Seção 01 - Resumo

A base de dados

Durante este projeto teremos uma base de dados com 9 tabelas que vão simular o armazenamento de registros para uma clínica médica. Entre elas, existem relacionamentos dos tipos 1:1, 1:N e N:N., conforme o diagrama a seguir:

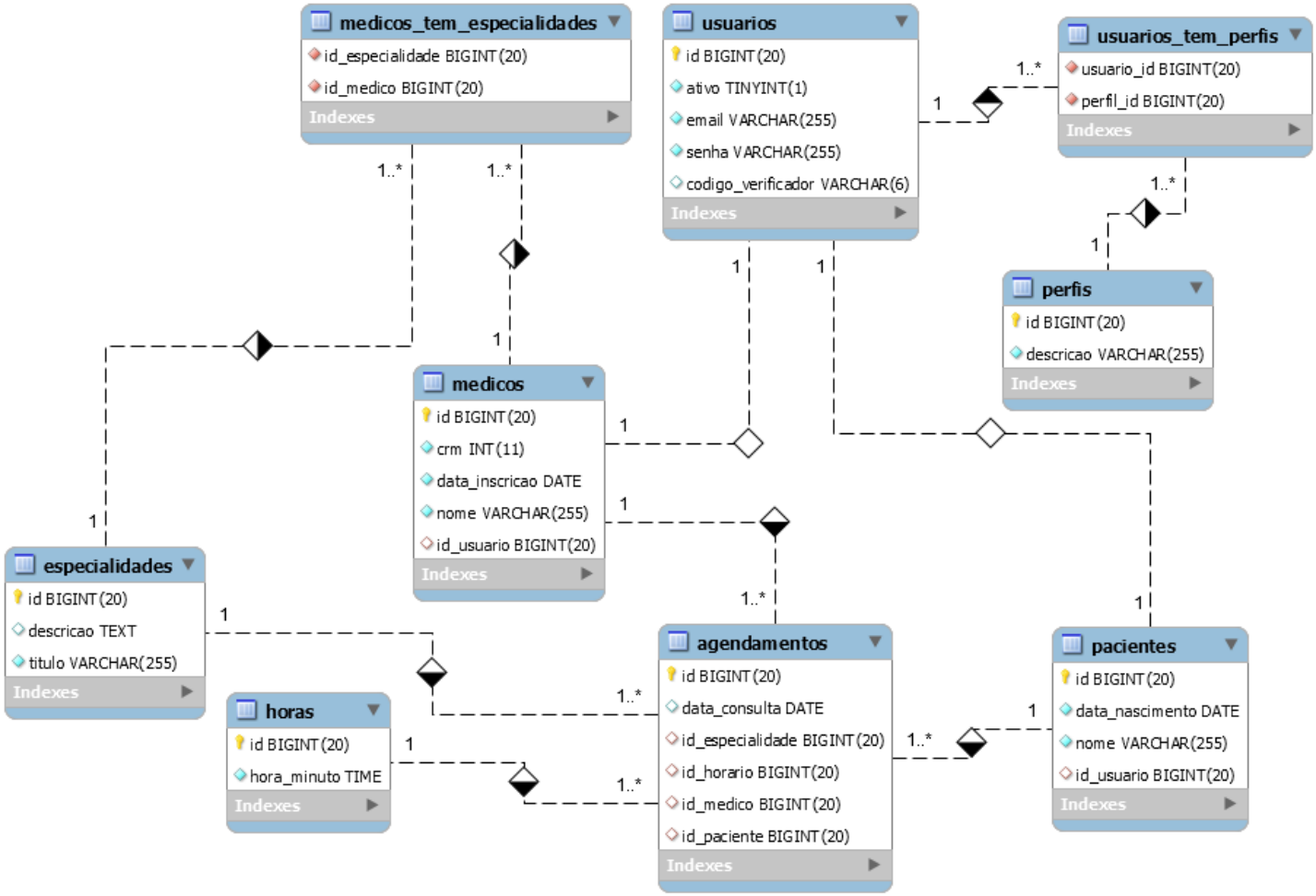

Diagrama Entidade Relacional

Revisando o diagrama podemos verificar que:

- Um usuário pode ter diversos perfis e um perfil poderá pertencer a diversos usuários. Os tipos de perfis serão `ADMIN`, `MEDICO` e `PACIENTE`, podendo ter uma combinação como `ADMIN` e `MEDICO`, tendo assim, um usuário com dois perfis cadastrados. Para isso temos as tabelas `usuarios` e `perfis` que geraram, por conta do relacionamento muitos para muitos a tabela `usuarios_tem_perfis` a qual fica responsável por armazenar os relacionamentos.

- Um usuário do tipo administrador não terá dados pessoais inseridos no banco de dados. As únicas informações armazenadas para o administrador serão aquelas presentes na tabela `usuarios`: `id`, `ativo`, `email`, `senha` e `codigo_verificador`. A coluna `email` será o nome de usuário, ou seja, o login do usuário no sistema de autenticação. Já `ativo` tem como função armazenar um valor que identifica se o cadastro do usuário está ativado (1) ou não (0). A `codigo_verificador` será um código gerado randomicamente para o processo de redefinição de senha.

- Os dados pessoais de usuários serão armazenados nas tabelas `medicos` e `pacientes`, respectivamente para usuários que tenham o perfil de `MEDICO` ou `PACIENTE`. Se um administrador for também um médico, eles terá dados pessoais na tabela `medicos`.

- Os pacientes e médicos terão um usuário no sistema, por conta disso, temos relacionamentos do tipo um para um entre `pacientes` e `usuarios` e `medicos` e `usuarios`. A chave dos relacionamentos (`id_usuario`) estarão presentes nas tabelas de `pacientes` e `medicos`.

- O paciente poderá agendar consultas médicas, para isso, ele vai selecionar a especialidade médica, o médico, a data da consulta e a hora da consulta. Essas informações serão armazenadas na tabela `agendamentos`. O médico poderá visualizar as consultas que foram agendadas para ele acessando também a tabela de agendamentos. Sendo assim, as tabelas `pacientes` e `medicos` possuem relacionamentos um para muitos com a tabela `agendamentos`.

- Outras duas tabelas têm relacionamentos um para muitos com `agendamentos`, são elas `especialidades` e `horas`.

- Um agendamento será único, sendo assim, não poderão existir mais de um agendamento com a mesma `data_consulta`, `id_especialidade`, `id_horario` e `id_medico`. Esta regra não foi criada a nível de banco de dados, mas poderia ser feita com um índice composto do tipo `unique`. Vamos controlar isso a nível de aplicação.

- Por fim, temos a tabela `especialidades` com um relacionamento muitos para muitos com a tabela `medicos`. Esse relacionamento gerou a tabela `medicos_tem_especialidades`.

Arquivo de propriedades

As propriedades de conexão com o banco de dados estão definidas no arquivo *application.properties*. Os valores destas propriedades foram definidos conforme os dados de acesso ao banco de dados do autor do curso. Caso necessário, altere estas propriedades conforme os dados de acesso ao seu banco de dados.

```
#mysql
spring.datasource.url= jdbc:mysql://localhost:3306/demo_security?useTimezone=true&serverTimezone=America/Sao_P
spring.datasource.username= root
spring.datasource.password= root
```

Os parâmetros `useTimezone` e `serverTimezone` são para definir um timezone específico, uma exigência de algumas versões de instalação do MySQL. Assim como, o parâmetro `useSSL` também pode ser uma exigência e define se a conexão é baseada em um url segura ou não.

Já na parte de JPA as propriedades são as seguintes:

```
#JPA
spring.jpa.hibernate.ddl-auto= none
spring.jpa.show-sql= true
spring.jpa.open-in-view= true
```

A `propriedades hibernate.ddl-auto` está setada com o valor `none`, para desabilitar o processo de gerenciamento das tabelas por parte do Hibernate. Isso porque, já temos a base dados definida via script. Se preferir, por algum motivo, deixar que o Hibernate gere as tabelas e colunas, troque `none` por `update`.
Por fim, encerramos com a propriedade de cache do thymeleaf que é definida como `false` em ambiente de desenvolvimento.

```
# THYMELEAF
spring.thymeleaf.cache= false
```

Classes de Entidades

Acredito que as aulas 04 e 05 tiveram uma boa abordagem sobre as classes de entidades e também a classe `AbstractEntity`. Por conta disso, não vou cita-las nesse documento. Caso tenha alguma duvida a respeito, deixe-a no sistema de cometários que será respondida.

Validação Back-End

Caso tenha criado seu projeto com uma versão do Spring Boot igual ou superior a 2.3.0, para que a validação back-end funcione sem problemas é necessário incluir no arquivo pom.xml um novo starter para validação, segue abaixo:

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-validation -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Referencias e downloads

- Lista de timezones: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

- MySQL Community Server (versão atual)
- MySQL Community Server 5.7
- MySQL Community Server 5.6
- MySQL Community Server 5.5
- MySQL Workbench
- Spring Tool Suite 4