

```

class Book implements Comparable {

    static withTable = 't_book'
    static transients = ['year']
    static embedded = ['price']

    String title
    Integer year
    Money price
    Date releaseDate = new Date() // default value

    Author mainAuthor
    Author coAuthor // b.add(to:'coAuthor', a)
    Map editions // b.editions=['first':e]
    Set publisher // b.addToPublisher(p)

    static belongsTo = Author
    static hasMany = [editions: Edition,
                     publisher: Publisher]

    String toString() {
        return "${id}: ${name}"
    }

    static constraints= {
        title(size:2..5,blank:false,unique:true)
        releaseDate(min:new Date(),nullable:false)
    }

    int compareTo(obj) {
        releaseDate.compareTo(obj.releaseDate)
    }
}

class Author {
    String name
    SortedSet books
    List coBooks // def b = a.coBooks[0]
    static hasMany = [books:Book, coBooks:Book]
    static mappedBy = [books:'mainAuthor',
                      coBooks:'coAuthor']
}

class Publisher {
    ...
    static mapping = {
        table 'publisher'
        cache usage:'read-only', include:'non-lazy'
        tablePerHierarchy false
        version false
        id generator:'hilo', params:
            [table:'hi_value',column:'next_value',max_lo:100]
        // id composite:['name', 'city'] // composite id
        columns {
            id column:'publisher_id' // custom id name
            books lazy:false, cache:'read-write'
            city column:'city_name', index:'city_idx'
            notes type:'text'
        }
    }
}
    
```

**Domain Class Validation**

blank	login(blank:false)	
creditcard	cardNo(creditcard:true)	
email	contactEmail(email:true)	
inList	name(inList:['A', 'B', 'CD'])	S,DB
matches	name(matches:['a-zA-Z'])	RE
max	age(max:new Date())   price(max:9.0f)	N,DB
maxSize	children(maxSize:10)	S,C,DB
min	age(min:new Date())   price(min:0.5f)	N,DB
minSize	children(minSize:5)	S,C,DB
notEqual	login(notEqual:'root')	
nullable	name(nullable:false)	
range	age(range:1..150)	N,DB
scale	price(scale:2)	N,DB
size	children(size:5..10)	S,C,DB
unique	login(unique:true)	

```

login(unique:'scope')
login(unique:['group','dept'])

url homepage(url:true)

validator even(validator:{it % 2 == 0})
           pwd(validator:{val,obj ->
                       obj.properties['pwd2'] == val
                       })
           // class.login.validator.error
           login(validator:{it.length != 0})
           // class.login.custom.error
           login(validator:{
               if(!it.endsWith('a'))
                   return ['custom.error']
           })
    
```

N - Number, S - String, C - Collection  
 DB - influences schema, RE - Regular Expression

**Domain Class Operations**

new	def obj = new Domain(name:'Abc')
.get(id)	def obj = Domain.get(1)
.getAll(..)	def lst = Domain.getAll(2,1,3)
	def lst = Book.getAll([1,2,3])
	def lst = Book.getAll()
.list(..)	def lst = Domain.list()
	def lst = Book.list(10)
	def lst = Book.list(max:10,offset:5)
	def lst = Book.list(sort:'col',order:'desc')
.listOrderBy*	def lst = Book.listOrderByName()
	def lst = Book.listOrderById(max:10,order:'desc')
.find(..)	Book.find('Book b where b.id=1')
	def b = Book.find('from Book as b where b.title=:title',[title:'Abc'])
	* 'from Book as book' won't work
	def b = new Book(title:'Abc')
	Book.find(b) // query by example
.findAll(..)	
.findBy*	
.findAllBy*	
.save(), .delete()	obj.save()   obj.save(flush:true) obj.save(false) // without validation obj.delete()   obj.delete(flush:true)
.where	
.add	
.add*	
.discard()	
.hasErrors()	
.properties	obj.properties = params obj.save()
.errors	
.constraints	
.ident()	
.merge()	
.refresh()	
.validate()	
.count()	if(Book.count()==0) { //add books }
.countBy*	
.createCriteria()	
.exists(id)	
.executeQuery()	
.withCriteria	
.withTransaction	
.lock()	obj.lock() // pessimistic locking

**References**

- <http://grails.org>
- [http://www.masukomi.org/writings/grails\\_crash\\_course.html](http://www.masukomi.org/writings/grails_crash_course.html)