Persistência em Java – Avaliação da Disciplina

Turma 32SCJ

Luiz Fillipe Ribeiro Navarro	**RM332867**
Bruno Lopes	**RM332786**
Wagner Bina dos Santos	**RM332061**
Vinicius Gasparin	**RM33**

PERSISTÊNCIA EM JAVA
AVALIAÇÃO DA DISCIPLINA

MBA

Sumário

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

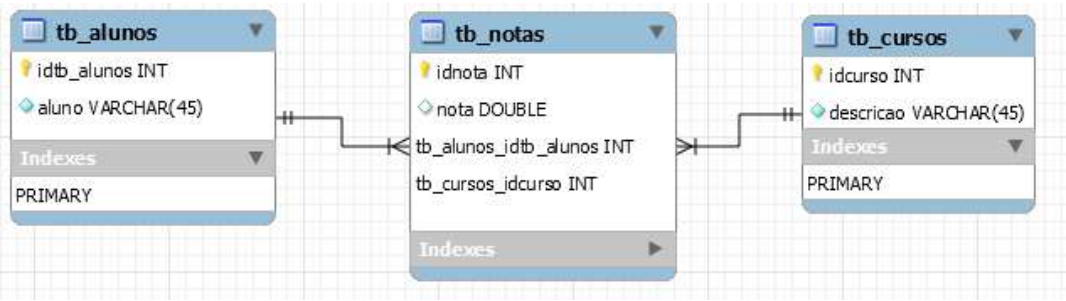
2. Objetivo

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

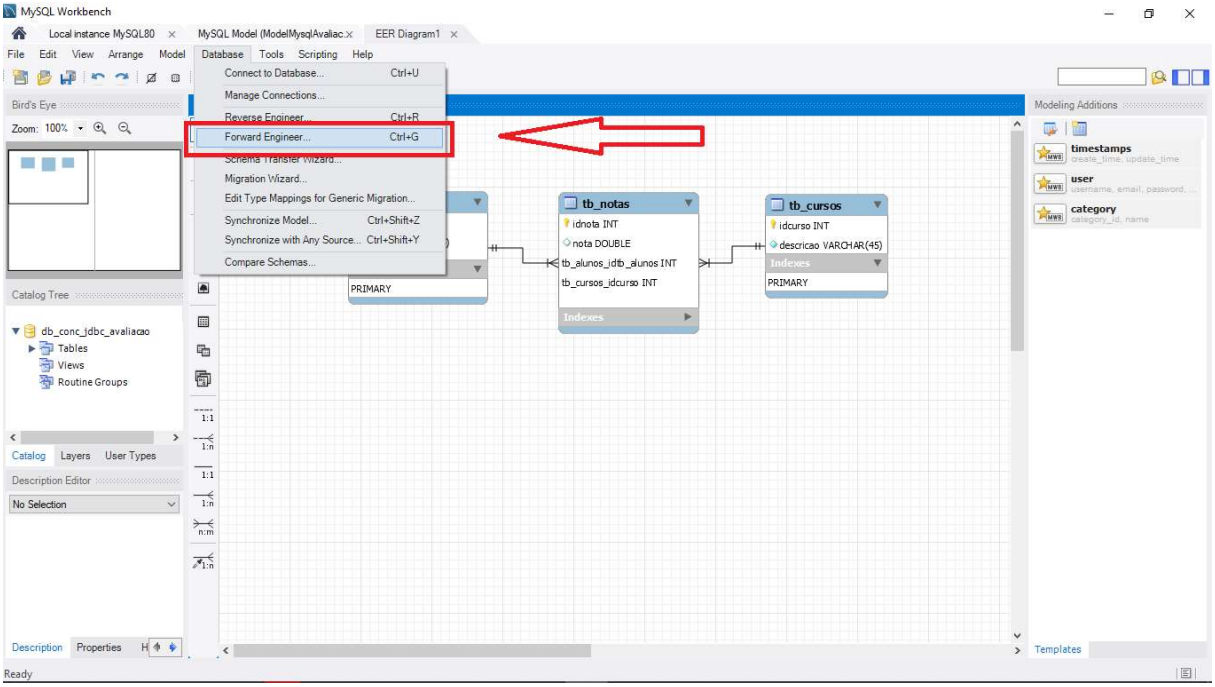


Fonte Própria(2019, p 3)

4. Gerando uma base no MySQL utilizando o modelo criado

Selecionar no menu Database a opção Foward Engineer

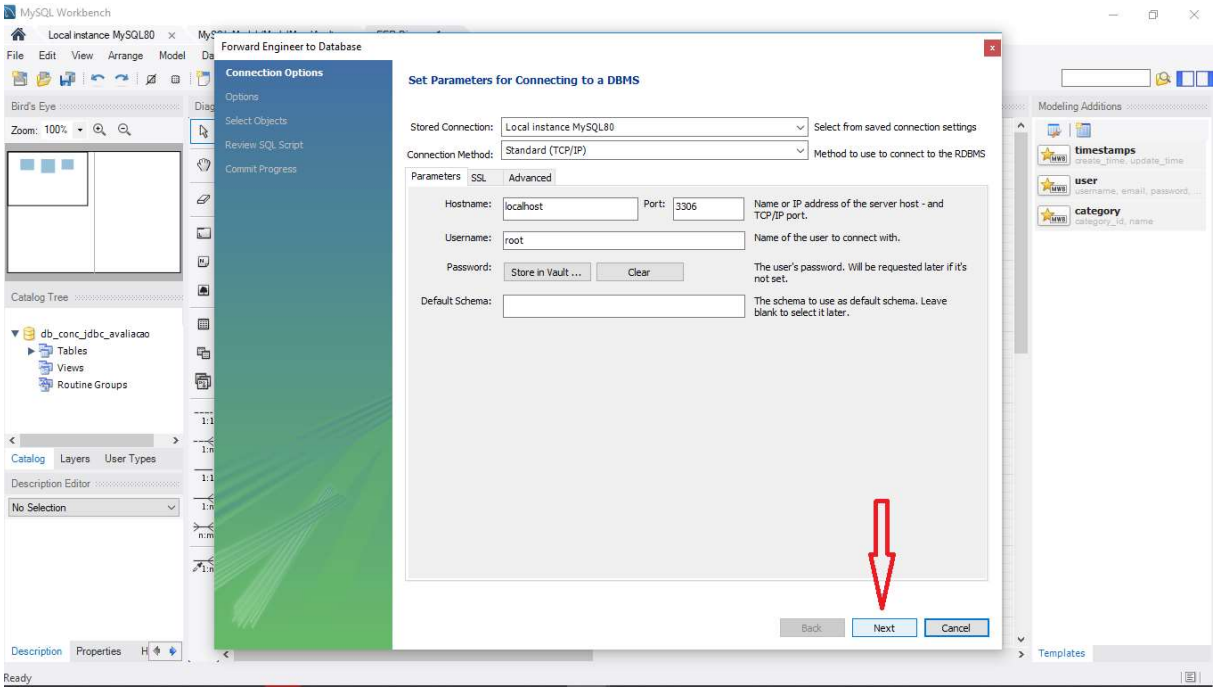
Figura 2 – Configuração base MySQL



Fonte Própria(2019, p 4)

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**.

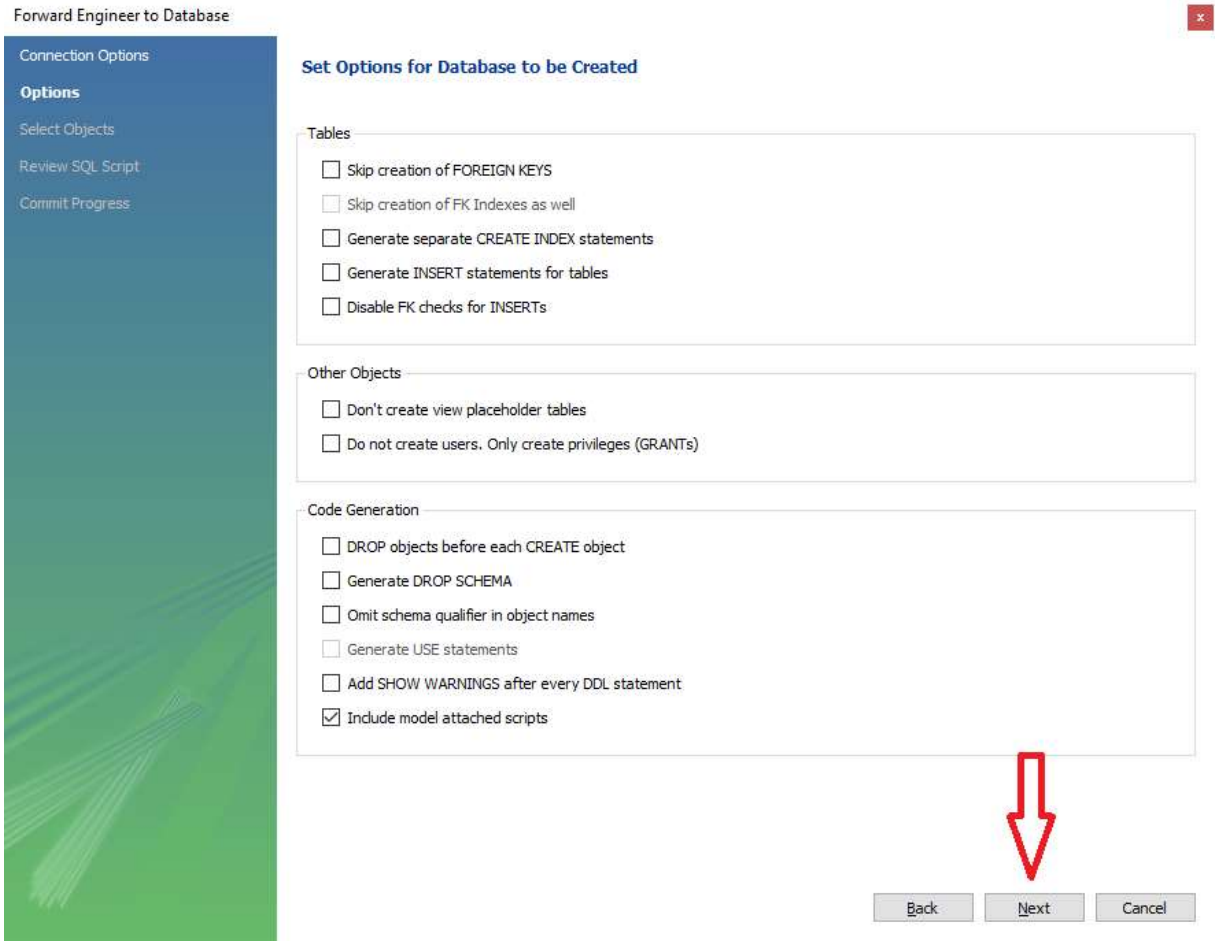
Figura 3 - Configuração Base



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.**

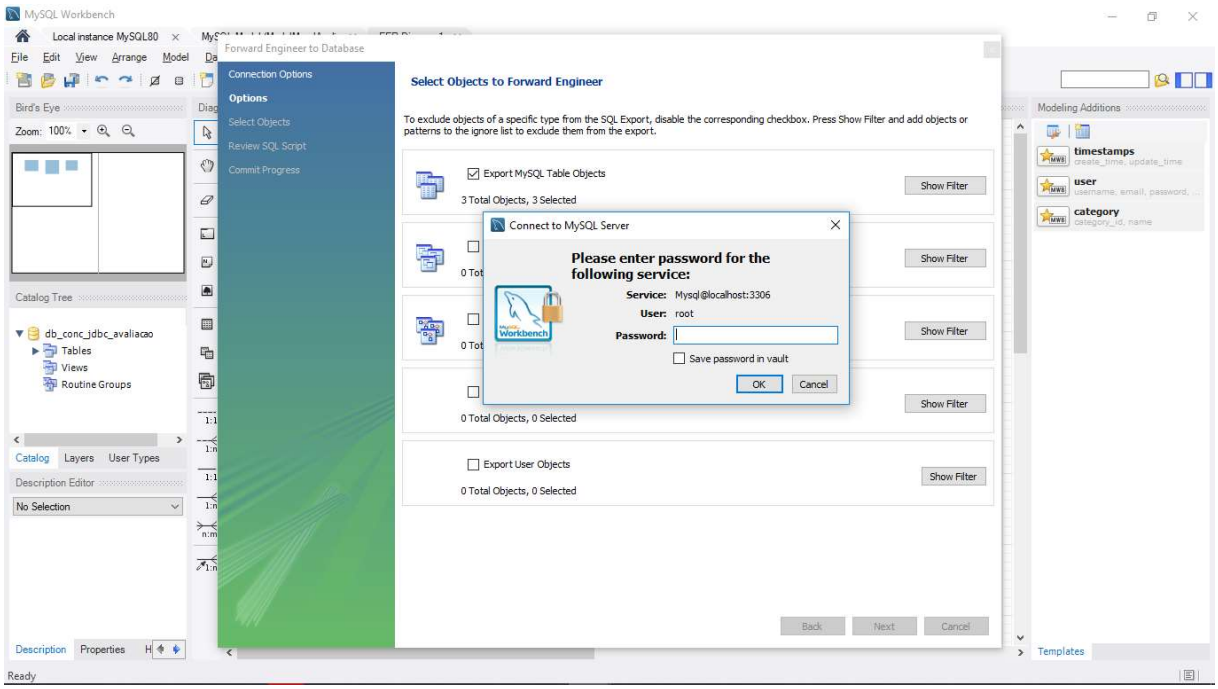
Figura 4 - Configuração base MySQL



Fonte Própria(2019, p 6)

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

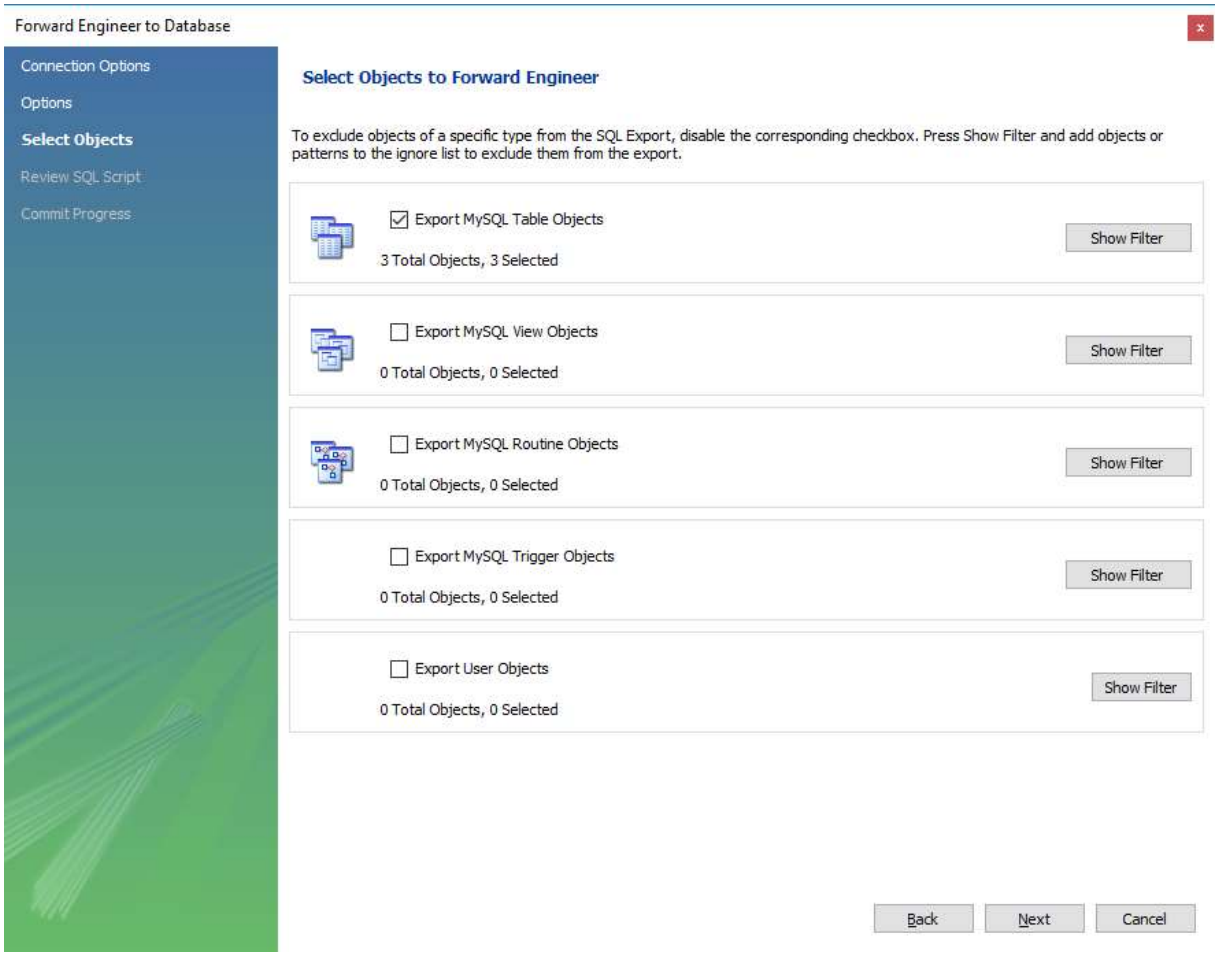
Figura 5 - Configuração base MySQL



Fonte Própria(2019, p 7)

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

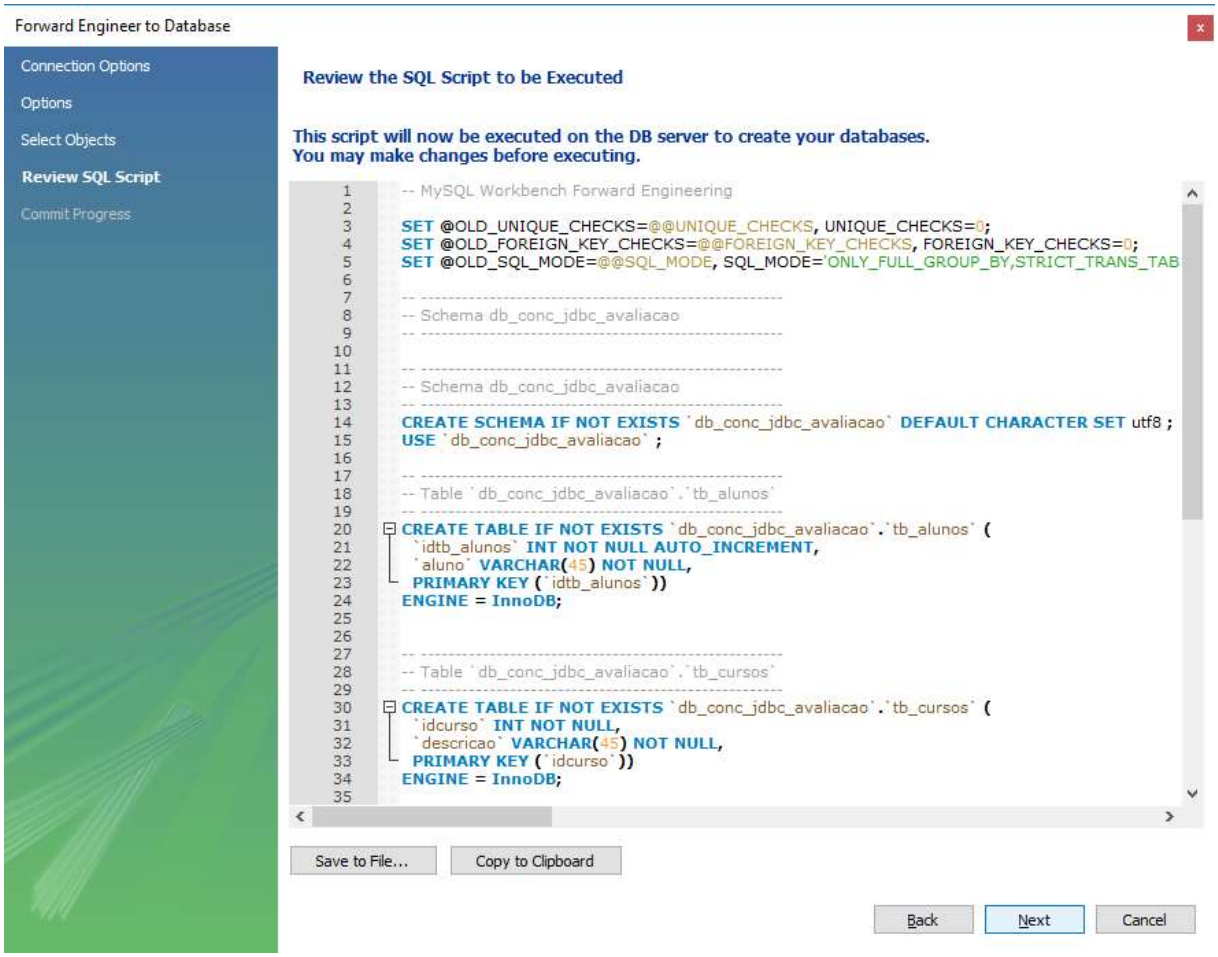
Figura 6 - Configuração base MySQL



Fonte Própria(2019, p 8)

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

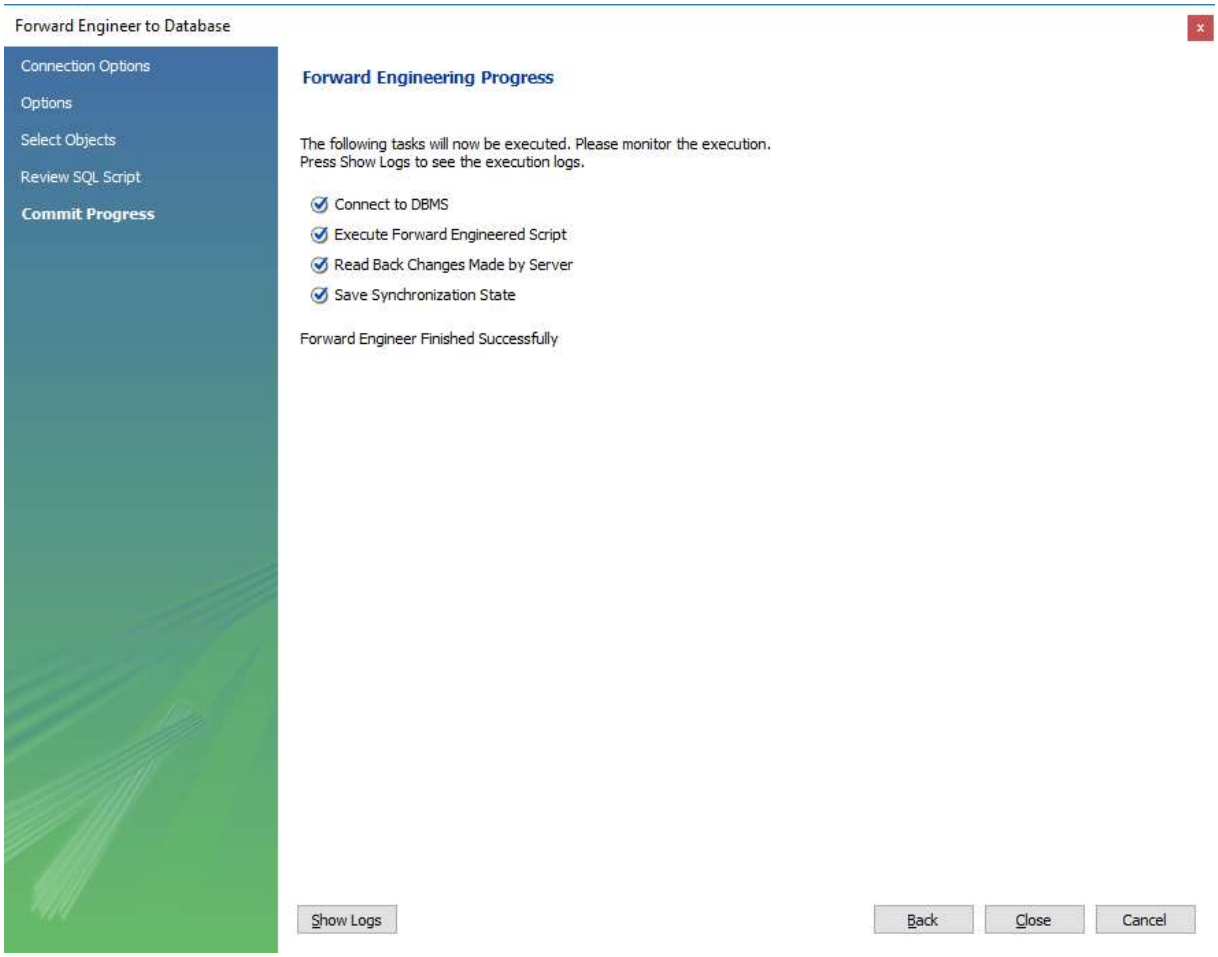
Figura 7 - Configuração base MySQL



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.

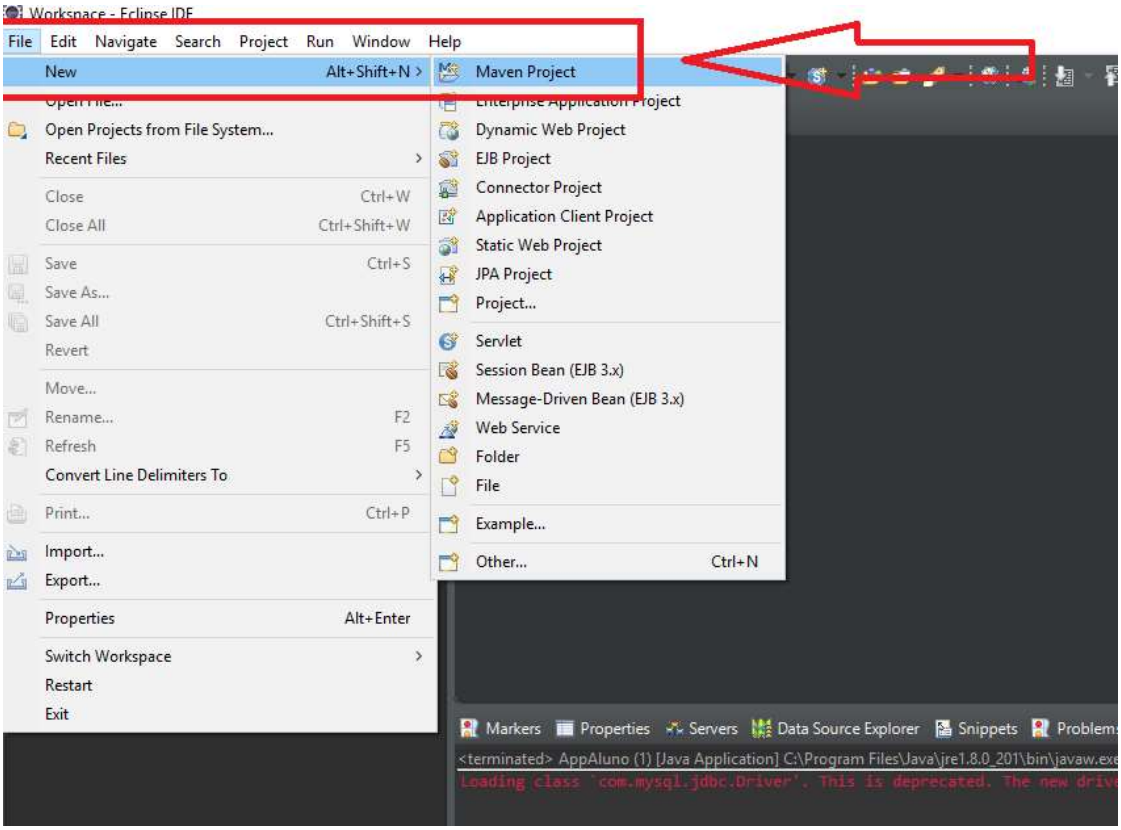
5.3 Frameworks

Spring JdbcTemplate e Maven.

6. Configuração da Persistência

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

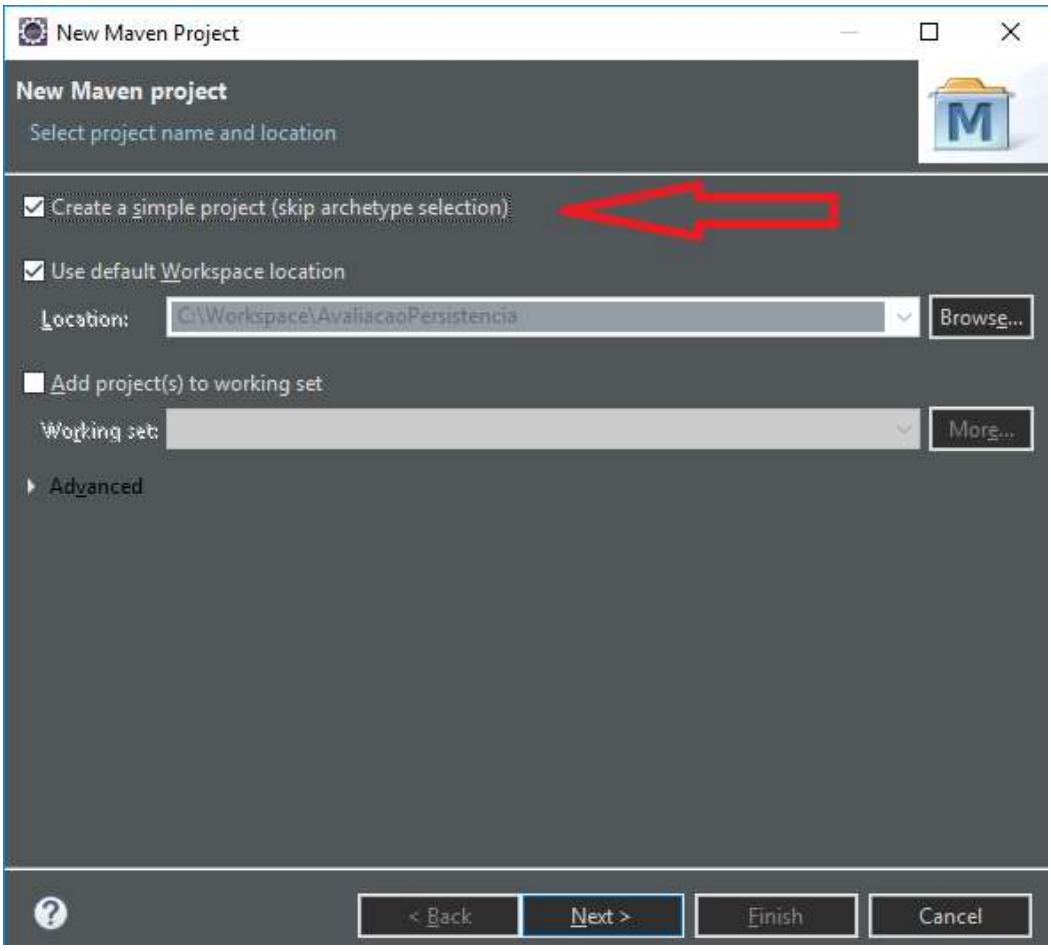
Figura 9 – Configuração da Persistência



Fonte Própria(2019, p. 11)

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

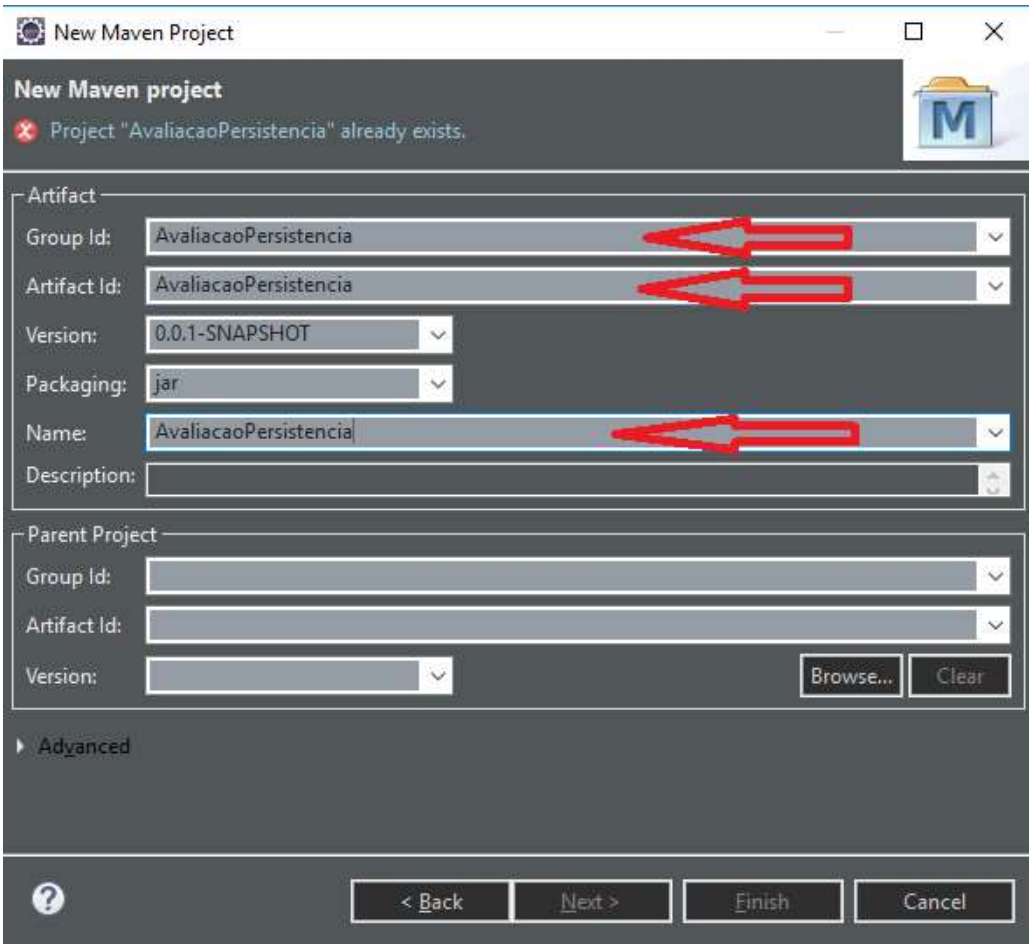
Figura 10 - Print de Tela 2



Fonte Própria(2019, p. 12)

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

Figura 11 – Configuração da Persistência

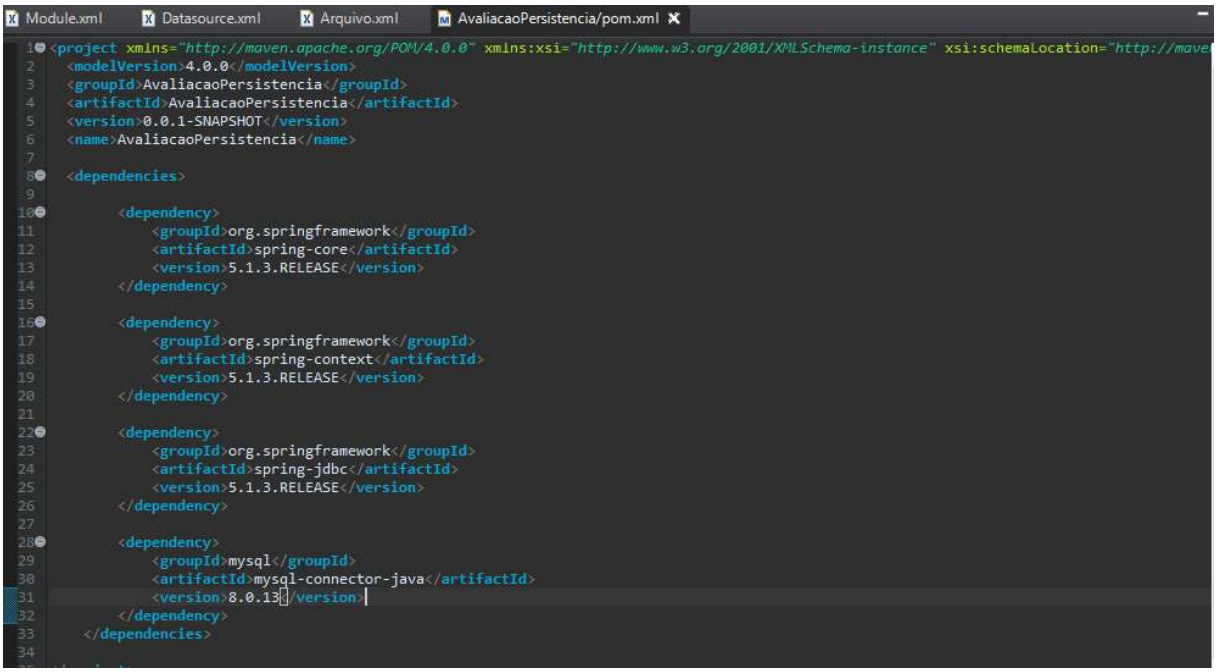


Fonte Própria(2019, p. 13)

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.

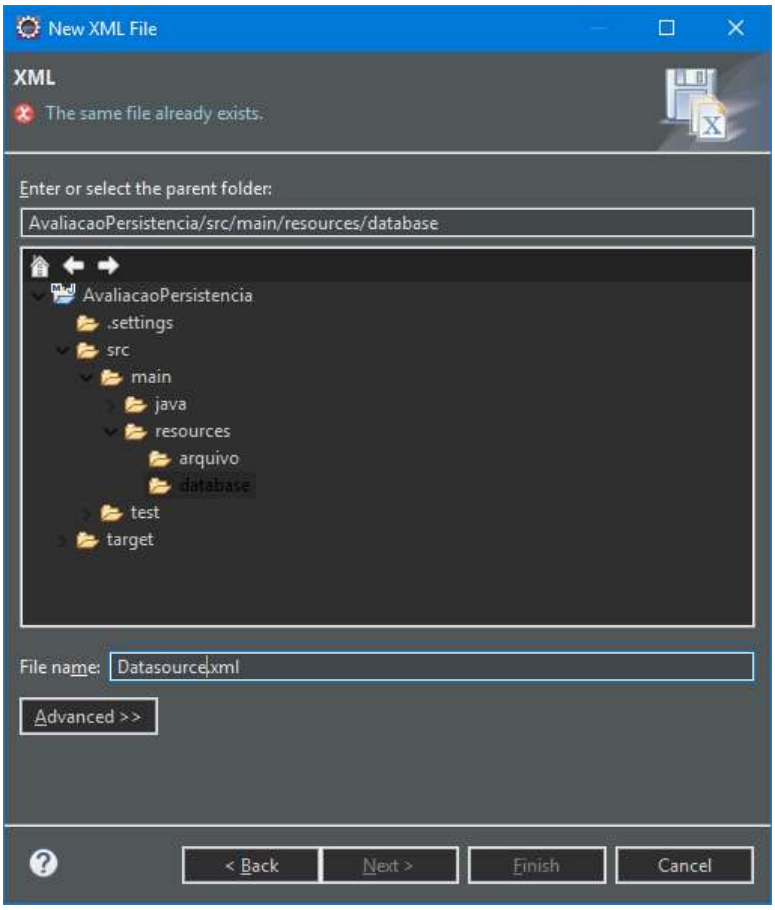
Figura 11 – Configuração da Persistência



Fonte Própria(2019, p. 14)

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

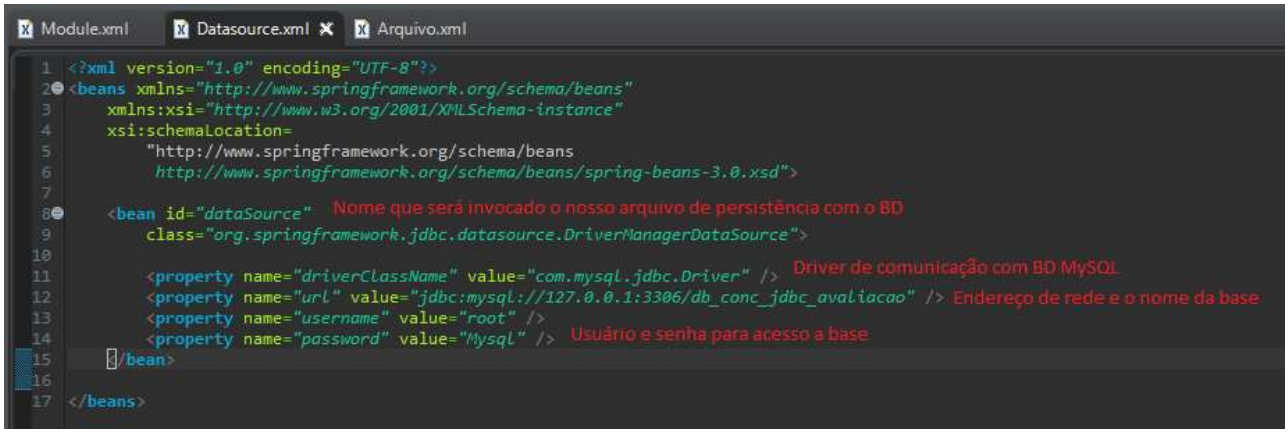
Figura 12 – Configuração da Persistência



Fonte Própria(2019, p. 15)

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

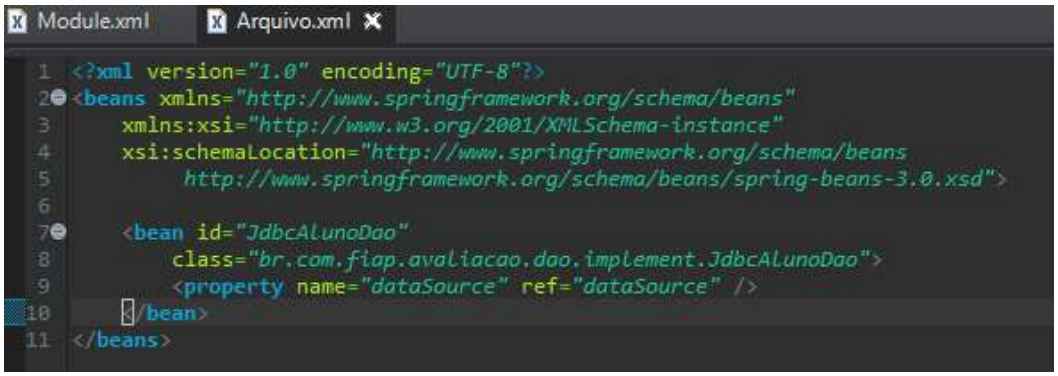
Figura 13 – Configuração da Persistência



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.xlm**. 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**”).

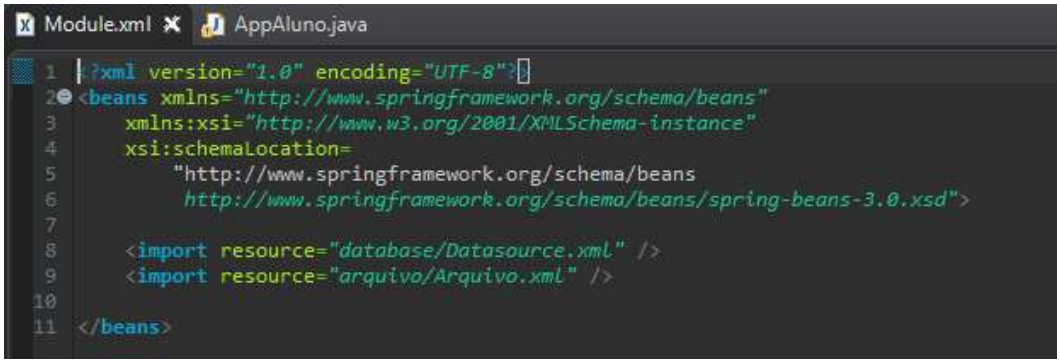
Figura 14 – Configuração da Persistência



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

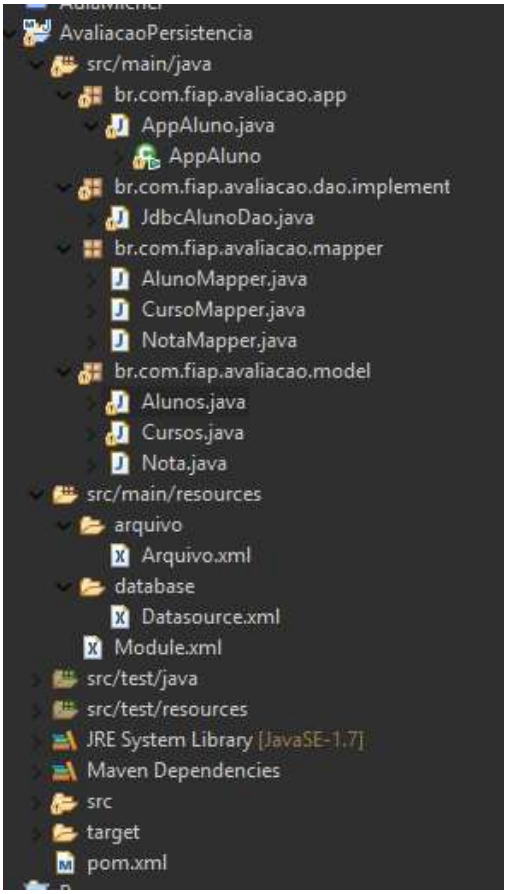


Fonte Própria(2019, p. 17)

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


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

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

public class Alunos {
    private Integer idAluno;
    private static String aluno;

    private List<Alunos> alunos = new ArrayList<>();


    public Alunos(Integer idAluno, String aluno) {
        this.idAluno = idAluno;
        this.aluno = aluno;

    }

    public Alunos() {

    }

    public Integer getIdAluno() {
        return idAluno;
    }
    public void setIdAluno(Integer idAluno) {
        this.idAluno = idAluno;
    }
    public static String getAluno() {
        return aluno;
    }
    public void setAluno(String aluno) {
        Alunos.aluno = aluno;
    }

    public List<Alunos> getAlunos() {
        return alunos;
    }

    public void setAlunos(List<Alunos> alunos) {
        this.alunos = alunos;
    }

}
```

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

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

public class Nota {
    private Integer idNota;
    private Double nota;
    private Integer idAluno;
    private Integer idCurso;


    public Integer getIdNota() {
        return idNota;
    }
    public void setIdNota(Integer idNota) {
        this.idNota = idNota;
    }
    public Double getNota() {
        return nota;
    }
    public void setNota(Double nota) {
        this.nota = nota;
    }
    public Integer getIdAluno() {
        return idAluno;
    }
    public void setIdAluno(Integer idAluno) {
        this.idAluno = idAluno;
    }
    public Integer getIdCurso() {
        return idCurso;
    }
    public void setIdCurso(Integer idCurso) {
        this.idCurso = idCurso;
    }
}
```

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

JdbcAlunoDao.java | AlunoMapper.java | CursoMapper.java | NotaMapper.java | AppAlun

```java
package br.com.fiap.avaliacao.dao.implement;

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

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;

import br.com.fiap.avaliacao.mapper.AlunoMapper;
import br.com.fiap.avaliacao.mapper.CursoMapper;
import br.com.fiap.avaliacao.mapper.NotaMapper;
import br.com.fiap.avaliacao.model.Alunos;
import br.com.fiap.avaliacao.model.Cursos;
import br.com.fiap.avaliacao.model.Nota;

public class JdbcAlunoDao {
    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public void incluirAluno(Alunos aluno) throws Exception {
        try {
            String sql = "INSERT INTO TB_ALUNOS" + "(ALUNO) VALUES (?)";

            this.jdbcTemplate.update(sql,
                    aluno.getAluno());

        } catch (Exception e) {
            throw e;
        }
    }

    public Alunos buscarAluno(int idAluno) throws Exception {
        Alunos aluno = null;
        try {
            String query = "SELECT * TB_ALUNOS WHERE IDTB_ALUNOS=?";
            aluno = this.jdbcTemplate.queryForObject(query,
                    new Integer[] {idAluno}, new AlunoMapper());

        } catch (Exception e) {
            throw e;
        }
                return aluno;
    }
```

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

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**;

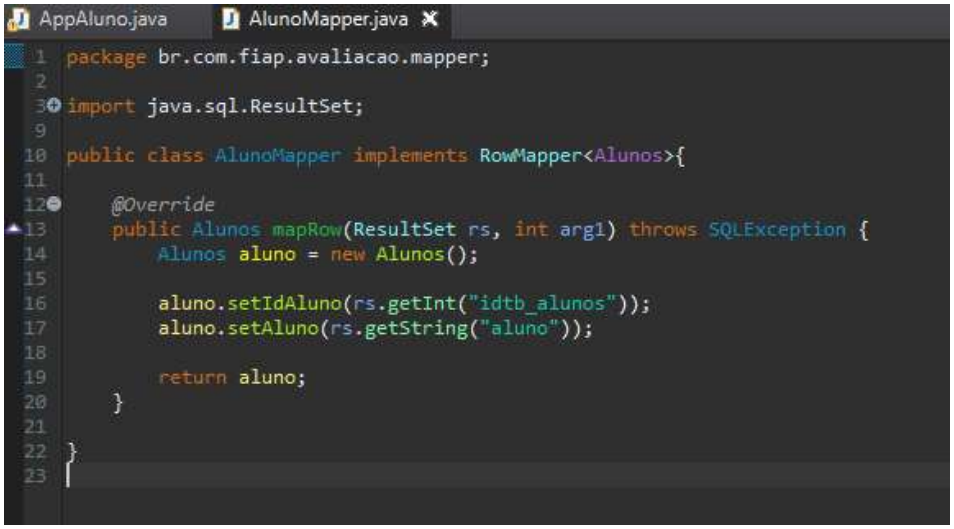
Figura 21 – Aplicação

```
public void incluirNotaAluno(Nota 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 Nota buscarNotaAluno(int idAluno) throws Exception {
    Nota nota = null;
    try {
        String query = "SELECT * TB_NOTAS WHERE TB_ALUNOS_IDTB_ALUNOS = ?";
        nota = this.jdbcTemplate.queryForObject(query,
                new Integer[] {idAluno}, new NotaMapper());

    } 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



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

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

import java.util.Scanner;

import javax.swing.JOptionPane;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import br.com.fiap.avaliacao.dao.implement.JdbcAlunoDao;
import br.com.fiap.avaliacao.model.Alunos;
import br.com.fiap.avaliacao.model.Cursos;
import br.com.fiap.avaliacao.model.Nota;

public class AppAluno {
    public static void main(String[] args) {
//      incluirAluno();
//      buscarIuno();
//      listarAlunos();

//      incluirCurso();

        incluirNotaAluno();
    }

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

            Scanner scan = new Scanner(System.in);

            Alunos aluno = new Alunos();
            System.out.print("Digite o nome do Aluno: ");
            aluno.setAluno(scan.nextLine());
            System.out.println("Aluno " + aluno.getAluno() + " incluído com sucesso!");
            scan.close();
            dao.incluirAluno(aluno);
            JOptionPane.showMessageDialog(null, "Aluno Incluído com Sucesso");

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

Fonte Própria(2019, p. 26)

Figura 24 – Aplicação

AppAluno.java

```
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() + " incluído com sucesso!");
59              scan.close();
60              dao.incluirCurso(curso);
61              JOptionPane.showMessageDialog(null, "Curso Incluído com Sucesso");
62
63          }catch (Exception e) {
64              e.printStackTrace();
65          }
66      }
67
```

Fonte Própria(2019, p. 27)

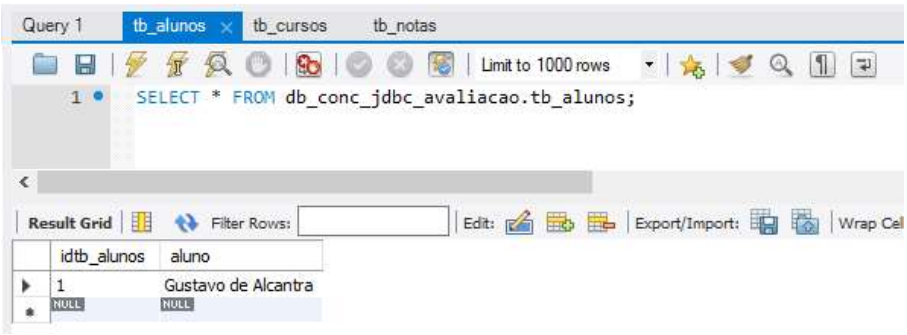
Figura 25 – Aplicação

```
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 Incluído com Sucesso");

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

Fonte Própria(2019, p. 27)

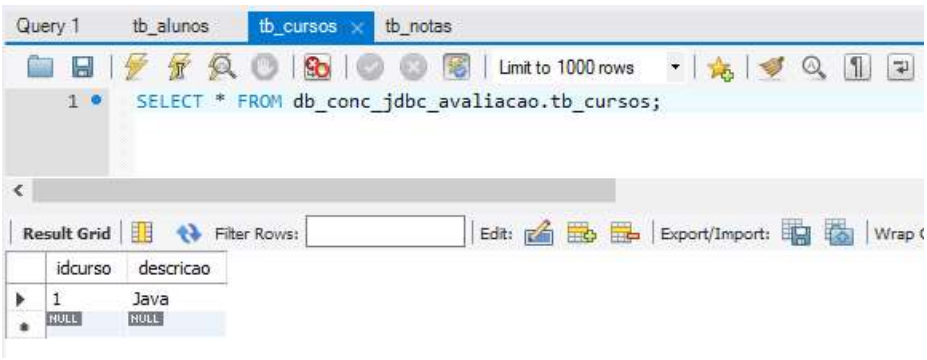
Segue alguns prints de tela da nossa aplicação rodando recebendo os dados que o usuário irá digitar. Para isso utilizamos a API **Swing** do JAVA que fornece uma interface gráfica e o componente **Scanner** da API UTIL também do JAVA que interage pela console da **IDE**.

Figura 26 – Aplicação



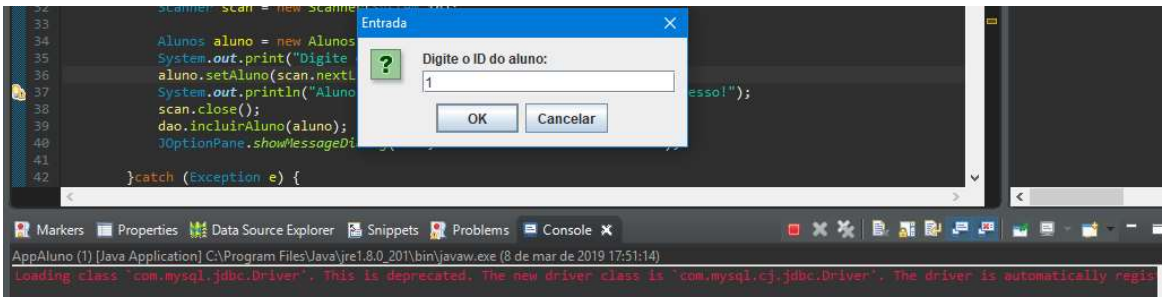
Fonte Própria(2019, p. 28)

Figura 27 – Aplicação



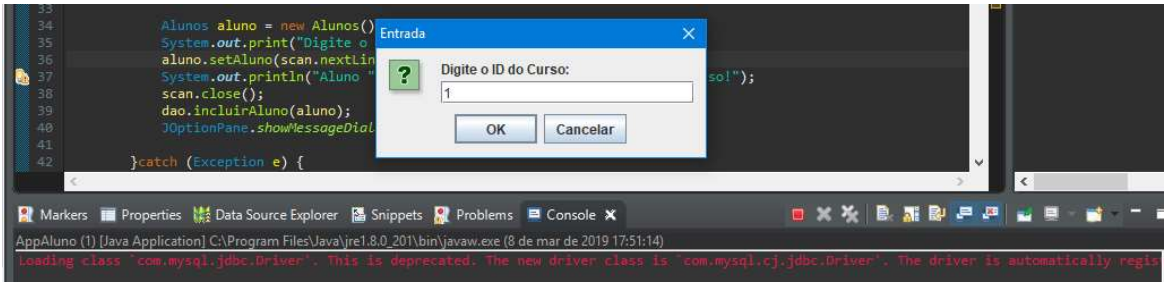
Fonte Própria(2019, p. 28)

Figura 28 – Aplicação



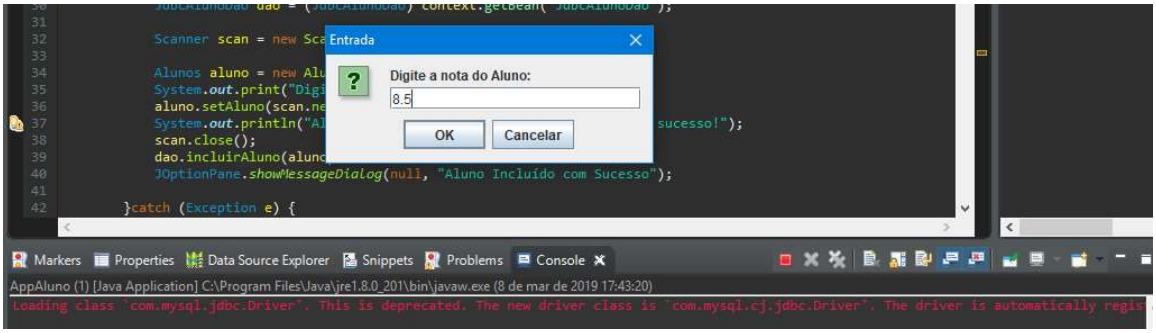
Fonte Própria(2019, p. 28)

Figura 29 – Aplicação



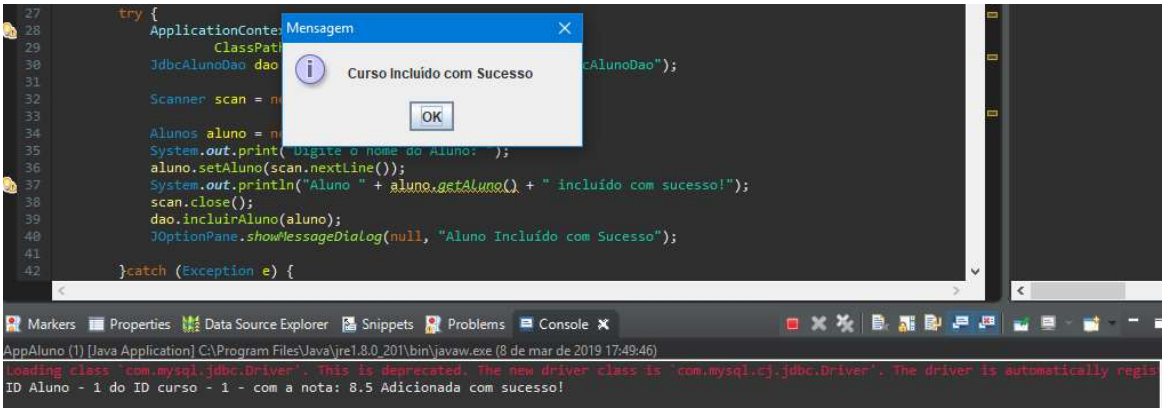
Fonte Própria(2019, p. 29)

Figura 30 – Aplicação



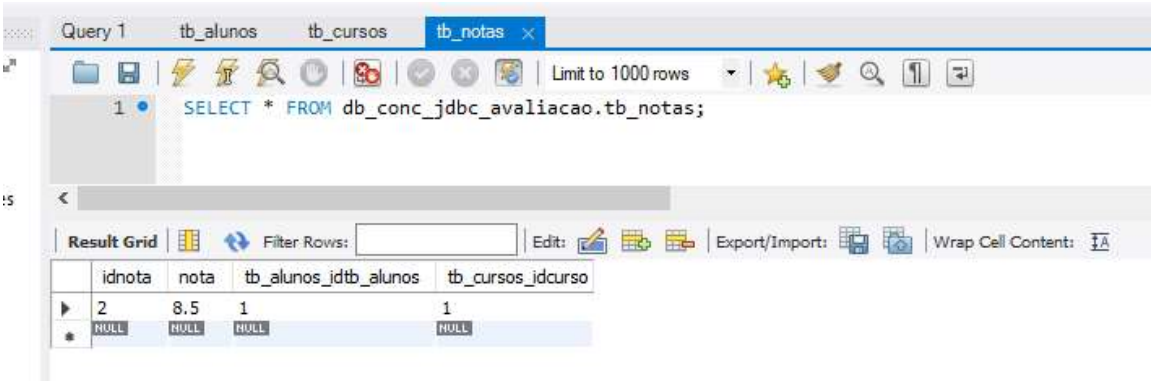
Fonte Própria(2019, p. 29)

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