue where (_.venuename eqs “Starbucks”)
        orderAsc (_.total_checkins)
        limit (5)
        bulkDelete_!!)

// How many Venues got deleted?
```

# rogue: phantom types

- Sel =:= Selected, Unselected
- Ord =:= Ordered, Unordered
- Lim =:= Limited, Unlimited
- Sk =:= Skipped, Unskipped

http://james-iry.blogspot.com/2010/10/phantom-types-in-haskell-and-scala.html

# rogue: phantom types

```
abstract sealed class Ordered
abstract sealed class Unordered
abstract sealed class Selected
abstract sealed class Unselected
abstract sealed class Limited
abstract sealed class Unlimited
abstract sealed class Skipped
abstract sealed class Unskipped
```

# rogue: initializing phantoms

```
class BaseQuery[M <: MongoRecord[M], R,
                Ord, Sel, Lim, Sk](...) {
  ...
}
```

```
def dfltQuery[M <: MongoRecord[M]](rec: M) =
  new BaseQuery[M, M,
                Unordered,
                Unselected,
                Unlimited,
                Unskipped](...)
```

# rogue: select phantom

```
class BaseQuery[M <: MongoRecord[M], R,
                Ord, Sel, Lim, Sk](...) {

  def select[F1, F2](
      f1: M => Field[F1, M],
      f2: M => Field[F2, M])
     (implicit ev: Sel =:= Unselected):
        BaseQuery[M, (F1, F2),
                  Ord, Selected, Lim, Sk] = ...
  ...
}
```

# rogue: =:=

```
sealed abstract class =:=[From, To] ...

implicit def tpEquals[A]: A =:= A = ...
```

# rogue: select solutions

```
val vs: List[???] =
  (Venue where (_.venuename eqs “Starbucks”)
         select (_.mayorid, _.mayorCheckins)
         select (_.venuename)
          fetch ())

// won’t compile!
```

# rogue: limit solutions

```
(Venue where (_.venuename eqs “Starbucks”)
      orderAsc (_.total_checkins)
      limit (5)
      bulkDelete_!!)

// won’t compile!
```

# rogue: index-aware

```
val vs: List[Checkin] =
  (Checkin
    where (_.userid eqs 646)
      and (_.venueid eqs vid)
    fetch ())
```

# rogue: index-aware

```
val vs: List[Checkin] =
  (Checkin
    where (_.userid eqs 646)
      and (_.venueid eqs vid) // hidden scan!
    fetch ())
```

# rogue: index-aware

```
val vs: List[Checkin] =
  (Checkin
    where (_.userid eqs 646)  // known index
     scan (_.venueid eqs vid) // known scan
    fetch ())
```

# rogue: logging & validation

- logging:
  - slf4j
  - Tracer
- validation:
  - radius, $in size
  - index checks

# rogue: pagination

```
val query: Query[Venue] = ...
val vs: List[Venue] =
  (query
    .countPerPage(20)
    .setPage(5)
    .fetch())
```

# rogue: cursors

```
val query: Query[Checkin] = ...

for (checkin <- query) {
  ... f(checkin) ...
}
```

# rogue: future directions

- iteratees for cursors
- compile-time index checking
- select partial objects
- generate full javascript for mapreduce

we’re hiring
(nyc & sf)

http://foursquare.com/jobs
@jliszka, @jorgeortiz85