

Universidade Estadual de Maringá

BANCO DE DADOS E UMA APLICAÇÃO WEB

Alex Moreira Lima - <http://din.uem.br/~ra59602>

Eduardo Sutil - <http://din.uem.br/~ra58703>

Henrique Vigando- <http://din.uem.br/~ra55761>

Pagina do projeto: <http://svlprojetos.webnode.com.br>

Resumo

Este trabalho apresenta os elementos que constituem uma estrutura de banco de dados juntamente com um caso de uso de uma aplicação web. Bem como a apresenta de forma geral dos conceitos de bancos de dados a sua aplicação em um desenvolvimento de aplicações para web, usando ferramentas de apoio para tais resultados.

Palavras-chave: Artigo científico. Banco de Dados. Aplicação Web.
Resolução Nº 061/2003-CEP

1 INTRODUÇÃO

A necessidade de armazenar e tratar os dados da carreira do Magistério Público de uma instituição é o desafio. As informações são muito importantes na maioria das organizações, desenvolveu-se uma série de conceitos e técnicas para gerenciar os dados (Silberchatz, Abraham, 2006).

O presente artigo demonstrará o desenvolvimento do GINFO (RH)-Promoção Funcional passo a passo. Mostrando os conceitos de Banco de Dados, das operações SQL (DDL e DML), também vamos discutir o mapeamento objeto relacional, conhecido como ORM, que podemos observar no Grails.

O Modelo Relacional tem por base principal neste modelo uma relação, que pode ser considerada como um conjunto de registros. Tendo um esquema como modelo de dados possuindo seu nome especificamente, o nome de cada campo, e o tipo do campo. (Ramakrishnan e Gehrke 2008).

Dentro deste contexto estaremos construindo nosso estudo de caso, o Ginfo que se baseia no documento **Resolução Nº 061/2003-CEP**. Onde o professor poderá cadastrar atividades acadêmicas realizadas na sua carreira, e terá seus pontos computados, de forma que possa progredir dentro da universidade, subindo de nível e classe.

Primeiramente, será apresentado o conceito de banco de dados, do modelo relacional, de SGBD, de SQL, para depois podermos avançar um pouco mais, falando das ferramentas e tecnologias que utilizaremos para o desenvolvimento da nossa aplicação, e estes conceitos são fundamentais para o entendimento da aplicação.

Delinearemos também sobre os métodos utilizados para construir nosso estudo de caso, ferramentas como o STS(Spring Source Tools), Workbench que é uma ferramenta que nos ajuda na modelagem dos dados, gerando SQLS para a criação das tabelas e dos relacionamentos entre elas. Também estaremos apresentando um pouco sobre o Grails um framework open-source para aplicação web que usa a linguagem Groovy e complementa o desenvolvimento Java Web. Dentro deste contexto abordaremos sobre o GORM(Mapeamento objeto relacional).

Após todo contexto nos inserido, iremos para a prática e desenvolvimento do nosso estudo de caso Ginfo. Apresentaremos uma parte de como foi realizado o desenvolvimento da aplicação. Como foi criado o banco de dados e as tabelas necessárias, e após isso como foram feitas nossas classes de domínios e os controllers

que as iriam manipular.

Por ultimo iremos mostrar os resultados da nossa modelagem, e também de como ficou a nossa aplicação, no contexto do desenvolvimento que apresentamos neste artigo.

No final do artigo estará disponível dois apêndices com os códigos SQL, e também como ficaram as classes de domínio.

2 FUNDAMENTAÇÃO TEÓRICA

.....2.1 Sistema de Gerenciamento de Banco de Dados

Segundo Raghu Ramakrishna e Johannes Gehrke (1998, p.3), banco de dados (BD) é uma coleção de dados que, tipicamente, descrevem as atividades de uma ou mais organizações relacionais. E um sistema de gerenciamento de banco de dados (SGBD) é um software projetado para auxiliar na manutenção e utilização de vastos conjuntos de dados.

Um SGBD abstrai a organizações dos dados que estão no banco de dados e permite o usuário, ou a aplicação, acessá-lo e manipulá-lo sem a necessidade do detalhamento de sua organização e armazenamento. Veja o exemplo da abstração na figura 2.1.

Aplicações necessitam que seus dados estejam íntegros e seguros, e o SGBD também oferece estas vantagens, além disso, auxilia no desenvolvimento da aplicação, proporcionando formas rápidas de estruturar os dados.

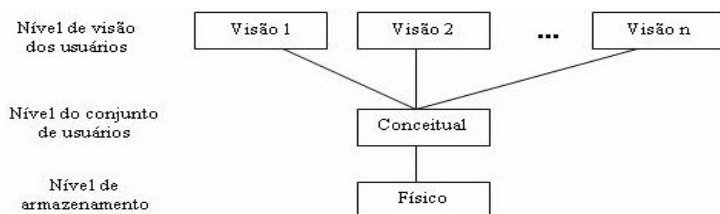


Figura 2.1 – abstração da visão de banco de dados.

.....2.2 SQL

SQL (Structured Query Language), como o nome diz, é uma Linguagem de Consulta Estruturada. A SQL dá suporte para DDL (Linguagem de definição de dados) e DML (Linguagem de modelagem de dados).

DDL é um subconjunto da SQL que permite criar, excluir, e modificar a estrutura das tabelas do banco de dados. Também permite definir restrições de inserções, modificações e exclusão de dados visando à integridade dos dados. No exemplo 2.2 podemos observar a definição da tabela *atividade_ensino* usada na aplicação G_Info.

DML é um subconjunto da SQL que permite inserir, excluir e modificar tuplas

(registros) nas tabelas já existentes no banco de dados. No exemplo 2.2 podemos ver como inserir, alterar e excluir registros.

Exemplo 2.2

```
INSERT`ginfo_artigo`.`departamento`  
  (id, nome, sigla)  
VALUES  
  (3, 'Informática', 'DIN');
```

***Exemplo de inserção: Inserindo o registro de id 3 na tabela departamento da base de dados ginfo_artigo. Atribui o nome 'Informática' ao departamento e a sigla 'DIN'.*

```
UPDATE`ginfo_artigo`.`departamento`  
  SET  nome = 'Engenharia Civil',  sigla = 'DEC'  
  WHERE id = 3
```

***Exemplo de alteração: alterando o atributo 'nome' do registro 3 para 'Engenharia Civil' e a 'sigla' para 'DEC'.*

```
DELETE FROM `ginfo_artigo`.`departamento`  
  WHERE id = 3
```

***Exemplo de exclusão: excluindo o registro de 'id' 3 da tabela 'departamento'.*

DQL (Linguagem de Consulta de Dados), DCL (de Controle de Dados) e DTL (Linguagem de Transação de Dados) também são subconjuntos da SQL. DQL permite recuperar dados do banco de dados. DCL permite controlar as operações no banco de dados para que pessoas não autorizadas os modifiquem. DTL permite definir transações de banco de dados que é uma sequência de operações.

.....2.3 Organização de chaves

Cada entidade possui uma ou mais chaves. As chaves são informações identificadoras dos registros. Por exemplo, se pensarmos na entidade Professor, pode-se definir o número de sua matrícula como chave, pois é única para cada professor, isso possibilita fazer buscas na tabela pelo número da matrícula.

Existem vários tipos de chaves que devem ser usados conforme a necessidade.

Chave primária é única para cada registro, porém podem existir registros únicos sem serem definidos como chaves primárias.

Pode-se usar uma combinação dos campos para gerar um registro único, denominada, chave composta.

Chave estrangeira é a coluna que armazena a chave primária de outra tabela. É através da chave estrangeira que é realizado os relacionamentos entre as entidades. Por exemplo, na aplicação G_Info, a Entidade *atividade_ensino*, contém o campo *disciplina_id*, que representa a chave primária da Entidade disciplina, o que significa que para cada atividade de Ensino deve-se cadastrar uma disciplina. Ver exemplo 2.2.

.....2.4 Relacionamentos

Os relacionamentos no banco de dados são definidos por chaves estrangeiras em uma tabela. Cada tabela possui sua chave que pode ser única (um único dado é a chave dessa tabela), composta (mais de um dado formam uma combinação única para se obter a chave de acesso de cada registros) e chave estrangeira (uma coluna de que guarda a chave primeira de outra tabela) que é a chave que forma os relacionamentos.

Existem três tipos de relacionamentos em um banco de dados relacional. O primeiro tipo é um para um (1:1), que define para cada registro de uma tabela, apenas um registro correspondente em outra tabela. Veja na imagem 2.4.1, a tabela Pessoa possui apenas um Endereco e cada registro inserido na tabela Endereco há apenas um registro correspondente na tabela Pessoa.

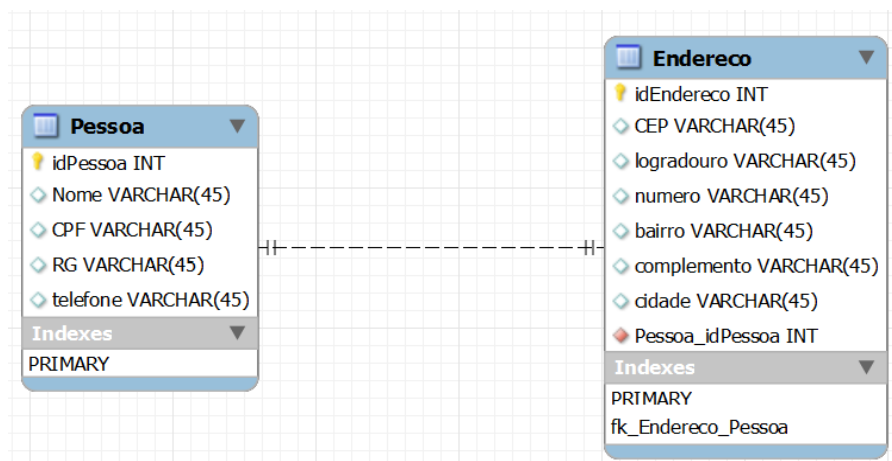


Imagem 2.4.1 Exemplo de relacionamento 1:1.

O relacionamento um para muitos (1:*) define que para um registro de uma tabela pode haver um ou mais registros correspondentes em outra tabela.

O relacionamento muitos para muitos (*:*) define que para cada registro em uma tabela existe um ou mais correspondentes em outra tabela, e para cada registro nessa tabela correspondente pode haver um ou mais registros correspondentes na primeira

tabela. Para montar esse relacionamento, o modelo Entidade Relacional cria uma terceira tabela que armazena no mesmo registro as chaves primárias das tabelas que estão se relacionando. Um exemplo disso, na aplicação G_Info é a produção acadêmica e o professor, onde cada professor pode conter várias produções acadêmicas e cada produção acadêmica pode conter vários professores.



Imagem 2.4.2 Exemplo de relacionamento 1:1.

Exemplo 2.4

```
CREATE TABLE `ginfo_artigo`.`atividade_ensino` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT ,
  `ano` INT(11) NOT NULL ,
  `carga_horaria_total` VARCHAR(255) NOT NULL ,
  `co_responsavel` VARCHAR(255) NOT NULL ,
  `disciplina_id` BIGINT(20) NULL DEFAULT NULL ,
  `numero_alunos` VARCHAR(255) NOT NULL ,
  `pontuacao` FLOAT(11) NOT NULL ,
  `tipo_atividade_ensino` VARCHAR(28) NOT NULL ,
  PRIMARY KEY (`id`) ,
  CONSTRAINT `FKD3B3E8F8FF744C45`
  FOREIGN KEY (`disciplina_id`)
  REFERENCES `ginfo_artigo`.`disciplina` (`id` ))
```

***Exemplo de criação de uma tabela na base de dados 'ginfo_artigo' usando o campo 'id' como chave primária e o campo 'disciplina_id' como chave estrangeira, fazendo referencia a tabela 'disciplina'.*

.....2.5 Modelo Objeto Relacional

O modelo relacional baseia-se no princípio de que os dados estão guardados em tabelas ou que entre si possuem um relacionamento. O modelo hierárquico é constituído por registros que são conectados em um modelo de árvore de forma que cada modelo tenha apenas um possuidor.

No modelo objeto relacional, temos uma junção do modelo Relacional com a Orientação a Objetos, e este modelo é utilizado para mapear a orientação da aplicação no modelo relacional do banco de dados.

Para desenvolver uma aplicação, é necessário desenvolver várias análises e especificações, e após essas verificações, desenvolve-se um modelo da base de dados, usando o conceito Entidade Relacional, ou seja, fazer o relacionamento entre as entidades, e após isso deve ser transformado em uma base de dados através de um SGBD.

Para cada entidade são definidos seus atributos, que no SGBD será uma coluna da tabela, ou uma variável da tupla.

Os relacionamentos são dados quando há uma relação entre as entidades, por exemplo, na aplicação que estudaremos neste artigo, teremos um relacionamento entre a entidade departamento e a entidade curso, onde um departamento possui vários cursos. Dessa forma é possível criar vários relacionamentos entre as entidades e organizar a base de dados para que os usuários tenham as informações que desejam.

3 METODOLOGIA

Nesta sessão vamos demonstrar quais foram as metodologias aplicadas para criarmos nosso banco de dados em conjunto da nossa aplicação web, nossos frameworks, sistemas e programas que usamos de ferramentas e quais são suas influencias para nosso desenvolvimento.

Para todos os casos de exemplo aplicados usaremos a **Resolução Nº 061/2003-CEP**, para nosso baseamento.

.....3.1 Springsouce Tools

Para auxiliar o desenvolvimento utilizaremos uma IDE (*Integrated Development Environment* – ambiente de desenvolvimento integrado) ou seja um programa que proporciona um ambiente de desenvolvimento de softwares e assim agilizando grande parte do processo, principalmente na organização dos arquivos e pacotes do nosso projeto de software.

A IDE que usamos é a Springsouce Tools (STS), desenvolvida pela Spring contida em uma divisão da VMware, com suporte especializada para desenvolvimento em Graís (veremos este framework a seguir), por isso sua escolha. O STS nos oferece muitas facilidades uma delas são as perspectivas, onde podemos ter uma visão de gerenciamento de banco de dados, suportando vários SGBD, como no nosso caso o MySQL. Mas sua maior facilidade é a edição e a organização dos nos arquivos gerados pelo nosso framework e da nossa aplicação.

.....3.2 Grails – Framework

É um framework para desenvolvimento de aplicações web, como linguagem de programação nativa o Groovy, que nada mais é que um Java “limpo”. Conhecido também como um ambiente de desenvolvimento, da qual utiliza outros frameworks para complementação se seu arcabouço.

Frameworks envolvidos:

- Hibernate (para comunicação com BD, persistências dos dados);

- Spring (desenvolvimento de JEE e MVC);
- Quartz (desenvolvimento para agendamento de tarefas);
- SiteMesh (desenvolvimento para layouts de paginas web).

A utilização do Grails é eficiente, quando tratamos de agilidade em desenvolvimentos de sistemas para web, pois nossas aplicações são rapidamente desenvolvidas devido à interligação dos outros frameworks. Assim criando uma aplicação web com um nível dinâmico e alto rapidamente.

Como exemplo, realizar um cadastro de pessoas pra determinada necessidade em alguma impressa. Isso levaria poucos dias. Este tipo de aplicação necessita um tempo reduzido de desenvolvimento, logo o framework facilita esse processo.

Além do mais o Grails implementa uma aplicação acima do Hibernate, chamada GORM (na qual veremos a seguir), na qual facilita muito a relação de persistência da nossa aplicação orientada a objetos em um banco de dados relacional, um exemplo é a simples configuração de um arquivo chamado `DataSouce.groovy` que na qual podemos configurar o arquivo de biblioteca do nosso banco e a url de conexão com nosso banco de dados com simples duas linhas de código.

.....3.3 GORM – Grails Mapeamento Objeto Relacional

O mapeamento objeto relacional é um recurso convencionalmente usado para amarrar um projeto (aplicação) de software, desenvolvido no paradigma de programação orientado a objeto, com banco de dados relacional. As classes desenvolvidas na aplicação são mapeadas para tabelas (entidade-relacionais), na qual podemos persistir as informações definidas como atributos de classes, em campos na tabela do banco de dados.

No Grails este mapeamento é feito totalmente pelo GORM, ele implementa o Hibernate 3, framework comumente usado para realiza o ORM para aplicações Java. Para usar o Hibernate é necessário fazer uma serie de configurações em nossa aplicação Java. Em Grails o GORM já está pré configurado para nossa aplicação Grails. Veremos a seguir quais são essas configurações e como podemos usa-las.

- A modelagem de uma aplicação Grails fica restrita as classes denominadas: classes de domínio (domain class). Nela definimos os atributos de nosso objeto das quais termos mapeados e persistidos pelo GORM no banco de dados, logo mais veremos a clouser *mapping* onde veremos opções de mapeamento de

nossos atributos.

- Com o GORM podemos definir os relacionamentos entre as entidades diretamente na classes de domínio, os tipos de relacionamentos possíveis são:
 - -Um-para-um ou muitos-para-um (one-to-one): Este tipo de relacionamento pode ser usado simplesmente por uma forma associativa, declarando um atributo de uma classe em outra classe ou ainda com a sintaxe *hasOne*, ou de forma é a agregativa usando a sintaxe *belongsTo*, criando o chamado “cascade”.
 - -Um-para-muitos (one-to-many): Esse tipo de relacionamento é facilmente definido com a sintaxe *hasMany*, exemplo: na classe professor podemos definir uma action *hasMany* que possa dizer que um professor possui muitas produções acadêmicas. Neste caso o GORM cria a tabela associativa do relacionamento, formando uma tabela de chave composta, uma chave primaria com o id do lado mais “forte” do relacionamento e subsequentemente a chave estrangeira com id de outra entidade.
 - -Muitos-para-muitos (many-to-many): Esse relacionamento é definido em nossas classes de domínios em ambos os lado do relacionamento usando a sintaxe *hasMany* e *belongsTo* que é definido somente em uma classe para que ela seja responsável pelo por persistir a informação. Assim como na relação um-para-muitos, o GORM cria uma tabela associativa também para este tipo de relacionamento, definida por sua chave composta.

Além dos relacionamentos o GORM oferece muitas outras opções para podermos usar os recursos de ORM segue alguns recursos uteis:

- *mapping* (define algumas ações q podemos aplicar em nível de aplicação em nosso banco de dados)
- *autoImport* – *cascade* – *joinTable* – *lazy* – *order* - *sort* ...
- *constrains* (insere restrições direto no banco de dadnos em nivel de aplicação)
- *blank* – *maxSize/minSize* – *nullable* – *unique* - *inList* ...

Temos alguns métodos para manusearmos nossos dados persistindo banco de maneira bem facil queo GORM oferece pra nós:

- *save*, *list*, *delete*, *discart*, *read*, *lock* ... (básicos para o CRUD)

- *find, findAll, findBy, findAllBy, findWhere, findAllWhere, get, getAll, removeFrom ...*

.....3.4 **MySQL Workbench**

MySQL Workbench é uma ferramenta visual de design de banco de dados que integra SQL desenvolvimento , administração , design de banco de dados , criação e manutenção em um único ambiente de desenvolvimento integrado para o MySQL . É o sucessor do DBDesigner 4 de fabFORCE.net, e substitui o pacote anterior do software, MySQL GUI Tools Bundle.

Esta será nossa ferramenta usada para nossa modelagem ER, por ela conseguimos gerar visualmente nossas tabelas no banco de dados e seus atributos. Além disso temos a facilidade de fazermos qualquer alterações em nossos esquemas e tabelas. A seguir segue algumas características.

- Geral :
 - Conexão de banco de dados e gerenciamento de instância
 - Assistente impulsionado itens de ação
 - Suporte para plugins personalizado
- Editor SQL:
 - Esquema de objeto navegação
 - SQL highlighter de sintaxe e analisador declaração
 - Múltipla, resultado editável sets
 - SQL trechos coleções
 - Conexão SSH tunneling
 - Suporte a Unicode
- Modelagem de dados:
 - ER diagramming
 - Modelagem visual Drag'n'Drop
 - Engenharia reversa de Scripts SQL e banco de dados ao vivo

- Engenharia para a frente Scripts SQL e banco de dados ao vivo
 - Sincronização esquema
 - Impressão de modelos
 - Importação de fabFORCE.net DBDesigner4
-
- Administração de banco de dados:
 - Iniciar e parar de instâncias de banco de dados
 - Configuração de instância
 - Gerenciamento de contas de banco de dados
 - Variáveis de instância navegação
 - Log de navegação de arquivos
 - Dados de despejo de exportação / importação

4 DESENVOLVIMENTO

Nesta sessão iremos apresentar como ficarão as persistências dos dados no disco. Usaremos o Mysql Workbench para definir as entidades e as relações entre elas.

Portanto iremos demonstrar uma parte de como esta sendo desenvolvida a aplicação, desde a criação das tabelas pelo workbench, como os códigos sql que estaremos utilizando para a geração das tabelas. E após a definição do banco de dados estaremos estabelecendo nossas classes de domínio e fazendo o mapeamento objeto-relacional pelo GORM na nossa aplicação.

.....4.1 Criação das tabelas SQL

Utilizando nosso estudo de caso, o GInfo, aonde ele faz um controle das atividades dos professores, e levando em consideração os critérios estabelecidos pela UEM para nomeação de cargos aos professores, temos que, tudo depende dos trabalhos realizados pelos mesmos, neste caso, uma entidade a ser criada é PROFESSOR, porém antes, vamos criar algumas tabelas das quais a entidade PROFESSOR irá utilizar.

Conforme a resolução **Resolução Nº 061/2003-CEP** do nosso estudo de caso Ginfo, concluímos que deveria ser criado uma tabela “Classe” e outra tabela “Nível”, fazendo o relacionamento de um para muitos entre “Classe” e “Nível” e depois disso realizar o relacionamento com a tabela “Professor”. Como um professor só pode estar em uma classe e um nível então o relacionamento dessas tabelas é de uma para um.

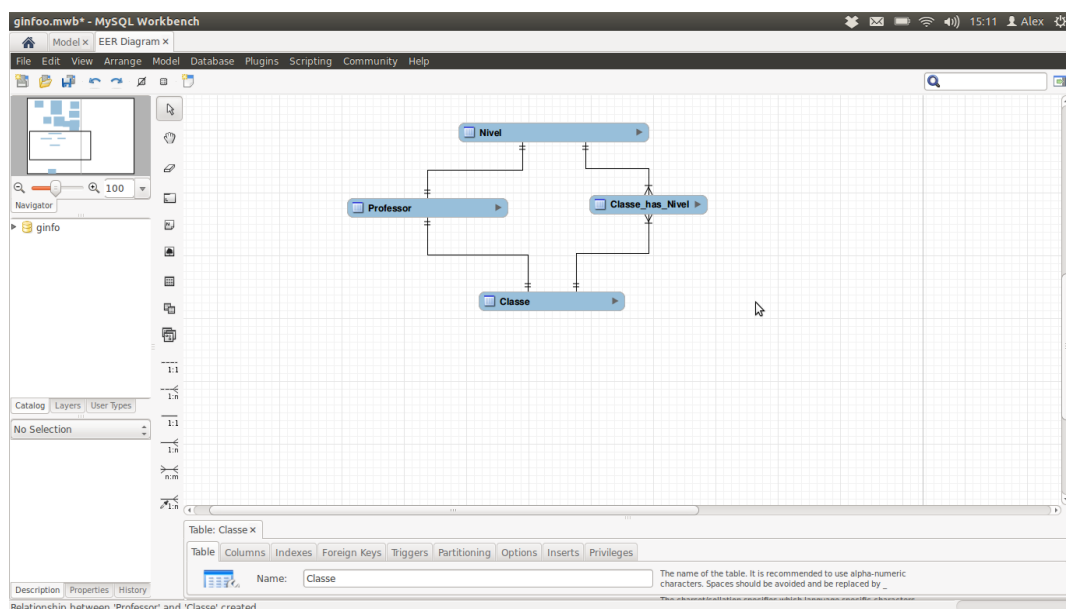


Figura 4.1 Relacionamento classe-nível com professor

Um professor é um funcionário da universidade, podemos ter esta tabela mais como a nossa aplicação sendo orientada a objetos e utilizando que um professor irá ter um extends a funcionário ele irá salvar tudo na mesma tabela. Caso desejemos ter o professor salvo em outra tabela teríamos que mapear na nossa aplicação utilizando o mapping. Mais não utilizamos esse recurso na nossa aplicação.

Então teremos na nossa aplicação que um funcionário pertencera a um departamento e com isso teremos uma tabela “Departamento” e terá um relacionamento de uma para um com a tabela “Professor”. Como mostra a figura 4.2.

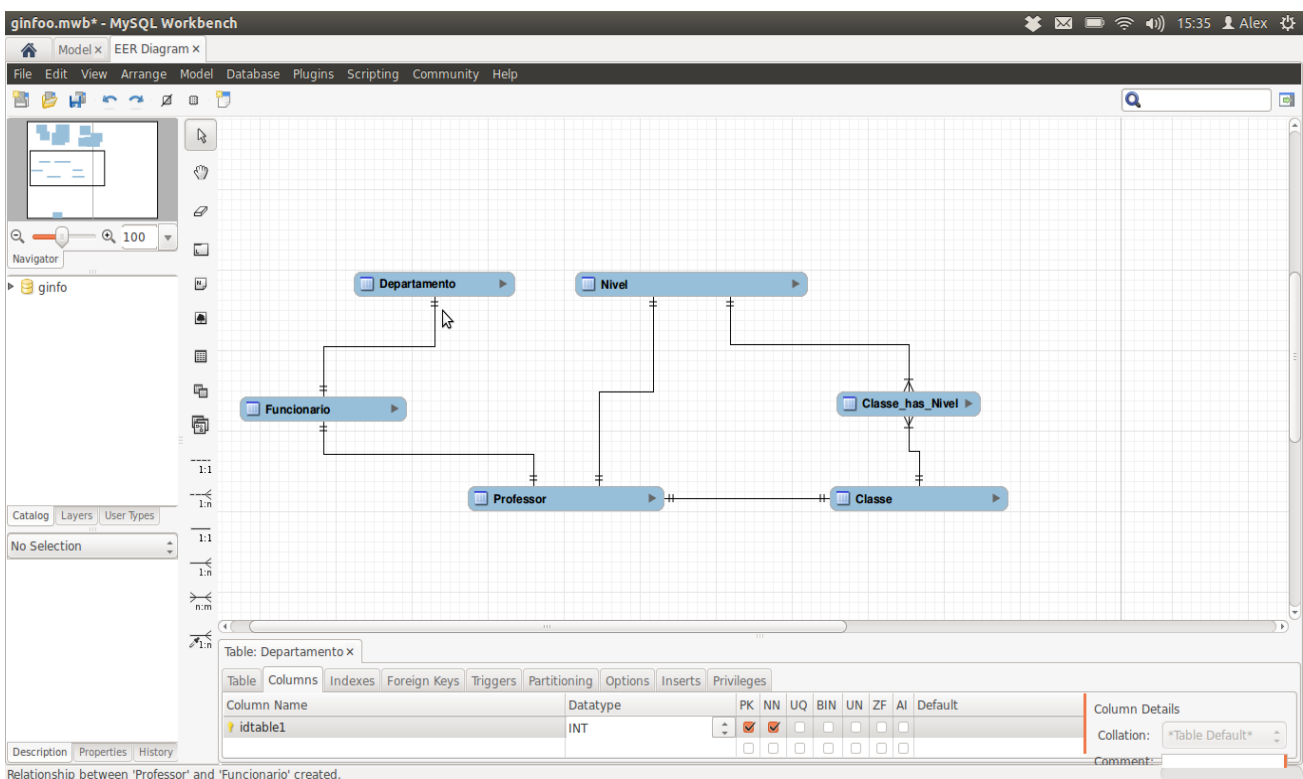


Figura 4.2 Relação funcionário com departamento.

Vimos a necessidade de amarrar o funcionário, com o departamento e não exatamente já o professor, pois teremos que ter o perfil de secretaria na nossa aplicação, que será um funcionário, e será esta pessoa quem irá aprovar as atividades cadastradas pelo professor, e também quem estará analisando os pedidos de memorial do professor, sendo esse memorial o meio para que ele consiga ser promovido de classe ou nível.

A criação da tabela de departamento foi executado com o seguinte código SQL.

```

CREATE TABLE IF NOT EXISTS `ginfo`.`departamento` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT ,
  `version` BIGINT(20) NOT NULL ,
  `centro` VARCHAR(255) NOT NULL ,
  `nome` VARCHAR(255) NOT NULL ,
  PRIMARY KEY (`id`) )
ENGINE = InnoDB
AUTO_INCREMENT = 2
DEFAULT CHARACTER SET = latin1
COLLATE = latin1_swedish_ci;

```

***Codigo sql Departamento.*

Agora com a definição do professor, funcionário, iremos modelar a parte de atividades no sistema iremos demonstrar primeiro a situação das atividades administrativas oque não estiver presente nesta seção colocaremos os códigos, nos apêndice no final deste artigo.

Então iremos criar primeiramente uma tabela de 'tipos_de_atividade_administrativa', sendo essa mais para o administrador do sistema poder cadastrar as atividades administrativa em que os professores poderão se cadastrar.

```

CREATE TABLE IF NOT EXISTS `ginfo`.`tipo_de_atividade_administrativa` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT ,
  `version` BIGINT(20) NOT NULL ,
  `nome_da_atividade` VARCHAR(255) NOT NULL ,
  `pontuacao` INT(11) NOT NULL ,
  PRIMARY KEY (`id`) )
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1
COLLATE = latin1_swedish_ci;

```

***Criação da tabela tipo_atividade_administrativa.*

Com a tabela de 'tipos_atividade_administrativa', teremos outra tabela que é a de atividade administrativa, que sera aonde o professor ao preencher uma atividade administrativa ira guardar seus dados. Ela terá uma relação de que pertence a um tipo de atividade administrativa, e um tipo de atividade administrativa terá muitas atividades administrativas. E um professore poderá ter muitas atividades administrativas.

O seguinte código SQL foi executado para a criação da tabela.

```

CREATE TABLE IF NOT EXISTS `ginfo`.`atividade_administrativa` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT ,
  `version` BIGINT(20) NOT NULL ,
  `ambito` VARCHAR(15) NOT NULL ,
  `atividade_id` BIGINT(20) NOT NULL ,
  `data_inicial` DATETIME NOT NULL ,
  `datafinal` DATETIME NOT NULL ,

```



```

`informacoes_adiconais` VARCHAR(255) NOT NULL ,
`portaria_resolucao` TINYBLOB NOT NULL ,
`status_id` BIGINT(20) NOT NULL ,
PRIMARY KEY (`id`) ,
INDEX `FKDCD4362C0305C46` (`status_id` ASC) ,
INDEX `FKDCD4362AABEDBEB` (`atividade_id` ASC) ,
CONSTRAINT `FKDCD4362AABEDBEB`
    FOREIGN KEY (`atividade_id` )
    REFERENCES `ginfo`.`tipo_de_atividade_administrativa` (`id` ),
CONSTRAINT `FKDCD4362C0305C46`
    FOREIGN KEY (`status_id` )
    REFERENCES `ginfo`.`status` (`id` ))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1
COLLATE = latin1_swedish_ci;

```

***Código SQL tabela atividade_administrativa*

E para criarmos o relacionamento de muitos para muitos que um professor tem com uma atividade administrativa criamos a seguinte tabela.

```

CREATE TABLE IF NOT EXISTS
`ginfo`.`professor_atividade_administrativa`(
`professor_atividades_administrativas_id` BIGINT(20) NULL DEFAULT NULL,
`atividade_administrativa_id` BIGINT(20) NULL DEFAULT NULL ,
INDEX `FKF333CB32DC5E9C98` (`professor_atividades_administrativas_id`
ASC) ,
INDEX `FKF333CB32750EBF45` (`atividade_administrativa_id` ASC) ,
CONSTRAINT `FKF333CB32750EBF45`
    FOREIGN KEY (`atividade_administrativa_id` )
    REFERENCES `ginfo`.`atividade_administrativa` (`id` ),
CONSTRAINT `FKF333CB32DC5E9C98`
    FOREIGN KEY (`professor_atividades_administrativas_id` )
    REFERENCES `ginfo`.`shiro_user` (`id` ))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1
COLLATE = latin1_swedish_ci;

```

***Código SQL criação da tabela de professor_atividade_administrativa*

Na criação da tabela de atividade administrativa podemos ver uma FOREIGN KEY “status_id”, que vem da tabela status, e foi criado um relacionamento de que uma atividade administrativa tivesse um status.

O que seria esse status? A necessidade de se criar uma tabela status, vem de que o professor não somente ira cadastrar uma atividade no sistema, essa atividade terá que ser aprovada pelo seu departamento no sistema. Em alguns casos deve se permitir ate que o professor adicione um anexo, para comprovar que realizou esta atividade. A tabela de status também sera utilizado na montagem do memorial do professor aonde ira para aprovação da banca, esse memorial possuíra também um status, e que se for aprovado

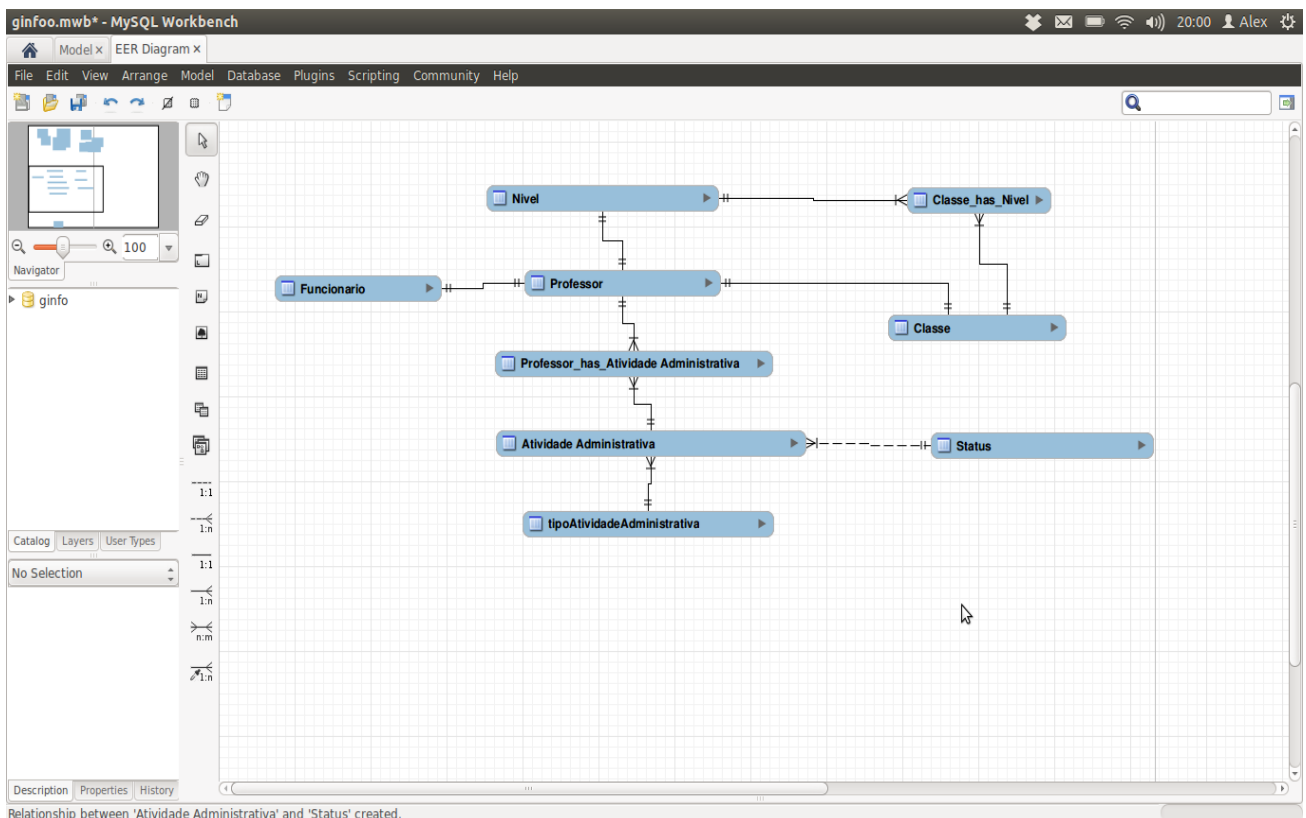
poderá o professor subir de cargo.

Iremos explicar mais adiante como será utilizado na aplicação o status. Agora apenas demonstraremos como foi realizado a criação da tabela status.

```
CREATE TABLE IF NOT EXISTS `ginfo`.`status` (  
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT ,  
  `version` BIGINT(20) NOT NULL ,  
  `status` VARCHAR(255) NOT NULL ,  
  PRIMARY KEY (`id`) )  
ENGINE = InnoDB  
DEFAULT CHARACTER SET = latin1  
COLLATE = latin1_swedish_ci;
```

***Criação da tabela status*

Agora com nossa modelagem de atividade administrativa realizada observamos



então na figura 4.8, como esta ficando nossa modelagem de dados no Workbench.

Figura 4.3 Modelo atividade administrativa.

A modelagem completa se encontra em resultados na próxima seção, e também mais alguns códigos SQL estarão disponíveis no apêndice.

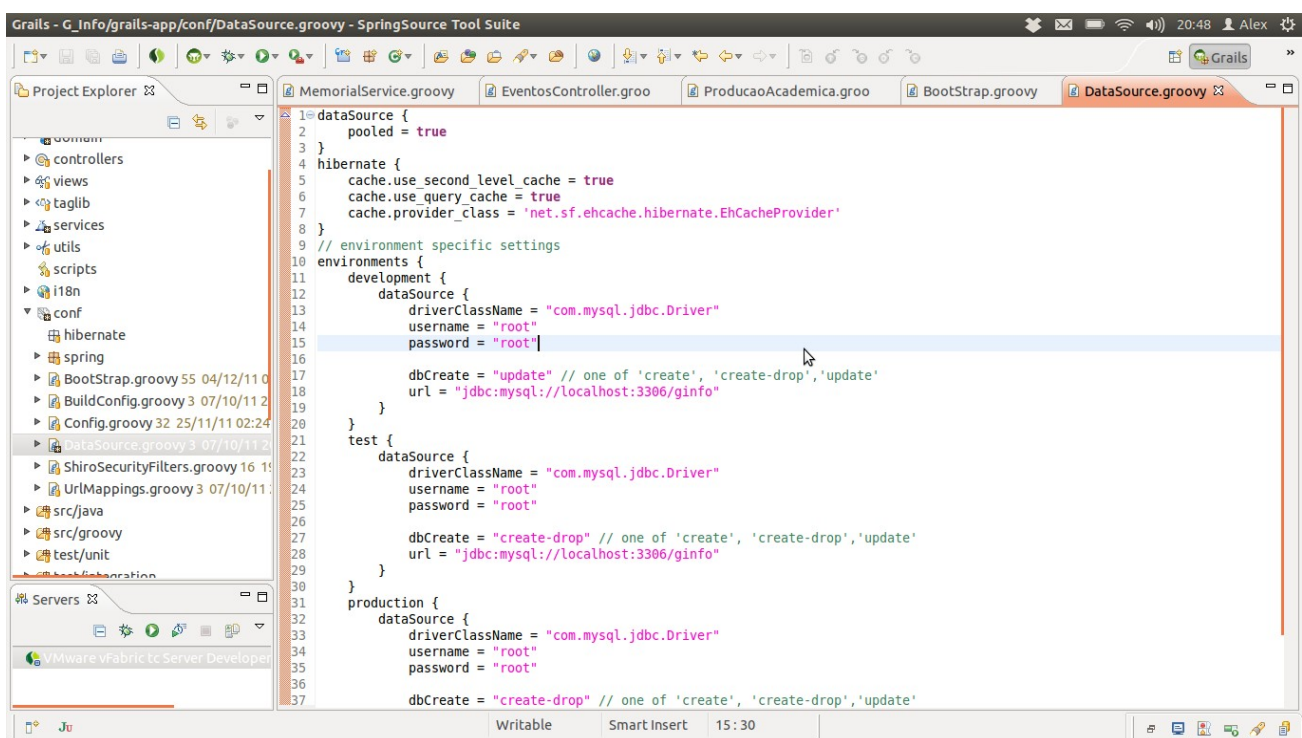
.....4.2 Aplicação com Grails

Agora que já mostramos como esta sendo realizado a criação das tabelas no banco de dados, pelo modelo relacional, iremos demonstrar uma abordagem diferente, que sera da nossa ferramenta de desenvolvimento o grails que através das nossas classes de domínios podemos mapear toda nossa aplicação, através das classes de domínio combinando as com as tabelas que estão no nosso banco de dados.

.....4.2.1 Comunicando com o banco de dados MySQL

Para que a nossa aplicação se comunique com o nosso banco de dados precisamos alterar apenas um arquivo que é o dataSource, é uma das poucas configurações que vamos fazer será conectar nossa aplicação ao banco de dados MySQL, para isso vamos colocar o conector 'mysql-connector-java-5.1.15-bin.jar' na pasta '.\GInfo\lib' e alteramos o arquivo de configuração '\Ginfo\grails-app\conf\DataSource.groovy'.

Como utilizamos a IDE "Spring Source Tools", nos oferece um ambiente fácil para



trabalhamos, trazendo todos diretórios ao lado que o Grails nos oferece, então alteramos o arquivo dataSource que se encontra no diretório conforme mostra a figura abaixo.

Figura 4.4 Configurando dataSource

.....4.2.2 Criando nossas classes de domínios

A classe de domínio cumpre o M no padrão Model View Controller (MVC) e representa uma entidade persistente que é mapeado para uma tabela de banco de dados subjacente. Grails em um domínio é uma classe que vive no grails-app/domain diretório. A classe de domínio podem ser criados com o *'create-domain-class'* comando:

```
grails create-domain
```

Para criar uma classe de domínio no nosso ambiente, e só criar uma classe de extensão groovy dentro do diretório domain. Para o ambiente do Spring, e só clicar em cima do projeto com o botão direito e ir em "New", e depois em domain, e colocarmos o pacote que queremos e depois ponto e o nome da nossa classe em maiúsculo.

Para a continuação do nosso estudo de caso ginjo, iremos então demonstrar como foram criadas as classes de domínio, iremos partir do mesmo exemplo da seção 4.1, aproveitando as nossas tabelas já criadas.

Primeiramente demonstraremos nossa classe de domínio professor. A nossa classe professor veremos que ela extends a funcionário. Observamos também o relacionamento belongsTo a um departamento, já que o professor pertence a um departamento. Por ultimo observamos o relacionamento static hasMany, aonde concluímos que um professor tera muitas atividades administrativas e varias outras atividades definidas na figura 4.10

Foi colocado na classe de professor também os atributos de classe e nível, sabendo que o professor estará em uma classe e em um nível.

Na classe de domínio podemos também definir restrições como esta sendo definido, nas constraints, e também utilizamos a sobrescrita do metodo toString().

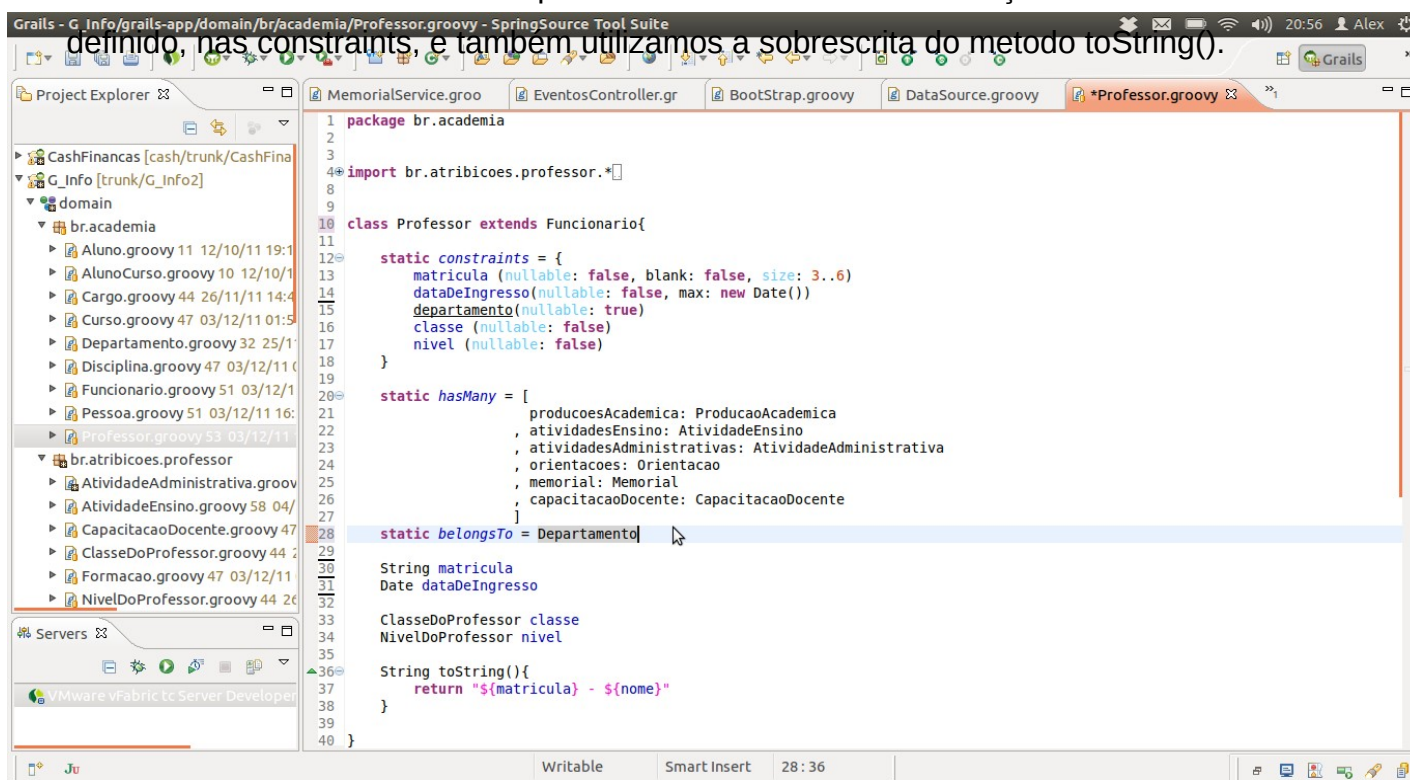
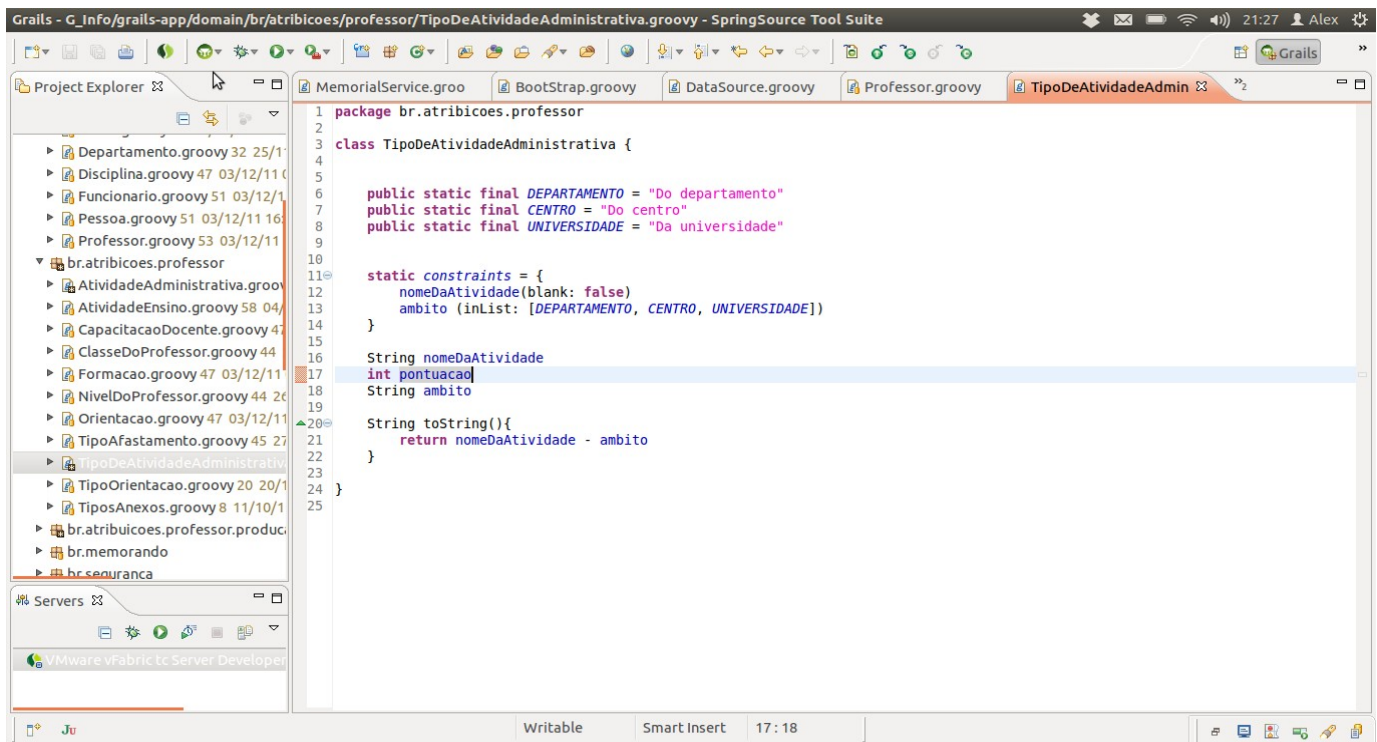


Figura 4.5 Classe de Domínio Professor

Com o nosso professor pronto, agora iremos, dar continuidade mostrando a criação da tabela de atividade administrativa, para que o professor possa cadastrar suas atividades.

Primeiro teremos que criar nossa tabela de “TipoAtividadeAdministrativa”, nossa tabela terá três atributos o de “nomeDaAtividade”, “ambito”, “pontuacao”, para a escolha do âmbito foram criadas três variáveis static com os valores “Departamento”, “Centro”, “Universidade”, quando o usuário escolher o âmbito ele apenas ira ter uma lista com essas três opções, para isso poder acontecer declaramos nas constraints a condição de inList para a variável âmbito.



```
1 package br.atribicoes.professor
2
3 class TipoAtividadeAdministrativa {
4
5     public static final DEPARTAMENTO = "Do departamento"
6     public static final CENTRO = "Do centro"
7     public static final UNIVERSIDADE = "Da universidade"
8
9
10
11     static constraints = {
12         nomeDaAtividade(blank: false)
13         ambito (inList: [DEPARTAMENTO, CENTRO, UNIVERSIDADE])
14     }
15
16     String nomeDaAtividade
17     int pontuacao
18     String ambito
19
20     String toString(){
21         return nomeDaAtividade - ambito
22     }
23
24 }
25
```

Figura 4.6 Classe TipoAtividadeAdministrativa

Próximo passo do nosso estudo de caso Ginfo, e criar a classe de atividade administrativa. Nossa classe terá um atributo que é o tipo de atividade administrativa, a resolução sendo um campo byte onde poderá ser feito um upload do documento, foi adicionado também o campo de status, em que nossa atividade sempre quando criada sera coloca para analise, e encaminhada a um departamento, para aprovação da secretaria do departamento, em que o professor, pertence. Para completar temos os campos de dataInicial, dataFinal, e também o campo informação adicional caso seja

necessário, o professor acrescentar alguma informação.

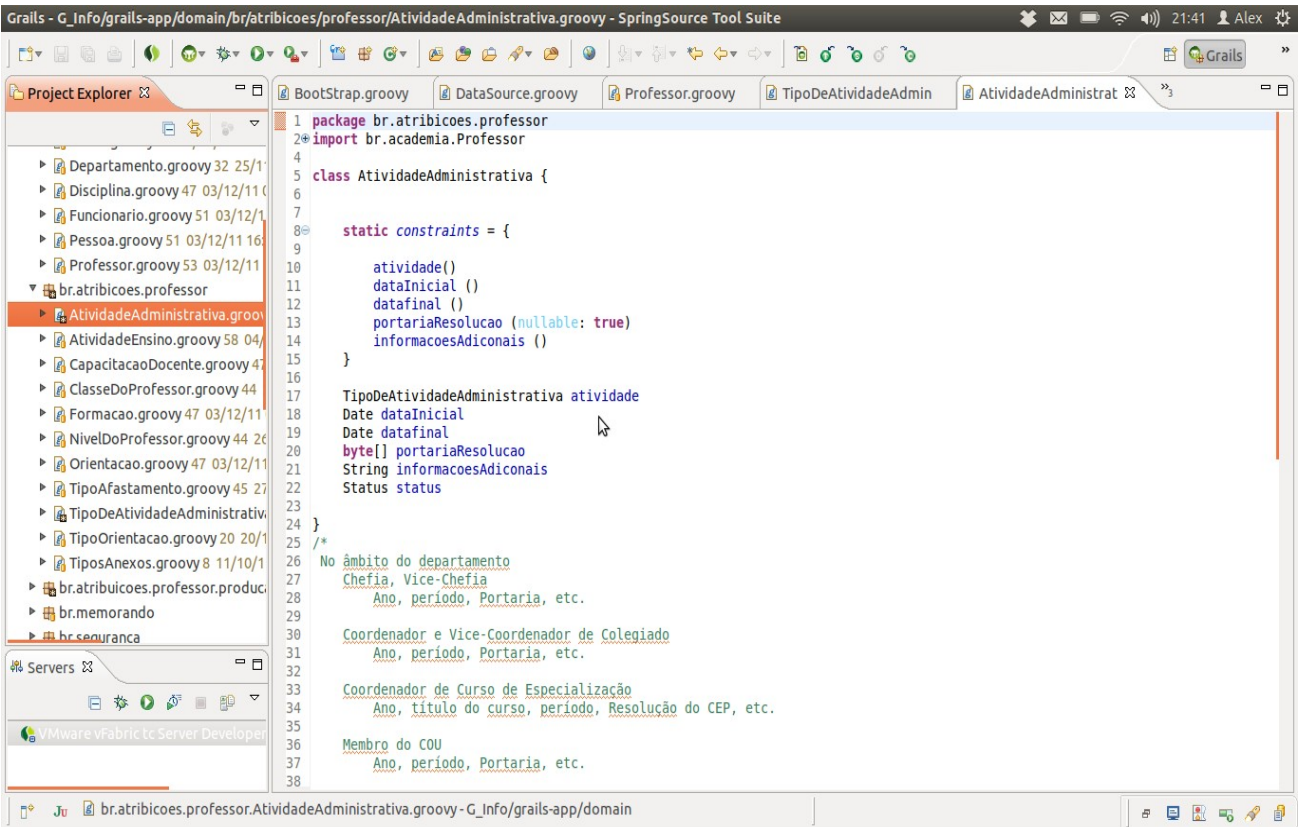


Figura 4.7 Criação da tabela Atividade Administrativa

.....4.2.3 Criando nossos controladores

Um controlador manipula as solicitações e cria ou prepara a resposta e é pedido de escopo. Em outras palavras uma nova instância é criada para cada pedido . Um controlador pode gerar a resposta ou delegar a uma visão. Para criar um controlador de simplesmente criar uma classe cujo nome termina com Controller e colocá-lo dentro do grails-app/controllers diretório.

O padrão de mapeamento URL configuração garante que a primeira parte de seu nome do controlador é mapeado para um URI e cada ação definida dentro de seu controlador para mapas URI dentro da URI nome do controlador.

Tente executar o seguinte comando a partir da raiz de um projeto Grails:

```
grails create-controller professor
```

Considerando que nossas classes de professor e atividade já estão prontas, para podermos começar a gravar nossos dados nela. Como estamos utilizando um framework web que se utiliza do padrão MVC, iremos agora criar nossos controladores para essas classes de domínios para podermos cadastrar professores e consequentemente após

isso o professor poderá cadastrar uma atividade administrativa.

```
def scaffold = Professor

def save = {

    def professorInstance = new Professor(params)

    log.debug("params: "+params)

    ClasseDoProfessor classeInicial = new ClasseDoProfessor(classe :
ClasseDoProfessor.PROFESSOR_AUXILIAR)
    classeInicial.save()
    NivelDoProfessor nivelInicial = new NivelDoProfessor(nivel :
NivelDoProfessor.A )
    nivelInicial.save()

    professorInstance.classe = classeInicial
    professorInstance.nivel = nivelInicial
    professorInstance.totalDePontos = 0

    //define o hash da senha (obrigatorio)
    professorInstance.passwordHash = new
Sha256Hash(professorInstance.passwordHash).toHex()

    //define acesso do professor
    professorInstance.addToPermissions("*:*)

    log.debug("professorInstance PROP: "+professorInstance.classe+" -
"+professorInstance.nivel)

    if (professorInstance.save(flush: true)) {
        flash.message = "${message(code: 'default.created.message', args:
[message(code: 'professor.label', default: 'Professor'),
professorInstance.id])}"
        redirect(action: "show", id: professorInstance.id)
    }
    else {
        render(view: "create", model: [professorInstance:
professorInstance])
    }
}
}
```

***Código do Controller Professor*

Como observamos na figura anterior temos o padrão dos controladores assim que criamos temos que e o “scaffold”, esse comando ira criar para nos todo o crud, operações básicas como create, delete, save. Só que na classe professor precisávamos sobrescrever o método save, então fizemos o save novamente porque precisamos que ao criar um professor ter uma logica a mais, criando um usuário para ele ter acesso ao sistema, e a atribuição já de uma classe e um nível para ele automático que, sera alterado pela secretaria do departamento, que sera quem estará fechando seu cadastro.

O próximo controller que estaremos observando será o da atividade administrativa. Neste controller utilizamos o scaffold para fazer o crud básico, e também sobrescrevemos o método save pela necessidade de pegar o professor logado no sistema que estará, realizando o cadastro da atividade, e após isso estamos adicionando a atividade administrativa nas atividades administrativas do professor. Para realizar isto fizemos um `findByEmail`, que está na sessão com os dados do usuário que está logado na aplicação.

```
package br.atribicoes.professor

import br.academia.Professor
import br.atribicoes.professor.producaoAcademica.ProducaoAcademica

class AtividadeAdministrativaController {

    def scaffold = AtividadeAdministrativa

    def save = {
        def atividadeAdministrativaInstance = new
AtividadeAdministrativa(params)

        Professor professor = Professor.findByEmail(session.user.email)
        if (professor){
            if (atividadeAdministrativaInstance.save(flush: true)) {

                professor.addToAtividadesAdministrativas(atividadeAdministrativaInstance)
                flash.message = "${message(code:
'default.created.message', args: [message(code: 'PublicacaoLivros.label',
default: 'PublicacaoLivros'), atividadeAdministrativaInstance.id])}"
                redirect(action: "show", id:
atividadeAdministrativaInstance.id)
            }else{
                flash.error = "ERROR"
                redirect(action: "show", id:
atividadeAdministrativaInstance.id)
            }
        }
        else {
            flash.message("Professor Invalido no Sistema!")
            render(view: "create", model: [publicacaoLivrosInstance:
atividadeAdministrativaInstance])
        }
    }
}
```

***Código Controller atividade administrativa*

Como podemos ver, foram geradas diversas tabelas, e faremos uma pequena discussão de como ficou o relacionamento das atividades do professor. Como foi mostrado passo-a-passo no nosso artigo o desenvolvimento da atividade administrativa, também tivemos a modelagem da atividade de ensino onde professor também possuirá uma relação de muitas atividades de ensino. A pontuação não é mostrada para o usuário ela é calculado por Ajax na aplicação depende do tanto de horas que o professor colocar

na atividade de ensino.

Não existe uma classe tipo atividade de ensino, existe apenas um inlist definido nas constraints da aplicação para o o professor escolher o tipo de atividade de ensino. Então tendo a tabela de atividade de ensino apenas uma tabela. Com as informações que o professor precisa preencher conforme está na resolução.

Continuando a análise do nosso modelo visando o armazenamento das informações das atividades cadastradas pelo professor temos também a situação considerada mais complexa que são as produções acadêmicas aonde foram divididas em varias tabelas como por exemplo de artigos, eventos, publicação livros, bancas defesa e mais algumas conforme está no modelo.

Mais para que tudo isso funcionasse criamos uma tabela de “GrupoProducaoAcademica”, onde estará informações como a classe em que ela sera gravada no banco a produção acadêmica exemplo “**Artigo**”, ira informa o nome do grupo como por exemplo “**Artigos publicados em periódicos especializados indexados**”, após criado essa tabela teremos a tabela de “tiposAtividadesProducoessAcademica”, onde conterà o nome da atividade, o grupo que ela pertence, e a pontuação dela. Um exemplo de produção acadêmicas então seria esse “**Artigos publicados em periódicos especializados indexados**”, do tipoAtividadeProducaoAcademica “**INTERNACIONAL**”, com **50 pontos**, assim o profesor escolhera o grupo que ele quer escolher e após isso criar a produção acadêmica. A nível de banco foram criadas essas 3 tabelas. E uma para cada classe vista que seria necessario pela resolução como de artigo, bancasLivre, bancasQualificacao, eventos, e etc.

No nosso modelo observa-se também a criação do memorial que nada mais é o pedido do professor de aumento, essa classe funciona como um histórico mesmo de pedidos que ele fez de memorial, onde sera encaminhado para o departamento esse pedido, e terá um status para saber se esse pedido do professor teve algum sucesso ou não

.....5.2 Aplicação

Neste item apresentaremos como ficou realmente a aplicação, demonstrando os resultados obtidos pela criação das nossas classes de domínio e consequentemente dos resultados obtidos pelo controllers criados criados no nosso desenvolvimento.

Primeiro observaremos a criação do professor, que também sera um usuário do sistema, pelos dados da classe professor e pelo nosso controller para ele obtemos o seguinte resultado conforme a figura 5.3.

Home Lista de Professor

Cria Professor

Nome de Usuário

Senha

Nome

Telefone

Matricula

Data De Ingresso

Departamento

CPF

Email

Figura 5.3 Create Professor

Mais visitados Getting Started Latest Headlines

Home Lista de Professor Incluir Professor

Professor 12 criado

Id 12

Username carniel

Nome Carniel

Telefone 333333333

Matricula 8989

Data De Ingresso 05/10/2000

Total De Pontos 0

Departamento DIN

Classe Professor Auxiliar

Nivel A

CPF 999999999

Atividades Administrativas

Atividades Ensino

Email carniel@gmail.com

Memorial

Orientacoes

Password Hash a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3

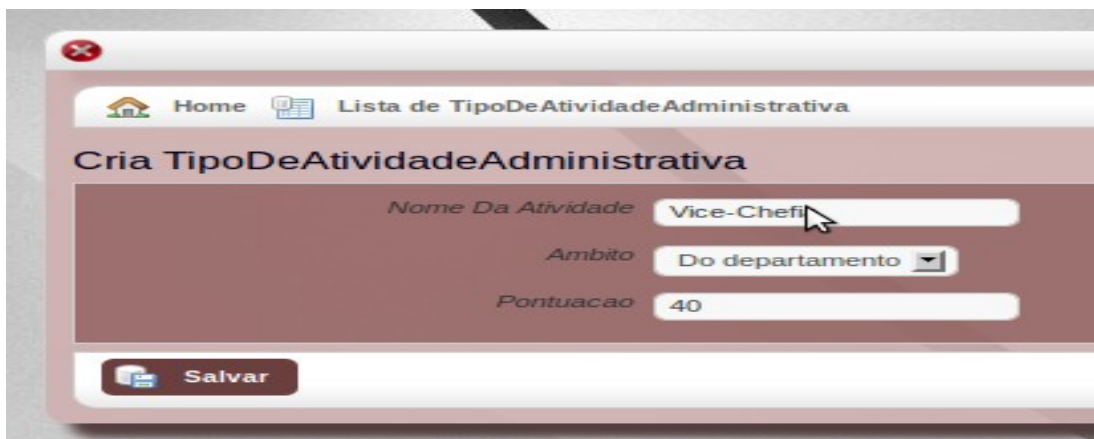
Permissions [":"]

Producoes Academica

Figura 5.4 Show Professor

Após visualizar como esta sendo realizado a criação do professor iremos demonstrar o próximo resultado do nosso desenvolvimento ,que seria a criação dos tipos de atividades administrativa e na sequencia a criação de atividades administrativa por um professor

A criação do “tipoAtividadeAdministrativa”, acontece da seguinte maneira.



The screenshot shows a web browser window with a title bar containing a close button and two tabs: 'Home' and 'Lista de TipoDeAtividadeAdministrativa'. The main content area is titled 'Cria TipoDeAtividadeAdministrativa'. It contains three input fields: 'Nome Da Atividade' with the value 'Vice-Chefe', 'Ambito' with a dropdown menu showing 'Do departamento', and 'Pontuacao' with the value '40'. At the bottom of the form is a 'Salvar' button.

Figura 5.5 Create TipoAtividadeAdministrativa



The screenshot shows the GINFO SISTEMA DE RECURSOS HUMANOS interface. The top navigation bar includes 'Home', 'Cadastros básicos', 'Professores', and 'Secretarias'. Below this is a modal window titled 'TipoDeAtividadeAdministrativa 2 criado'. The modal displays the following information: 'Id 2', 'Nome Da Atividade Vice-Chefe', 'Ambito Do departamento', and 'Pontuacao 40'. At the bottom of the modal are 'Editar' and 'Excluir' buttons.

Figura 5.6 Show Tipo Atividade Administrativa

Tendo nosso tipo de atividade administrativa no sistema, então entraremos como professor, e escolheremos para cadastrar uma atividade administrativa, e realizaremos o cadastro de uma atividade administrativa conforme segue a imagem.

The screenshot shows a web application interface for creating an administrative activity. At the top, there is a navigation bar with 'Home' and 'Lista de AtividadeAdministrativa'. The main heading is 'Cria AtividadeAdministrativa'. The form contains the following fields:

- Atividade:** A dropdown menu with 'Chefia' selected.
- Data Inicial:** Three dropdown menus for day, month, and year, with values '5', 'Dezembro', and '2011' respectively.
- Datafinal:** Three dropdown menus for day, month, and year, with values '5', 'Dezembro', and '2013' respectively.
- Portaria Resolucao:** A text input field containing '/home/alex/Downloads/cifrao.jpg' and a 'Selecionar arquivo...' button.
- Informacoes Adiconais:** A text input field containing 'Solicito esta atividade'.

Figura 5.7 Create Atividade Administrativa

Esta próxima pagina, é mostrada sempre que criamos algo no sistema ele mostra um resumo do que foi inserido no sistema e observamos que a atividade administrativa cadastrada está para analise, esperando ser aprovado pelo departamento para contar ponto para o professor.

The screenshot shows a web application interface displaying the details of a created administrative activity. The navigation bar includes 'Home', 'Lista de AtividadeAdministrativa', and 'Incluir AtividadeAdministrativa'. A blue banner at the top reads 'PublicacaoLivros 3 criado'. The main content area displays the following information:

- Id:** 3
- Atividade:** Chefia
- Data Inicial:** 05/12/2011
- Datafinal:** 05/12/2014
- Portaria Resolucao:**
- Informacoes Adiconais:** Solicito esta atividade
- Status:** Em análise

At the bottom, there are two buttons: 'Editar' and 'Excluir'.

Figura 5.8 Show Atividade Administrativa

6 CONCLUSÃO

Neste artigo conhecemos alguns conceitos de banco de dados, como a modelagem dos dados, a criação das tabelas, as consultas ao banco, as restrições de chave. Na perspectiva da aplicação mostramos como estão sendo feita a criação dos domínios, aninhando as operação do banco de dados com a aplicação, porque elas não são compatíveis e precisamos fazer o mapeamento objeto relacional (ORM) através do GORM que o grails nos oferece.

Nesta ultima etapa do trabalho conseguimos fazer o mapeamento pelo GORM, e gerando os controller para manipular os dados das nossas aplicação. Operações de SQL, consultas no banco foram feitas em service com transactions para garantir as consultas dos dados.

Por final chegamos a uma aplicação web, onde cadastrarmos e armazenamos as informações da carreira academica do professor conforme a **Resolução Nº 061/2003-CEP**. E assim sendo encaminhadas para o departamento aprovar ou nao as atividades, contando pontos para o professor e possibilitando ao professor subir de nivel.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ABRAHAM SILBERSCHATZ Ano: 2006; Edição: 5
Sistema de Banco de Dados

GLEN SMITH AND PETER LEDBROOK Ano: 2009 Edição: 2
Grais in Action

[http://grails.org/doc/1.0.x/guide/5.%20Object%20Relational%20Mapping%20\(GORM\).html](http://grails.org/doc/1.0.x/guide/5.%20Object%20Relational%20Mapping%20(GORM).html)

<http://mysql.com>

<http://grails.org>