

ECM251 - Linguagens de Programação I

15 - Introdução ao JDBC e SQLite

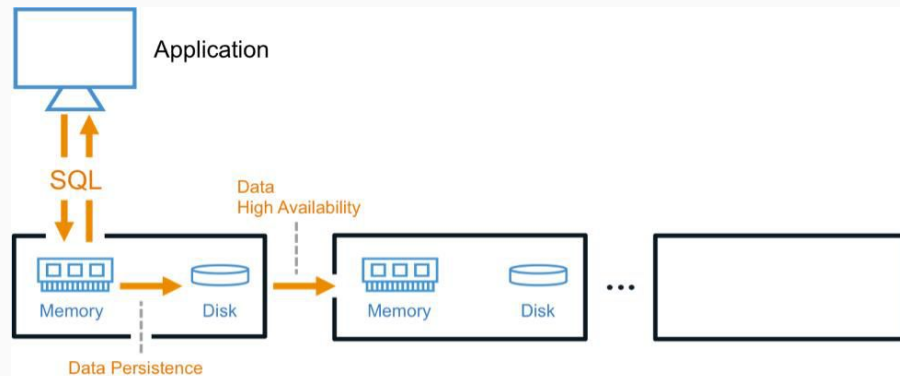
Prof. Murilo Zanini de Carvalho

Prof. Tiago Sanches da Silva

Persistência de Dados

Persistência de dados consiste em armazenar um conjunto de dados para uso posterior por um conjunto de aplicações.

Idealmente, ao persistir um conjunto de dados, estes devem possuir grande disponibilidade, garantia de integridade e possibilidade de acesso por múltiplas instâncias



Retirado de (
https://www.safaribooksonline.com/library/view/building_realtimedata/9781491975879/assets/brtd_0901.png), em
02/09/2018

Persistência de Dados

Uma das formas de manipular esses dados é utilizando algum banco de dados.



Retirado de
(https://zhanglit ing.github.io/images/main_.jpg), em
02/09/2018

Conceitos Básicos de Banco de Dados

O SQLite permite utilizar comandos SQL em um arquivo de texto (base de dados).

Possibilita implementar algumas funcionalidades sem a presença de um servidor de dados.



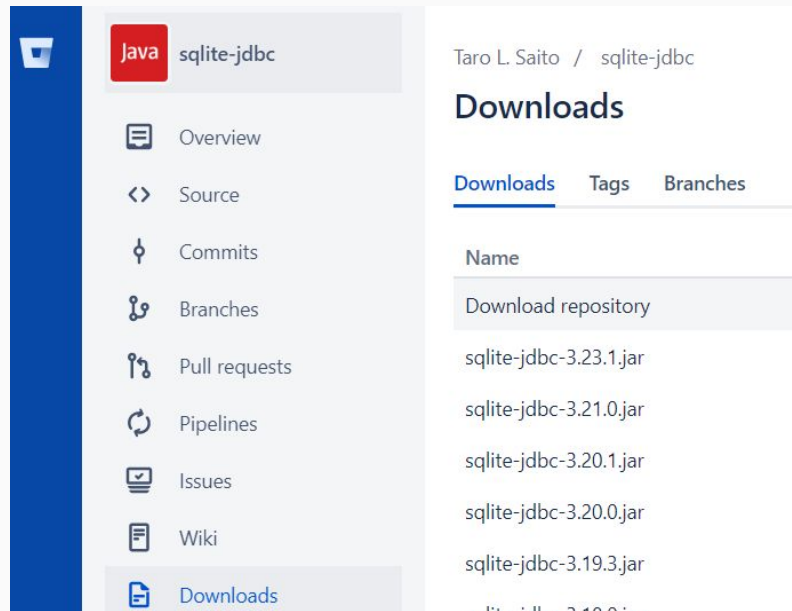
Retirado de
<http://www.sqlitetutorial.net/wpcontent/uploads/2015/12/SQLiteJava.jpg> , em
02/09/2018

Conceitos Básicos de Banco de Dados

Para conectar a um banco de dados SQLite ou qualquer outro utilizando JAVA, é necessário trazer o conector da JDBC ao projeto. Mais sobre conectores ainda nessa aula

Conector SQLite:

<https://bitbucket.org/xerial/sqlite-jdbc/downloads/>



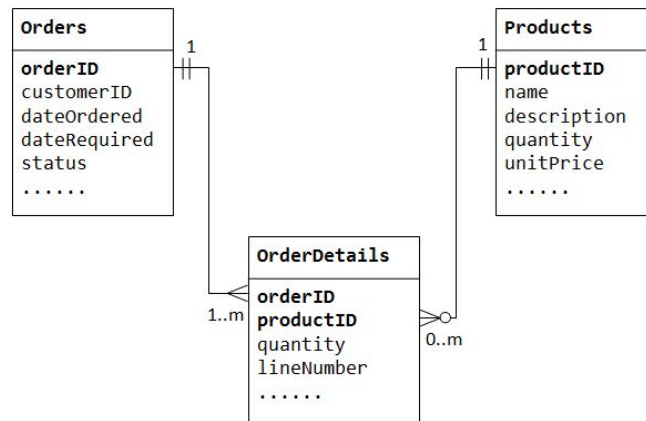
The screenshot shows the Bitbucket web interface for the 'sqlite-jdbc' repository. On the left is a dark blue sidebar with icons for Overview, Source, Commits, Branches, Pull requests, Pipelines, Issues, Wiki, and Downloads. The main content area has a header with the repository name 'sqlite-jdbc' and the owner 'Taro L. Saito'. Below the header is a navigation bar with 'Downloads', 'Tags', and 'Branches'. The 'Downloads' tab is active, showing a table of available JAR files for download.

Name
Download repository
sqlite-jdbc-3.23.1.jar
sqlite-jdbc-3.21.0.jar
sqlite-jdbc-3.20.1.jar
sqlite-jdbc-3.20.0.jar
sqlite-jdbc-3.19.3.jar

Conceitos Básicos de Banco de Dados

Em geral, os bancos precisam ser modelados para ser utilizados.

Em um SGBD, podem existir diversos bancos de dados, cada um deles com diversas tabelas de dados com as informações referentes a aplicação desenvolvida.



Retirado de
(https://instanteduhelp.com/wp-content/uploads/2016/03/many_to_many.png), em 25/08/2019

Conceitos Básicos de Banco de Dados

A criação das tabelas que serão utilizadas em um banco vem no momento anterior ao seu projeto de uso.

As tabelas podem ser criadas em código, mas em geral, elas são criadas utilizando alguma ferramenta que permita a modelagem de suas interações.

Conceitos Básicos de Banco de Dados

Para a construção do banco que será utilizado, podemos utilizar um editor de SQLite ou fazer sua criação diretamente no código.

Para utilizar o editor, vamos utilizar o DB Browser for SQLite.

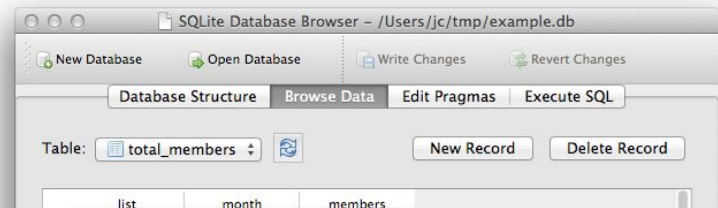
<https://sqlitebrowser.org/dl/>

[About](#)[Download](#)[Blog](#)[Docs](#)[GitHub](#)[Gitter](#)[Slack](#)[Stats](#)[T](#)

DB Browser for SQLite

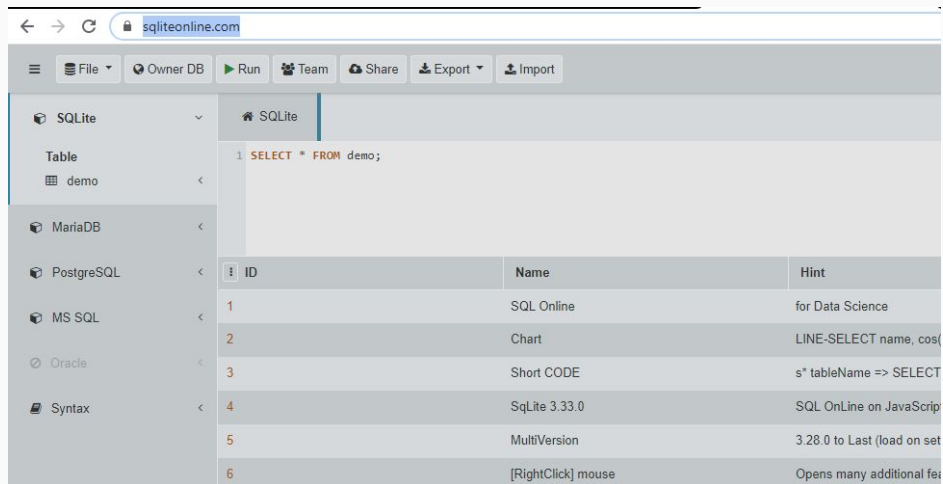
The Official home of the DB Browser for SQLite

Screenshot



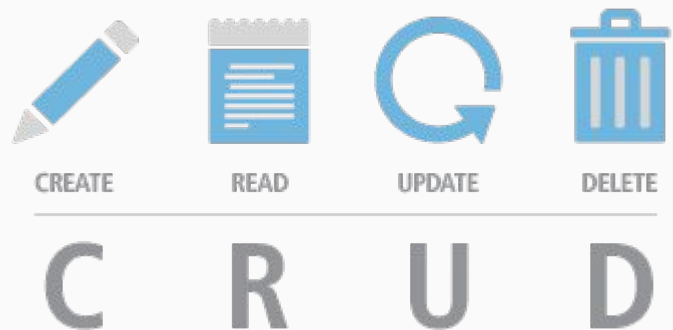
Conceitos Básicos de Banco de Dados

Algumas ferramentas online também podem ser utilizadas para construir os arquivos de banco de dados, com o: <https://sqliteonline.com/>



Conceitos Básicos de Banco de Dados

As operações básicas que costumam ser realizadas com um banco de dados são chamadas de CRUD.



Retirado de
(<https://www.luiztools.com.br/wp-content/uploads/2017/07/CRUD.png>), em 23/08/2020

Conceitos Básicos de Banco de Dados

- CREATE: criar novos registros (entradas) no banco.
- REQUEST: buscar informações que já estão no banco.
- UPDATE: atualizar os registros (entradas) presentes dentro do banco.
- DELETE: apagar alguma das entrada no banco.

RELEMBRAR É VIVER:

Operações com SQL

Para manipular os dados com banco de dados, em geral, operações com SQL devem ser realizadas.

Alguns dos comandos básicos de SQL serão apresentados a seguir, para estudar eles com mais profundidade, verificar:

- <https://www.w3schools.com/sql/default.asp>
- <https://www.kaggle.com/learn/intro-to-sql>
- <https://www.kaggle.com/learn/advanced-sql>

CODE ON!

Projeto 01



Interface CLI

(Retirado de
(https://www.clipartmax.com/png/middle/131-1314974_robin-classic-teen-titans-go-png-by-whitej2-robin-teen-titans-go.png), em 16/08/2020)

Criar um banco de dados (Arquivo DB)

- Criar um banco de dados para armazenar os dados de vendas de uma loja de peças de componentes de computador.
- A loja deve possuir dados sobre os seus produtos, podendo cadastrar novos produtos, consultar os que já estão ali, realizar vendas, modificar preços e até mesmo excluir algum produto que foi cadastrado de forma incorreta.
- Os registros devem conter:
 - Código de barras do produto
 - Nome do produto
 - Breve descrição do produto
 - Valor de compra (custo do produto)
 - Valor para venda (custo do produto na loja)
 - Quantidade no estoque

JDBC

Persistência de Dados com Java

Conectar-se a um banco de dados com Java é feito de maneira elegante.

Para evitar que cada banco tenha a sua própria API e conjunto de classes e métodos, temos um único conjunto de interfaces muito bem definidas que devem ser implementadas.

Esse conjunto de interfaces fica dentro do pacote `java.sql` e nos referimos à ela como JDBC.

Java DataBase Connectivity

Persistência de Dados com Java

Interfaces java.sql:

<https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html>

JDBC API:

<https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>

O que é o JDBC?

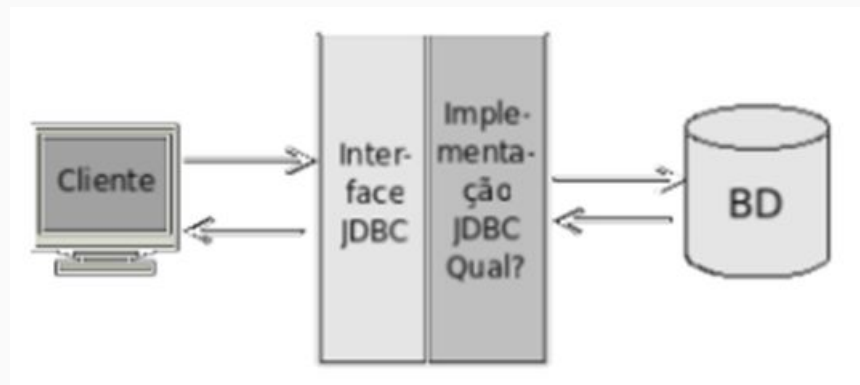
Pode-se dizer que é uma API (Interface de Programação de Aplicativos) que reúne conjuntos de classes e interfaces escritas na linguagem Java na qual possibilita se conectar através de um driver específico do banco de dados desejado. Com esse driver pode-se executar instruções SQL de qualquer tipo de banco de dados relacional.

Para fazer a comunicação entre a aplicação e o Banco de Dados é necessário possuir um driver para a conexão desejada. Geralmente, as empresas de Banco de Dados oferecem o driver de conexão que seguem a especificação JDBC.

Driver?

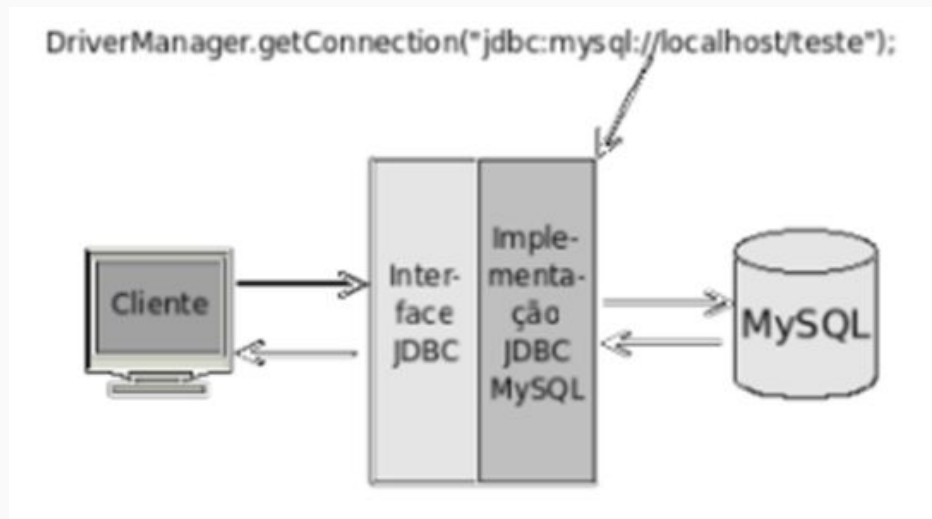
Caso queiramos trabalhar com o SQLite ou o MySQL, precisamos de classes concretas que implementem essas interfaces do pacote java.sql.

Esse conjunto de classes concretas é quem fará a ponte entre o código cliente que usa a API JDBC e o banco de dados. São essas classes que sabem se comunicar através do protocolo proprietário do banco de dados. Esse conjunto de classes recebe o nome de driver.



Driver?

Todos os principais bancos de dados do mercado possuem drivers JDBC para que você possa utilizá-los com Java.



ATENÇÃO

Importe do java.sql

Existe um ponto de atenção na importação das classes ou interfaces relacionadas ao pacote a ser usado no momento do desenvolvimento.

A correta a importação do pacote referente à classe Connection pertencente ao pacote java.sql.

Esse é um fator a ser observado com cautela, pois isso é considerado um dos erros mais comuns justamente pelo fato do desenvolvedor pensar muitas vezes em usar o pacote com.mysql.jdbc sendo que está utilizando o driver JDBC do banco MySQL.

Esse pacote oferece a biblioteca Java o acesso e processamento de dados em uma banco de dados. As classes e interfaces mais importantes são:

Classe	Interface
DriverManager	Driver
	Connection
	Statement
	ResultSet
	PreparedStatement

DriverManager

Para abrir uma conexão com um banco de dados, precisamos utilizar sempre um driver. A classe DriverManager é a responsável por se comunicar com todos os drivers que você deixou disponível.

Para isso, invocamos o método estático getConnection com uma String que indica a qual banco desejamos nos conectar.

Essa String - chamada de String de conexão JDBC - que utilizaremos para acessar o MySQL tem sempre a seguinte forma:

```
jdbc:mysql://ip/nome_do_database
```


Mas eu vi em algum tutorial...

E o `Class.forName()`?

Até a versão 3 do JDBC, antes de chamar o `DriverManager.getConnection()` era necessário registrar o driver JDBC que iria ser utilizado através do método `Class.forName("com.mysql.jdbc.Driver")`, no caso do MySQL, que carregava essa classe, e essa se comunicava com o `DriverManager`.

A partir do JDBC 4, que está presente no Java 6, esse passo não é mais necessário. Mas lembre-se: caso você utilize JDBC em um projeto com Java 5 ou anterior, será preciso fazer o registro do Driver JDBC, carregando a sua classe, que vai se registrar no `DriverManager`.

Interface Connection

Representa uma conexão ao banco de dados. Nessa interface são apresentados os métodos mais utilizados.

Caso o DriverManager consiga realizar a conexão com o banco de dados ele retorna uma instância de um objeto Connection.

Com ele você conseguirá executar queries.

CODE ON!

Projeto 01

Continuação



Interface CLI

(Retirado de
(https://www.clipartmax.com/png/middle/131-1314974_robin-classic-teen-titans-go-png-by-whitej2-robin-teen-titans-go.png), em 16/08/2020)

Ligando arquivo DB com o programa Java

- OK, o seu arquivo com o banco está criado, mas agora eu preciso utilizar ele!
- Criar um programa em Java que possibilite eu utilizar as funcionalidades que você propôs.
- NÃO ESQUECER DE BAIXAR O ARQUIVO *.db QUE FOI CRIADO NO PROJETO ANTERIOR.

Driver SQLite da JDBC

Verificar a documentação disponível em: <https://github.com/xerial/sqlite-jdbc>

Adicionar no arquivo pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.25.2</version>
  </dependency>
</dependencies>
```

Fluxo de Trabalho

1. Utilizando o DriverManager, criar uma Connection com o banco. Enviar como parâmetro uma connectionString:
 - a. jdbc:sqlite:banco.db
 - b. jdbc:sqlite:/caminho/para/o/banco.db
 - c. jdbc:sqlite::memory
2. Com a Connection criada, criar os comandos para enviar ao banco:
 - a. Statement
 - b. PreparedStatement
3. Receber um ResultSet com os resultados.

1. Utilizando o DriverManager, criar uma Connection com o banco. Enviar como parâmetro uma connectionString:
 - a. jdbc:sqlite:banco.db
 - b. jdbc:sqlite:/caminho/para/o/banco.db
 - c. jdbc:sqlite::memory

```
private Connection connection;  
  
public ProdutoDAO(String myConnectionString) {  
    try {  
        this.connection = DriverManager.getConnection(myConnectionString);  
    } catch (SQLException throwables) {  
        throwables.printStackTrace();  
        this.connection = null;  
    }  
}
```

2. Com a Connection criada, criar os comandos para enviar ao banco:
 - a. Statement
 - b. PreparedStatement

```
Statement comandoSQL = connection.createStatement();
ResultSet resultSet = comandoSQL.executeQuery( sql: "SELECT * FROM meus_produtos");
while(resultSet.next()) {
    //...
}
resultSet.close();
```


2. Com a Connection criada, criar os comandos para enviar ao banco:
 - a. Statement
 - b. PreparedStatement

```
PreparedStatement comandoSQL = connection.prepareStatement("SELECT * FROM meus_produtos");  
ResultSet resultSet = comandoSQL.executeQuery();  
while(resultSet.next()) {  
    //...  
}  
resultSet.close();
```

2. Com a Connection criada, criar os comandos para enviar ao banco:



Josias Martins Maceda (183.3k xp, 14 posts)

Compartilhando.

Comandos:

execute -> Executa qualquer tipo de instrução SQL, seja um SELECT, UPDATE, INSERT, DELETE;

executeQuery -> Executa uma instrução sql SELECT, no entanto diferente da execute onde era necessário executar o comando `statement.getResultSet()` ou `preparedStatement.getResultSet()` a mesma já retorna o `ResultSet`;

executeUpdate -> Executa operações como INSERT, UPDATE, DELETE, no entanto nesta operação temos como retorno o numero de linhas afetadas, não sendo necessário executar o comando `statement.getUpdateCount()`;

Compartilhem outros métodos interessantes. Se estiver equivocado em algo por favor me corrijam. :)

3. Receber um ResultSet com os resultados.

```
ResultSet resultSet = comandoSQL.executeQuery( sql: "SELECT * FROM meus_produtos");  
while(resultSet.next()){  
    Produto produto = new Produto(  
        resultSet.getString( columnName: "codigo"),  
        resultSet.getString( columnName: "nome"),  
        resultSet.getString( columnName: "descricao"),  
        resultSet.getDouble( columnName: "custo"),  
        resultSet.getDouble( columnName: "venda"),  
        resultSet.getInt( columnName: "quantidade")  
    );  
    produtos.add(produto);  
}  
resultSet.close();
```

Atualizando dados no banco

```
public void update(Produto produto) {  
    try{  
        PreparedStatement comandoSQL = connection.prepareStatement( sql: "UPDATE meus_produtos SET  
codigo = ?, nome = ?, descricao = ?, custo = ?, venda = ? , quantidade = ?, WHERE codigo = ?");  
        comandoSQL.setString( parameterIndex: 1, produto.getCodigo());  
        comandoSQL.setString( parameterIndex: 2, produto.getNome());  
        comandoSQL.setString( parameterIndex: 3, produto.getDescricao());  
        comandoSQL.setDouble( parameterIndex: 4, produto.getCusto());  
        comandoSQL.setDouble( parameterIndex: 5, produto.getVenda());  
        comandoSQL.setInt( parameterIndex: 6, produto.getQuantidade());  
        comandoSQL.setString( parameterIndex: 7, produto.getCodigo());  
        comandoSQL.executeUpdate();  
        connection.commit();  
    }catch (Exception e){  
        e.printStackTrace();  
    }  
}
```

Deletando dados no banco

```
@Override
public void delete(Produto produto) {
    try{
        PreparedStatement comandoSQL = connection.prepareStatement("DELETE FROM meus_produtos
WHERE codigo = ?");
        comandoSQL.setString( parameterIndex: 1, produto.getCodigo());
        comandoSQL.executeUpdate();
        connection.commit();
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

Inserindo dados no banco

```
@Override
public void add(Produto produto) {
    try{
        PreparedStatement comandoSQL = connection.prepareStatement(
            "INSERT INTO meus_produtos (codigo, nome, descricao, custo, venda, quantidade) VALUES (?,?,?,?,?,?)");
        comandoSQL.setString(1, produto.getCodigo());
        comandoSQL.setString(2, produto.getNome());
        comandoSQL.setString(3, produto.getDescricao());
        comandoSQL.setDouble(4, produto.getCusto());
        comandoSQL.setDouble(5, produto.getVenda());
        comandoSQL.setInt(6, produto.getQuantidade());
        comandoSQL.executeUpdate();
        connection.commit();
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

Pegando dados no banco

```
@Override
public Produto get(int index) {
    Produto produto = null;
    try{
        Statement comandoSQL = connection.createStatement();
        ResultSet resultSet = comandoSQL.executeQuery( sql: "SELECT * FROM meus_produtos");
        while(resultSet.next()){
            produto = new Produto(
                resultSet.getString( columnLabel: "codigo"),
                resultSet.getString( columnLabel: "nome"),
                resultSet.getString( columnLabel: "descricao"),
                resultSet.getDouble( columnLabel: "custo"),
                resultSet.getDouble( columnLabel: "venda"),
                resultSet.getInt( columnLabel: "quantidade")
            );
        }
        resultSet.close();
    } catch (Exception e){
        e.printStackTrace();
    }
    return produto;
}
```

ATENÇÃO: Codar e voltar aqui
depois!

Padrão de Projeto:

Data Access Object - DAO

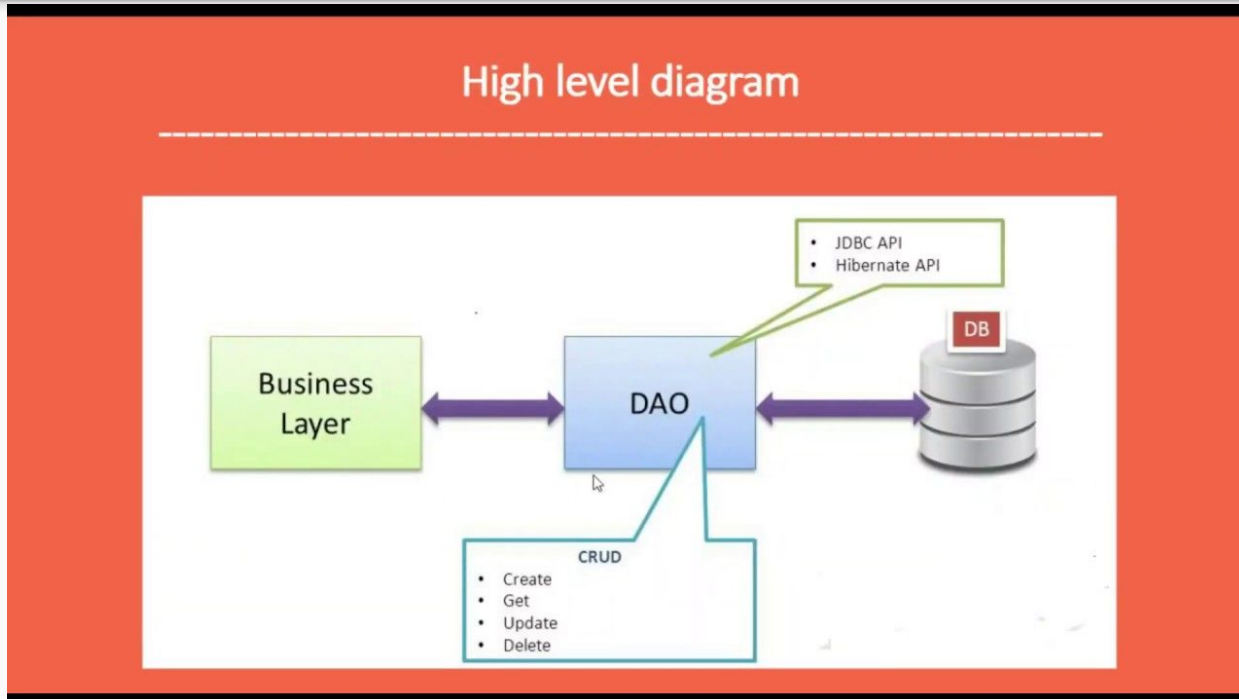
Possibilita separar o código de interface com o banco de dados da lógica do problema que está sendo implementada.

Para saber mais:

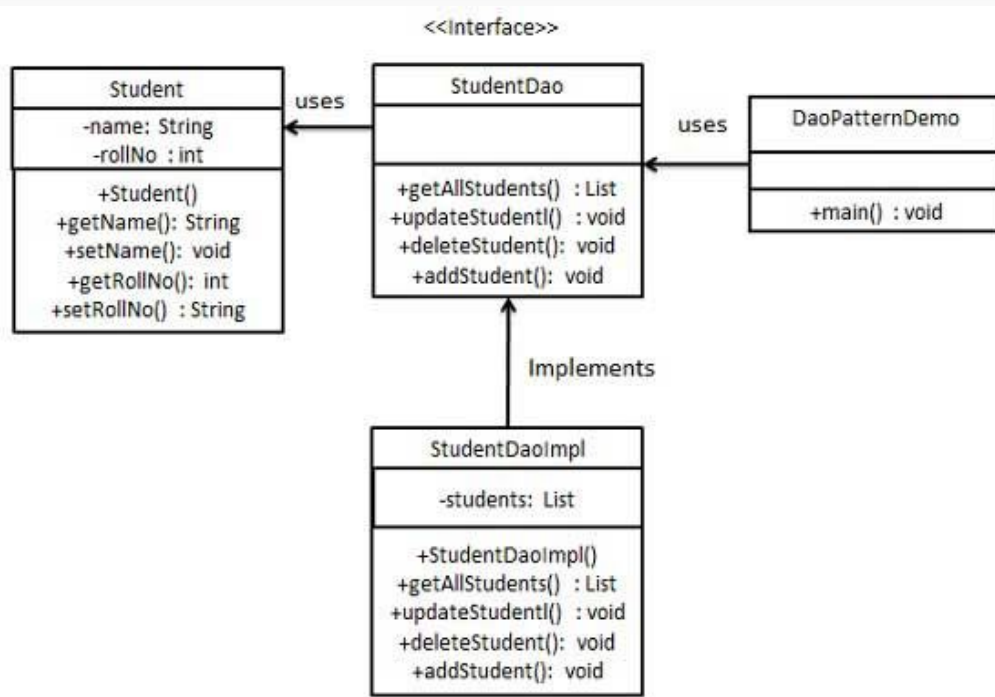
- https://en.wikipedia.org/wiki/Data_access_object
- https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm
- <https://www.baeldung.com/java-dao-pattern>

Padrão de Projeto: *Data Access Object - DAO*

Retirado de
(<https://i.ytimg.com/vi/ui01Li4vqDc/maxresdefault.jpg>)
, em 23/08/2020

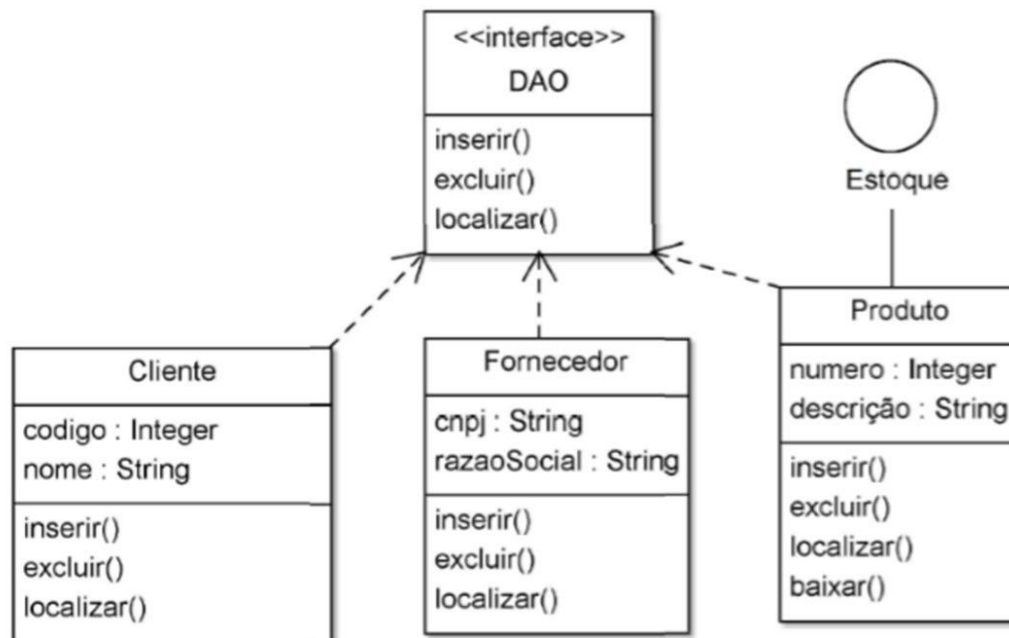


Padrão de Projeto: *Data Access Object - DAO*



Retirado de
(<https://rogeriofontesps.files.wordpress.com/2016/04/data-access-object-uml-diagram.png>), em
23/08/2020

Padrão de Projeto: *Data Access Object - DAO*



Refatorar e Continuar!

CODE ON!

Projeto 01

Final



Interface CLI

(Retirado de
(https://www.clipartmax.com/png/middle/131-1314974_robin-classic-teen-titans-go-png-by-whitej2-robin-teen-titans-go.png), em
16/08/2020)

- Adicionar agora as tabelas de usuarios, clientes e vendas.