



Persistência em Java – Avaliação da Disciplina

Turma 32SCJ

**Luiz Fillipe Ribeiro Navarro
Bruno Lopes
Wagner Bina dos Santos
Vinicius Gasparin**

**RM332867
RM332786
RM332061
RM33**



Sumário

1. Introdução	2
2. Objetivo	2
3. Modelagem do Banco de Dados.....	2
4. Gerando uma base no MySQL utilizando o modelo criado	3
5. Componentes, bibliotecas, frameworks utilizados para o desenvolvimento da aplicação10	
5.1 Componentes.....	21
5.2 Bibliotecas.....	10
5.3 Frameworks	10
6. Configuração da Persistência.....	11
7. Configuração da Persistência Spring JDBC.....	14
8. Desenvolvimento da Aplicação.....	18
9. Referências.....	29



1. Introdução

Trata-se de um sistema de cadastro e consulta de situações de alunos em uma escola. Considera-se os seguintes requisitos:

- Uma escola possui vários cursos de capacitação
- Cada curso possui vários alunos
- Cada aluno pode estar matriculado em mais de um curso
- Para cada curso, o aluno recebe uma nota, de 0 a 10

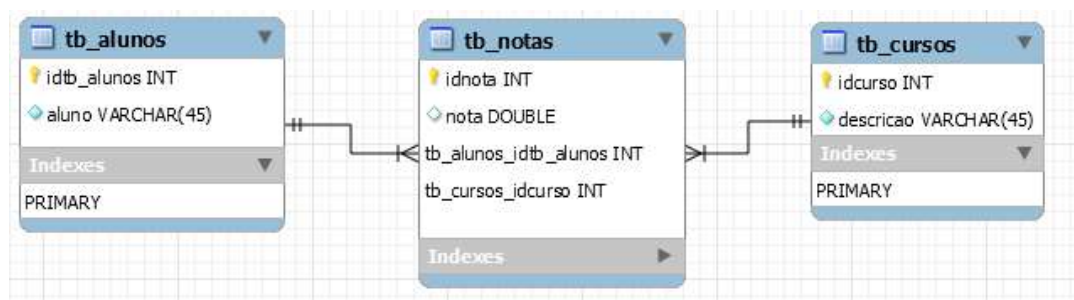
2. Objetivo

Este documento tem como objetivo descrever os requisitos para a elaboração de um projeto usado como avaliação da disciplina.

3. Modelagem do Banco de Dados

3.1 Utilizamos três tabelas para atender a premissa do projeto onde a entidade aluno pode ter mais do que uma nota se ele estiver matriculado em mais do que um curso e a entidade curso tem mais do que uma nota para mais de um aluno.

Figura 1 – MySQL Model



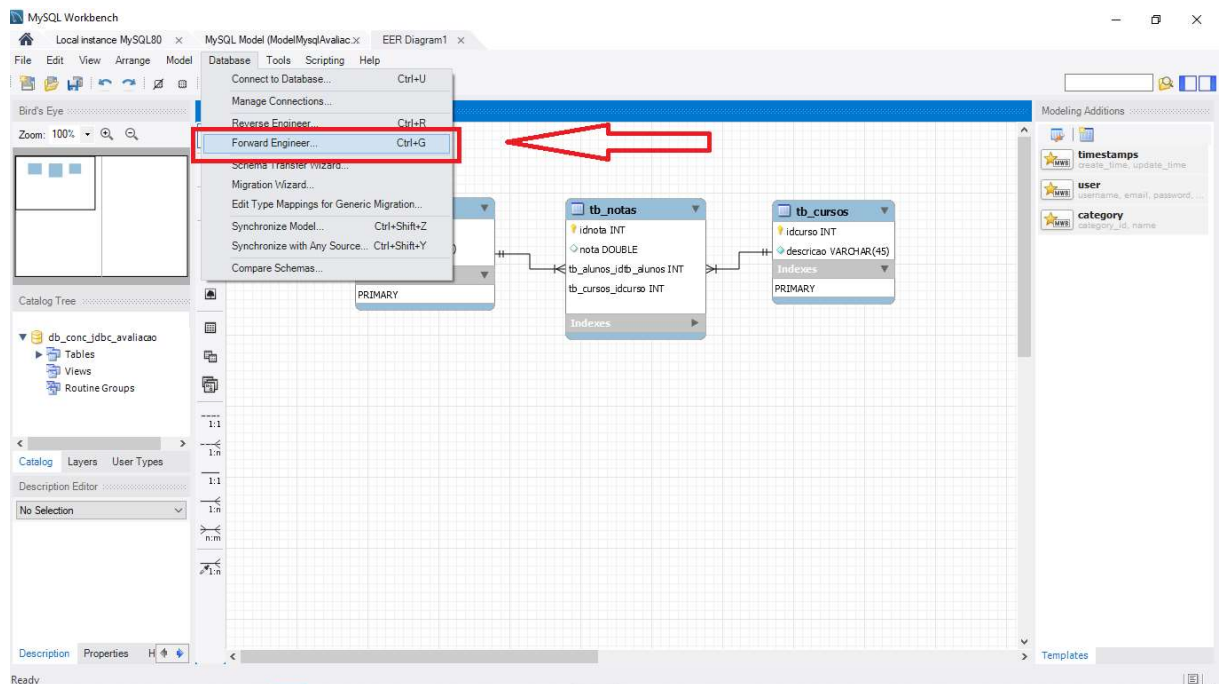


Fonte Própria(2019, p 3)

4. Gerando uma base no MySQL utilizando o modelo criado

Selecionar no menu Database a opção Forward Engineer

Figura 2 – Configuração base MySQL



Fonte Própria(2019, p 4)

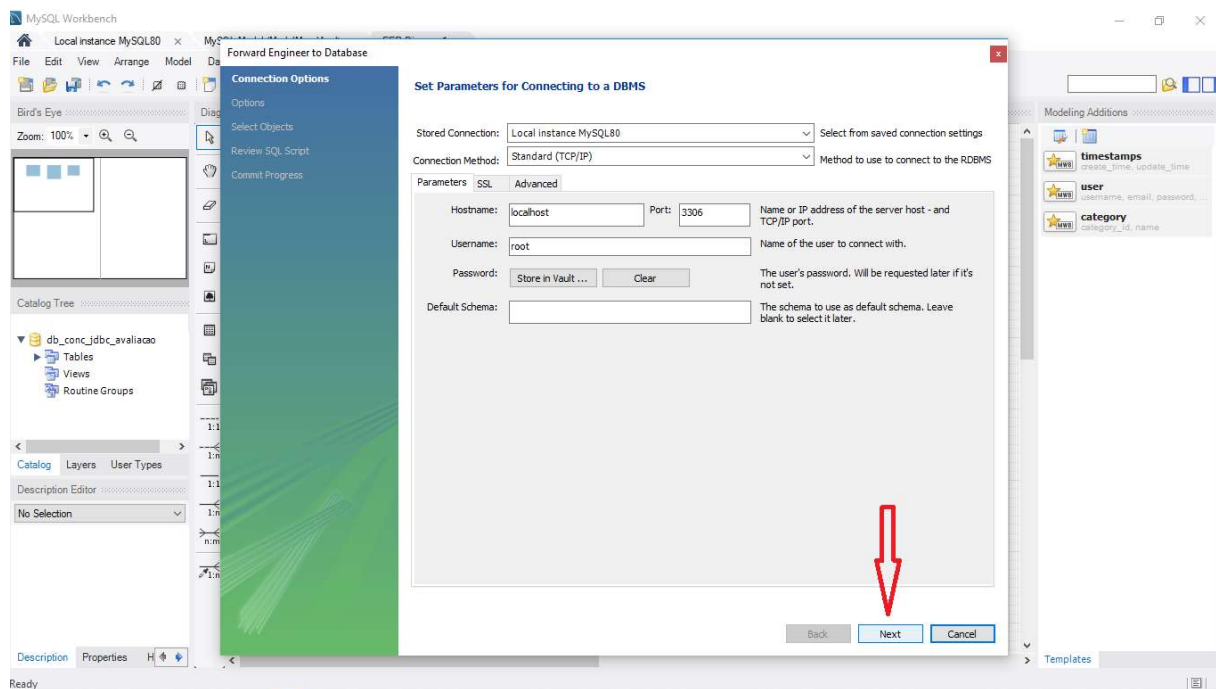


PERSISTÊNCIA EM JAVA AVALIAÇÃO DA DISCIPLINA



No campo **Hostname** podemos definir o endereço de rede da nossa base e no Campo **Port** a porta de comunicação que iremos utilizar. No caso vamos manter as configurações de fábrica e vamos clicar no botão **Next**.

Figura 3 - Configuração Base



Fonte Própria(2019, p 5)



Nesta tela podemos alterar as configurações do Modelo que foi gerado. Em nossa aplicação vamos manter a configurações que modelamos em nosso projeto selecionando o botão **Next**.

Figura 4 - Configuração base MySQL

The screenshot shows a software interface titled "Forward Engineer to Database". On the left is a sidebar with a blue header and a green gradient background, containing the following menu items: "Connection Options", "Options" (highlighted), "Select Objects", "Review SQL Script", and "Commit Progress". The main area is titled "Set Options for Database to be Created" and contains three sections of checkboxes:

- Tables**
 - ☐ Skip creation of FOREIGN KEYS
 - ☐ Skip creation of FK Indexes as well
 - ☐ Generate separate CREATE INDEX statements
 - ☐ Generate INSERT statements for tables
 - ☐ Disable FK checks for INSERTs
- Other Objects**
 - ☐ Don't create view placeholder tables
 - ☐ Do not create users. Only create privileges (GRANTS)
- Code Generation**
 - ☐ DROP objects before each CREATE object
 - ☐ Generate DROP SCHEMA
 - ☐ Omit schema qualifier in object names
 - ☐ Generate USE statements
 - ☐ Add SHOW WARNINGS after every DDL statement
 - ☒ Include model attached scripts

At the bottom right, there are three buttons: "Back", "Next", and "Cancel". A large red arrow points directly to the "Next" button.

Fonte Própria(2019, p 6)

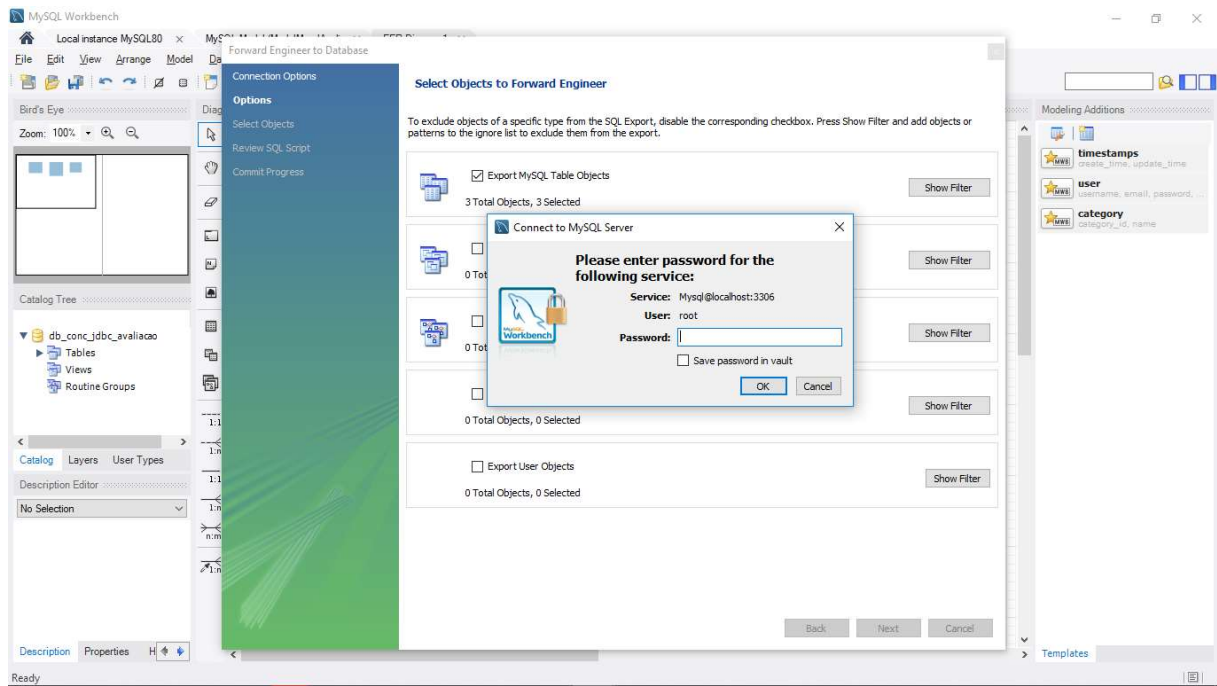


PERSISTÊNCIA EM JAVA
AVALIAÇÃO DA DISCIPLINA



Digite a senha do usuário Admin do MySQL e clique no botão **OK**.

Figura 5 - Configuração base MySQL



Fonte Própria(2019, p 7)



Vamos manter as tabelas que criamos no Modelo e selecionar o botão **Next**.

Figura 6 - Configuração base MySQL

Forward Engineer to Database

Connection Options
Options
Select Objects
Review SQL Script
Commit Progress

Select Objects to Forward Engineer

To exclude objects of a specific type from the SQL Export, disable the corresponding checkbox. Press Show Filter and add objects or patterns to the ignore list to exclude them from the export.

	<input checked="" type="checkbox"/> Export MySQL Table Objects 3 Total Objects, 3 Selected	Show Filter
	<input type="checkbox"/> Export MySQL View Objects 0 Total Objects, 0 Selected	Show Filter
	<input type="checkbox"/> Export MySQL Routine Objects 0 Total Objects, 0 Selected	Show Filter
	<input type="checkbox"/> Export MySQL Trigger Objects 0 Total Objects, 0 Selected	Show Filter
	<input type="checkbox"/> Export User Objects 0 Total Objects, 0 Selected	Show Filter

Back Next Cancel

Fonte Própria(2019, p 8)

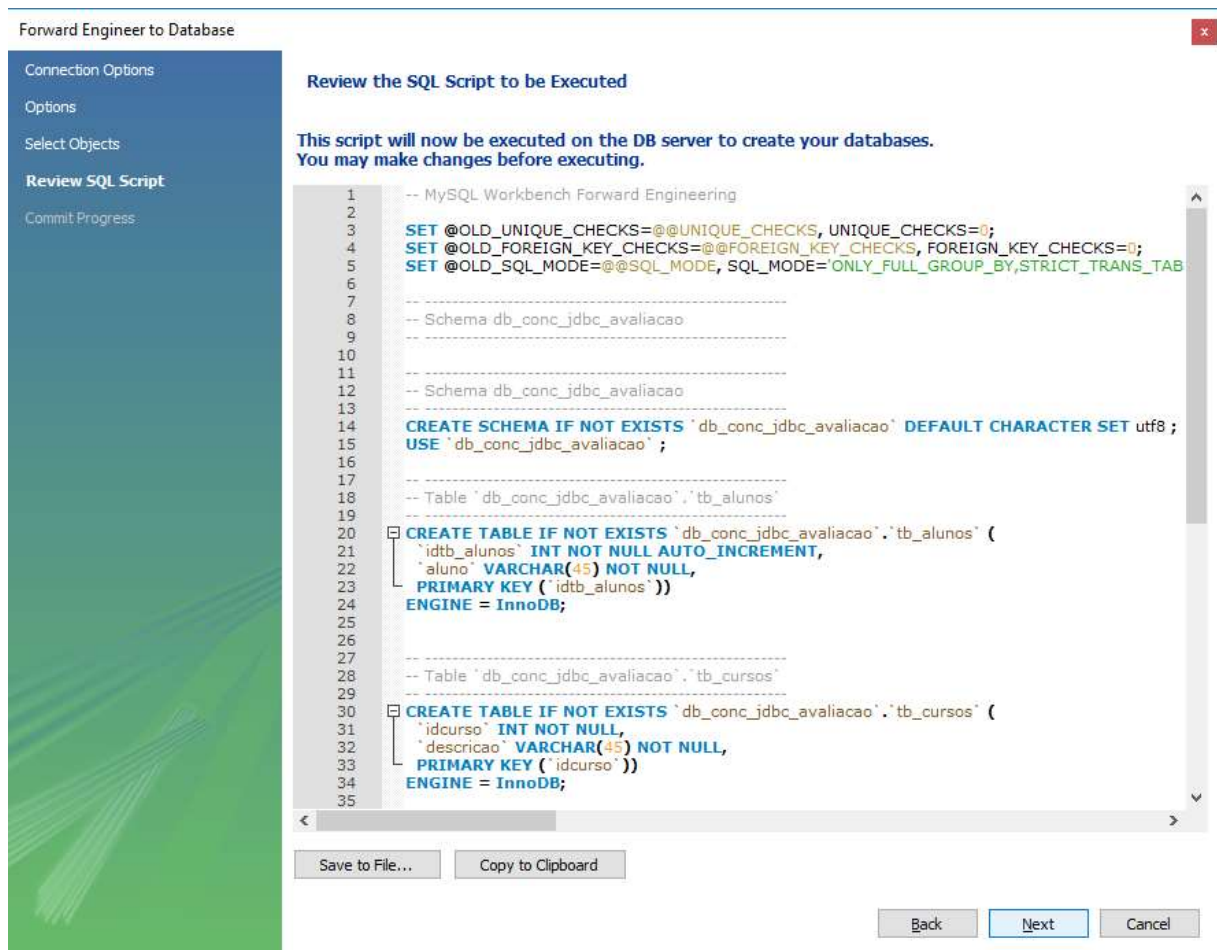


PERSISTÊNCIA EM JAVA
AVALIAÇÃO DA DISCIPLINA



Nesta tela temos a opção de salvar o script de criação da base. Vamos seguir com a criação da base clicando no botão **Next**.

Figura 7 - Configuração base MySQL

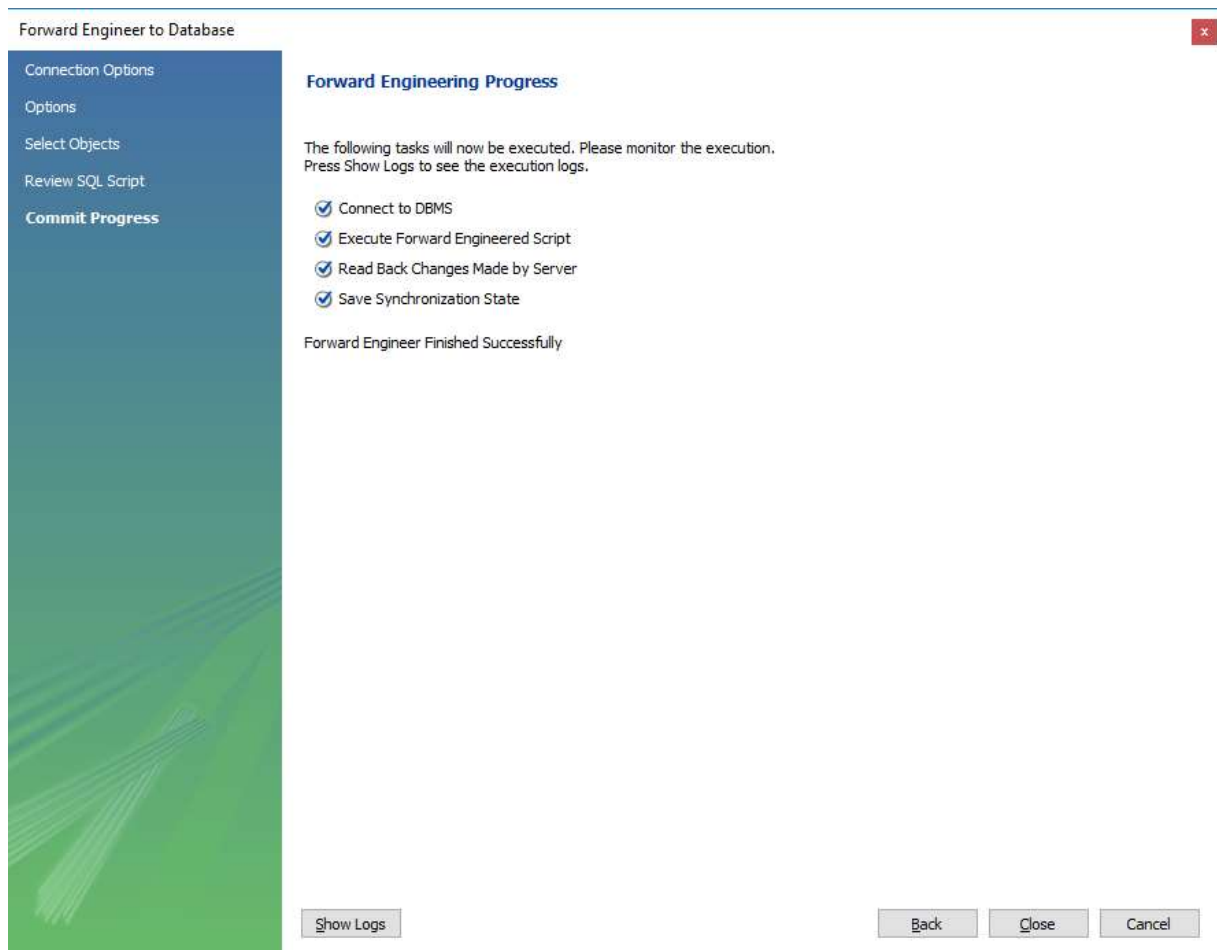


Fonte Própria(2019, p 9)



Nesta tela o MySQL conclui o processo de criação da base. Após o término do processo vamos clicar no botão **Close**.

Figura 8 - Configuração base MySQL



Fonte Própria(2019, p 10)



5. Componentes, bibliotecas, frameworks utilizados para o desenvolvimento da aplicação

5.1 Componentes

Java Util ArrayList, Java Util List, Java SQL SQLException, Java SQL ResultSet, Javax SQL DataSource, Spring Framework JDBC Core RowMapper e Spring Framework JDBC Core JdbcTemplate

5.2 Bibliotecas

Java Util, Java Sql, Java Swing, Spring JDBC Core, Spring Context, Spring JDBC Core e Spring Context.

5.3 Frameworks

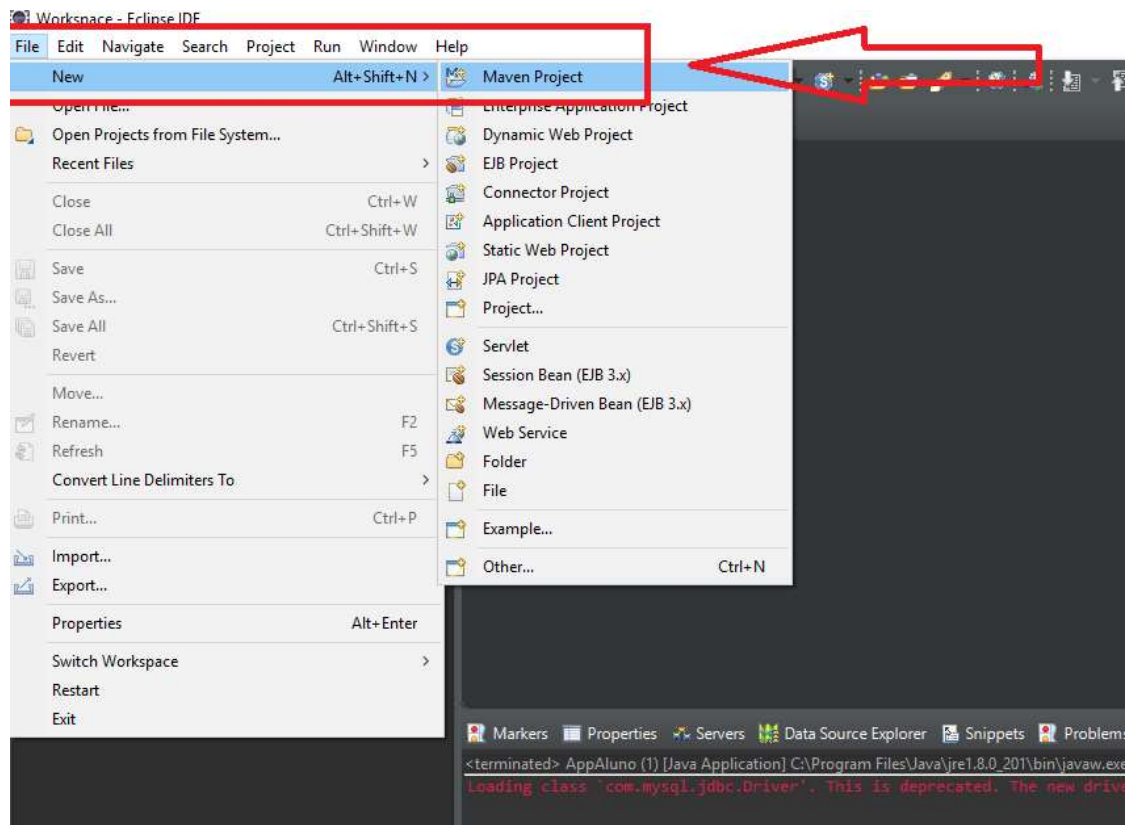
Spring JdbcTemplate e Maven.



6. Configuração da Persistência

Clicar no menu **File** e, selecionar a opção **New** e clicar na opção **Maven Project**

Figura 9 – Configuração da Persistência

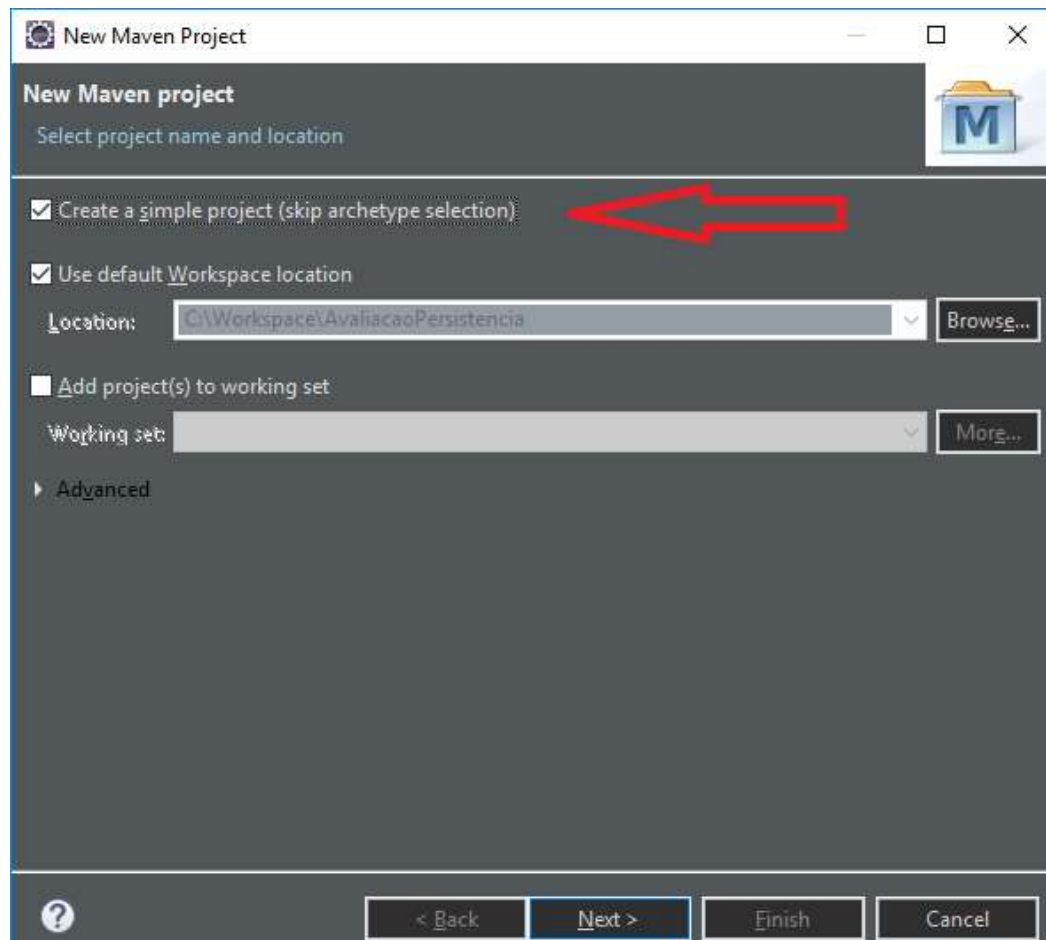


Fonte Própria(2019, p. 11)



Nesta tela vamos clicar no Campo **Create a simple Project (skip archetype selection)** para criar um projeto simples e em seguida clicar no botão **Next**.

Figura 10 - Print de Tela 2



Fonte Própria(2019, p. 12)



Devemos preencher os Campos **Group Id**, **Artifact Id** e **Name**. Vamos preencher com o mesmo nome do projeto e clicar no botão **Finish**.

Figura 11 – Configuração da Persistência

New Maven Project

New Maven project

Project "AvaliacaoPersistencia" already exists.

Artifact

Group Id: AvaliacaoPersistencia

Artifact Id: AvaliacaoPersistencia

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name: AvaliacaoPersistencia

Description:

Parent Project

Group Id:

Artifact Id:

Version:

Browse... Clear

Advanced

< Back Next > Finish Cancel

Fonte Própria(2019, p. 13)



7. Configuração da Persistencia Spring JDBC

Vamos abrir o arquivo **pom.xml** e edita-lo incluindo as dependências necessárias para o nosso projeto. No caso estamos utilizando o MySQL versão 8.0.13 para o nosso Banco de Dados. Uma vez configurado o **pom.xml**, ao salvar o arquivo o **Maiven** inicia o download de todas dependências.

Figura 11 – Configuração da Persistência

```

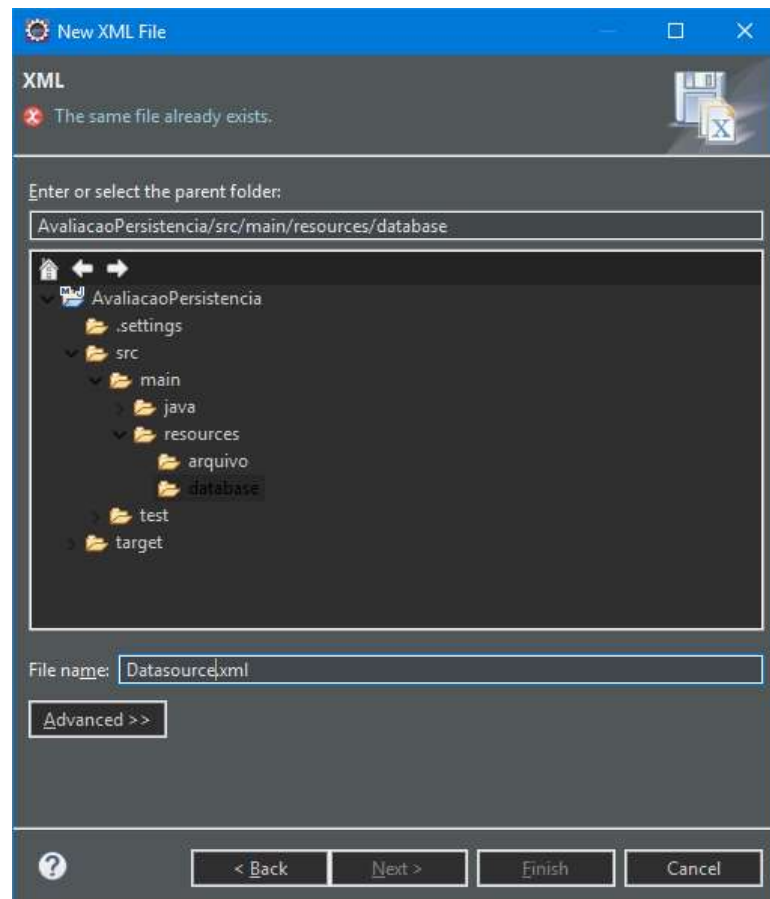
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <groupId>AvaliacaoPersistencia</groupId>
4   <artifactId>AvaliacaoPersistencia</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6   <name>AvaliacaoPersistencia</name>
7
8   <dependencies>
9
10    <dependency>
11      <groupId>org.springframework</groupId>
12      <artifactId>spring-core</artifactId>
13      <version>5.1.3.RELEASE</version>
14    </dependency>
15
16    <dependency>
17      <groupId>org.springframework</groupId>
18      <artifactId>spring-context</artifactId>
19      <version>5.1.3.RELEASE</version>
20    </dependency>
21
22    <dependency>
23      <groupId>org.springframework</groupId>
24      <artifactId>spring-jdbc</artifactId>
25      <version>5.1.3.RELEASE</version>
26    </dependency>
27
28    <dependency>
29      <groupId>mysql</groupId>
30      <artifactId>mysql-connector-java</artifactId>
31      <version>8.0.13</version>
32    </dependency>
33  </dependencies>
34
35 </project>
  
```

Fonte Própria(2019, p. 14)



Criar em AvaliacaoPersistencia/src/main/resources a pasta **database** e gerar o arquivo **Datasource.xml**

Figura 12 – Configuração da Persistência



Fonte Própria(2019, p. 15)



PERSISTÊNCIA EM JAVA AVALIAÇÃO DA DISCIPLINA



Vamos editar o arquivo xml incluindo a tag **<beans>** com o cabeçalho apontando para a página do Spring

Figura 13 – Configuração da Persistência

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="
5         http://www.springframework.org/schema/beans
6         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7
8     <bean id="dataSource" Nome que será invocado o nosso arquivo de persistência com o BD
9         class="org.springframework.jdbc.datasource.DriverManagerDataSource">
10
11         <property name="driverClassName" value="com.mysql.jdbc.Driver" /> Driver de comunicação com BD MySQL
12         <property name="url" value="jdbc:mysql://127.0.0.1:3306/db_conc_jdbc_avalicao" /> Endereço de rede e o nome da base
13         <property name="username" value="root" />
14         <property name="password" value="Mysql" /> Usuário e senha para acesso a base
15     </bean>
16
17 </beans>
  
```

Fonte Própria(2019, p. 16)

Vamos criar uma nova pasta em **AvaliacaoPersistencia/src/main/resources** que chamaremos de arquivo e dentro um novo arquivo que chamaremos também de **Arquivo.xml**. A função deste arquivo é informar para a nossa aplicação o mapeamento de onde está o arquivo que criaremos mais a diante com o nome **JdbcAlunoDao.java** onde ficará todos os métodos de persistência com o Banco de Dados e vamos utilizar a tag **<property>** para informar o nome do nosso arquivo de comunicação de persistência(**"dataSource"**).

Figura 14 – Configuração da Persistência

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
6
7     <bean id="JdbcAlunoDao"
8         class="br.com.fiap.avalicao.dao.implement.JdbcAlunoDao">
9         <property name="dataSource" ref="dataSource" />
10     </bean>
11 </beans>
  
```

Fonte Própria(2019, p. 16)



Agora vamos criar na raiz de **AvaliacaoPersistencia/src/main/resources** o arquivo **Module.xml** que fará a função de importar os dois arquivos de configuração de persistência que acabamos de criar para a aplicação, quando a aplicação for executada utilizando o componente **ApplicationContext**(ex: `ApplicationContext contexto = new ClassPathXmlApplictionContext("Module.xml");`);

Figura 14 – Configuração da Persistência

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation=
5         "http://www.springframework.org/schema/beans
6          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7
8   <import resource="database/Datasource.xml" />
9   <import resource="arquivo/Arquivo.xml" />
10
11 </beans>
```

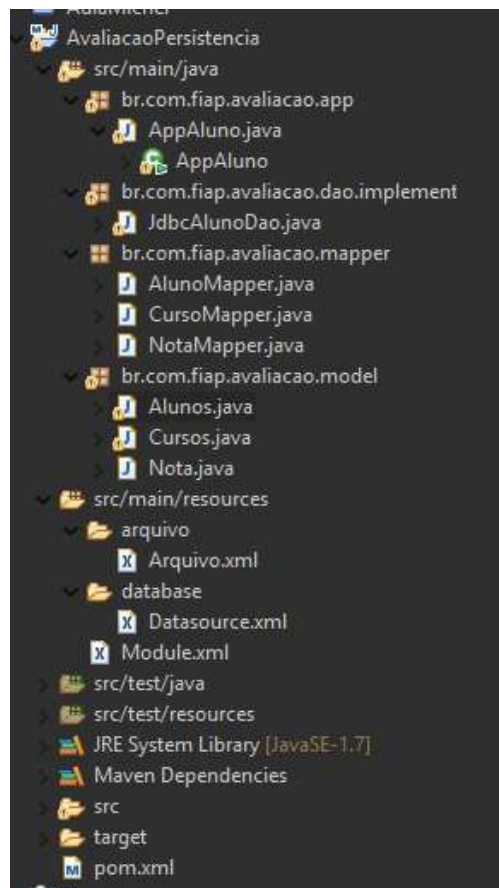
Fonte Própria(2019, p. 17)



8. Desenvolvimento da Aplicação

Para o nosso projeto utilizamos a seguinte estrutura de pastas;

Figura 15 – Aplicação



Fonte Própria(2019, p. 18)



Vamos criar a classe **Alunos** com os seguintes atributos;

Figura 16 – Aplicação

```
JdbcAlunoDao.java  AlunoMapper.java  CursoMapper.java
1  package br.com.fiap.avaliacao.model;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6
7  public class Alunos {
8      private Integer idAluno;
9      private static String aluno;
10
11      private List<Alunos> alunos = new ArrayList<>();
12
13
14      public Alunos(Integer idAluno, String aluno) {
15          this.idAluno = idAluno;
16          this.aluno = aluno;
17      }
18
19
20      public Alunos() {
21      }
22
23
24      public Integer getIdAluno() {
25          return idAluno;
26      }
27      public void setIdAluno(Integer idAluno) {
28          this.idAluno = idAluno;
29      }
30      public static String getAluno() {
31          return aluno;
32      }
33      public void setAluno(String aluno) {
34          Alunos.aluno = aluno;
35      }
36
37      public List<Alunos> getAlunos() {
38          return alunos;
39      }
40
41      public void setAlunos(List<Alunos> alunos) {
42          this.alunos = alunos;
43      }
44
45  }
```

Fonte Própria(2019, p. 19)



Vamos criar a classe **Cursos**;

Figura 17 – Aplicação

```
package br.com.fiap.avaliacao.model;

import java.util.ArrayList;
import java.util.List;

public class Cursos {

    private Integer idCurso;
    private static String descricao;

    private List<Cursos> cursos = new ArrayList<>();

    public Cursos(Integer idCurso, String descricao) {
        this.idCurso = idCurso;
        this.descricao = descricao;
    }

    public Cursos() {
    }

    public Integer getIdCurso() {
        return idCurso;
    }

    public void setIdCurso(Integer idCurso) {
        this.idCurso = idCurso;
    }

    public static String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        Cursos.descricao = descricao;
    }

    public List<Cursos> getCursos() {
        return cursos;
    }

    public void setCursos(List<Cursos> cursos) {
        this.cursos = cursos;
    }
}
```

Fonte Própria(2019, p. 20)



Vamos criar a classe **Nota**;

Figura 18 – Aplicação

```
1 package br.com.fiap.avaliacao.model;
2
3 public class Nota {
4     private Integer idNota;
5     private Double nota;
6     private Integer idAluno;
7     private Integer idCurso;
8
9
10    public Integer getIdNota() {
11        return idNota;
12    }
13    public void setIdNota(Integer idNota) {
14        this.idNota = idNota;
15    }
16    public Double getNota() {
17        return nota;
18    }
19    public void setNota(Double nota) {
20        this.nota = nota;
21    }
22    public Integer getIdAluno() {
23        return idAluno;
24    }
25    public void setIdAluno(Integer idAluno) {
26        this.idAluno = idAluno;
27    }
28    public Integer getIdCurso() {
29        return idCurso;
30    }
31    public void setIdCurso(Integer idCurso) {
32        this.idCurso = idCurso;
33    }
34 }
35
```

Fonte Própria(2019, p. 21)



Agora vamos construir o arquivo **JdbcAlunoDao.java** que será o responsável por gerar métodos de acesso ao banco de dados o famoso **CRUD**(Create, Read, Update e Delete);

Figura 19 – Aplicação

```

1 package br.com.fiap.avaliacao.dao.implement;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.sql.DataSource;
7
8 import org.springframework.jdbc.core.JdbcTemplate;
9
10 import br.com.fiap.avaliacao.mapper.AlunoMapper;
11 import br.com.fiap.avaliacao.mapperCursoMapper;
12 import br.com.fiap.avaliacao.mapper.NotaMapper;
13 import br.com.fiap.avaliacao.model.Alunos;
14 import br.com.fiap.avaliacao.model.Cursos;
15 import br.com.fiap.avaliacao.model.Nota;
16
17 public class JdbcAlunoDao {
18     private JdbcTemplate jdbcTemplate;
19
20     public void setDataSource(DataSource dataSource) {
21         this.jdbcTemplate = new JdbcTemplate(dataSource);
22     }
23
24     public void incluirAluno(Alunos aluno) throws Exception {
25         try {
26             String sql = "INSERT INTO TB_ALUNOS" + "(ALUNO) VALUES (?)";
27
28             this.jdbcTemplate.update(sql,
29                 aluno.getAluno());
30
31         } catch (Exception e) {
32             throw e;
33         }
34     }
35
36     public Alunos buscarAluno(int idAluno) throws Exception {
37         Alunos aluno = null;
38         try {
39             String query = "SELECT * TB_ALUNOS WHERE IDTB_ALUNOS=?";
40             aluno = this.jdbcTemplate.queryForObject(query,
41                 new Integer[] {idAluno}, new AlunoMapper());
42
43         } catch (Exception e) {
44             throw e;
45         }
46
47         return aluno;
48     }
49 }

```

Fonte Própria(2019, p. 22)



Note no código que criamos um objeto do tipo **String** e incluímos uma sintaxe SQL para realizar a função desejada;

Figura 20 – Aplicação

```

49 public List<Alunos> listarAlunos() throws Exception {
50     List<Alunos> alunos = new ArrayList<>();
51     try {
52         alunos = this.jdbcTemplate.query(
53             "SELECT * FROM TB_ALUNOS",
54             new AlunoMapper());
55     } catch (Exception e) {
56         throw e;
57     }
58     return alunos;
59 }
60
61 public void incluirCurso(Cursos curso) throws Exception {
62     try {
63         String sql = "INSERT INTO TB_CURSOS" + "(DESCRICAO) VALUES (?)";
64
65         this.jdbcTemplate.update(sql,
66             curso.getDescricao());
67     } catch (Exception e) {
68         throw e;
69     }
70 }
71
72
73 public Cursos buscarCurso(int idCurso) throws Exception {
74     Cursos curso = null;
75     try {
76         String query = "SELECT * TB_CURSOS WHERE IDCURSO = ?";
77         curso = this.jdbcTemplate.queryForObject(query,
78             new Integer[] {idCurso}, new CursoMapper());
79     } catch (Exception e) {
80         throw e;
81     }
82     return curso;
83 }
84
85
86 public List<Cursos> listarCursos() throws Exception {
87     List<Cursos> cursos = new ArrayList<>();
88     try {
89         cursos = this.jdbcTemplate.query(
90             "SELECT * FROM TB_CURSOS",
91             new CursoMapper());
92     } catch (Exception e) {
93         throw e;
94     }
95     return cursos;
96 }
97

```

Linguagem SQL = Linha de comando para selecionar todos os cursos cadastrados na tabela tb_cursos.

Fonte Própria(2019, p. 23)



Para tratar **insert**, **update**, **select**, e outras funções de retorno ou input no banco trabalhamos com exclamação. E para gravar ou guardar os valores de retorno do banco em nossas classes de entidades utilizamos a API do **Spring JdbcTemplate**;

Figura 21 – Aplicação

```
public void incluirNotaAluno(Note nota) throws Exception {
    try {
        String sql = "INSERT INTO TB_NOTAS" + "( TB_ALUNOS_IDTB_ALUNOS, TB_CURSOS_IDCURSO, NOTA) VALUES (?, ?, ?)";

        this.jdbcTemplate.update(sql,
            nota.getIdAluno(),
            nota.getIdCurso(),
            nota.getNota());
    } catch (Exception e) {
        throw e;
    }
}

public Note buscarNotaAluno(int idAluno) throws Exception {
    Note nota = null;
    try {
        String query = "SELECT * TB_NOTAS WHERE TB_ALUNOS_IDTB_ALUNOS = ?";
        nota = this.jdbcTemplate.queryForObject(query,
            new Integer[] {idAluno}, new NoteMapper());
    } catch (Exception e) {
        throw e;
    }
    return nota;
}
```

Fonte Própria(2019, p. 24)



Criamos uma classe para a nossa aplicação que fará o papel de mapear todos os atributos das nossas entidades. Em nosso projeto utilizamos o componente **RowMapper** que cria um **Array** da entidade Alunos com todos os seus atributos e utilizamos o componente **ResultSet** para guardar o resultado de nossa pesquisa em uma estrutura de dados que pode ser percorrida, de forma que possamos ler os dados do banco. Criamos uma classe Mapper para cada entidade;

Figura 22 – Aplicação

```
AppAluno.java  AlunoMapper.java X
1 package br.com.fiap.avaliacao.mapper;
2
3 import java.sql.ResultSet;
4
5
6
7
8 public class AlunoMapper implements RowMapper<Alunos>{
9
10
11
12 @Override
13 public Alunos mapRow(ResultSet rs, int arg1) throws SQLException {
14     Alunos aluno = new Alunos();
15
16     aluno.setIdAluno(rs.getInt("idtb_alunos"));
17     aluno.setAluno(rs.getString("aluno"));
18
19     return aluno;
20 }
21
22 }
23 }
```

Fonte Própria(2019, p. 25)



Agora em nossa aplicação utilizamos o componente **Context** do Spring que aponta para o arquivo **Module.xml** onde estão mapeados os arquivos de configuração e o **Bean** que configuramos os dados de acesso ao banco que estamos utilizando para o nosso projeto.

Figura 23 – Aplicação

```

AppAluno.java
1 package br.com.fiap.avaliacao.app;
2
3 import java.util.Scanner;
4
5 import javax.swing.JOptionPane;
6
7 import org.springframework.context.ApplicationContext;
8 import org.springframework.context.support.ClassPathXmlApplicationContext;
9
10 import br.com.fiap.avaliacao.dao.implement.JdbcAlunoDao;
11 import br.com.fiap.avaliacao.model.Alunos;
12 import br.com.fiap.avaliacao.model.Cursos;
13 import br.com.fiap.avaliacao.model.Nota;
14
15 public class AppAluno {
16     public static void main(String[] args) {
17         // incluirAluno();
18         // buscarAluno();
19         // listarAlunos();
20
21         // incluirCurso();
22
23         incluirNotaAluno();
24     }
25
26     public static void incluirAluno() {
27         try {
28             ApplicationContext context = new
29                 ClassPathXmlApplicationContext("Module.xml");
30             JdbcAlunoDao dao = (JdbcAlunoDao) context.getBean("JdbcAlunoDao");
31
32             Scanner scan = new Scanner(System.in);
33
34             Alunos aluno = new Alunos();
35             System.out.print("Digite o nome do Aluno: ");
36             aluno.setAluno(scan.nextLine());
37             System.out.println("Aluno " + aluno.getAluno() + " incluído com sucesso!");
38             scan.close();
39             dao.incluirAluno(aluno);
40             JOptionPane.showMessageDialog(null, "Aluno Incluído com Sucesso");
41
42         } catch (Exception e) {
43             e.printStackTrace();
44         }
45     }
46 }

```

Fonte Própria(2019, p. 26)



Figura 24 – Aplicação

```
AppAluno.java x
46
47 public static void incluirCurso() {
48     try {
49         ApplicationContext context = new
50             ClassPathXmlApplicationContext("Module.xml");
51         JdbcAlunoDao dao = (JdbcAlunoDao) context.getBean("JdbcAlunoDao");
52
53         Scanner scan = new Scanner(System.in);
54
55         Cursos curso = new Cursos();
56         System.out.print("Digite o nome do Curso: ");
57         curso.setDescricao(scan.nextLine());
58         System.out.println("Curso " + curso.getDescricao() + " incluido com sucesso!");
59         scan.close();
60         dao.incluirCurso(curso);
61         JOptionPane.showMessageDialog(null, "Curso Incluido com Sucesso");
62     } catch (Exception e) {
63         e.printStackTrace();
64     }
65 }
66
67
```

Fonte Própria(2019, p. 27)

Figura 25 – Aplicação

```

public static void incluirNotaAluno() {
    try {
        ApplicationContext context = new
            ClassPathXmlApplicationContext("Module.xml");
        JdbcAlunoDao dao = (JdbcAlunoDao) context.getBean("JdbcAlunoDao");

        Nota nota = new Nota();

        String digiteIdAluno = JOptionPane.showInputDialog("Digite o ID do aluno: ");
        int converterIdAluno = Integer.parseInt(digiteIdAluno);
        nota.setIdAluno(converterIdAluno);

        String digiteIdCurso = JOptionPane.showInputDialog("Digite o ID do Curso: ");
        int converterIdCurso = Integer.parseInt(digiteIdCurso);
        nota.setIdCurso(converterIdCurso);

        String digiteNota = JOptionPane.showInputDialog("Digite a nota do Aluno: ");
        double converterNota = Double.parseDouble(digiteNota);
        nota.setNota(converterNota);

        System.out.println("ID Aluno - " + nota.getIdAluno() + " do ID curso - " + nota.getIdCurso()
            + " - com a nota: " + nota.getNota() + " Adicionada com sucesso!");

        dao.incluirNotaAluno(nota);
        JOptionPane.showMessageDialog(null, "Curso Incluido com Sucesso");

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Fonte Própria(2019, p. 27)

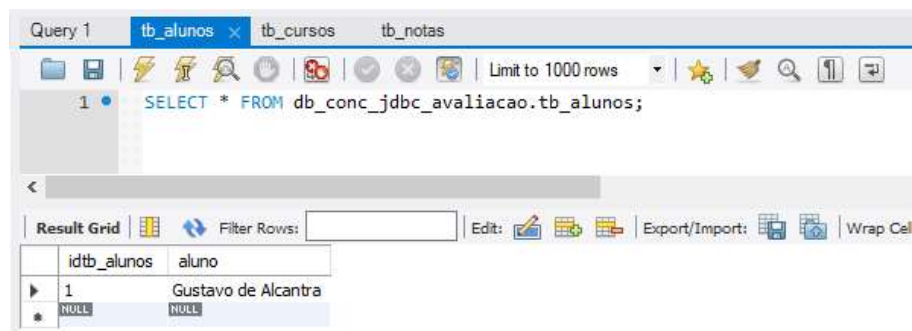


PERSISTÊNCIA EM JAVA AVALIAÇÃO DA DISCIPLINA



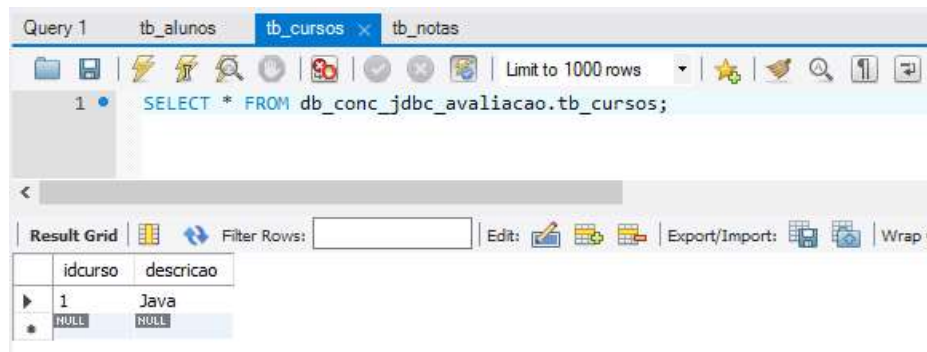
Segue alguns prints de tela da nossa aplicação rodando recebendo os dados que o usuário irá digitar. Para isso utilizamos a API **Swing** do JAVA que fornece uma interface gráfica e o componente **Scanner** da API UTIL também do JAVA que interage pela console da **IDE**.

Figura 26 – Aplicação



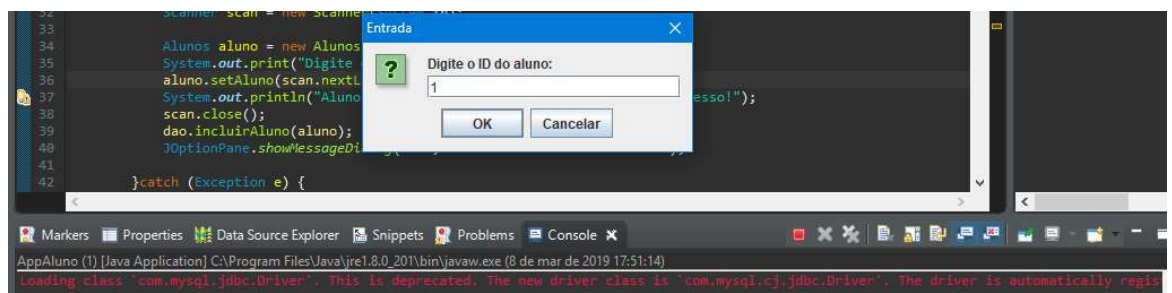
Fonte Própria(2019, p. 28)

Figura 27 – Aplicação



Fonte Própria(2019, p. 28)

Figura 28 – Aplicação



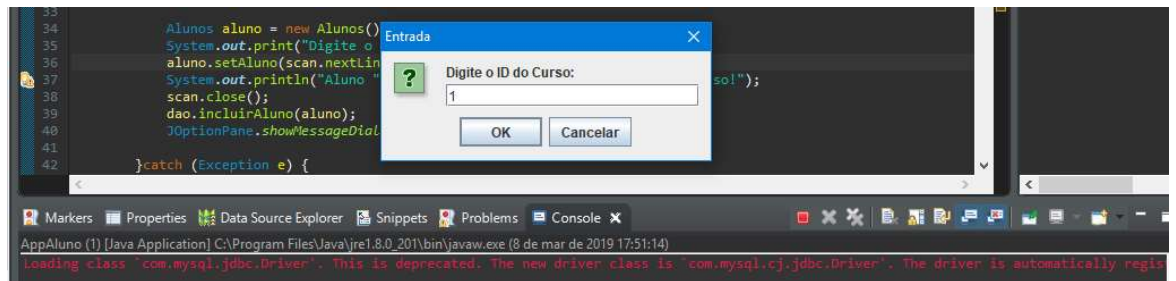
Fonte Própria(2019, p. 28)



PERSISTÊNCIA EM JAVA AVALIAÇÃO DA DISCIPLINA

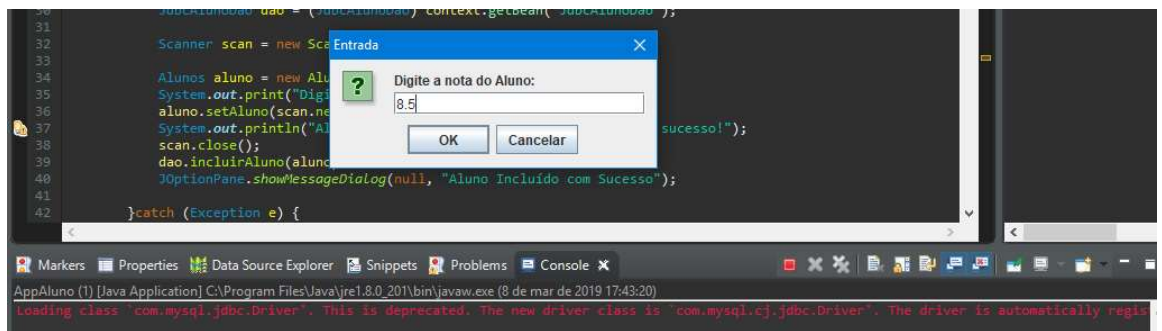


Figura 29 – Aplicação



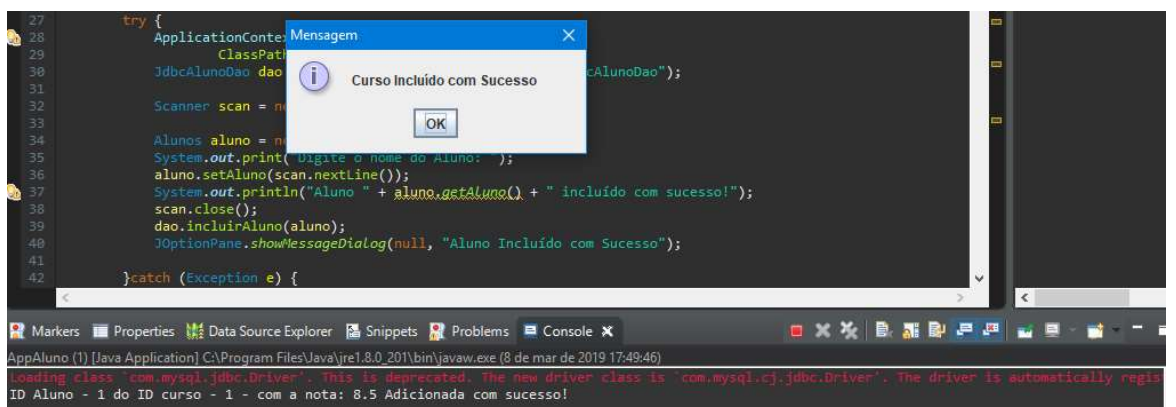
Fonte Própria(2019, p. 29)

Figura 30 – Aplicação



Fonte Própria(2019, p. 29)

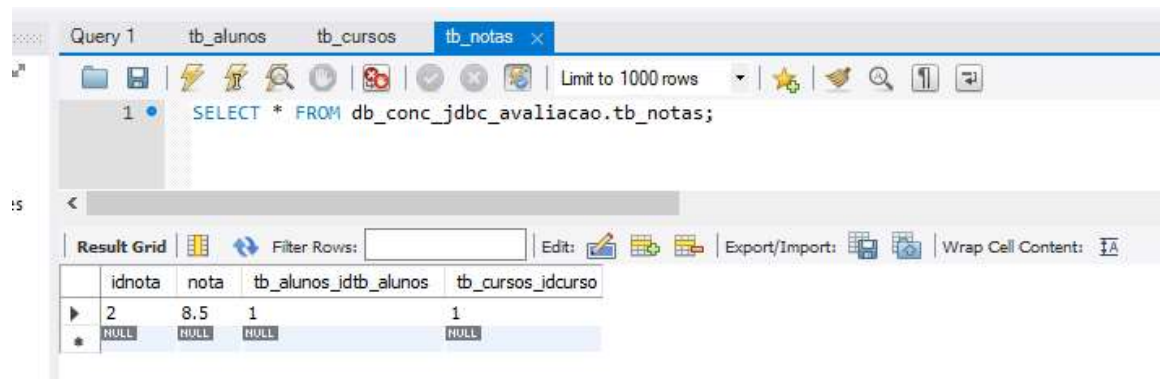
Figura 31 – Aplicação



Fonte Própria(2019, p. 29)



Figura 32 – Aplicação



Fonte Própria(2019, p. 30)

9. Referências

Segue um link para a página do Spring. Um material muito completo que nos fornece três tipos de implementação do Framework Spring JDBC:

<https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/jdbc.html>

<https://docs.oracle.com/javase/6/docs/api/java>