

# Tecnologias JDBC e JPA da Teoria à Prática.

Dionisio Gause Junior  
Tutor Externo: Robson Ortiz Rodrigues

## RESUMO

*A persistência e cuidado das informações que necessitam ser armazenadas dependem da linguagem de programação escolhida para seu desenvolvimento bem como qual o gerenciador de banco de dados será utilizado ou que já esteja em uso, forçando desta forma a busca e a utilização de ferramentas disponíveis que sirvam de ponte de comunicação entre a linguagem escolhida e o gerenciador. Entender os problemas enfrentados no dia a dia e as soluções empregadas, demonstrando de forma cuidadosa os passos que o futuro profissional e desenvolvedor de sistemas deve ter e trazendo a luz do conhecimento do graduando a Linguagem de Programação JAVA. Valemo-nos para tanto da pesquisa documental trazendo informações relevantes para a tomada de decisão sobre a utilização de APIs (Application Programming Interface) em português “Interface de Programação de Aplicativos”, abordando a API JDBC (Java Database Connectivity) em português “Conexão a Base de Dados Java” projetada unicamente para se trabalhar com bancos de dados relacionais e a mais atual API JPA (Java Persistence API) em português “Interface de Programação de Aplicativos para persistência em Java”, bem como a utilização das implementações como o Hibernate, utilizada para as operações sql junto aos bancos de dados relacionais. Munir o graduando e futuro profissional destas informações quanto ao uso das ferramentas, as quais podem impactar em seus futuros projetos de desenvolvimento de sistemas demonstrando de forma teórica e prática o seu uso na construção de um sistema computacional robusto e confiável na guarda e manipulação dos dados sendo de suma importância ao futuro profissional da área de Tecnologia da Informação.*

Palavras-chaves: Java , JDBC, JPA, Hibernate , Linguagens de Programação.

## 1. INTRODUÇÃO

O desenvolvimento do presente trabalho buscou compreender as funcionalidades disponíveis na linguagem de programação JAVA referente a utilização das APIs JDBC e JPA e sua implementação Hibernate, para a operação com banco de dado relacional MySQL.

A pesquisa documental nos revelou uma vasta fonte de dados a serem pesquisados, bem como códigos fontes disponíveis e uma comunidade de desenvolvedores dentro e fora do nosso país que compartilham seus conhecimentos.

A mudança do paradigma de linguagem orientada a objeto como é o JAVA para o trabalho com os bancos de dados relacionais se deu-se ao fato de que estes estão há muito mais tempo no mercado, além de já terem superados grandes desafios quanto a segurança, persistência e controle interno e que têm em seus gerenciadores, ferramentas bem desenvolvidas.

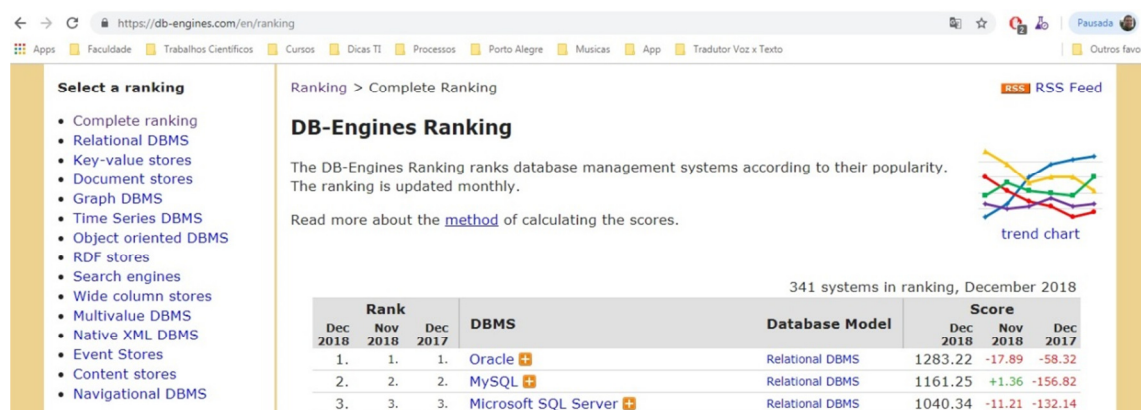
Com o desenvolvimento da programação orientada a objeto, bancos de dados com este objetivo como o NoSql estão tomando espaço lentamente junto a mercado, mas como o predomínio ainda está sob a tutela dos bancos relacionais, esta tendência deve afetar o futuro dos profissionais.

## 2. FUNDAMENTAÇÃO TEÓRICA

O desenvolvedor de um sistema deve conhecer as ferramentas disponíveis no mercado e optar por aquela que atenda de forma eficaz as necessidades das soluções que irá disponibilizar aos usuários e clientes, soluções que devem garantir a persistência dos dados de forma segura e confiável, se tornando uma solução rentável e com baixo custo de manutenção, observando a dinamicidade do mercado da tecnologia da informação.

Nos anos 90 nasce a linguagem de programação Java e de lá para cá ela tem se desenvolvido e se adaptado constantemente ao mercado, e uma dessas adaptações importantes realizadas foi a forma com que a linguagem voltada ao desenvolvimento orientado a objeto tratou a necessidade de operar com bancos de dados relacionais, consolidados há anos e que dominam o mercado.

A pesquisa divulgada no site da *DB-Engines*<sup>1</sup> aponta que os três primeiros lugares são ocupados por gerenciadores de bancos de dados relacionais.



Fonte: <<https://db-engines.com/en/ranking> - acessado em 01/12/2018 - 11:03>

<sup>1</sup> <<https://db-engines.com/en/ranking> - acessado em 01/12/2018 - 11:03>

A reflexão sobre o conceito conhecido como mapeamento objeto relacional (MOR) se faz necessário uma vez que a teoria da programação orientada a objeto é baseada na *teoria dos grafos*<sup>2</sup> onde trata o problema de forma simplificada e objetiva com o uso de pontos ou vértices e linhas ou arestas conforme explicação realizada em vídeo do Professor Marcos Paulo Ferreira de Araújo no Programa de Iniciação Científica da OBMEP, e os bancos de dados relacionais são baseados na *teoria dos conjuntos que trata o problema de forma matemática capaz de agrupar elementos*<sup>3</sup>, levando a linguagem Java a adaptar-se.

Desta forma abordamos o conceito conhecido como mapeamento objeto relacional (MOR) que se faz necessário para que as funcionalidades das APIs JDBC e JPA sejam implementadas.

Quando trabalhamos com uma aplicação Java, seguimos o paradigma orientado a objetos, onde representamos nossas informações por meio de classes e atributos. Além disso, podemos utilizar também herança, composição para relacionar atributos, polimorfismo, enumerações, entre outros.<sup>4</sup>

Cardim(2010) afirma que objetos, relações e registros são conceitos de natureza fundamentalmente diferentes, apesar de apresentarem algumas similaridades deceptivas. A navegação entre objetos ocorre através de ponteiros e referências, enquanto as relações são unidas através de produtos cartesianos e chaves estrangeiras, o que acaba levando a soluções de otimização diferentes para ambos os casos, além das próprias operações de acesso.<sup>5</sup>

Para construir uma ponte de comunicação entre estes dois paradigmas em que *“a todo momento devemos “transformar” objetos em registros e registros em objetos”*<sup>6</sup> é que as *“ferramentas para auxiliar nesta tarefa tornaram-se popular entre os desenvolvedores Java e são conhecidas como ferramentas de mapeamento objeto-relacional (ORM)”*<sup>7</sup>.

Conforme JavaFree(2014), a Java Database Connectivity ou JDBC é um conjunto de classes e interfaces escritas em Java que faz o envio de instruções SQL para qualquer banco de dados relacional.<sup>8</sup>

O funcionamento da JPA é o de trabalhar em conjunto com outra ferramenta de mapeamento que se encarrega de *“gerar o SQL que serve para um determinado banco de dados, já que cada banco fala um “dialetto” diferente dessa linguagem.”*<sup>9</sup>.

Franco(2014, p.194) aponta que:

*Uma das soluções mais aplicadas para o problema de mapeamento objeto relacional é a utilização do padrão de projeto conhecido como DAO (data*

---

<sup>2</sup> <<https://www.youtube.com/watch?v=Frnwtdter-vQ> – Acessado em 01/12/2018 – 14:10>

<sup>3</sup> <<https://www.todamateria.com.br/teoria-dos-conjuntos/> - acessado em 01/12/2018 - 16:00>

<sup>4</sup> Franco (2014, p.210)

<sup>5</sup> Cardim (2010) Apud Franco (2014, p.192)

<sup>6</sup> Franco (2014, p.210)

<sup>7</sup> Franco (2014, p.210)

<sup>8</sup> Franco (2014, p.196).

<sup>9</sup> Franco (2014, p.211)

*access object). Esse padrão consiste em implementar uma camada para fazer o mapeamento entre os objetos e as tabelas do banco de dados.*<sup>10</sup>

## 2. DESENVOLVENDO A APLICAÇÃO

Um dos primeiros passos ao que o desenvolvedor deve se ater é a configuração do ambiente de desenvolvimento ao qual se propõe criar sistemas. Tendo isto em mente os procedimentos de levantamento de requisitos necessários para a criação de nossa aplicação forçou a criação de uma “checklist”.

Configuração do Ambiente computacional, Instalações de ferramentas de desenvolvimento, instalação de gerenciador de Banco de Dados e o desenvolvimento propriamente dito.

## 3. CONFIGURAÇÃO DO AMBIENTE

A configuração inicial de hardware e softwares utilizados para o desenvolvimento do sistema possui: Processador Intel Core i3 2.10Ghz, 4GB Memória, Windows 7 SP1, Eclipse IDE 2018/2019 (4.9.0).

As configurações adicionais necessárias para o desenvolvimento foram implementadas conforme segue: Instalação pacote JBoss Tools 4.9 no Eclipse, Instalação do Wildfly-14.0.1.Final e posterior configuração no Eclipse e Instalação do MySQL 8.0.13.0 community e posterior configuração.

Para que os sistemas desenvolvidos façam as conexões JDBC em clientes é necessário o download de Drivers e posterior configuração da conexão nos equipamentos. Os Drivers são disponibilizados no site do MySQL que apresenta a seguinte afirmativa *“MySQL provides standards-based drivers for JDBC, ODBC, and .Net enabling developers to build database applications in their language of choice.”*<sup>11</sup> traduzindo para o português *“O MySQL fornece drivers baseados em padrões para JDBC, ODBC e .Net, permitindo que os desenvolvedores criem aplicativos de banco de dados em seu idioma de preferência.”*.

Após a configuração do ambiente para o desenvolvimento do sistema inicia-se o desenvolvimento propriamente dito.

## 4. MODELANDO A BASE DE DADOS

Para o Analista e Desenvolvedor de sistemas não basta estar com o ambiente configurado e “colocar a mão na massa” para começar a programar. Algumas etapas devem ser observadas para o sucesso do projeto, pois se faz necessário a criação de um Modelo de Processo, conforme afirma Aléssio; Simone Cristina (2017, p.30) quando disserta sobre as atividades a serem desenvolvidas, criando um esboço do processo com as etapas bem definidas e seus responsáveis, onde destacam-se a Análise e especificação de requisitos e o Projeto em si.

---

<sup>10</sup> Franco(2014, p.194)

<sup>11</sup> <disponível em - <https://www.mysql.com/products/connector/> >

No Projeto a etapa de modelagem de banco de dados, parte importante no desenvolvimento, busca retratar o mais fielmente possível a realidade e as solicitações dos usuários/clientes, tendo como base fundamental o levantamento de requisitos.

Na aplicação desenvolvida, criou-se o Banco de Dados *cadastro* e a tabela *cliente* com seus atributos *codigo*, *nome*, *telefone* e *email*, para demonstrar as funcionalidades e especificidades da utilização das APIs JDBC e JPA, suas conexões e as principais operações básicas com os registros, como inclusão, alteração e exclusão e listagem.

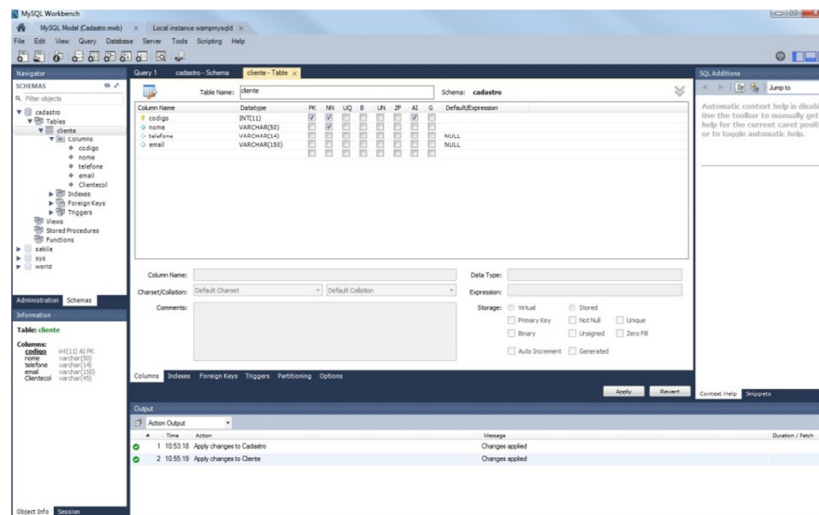
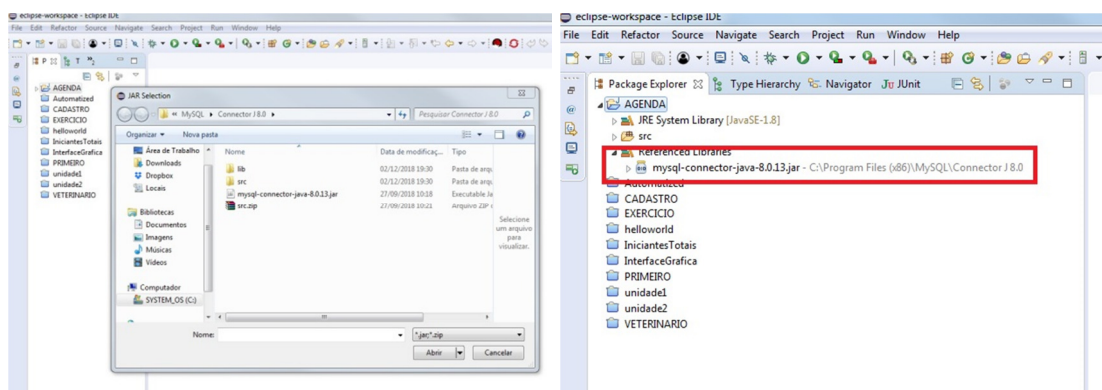


Figura: Criação Banco de Dados cadastro e Tabela cliente

## 5. DESENVOLVENDO A APLICAÇÃO COM Eclipse e API JDBC

O passo seguinte à criação do banco de dados foi o desenvolvimento da aplicação no ECLIPSE.

Após criar um novo projeto procede-se a adição do conector do Java para o MySQL no projeto, clicando com o botão direito do mouse no projeto, clicando em <Build Path> e em <Add External Archives> e escolhe o arquivo do conector e clica no botão <Abrir>.



Criamos duas classes uma Agenda e outra BancoDeDados, onde a classe Agenda terá o método psvm - “public static void main(String[] args);” o qual inicializa o nosso aplicativo e no qual instanciamos outros métodos disponíveis em nosso pacote.

A classe BancoDeDados conterá os métodos para as conexões e manipulações dos dados em nosso aplicativo.

O método estaConectado(), testa se a conexão com o banco de dados está ativa, retornando true ou false. A inclusão de registros é realizado pelo método inserirCliente() com seus parâmetros, as alterações nos registros é realizado pelo método editarCliente() com o parâmetros código como Chave para a busca do registro e alteração dos dados conforme os parâmetros, o método apagarCliente() exclui o registro tendo como chave o código e o método desconectar() realiza a desconexão do banco de dados.

A classe Agenda conterá as instancias de um objeto banco de dados e os métodos da classe BancoDeDados. Instanciamos o objeto Banco de Dados da Classe BancoDeDados, chamando o método conectar(), criamos o teste para verificar a conexão através do método estaConectado(), que faz os procedimentos e retorna com uma mensagem caso ocorra de não estar conectado, através do retorno true ou false do método, estando true, procede com os métodos que em nosso exemplo deixamos comentados e executa os métodos, o método listagem(), inserirCliente(), editarCliente(), apagarCliente() e desconectar(), os quais podemos rodar um a um para observar o funcionamento de cada método, demonstrando de forma simplificada, sem tratamento dos dados inseridos, mas respeitando o tipo de cada parâmetro.

Cabe observar que a implementação da chave primária é realizada diretamente na tabela do banco de dados ficando a cargo do gerenciador do banco a sua criação e controle. Ainda nas linhas 14, 16 e 18 ficam comentadas para que sejam testados método a método, como informado anteriormente demonstrando o funcionamento de cada método e da API JDBC.

## Código Fonte da classe Agenda.java

```
1  Agenda.java  BancoDeDados.java
2  public class Agenda {
3
4  public static void main(String[] args) {
5
6      BancoDeDados bancoDeDados = new BancoDeDados();
7
8      bancoDeDados.conectar();
9
10     if(bancoDeDados.estaConectado()) {
11
12         bancoDeDados.listarClientes();
13
14         //bancoDeDados.inserirCliente("Maria", "999999999", "maria@hotmail.com");
15
16         //bancoDeDados.editarCliente(1, "Dionisio Gause Junior", "981294394", "dgausejr@gmail.com");
17
18         //bancoDeDados.apagarCliente(1);
19
20         bancoDeDados.desconectar();
21     } else {
22         System.out.println("Não foi possível conectar ao banco de dados!");
23     }
24 }
25
26
27 }
```

## Código Fonte da classe BancoDeDados.java

```
1
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.ResultSet;
5 import java.sql.Statement;
6
7 public class BancoDeDados {
8     private Connection connection = null;
9     private Statement statement = null;
10    private ResultSet resultSet = null;
11
12    public void conectar() {
13        String servidor = "jdbc:mysql://localhost:3306/cadastro?useTimezone=true&serverTimezone=UTC";
14        String usuario = "root";
15        String senha = "8032";
16        String driver = "com.mysql.cj.jdbc.Driver";
17        try {
18            Class.forName(driver);
19            this.connection = DriverManager.getConnection(servidor, usuario, senha);
20            this.statement = this.connection.createStatement();
21        } catch (Exception e) {
22            System.out.println("Erro: " + e.getMessage());
23        }
24    }
25
26    public boolean estaConectado() {
27        if(this.connection != null) {
28            return true;
29        } else {
30            return false;
31        }
32    }
33
34    public void listarClientes() {
35        try {
36            String query = "SELECT * FROM cliente ORDER BY nome";
37            this.resultset = this.statement.executeQuery(query);
38            //this.statement = this.connection.createStatement();
39            while(this.resultset.next()) {
40                System.out.println("Codigo: " + this.resultset.getString("codigo") +
41                    " Nome: " + this.resultset.getString("nome") +
42                    " Telefone: " + this.resultset.getString("telefone") +
43                    " Email: " + this.resultset.getString("email"));
44            }
45        } catch (Exception e) {
46            System.out.println("Erro: " + e.getMessage());
47        }
48    }
49
50    public void inserirCliente(String nome, String telefone, String email) {
51        try {
52            String query = "INSERT INTO cliente (nome,telefone,email) VALUES ('"+nome+"', '"+telefone+"', '"+email+"')";
53            this.statement.executeUpdate(query);
54        } catch (Exception e) {
55            System.out.println("Erro: " + e.getMessage());
56        }
57    }
58
59    public void editarCliente(int codigo, String nome, String telefone, String email) {
60        try {
61            String query = "UPDATE cliente SET nome = '"+nome+"', telefone = '"+telefone+"', email = '"+email+"' WHERE codigo = '"+codigo+"'";
62            this.statement.executeUpdate(query);
63        } catch (Exception e) {
64            System.out.println("Erro: " + e.getMessage());
65        }
66    }
67
68    public void apagarCliente(int codigo) {
69        try {
70            String query = "DELETE FROM cliente WHERE codigo = '"+codigo+"'";
71            this.statement.executeUpdate(query);
72        } catch (Exception e) {
73            System.out.println("Erro: " + e.getMessage());
74        }
75    }
76    public void desconectar() {
77        try {
78            this.connection.close();
79        } catch (Exception e) {
80            System.out.println("Erro: " + e.getMessage());
81        }
82    }
83
84 }
85 }
```

## 6. DESENVOLVENDO A APLICAÇÃO COM Eclipse e API JPA

O Java Persistence API(JPA) utiliza-se de ferramentas de mapeamento objeto relacional (ORM), que “*abstrai o seu código SQL, toda camada JDBC e o SQL será gerado em tempo de execução. Mais que isso, ele vai gerar p SQL que serve para um determinado banco de dados, já que cada banco fala um “dialetto” diferente desta linguagem*”<sup>12</sup>.

É importante lembrar que “*entre as implementações mais comuns, podemos citar: Hibernate da JBoss , EclipseLink da Eclipse Foundation e o OpenJPA da Apache.*”<sup>13</sup>

---

<sup>12</sup> Franco (2014, p.211)

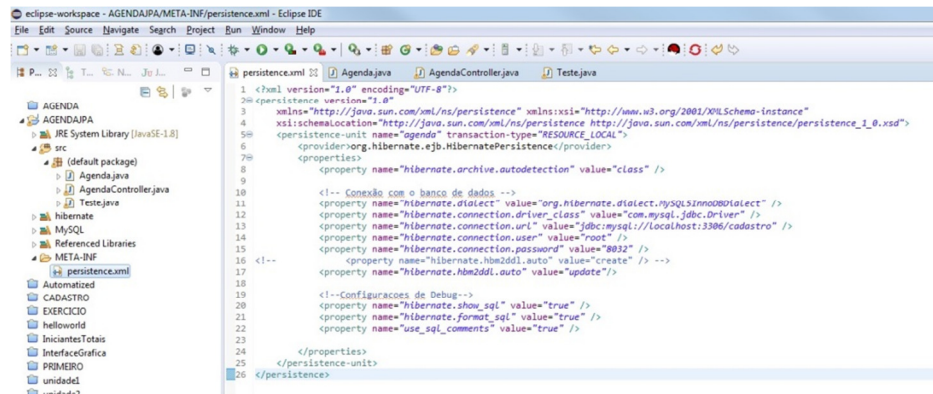
<sup>13</sup> Franco (2014, p.210)



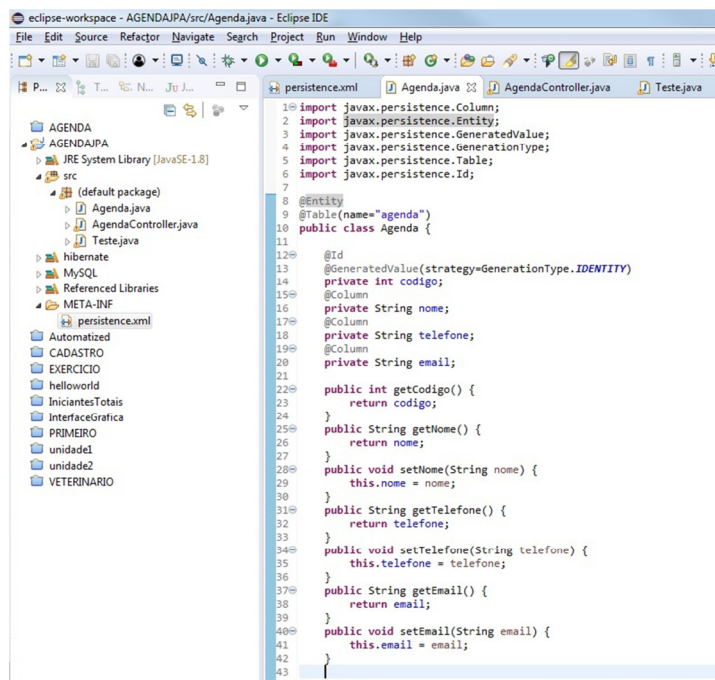
Após criar um novo projeto procede-se a adição das libs em <Build Path> e em <Add External Archives>, seleciona-se os arquivos e clicamos no botão <Abrir>.

Observando que iremos inserir os arquivos do Hibernate disponível no site <<https://sourceforge.net/projects/hibernate/files/hibernate-shards/>>

Após a inclusão das bibliotecas devemos criar um folder com o nome META-INF especificamente e dentro criar o arquivo persistence.xml que conterá as configurações de conexão com o banco de dados e o nome da unidade de persistência em nossa aplicação.

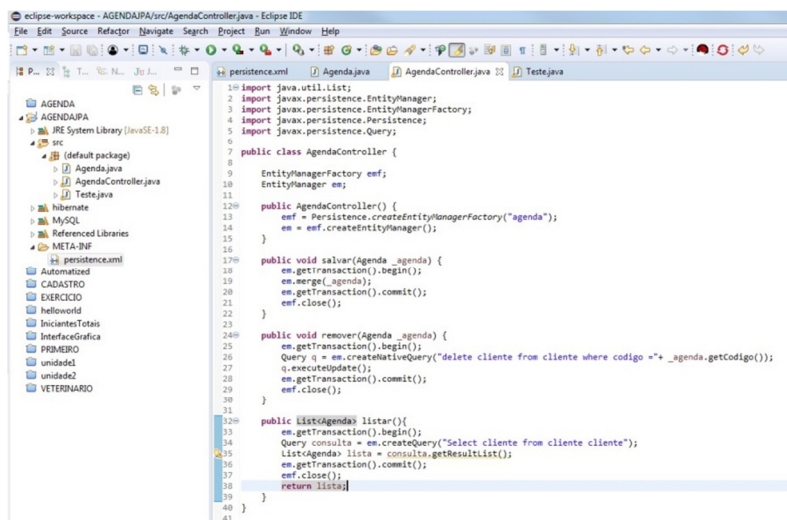


Criamos a classe Agenda instanciando através de @Entity a entidade e @Table a nossa tabela no banco de dados o @Id o campo da chave primaria que servira de índice e @Column os atributos da entidade, bem como a criação dos getters e setters, métodos utilizados para visualizar e acessar os atributos da classe.



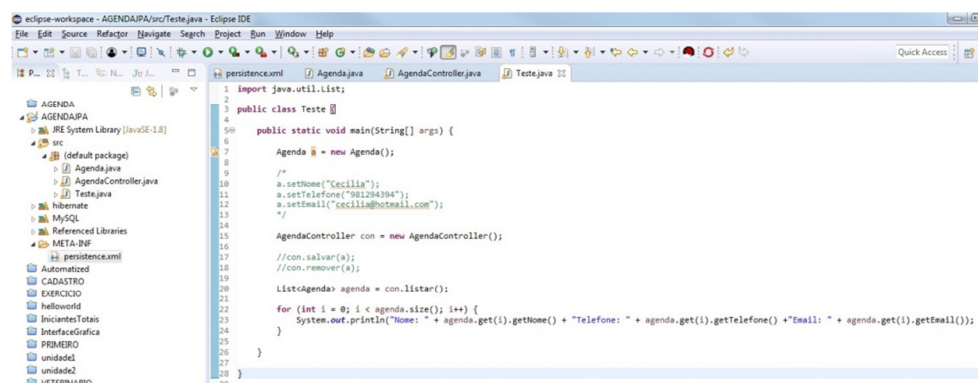


Criamos a classe de controle com os métodos para inserir, excluir e listar os dados da entidade “agenda”.



```
1 import java.util.List;
2 import javax.persistence.EntityManager;
3 import javax.persistence.EntityManagerFactory;
4 import javax.persistence.Persistence;
5 import javax.persistence.Query;
6
7 public class AgendaController {
8
9     EntityManagerFactory emf;
10    EntityManager em;
11
12    public AgendaController() {
13        emf = Persistence.createEntityManagerFactory("agenda");
14        em = emf.createEntityManager();
15    }
16
17    public void salvar(Agenda agenda) {
18        em.getTransaction().begin();
19        em.merge(agenda);
20        em.getTransaction().commit();
21        em.close();
22    }
23
24    public void remover(Agenda agenda) {
25        em.getTransaction().begin();
26        Query q = em.createQuery("delete cliente from cliente where codigo = " + agenda.getCodigo());
27        q.executeUpdate();
28        em.getTransaction().commit();
29        em.close();
30    }
31
32    public List<Agenda> listar() {
33        em.getTransaction().begin();
34        Query consulta = em.createQuery("Select cliente from cliente cliente");
35        List<Agenda> lista = consulta.getResultList();
36        em.getTransaction().commit();
37        em.close();
38        return lista;
39    }
40 }
41
```

Por fim criamos a classe teste.



```
1 import java.util.List;
2
3 public class Teste {
4
5     public static void main(String[] args) {
6
7         Agenda a = new Agenda();
8
9         /*
10        a.setNome("Cecilia");
11        a.setTelefone("9112345678");
12        a.setEmail("cecilia@hotmail.com");
13        */
14
15        AgendaController con = new AgendaController();
16
17        //con.salvar(a);
18        //con.remover(a);
19
20        List<Agenda> agenda = con.listar();
21
22        for (int i = 0; i < agenda.size(); i++) {
23            System.out.println("Nome: " + agenda.get(i).getNome() + " Telefone: " + agenda.get(i).getTelefone() + " Email: " + agenda.get(i).getEmail());
24        }
25    }
26 }
27
28
29
30
```

## 7. CONCLUSÃO

A utilização da API JDBC é necessária para a conexão com o banco de dado relacional, bem como sua utilização na linguagem JAVA, porém com a API JPA e suas implementações, um novo paradigma surge no desenvolvimento JAVA, e o trato com as instruções SQL fica com a responsabilidade da API, trazendo uma maior flexibilidade e agilidade ao programador, diminuindo custos e aumentando lucros.

## 8. REFERÊNCIAS

FRANCO, CRISTIANO ROBERTO – Programação Orientada a Objetos – UNIASSELVI – 2014 – 222p.

Aléssio; Simone Cristina – Processos de Software – UNIASSELVI – 2017 – 235p.

<<http://tiinside.com.br/tiinside/services/12/04/2016/ranking-de-sistemas-de-bancos-de-dados-mais-usados-em-20152016/> - acessado em 01/12/2018 – 15:53>

*<https://db-engines.com/en/ranking - acessado em 01/12/2018 - 11:03>*

*<https://www.youtube.com/watch?v=Frmwdter-vQ - acessado em 01/12/2018 - 14:10>*

*<https://www.todamateria.com.br/teoria-dos-conjuntos/ - acessado em 01/12/2018 - 16:00>*

*<https://www.youtube.com/watch?v=FyqIxyUu8cE - Instalando Wildfly no eclipse. acessado em 02/12/2018 - 17:00>*

*<https://www.mysql.com/products/connector/ - acessado em 03/12/2018 09:47>*

*<https://www.youtube.com/watch?v=80tZQFicXn4 - acessado em 04/12/2018 22:34>*

*< https://www.youtube.com/watch?v=AD-gND0Qkeo - acessado em 06/12/2018 - 22:30>*

*<https://portaldoalunoead.uniasselvi.com.br/extranet/layout/request/direciona\_materia\_l\_aluno\_05.php/atividadepratica\_prog\_java\_web.pdf>*