Controle de Votos na Urna

Estevam Galvão Albuquerque - 16/0005663
Hugo Nascimento Fonseca - 16/0008166
José Luiz Gomes Nogueira - 16/0032458
Rafael Martins Pereira Chianca - 15/0145608

[1] Universidade de Brasília
[2] Campus Universitário Darcy Ribeiro, Asa Norte, Brasília-DF, Brasil
[3] Instituto de Ciências Exatas, Departamento de Ciência da Computação

Abstract. Este documento consiste no relatório do projeto final da disciplina de Bancos de Dados, semestre 2018/1, na Universidade de Brasília. Agradecimento especial para a Prof. Dra. Maristela Terto de Holanda pelo excelente trabalho desenvolvido junto aos seus alunos.

Keywords: Bancos de Dados · Eleições · CRUD · SQL.

1 Objetivos

O presente projeto consiste em desenvolver um banco de dados para o controle de votos na urna durante um processo eleitoral. Além disso, deve ser desenvolvida uma aplicação para manipulação do banco atráves de funções *CRUD (Create, Retrieve, Update, Delete).*

1.1 Requisitos do Projeto

- Utilizar ao menos uma *view* e uma *Procedure* (com comandos condicionais).
- Mostrar os diagramas Entidade-Relacionamento e Relacional do banco de dados.
- Avaliar as formas normais em cinco tabelas do banco de dados.
- Mostrar a realização de cinco consultas em álgebra relacional, envolvendo pelo menos três tabelas.
- Mostrar o script SQL para geração do banco de dados.
- Descrever e construir diagrama da camada de persistência da aplicação.

2 Sobre o Desenvolvimento

Todos os documentos, códigos-fonte e outros artefatos de software aqui mencionados podem ser encontrados e acompanhados no repositório público do projeto presente no GitHub[1].

3 Modelagem do Banco de Dados

3.1 Modelo Relacional e Entidade-Relacionamento

O Modelo Entidade Relacionamento nos fornece uma visão geral dos dados em alto nível, ou seja, não existe foco sobre como os dados serão armazenados. Segue abaixo o diagrama entidade relacionamento do modelo implementado.

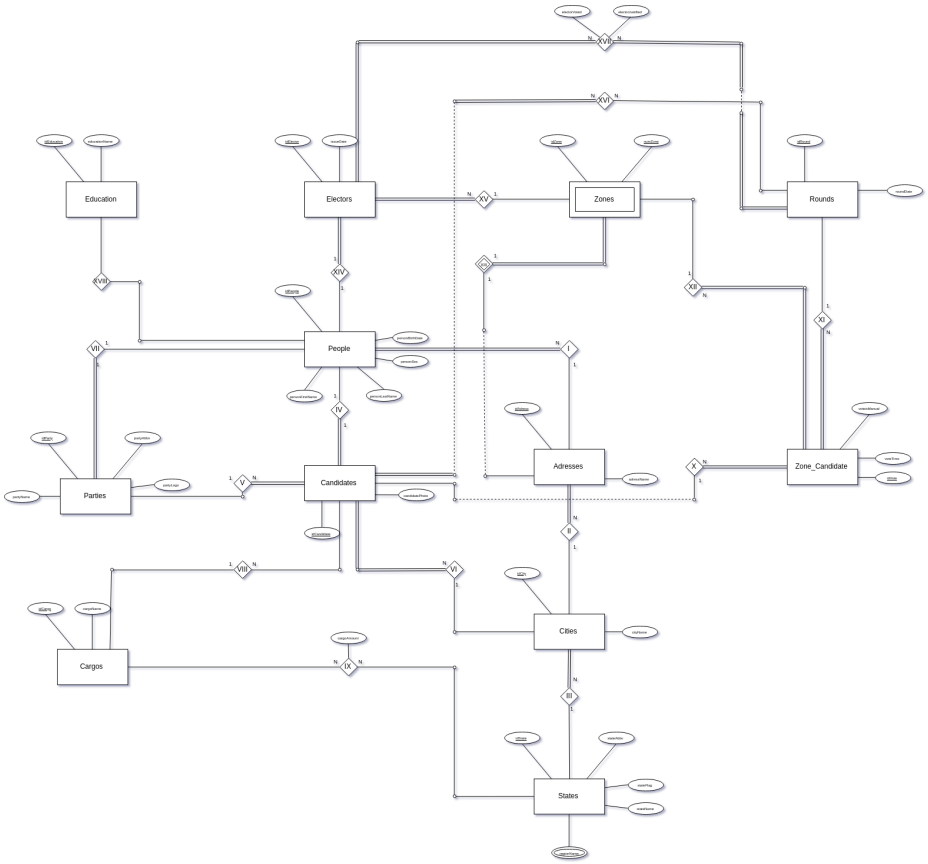

Fig. 1. Diagrama Entidade Relacionamento para o banco de dados

O Modelo Relacional, por sua vez, representa os dados em um esquema contendo uma coleção de tabelas relacionadas entre si. Vale ressaltar que essa representação independe do SGBD em que se está trabalhando, dito isso, segue abaixo o diagrama relacional do modelo implementado, produzido no MySQL Workbench.

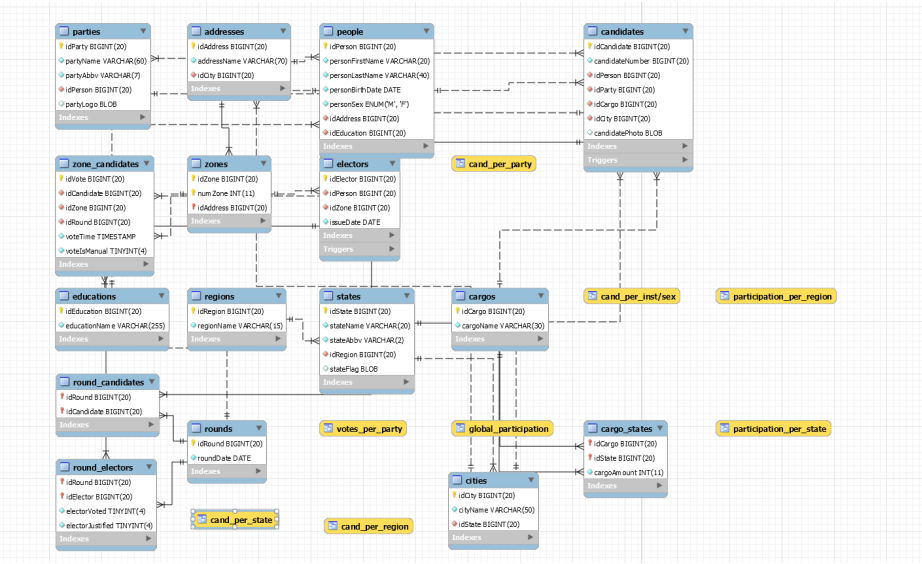

Fig. 2. Diagrama Relacional para o banco de dados

Após a análise de ambos diagramas, é possível ver o papel das principais entidades do banco:

1. People: cada instância representa uma pessoa no banco de dados, contendo suas informações pessoais, dentre essas usa-se o CPF como chave primária.
2. Electors: cada instância representa um eleitor no banco, sendo que o eleitor é identificado pelo seu título de eleitor e é uma especialização da entidade People.
3. Candidate: cada instância representa um candidato do processo eleitoral, sendo que o candidato é uma especialização da entidade People.
4. Parties: cada instância representa um partido político envolvido na eleição.
5. Cargos: cada instância representa um dos cargos disponíveis no processo. Essa entidade é relacionada com cada Estado em um relacionamento N:M
6. Addresses: cada instância representa um endereço do território nacional, sendo o CEP a sua chave primária.
7. Zones: cada instância associa o endereço onde existe uma urna com os seus eleitores.
8. Round: cada instância representa um turno da eleição, guardando a sua data de realização. Essa entidade é relacionada com os seus candidatos e eleitores participantes.
9. Zone_Candidates: é um relacionamento N:M onde suas instâncias representam os votos que um candidato recebeu em uma determinada zona eleitoral.

3.2 Álgebra Relacional

Essa seção é dedicada a mostrar como seriam feitas algumas consultas ao banco de dados utilizando a notação da álgebra relacional, explicitando qual seria o significado da tabela resultante da consulta .

$$\pi_{regionName,stateName,stateAbbv,cityName}(\sigma_{states.idRegion=regions.idRegion\ AND\ states.idState=cities.idState}(states\ x\ regions\ x\ cities))$$

Fig. 3. Tabela que mostra a qual estado e região uma cidade pertence

$$\pi_{personFirstName,personLastName,addressName,educationName}(\sigma_{people.idEducation=educations.idEducation\ AND\ people.idAddress=addresses.idAddress}(people\ x\ educations\ x\ addresses))$$

Fig. 4. Tabela com todas as pessoas, seus endereços e níveis de educação

$$\pi_{c.idCandidate,personFirstName,personLastName,partyAbbv,cargoName,cityName}(\sigma_{c.idCargo=cargos.idCargo\ AND\ c.idParty=pr.idParty\ AND\ c.idPerson=pp.idPerson\ AND\ c.idCity=cities.idCity}(\rho_{pp}(people)\ x\ \rho_{pr}(parties)\ x\ cargos\ x\ cities\ x\ \rho_{c}(candidates)\ ))$$

Fig. 5. Tabela com candidatos, seus partidos, qual cargo estão concorrendo e em qual cidade concorrem

$$\pi_{e.idElector,personFirstName,personLastName,idRound,electorVoted,electorJustified}($$
$$\sigma_{re.idElector=e.idElector\ AND\ e.idPerson=p.idPerson}($$
$$\rho_p(people)\ x\ \rho_{re}(round-electors)\ x\ \rho_e(electors)\ ))$$

Fig. 6. Tabela com os candidatos por turno, além de seus nomes, cargos e partidos

$$\pi_{personFirstName,personLastName,partyAbbv,cargoName,idRound}($$
$$\sigma_{rc.idCandidate=ca.idCandidate\ AND\ ca.idPerson=p.idPerson\ AND\ ca.idParty=pr.idParty\ AND\ ca.idCargo=c.idCargo}($$
$$\rho_p(people)\ x\ \rho_{pr}(parties)\ x\ \rho_c(cargos)\ x\ \rho_{rc}(round-candidates)\ x\ \rho_{ca}(candidates)\ ))$$

Fig. 7. Tabela que mostra a qual estado e região uma cidade pertence

3.3 Normalização

Essa seção é dedicada a mostrar a avaliação das formas normais em cinco tabelas do banco de dados, explicitando cada uma delas (**1FN**, **2FN** e **3FN**).

1. Tabela states
   - **1FN**: Como todos os atributos desta tabela já são atômicos, a primeira forma normal já está aplicada.
   - **2FN**: States (idState, stateName, stateAbbv, idRegion {FK}, stateFlag). Com a primeira forma normal aplicada, pode-se verificar na tabela se existem colunas que não são funcionalmente dependentes da(s) chave(s) primária(s) desta tabela. A tabela em questão já se encontra na segunda forma normal, uma vez que todos seus atributos são funcionalmente dependentes do idState, que rege todas as outras informações do estado, como abreviação, nome e bandeira. O outro atributo presente nessa tabela é uma chave estrangeira para a tabela de regiões.
   - **3FN**: Analisando a tabela para ver se a mesma se apresenta na terceira forma normal, devemos "ignorar" a chave primária e ver se algum dos atributos é dependente de um outro atributo desta mesma tabela. States (_____, stateName, stateAbbv, idRegion FK, stateFlag). Como todos os atributos dessa tabela são definidos por idState, não há necessidade de gerar nenhuma tabela a partir dessa, ou seja, essa tabela já se encontra na terceira forma normal.
2. Tabela cities
   - **1FN**: Como todos os atributos desta tabela já são atômicos, a primeira forma normal já está aplicada.
   - **2FN**: Cities (idCity, cityName, idState {FK}). Com a primeira forma normal aplicada, pode-se verificar na tabela se existem colunas que não são funcionalmente dependentes da(s) chave(s) primária(s) desta tabela.

A tabela em questão já se encontra na segunda forma normal, uma vez que todos seus atributos são funcionalmente dependentes do idCity, que rege todas as outras informações da cidade, como seu nome. O outro atributo presente nessa tabela é uma chave estrangeira para a tabela de estados.

- **3FN**: Analisando a tabela para ver se a mesma se apresenta na terceira forma normal, devemos "ignorar" a chave primária e ver se algum dos atributos é dependente de um outro atributo desta mesma tabela. Cities (_____, cityName, idState {FK}). Como todos os atributos dessa tabela são definidos por idCity, não há necessidade de gerar nenhuma tabela a partir dessa, ou seja, essa tabela já se encontra na terceira forma normal.

3. Tabela adresses
   - **1FN**: Como todos os atributos desta tabela já são atômicos, a primeira forma normal já está aplicada.
   - **2FN**: Addresses (idAddress, addressName, idCity {FK}). Com a primeira forma normal aplicada, pode-se verificar na tabela se existem colunas que não são funcionalmente dependentes da(s) chave(s) primária(s) desta tabela. A tabela em questão já se encontra na segunda forma normal, uma vez que todos seus atributos são funcionalmente dependentes do idAddress, que rege todas as outras informações da tabela, como seu nome (que no caso é a string que contém o endereço propriamente dito). O outro atributo presente nessa tabela é uma chave estrangeira para a tabela de cidades.
   - **3FN**: Analisando a tabela para ver se a mesma se apresenta na terceira forma normal, devemos "ignorar" a chave primária e ver se algum dos atributos é dependente de um outro atributo desta mesma tabela. Addresses (_____, addressName, idCity {FK}). Como todos os atributos dessa tabela são definidos por idAddresses, não há necessidade de gerar nenhuma tabela a partir dessa, ou seja, essa tabela já se encontra na terceira forma normal.
4. Tabela people
   - **1FN**: Como todos os atributos desta tabela já são atômicos, a primeira forma normal já está aplicada.
   - **2FN**: People (idPerson, personFirstName, personLastName, personBirthDate, personSex, idAddress ¡FK¿, idEducation {FK}). Com a primeira forma normal aplicada, pode-se verificar na tabela se existem colunas que não são funcionalmente dependentes da(s) chave(s) primária(s) desta tabela. A tabela em questão já se encontra na segunda forma normal, uma vez que todos seus atributos são funcionalmente dependentes do idPerson, além de suas chaves estrangeiras para as tabelas de endereços e educação.
   - **3FN**: Analisando a tabela para ver se a mesma se apresenta na terceira forma normal, devemos "ignorar" a chave primária e ver se algum dos atributos é dependente de um outro atributo desta mesma tabela. People (_____, personFirstName, personLastName, personBirthDate, personSex, idAddress {FK}, idEducation {FK}). Assim como nas outras

tabelas descritas acima, fica evidente que essa tabela está completamente normalizada.

5. Tabela cargo_states
   - **1FN**: Como todos os atributos desta tabela já são atômicos, a primeira forma normal já está aplicada.
   - **2FN**: Cargo_States (idCargo {FK}, idState {FK}, cargoAmount). Com a primeira forma normal aplicada, pode-se verificar na tabela se existem colunas que não são funcionalmente dependentes da(s) chave(s) primária(s) desta tabela. A tabela em questão já se encontra na segunda forma normal, uma vez que todos seus atributos são funcionalmente dependentes simultaneamente de idCargo e idState.
   - **3FN**: Analisando a tabela para ver se a mesma se apresenta na terceira forma normal, devemos "ignorar" a chave primária e ver se algum dos atributos é dependente de um outro atributo desta mesma tabela. Cargo_States (_____, _____, cargoAmount). Como só sobrou um atributo que não é chave primária, a tabela está, automaticamente, normalizada.

3.4 Implementação no MySQL

- Triggers
  1. Tabela Electors: contém um Trigger *AFTER INSERT* para registrar o eleitor recém inserido nos turnos da eleição.
  2. Tabela Candidates: contém um Trigger *AFTER INSERT* para registrar o candidato recém inserido nos turnos da eleição.
- Procedures
  1. get_candidates: essa função retorna os candidatos que podem receber voto dado o eleitor atualmente votando, ou seja, seus argumentos são o id do estado, id do cargo e o id do turno atual.
  2. get_result: essa função calcula o resultado da votação, ordenando do candidato mais votado para o menos votado, dado o turno, cargo e estado.
- Functions
  1. total_votes: essa função é utilizada para gerar as estatísticas e calculo do resultado da eleição, uma vez que os valores são mostrados em porcentagem, é necessário saber qual foi o total de votos para um determinado cargo, em um determinado estado e em determinado turno.
- Views

  As views desse banco têm o intuito de calcular estatísticas sobre o processo eleitoral, agrupando por características distintas. As views disponíveis são:
  1. cand_per_inst_sex: mostra o número de candidatos, agrupados por cada escolaridade, dividido entre os dois sexos.
  2. cand_per_party: mostra o número de candidatos inscritos por partido.
  3. cand_per_region: mostra o número de candidatos inscritos por região.
  4. cand_per_state: mostra o número de candidatos inscritos por estado.
  5. global_participation: mostra estatistícas sobre a participação nacional do eleitorado.

6. participation_per_region: mostra estatistícas sobre a participação do eleitorado por região.
7. participation_per_state: mostra estatistícas sobre a participação do eleitorado por estado.
8. votes_per_party: mostra o número de votos de cada partido

4 Aplicação desenvolvida

4.1 Descrição

A aplicação para a manipulação/interface com o banco de dados foi desenvolvida em Ruby (v.2.5.1), baseada no framework Ruby on Rails (v.5.2). A aplicação foi realizada seguindo os padrões de programação em rails, entretando algumas operações foram realizadas em mais baixo nível, para melhor visualização do funcionamento da comunicação com o banco de dados e os tipos de dados retornados, porém ainda utilizando a camada de comunicação com banco de dados do rails. As operações de CRUD foram realizadas utilizando a camada de persistência já criada contida na classe ActiveRecords do rails

4.2 Arquitetura MVC e Camada de Persistência

O padrão de arquitetura do rails é o MVC, logo nossa camada de aplicação é como ilustrado abaixo (Figura 8):

Onde as views são as aplicações (onde o usuário ira interagir), a controller é a camada media entre a view e a model (onde é tratado os requests do usuário por meio do protocolo http) e a model é o objeto relacional do banco de dados, que herda a classe Active Records do rails, onde contém as conexões e persistência polimorfisados do banco de dados.
As comunicações entre MVC por padrão podem ser tanto View >Controler >Model quanto o apresentado no diagrama, onde a view pode se comunicar diretamente com a model, uma vez que a camada de persistência reside na Model, pulando assim somente a controller.
Entretanto pelo padrão rails, não são todas as funções encapsuladas que possuem permissão para serem chamadas diretamente na view por questões de segurança.
Abaixo temos as localizações para cada arquivo MVC e arquivo de configuração do banco de dados.

- Controller: app >controllers
- Model: app >models
- View: app >views
- config DataBase: config >database.yml

Aprofundado um pouco mais na camada de persistência do rails, temos que a classe ActiveRecords encapsula praticamente todas as camadas para tratamento de manipulação, dentro destas camandas estão algumas importantes que podemos resaltar:

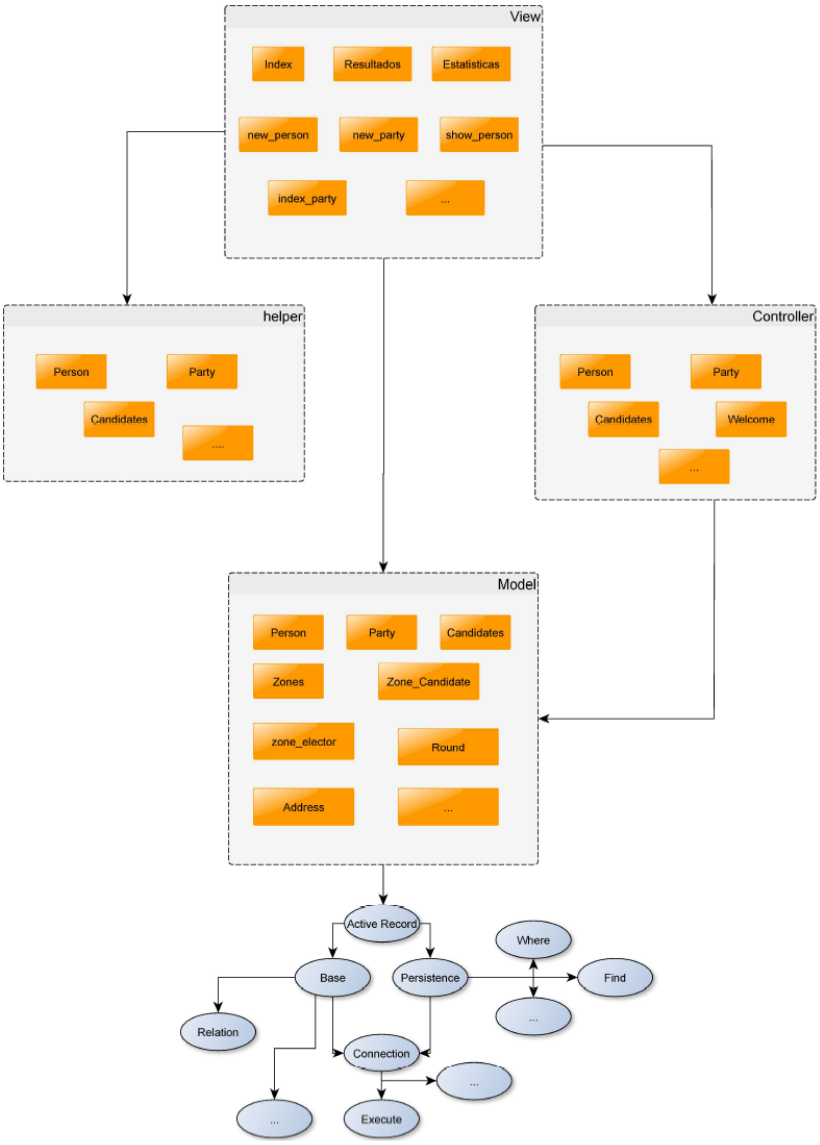

Fig. 8. Diagrama de Camada de persistência

– Conection: Permite acesso direto ao banco de dados, trantando somente da conexão. Possui methodos para escrita direta de querys. Os metodos deste modulo foram utilizados nas views e procedures do banco de dados em Controllers >welcome e Helpers >welcome.
– Base: Utilizado em todo CRUD, seus methodos são resposáveis pela manipulação e geração do objeto que representa aquela tabela.
– Persistence: Juntamente com :Base foi utilizado em todo o CRUD , sendo resposável pela deletação e atualização de um ou vários objetos da tabela.
– QueryMethods: Utilizado em alguns pontos diretamente da view e em controllers, sendo responsável pelo encapsulamento de querys especificas (where, selec, from e etc), podendo ser utilizado em conjunto para formar uma única query (por exemplo selec().where().from()), possibilitando assim maior flexibilidade ao acesso a camada de persistência. Entretanto pelo padrão de programação rails deveria-se ser utilizado somente em helpers, controllers e models, deixando para acesso direto na view somente encapsulamentos completos, minimizando o código na view, maximizando segurança entre outros.

Entretanto esse modulo foi projetado para tabelas que possuem uma única chave primária, logo algumas funções, ficam inútilizadas, entratanto podem ser contornadas. Um exemplo de função inútilizada é a tentativa de deletação em cascata de algum objeto que esta atrelado a alguma tabela que possui mais de uma chave primária, por exemplo Electors que esta atrelado a Round_Electos (Elector (fk)(pk), Round (fk)(pk)), logo essa tentativa ocasionará em um erro e o objeto não será deletado, sendo necessário a delatação em Round_elector primeiro. Para contornar essa questão pode se criar metodos na Model conflitante que substitua o metodo de deletação do modulo.

4.3 Executando a Aplicação

Para a execução da aplicação deve-se ter instalado previamente as versões do ruby e do rails supracitadas e o yarn (Gerenciador de dependências). Uma vez instalado e configurado o ambiente ruby on rails e o yarn, deve-se alterar o arquivo database.yml para o banco de dados local. Por fim deve-se executar dentro da pasta bin os comandos em ordem:

– gem install bundler
– bundler install
– yarn install
– rake db:create
– rake db:migrate
– rails s

References

1. Repositório GitHub, https://www.github.com/01oseluiz/BD_Eleicoes. Last accessed 20 Jun 2018

2. Modulo Base ActiveRecords, http://api.rubyonrails.org/classes/ActiveRecord/Base.html.Last accessed 17 jun 2018
3. Introdução ao Modulo ActiveRecords, http://guides.rubyonrails.org/active_record_basics.html.Last accessed 17 jun 2018
4. Modulo Persistence ActiveRecords, http://api.rubyonrails.org/classes/ActiveRecord/Persistence.html.Last accessed 17 jun 2018
5. Modulo Connection ActiveRecords, https://apidock.com/rails/ActiveRecord/Base/connection.Last accessed 17 jun 2018
6. Modulo QueryMethods ActiveRecords, http://api.rubyonrails.org/classes/ActiveRecord/QueryMethods.html.Last accessed 17 jun 2018
7. Rails Boas Práticas, https://rails-bestpractices.com/tag/model/.Last accessed 17 jun 2018