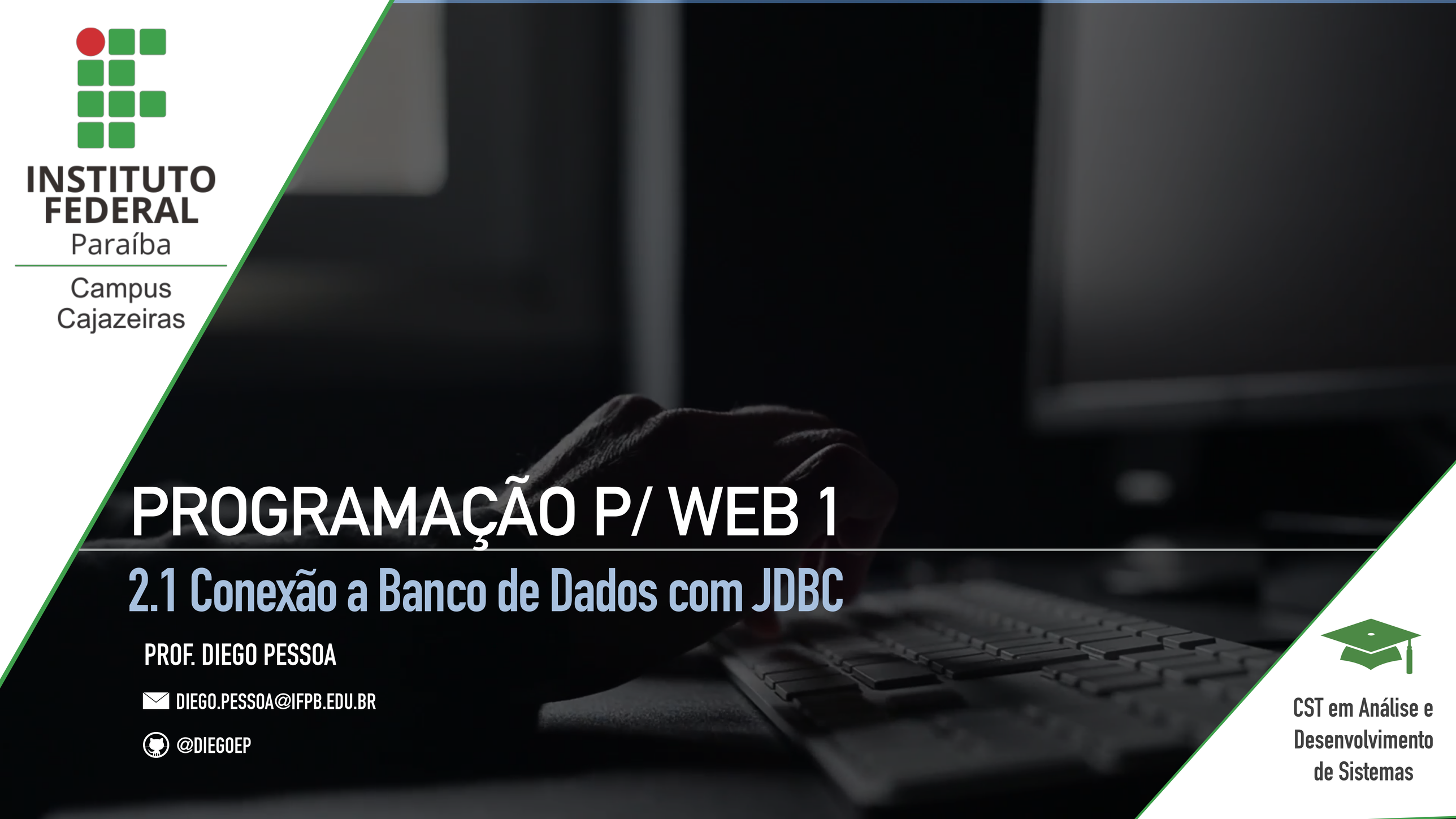


INSTITUTO FEDERAL
Paraíba
Campus Cajazeiras

# PROGRAMAÇÃO P/ WEB 1

## 2.1 Conexão a Banco de Dados com JDBC


PROF. DIEGO PESSOA

DIEGO.PESSOA@IFPB.EDU.BR

@DIEGOEP


CST em Análise e Desenvolvimento de Sistemas

# Introdução

- Desde o surgimento dos SGBD´s, surgiu a necessidade das aplicações se comunicarem com os mesmos de uma forma eficiente;
  - **Problema:** Cada SGBD tinha a sua própria forma de acesso;
- Então, surgiu a necessidade de uniformizar o acesso aos vários SDBD´s existentes;
  - **Solução:** Surge o padrão ODBC;

# O Padrão ODBC

- Open Database Connectivity;
- Lançado em 1992 pela Microsoft;
- Permite a conectividade com vários SGBD´s convencionais;
- Implementado na linguagem C;
- As chamadas da aplicação são traduzidas para chamadas ODBC, que fazem a chamada no SGBD e retornam o resultado para a aplicação;
  - ODBC faz o intermédio entre aplicação e SGBD.

# O Padrão ODBC

- A medida em que a linguagem Java foi ganhando popularidade, o padrão ODBC mostrou-se ineficiente;
- Os principais problemas encontrados foram:
  - ODBC é escrito na linguagem C;
  - Problemas ao ser usado diretamente com java;
  - Chamada de métodos nativos ainda tem problemas com segurança, robustez, portabilidade, etc;
  - Java não tem a noção de ponteiros, que são muito utilizados em ODBC;
- *Solução: Na Época, a SUN desenvolve o seu próprio padrão, chamado JDBC;*

# JDBC

- Java Database Connectivity;
- É um conjunto de classes e interfaces que permitem o acesso a bancos de dados através da linguagem java;
- Oferece um acesso uniforme a uma grande quantidade de SGBD´s;
  - Access, SQL Server, Oracle, PostgreSQL, etc;
- Oferece a portabilidade da linguagem aos drivers de acesso ao SGBD;

# JDBC

- Através de JDBC podemos:
  - Estabelecer conexões (locais ou remotas) com SGBD´s;
  - Executar comandos DDL e DML;
  - Receber e manipular um resultado;
  - Executar procedimentos armazenados;
  - Executar transações;
  - Etc;

# JDBC

- A tecnologia JDBC é constituída de quatro versões: JDBC 1.0; JDBC 2.0;  JDBC 3.0 e JDBC 4.0;
  - JDBC 1.0:
    - Oferece as funcionalidades básicas de acesso aos dados;
    - Conexões e execução de comandos SQL: DDL e DML;
  - JDBC 2.0:
    - O escopo da API foi ampliado para prover suporte a aplicações mais avançadas;
    - Suporte para características requeridas por servidores de aplicação;
    - Melhoria na performance;

# JDBC

- A tecnologia JDBC é constituída de quatro versões: JDBC 1.0; JDBC 2.0;  JDBC 3.0 e JDBC 4.0;
  - JDBC 3.0:
    - O objetivo desta versão era complementar a API com pequenas funcionalidades que ainda não eram oferecidas;
    - Suporte a savepoints, reuso de prepared statements em pools de conexões, pools de conexões configuráveis, etc;
  - JDBC 4.0:
    - Objetivo de facilitar o desenvolvimento para todos os desenvolvedores que trabalham com SQL na plataforma java;
    - Visa prover funcionalidades para expor a API JDBC para ferramentas e APIs mais poderosas que gerenciam fontes de dados;

# JDBC

- Uma "conversação" entre uma aplicação java e o SGBD via JDBC contém os seguintes passos:
  - Carregar o driver JDBC;
  - Estabelecer uma conexão com o SGBD;
  - Executar comandos SQL;
  - Liberar os recursos;
  - Fechar a conexão;

# Carregando o Driver JDBC

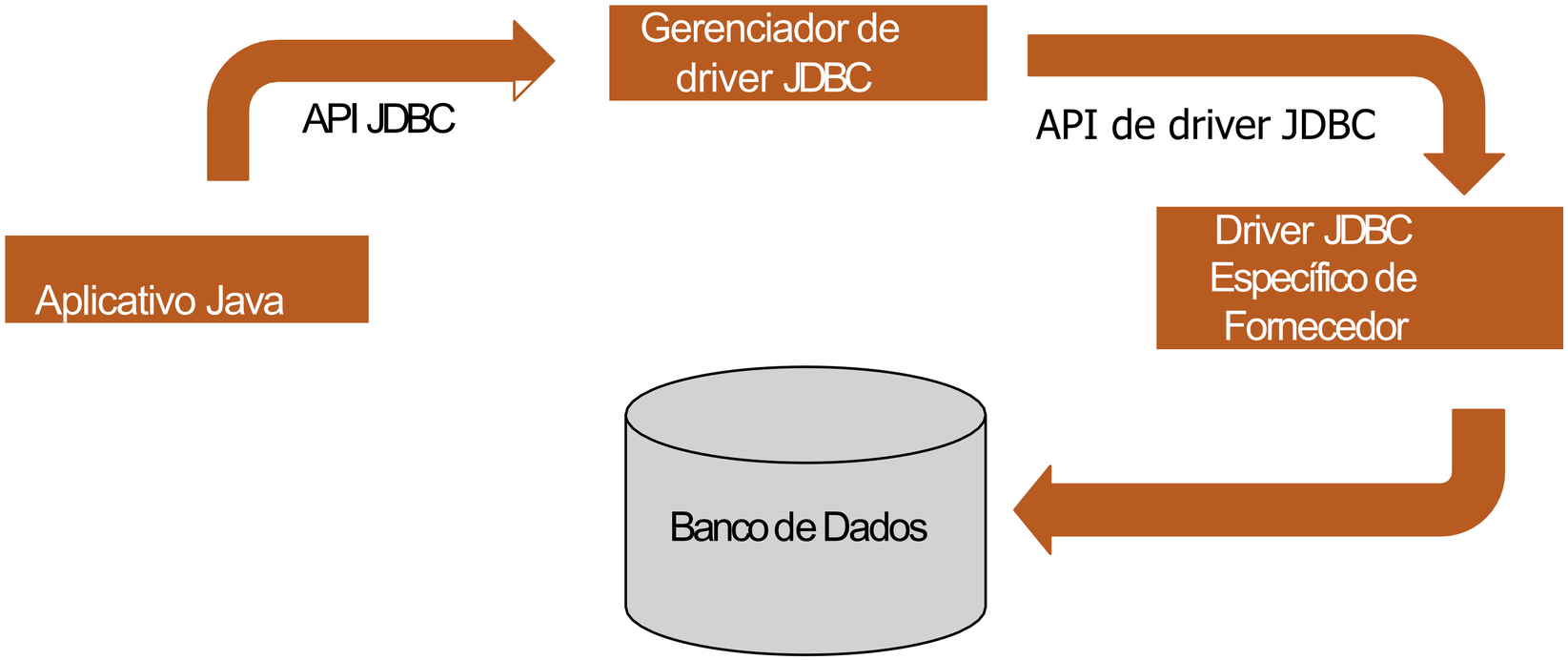
- Podemos acessar um SGBD via JDBC de suas formas:
  - Através de um driver JDBC;
    - Driver que implementa a API JDBC para um SGBD específico;
    - Implementação em java da API;
    - Hoje, os principais SGBD´s já oferecem drivers compatíveis com esta API;
    - É a melhor forma de usar esta tecnologia;
  - Através de uma ponte JDBC-ODBC;
    - As ações JDBC são convertidas para o padrão ODBC, que faz o acesso ao SGBD;
    - Útil quando o SGBD não tem um driver JDBC correspondente;
    - Opção que só deve ser usada em último caso;

# Carregando o Driver JDBC

- Usamos o método estático forName da classe Class para carregar o driver JDBC;
- Caso o acesso seja feito através de um driver, passamos o nome do mesmo como parâmetro:
  - Class.forName("org.postgresql.Driver");
  - Class.forName("com.informix.jdbc.IfxDriver");
- Caso o acesso seja feito através de uma ponte, usamos a sintaxe:
  - Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
- O driver JDBC deve estar no classpath da aplicação;
- ** A partir da versão 4.0 da API JDBC, não é mais necessário carregar o driver JDBC explicitamente (basta adicionar o driver ao classpath da aplicação)*

# Carregando o Driver JDBC

- O driver é a parte do software que sabe como conversar com o servidor de banco de dados.



# Estabelecendo uma Conexão

- Para uma aplicação se comunicar com um SGBD ela deve primeiramente estabelecer uma conexão
  - Representa um canal de conversação entre a aplicação e o banco;
  - Este canal de comunicação também é chamado de sessão;
- Uma aplicação pode ter mais de uma conexão aberta com o SGBD simultaneamente;
  - E também abrir conexões simultâneas com vários SGBD´s;
- Em JDBC, conexões com SGBD´s são representadas através da interface Connection;

# Estabelecendo uma Conexão

- Obtemos uma conexão com o SGBD através do método  estático getConnection da classe DriverManager;
- A sintaxe do método é a seguinte:
  - DriverManager.getConnection (url, login, password);
  - Onde:
    - URL corresponde à URL do SGBD;
    - login corresponde ao nome do usuário (SGBD);
    - password corresponde à senha deste usuário no SGBD;

# Estabelecendo uma Conexão

- O formato da URL é o seguinte:
  - jdbc:<subprotocol>:<subname>:<banco>
- Onde:
  - jdbc é o protocolo;
  - <subprotocol> e <subname> variam de acordo com o driver;
  - <banco> representa a identificação do banco de dados a ser acessado;

# Estabelecendo uma Conexão

- Caso o acesso seja feito através de uma ponte, a url terá a seguinte forma:
  - jdbc:odbc:NomeDaFonteOdbc;
- Caso o acesso seja feito através de um driver JDBC, a documentação do mesmo deve ser verificada;
  - A documentação indica como criar URL´s para o driver;

# Estabelecendo uma Conexão

- O método retorna um objeto que implementa a interface Connection, caso seja executado com sucesso;
  - Este objeto será usado para realizar operações no banco de dados;
- Caso ocorra uma falha na conexão, uma SQLException é lançada;

# Estabelecendo uma Conexão

- Exemplo de um fragmento de código que carrega o driver JDBC e estabelece uma conexão com um banco de dados via uma ponte JDBC/ODBC;

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url =  "jdbc:odbc:jdbcdb";
String login =  "mylogin";
String password =  "mypasswd";
Connection connection =  DriverManager.getConnection(url, login, password);
```

# Estabelecendo uma Conexão

- Exemplo de um fragmento de código que carrega o driver JDBC e estabelece uma conexão com um banco de dados via JBDC.

```
Class.forName (“org.postgresql.Driver");
String url =  “jdbc:postgresql://localhost:5432/BancoTeste";
String login =  "mylogin";
String password = "mypasswd";
Connection connection =  DriverManager.getConnection(url, login,  password);
```

# Estabelecendo uma Conexão

- Defina a URL de conexão.
- URLs que referenciam banco de dados usam o protocolo jdbc: e especificam o host do servidor, porta e nome do banco de dados.
  - O formato exato é definido na documentação de cada driver.

# Estabelecendo uma Conexão

- A classe Connection estabelece alguns métodos úteis:
  - **prepareStatement:** cria consultas pré-compiladas para submissão ao banco de dados.
  - **prepareCall:** Acessa procedimento armazenados no banco de dados.
  - **rollback/commit:** Controla o gerenciamento de transação.
  - **close:** Encerra a conexão aberta.
  - **isClosed:** Determina se a conexão expirou ou foi fechada explicitamente.

# Executando Comandos SQL

- A interface Statement:
  - Interface que permite a execução de comandos SQL dentro do SGBD;
  - Ela executa um comando SQL no banco de dados e retorna (no caso de consultas) o resultado para a aplicação java;
  - Podemos criar um Statement chamando o método createStatement a partir da conexão criada;
  - O resultado do método é um objeto que implementa esta interface;

# Executando Comandos SQL

- Criar um objeto Statement :
  - Um objeto Statement habilita realizar consultas e comandos ao banco de dados.

# Executando Comandos SQL

- O método executeUpdate:
  - Método da interface Statement permite a execução de:
    - Comandos DML (Inserções, Atualizações e Exclusões de dados, por exemplo);
    - Comandos DDL (Criação, Atualizações e Exclusões de tabelas, por exemplo);
- As operações são executadas através de comandos SQL;
- O método recebe como parâmetro de entrada um String, que descreve o comando SQL que deve ser executado;
- Uma SQLException é lançada caso algum erro aconteça durante a execução do comando;

# Executando Comandos SQL

- O método executeUpdate:
- Exemplo de um trecho de código que mostra a criação de uma tabela via JDBC;

```
Statement statement = connection.createStatement();
String create = "CREATE TABLE Aluno (Matricula VARCHAR(10),
Nome VARCHAR(30), Idade INT, PRIMARY KEY(Matricula))";
statement.executeUpdate(create);
```

# Executando Comandos SQL

- O método executeUpdate:
- Exemplo de um trecho de código que mostra a inserção de uma tupla na referida tabela;

```
[...]
String insertion = "INSERT INTO Aluno VALUES
        ('20180405','Maria', 20)";
Statement statement = connection.createStatement();
statement.executeUpdate(insertion);
[...]
```

# Executando Comandos SQL

- O método executeQuery:
- Método da interface Statement que permite a realização de consultas SQL;
- Recebe como parâmetro de entrada um String
- correspondente ao script SQL da consulta a ser executada;
- O resultado do método é um objeto do tipo ResultSet;
  - Caso a não consulta não selecione nenhuma tupla, o resultado é um ResultSet vazio;
- Um Statement só pode realizar uma consulta de cada vez;

# Executando Comandos SQL

- O método executeQuery:

  Uma SQLException é lançada caso algum erro ocorra durante a consulta;

  Exemplo de um trecho de código que mostra a realização de uma consulta:

  - String query = "SELECT * FROM Aluno";
  - ResultSet result = statement.executeQuery(query);

# Executando Comandos SQL

- Percorrendo o resultado de uma consulta:
- O ResultSet possui um cursor que nos permite navegar no resultado de uma consulta;
- A navegação é feita através dos métodos next() e last();
  - Estas operações retornam um boolean;
    - true, quando ela é feita com sucesso.
    - false, quando o cursor é movido para uma posição inválida;
  - Ao terminarmos a navegação, usamos o método close() para fecharmos o cursor;

# Executando Comandos SQL

- Percorrendo o resultado de uma consulta:
- Quando realizamos uma consulta, o cursor começa apontando para uma posição vazia;
- O primeiro next() executado no ResultSet posiciona o cursor sobre a primeira tupla do resultado;
- Caso esta operação retorne false, significa que nenhuma tupla foi selecionada pela consulta;

# Executando Comandos SQL

- Recuperando o valor de uma coluna:
  - Para acessar o valor de uma coluna usamos um dos métodos getters da classe ResultSet;
    - getInt(), getString(), getFloat(), etc;
  - Usamos o método que está associado ao tipo de dado armazenado na coluna;
    - Tipo de dado java;
    - Veremos as conversões de tipos entre Java e SQL posteriormente;
  - Devemos passar como parâmetro de entrada o nome da coluna a ser acessada;

# Executando Comandos SQL

- Recuperando o valor de uma coluna:
  - Podemos especificar a coluna através do seu nome:
    - Exemplo: Se executarmos o comando result.getString(“NomeDaColuna”) acessamos o valor do atributo NomeDaColuna;
      - O método recupera o valor do nome para a tupla que está sendo referenciada atualmente pelo cursor;
    - O valor do nome da coluna nesta chamada não é Case Sensitive;

# Executando Comandos SQL

- Recuperando o valor de uma coluna:
  - Podemos acessar uma coluna através de um inteiro, que representa o índice da coluna a ser acessada;
    - Neste caso, é usada a ordem em que os atributos foram especificados na consulta;
    - A indexação começa pelo valor 1;
    - Por exemplo, na nossa tabela Pessoa, o método getString(1) recupera o valor da coluna Matricula;

# Executando Comandos SQL

- Tabela de correspondência dos principais tipos Java/SQL;

| SQL | Java | SQL | Java |
|---|---|---|---|
| Char | String | BigInt | long |
| Varchar | String | Real | float |
| Numeric | java.lang.Bignum | Float | double |
| Decimal | java.lang.Bignum | Binary | byte[] |
| Bit | boolean | Date | java.sql.Date |
| Smallint | short | Time | java.sql.Time |
| Integer | int | TimeStamp | Java.sql.Timestamp |

# Executando Comandos SQL

- Exemplo de um fragmento de código que recupera o nome de todos os alunos:

```
[…]
String query = “Select Nome FROM Aluno”;
ResultSet result = stat.executeQuery(query);
Collection nomeDosAlunos = new Vector();
while(result.next())
    nomeDosAlunos .add(result.getString(“Nome”));
```

# Executando Comandos SQL

- A classe PreparedStatement:
  - Usada quando queremos usar nos scripts SQL parâmetros definidos pelo usuário;
  - Na hora de especificar o comando SQL que deve ser executado, usamos o valor ? para especificar os valores que serão definidos pelo usuário;

# Executando Comandos SQL

- A classe PreparedStatement:
  - Depois usamos os métodos setters para substituir as interrogações pelos valores desejados;
    - setInt, setFloat, SetString, etc;
  - O método setter escolhido deve ser aquele que se relaciona ao tipo de dado java que será substituído na consulta;

# Executando Comandos SQL

- A classe PreparedStatement:
  - Cada método setter deve receber dois parâmetros de entrada, na respectiva ordem;
    - Um inteiro, indicando qual ? será substituída;
    - Um valor, indicando o valor que substituirá a correspondente ?;
      - O tipo do valor varia de acordo com cada método setter;

# Executando Comandos SQL

- A classe PreparedStatement:
- Usamos o método prepareStatement da interface Connection para preparar a execução do comando;
  - A String que corresponde ao comando com as ? deve ser passado como parâmetro;
- Depois, feitas as substituições de todas as ?, chamamos o método executeQuery ou executeUpdate para executar o comando desejado;
- *Exemplo:* Vamos supor que queremos oferecer um método que retorna um iterador para o nome dos alunos que estão dentro desta faixa de idade definida pelo usuário;

# Executando Comandos SQL

- Exemplo (continuação):

```
public Iterator getAlunos(int minIdade, int maxIdade) throws SQLEXception{
    String query =  "SELECT Nome FROM Aluno WHERE Idade >=  ?AND Idade <=  ?";
    PreparedStatement stat = connection.prepareStatement(query);
   stat.setInt (1, minIdade);
   stat.setInt(2, maxIdade);
   Collection result =  new Vector();
   ResultSet rs =  stat.executeQuery();
   while(rs.next())
        result.add(rs.getString(“Nome”));
        return result.iterator();
}
```

# Executando Comandos SQL

- A classe PreparedStatement:
  - A consulta anterior funciona como se estivéssemos executando o seguinte comando:
    - "SELECT Nome FROM Aluno WHERE Idade >="+minIdade +" AND Idade <= " + maxIdade;
- Para agilizar o processo de consultas e evitar SQL Injection, seria ideal a utilização de um objeto do tipo *PreparedStatement*, pois as consultas podem ser compiladas pelo banco para múltiplas execuções.

# Liberando os Recursos

- Terminadas todas as ações desejadas, devemos liberar todos os recursos que utilizamos;
- Isto inclui:
  - Fechar todos os Statements;
    - Chamando o método close no objeto que representa cada Statement;
  - Encerrar a conexão;
    - Chamando o método close do objeto que representa a conexão;

```java
public class Pessoa {
    private Integer id;
    private String nome;
    private String email;
    public Pessoa() {
    }
    public Pessoa(Integer id, String nome, String email)
        this.id = id;
        this.nome = nome;
        this.email = email;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    @Override
    public String toString() {
        return this.id + " - " + this.nome + " - " + this.email;
    }
}
```

```java
public class TesteJDBC {

    public static void main(String[] args) {
        try {
            //Carregando o driver JDBC para para o SGBD PostgreSQL
            Class.forName("org.postgresql.Driver");
            //Estabelecendo a conexão com a base de dados do SGBD
            Connection connection = DriverManager.getConnection("jdbc:postgresql://localhost:5432/PSD_JDBC",
                    "postgres", "12345");
            //Criando um objeto do tipo da interface Statement
            Statement statement = connection.createStatement();
            //Executando comandos SQL através do objeto criado e inserindo num ResultSet
            ResultSet resultado = statement.executeQuery("Select * from Pessoa");
            //Criando a lista de objetos que armazena a lista com o resultado da consulta
            List<Pessoa> pessoas = new LinkedList<Pessoa>();
            //Percorrendo no resultado
            while(resultado.next()){
                int id= resultado.getInt("id");
                String nome= resultado.getString("nome");
                String email = resultado.getString("email");
                Pessoa pessoa = new Pessoa(id, nome, email);
                pessoas.add(pessoa);
            }
            //Apenas imprimindo o reusultado da consulta
            for (Pessoa p : pessoas) {
                System.out.println(p);
            }
        } catch (Exception ex) {
            ex.getMessage();
        }
    }
```

# Exemplo Simples

- Algumas considerações:
  - Programar Orientado a Objetos exige organização;
  - Realizar todos os passos em uma única classe foi apenas para facilitar o aprendizado, mas não para fazer nos projetos (lembre do padrão MVC visto anteriormente).

# Exemplo

- Salvando...

```
public class PessoaDAO {

    private Connection conexao;
    private PreparedStatement statment;

    public PessoaDAO() throws SQLException {...}

    public void salvar(Pessoa pessoa) throws SQLException {
        try {
            abrir();
            String comando = "insert into pessoa (nome, email) values (?, ?)";
            this.statment = this.conexao.prepareStatement(comando);
            this.statment.setString(1, pessoa.getNome());
            this.statment.setString(2, pessoa.getEmail());
            this.statment.executeUpdate();
        } finally {
            this.liberar();
        }
    }
```

# Exemplo

- Configurando a conexão....

```
public class ConnectionFactory {

    private String url;
    private String password;
    private String user;
    private String driver;
    private static ConnectionFactory instance = null;

    private ConnectionFactory(String resouceName) {...}

    public static ConnectionFactory getInstance(String resouceName) {...}

    public Connection getConnection() throws SQLException {
        try {
            Class.forName(this.driver);
            return DriverManager.getConnection(this.url, this.user, this.password);
        } catch (ClassNotFoundException ex) {
            throw new SQLException(ex);
        }
    }
}
```

# Exemplo – Consultando...

```
public List<Pessoa> listar(String comando) throws SQLException {
    List pessoas = new ArrayList<Pessoa>();
    ResultSet rs;
    try {
        abrir();
        this.statment = this.conexao.prepareStatement(comando);
        rs = this.statment.executeQuery();
        while (rs.next()) {
            Integer id = rs.getInt("id");
            String nome = rs.getString("nome");
            String email = rs.getString("email");
            Pessoa pessoa = new Pessoa();
            pessoa.setId(id);
            pessoa.setNome(nome);
            pessoa.setEmail(email);
            pessoas.add(pessoa);
        }
        rs.close();
    } finally {
        this.liberar();
        return pessoas;
    }
```

# Considerações Finais

- O que vimos aqui foi apenas o básico de JDBC;
- No entanto, ela é uma API muito abrangente, com muito mais recursos do que aqueles que estudamos aqui;
- Você deve se encarregar de aprofundar seus conhecimentos;
- O material da Oracle é a melhor fonte para você estudar;
  - https://www.oracle.com/technetwork/java/overview-141217.html