

INSTITUTO MAUÁ DE TECNOLOGIA




Linguagens I

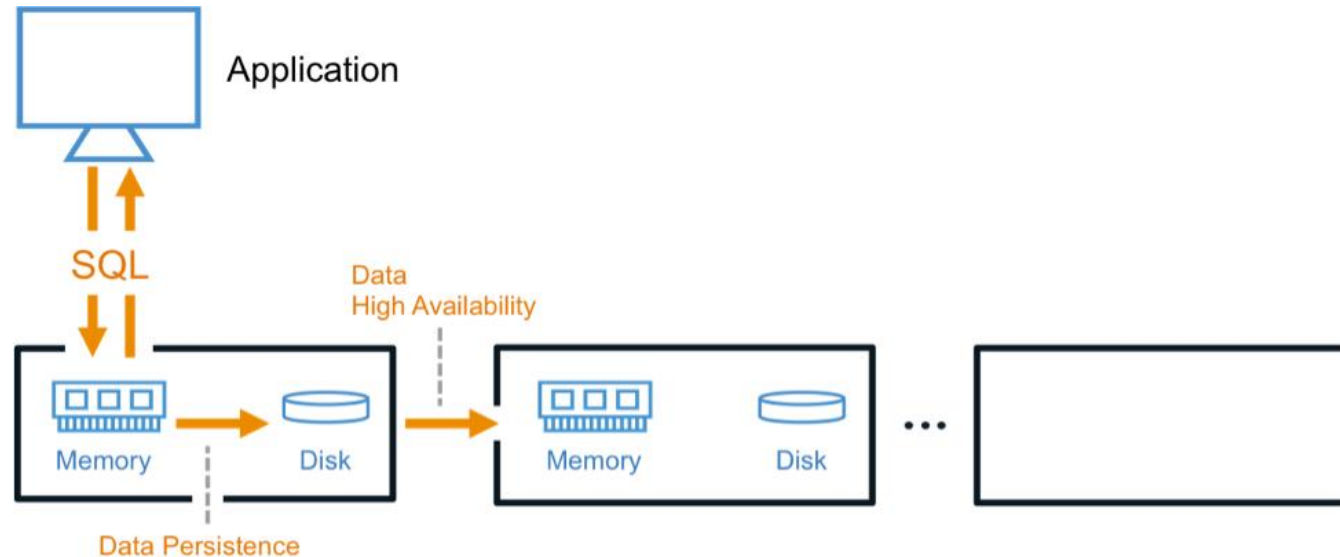
Java + Banco de Dados

Profº. Tiago Sanches da Silva
Prof. Murilo Zanini de Carvalho

Conceito Básico de Banco 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, esses devem possuir grande disponibilidade, garantia de integridade e possibilidade de acesso por múltiplas instâncias.
- 

Conceito Básico de Banco de Dados



Retirado de (https://www.safaribooksonline.com/library/view/building-real-time-data/9781491975879/assets/btrtd_0901.png), em 02/09/2018

Conceito Básico de Banco de Dados

- Uma das formas de manipular esses dados é utilizando bando de dados.



Retirado de (<https://i.ytimg.com/vi/etReM7odebE/maxresdefault.jpg>), em 02/09/2018

Retirado de (https://zhangliting.github.io/images/main_databases.jpg), em 02/09/2018

Conceito Básico 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

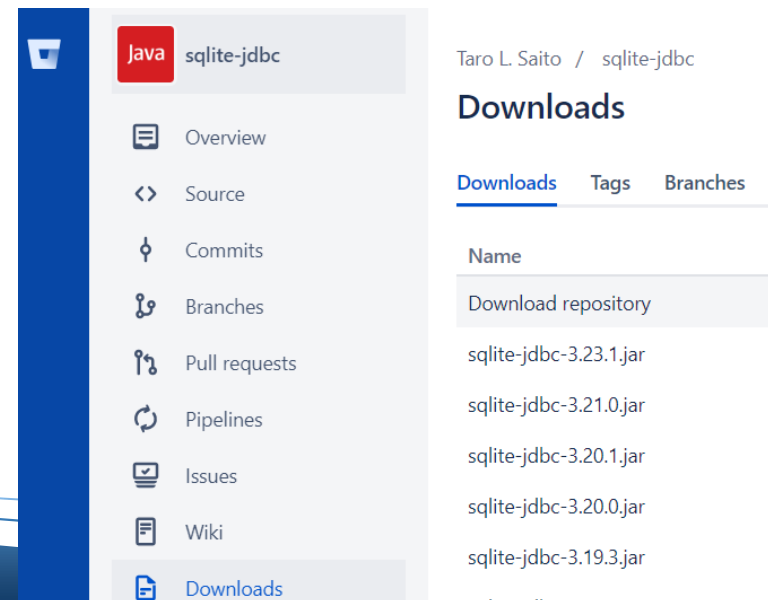
(<http://www.sqlitetutorial.net/wp-content/uploads/2015/12/SQLite-Java.jpg>), em 02/09/2018



Conceito Básico 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 repository page for the 'sqlite-jdbc' project. The repository is owned by 'Taro L. Saito'. The 'Downloads' tab is selected, showing a list of available JAR files for download. The left sidebar contains navigation links for Overview, Source, Commits, Branches, Pull requests, Pipelines, Issues, Wiki, and Downloads.

sqlite-jdbc		
Taro L. Saito / sqlite-jdbc		
Downloads		
Downloads	Tags	Branches
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		

Conceito Básico de Banco de Dados

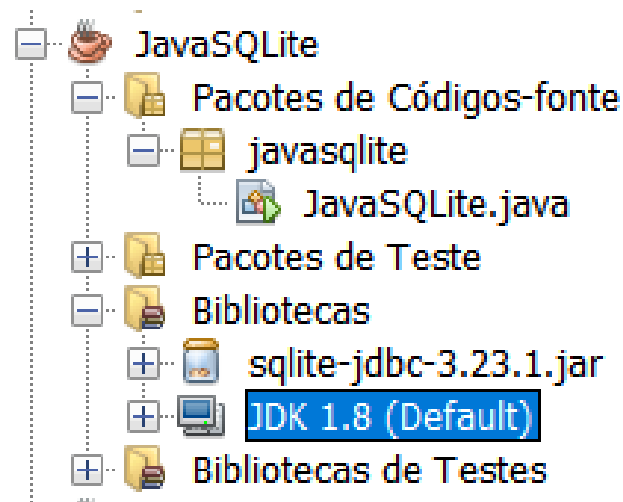
- Em geral, os bancos precisam ser modelados para ser utilizados.
 - Em um SGBD, podem existir diversos bancos de dados, cada um deles com diversas tabelas de dados com as informações referentes a aplicação desenvolvida.
- 

Conceito Básico 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.
- 

Conceito Básico de Banco de Dados

- Vamos criar as tabelas utilizando um projeto com o SQLite (comparar com o processo de criação utilizando o MySQL).
- Criar um novo projeto e adicionar o JAR na bibliotecas dele.



Conceito Básico de Banco de Dados

- Vamos colocar o código para criar uma tabela e rodar ele apenas uma vez, depois vamos manipular essa tabela.

```
public static void main(String[] args) {  
    Connection c = null;  
    Statement stmt = null;  
    try {  
        //SQLite  
        c = DriverManager.getConnection("jdbc:sqlite:test.db");  
  
        System.out.println("Conexão realizada com sucesso!");  
  
        stmt = c.createStatement();  
        String sql = "CREATE TABLE IF NOT EXISTS estoque " +  
            "(ID INT PRIMARY_KEY    AUTO_INCREMENT NOT NULL, " +  
            " NAME                    TEXT      NOT NULL, " +  
            " PRICE                   REAL      NOT NULL, " +  
            " COUNT                   INT);";  
        stmt.executeUpdate(sql);  
        stmt.close();  
        c.close();  
    } catch (Exception e) {  
        System.err.println( e.getClass().getName() + ": " + e.getMessage() );  
        System.exit(0);  
    }  
    System.out.println("Table created successfully");  
}
```

Conceito Básico de Banco de Dados



- Operações mais comuns de serem realizadas com um banco de dados.

Conceito Básico de Banco de Dados

- Ajustar o código para inserir alguns itens na tabela do banco.

```
public static void main(String[] args) {  
    Connection c = null;  
    Statement stmt = null;  
    try {  
        //SQLite  
        c = DriverManager.getConnection("jdbc:sqlite:test.db");  
        c.setAutoCommit(false);  
  
        stmt = c.createStatement();  
  
        String sql = "INSERT INTO estoque VALUES(1, 'Pokebola', 50, 4);";  
        stmt.executeUpdate(sql);  
        sql = "INSERT INTO estoque VALUES(2, 'Poção', 100, 10);";  
        stmt.executeUpdate(sql);  
        sql = "INSERT INTO estoque VALUES(3, 'Reviver', 400, 2);";  
        stmt.executeUpdate(sql);  
        sql = "INSERT INTO estoque VALUES(4, 'Ultrabola', 150, 10);";  
        stmt.executeUpdate(sql);  
        stmt.close();  
        c.commit();  
        c.close();  
    } catch (Exception e) {  
        System.err.println( e.getClass().getName() + ": " + e.getMessage() );  
        System.exit(0);  
    }  
    System.out.println("Operação realizada com sucesso!");  
}
```

Conceito Básico de Banco de Dados

- Outra forma de inserir os dados no banco é utilizando a seguinte sintaxe:

```
sql = "INSERT INTO estoque VALUES(?,?, ?, ?);";  
PreparedStatement ps = c.prepareStatement(sql);  
ps.setInt(1, 5);  
ps.setString(2, "Ultrabola");  
ps.setFloat(3, 300.50f);  
ps.setInt(4, 6);  
ps.executeUpdate();
```

Conceito Básico de Banco de Dados

- Ajustar o código para consultar na tabela do banco.

```
public static void main(String[] args) {  
    Connection c = null;  
    Statement stmt = null;  
    try {  
        //SQLite  
        c = DriverManager.getConnection("jdbc:sqlite:test.db");  
        c.setAutoCommit(false);  
  
        stmt = c.createStatement();  
  
        ResultSet rs = stmt.executeQuery("SELECT * FROM estoque;");  
  
        while(rs.next()){  
            int id = rs.getInt("id");  
            String nome = rs.getString("name");  
            double preco = rs.getFloat("price");  
            int quant = rs.getInt("count");  
            System.out.println("ID: " + id);  
            System.out.println("Nome: " + nome);  
            System.out.println("Preço: " + preco);  
            System.out.println("Quant.: " + quant);  
        }  
        rs.close();  
        stmt.close();  
        c.commit();  
        c.close();  
    } catch (Exception e) {  
        System.err.println( e.getClass().getName() + ": " + e.getMessage() );  
        System.exit(0);  
    }  
    System.out.println("Operação realizada com sucesso!");  
}
```

Conceito Básico de Banco de Dados

- Ajustar o código para atualizar na tabela do banco.

```
public static void main(String[] args) {  
    Connection c = null;  
    Statement stmt = null;  
    try {  
        //SQLite  
        c = DriverManager.getConnection("jdbc:sqlite:test.db");  
        c.setAutoCommit(false);  
  
        stmt = c.createStatement();  
        String sql = "UPDATE estoque SET count = 0 WHERE ID=1;";  
        stmt.executeUpdate(sql);  
        c.commit();  
  
        ResultSet rs = stmt.executeQuery("SELECT * FROM estoque;");  
  
        while(rs.next()){  
            int id = rs.getInt("id");  
            String nome = rs.getString("name");  
            double preco = rs.getFloat("price");  
            int quant = rs.getInt("count");  
            System.out.println("ID: " + id);  
            System.out.println("Nome: " + nome);  
            System.out.println("Preço: " + preco);  
            System.out.println("Quant.: " + quant);  
        }  
        rs.close();  
        stmt.close();  
        c.commit();  
        c.close();  
    } catch (Exception e) {  
        System.err.println( e.getClass().getName() + ": " + e.getMessage() );  
        System.exit(0);  
    }  
    System.out.println("Operação realizada com sucesso!");  
}
```

Conceito Básico de Banco de Dados

- Ajustar o código para deletar na tabela do banco.

```
public static void main(String[] args) {
    Connection c = null;
    Statement stmt = null;
    try {
        //SQLite
        c = DriverManager.getConnection("jdbc:sqlite:test.db");
        c.setAutoCommit(false);

        stmt = c.createStatement();
        String sql = "DELETE FROM estoque WHERE id=1;";
        stmt.executeUpdate(sql);
        c.commit();

        ResultSet rs = stmt.executeQuery("SELECT * FROM estoque;");

        while(rs.next()){
            int id = rs.getInt("id");
            String nome = rs.getString("name");
            double preco = rs.getFloat("price");
            int quant = rs.getInt("count");
            System.out.println("ID: " + id);
            System.out.println("Nome:" + nome);
            System.out.println("Preço:" + preco);
            System.out.println("Quant.:" + quant);
        }
        rs.close();
        stmt.close();
        c.commit();
        c.close();
    } catch (Exception e) {
        System.err.println( e.getClass().getName() + ": " + e.getMessage() );
        System.exit(0);
    }
    System.out.println("Operação realizada com sucesso!");
}
```


Exercício com SQLite

- Um dado pesquisador muito famoso, biólogo, te enviou para uma missão muito específica: “Consertar o que o programador anterior fez de estranho”.

Programador Anterior Retirado de
(https://abrilexame.files.wordpress.com/2016/09/size_960_16_9_20151019-14270-17wsxzl.jpg), em 02/09/2018



Exercício com SQLite

- Você foi recomendado(a) por esse programador, que transmitiu muito confiança ao pesquisador.

Programador que te recomendou Retirado de
(<https://i.ytimg.com/vi/KEkrWRHCDQU/maxresdefault.jpg>), em 02/09/2018



Exercício com SQLite

- Seu trabalho será construir um banco de dados que permita que o pesquisador armazene o nome, onde ele encontrou (lat e lon), a altura e o peso (esse pesquisador é famoso por definir isso apenas com um olhar) e o tipo do animal avistado.
- Como, por questões de Marketing, o tipo do pok... Animal muda a todo momento, seu programa deve permitir que ele atualize os dados inseridos.

Exercício com SQLite

- OBSERVAÇÃO: para essa interação, ainda não é necessário construir uma interface gráfica. Esperar pela próxima interação (MySQL).

Persistência através de sockets

É possível conectar-se com qualquer base de dados através da abertura de um socket TCP com o servidor que o hospeda, por exemplo um Oracle ou MySQL e nos comunicarmos com ele através de seu protocolo proprietário.

Porém conhecer o protocolo proprietário complexo em profundidade é difícil, e trabalhar com ele é muito trabalhoso.

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 referiremos a ela como **JDBC**.

Java DataBase Connectivity



Conexão em 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**.

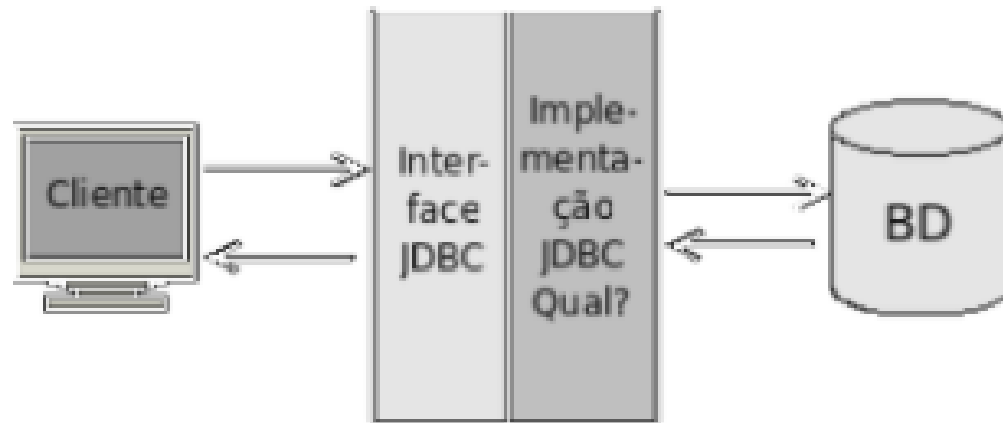
MySQL: <https://dev.mysql.com/downloads/connector/j/>



Driver?

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

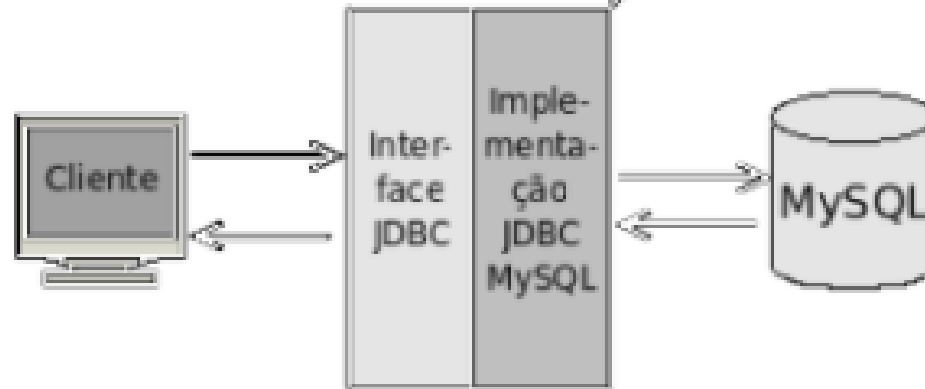
Esse conjunto de classes concretas é quem fará a ponte entre o código cliente que usa a **API JDBC** e o **banco de dados**. São essas classes que sabem se **comunicar através do protocolo proprietário** do banco de dados. Esse conjunto de classes recebe o nome de **driver**.



Driver?

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

```
DriverManager.getConnection("jdbc:mysql://localhost/teste");
```



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**.

Pacote java.sql

Pacote java.sql

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
```

DriverManager - Exemplo

`jdbc:mysql://ip/nome_do_database`

```
DriverManager.getConnection("jdbc:mysql://localhost/alunosteste", "root", "XXXXXXXX");
```

Ao tentar executar essa linha sem ter carregado o pacote com o driver correto receberemos uma exception.

```
java.sql.SQLException: No suitable driver found for
```

A conexão não pôde ser aberta por que?

Carregar o pacote do driver para o projeto

- O que precisamos fazer é adicionar o driver do **MySQL** ao **classpath**, o arquivo **.jar** contendo a implementação **JDBC** do **MySQL** (**mysql connector**) precisa ser colocado em um lugar visível pelo seu projeto ou adicionado à variável de ambiente **CLASSPATH**.
- O arquivo jar pode ser encontrado em:
<https://dev.mysql.com/downloads/connector/j/8.0.html>

Generally Available (GA) Releases


Connector/J 8.0.12

Select Operating System:

Platform Independent ▾

Looking for previous GA versions?

Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-java-8.0.12.tar.gz)	8.0.12	4.8M	Download
MD5: 368c686aec816ca7d2ad9c98ac0e739f Signature			
Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-java-8.0.12.zip)	8.0.12	5.5M	Download
MD5: d7f2ef57f603d245dc7b86db2941ccd6 Signature			

 We suggest that you use the [MD5 checksums](#) and [GnuPG signatures](#) to verify the integrity of the packages you download.

Adaptado de
(<https://dev.mysql.com/downloads/connector/j/8.0.html>), em
02/09/2018

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 instancia de um objeto **Connection**.

Com ele você conseguirá executar **queries**.



Ok. Show me the code!

Perguntas?

Referências

- DevMedia (Thiago Vinícius)
- Oracle
- Caelum