Groovy & Grails: Desarrollo rápido de aplicaciones

Sesión 4: Librerías propias de Groovy

Librerías propias de Groovy

- Groovy Builders
- DSL's
- Tratamiento de archivos XML
- Bases de datos

Groovy Builders

- Los Builders facilitan el trabajo con determinadas tareas complejas
  - Generación de archivo XML y HTML con **MarkupBuilder**
  - Automatización de tareas con **AntBuilder**

MarkupBuilder

- Permite generar archivos XML de forma rápida y sencilla
- En ocasiones el código fuente para la creación de estos archivos es incomprensible para quien no lo ha creado
- Con este Builder el código fuente está indentado prácticamente igual que el archivo XML creado

MarkupBuilder

- El siguiente archivo en Groovy

```
writer = new StringWriter()
builder = new groovy.xml.MarkupBuilder(writer)
facturas = builder.facturas {
    for (dia in 1..3) {
        factura(fecha: new Date(106,0,dia)) {
            item(id:dia){
                producto(nombre: 'Teclado', euros:876)
            }
        }
    }
}
```

MarkupBuilder

- Generaría el siguiente archivo XML

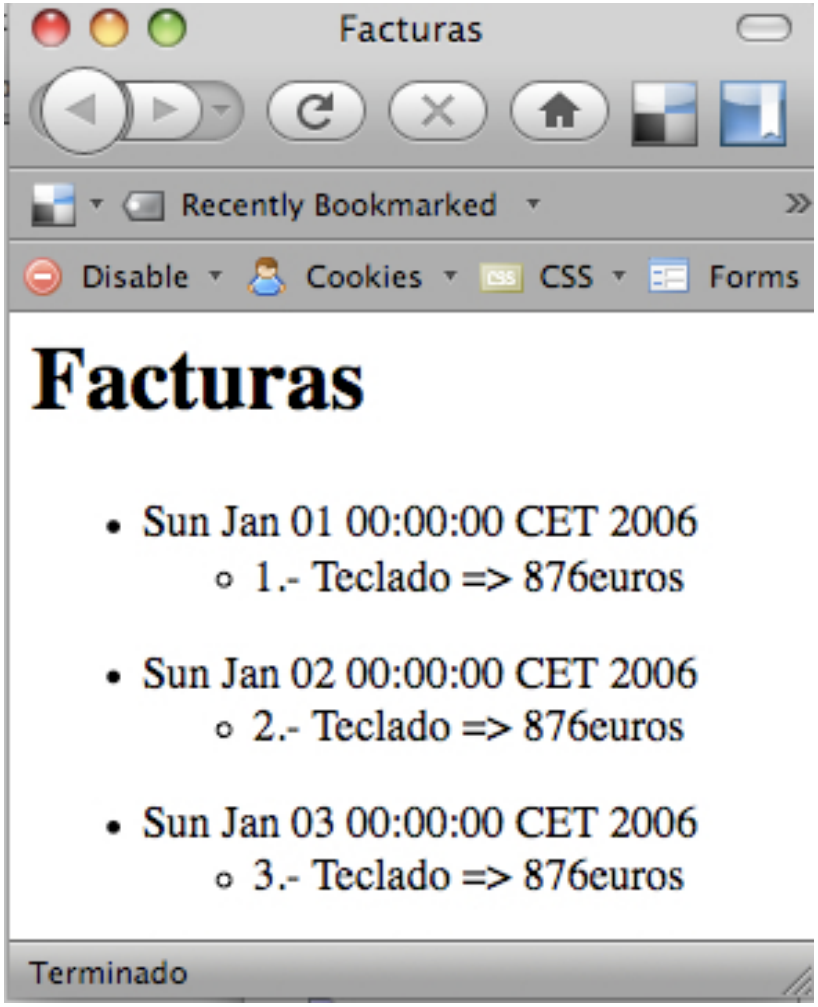
```
<facturas>
        <factura fecha='Sun Jan 01 00:00:00 CET 2006'>
                <item id='1'>
                        <producto nombre='Teclado' euros='876' />
                </item>
        </factura>
        <factura fecha='Mon Jan 02 00:00:00 CET 2006'>
                <item id='2'>
                        <producto nombre='Teclado' euros='876' />
                </item>
        </factura>…..
</facturas>
```

MarkupBuilder

- El código fuente sigue el mismo indentado que el archivo XML generado
- MarkupBuilder también puede generar HTML

MarkupBuilder

- Para generar la siguiente página HTML



MarkupBuilder

```
def writer = new StringWriter()
def builder = new groovy.xml.MarkupBuilder()
builder.html {
    head {
        title 'Facturas'
    }
    body {
        h1 'Facturas'
        for (dia in 1..3){
            ul{
                li new Date(106,0,dia).toString()
                ul {
                    li "$dia.- Teclado => 876euros"
                }
            ....
```

AntBuilder

- Utilizado para la automatización de tareas
  - Manipulación del sistema de ficheros
  - Compilación de ficheros fuente
  - Ejecución de pruebas

AntBuilder

- En ocasiones, las librerías para la automatización de tareas con Ant en Java se complican demasiado

- El siguiente ejemplo utilizando Ant elimina todo los archivos de un directorio destino y copia los archivos .doc en otro directorio, obviando los archivo temporales.

AntBuilder

```
<project name="prepararDirectorioLibros" default="copy">
        <property name="destino.dir" value="destino"/>
        <property name="capitulos.dir" value="capitulos"/>

        <target name="copy">
                <delete dir="${destino.dir}" />
                <copy todir="${destino.dir}">
                        <fileset dir="${capitulos.dir}"
                                includes="*.doc"
                                excludes="~*" />
                </copy>
        </target>
</project>
```

AntBuilder

- En Groovy ya no tenemos que utiliza la sintaxis de Ant tipo XML

```
DESTINO_DIR = 'destino'
CAPITULOS_DIR = 'capitulos'
ant = new AntBuilder()

ant.delete(dir:DESTINO_DIR)
ant.copy(todir:DESTINO_DIR){
        fileset(dir:CAPITULOS_DIR, includes:'*.doc', excludes:'~*')
}
```

AntBuilder

- Para convertir archivos en formato Ant para que funcionen con AntBuilder, debemos seguir las siguientes reglas:
  - Las tareas en Ant se corresponden con métodos en AntBuilder
  - Los atributos de Ant se pasan en forma de mapa a los métodos de AntBuilder
  - Donde en Ant era obligatorio utilizar cadenas de texto para los valores (valor:”1”, analizado:”true”), con AntBuilder es posible utilizar el correspondiente tipo de dato (valor:1, analizado:true)

DSL's

- Significan Domain Specific Language (Lenguaje Específico de Dominio)
- Constituyen lenguajes de programación reducido que permite representar el conocimiento de un campo específico
- DSL's típicos: reglas financieras o bases de datos

DSL's

- Groovy es un buen lenguaje para escribir DSL's por:
  - Existencia de Builders para crear nuevas estructuras
  - Nuevos métodos y propiedades con las categorías y las metaclases para tener cosas como 4.veces o 24.horas
  - La mayoría de los operadores se pueden sobrecargar para tener *totalMinutos = 4.dias + 12.horas*

DSL's

- Groovy es un buen lenguaje para escribir DSL's por:
  - Utilizar mapas como parámetros como por ejemplo *mover(x:500.metros, y:2.kilometros)*
  - Nuevas estructuras de control con los closures como por ejemplo *siFaltaMenosDeUnDia {…}*

DSL's

- Un ejemplo de DSL puede ser GORM, el framework de persistencia basado en Hibernate que utiliza Grails.

- Podemos utilizar GORM sin haber visto en nuestra vida Hibernate simplemente conociendo el modelo de datos

DSL's

```
class Persona{
        Date fechaNacimiento
        String nombre
        String apellidos
}

//Para acceder a los datos de las Personas con GORM se haría así
def lista = Persona.findByNombre("Manuel")
def lista = Persona.findByNombreNotNull()
def lista = Persona.findByNombreLike("Ma%")
def lista = Persona.findByFechaNacimientoBetween(fecha1,fecha2)
def lista = Persona.findByNombreLikeOrApellidoLike("M%", "L%")
```

DSL's

- Con GORM el programador no necesita conocer la sintaxis SQL y puede centrarse en la lógica de negocio

DSL's

- Conversión de unidades de longitud mediante un DSL
- Cualquier personas sin conocimientos de Groovy pueda escribir unas líneas de código
- Se deben permitir algunas operaciones entre varias unidades (metros, kilómetros, centímetros, etc)
- Crearemos la clase *Distancia* y sobrecargaremos los operadores + y -

DSL's

```
class Distancia implements Comparable {
        BigDecimal longitud
        Unidad unidad
        Distancia plus(Distancia operando) {
                def nuevaLongitud = this.longitud +
Unidad.convertirUnidad(operando, this.unidad)
                new Distancia(longitud : nuevaLongitud, unidad : this.unidad)
        }
        Distancia minus(Distancia operando) {
                def nuevaLongitud = this.longitud -
Unidad.convertirUnidad(operando, this.unidad)
                new Distancia(longitud : nuevaLongitud, unidad : this.unidad)
        }
        ....
}
```

DSL's

```
class Distancia implements Comparable {
        ....
        int compareTo(otro) {
          if(this.unidad == otro.unidad)
              return this.longitud <=> otro.longitud
          return this.longitud <=> Unidad.convertirUnidad(otro, this.unidad)
        }

        String toString() {
                "$longitud $unidad.nombre"
        }
}
```

DSL's

- El siguiente paso será crear la clase *Unidad* que será la encargada de convertir las unidades para que puedan ser operadas entre si

DSL's

```
class Unidad {
        def ratio
        String nombre

        static def convertirUnidad(Distancia d, Unidad nuevaUnidad) {
            def factor = ratioTabla[d.unidad.ratio][nuevaUnidad.ratio]
            if(factor)
                return d.longitud * factor
            else
                return d.longitud / ratioTabla[nuevaUnidad.ratio][d.unidad.ratio]
        }
        ....
}
```

DSL's

```
class Unidad {
    ....
    static ratioTabla = [
            //       mm,        cm,         m,        km,    y, mi
            [        1,         0,          0,         0,    0, 0 ],    // mm
            [       10,         1,          0,         0,    0, 0 ],    // cm
            [      1e3,       1e2,          1,         0,    0, 0 ],    //  m
            [      1e6,       1e5,       1e3,         1,    0, 0 ],    // km
            [    914.4,     91.44,    0.9144, 0.9144e-3,    1, 0 ],    // yd
            [ 1.609344e6, 1.609344e5, 1.609344e3,  1.609344, 1760, 1 ],    // mi
    ]
    ....
}
```

DSL's

```
class Unidad {
    ....
   public static final mm = new Unidad(ratio : 0, nombre : "milímetros")
   public static final cm = new Unidad(ratio : 1, nombre : "centímetros")
   public static final  m = new Unidad(ratio : 2, nombre : "metros")
   public static final km = new Unidad(ratio : 3, nombre : "kilometros")
   public static final yd = new Unidad(ratio : 4, nombre : "yarda")
   public static final mi = new Unidad(ratio : 5, nombre : "milla(s)")
}
```

DSL's

- Por último, necesitamos crear una *categoría* que implemente los métodos *get()* para los tipos de datos *Number* y *Distancia*

- Al hacer esto con *Number* vamos a poder utilizar la notación 4.3.mi

- Al hacerlo con la clase *Distancia* utilizaremos 4.3.mi.km para hacer conversiones al vuelo

DSL's

```
class DistanciaCategoria {
    static Distancia getMm(Number n) {
            new Distancia(longitud : n, unidad : Unidad.mm)
    }
    static Distancia getMm(Distancia d) {
            new Distancia(longitud : Unidad.convertirUnidad(d,
Unidad.mm), unidad : Unidad.mm)
    }
    static Distancia getCm(Number n) {
            new Distancia(longitud : n, unidad : Unidad.cm)
    }
            ....
}
```

DSL's

```
class DistanciaCategoria {
    ....
    static Distancia getCm(Distancia d) {
            new Distancia(longitud : Unidad.convertirUnidad(d,
Unidad.cm), unidad : Unidad.cm)
    }
    static Distancia getM(Number n) {
            new Distancia(longitud : n, unidad : Unidad.m)
    }
    static Distancia getM(Distancia d) {
            new Distancia(longitud : Unidad.convertirUnidad(d,
Unidad.m), unidad : Unidad.m)
    }
    ....
}
```

DSL's

```
class DistanciaCategoria {
    ....
    static Distancia getM(Distancia d) {
            new Distancia(longitud : Unidad.convertirUnidad(d,
Unidad.m), unidad : Unidad.m)
    }
    static Distancia getKm(Number n) {
            new Distancia(longitud : n, unidad : Unidad.km)
    }
    static Distancia getKm(Distancia d) {
            new Distancia(longitud : Unidad.convertirUnidad(d,
Unidad.km), unidad : Unidad.km)
    }
    ....
}
```

DSL's

```
class DistanciaCategoria {
    ....
    static Distancia getYd(Number n) {
            new Distancia(longitud : n, unidad : Unidad.yd)
    }
    static Distancia getYd(Distancia d) {
            new Distancia(longitud : Unidad.convertirUnidad(d,
Unidad.yd), unidad : Unidad.yd)
    }
    static Distancia getMi(Number n) {
            new Distancia(longitud : n, unidad : Unidad.mi)
    }
    ....
}
```

DSL's

```
class DistanciaCategoria {
    ....
    static Distancia getMi(Distancia d) {
            new Distancia(longitud : Unidad.convertirUnidad(d,
Unidad.mi), unidad : Unidad.mi)
    }
}
```

DSL's

- Sólo nos queda probar que todo funciona correctamente

DSL's

```
use(DistanciaCategoria.class) {
        def d1 = 1.m
        def d2 = 1.yd
        def d3 = 1760.yd
        def d4 = 100.cm
        println d1 + 1.yd
        println 1.yd + 1.mi
        println 1.m - 1.yd
        println d2.m
        println d3.mi
        println d4.m
        println 1000.yd.km
        println 1000.yd
}
```

Tratamiento de archivos XML

- Groovy dispone de tres métodos para la lectura de archivos XML
  - Rutas del DOM y XmlParser y XmlSlurper
  - Eventos SAX
  - Analizar StAX

Tratamiento de archivos XML

```
<plan>
        <semana maximo="10">
                <tarea prevision="3" hechas="3" titulo="Aserciones"/>
                <tarea prevision="4" hechas="4" titulo="GroovyBeans"/>
                <tarea prevision="2" hechas="1" titulo="Librería GString"/>
        </semana>
        <semana maximo="10">
                <tarea prevision="3" hechas="0" titulo="Groovy Builders"/>
                <tarea prevision="2" hechas="0" titulo="GORM"/>
                <tarea prevision="2" hechas="0" titulo="DSLs"/>
        </semana>
</plan>
```

Tratamiento de archivos XML

- Con XmlParser accederemos a los nodos del archivo XML como si lo hiciéramos sobre un array
- Los atributos serán accesibles por medio del operador @

Tratamiento de archivos XML

```
def plan = new XmlParser().parse(new File('datos/plan.xml'))

assert 'plan' == plan.name()
assert 'semana' == plan.semana[0].name()
assert 'tarea' == plan.semana[0].tarea[0].name()
assert 'GroovyBeans' == plan.semana[0].tarea[1].@titulo
```

Tratamiento de archivos XML

- *XmlParser* forma parte del paquete *groovy.util*, con lo que no es necesario importarlo
- El analizador devuelve objetos de tipo *groovy.util.Node*
- Groovy también tiene la clase *XmlSlurper*
- *XmlParser* y *XmlSlurper* tienen el método *parse()*

Tratamiento de archivos XML

- Método *parse()* con diferentes parámetros

Método	Comentario
parse(InputSource is)	Lee un objeto de la clase *org.xml.sax.InputSource*
parse(File file)	Lee un objeto de la clase *java.io.File*
parse((InputStream is)	Lee un objeto de la clase *java.io.InputStream*
parse(Reader r)	Lee un objeto de la clase *java.io.Reader*
parse(String uri)	Lee el contenido apuntado por la dirección *uri*
parse(String text)	Utiliza el parámetro *texto* como entrada

Tratamiento de archivos XML

- *XmlParser* devuelve objetos del tipo *groovy.util.Node, mientras que XmlSlurper lo hace del tipo GPathResult*

Tratamiento de archivos XML

- Métodos comunes de ambas clases

Método XmlParser	Método XmlSlurper
Object name()	String name()
String text()	String text()
String toString()	String toString()
Node parent()	GPathResult parent()
List children()	GPathResult children()
Map attributes()	Map attributes()
Iterator iterator()	Iterator iterator()
List depthFirst()	Iterator depthFirst()
List breadthFirst()	Iterator breadthFirst()

Tratamiento de archivos XML

- Acceso a los elementos y los atributos

Node(XmlParser)	GPathResult(XmlSlurper)
[‘nombreElemento’] .nombreElemento	[‘nombreElemento’] .nombreElemento
[indice]	[indice]
[‘@nombreAtributo’] .’@nombreAtributo’	[‘@nombreAtributo’] .’@nombreAtributo’ .@nombreAtributo

Tratamiento de archivos XML

```
def node = new XmlParser().parse(new File('datos/plan.xml'))
def gpath = new XmlSlurper().parse(new File('datos/plan.xml'))

assert 'plan' == node.name()
assert 'plan' == gpath.name()

assert 2 == node.children().size()
assert 2 == gpath.children().size()

assert 6 == node.semana.tarea.size()
assert 6 == gpath.semana.tarea.size()

assert 8 == node.semana.tarea.'@hechas'*.toInteger().sum()

assert gpath.semana[1].tarea.every{ it.'@hechas' == '0'}
```

Tratamiento de archivos XML

- Con ejemplos sencillos, ambas clases tienen rendimientos similares
- XmlParser necesita almacenar en memoria el contenido del archivo xml
- XmlSlurper realiza el procesamiento sin almacenar en memoria
- Con ejemplos complejos, XmlSlurper funciona mejor al no consumir memoria que no se va a utilizar

Bases de datos

- Trabajar directamente con SQL
- Trabajar con DataSets
- GORM (Groovy ORM)

Trabajar directamente con SQL

- Conexión con la base de datos
  - Url de la base de datos
  - Usuario
  - Contraseña
  - Controlador

```
import groovy.sql.Sql
db = Sql.newInstance(
    'jdbc:hsqldb:mem:Biblioteca',
    'sa',
    '',
    'org.hsqldb.jdbcDriver')
```

Trabajar directamente con SQL

- Ejecutar sentencias SQL sobre la base de datos

```
db.execute '''
    DROP TABLE Escritores IF EXISTS;
    CREATE TABLE Escritores (
        idEscritor    INTEGER GENERATED BY DEFAULT as IDENTITY,
        nombre    VARCHAR(128),
        apellidos    VARCHAR (128),
        fechaNacimiento    DATE
    );
            '''
```

Trabajar directamente con SQL

- Insertar datos

```
db.execute '''
    INSERT INTO Escritores (nombre, apellidos, fechaNacimiento)
        VALUES ('Camilo José', 'Cela Trulock', '1916-05-11');
    INSERT INTO Escritores (nombre, apellidos, fechaNacimiento)
        VALUES ('Miguel', 'de Cervantes Saavedra', '1547-09-29');
    INSERT INTO Escritores (nombre, apellidos, fechaNacimiento)
        VALUES ('Miguel', 'Hernández Gilabert', '1910-10-30');
    INSERT INTO Escritores (nombre, apellidos, fechaNacimiento)
        VALUES ('Felix', 'Lope de Vega y Carpio', '1562-11-25');
    '''
```

Trabajar directamente con SQL

- Insertar datos con *preparedStatement*

```
String insertaEscritores = '''
    INSERT INTO Escritores (nombre, apellidos, fechaNacimiento)
        VALUES (?,?,?);
    '''
db.execute insertaEscritores, ['Camilo José', 'Cela Trulock', '1916-05-11']
db.execute insertaEscritores, ['Miguel', 'de Cervantes Saavedra', '1547-09-29']
db.execute insertaEscritores, ['Miguel', 'Hernández Gilabert', '1910-10-30']
db.execute insertaEscritores, ['Felix', 'Lope de Vega y Carpio', '1562-11-25']
```

Trabajar directamente con SQL

- Insertar datos con un closure

```
def escritores = [
  [nombre:'Camilo José', apellidos:'Cela Trulock', fechaNacimiento:'1916-05-11'],
  [nombre:'Miguel', apellidos:'de Cervantes', fechaNacimiento:'1547-09-29'],
  [nombre:'Miguel', apellidos:'Hernández', fechaNacimiento:'1910-10-30'],
  [nombre:'Felix', apellidos:'Lope de Vega', fechaNacimiento:'1562-11-25']
]
escritores.each{
        db.execute """
          INSERT INTO Escritores (nombre, apellidos, fechaNacimiento)
          VALUES (${it.nombre}, ${it.apellidos}, ${it.fechaNacimiento});
                """
}
```

Trabajar directamente con SQL

- Eliminar datos

```
db.execute '''
    DELETE FROM Escritores WHERE apellidos = 'Hernández Gilabert'
    '''
```

Trabajar directamente con SQL

- Actualizar datos

```
db.execute '''
    UPDATE Escritores SET nombre='Felix' WHERE nombre='Félix'
    '''
```

Trabajar directamente con SQL

- Actualizar datos con *executeUpdate*

```
db.executeUpdate '''
    UPDATE Escritores SET nombre='Félix' WHERE nombre='Felix'
    '''
```

Trabajar directamente con SQL

- Recuperar datos almacenador
  - void eachRow()
  - void query()
  - list rows()
  - object firstRow()

Trabajar directamente con SQL

- Método *eachRow()*

```
println "------------------- Escritores --------------------"
db.eachRow('SELECT * FROM Escritores'){
    println "${it.nombre} ${it.apellidos}, nacido el ${it.fechaNacimiento}"
    println "-" * 52
}
```

```
println "------------------- Escritores --------------------"
db.eachRow('SELECT nombre, apellidos, fechaNacimiento FROM Escritores')
{ row ->
    println row[0] + ' ' + row[1] + ', nacido el ' + row[2]
    println "-" * 52
}
```

Trabajar directamente con SQL

- Método *query()*

```
db.query('SELECT nombre, apellidos, fechaNacimiento FROM Escritores')
{ resultSet ->
      if (resultSet.next()){
          println resultSet.getString('nombre') + ' '+resultSet.getString(2) + ',
nacido el '+ resultSet.getDate('fechaNacimiento')
      }
  }
```

Trabajar directamente con SQL

- Método *rows()*

```
List escritores = db.rows('SELECT nombre, apellidos FROM Escritores')
println "Tenemos ${escritores.size()} en la base de datos:"
println escritores.collect{"${it.nombre} ${it.apellidos}"}.join(", ")
```

Trabajar directamente con SQL

- Método *firstRow()*

```
def primerEscritor = db.firstRow('SELECT nombre, apellidos FROM Escritores')
println primerEscritor.nombre + ' ' +primerEscritor.apellidos
```

Trabajar con DataSets

- Evitar la necesidad de tener conocimientos de SQL
- Los DataSets permiten añadir filas a una tabla y extraer la información de sus registros
- Los DataSets no permiten crear el esquema de la base de datos, ni operaciones de tipo *delete* o *update*
- Los DataSets se crean con el método *dataSet()* de la clase *groovy.sql.Sql*

Trabajar con DataSets

```
dataSetEscritores = db.dataSet('Escritores')
dataSetEscritores.add(
      nombre: 'Luis',
      apellidos: 'de Góngora y Argote',
      fechaNacimiento: '1561-07-11'
  )
```

Trabajar con DataSets

- Recorrer todos los registros de un DataSet

```
dataSetEscritores.each {
    println "${it.nombre} ${it.apellidos}, nacido el ${it.fechaNacimiento}"
}
```

Trabajar con DataSets

- Recorrer algunos registros de un DataSet con el método *findAll()*

```
escritoresSigloXX = dataSetEscritores.findAll{ it.fechaNacimiento > '1900-1-1' }
escritoresSigloXX.each{ println it.nombre +' '+it.apellidos }
```

Trabajar con DataSets

- Propiedades de los DataSets

```
println escritoresSigloXX.sql
println escritoresSigloXX.parameters

----Resultado-------------------------
select * from Escritores where fechaNacimiento > ?
[1900-1-1]
```

Trabajar con DataSets

- Un ejemplo más complejo

Trabajar con DataSets

```
db.execute '''
    DROP TABLE Libros IF EXISTS;
    CREATE TABLE Libros (
        idLibro    INTEGER GENERATED BY DEFAULT as IDENTITY,
        titulo    VARCHAR(128),
        anyo    INTEGER,
        fkEscritor    INTEGER
    );
    '''
def insertaLibros(titulo, anyo, apellidos) {
            db.execute """ INSERT INTO Libros(titulo, anyo, fkEscritor)
                    SELECT $titulo, $anyo, idEscritor
                    FROM Escritores WHERE apellidos=$apellidos
                """
}
```

Trabajar con DataSets

```
insertaLibros('La colmena', 1951, 'Cela Trulock')
insertaLibros('Rol de cornudos', 1976, 'Cela Trulock')
insertaLibros('La galatea', 1585, 'de Cervantes Saavedra')
insertaLibros('El ingenioso hidalgo don Quijote de la Mancha', 1605, 'de
Cervantes Saavedra')
insertaLibros('La dorotea', 1632, 'Lope de Vega y Carpio')
insertaLibros('La dragontea', 1602, 'Lope de Vega y Carpio')
```

Trabajar con DataSets

- Listado de libros con su correspondiente autor

```
db.execute '''
    DROP VIEW EscritoresLibros IF EXISTS;
    CREATE VIEW EscritoresLibros AS
        SELECT * FROM Escritores INNER JOIN Libros
            ON fkEscritor=idEscritor
'''

escritoresLibros = db.dataSet('EscritoresLibros')
escritoresLibros.each {
    println it.titulo + '(' + it.anyo + '), escrito por ' +it.nombre + ' ' + it.apellidos
}
```

GORM

- Facilita el acceso a las bases de datos
- Se basa en Hibernate
- Utilizado en Grails