Disciplina de Desenvolvimento Web III

PROFESSOR JEFFERSON CHAVES

jefferson.chaves@ifpr.edu.br

ROTEIRO DE AULA

PERSISTÊNCIA DE DADOS

Objetivo geral:

- Conhecer os principais recursos para acesso a um bancos de dados (teoria e prática);
- Elaborar a estrutura básica de um projeto;
- Implementar o padrão Repository

Ferramentas necessárias

- [ ] Instalar o MySQL Server;
- [ ] MySQL Workbench

Preparação do projeto

- [ ] Usando o Workbench, crie uma base de dados "ifpr_store";
- [ ] Popule seu banco de dados usando este script;
- [ ] Certifique-se que seu base funciona. realize consultas e inserções;
- [ ] Crie um novo Projeto Java Web;
- [ ] Inclua a biblioteca MySQL Connector ao projeto (POM.xml)

```
<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <version>8.0.33</version>
</dependency>
```

- [ ] Crie um pacote chamado connection;
- [ ] No pacote "connection", crie uma classe ConnectionFactory;
- [ ] Crie um pacote chamado exceptions;

- [ ] No pacote "exceptions", criar uma exceção personalizada DatabaseException
- [ ] Crie um pacote chamado repository;
- [ ] Neste pacote crie uma classe chamada SellerRepository
- [ ] Crie uma outra classe chamada DepartmentRepository
- [ ] Obter e fechar uma conexão com o banco

Operação 1: Fábrica de Conexões

Material de apoio: ConnectionFactory.java

Ao utilizar o JDBC, devemos passar os dados necessários para fazer a conexão com o banco, tais como usuário, senha e nome do banco e as instruções SQL para realizar as operações na base, entre elas, inclusão, alteração, pesquisa ou exclusão de registros.

Uma das formas de se fazer uma conexão é criar um método para retornar o resultado dessa operação. A vantagem de criar um método para fazer a conexão é que poderemos reutilizá-lo outras vezes na aplicação. Assim, atendemos aos princípios da programação orientada a objeto de reaproveitamento de código.

Confira o método, a seguir:

```
public class ConnectionFactory {

    public Connection getConnection() {
        try {
            DriverManager.registerDriver(new com.mysql.jdbc.Driver());
            return DriverManager.getConnection( url: "jdbc:mysql://localhost/agenda", user: "root", password: "root");

        } catch(SQLException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }
}
```

No algoritmo acima, o método getConnection realiza a conexão com o banco de dados e retorna um objeto do tipo Connection, que representa e contém os dados dessa conexão. O método getConnection deverá ser utilizado sempre que uma conexão com o banco de dados for necessária.

Operação 2: recuperar registros

Material de apoio: Seller.java, SellerReposotory.java

Para este exemplo, vamos listar todos os vendedores persistidos na tabela [seller] do nosso banco de dados. Para isso, precisamos criar uma classe chamada Seller, que fará a representação do modelo de "Seller" orientado a objetos em nossa aplicação. Veja o código, a seguir:

models/Seller.java

```
public class Seller {

    private String Id;
    private String name;
    private String email;
    private LocalDate birthDate;
    private Double BaseSalary;
    //private Department department;


    //getters e setters para cada atributo
}
```

Criada a classe de modelo, estamos prontos para criar nossa classe de acesso ao banco de dados (código java):

```
public class SellerRepository {

    private final Connection conn;

    public SellerRepository(){
        ConnectionFactory connectionFactory = new ConnectionFactory();
        conn = connectionFactory.getConnection();
    }

    public void getSellers(){

        Statement statement = null;
        ResultSet resultSet = null;

        try {
            statement = conn.createStatement();
            resultSet = statement.executeQuery( s: "SELECT * FROM seller");

            while (resultSet.next()){
                System.out.println(resultSet.getInt( s: "Id") + " " + resultSet.getString( s: "Name"));
            }

        } catch (SQLException e) {...} finally {

        }
    }
}
```

Perceba que chamamos a função getConnection para estabelecer a conexão com o banco de dados e armazenamos o resultado no atributo “conn”. A seguir, utilizamos o objeto Statement para realizar a consulta ao banco de dados com o comando SQL desejado.

```
statement.executeQuery("SELECT * FROM seller");
```

Depois, utilizamos o objeto ResultSet para armazenar o resultado do comando realizado no banco de dados. Veja o comando:

```
resultSet = statement.executeQuery("SELECT * FROM seller");
```

Na linha seguinte, usamos um laço while para ler o resultado retornado do banco (armazenado no formato intermediário ResultSet) e exibi-lo no console.

```
while (resultSet.next()){
   System.out.println(resultSet.getInt("Id") + " " + resultSet.getString("Name"));
}
```

Um olhar sobre a API JDBC:

- Statement:
  - Classe Statement;
- ResultSet
  - Possui métodos para se posicionar na tabela de resultados;
  - método first() → move para posição 1, se houver;
  - método beforeFirst() → move para posição 0;
  - método next() → move para o próximo, retorna false se já estiver no último;
  - método absolute(int) → move para a posição dada, lembrando que dados reais começam em 1, não em 0.

posição	id	Name
1	1	Computers
2	2	Eletronics
3	3	Fashion
4	4	Books

Atenção: objetos da classe **ResultSet** apresentam dados em um formato intermediário, armazenando dados na forma de tabela!

Atividade:

- Implemente o método toString na classe Seller;
- Faça com que o método getSellers retorne uma lista de vendedores: List<Seller>;
- Na classe ConnectionFactory, crie métodos auxiliares estáticos para fechar ResultSet e Statement (opcional);
- Implemente o código necessário para recuperar os dados dos

departamentos;

Operação 3: inserir registros

Material de apoio: SellerRepository (insert), SellerRepository (insert with keys)

A próxima atividade é inserir registros com informações de sellers, programaticamente no banco de dados. Para tanto utilizaremos as classes métodos da API JDBC:

- Classe PreparedStatement
- Método executeUpdate()
- Statement.RETURN_GENERATED_KEYS
- getGeneratedKeys

Atividade:

- Realize uma Inserção simples com preparedStatement;
- Refatore sua aplicação para retornar o objeto com com o Id do banco de dados;
- Implemente o código necessário para inserção de departamentos;

Qual a diferença entre o Statement e o PreparedStatement?

A diferença vai além da simples adição de parâmetros.

A maioria dos bancos de dados relacionais lida com uma consulta (query) JDBC / SQL em quatro passos:

1 - Interpretar (parse) a consulta SQL;
2 - Compilar a consulta SQL;
3 - Planejar e otimizar o caminho de busca dos dados;
4 - Executar a consulta otimizada, buscando e retornando os dados.

Um Statement irá sempre passar pelos quatro passos acima para cada consulta SQL enviada para o banco.

Já um Prepared Statement pré-executa os passos (1) a (3). Isso significa

que, ao criar um Prepared Statement, pré-otimizações são feitas de imediato. O efeito disso é que, se você pretende executar a mesma consulta repetidas vezes mudando apenas os parâmetros de cada uma, a execução usando Prepared Statements será mais rápida e com menos carga sobre o banco.

Outra vantagem dos Prepared Statements é que, se utilizados corretamente, ajudam a evitar ataques de Injeção de SQL. Note que para isso é preciso que os parâmetros da consulta sejam atribuídos através dos métodos setInt(), setString(), etc. presentes na interface PreparedStatement e não por concatenação de strings.

Para consultas que não requerem nenhum parâmetro e que serão executadas poucas vezes, um Statement é suficiente. Para os demais casos, prefira PreparedStatement.

Operação 4: atualizar registros

Material de apoio: SellerRepository (update)

Atividade:

- Implemente o código necessário para atualização de registros de departamentos;

Operação 5: deletar registros

Material de apoio: SellerRepository (delete)

Problema com integridade referencial: Integridade referencial é uma propriedade de um banco de dados que garante que as relações entre as tabelas sejam mantidas, evitando dados inconsistentes ou inválidos. Quando um banco de dados possui integridade referencial, ele garante que uma chave estrangeira em uma tabela sempre se refere a uma chave primária correspondente em outra tabela.

A integridade referencial é importante porque ajuda a garantir a consistência e a precisão dos dados em um banco de dados. Sem ela, os dados podem ficar desatualizados ou inconsistentes, o que pode levar a erros e problemas de integridade. Portanto, é uma boa prática implementar integridade referencial em um banco de dados sempre que possível.

Atividades:

- No pacote exception crie uma nova exceção chamada DatabaseIntegrityException;
- Tratar a exceção de integridade referencial;
- Implemente o código necessário para exclusão de registros de departamentos;

Padrão de projeto Repository

Colocar código SQL dentro de suas classes de domínio ou negócio é algo nem um pouco elegante e muito menos viável quando você precisa manter o seu código;

A ideia dos padrões Repository (e outros como o Data Access Object (DAO)) é separar o código de acesso ao banco de dados de suas classes de lógica e colocá-los em uma camada responsável pelo acesso aos dados. Dessa forma o código de acesso ao banco de dados fica em um lugar só, tornando mais fácil a manutenção.

Para cada entidade, deverá haver (mas nem sempre) um objeto responsável por fazer acesso a dados relacionados a esta entidade. Por exemplo:

- Seller → SellerRepository;
- Department →DepartmentRepository

Cada Repository será definido por uma interface. A injeção de dependência pode ser feita por meio do padrão de projeto Factory.

Operação 6: FindById

Consulta SQL:

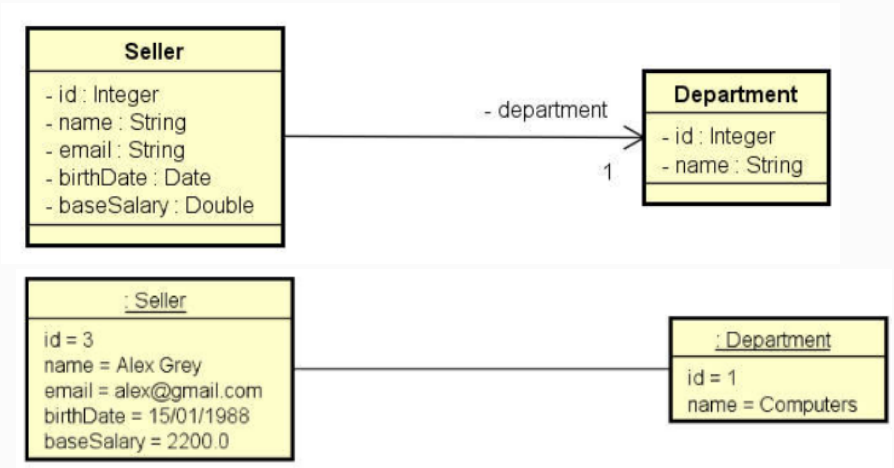
```
SELECT seller.*,department.Name as DepName
FROM seller INNER JOIN department
ON seller.DepartmentId = department.Id
WHERE seller.Id = ?
```

ResultSet(formato de tabela):

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	Id	Name	Email	BirthDate	BaseSalary	DepartmentId	DepName
1	1	Bob Brown	bob@gmail.com	1998-04-21 00:00:00	1000	1	Computers

Associated Objects



Refatoração: Reusing Instantiation

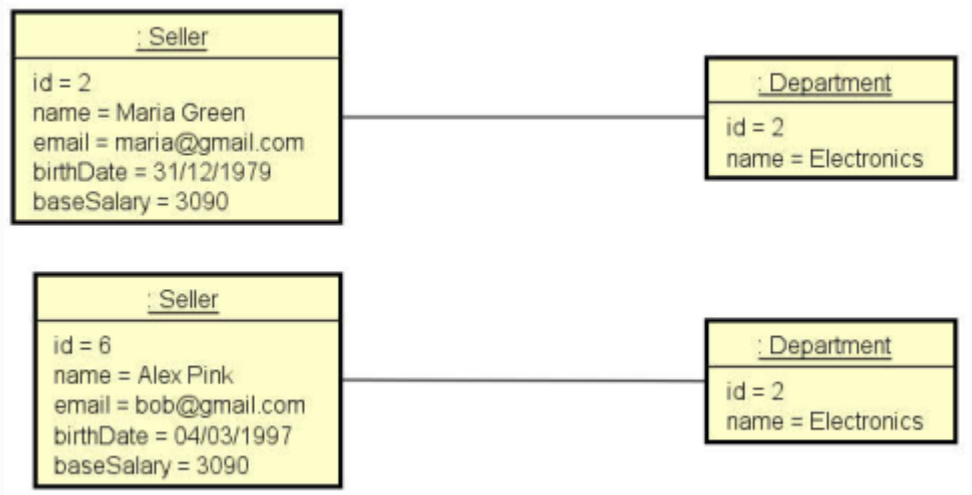
```
private Seller instantiateSeller(ResultSet rs, Department dep) throws SQLException {
	Seller obj = new Seller();
	obj.setId(rs.getInt("Id"));
	obj.setName(rs.getString("Name"));
	obj.setEmail(rs.getString("Email"));
	obj.setBaseSalary(rs.getDouble("BaseSalary"));
	obj.setBirthDate(rs.getDate("BirthDate"));
	obj.setDepartment(dep);
	return obj;
}

private Department instantiateDepartment(ResultSet rs) throws SQLException {
	Department dep = new Department();
	dep.setId(rs.getInt("DepartmentId"));
	dep.setName(rs.getString("DepName"));
	return dep;
}
```

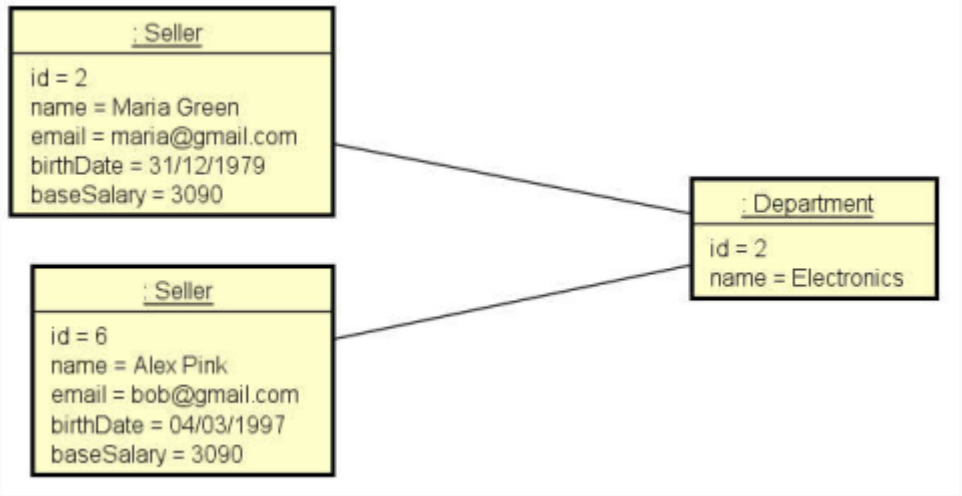
Operação 7: findByDepartment

Consulta SQL:

```
SELECT seller.*,department.Name as DepName
FROM seller INNER JOIN department
ON seller.DepartmentId = department.Id
WHERE DepartmentId = ?
ORDER BY Name
```



Errado


: Seller
id = 2
name = Maria Green
email = maria@gmail.com
birthDate = 31/12/1979
baseSalary = 3090
: Seller
id = 6
name = Alex Pink
email = bob@gmail.com
birthDate = 04/03/1997
baseSalary = 3090
: Department
id = 2
name = Electronics

Correto