Manual Spring Boot JPA - 2024.1

✂--- IntelliJ IDEA + MariaDB ---✂

1 Criando uma Base de Dados SQL

Considerando-se bases de modelo SQL - neste manual utilizaremos MariaDB em específico - o recomendado é se criar bases com as seguintes definições:

- Charset: utf8mb4
- Collation: utf8mb4_unicode_ci

E então, considerando-se uma base de nome "`database_name`", utiliza-se o seguinte comando para criação de tal:

```
MariaDB [(none)]> CREATE DATABASE database_name
                    CHARACTER SET = 'utf8mb4'
                    COLLATE = 'utf8mb4_unicode_ci';
```

Caso ainda não tenha um usuário local (localhost) definido junto à base, crie um - no exemplo o usuário é "`user`" e a senha do mesmo "`password`":

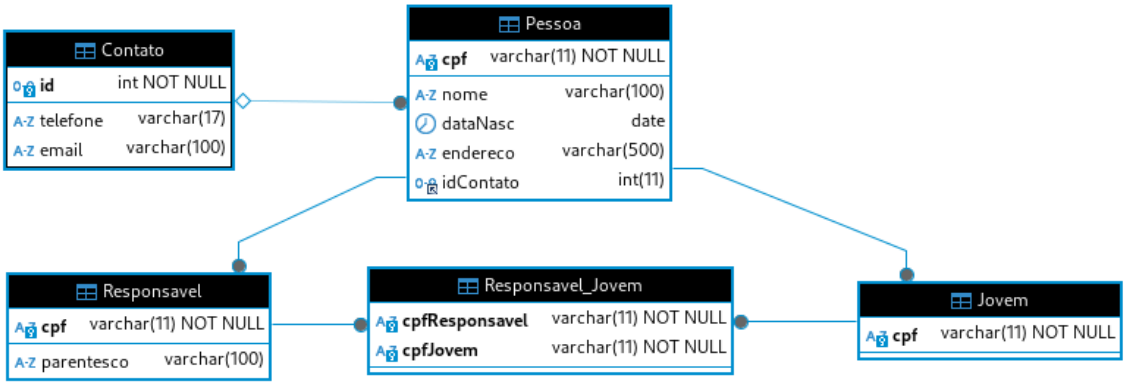
```
MariaDB [(none)]> CREATE USER 'user'@localhost IDENTIFIED BY 'password';
```

E então dê acesso completo ao usuário criado (`user`) à base criada anteriormente (`database_name`):

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON 'database_name'.* TO 'user'@localhost;
MariaDB [(none)]> FLUSH PRIVILEGES;
```

A ferramenta recomendada para modelagem de Bases de Dados é o DBeaver, a qual, na verdade, espelha diretamente uma implementação real existente - note-se que não se trata de modelagem *per se*, afinal se está trabalhando com tabelas sendo realmente criadas em uma base existente.

Para este manual utilizaremos uma base simplifica, em modelo criado com o DBeaver, abaixo:



2 Criando Projeto Spring Boot JPA no IntelliJ IDEA

1. Clique no menu "hamburguer" no topo a esquerda.
2. `File` >> `New` >> `Project...`
3. Em `Generators` selecione `Spring Boot`
4. Mantenha as opções padrão (Language, Type e Packaging) e adicione sua package padrão e nome de projeto (mantenha a *package name*, gerada automaticamente):
   - Language: Jave
   - Type: Gradle - Groovy
   - Group: br.edu.ifsc.db
   - Artifact: NomeDoProjeto
   - Package name: br.edu.ifsc.db.nomedoprojeto
   - Packaging: Jar
5. Selecione os campos JDK e Java de maneira a trabalharem na mesma versão e clique em `Next`.
6. No campo `Spring Boot` selecione a última versão estável da ferramenta e selecione as seguintes dependências:
   - SQL >> Spring Data JPA
   - SQL >> MariaDB Driver
7. Clique em `Create`.

3 Configurando Projeto Spring Boot JPA no IntelliJ IDEA

No arquivo `resources/application.properties` adicione as seguintes propriedades:

```
spring.application.name=NomeDoProjeto
spring.datasource.url=jdbc:mariadb://:3306/database_name
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.username=user
spring.datasource.password=password
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.
PhysicalNamingStrategyStandardImpl
```

Note-se que `NomeDoProjeto` é o nome já previamente definido - tal linha já vai estar no arquivo; `database_name` é o nome que fora definido anteriormente à base de dados; `user` é o nome de usuário criado junto à base anteriormente; e `password` é a senha definida quando da criação do usuário. Adicionalmente, a quebra de linha no último argumento não deve existir; a mesma existe neste documento apenas por questão de visibilidade.

No campo `spring.jpa.hibernate.naming.physical-strategy` o valor definido acima indica que deve-se tratar a base considerando-se letras maiúsculas. Se tal parâmetro não for assim definido será considerado automaticamente que todas as bases possuem os nome em letras minúscula.

Em um caso especial, não o apresentado aqui, é possível se definir somente as classes em Java e definir-se criação automática de tabelas na base adicionando-se:

```
spring.jpa.hibernate.ddl-auto=update
```

4 Criando Classes e Configurando Entidades JPA

4.1 Classes e Mapeamento JPA Simples

Neste caso se enquadra a tabela `Contato`, a qual irá dar origem à classe `Contato`:

```java
package br.edu.ifsc.db.nomedoprojeto;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;

@Entity
public class Contato {
    @Id
    @GeneratedValue
    private int id;
    @Column
    private String telefone;
    @Column
    private String email;

    public Contato() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Contato{" +
                "id=" + id +
                ", telefone='" + telefone + '\'' +
                ", email='" + email + '\'' +
                '}';
    }
}
```

Sendo também necessário se definir sua interface de repositorio `ContatoRepository`:

```
package br.edu.ifsc.db.nomedoprojeto;

import org.springframework.data.repository.CrudRepository;

public interface ContatoRepository  extends CrudRepository<Contato, String> {
}
```

4.2 Classes e Mapeamento JPA com Relacionamento N:1

Neste caso se enquadra a tabela `Pessoa`, a qual irá dar origem à classe `Pessoa`:

```
package br.edu.ifsc.db.nomedoprojeto;

import jakarta.persistence.*;

import java.util.Date;

@Entity
public class Pessoa {

    @Id
    private String cpf;
    @Column(name="nome")
    private String nome;
    @Column(name="dataNasc")
    private Date dataNascimento;
    @Column(name="endereco")
    private String endereco;
    @ManyToOne
    @JoinColumn(name = "idContato", referencedColumnName="id")
    private Contato contato;

    public Pessoa() {
    }

    public Pessoa(String cpf, String nome, Date dataNascimento, String endereco) {
        this.cpf = cpf;
        this.nome = nome;
        this.dataNascimento = dataNascimento;
        this.endereco = endereco;
    }

    public Contato getContato() {
        return contato;
    }

    public void setContato(Contato contato) {
        this.contato = contato;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    public String getCpf() {
        return cpf;
    }

    public String getNome() {
        return nome;
    }
```

```
51
52      public void setNome(String nome) {
53          this.nome = nome;
54      }
55
56      public Date getDataNascimento() {
57          return dataNascimento;
58      }
59
60      public void setDataNascimento(Date dataNascimento) {
61          this.dataNascimento = dataNascimento;
62      }
63
64      public String getEndereco() {
65          return endereco;
66      }
67
68      public void setEndereco(String endereco) {
69          this.endereco = endereco;
70      }
71
72      @Override
73      public String toString() {
74          return "Pessoa{" +
75                  "cpf='" + cpf + '\'' +
76                  ", nome='" + nome + '\'' +
77                  ", dataNascimento=" + dataNascimento +
78                  ", endereco='" + endereco + '\'' +
79                  ", contato=" + contato +
80                  '}';
81      }
82  }
```

Sendo também necessário se definir sua interface de repositorio `PessoaRepository`:

```
1  package br.edu.ifsc.db.nomedoprojeto;
2
3  import org.springframework.data.repository.CrudRepository;
4
5  public interface PessoaRepository extends CrudRepository<Pessoa, String> {
6  }
```

4.3 Classes e Mapemaneto JPA com "Herança" - Relacionamento 1:1

Neste caso se enquadram as tabela `Responsavel` e `Jovem`, as quais irão dar origem às classes `Responsavel` e `Jovem` - para utilizar a definição de `Jovem` abaixo, você irá precisar antes definir `Responsavel_Jovem` -, respectivamente:

```
1  package br.edu.ifsc.db.nomedoprojeto;
2
3  import jakarta.persistence.*;
4
5  @Entity
6  public class Responsavel {
7      @Id
8      private String cpf;
9      @Column
10     private String parentesco;
11     @OneToOne
12     @JoinColumn(name = "cpf", referencedColumnName="cpf")
13     private Pessoa pessoa;
14
```

```
15      public Responsavel() {
16      }
17
18      public Responsavel(String cpf, String parentesco) {
19          this.cpf = cpf;
20          this.parentesco = parentesco;
21      }
22
23      public String getCpf() {
24          return cpf;
25      }
26
27      public void setCpf(String cpf) {
28          this.cpf = cpf;
29          pessoa.setCpf(cpf);
30      }
31
32      public String getParentesco() {
33          return parentesco;
34      }
35
36      public void setParentesco(String parentesco) {
37          this.parentesco = parentesco;
38      }
39
40      public String getNome() {
41          return pessoa.getNome();
42      }
43
44      public void setNome(String nome) {
45          pessoa.setNome(nome);
46      }
47
48      public Date getDataNascimento() {
49          return pessoa.getDataNascimento();
50      }
51
52      public void setDataNascimento(Date dataNascimento) {
53          pessoa.setDataNascimento(dataNascimento);
54      }
55
56      public String getEndereco() {
57          return pessoa.getEndereco();
58      }
59
60      public void setEndereco(String endereco) {
61          pessoa.setEndereco(endereco);
62      }
63
64      public Contato getContato() {
65          return pessoa.getContato();
66      }
67
68      public void setContato(Contato contato) {
69          pessoa.setContato(contato);
70      }
71
72      @Override
73      public String toString() {
74          return "Responsavel{" +
75                  "cpf='" + cpf + '\'' +
76                  ", parentesco='" + parentesco + '\'' +
77                  ", nome='" + pessoa.getNome() + '\'' +
78                  ", dataNascimento=" + pessoa.getDataNascimento() +
79                  ", endereco='" + pessoa.getEndereco() + '\'' +
80                  ", telefone='" + pessoa.getContato().getTelefone() + '\'' +
```

```
                ", email='" + pessoa.getContato().getEmail() + '\'' +
                '}';
    }
}
```

```
package br.edu.ifsc.db.nomedoprojeto;

import jakarta.persistence.*;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Entity
public class Jovem {
    @Id
    private String cpf;
    @OneToOne
    @JoinColumn(name = "cpf", referencedColumnName="cpf")
    private Pessoa pessoa;
    @OneToMany(fetch = FetchType.EAGER)
    @JoinColumn(name = "cpfJovem", referencedColumnName="cpf")
    private List<ResponsavelJovem> jr;

    public Jovem() {
        jr = new ArrayList<ResponsavelJovem>();
    }

    public Jovem(String cpf) {
        this.cpf = cpf;
    }

    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
        pessoa.setCpf(cpf);
    }

    public Pessoa getPessoa() {
        return pessoa;
    }

    public void setPessoa(Pessoa pessoa) {
        this.pessoa = pessoa;
    }

    public List<ResponsavelJovem> getJr() {
        return jr;
    }

    public void setJr(List<ResponsavelJovem> jr) {
        this.jr = jr;
    }

    public String getNome() {
        return pessoa.getNome();
    }

    public void setNome(String nome) {
        pessoa.setNome(nome);
    }

```

```
61      public Date getDataNascimento() {
62          return pessoa.getDataNascimento();
63      }
64
65      public void setDataNascimento(Date dataNascimento) {
66          pessoa.setDataNascimento(dataNascimento);
67      }
68
69      public String getEndereco() {
70          return pessoa.getEndereco();
71      }
72
73      public void setEndereco(String endereco) {
74          pessoa.setEndereco(endereco);
75      }
76
77      public Contato getContato() {
78          return pessoa.getContato();
79      }
80
81      public void setContato(Contato contato) {
82          pessoa.setContato(contato);
83      }
84
85      @Override
86      public String toString() {
87          String resposta = "Jovem{" +
88                  "cpf='" + cpf + '\'' +
89                  ", nome='" + getNome() + '\'' +
90                  ", dataNascimento=" + getDataNascimento() +
91                  ", endereco='" + getEndereco() + '\'' +
92                  ", telefone='" + getContato().getTelefone() + '\'' +
93                  ", email='" + getContato().getEmail() + '\'' +
94                  ", responsaveis ="+"[";
95          for(ResponsavelJovem rj : jr) {
96              resposta += '\'' + rj.getResponsavel().toString() + '\'' + ",";
97          }
98
99          resposta += "]}"+'\n';
100         return resposta;
101     }
102 }
```

Sendo também necessário se definir suas respectivas interfaces de repositorio `ResponsavelRepository` e `JovemRepository`:

```
1  package br.edu.ifsc.db.nomedoprojeto;
2
3  import org.springframework.data.repository.CrudRepository;
4
5  public interface ResponsavelRepository extends CrudRepository<Responsavel, String> {
6  }
```

```
1  package br.edu.ifsc.db.nomedoprojeto;
2
3  import org.springframework.data.repository.CrudRepository;
4
5  public interface JovemRepository extends CrudRepository<Jovem, String> {
6  }
```

4.4 Classes e Mapeamento JPA com Relacionamento N:N

Neste caso se enquadra a tabela `Responsavel_Jovem`, a qual irá dar origem à classe `ResponsavelJovem`:

```java
package br.edu.ifsc.db.nomedoprojeto;

import jakarta.persistence.*;

import java.io.Serializable;

class CompositeKey implements Serializable {
    private String cpfResponsavel;
    private String cpfJovem;
}

@Entity
@IdClass(CompositeKey.class)
@Table(name = "Responsavel_Jovem")
public class ResponsavelJovem implements Serializable {
    @Id
    private String cpfResponsavel;
    @Id
    private String cpfJovem;
    @OneToOne
    @JoinColumn(name = "cpfResponsavel", referencedColumnName="cpf")
    private Responsavel responsavel;

    public ResponsavelJovem() {
    }

    public String getCpfResponsavel() {
        return cpfResponsavel;
    }

    public void setCpfResponsavel(String cpfResponsavel) {
        this.cpfResponsavel = cpfResponsavel;
    }

    public String getCpfJovem() {
        return cpfJovem;
    }

    public void setCpfJovem(String cpfJovem) {
        this.cpfJovem = cpfJovem;
    }

    public Responsavel getResponsavel() {
        return responsavel;
    }

    public void setResponsaveis(Responsavel responsavel) {
        this.responsavel = responsavel;
    }
    @Override
    public String toString() {
        return "Jovem_Responsavel{" +
                "cpfResponsavel='" + cpfResponsavel + '\'' +
                ", cpfJovem='" + cpfJovem + '\'' +
                '}';
    }
}
```

Sendo também necessário se definir sua interface de repositorio `ResponsavelJovemRepository`:

```java
package br.edu.ifsc.db.nomedoprojeto;

import org.springframework.data.repository.CrudRepository;

public interface ResponsavelJovemRepository extends CrudRepository<ResponsavelJovem,
    String> {
}
```

5 Testando as Classes e Acesso à Base

Na classe NomeDoProjetoApplication - esta classe, na verdade, começará com o nome dado ao projeto - faça os testes para cada classe ou, neste caso, apenas para classe `Jovem`, já que a mesma se utiliza de todas as outras:

```java
package br.edu.ifsc.db.nomedoprojeto;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class NomeDoProjetoApplication {

    public static void main(String[] args) {
        SpringApplication.run(NomeDoProjetoApplication.class, args);
    }

    @Bean
    public CommandLineRunner run(JovemRepository repository) {
        return (args -> {
            System.out.println(repository.findAll());
        });
    }

}
```