

Grails and Legacy Databases

Recipes for getting to your data



Ken Rimple
Director, Education Services
Chariot Solutions

Recipes

- Get your Table into Grails...
- Reverse Engineering DBs
- Get your Query into Grails...
- Get your Procedures into Grails...

Pre-Requirement Tracing/ Stats

- Logging SQL - we all know how?
 - `loggingSql = true` in `DataSource.groovy`
- The p6spy plugin - logging SQL - more features and control
- Hibernate JMX statistics - using jConsole

Setting up Stats

```
// grails-app/conf/spring/resources.groovy

import org.springframework.jmx.support.MBeanServerFactoryBean
import org.springframework.jmx.export.MBeanExporter
import org.hibernate.jmx.StatisticsService
import grails.util.GrailsUtil
beans = {
    hibernateStats (StatisticsService) {
        sessionFactory=ref("sessionFactory")
        statisticsEnabled=true
    }

    mbeanServer (MBeanServerFactoryBean) {
        locateExistingServerIfPossible=true
    }

    exporter (MBeanExporter) {
        server = mbeanServer
        beans = ["org.hibernate:name=statistics":hibernateStats]
    }
}
```

Source: <http://grails.org/MBean+export+the+Groovy+way?xhr=true>

Note: set org.hibernate logging to debug, then in jConsole:

org.hibernate.statistics MBean -> operations -> logSummary

Get Your Table Into Grails

- Recipe #1 - use your hbm.xml files
 - `grails create-hibernate-cfg.xml`
 - Place *.hbm.xml files in your `grails-app/hibernate` directory
 - POJO/POGOs in `src/main/java` or `groovy`, or in library classpath JARs
 - Add the includes to `grails-app/hibernate/hibernate.cfg.xml`

GORM & Hibernate

- You can use GORM API calls from your Hibernate objects
- constraints? Put in shadow
DomainNameConstraints.groovy files in the `src/java` directory (no package statement)
- Why there? This needs to be source code, and `src/java` is the only dir that preserves source

GORM & Hibernate

- Option 2 - Use the reverse engineering plugin
 - Creates GORM Domains from entities, primary key types
 - `grails install-plugin db-reverse-engineer`
 - modify `DataSource.groovy`, `Config.groovy`
 - run `grails db-reverse-engineer`
- Tip - set ddl mode to 'verify' or 'none'!!

Options

- Can control what / how domains are created... Add these to Config.groovy
 - `grails.plugin.reveng.packageName`
 - `grails.plugin.reveng.manyToManyBelongsTo`
 - `grails.plugin.reveng.includeTables/`
`includeTableRegexes/`
`includeTablesAntPatterns` (same w/
`exclude`)
 - See excellent plugin documentation

Gotchas

- You're gonna have to have a primary key
 - NullPointerException if not - skip that table
- Names of objects = names of tables, columns (ewww)
- Good start for an application, but then you should customize and Grails-ize the domain

Use static mappings

- Great to modify reversed domains and Groovy-ize the names
- Add the static mapping closure to your class definitions and use the mapping DSL
- Well documented on grails.org
- General hibernate options available - datatypes, column, table names, indexes, etc.

Before

```
package org.grailsug.dell

class Orders {

    Date orderdate
    BigDecimal netamount
    BigDecimal tax
    BigDecimal totalamount
    Customers customers

    static hasMany = [orderlineses: Orderlines]
    static belongsTo = [Customers]

    static mapping = {
        id column: "orderid", generator: "assigned"
        version false
    }

    static constraints = {
        orderdate maxSize: 13
    }
}
```

After

```
package org.grailsug.dell

class Order {

    Date orderDate
    BigDecimal netAmount
    BigDecimal taxAmount
    BigDecimal totalamount
    Customers customers

    static hasMany = [orderlines: Orderlines]
    static belongsTo = [Customers]

    ...
    static mapping = {
        id column: "orderid", generator: "assigned"
        table: "orders"
        orderDate column: "orderdate"
        netAmount column: "netamount"
        taxAmount column: "tax"
        totalAmount column: "total"
        version false
    }
}
```

Use existing Spring Configs

- Use `BuildConfig.groovy` to set up your dependencies
 - `mavenCentral()` - main maven site
 - `mavenLocal()` - your `.m2/repository`
 - Define another repo/location
 - Create a `resources.xml` and import your app context from classpath, etc...

Existing Spring Config (part 2)

- Your Grails datasource is 'dataSource'
- It can be injected
- Watch out for stale changes from maven apps (don't forget that mvn install step)
- You can inject dataSource, sessionFactory, other beans with just ref= keywords

Configure Dependency

- Add Maven coordinates to BuildConfig.groovy

```
dependencies {  
    runtime 'postgresql:postgresql:9.0-801.jdbc4'  
    compile 'world-app-services:world-app-services:1.1-SNAPSHOT'  
}
```

- Then use beans in Hibernate config or Spring Beans in resources.groovy / resources.xml

```
<beans ...>  
  
    <import resource=  
        "classpath:applicationContext-worldappservices.xml"/>  
  
</beans>
```

Hibernate / JPA I

- Grails supports Hibernate and JPA annotations
 - JPA 1.0 for now
 - Grails 1.4 should support JPA 2.0 via upgraded Hibernate commons annotations

Tips

- Must currently use the mapping-class entries to map each entity
- Can add constraints - shadow package structure and create *DomainConstraints.java* in `src/java`
 - No package in file
 - Add constraints as per Grails

Getting your Query into Grails

- Calling Raw SQL w/GSQL
- Using JdbcTemplate
- Injecting dataSource and doing what you wish... (eww)

Getting your Stored Procedure into Grails

- Calling straight from GSQL
- Calling from JdbcTemplate
- Calling from StoredProcedure class
- Calling from CallableStatement class

Calling w/Groovy SQL

```
def sumBookSalesByBookIdGSQL(int bookId) {  
    def gsql = new groovy.sql.Sql(dataSource)  
    def results = gsql.rows("select salesByBookId(${bookId})")  
    results["salesbybookid"][0]  
}
```

- Build a service
- Inject dataSource
- new a groovy.sql.Sql object & go
- rows returns map of result sets

Nothing wrong with JdbcTemplate

- But GroovySQL is easier
- Just do the same thing - construct from `dataSource` & go
- Maybe install it with `resources.groovy`
- Can re-use existing Spring JDBC Template code in Grails

StoredProcedure

- Class that wraps a StoredProcedure definition
- Provides parameter binding
- Can deal with output parameters
- Can compile (bind) procedure for fast setup

Example

```
class StoredProcSubclass extends StoredProcedure {
    public StoredProcSubclass(DataSource ds) {
        super(ds, "salesByBookId")
        setFunction true
        declareParameter(
            new SqlParameter("book_id", java.sql.Types.INTEGER))
        compile()
    }

    public Map execute(Integer bookId) {
        Map inputs = new HashMap()
        inputs.put("book_id", bookId)
        return super.execute(inputs)
    }
}
```

Calling the Procedure

- Details leak out... You get n result sets, named `#result-set- n`
- Results are returned in an array fashion
- Use Groovy Console to test this stuff!!

```
// on startup, in resources.groovy, do equivalent of:
  def spcaller = new StoredProcSubclass(dataSource)
...
def sumBookSalesByBookIdWithSPSubclass(int bookId) {
  def map = spcaller.execute(bookId)
  // result is in entry '#result-set-1',
  // with field 'result' which is a list, taking entry 0
  map['#result-set-1']['result'][0]
}
```

Got a Function?

```
// some startup process
def sqlFunction = new SqlFunction(
    dataSource, "select salesByBookId(${bookId})")
sqlFunction.compile()
...

result = sqlFunction.run(1353)
```

- Use SqlFunction to wrap it
- OR setFunction(true) in StoredProcedure
- This doesn't use CallableStatement
- Assumes 1 row 1 value (int by default)

Wrap-up

- SQL Support in Grails is Strong
 - Spring JDBC, Groovy SQL
 - Stored Procedure objects
- Reverse Engineering Plugin + static mappings tweaks FTW!
- Sample code available at
 - <https://github.com/krimple/grails-dbmapping-demos>

Thank you

- Plugs
 - Emerging Technologies for the Enterprise 2011 - selling out soon!
phillyemergingtech.com
 - Spring, Hibernate, Grails, on-site or public education courses at Chariot Solutions -
chariotsolutions.com/education