Object Relational Mapping in Grails

1

Object Relational Mapping in Grails

- Grails Object Relational Mapping
- CRUD
- Constraints/Validation
- Relationships

2

Domain Class Basics

3

Domain Class Location Convention

```
|____grails-app
| |____conf
| | |____hibernate
| | |____spring
| |____controllers
| |____domain
| |____i18n
| |____services
| |____taglib
| |____utils
| |____views
```

The **domain** subdirectory is for any class that you want to be persistent. These classes are automatically mapped to the DB through Hibernate (or other GORM implementation)

4

Database Creation Magic

```
dataSource {
    pooled = false
    username = "sa"
    password = ""
    loggingSql = true
}

environments {
    development {
        dataSource {
            dbCreate = "create-drop"
            //...
        }
    }
    test {
    //...
```

5

DB creation options

- create-drop
  - Drop and re-create the database when Grails is run
- create
  - Create the database if it doesn't exist, but don't modify it if it does
- update
  - Create the database if it doesn't exist, and modify it if it does exist
- not defined
  - do nothing to the database

6

Helpful Tool: DB Console

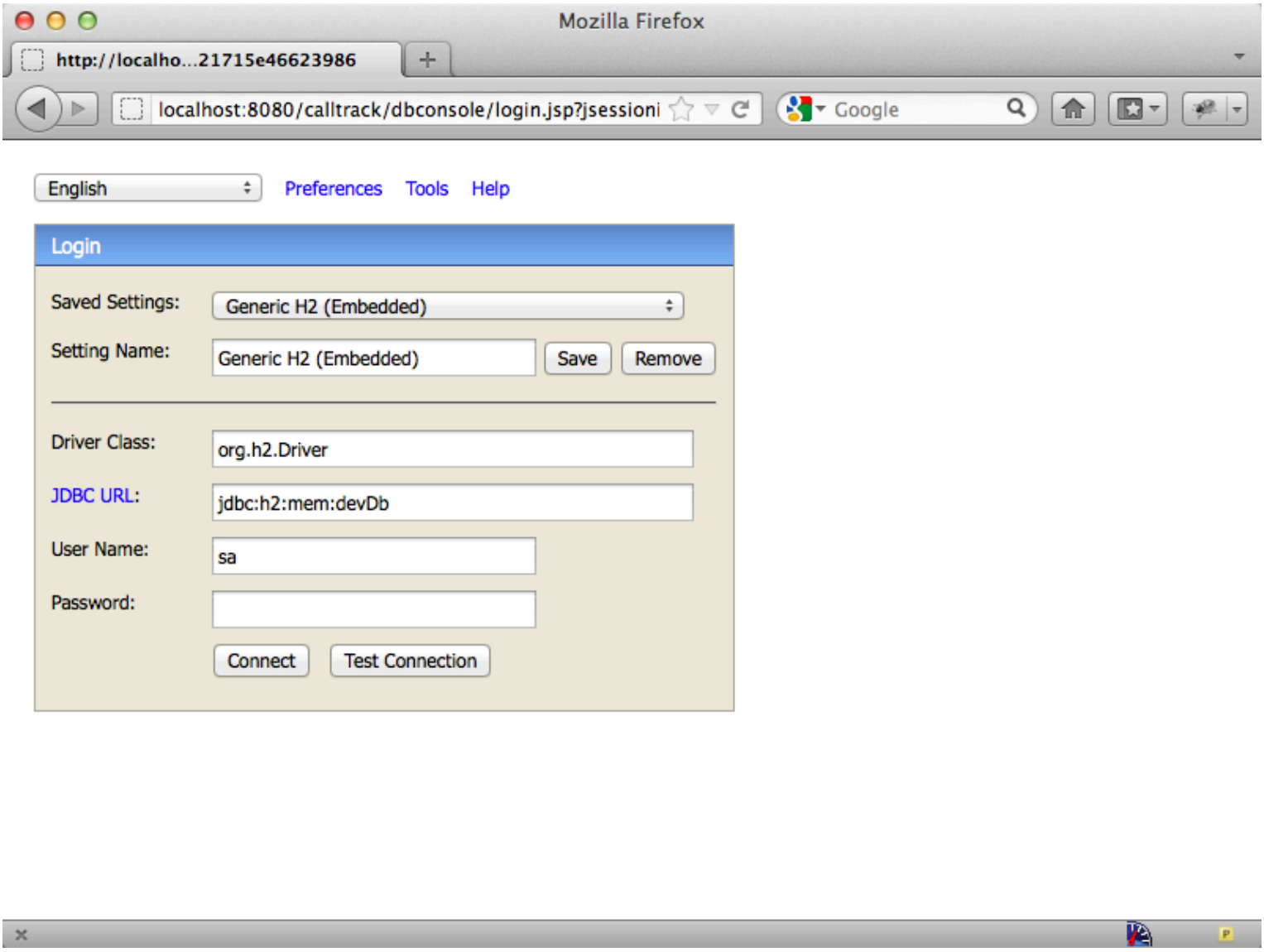
- grails run-app
- browse to http://localhost:8080/dbconsole

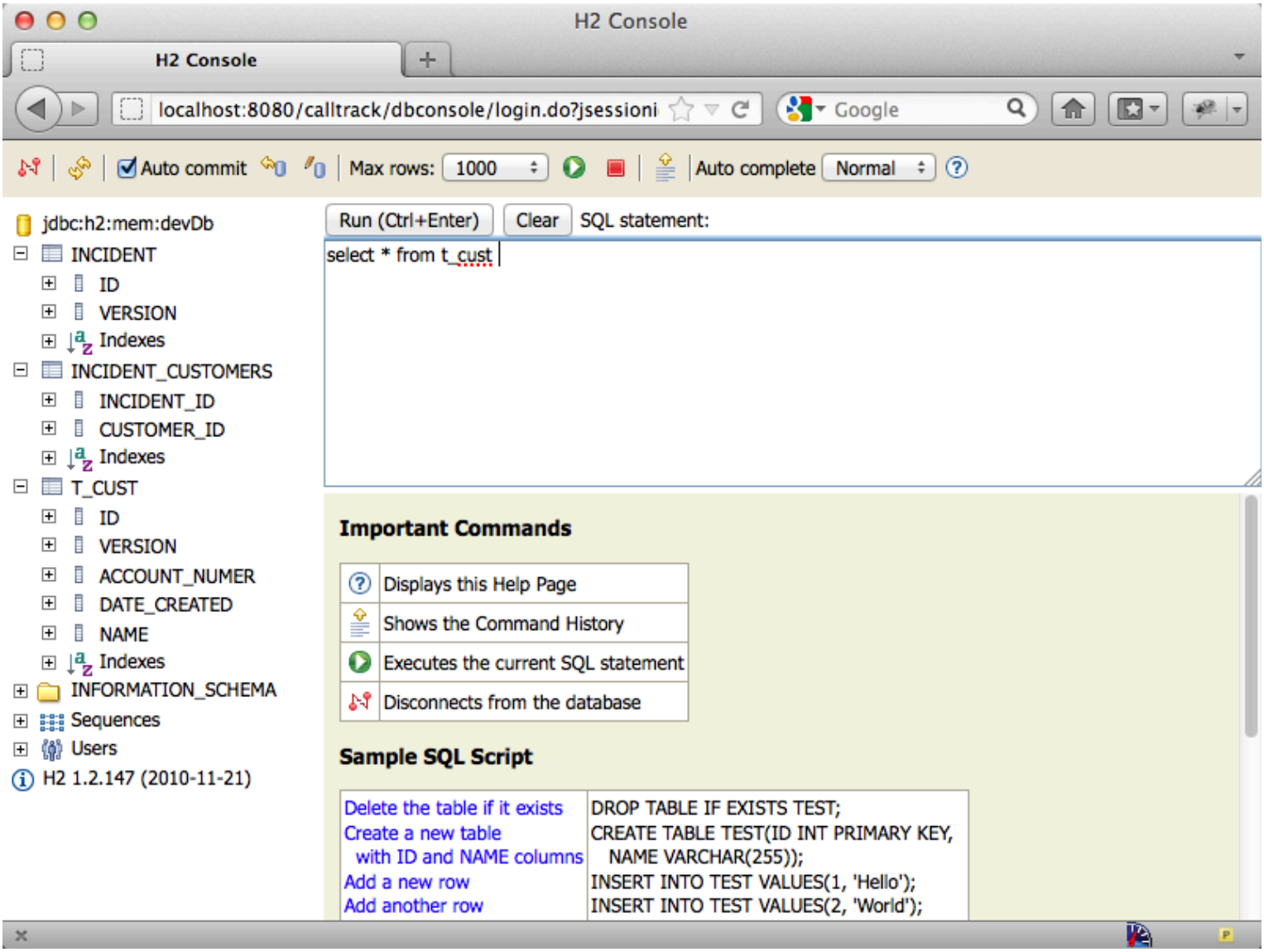
7

Helpful Tool: DB Console

- use driver, username, password, url from `grails-app/conf/DataSource.groovy`

```
dataSource {
    driverClassName = "org.h2.Driver"
    username = "sa"
    password = ""
}
environments {
    development {
        dataSource {
            dbCreate = "create-drop"
            url = "jdbc:h2:mem:devDb;MVCC=TRUE"
        }
    }
```


Mozilla Firefox
http://localho...21715e46623986
localhost:8080/calltrack/dbconsole/login.jsp?jsessioni
Google
English
Preferences
Tools
Help
Login
Saved Settings:
Generic H2 (Embedded)
Setting Name:
Generic H2 (Embedded)
Save
Remove
Driver Class:
org.h2.Driver
JDBC URL:
jdbc:h2:mem:devDb
User Name:
sa
Password:
Connect
Test Connection



A day in the life of a Grails Domain Class


```
1  class Customer {
2      String name
3      String accountNumber
4  }
```

```
mysql> describe customer;
+----------------+-------------+------+-----+---------+
| Field          | Type        | Null | Key | Default |
+----------------+-------------+------+-----+---------+
| id             | bigint(20)  | NO   | PRI | NULL    |
| version        | bigint(20)  | NO   |     | NULL    |
| account_number | varchar(255)| NO   |     | NULL    |
| name           | varchar(255)| NO   |     | NULL    |
+----------------+-------------+------+-----+---------+
4 rows in set (0.02 sec)
```

class name becomes table name

12

```
class Customer {
    String name
    String accountNumber
}
```

```
mysql> describe customer;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| id             | bigint(20)  | NO   | PRI | NULL    | auto_increment |
| version        | bigint(20)  | NO   |     | NULL    |                |
| account_number | varchar(255)| NO   |     | NULL    |                |
| name           | varchar(255)| NO   |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
4 rows in set (0.02 sec)
```

attribute names are converted into column names

13

```
class Customer {
    String name
    String accountNumber
}
```

```
mysql> describe customer;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| id             | bigint(20)  | NO   | PRI | NULL    | auto_increment |
| version        | bigint(20)  | NO   |     | NULL    |                |
| account_number | varchar(255)| NO   |     | NULL    |                |
| name           | varchar(255)| NO   |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
4 rows in set (0.02 sec)
```

id (PK) and version (optimistic locking) columns are added to DB, but don't need to be explicitly specified in code

14

ID and Version - invisible attributes?

- ID and Version were added to the DB table for Customer
- Where did they come from?

15

---

javap

```
1  javap Customer
2  Compiled from "Customer.groovy"
3  public class calltrack.Customer
4      extends java.lang.Object
5      implements groovy.lang.GroovyObject {
6
7          java.lang.Long id;
8          java.lang.Long version;
9
10         private java.lang.String name;
11         private java.lang.String accountNumber;
12
13         //. getters, setters, etc
14     }
```

16

Test early, test often

17

Types of Tests

- Unit Tests
  - Use `@TestFor` to mock Domain class behavior
  - Concurrent HashMap ORM implementation
  - No support for
    - transactions
    - HQL queries
    - composite keys
    - dirty checking methods

18

Types of Tests

- Integration Tests
  - Full grails environment
  - Hibernate ORM and H2 in-memory DB
  - HQL Queries
  - Dependencies injected automatically

19

Which type to write?

- Personal/team preference, but...
  - **Unit** test as much as possible
  - **Integration** test when required
- Or where specifically requested (in requirements on the assignment)

CRUD

CRUD

- Four basic functions of persistent storage:
  - CREATE - `INSERT`
  - READ - `SELECT`
  - UPDATE - `UPDATE`
  - DELETE - `DELETE`

Helpful Tool: SQL Logging

- turn on SQL logging with `logSql = true` in `grails-app/conf/DataSource.groovy`
- optionally format sql logging with `formatSql = true` in `grails-app/conf/DataSource.groovy`

```
1  dataSource {
2      pooled = false
3      username = "sa"
4      password = ""
5      logSql = true
6      formatSql = true
7  }
```

Create

```
1  void testCreate() {
2      Customer customer =
3          new Customer(name: 'Mike', accountNumber: '123')
4      customer.save()
5  }
```

```
INSERT INTO customer
    (
        id, version, account_number, date_created, name
    )
    VALUES
    (
        NULL, ?, ?, ?, ?
    );
```

Read

```
void testRead() {
    Customer customer = Customer.read(1)
}
void testGet() {
    Customer customer = Customer.get(1)
}
```

```
 SELECT
     customer0_.id             AS id0_0_,
     customer0_.version        AS version0_0_,
     customer0_.account_number AS account3_0_0_,
     customer0_.date_created   AS date4_0_0_,
     customer0_.name           AS name0_0_
 FROM
     customer customer0_
 WHERE
     customer0_.id=?
```

Update

```
void testUpdate() {
    //setup
    def c = new Customer(name: 'Mike',
            accountNumber: '123').save()
    //update
    Customer customer = Customer.get(c.id)
    customer.name = 'new value'
    customer.save()
}
```

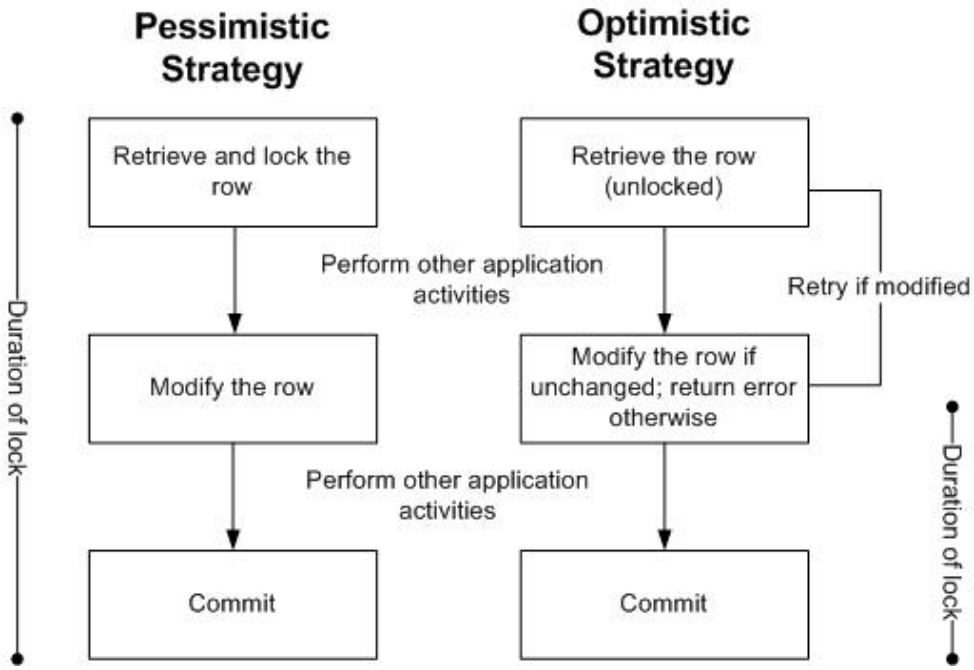
```
UPDATE customer
SET
    version=?,
    account_number=?,
    date_created=?,
    name=?
WHERE
    id=?
AND version=?
```

Delete

```
1  void testDelete() {
2      //setup
3      def c = new Customer(name: 'Mike',
4              accountNumber: '123').save()
5      //delete
6      Customer customer = Customer.get(c.id)
7      customer.delete()
8  }
```

```
1  delete from customer where id=? and version=?
```


Locking



http://www.toadworld.com/Portals/0/GuyH/ Contention/Feb2008/optimisticPessimistic.jpg

Optimistic Locking (Default)

- Version column

```
1  delete from customer where id=? and version=?
```

```
UPDATE customer
SET
    version=?,
    account_number=?,
    date_created=?,
    name=?
WHERE
    id=?
AND version=?
```

- Version incremented with each update

29

Pessimistic Locking

```
void testLock() {
    //setup
    def c = new Customer(name: 'Mike',
        accountNumber: '123').save()
    //use lock method to perform SELECT ... FOR UPDATE
    Customer.withTransaction {
        Customer customer = Customer.lock(c.id)
        customer.name = 'John'
        customer.save(flush: true)
    }
    // lock is automatically released at end of transaction
    assert 'John' == Customer.get(c.id).name
}
```

30

Constraints and Validation

- By default, all properties are required (i.e. can't be NULL)

```
def c = new Customer()
c.save()
println c.errors
```

```
org.springframework.validation.BeanPropertyBindingResult:
  2 errors
Field error in object 'Customer' on field 'accountNumber':
    rejected value [null]; codes [......; default message
    [Property [{0}] of class [{1}] cannot be null]
Field error in object 'Customer' on field 'name':
    rejected value [null]; codes [......; default message
    [Property [{0}] of class [{1}] cannot be null]
```

31

Constraints closure

- By default, all properties are required (i.e. can't be NULL)

```
class Customer {
    String name
    String accountNumber
}
```

is the same as

```
class Customer {
    String name
    String accountNumber
    static constraints = {
        name(nullable:false)
        accountNumber(nullable:false)
    }
}
```

Testing Constraints

```
void testTitleIsRequired() {
    Book book = new Book()
    book.save()
    assert "nullable" == book.errors['title'].code
}
```

Built In Constraints

- `blank` - cannot be empty string
- `creditCard` - matches a credit card number
- `email` - valid email address
- `inList` - contained in a list of values
- `matches` - applies a regular expression
- `max` - maximum value of a class that implements comparable
- `min` - minimum value of a class that implements comparable
- `notEqual` - not equal to a specified value

Built In Constraints

- `nullable` - set to false for NOT NULL constraint
- `range` - within a min and max
- `scale` - performs rounding to a specified precision (doesn't generate error messages)
- `size` - length of a string or collection
- `unique` - unique value
- `url` - URL address
- `validator` - custom validator

Some constraints influence schema

```
String name
String accountNumber
String phone
Date dateCreated
static constraints = {
  // name must be at least 2 chars; no more than 100
  name(nullable:false, size:2..100)
  // account number min length of 1 and max of 5 chars
  accountNumber(nullable:true, size:1..5)
  // phone number must be 10 digits
  phone(nullable:true, matches: /(\d{10})?/)
}
```

```
// ...
static constraints = {
    // name must be at least 2 chars; no more than 100
    name(nullable:false, size:2..100)
    // account number min length of 1 and max of 5 chars
    accountNumber(nullable:true, size:1..5)
    // phone number must be 10 digits
    phone(nullable:true, matches: /(\d{10})?/)
    // dateCreated not specified
}
```

```
mysql> describe customer;
+----------------+--------------+------+-----+---------+
| Field          | Type         | Null | Key | Default |
+----------------+--------------+------+-----+---------+
| id             | bigint(20)   | NO   | PRI | NULL    |
| version        | bigint(20)   | NO   |     |         |
| account_number | varchar(5)   | YES  |     | NULL    |
| date_created   | datetime     | NO   |     |         |
| name           | varchar(100) | NO   |     |         |
| phone          | varchar(255) | YES  |     | NULL    |
+----------------+--------------+------+-----+---------+
```

name field gets max length in DB of 100

```
// ...
static constraints = {
    // name must be at least 2 chars; no more than 100
    name(nullable:false, size:2..100)
    // account number min length of 1 and max of 5 chars
    accountNumber(nullable:true, size:1..5)
    // phone number must be 10 digits
    phone(nullable:true, matches: /(\d{10})?/)
    // dateCreated not specified
}
```

```
mysql> describe customer;
+----------------+-------------+------+-----+---------+
| Field          | Type        | Null | Key | Default |
+----------------+-------------+------+-----+---------+
| id             | bigint(20)  | NO   | PRI | NULL    |
| version        | bigint(20)  | NO   |     |         |
| account_number | varchar(5)  | YES  |     | NULL    |
| date_created   | datetime    | NO   |     |         |
| name           | varchar(100)| NO   |     |         |
| phone          | varchar(255)| YES  |     | NULL    |
+----------------+-------------+------+-----+---------+
```

account number nullable and max length of 5

```groovy
// ...
static constraints = {
    // name must be at least 2 chars; no more than 100
    name(nullable:false, size:2..100)
    // account number min length of 1 and max of 5 chars
    accountNumber(nullable:true, size:1..5)
    // phone number must be 10 digits
    phone(nullable:true, matches: /(\d{10})?/) // no size
    // dateCreated not specified
}
```

```
mysql> describe customer;
+----------------+-------------+------+-----+---------+
| Field          | Type        | Null | Key | Default |
+----------------+-------------+------+-----+---------+
| id             | bigint(20)  | NO   | PRI | NULL    |
| version        | bigint(20)  | NO   |     |         |
| account_number | varchar(5)  | YES  |     | NULL    |
| date_created   | datetime    | NO   |     |         |
| name           | varchar(100)| NO   |     |         |
| phone          | varchar(255)| YES  |     | NULL    |
+----------------+-------------+------+-----+---------+
```

phone number gets default max size of 255 and nullable

```groovy
// ...
static constraints = {
    // name must be at least 2 chars; no more than 100
    name(nullable:false, size:2..100)
    // account number min length of 1 and max of 5 chars
    accountNumber(nullable:true, size:1..5)
    // phone number must be 10 digits
    phone(nullable:true, matches: /(\d{10})?/) // no size
    // dateCreated not specified
}
```

```
mysql> describe customer;
+----------------+--------------+------+-----+---------+
| Field          | Type         | Null | Key | Default |
+----------------+--------------+------+-----+---------+
| id             | bigint(20)   | NO   | PRI | NULL    |
| version        | bigint(20)   | NO   |     |         |
| account_number | varchar(5)   | YES  |     | NULL    |
| date_created   | datetime     | NO   |     |         |
| name           | varchar(100) | NO   |     |         |
| phone          | varchar(255) | YES  |     | NULL    |
+----------------+--------------+------+-----+---------+
```

date_created gets default not nullable

40

Validation

41

Validation

- validation is implicitly called when .save() is called, but can also be called independently:

```
groovy> def c = new Customer()
groovy> println c.validate()
false
```

42

Validation

- if validation fails, errors are placed in the errors property of the domain class
  - can also check hasErrors()

```
groovy> def c = new Customer()
groovy> c.validate()
groovy> println c.hasErrors()
true
```

43

Validation

- Note that by default .save() does not throw an exception if validation fails and entity is NOT persisted

```
groovy> def c = new Customer()
groovy> c.validate()
groovy> println c.hasErrors()
true
```

```
mysql> select count (*) from customer;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

Validation

- Can be overridden by using “failOnError:true” in the .save() call

```
groovy> new Customer().save(failOnError:true)
//...
Exception thrown
//...
grails.validation.ValidationException:
  Validation Error(s) occurred during save():
- Field error in object 'caller.Customer' on field
  'address': rejected value [null]; codes
```

- Can also be overridden globally in `grails-app/conf/Config.groovy`
  - `grails.gorm.failOnError=true`

Relationships

Many to One - Unidirectional

```
class Customer {
    Address address
}
class Address {
}
```

- Unidirectional many-to-one relationship from Customer to Address
- Address is not aware of Customer

Many to One - Unidirectional

```
class Customer {
    Address address
}
class Address {
}

customer.address = new Address()
customer.save()
//transient object exception

def address = new Address()
address.save()
customer.address = address
customer.save()
//OK
```

- Updates do not cascade

48

Many to One - Unidirectional

```
class Customer {
    Address address
}
class Address {
}

def address = new Address()
address.save()
customer.address = address
customer.save()

address.delete()
//FK constraint exception

customer.delete()
address.delete()
//OK
```

- Deletes do not cascade

49

Many to One - Bidirectional

```
class Customer {
    Address address
}
class Address {
    static belongsTo = [customer:Customer]
}
```

- Address belongs to Customer
- Saves and Deletes to Customer will cascade to Address

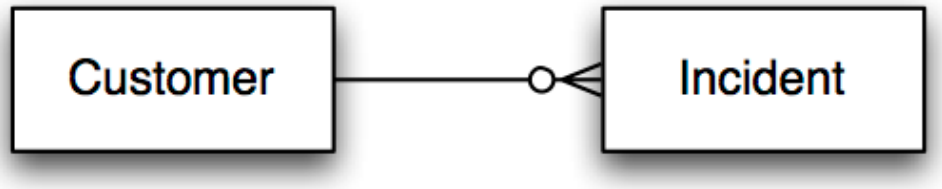
50

Many to One - Bidirectional

```
customer.address = new Address()
customer.save()
//OK
//saves both customer and address
customer.delete()
//OK
//deletes customer and address
```

51

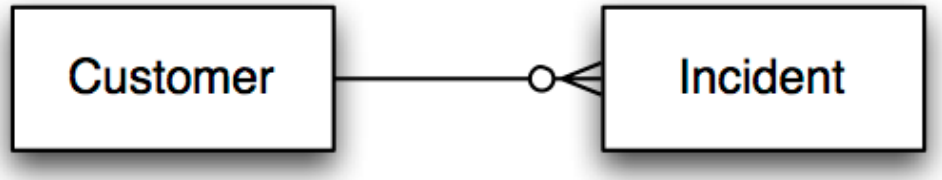
One to Many



- use "static hasMany = ..." definition
- inserts and updates are cascaded
- deletes cascade if belongsTo is defined

52

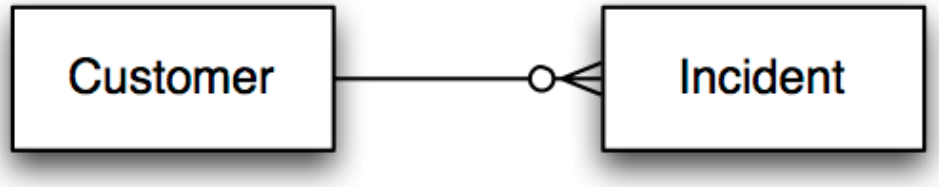
One to Many



- default is java.util.Set; can use
  - SortedSet (must implement comparable)
  - List (adds `{table_name}_idx` column)
- convenience methods:
  - `customer.addToIncidents(incident)`
  - `customer.removeFromIncidents(incident)`

53

One to Many




```groovy
class Customer {
    static hasMany = [incidents:Incident]
}

class Incident {
    static belongsTo = [customer:Customer]
}

//saves both Customer and Incident
def c = new Customer()
c.addToIncidents(new Incident())
c.save()

//deletes customer and all associated incidents
c.delete()
```

Many to Many

- both sides define hasMany
- must define an owner (using belongsTo on the owned object)

Many to Many

```
class Customer {
    static hasMany = [products:Product]
}

class Product {
    static hasMany = [customers:Customer]
    //Customer owns the relationship
    static belongsTo = Customer
    //inserts will cascade from customer not from product
}
```

Many to Many

- Realistically, there is often a concrete entity between the two sides of the relationship

```
class Customer {
    static hasMany = [products:CustomerProduct]
}

class CustomerProduct {
    Customer customer
    Product product
    Date datePurchased
    static belongsTo = [Customer, Product]
}

class Product {
    static hasMany = [customers:CustomerProduct]
}
```