Working with Databases and Groovy

Dr Paul King

Groovy Lead for Object Computing Inc.

@paulk_asert

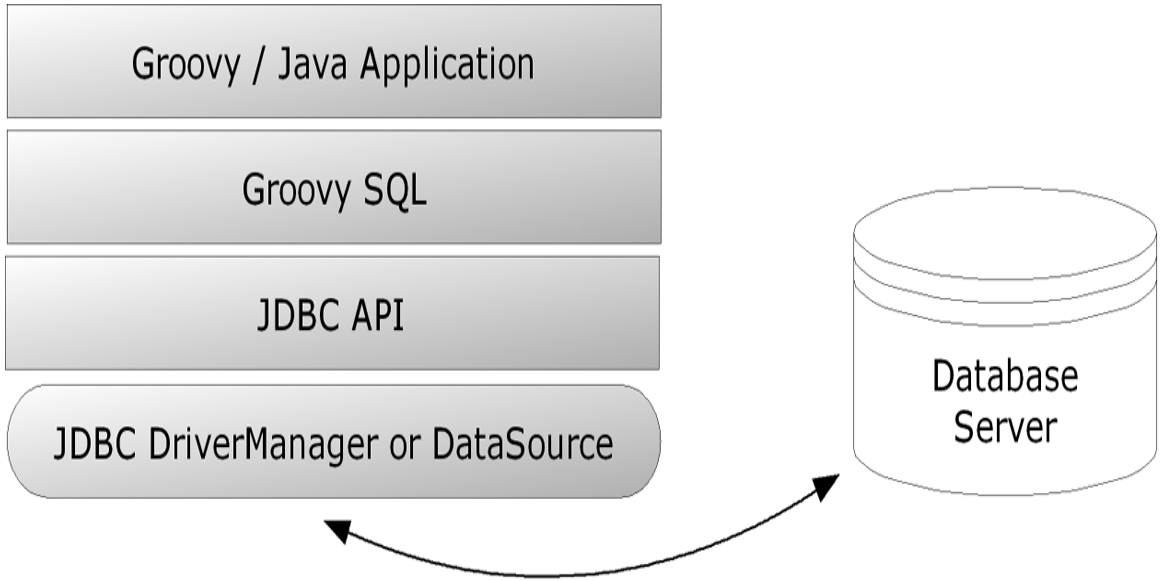
http://slideshare.net/paulk_asert/groovy-databases
https://github.com/paulk-asert/groovy-databases

Contents

- groovy.sql.Sql
  - Connecting to a database
  - Writing to a database
  - Reading from a database
  - Stored Procedures
  - Advanced Reading/Writing
- groovy.sql.DataSet
- MongoDB
- Neo4j

groovy.sql.Sql

- Groovy-friendly higher-level API sitting over JDBC



Connecting to a database…

- Sql.newInstance

```
import groovy.sql.Sql

def url = 'jdbc:hsqldb:mem:marathon'
def user = 'sa'
def password = ''
def driver = 'org.hsqldb.jdbcDriver'
def sql = Sql.newInstance(url, user, password, driver)

// use 'sql' instance

sql.close()
```

  - assumes driver is on the classpath
  - other **`newInstance`** variants available

…Connecting to a database…

Advanced

- Some variations

```
def sql = Sql.newInstance(
    url: 'jdbc:hsqldb:mem:marathon',
    user: 'sa',
    password: '',
    driver: 'org.hsqldb.jdbcDriver',
    cacheStatements: true,
    resultSetConcurrency: CONCUR_READ_ONLY
)
```

```
@Grab('org.hsqldb:hsqldb:2.3.3')
@GrabConfig(systemClassLoader=true)
import groovy.sql.Sql
// ...
```

```
Sql.withInstance(url, user, password, driver) {
    // use 'sql' instance
}
// no close() required
```

…Connecting to a database…

- new Sql() constructor

```
import groovy.sql.Sql
import org.hsqldb.jdbc.JDBCDataSource

def dataSource = new JDBCDataSource(
    database: 'jdbc:hsqldb:mem:marathon',
    user: 'sa', password: '')
def sql = new Sql(dataSource)

// use 'sql' instance

sql.close()
```

  - if you have a DataSource or existing Connection or Sql instance
  - Likely to be the **preferred approach with JDK9 Jigsaw**

…Connecting to a database

- new Sql() constructor (cont'd)

Advanced

```
@Grab('org.hsqldb:hsqldb:2.3.3')
@Grab('commons-dbcp:commons-dbcp:1.4')
import groovy.sql.Sql
import org.apache.commons.dbcp.BasicDataSource

def url = 'jdbc:hsqldb:mem:marathon'
def driver = 'org.hsqldb.jdbcDriver'
def dataSource = new BasicDataSource(
        driverClassName: driver, url: url,
        username: 'sa', password: '')
def sql = new Sql(dataSource)

// use 'sql' instance

sql.close()
```

Writing to a database…

```
def DDL = '''
    DROP   TABLE Athlete IF EXISTS;
    CREATE TABLE Athlete (
      athleteId   INTEGER GENERATED BY DEFAULT AS IDENTITY,
      firstname   VARCHAR(64),
      lastname    VARCHAR(64),
      dateOfBirth DATE,
    );
'''

sql.execute(DDL)

sql.execute '''
    INSERT INTO Athlete (firstname, lastname, dateOfBirth)
    VALUES ('Paul', 'Tergat', '1969-06-17')
'''
```

…Writing to a database

Advanced

```
def data = [first: 'Khalid', last: 'Khannouchi', birth: '1971-12-22']
sql.execute """
    INSERT INTO Athlete (firstname, lastname, dateOfBirth)
    VALUES (${data.first},${data.last},${data.birth})
"""

String athleteInsert = 'INSERT INTO Athlete (firstname, lastname) VALUES (?,?)'
def keys = sql.executeInsert athleteInsert, ['Ronaldo', 'da Costa']
assert keys[0] == [2]
def rowsUpdated = sql.executeUpdate '''
    UPDATE Athlete SET dateOfBirth='1970-06-07' where lastname='da Costa'
'''
assert rowsUpdated == 1

sql.execute "delete from Athlete where lastname = 'Tergat'"
```

Reading from a database…

```
sql.query('SELECT firstname, lastname FROM Athlete') { resultSet ->
    while (resultSet.next()) {
        print resultSet.getString(1)
        print ' '
        println resultSet.getString('lastname')
    }
}


sql.eachRow('SELECT firstname, lastname FROM Athlete') { row ->
    println row[0] + ' ' + row.lastname
}
```

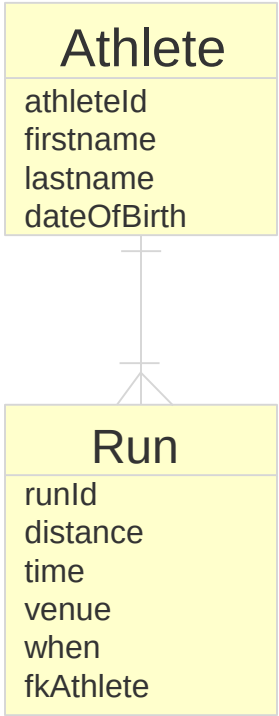
…Reading from a database

```
def first = sql.firstRow('SELECT lastname, dateOfBirth FROM Athlete')
def firstString = first.toMapString().toLowerCase()
assert firstString == '[lastname:tergat, dateofbirth:1969-06-17]'


List athletes = sql.rows('SELECT firstname, lastname FROM Athlete')
println "There are ${athletes.size()} Athletes:"
println athletes.collect { "$it.FIRSTNAME ${it[-1]}" }.join(", ")


assert sql.firstRow('SELECT COUNT(*) as num FROM Athlete').num == 3
```

Invoking Stored Procedures…

Advanced

```
def FULL_DLL = '''
    DROP   TABLE Athlete IF EXISTS;
    CREATE TABLE Athlete (
      athleteId   INTEGER GENERATED BY DEFAULT AS IDENTITY,
      firstname   VARCHAR(64),
      lastname    VARCHAR(64),
      dateOfBirth DATE
    );
    DROP   INDEX idx IF EXISTS;
    CREATE INDEX idx ON Athlete (athleteId);
    DROP   TABLE Run IF EXISTS;
    CREATE TABLE Run (
      runId       INTEGER GENERATED BY DEFAULT AS IDENTITY,
      distance    INTEGER,    -- in meters
      time        INTEGER,    -- in seconds
      venue       VARCHAR(64),
      when        TIMESTAMP,
      fkAthlete   INTEGER,
      CONSTRAINT fk FOREIGN KEY (fkAthlete)
        REFERENCES Athlete (athleteId) ON DELETE CASCADE
    );
    '''
sql.execute FULL_DLL
```



…Invoking Stored Procedures…

- And assume some data has been populated …

```
insertAthlete('Paul', 'Tergat', '1969-06-17')
insertAthlete('Khalid', 'Khannouchi', '1971-12-22')
insertAthlete('Ronaldo', 'da Costa', '1970-06-07')
insertRun(2, 4, 55, 'Berlin', '2003-09-28', 'Tergat')
insertRun(2, 5, 38, 'London', '2002-04-14', 'Khannouchi')
insertRun(2, 5, 42, 'Chicago', '1999-10-24', 'Khannouchi')
insertRun(2, 6, 05, 'Berlin', '1998-09-20', 'da Costa')
```

- We've refactored our previous insert code into some helper methods.

…Invoking Stored Procedures…

```
db.execute '''
  CREATE FUNCTION SELECT_ATHLETE_RUN ()
  RETURNS TABLE (lastname VARCHAR(64), venue VARCHAR(64), whenRun DATE)
  READS SQL DATA
  RETURN TABLE (
    select Athlete.lastname, Run.venue, Run.whenRun
    from Athlete, Run
    where Athlete.athleteId = Run.fkAthlete
    order by whenRun
  )
'''
db.eachRow('CALL SELECT_ATHLETE_RUN()') {
  println "$it.lastname $it.venue $it.whenRun"
}
```

```
da Costa Berlin 1998-09-20
Khannouchi Chicago 1999-10-24
Khannouchi London 2002-04-14
Tergat Berlin 2003-09-28
```

…Invoking Stored Procedures…

```
db.execute '''
  CREATE FUNCTION FULL_NAME (p_lastname VARCHAR(64))
  RETURNS VARCHAR(100)
  READS SQL DATA
  BEGIN ATOMIC
    declare ans VARCHAR(100);
    SELECT CONCAT(firstname, ' ', lastname) INTO ans
    FROM Athlete WHERE lastname = p_lastname;
    return ans;
  END
'''

assert db.firstRow("{? = call FULL_NAME(?)}", ['Tergat'])[0] == 'Paul Tergat'
```

…Invoking Stored Procedures

```
db.execute '''
  CREATE PROCEDURE CONCAT_NAME (OUT fullname VARCHAR(100),
    IN first VARCHAR(50), IN last VARCHAR(50))
  BEGIN ATOMIC
    SET fullname = CONCAT(first, ' ', last);
  END
'''

db.call("{call CONCAT_NAME(?, ?, ?)}", [Sql.VARCHAR, 'Paul', 'Tergat']) {
  fullname -> assert fullname == 'Paul Tergat'
}
```

Advanced Reading…

- Rowset metadata

```
def dump(sql, tablename) {
  println " CONTENT OF TABLE ${tablename} ".center(40, '-')
  sql.eachRow('SELECT * FROM ' + tablename) { rs ->
    def meta = rs.getMetaData()
    if (meta.columnCount <= 0) return
    for (i in 0..<meta.columnCount) {
      print "${i}: ${meta.getColumnLabel(i + 1)}".padRight(20)  // counts from 1
      print rs[i]?.toString()                                   // counts from 0
      print "\n"
    }
    println '-' * 40
  }
}
```

…Advanced Reading…

```
def dump(sql, tablename) {
  println " CONTENT OF TABLE ${tablename} ".center(40, '-')
  sql.eachRow('SELECT * FROM ' + tablename) { rs ->
    def meta = rs.getMetaData()
    if (meta.columnCount <= 0) return
    for (i in 0..<meta.columnCount) {
      print "${i}: ${meta.getColumnLabel(i + 1)}".padRight(20)   // counts from 1
      print rs[i]?.toString()                                    // counts from 0
      print "\n"
    }
    println '-' * 40
  }
}
```

```
---------------------- CONTENT OF TABLE Athlete ----------------------
ATHLETEID           FIRSTNAME           LASTNAME            DATEOFBIRTH
----------------------------------------------------------------------
1                   Paul                Tergat              1969-06-17
2                   Khalid              Khannouchi          1971-12-22
3                   Ronaldo             da Costa            1970-06-07
```

…Advanced Reading…

metadata closure

```
def dump2(sql, tablename) {
  def printColNames = { meta ->
    def width = meta.columnCount * 18
    println " CONTENT OF TABLE ${tablename} ".center(width, '-')
    (1..meta.columnCount).each {
      print meta.getColumnLabel(it).padRight(18)
    }
    println()
    println '-' * width
  }
  def printRow = { row ->
    row.toRowResult().values().each {
      print it.toString().padRight(18) }
    println()
  }
  sql.eachRow('SELECT * FROM ' + tablename, printColNames, printRow)
}
```

…Advanced Reading

- Pagination

```
qry = 'SELECT * FROM Athlete'
assert sql.rows(qry, 1, 4)*.lastname ==
        ['Tergat', 'Khannouchi', 'da Costa', 'Gebrselassie']
assert sql.rows(qry, 5, 4)*.lastname ==
        ['Makau', 'Radcliffe', 'Ndereba', 'Takahashi']
assert sql.rows(qry, 9, 4)*.lastname ==
        ['Loroupe', 'Kristiansen']
```

Advanced Writing…

Advanced

- Simple transaction support

```
sql.withTransaction {
  insertAthlete('Haile', 'Gebrselassie', '1973-04-18')
  insertAthlete('Patrick', 'Makau', '1985-03-02')
}
```

…Advanced Writing…

- Batching of ad-hoc SQL statements

```
sql.execute "delete from Athlete where lastname = 'Tergat'"
sql.withBatch { stmt ->
  stmt.addBatch '''
    INSERT INTO Athlete (firstname, lastname, dateOfBirth)
    VALUES ('Paul',   'Tergat', '1969-06-17')'''
  stmt.addBatch """
    INSERT INTO Run (distance, time, venue, when, fkAthlete)
      SELECT 42195, ${2*60*60+4*60+55}, 'Berlin', '2003-09-28',
      athleteId FROM Athlete WHERE lastname='Tergat'"""
}
//22/04/2013 6:34:59 AM groovy.sql.BatchingStatementWrapper processResult
//FINE: Successfully executed batch with 2 command(s)
```

…Advanced Writing…

- Batching with a prepared statement

```
def qry = 'INSERT INTO Athlete (firstname, lastname, dateOfBirth) VALUES (?,?,?);'
sql.withBatch(3, qry) { ps ->
  ps.addBatch('Paula', 'Radcliffe', '1973-12-17')
  ps.addBatch('Catherine', 'Ndereba', '1972-07-21')
  ps.addBatch('Naoko', 'Takahashi', '1972-05-06')
  ps.addBatch('Tegla', 'Loroupe', '1973-05-09')
  ps.addBatch('Ingrid', 'Kristiansen', '1956-03-21')
}
// If logging is turned on:
// 20/04/2013 2:18:10 AM groovy.sql.BatchingStatementWrapper processResult
// FINE: Successfully executed batch with 3 command(s)
// 20/04/2013 2:18:10 AM groovy.sql.BatchingStatementWrapper processResult
// FINE: Successfully executed batch with 2 command(s)
```

…Advanced Writing

- Named parameters (two variants) and named-ordinal parameters

```
@Canonical class Athlete { String first, last, dob }

def ndereba = new Athlete('Catherine', 'Ndereba', '1972-07-21')
def takahashi = new Athlete('Naoko', 'Takahashi')
def takahashiExtra = [dob: '1972-05-06']
def loroupe = [first: 'Tegla', last: 'Loroupe', dob: '1973-05-09']

def insertPrefix = 'INSERT INTO Athlete (firstname, lastname, dateOfBirth) VALUES '
sql.execute insertPrefix + ' (?.first,?.last,?.dob)', ndereba
sql.execute insertPrefix + ' (?1.first,?1.last,?2.dob)', takahashi, takahashiExtra
sql.execute insertPrefix + ' (:first,:last,:dob)', loroupe
sql.execute insertPrefix + ' (:first,:last,:dob)', first: 'Ingrid',
    last: 'Kristiansen', dob: '1956-03-21'
```

Topics

- groovy.sql.Sql
  - *Connecting to a database*
  - *Writing to a database*
  - *Reading from a database*
  - *Stored Procedures*
  - *Advanced Reading/Writing*
- ➢ **groovy.sql.DataSet**
- MongoDB
- Neo4j

groovy.sql.DataSet

- Allows any table within an RDBMS to appear as if it was a Java-like collection of POGOs
- Can be thought of as a poor man's object-relational mapping system
- Has some smarts for lazy execution of database queries
- Not meant to be as sophisticated as hibernate

DataSet example…

```
def athletes = sql.dataSet('Athlete')
athletes.each { println it.firstname }
athletes.add(
    firstname: 'Paula',
    lastname: 'Radcliffe',
    dateOfBirth: '1973-12-17'
)
athletes.each { println it.firstname }


def runs = sql.dataSet('AthleteRun').findAll { it.firstname == 'Khalid' }
runs.each { println "$it.lastname $it.venue" }
```

```
Paul
Khalid
Ronaldo
Paul
Khalid
Ronaldo
Paula
Khannouchi London
Khannouchi Chicago
```

…DataSet example

```
select * from Athlete where firstname >= ? and dateOfBirth > ? order by dateOfBirth DESC
[P, 1970-01-01]
[Paula, Ronaldo]
```

```
def query = athletes.findAll { it.firstname >= 'P' }
query = query.findAll { it.dateOfBirth > '1970-01-01' }
query = query.sort { it.dateOfBirth }
query = query.reverse()
println query.sql
println query.parameters
println query.rows()*.firstname // One SQL query here!
```

Overcoming groovy.sql.DataSet limitations

```
def DLL_WITH_VIEW = '''
    DROP   TABLE Athlete IF EXISTS;
    CREATE TABLE Athlete (
      athleteId   INTEGER GENERATED BY DEFAULT AS IDENTITY,
      firstname   VARCHAR(64),
      lastname    VARCHAR(64),
      dateOfBirth DATE
    );
    DROP   INDEX idx IF EXISTS;
    CREATE INDEX idx ON Athlete (athleteId);
    DROP   TABLE Run IF EXISTS;
    CREATE TABLE Run (
      runId       INTEGER GENERATED BY DEFAULT AS IDENTITY,
      distance    INTEGER,    -- in meters
      time        INTEGER,    -- in seconds
      venue       VARCHAR(64),
      when        TIMESTAMP,
      fkAthlete   INTEGER,
      CONSTRAINT fk FOREIGN KEY (fkAthlete)
        REFERENCES Athlete (athleteId) ON DELETE CASCADE
    );
    DROP   VIEW AthleteRun IF EXISTS;
    CREATE VIEW AthleteRun AS
      SELECT * FROM Athlete LEFT OUTER JOIN Run
        ON fkAthlete=athleteId;
  '''
db.execute DLL_WITH_VIEW
```

Topics

- groovy.sql.Sql
  - *Connecting to a database*
  - *Writing to a database*
  - *Reading from a database*
  - *Stored Procedures*
  - *Advanced Reading/Writing*
- groovy.sql.DataSet
- ➢ **MongoDB**
- Neo4j

MongoDB…

```
@Grab('com.gmongo:gmongo:1.3')
import com.gmongo.GMongo
import com.mongodb.util.JSON
import groovy.transform.Field

@Field db = new GMongo().getDB('athletes')
db.athletes.drop()
db.athletes << [first: 'Paul', last: 'Tergat', dob: '1969-06-17', runs: [
    [distance: 42195, time: 2 * 60 * 60 + 4 * 60 + 55,
        venue: 'Berlin', when: '2003-09-28']
]]

def insertAthlete(first, last, dob) {
  db.athletes << [first: first, last: last, dob: dob]
}
```

…MongoDB…

```
def insertRun(h, m, s, venue, date, lastname) {
  db.athletes.update(
      [last: lastname],
      [$addToSet: [runs: [distance: 42195,
          time: h * 60 * 60 + m * 60 + s,
          venue: venue, when: date]]]
  )
}

insertAthlete('Khalid', 'Khannouchi', '1971-12-22')
insertAthlete('Ronaldo', 'da Costa', '1970-06-07')

insertRun(2, 5, 38, 'London', '2002-04-14', 'Khannouchi')
insertRun(2, 5, 42, 'Chicago', '1999-10-24', 'Khannouchi')
insertRun(2, 6, 05, 'Berlin', '1998-09-20', 'da Costa')
```

…MongoDB…

```
516b15fc2b10a15fa09331f2 Tergat 1
516b15fc2b10a15fa09331f3 Khannouchi 2
516b15fc2b10a15fa09331f4 da Costa 1
516b15fc2b10a15fa09331f5 Radcliffe 1
```

```
def radcliffe = """{
    first: 'Paula',
    last: 'Radcliffe',
    dob: '1973-12-17',
    runs: [
        {distance: 42195, time: ${2 * 60 * 60 + 15 * 60 + 25},
            venue: 'London', when: '2003-04-13'}
    ]
}"""

db.athletes << JSON.parse(radcliffe)

assert db.athletes.count == 4
db.athletes.find().each {
  println "$it._id $it.last ${it.runs.size()}"
}
```

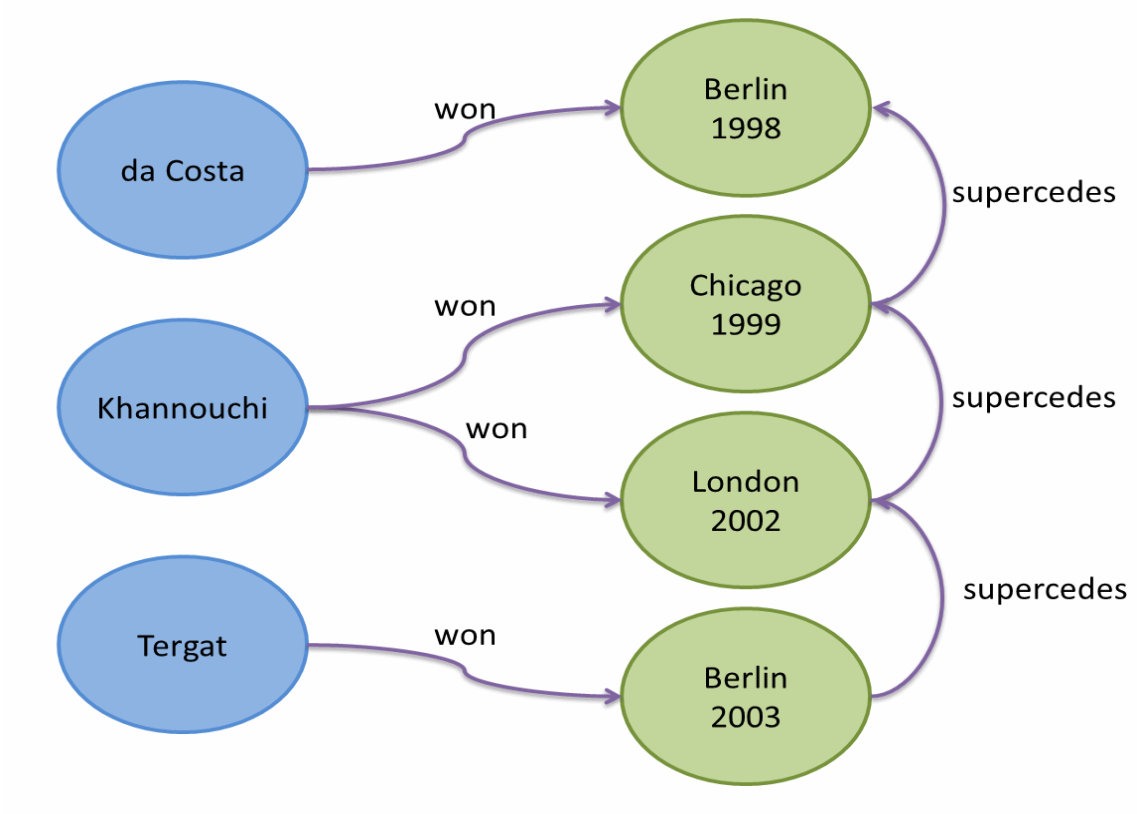
…MongoDB

```
def londonAthletes = db.athletes.find('runs.venue': 'London')*.first
assert londonAthletes == ['Khalid', 'Paula']



def youngAthletes = db.athletes.aggregate(
    [$project: [first: 1, dob: 1]],
    [$match: [dob: [$gte: '1970-01-01']]],
    [$sort: [dob: -1]]
)

assert youngAthletes.results()*.first == ['Paula', 'Khalid', 'Ronaldo']
```

Topics

- groovy.sql.Sql
  - *Connecting to a database*
  - *Writing to a database*
  - *Reading from a database*
  - *Stored Procedures*
  - *Advanced Reading/Writing*
- groovy.sql.DataSet
- MongoDB
- ➢ **Neo4j**

Neo4j…


da Costa
won
Berlin
1998
supercedes
Khannouchi
won
Chicago
1999
supercedes
won
London
2002
supercedes
Tergat
won
Berlin
2003

…Neo4j…

```
@Grab('org.neo4j:neo4j-kernel:2.1.4')
@Grab('com.tinkerpop.gremlin:gremlin-groovy:2.5.0')
//@Grab('com.tinkerpop.blueprints:blueprints-neo4j-graph:2.5.0')
@Grab('com.tinkerpop.blueprints:blueprints-neo4j-graph:2.5.0;transitive=false')
@Grab('com.tinkerpop.blueprints:blueprints-core:2.5.0')
//@Grab('codehaus-stax:stax:1.1.1')
//@GrabResolver('https://repository.jboss.org/nexus/content/repositories/thirdparty-releases')
@GrabExclude('org.codehaus.groovy:groovy')
import com.tinkerpop.blueprints.Graph
import com.tinkerpop.blueprints.impls.neo4j.Neo4jGraph
//import com.tinkerpop.blueprints.util.io.graphml.GraphMLWriter
import com.tinkerpop.gremlin.groovy.Gremlin
import groovy.transform.Field
import org.neo4j.graphdb.traversal.Evaluators
import org.neo4j.kernel.EmbeddedGraphDatabase
import org.neo4j.graphdb.*
import org.neo4j.kernel.Traversal
import org.neo4j.kernel.Uniqueness

@Field graphDb = new EmbeddedGraphDatabase("athletes")

enum MyRelationshipTypes implements RelationshipType { ran, supercedes }
```

…Neo4j…

```
// some optional metaclass syntactic sugar
EmbeddedGraphDatabase.metaClass {
  createNode { Map properties ->
    def n = delegate.createNode()
    properties.each { k, v -> n[k] = v }
    n
  }
}
Node.metaClass {
  propertyMissing { String name, val -> delegate.setProperty(name, val) }
  propertyMissing { String name -> delegate.getProperty(name) }
  methodMissing { String name, args ->
    delegate.createRelationshipTo(args[0], MyRelationshipTypes."$name")
  }
}
Relationship.metaClass {
  propertyMissing { String name, val -> delegate.setProperty(name, val) }
  propertyMissing { String name -> delegate.getProperty(name) }
}
```

…Neo4j…

```
def insertAthlete(first, last, dob) {
  graphDb.createNode(first: first, last: last, dob: dob)
}

def insertRecord(h, m, s, venue, when, athlete) {
  def record = graphDb.createNode(distance: 42195,
      time: h * 60 * 60 + m * 60 + s, venue: venue, when: when)
  athlete.won(record)
  record
}

Gremlin.load()

def tx = graphDb.beginTx()
def athlete1, athlete2, athlete3, athlete4
def marathon1, marathon2a, marathon2b, marathon3, marathon4a, marathon4b
```

…Neo4j…

```
try {
  athlete1 = graphDb.createNode()
  athlete1.first = 'Paul'
  athlete1.last = 'Tergat'
  athlete1.dob = '1969-06-17'
  marathon1 = graphDb.createNode()
  marathon1.distance = 42195
  marathon1.time = 2 * 60 * 60 + 4 * 60 + 55
  marathon1.venue = 'Berlin'
  marathon1.when = '2003-09-28'
  athlete1.won(marathon1)

  athlete2 = insertAthlete('Khalid', 'Khannouchi', '1971-12-22')
  marathon2a = insertRecord(2, 5, 38, 'London', '2002-04-14', athlete2)
  marathon2b = insertRecord(2, 5, 42, 'Chicago', '1999-10-24', athlete2)

  athlete3 = insertAthlete('Ronaldo', 'da Costa', '1970-06-07')
  marathon3 = insertRecord(2, 6, 5, 'Berlin', '1998-09-20', athlete3)

  athlete4 = insertAthlete('Paula', 'Radcliffe', '1973-12-17')
  marathon4a = insertRecord(2, 17, 18, 'Chicago', '2002-10-13', athlete4)
  marathon4b = insertRecord(2, 15, 25, 'London', '2003-04-13', athlete4)
```

…Neo4j…

```
def venue = marathon1.venue
def when = marathon1.when
println "$athlete1.first $athlete1.last won the $venue marathon on $when"

def allAthletes = [athlete1, athlete2, athlete3, athlete4]
def wonInLondon = allAthletes.findAll { athlete ->
  athlete.getRelationships(MyRelationshipTypes.won).any { record ->
    record.getOtherNode(athlete).venue == 'London'
  }
}
assert wonInLondon*.last == ['Khannouchi', 'Radcliffe']

marathon2b.supercedes(marathon3)
marathon2a.supercedes(marathon2b)
marathon1.supercedes(marathon2a)
marathon4b.supercedes(marathon4a)
```

…Neo4j…

```
println "World records following $marathon3.venue $marathon3.when:"
def t = new Traversal()
for (Path p in t.description().breadthFirst().
    relationships(MyRelationshipTypes.supercedes).
    evaluator(Evaluators.fromDepth(1)).
    uniqueness(Uniqueness.NONE).
    traverse(marathon3)) {
  def newRecord = p.endNode()
  println "$newRecord.venue $newRecord.when"
}

Graph g = new Neo4jGraph(graphDb)
def pretty = { it.collect { "$it.venue $it.when" }.join(', ') }
def results = []
g.V('venue', 'London').fill(results)
println 'London world records: ' + pretty(results)

results = []
g.V('venue', 'London').in('supercedes').fill(results)
println 'World records after London: ' + pretty(results)
```

…Neo4j

```
  results = []
  def emitAll = { true }
  def forever = { true }
  def berlin98 = { it.venue == 'Berlin' && it.when.startsWith('1998') }
  g.V.filter(berlin98).in('supercedes').loop(1, forever, emitAll).fill(results)
  println 'World records after Berlin 1998: ' + pretty(results)
//    def writer = new GraphMLWriter(g)
//    def out = new FileOutputStream("c:/temp/athletes.graphml")
//    writer.outputGraph(out)
//    writer.setNormalize(true)
//    out.close()

  tx.success()
} finally {
  tx.finish()
  graphDb.shutdown()
}
```

More Information: Groovy in Action, 2ed