



PostgreSQL Introdução

Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro
Prof. Edwar Saliba Júnior
Setembro de 2021



O que é o PostgreSQL?

- SGBD Relacional, gratuito.
- Possui conceitos avançados como:
 - classes,
 - heranças,
 - tipos e
 - funções;
- Outras funcionalidades:
 - constraints,
 - triggers,
 - rules e
 - transaction integrity.



PostgreSQL

- As funcionalidades apresentadas no *slide* anterior classificam o PostgreSQL como sendo:
 - SGBD *Object-relational*.
- Entenda! PostgreSQL **não é** um SGBD *Object-oriented*.



Notação

- **[]** (colchetes) indicam uma frase, palavra-chave ou comando facultativo.
- **{ }** (chaves) contendo **|** (barra vertical) entre comandos, indica que você pode escolher um ou outro.
- **()** (parêntesis) são usados para agrupar expressões booleanas.



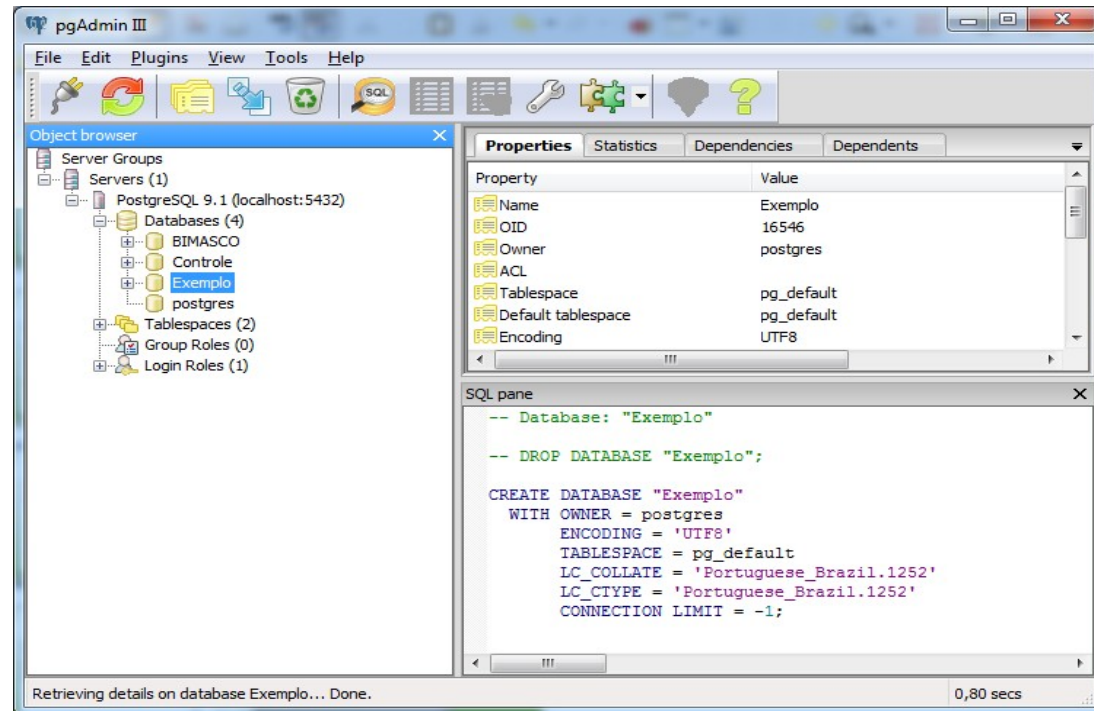
Linguagem SQL

- *Structured Query Language*
- Usada para fazer consultas através de relações que são criadas entre registros (linhas), campos (colunas) e tabelas.
- Utilizada também para:
 - inserir,
 - apagar e
 - atualizar registros nas tabelas.



PgAdmin 3

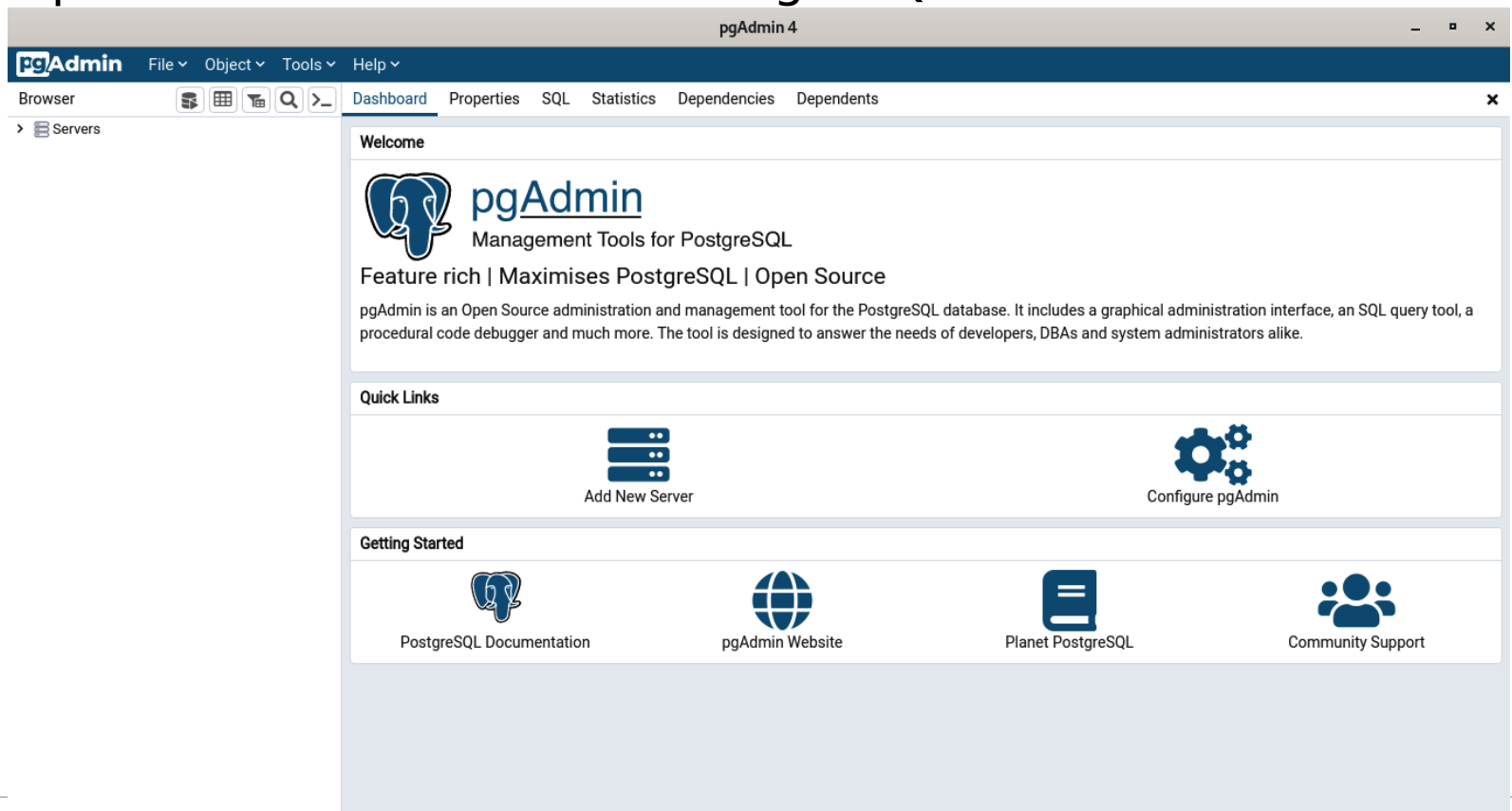
- Ferramenta gráfica utilizada para facilitar as operações que devem ser feitas no PostgreSQL.





PgAdmin 4

- Ferramenta gráfica utilizada para facilitar as operações que devem ser feitas no PostgreSQL.





pgAdmin

- O *pgAdmin* vem junto com a instalação do PostgreSQL.
- Nele podem ser feitas, de forma prática e sem muito conhecimento do SGBD, praticamente todas as operações de *Data Definition Language* existentes, ou seja, pode-se:
 - criar bancos, tabelas, *views* e etc.
- Tudo utilizando-se a facilidade da interface gráfica.



Observação

- O PostgreSQL é um SGBD *case-sensitive*, ou seja, sensível a caixa.
- No entanto, caso você queira fazer uso deste recurso deverá colocar o nome das estruturas entre aspas duplas.
- Exemplos:
 - aplicação do recurso,
 - não aplicação do recurso.



Aplicação do Recurso

- A sentença a seguir cria a seguinte tabela:

```
create table "Cliente"(  
    codigo integer,  
    "primeiroNome" character varying,  
    "CPF" character varying  
);
```

Cliente		
codigo	primeiroNome	CPF



Não Aplicação do Recurso

- A sentença a seguir cria a seguinte tabela:

```
create table Cliente(  
    codigo integer,  
    primeiroNome character varying,  
    CPF character varying  
);
```

cliente		
codigo	primeironome	cpf



Case-sensitive

Cliente		
codigo	primeiroNome	CPF

- Se você criou a tabela acima e quer consultar todos os registros, então:

```
select codigo, "primeiroNome", "CPF"  
  from "Cliente";
```

- pois, nenhuma das sentenças abaixo funcionará:

```
select codigo, primeiroNome, CPF  
  from Cliente;  
select codigo, primeironome, cpf  
  from cliente;
```



Comandos Abordados Nesta Aula

- Não exatamente nesta ordem:
 - *select*,
 - *union*,
 - *intersect*,
 - *except*,
 - *create table*,
 - *create index*,
 - *create view*,
 - *drop table*,
 - *drop index*,
 - *drop view*,
 - *insert into*,
 - *update* e
 - *delete*.



Tabelas Exemplo

produto			
codigo	nome	preco	codtipo
1	arroz	5.00	4
2	feijão	9.30	4
3	batata	2.45	3
4	ervilha	1.78	2

tipo	
codigo	descricao
1	enlatados
2	farináceos
3	legumes
4	grãos

Vamos pressupor que já temos um banco de dados criado no nosso SGBD e que as tabelas acima já existem, exatamente como mostrado, dentro deste banco de dados.



Select

- Sintaxe:

```
SELECT [ALL|DISTINCT]{ * | expr_1 [AS c_alias_1] [,
    ...[, expr_k [AS c_alias_k]]]}
FROM table_name_1 [t_alias_1] [, ... [, table_name_n
    [t_alias_n]]]
[WHERE condition]
[GROUP BY name_of_attr_i [,... [, name_of_attr_j]]]
[HAVING condition]
[{UNION [ALL] | INTERSECT | EXCEPT} SELECT ...]
[ORDER BY name_of_attr_i [ASC|DESC][, ... [,
    name_of_attr_j [ASC|DESC]]]];
```



Exemplo *Select*

- Comando utilizando álgebra relacional:

```
select *  
  from produto  
 where preco > 3
```

- Resultado:

	codigo	nome	preco	codtipo
1		arroz	5.00	4
2		feijão	9.30	4

- Observação: o * é utilizado para definir que todos os campos da tabela deverão ser mostrados no resultado da consulta.



Exemplo *Select*

- Comando selecionando campos:

```
select nome, preco  
from produto  
where preco > 3
```

- Resultado:

nome	preco
arroz	5.00
feijão	9.30



Exemplo *Select*

- Comando utilizando conectores `and` e `or`:

```
select nome, preco
  from produto
 where nome = 'ervilha'
        or (preco > 4
           and preco < 6)
```

- Resultado:

nome	preco
arroz	5.00
ervilha	1.78



Exemplo *Select*

- Comando utilizando expressões aritméticas:

```
select nome, preco
  from produto
 where preco * 2 <= 6
```

- Resultado:

nome	preco
batata	2.45
ervilha	1.78



Exemplo *Select*

- Comando utilizando `join` entre tabelas:

```
select p.nome, p.preco, t.descricao
  from produto p, tipo t
 where p.codtipo = t.codigo
       and t.descricao = 'grãos'
```

- Resultado:

nome	preco	descricao
arroz	5.00	grãos
feijão	9.30	grãos



Operadores de Agregação

- A linguagem SQL provê operadores de agregação que utilizam, para cálculo, as colunas (atributos) da tabela:
 - AVG (média),
 - COUNT (contador),
 - SUM (soma),
 - MIN (mínimo) e
 - MAX (máximo).
- O valor resultante destes operadores, é obtido com o cálculo aplicado sobre todos os valores de todos os registros especificados na consulta SQL.



Exemplo *Select*

- Comando utilizando o agregador `avg` (média):

```
select avg(preco) as Media_de_Precos  
from produto
```

- Resultado:

Media_de_Precos
4.6325

- Observação: “`as`” modifica o nome de uma coluna na visualização da consulta.



Exemplo *Select*

- Comando utilizando o agregador `count` (contador):

```
select count(*) as Qtdade_Produtos  
from produto
```

- Resultado:

Qtdade_Produtos
4



Agregação por Grupos

- A linguagem SQL permite que você divida as tuplas (registros ou linhas) de uma tabela por grupos.
- Se assim for, o valor totalizado pelo operador de agregação utilizado, não será calculado sobre todos os valores contidos na coluna especificada da tabela inteira, mas sobre todos os valores do grupo.
- O particionamento das tuplas em grupos é feito utilizando-se o comando GROUP BY seguido dos atributos que definem os grupos.



Exemplo de Agregação por Grupo

- Queremos saber quantos produtos de cada tipo existem cadastrados:

```
select t.codigo, t.descricao,  
       count(p.codigo) as Qtdade  
from tipo t, produto p  
where t.codigo = p.codtipo  
group by t.codigo
```

- Resultado:

codigo	descricao	Qtdade
1	farináceos	1
3	legumes	1
4	grãos	2



Having

- A cláusula `having` funciona como a cláusula `where`, porém ela é aplicada sobre os grupos que são classificados pela cláusula `group by` na sentença SQL.
- As expressões permitidas na cláusula `having` precisam:
 - envolver funções de agregação ou
 - usar todos os atributos descritos na cláusula `where` ou
 - usar todas as expressões que utilizam funções de agregação e que foram utilizadas na consulta.



Exemplo com *Having*

- Queremos saber quantos produtos de cada tipo existem cadastrados onde a “Qtidade” seja maior que 1:

```
select t.codigo, t.descricao,  
       count(p.codigo) as Qtidade  
from tipo t, produto p  
where t.codigo = p.codtipo  
group by t.codigo  
having count(p.codigo) > 1
```

- Resultado:

codigo	descricao	Qtidade
4	grãos	2



Subqueries

- Nas cláusulas **where** e **having** é permitido usar, onde se espera um valor, uma subquery.
- Para casos como este, a execução da sentença principal se dará somente depois da execução da(s) sentença(s) secundária(s).
- O uso de subqueries estende o poder e as possibilidades de utilização das consultas SQL.



Exemplo com *Subquery*

- Queremos listar todos os produtos que pertencem ao tipo “grãos” ou “legumes”:

```
select codigo, nome
  from produto
 where codtipo in (select codigo
                   from tipo
                   where descricao = 'grãos'
                   or descricao = 'legumes')
```

- Resultado:

codigo	nome
1	arroz
2	feijão
3	batata



Union

- Utilizada para fazer a união de dois resultados de duas *queries* em um único *resultset*.



Exemplo Utilizando *Union*

- Queremos unir os resultados de duas *queries*:

```
select codigo, nome  
  from produto  
 where preco = 5
```

```
union
```

```
select codigo, nome  
  from produto  
 where codtipo = (select codigo  
                  from tipo  
                  where descricao = 'legumes')
```

- Resultado:

codigo	nome
1	arroz
3	batata



Intersect

- Utilizada para gerar como *resultset*, a interseção entre duas *queries*.



Exemplo Utilizando *Intersect*

- Queremos as tuplas que compõem a interseção dos resultados das duas *queries*:

```
select codigo, nome
  from produto
 where preco <= 5
intersect
select codigo, nome
  from produto
 where codtipo = (select codigo
                  from tipo
                  where descricao = 'legumes')
```

- Resultado:

codigo	nome
3	batata



Exceção

- Mostra a diferença, em relação às tuplas, geradas pela execução de duas *queries*.



Exemplo Utilizando *Except*

- Queremos as tuplas que compõem o *resultset* da primeira *query*, menos as tuplas do *resultset* da segunda *query*:

```
select codigo, nome
  from produto
 where codigo > 1
except
select codigo, nome
  from produto
 where codigo > 3
```

- Resultado:

codigo	nome
2	feijão
3	batata



Data Definition Language

- Existem comandos na linguagem SQL que são usados para “definição de dados”, ou melhor para definir onde e como os dados serão alocados dentro do Banco de Dados.



Create Table

- Um dos comandos mais importantes na definição de novas relações, ou seja, novas tabelas.
- Sintaxe:

```
CREATE TABLE table_name(  
    name_of_attr_1 type_of_attr_1  
    [, name_of_attr_2 type_of_attr_2  
    [, ...]]  
);
```



Exemplo *Create Table*

- Criação da tabela “**Tipo**”:

```
create table tipo(  
    codigo integer,  
    descricao varchar(15),  
    primary key(codigo)  
)
```

- Observação: O nome da tabela a ser criada deve vir sempre no singular.



Exemplo *Create Table*

- Criação da tabela “**Produto**”:

```
create table produto(  
    codigo integer,  
    nome varchar(30),  
    preco float,  
    codtipo integer,  
    primary key(codigo),  
    foreign key(codtipo) references tipo(codigo)  
)
```

- Observação: O nome da tabela a ser criada deve vir sempre no singular.



Tipos de Dados em SQL

- Alguns tipos suportados:
 - **Integer**: número (tamanho de uma palavra) com sinal e 31 bits de precisão;
 - **SmallInt**: número (tamanho de meia palavra) com sinal e 15 bits de precisão;
 - **Decimal(p[,q])**: número com sinal num pacote de p dígitos de precisão com q destes p's dígitos de casas decimais ($15 \geq p \geq q$ e $q \geq 0$). Se o valor de q for omitido então este assumirá o valor 0;
 - **Float**: número (tamanho de duas palavras) com sinal em ponto flutuante;
 - **Char(N)**: String com tamanho fixo de N caracteres;
 - **Varchar(N)**: String com tamanho variante de no máximo N caracteres.



Create Index

- Índices são utilizados para melhorar o desempenho na recuperação de dados numa relação (tabela).
- Sintaxe:

```
CREATE INDEX index_name ON table_name  
    ( name_of_attribute );
```



Exemplo *Create Index*

- Criação de índice para o campo nome da tabela “Produto”:

```
create index iNomProd on produto(nome);
```

- Observação: O índice, quando bem utilizado pode melhorar muito a performance de um sistema. No entanto, deve-se usar com moderação e escolher adequadamente os campos que serão classificados como tal, para que um efeito inverso não ocorra.



Create View

- Uma *view* pode ser considerada uma tabela virtual, ou seja, uma tabela que não existe fisicamente no banco de dados, mas que aos olhos do usuário do *software* parece existir.
- Sintaxe:

```
CREATE VIEW view_name AS  
    select_statement
```



Exemplo de *Create View*

- Criação de uma view mostrando o código do produto, seu nome e seu tipo:

```
create view produtoTipo as
  select p.codigo, p.nome, t.descricao
  from produto p, tipo t
  where p.codtipo = t.codigo
```

- Resultado:

	codigo	nome	descricao
1		arroz	grãos
2		feijão	grãos
3		batata	legumes
4		ervilha	enlatados



Drop

- Comando utilizado para apagar por completo, inclusive a estrutura, de tabelas, *views* e índices.
- Sintaxe:
 - para tabelas:
`drop table table_name;`
 - para *views*:
`drop view view_name;`
 - para índices:
`drop index index_name;`



Exemplo de *Drop*

- Tabela:

```
drop table produto;
```

- *View*:

```
drop view produtoTipo;
```

- Índice:

```
drop index iNomProd;
```



Data Manipulation Language

- Comandos para manipulação de dados nas tabelas.



Insert Into

- Uma vez que uma tabela é criada, ela deverá ser preenchida com tuplas, usando-se o comando **insert into**.
- Sintaxe:

```
INSERT INTO table_name (name_of_attr_1  
    [, name_of_attr_2 [...]])  
VALUES (val_attr_1 [, val_attr_2  
    [, ...]]);
```




Exemplo insert into

- Para inserir a primeira tupla na tabela produto, utilizamos a seguinte sentença SQL:

```
insert into produto (codigo, preco, nome,  
codtipo) values (1, 5, 'arroz', 4);
```

- Porém, se os valores a serem colocados na tabela estiverem na ordem dos campos, podemos omitir o nome dos campos, simplificando assim, o comando:

```
insert into produto values (1, 'arroz', 5, 4);
```



Update

- Para modificar um ou mais atributos (campos) de uma relação, usa-se o comando `update`.
- Sintaxe:

```
UPDATE table_name  
    SET name_of_attr_1 = value_1  
        [, ... [, name_of_attr_k = value_k]]  
WHERE condition;
```



Delete

- Para apagar uma tupla de uma tabela, utilizamos o comando `delete`.
- Sintaxe:

```
DELETE FROM table_name WHERE condition;
```



Exemplo de update

- Vamos aumentar o preço do arroz em 30%. Atualmente o arroz está com o preço de R\$5,00. Então nosso arroz passará a custar R\$6,50.
- Vamos atualizar nossa tabela:

```
update produto  
  set preco = 6.5  
  where codigo = 1;
```



Exemplo de delete

- Suponhamos que, por um motivo qualquer, não venderemos mais o produto “batata”.
- Neste caso, vamos excluí-lo da tabela produto:

```
delete from produto where codigo = 3;
```



Conexão: Java e PostgreSQL

- Antes de tentar conectar seu programa com o PostgreSQL, primeiramente você deve:
 - criar um banco de dados e
 - criar as tabelas necessárias.
- Feito isto, então tudo bem. Vamos ao código!



Conexão: Java e PostgreSQL

```
package conexaopostgresql;

import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;

/**
 *
 * @author Edwar Saliba Júnior
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        try {

            String url = "jdbc:postgresql://localhost:5432/TesteJava";
            String usuario = "postgres";
            String senha = "123456";

            Class.forName("org.postgresql.Driver");

            Connection con;

            con = DriverManager.getConnection(url, usuario, senha);

            System.out.println("Conexão realizada com sucesso.");

            Statement stm = con.createStatement();

            // stm.executeQuery("INSERT INTO teste VALUES (1,'Cynthia')");

            stm.executeUpdate("INSERT INTO teste VALUES (1,'Cynthia')");
            //Editado 21/09/2011 para correção: executeQuery é usado para pesquisa, executeUpdate deve ser usado para inserir
            con.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Essa foi minha primeira conexão com o SGBD PostgreSQL.

E como diz o dito popular, a primeira conexão a gente nunca esquece!



Você não deve usar este código. Pois, está ultrapassado.



Observações:

- Este código só insere um registro na tabela criada no PostgreSQL. Mas ele, até então não funciona.
- Para que o código no *slide* anterior venha a funcionar, você deverá colocar o *driver* do PostgreSQL nas bibliotecas do projeto que você criou.
- Faça isto da seguinte forma:
- Se você ainda não tem o *driver*, então vá ao *site* a seguir e faça o *download*:

<http://jdbc.postgresql.org/>

- Caso você já tenha o *driver* eu sugiro que, para fins de teste, crie uma pasta chamada "lib" dentro da pasta do seu projeto e copie o *driver* para lá, isto se você estiver usando a IDE Oracle NetBeans. Caso você esteja usando a IDE Eclipse, basta adicionar o *driver* no projeto (veja o próximo *slide*).



Driver PostgreSQL na IDE Oracle NetBeans

- Se você ainda usa a **IDE Oracle NetBeans** ou está trabalhando com um projeto antigo, na IDE Apache NetBeans, e este projeto ainda não possui o arquivo `pom.xml`, então faça o seguinte:
 - Supondo que você já adicionou o projeto na IDE, então:
 - clique com o botão direito do *mouse* em cima do nome do projeto;
 - escolha a opção "Propriedades";
 - na janela que abrir, em "Categorias", escolha "Bibliotecas";
 - clique no botão "Adicionar JAR/pasta";
 - vá até o diretório "lib" dentro da pasta do projeto e escolha o arquivo que está lá (que é o *driver* do PostgreSQL).



Driver PostgreSQL na IDE Eclipse

Para adicionar o *driver* faça o seguinte:

- 1) clique com o botão direito do *mouse* sobre o nome do projeto;
- 2) escolha a opção *Properties*;
- 3) na janela que abrir, em *Java Build Path* | *Libraries*;
- 4) clique no botão *Add External JARs...*;
- 5) vá até o diretório onde você salvou o *driver* e escolha o arquivo que está lá.
Feito isto, é só executar o projeto!

```
1 package Controller;
2
3 import java.sql.Connection;
4
5
6
7
8 public class Conexao {
9     // Design Pattern - Singleton
10    private static Connection conn = null;
11
12    public static Connection getConexao() throws SQLException{
13        if((conn == null) || (conn.isClosed())) {
14            conn = fabricaDeConexoes();
15        }
16
17        return conn;
18    }
19
20    // Fábrica de Conexões
21    public static Connection fabricaDeConexoes() {
22        try{
23            Class.forName("org.postgresql.Driver");
24
25            Properties prop = new Properties();
26            prop.put("user", "postgres");
27            prop.put("password", "123456");
28            prop.put("charset", "UTF-8");
29            prop.put("lc_ctype", "IS08859_1");
30
31            return DriverManager.getConnection(
32                "jdbc:postgresql://localhost:5432/Exemplo");
```



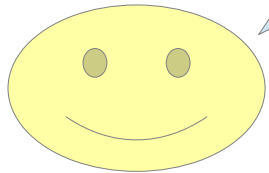
Design Pattern Singleton

- É um padrão de projeto que garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto;
- em acesso a sistemas gerenciadores de banco de dados (SBGD's), este padrão é muito útil para evitar o estouro do número de conexões do *software* com o SGBD.
- **Observação:** a conexão apresentada nos *slides* anteriores não está fazendo uso deste padrão de projeto e portanto não deve ser utilizada, senão para testes.



Exemplo de Classe de Conexão

Essa você pode usar para qualquer fim.



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class Conexao {
    private static Connection conn = null;

    // Design Pattern - Singleton
    public static Connection getConexao() throws SQLException{
        if((conn == null) || (conn.isClosed())) {
            conn = fabricaDeConexoes();
        }

        return conn;
    }

    // Fábrica de Conexões
    public static Connection fabricaDeConexoes() {
        try{
            Class.forName("org.postgresql.Driver");

            Properties props = new Properties();
            props.put("user", "postgres");
            props.put("password", "123456");
            props.put("charset", "UTF-8"); // Uso facultativo.
            props.put("lc_ctype", "ISO8859_1"); // Uso facultativo.

            return DriverManager.getConnection(
                "jdbc:postgresql://localhost:5432/ExemploConexao", props);
        }
        catch(Exception e){
            System.err.println("Erro: " + e.getMessage());
        }

        return null;
    }
}
```



Dica!

- Se você está usando o **Apache NetBeans**, então, para que o mesmo reconheça o SGBD PostgreSQL você deverá fazer a inserção da dependência abaixo no arquivo `pom.xml` do projeto:

```
<dependencies>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.18</version>
  </dependency>
</dependencies>
```

A versão tem que ser a que estiver instalada no seu computador.



Inserção de Dependência no pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <groupId>br.eti.esj</groupId>
8     <artifactId>ConexaoMVComPostgreSQL</artifactId>
9     <version>1.0-SNAPSHOT</version>
10    <packaging>jar</packaging>
11    <properties>
12        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
13        <maven.compiler.source>11</maven.compiler.source>
14        <maven.compiler.target>11</maven.compiler.target>
15    </properties>
16    <dependencies>
17        <dependency>
18            <groupId>org.postgresql</groupId>
19            <artifactId>postgresql</artifactId>
20            <version>42.2.18</version>
21        </dependency>
22    </dependencies>
23 </project>
```

Após a inserção da dependência e compilação do projeto, o driver será instalado no projeto de forma automática.



Bibliografia

- LOKHART, Thomas. **The PostgreSQL Development Team.** Disponível em: <<http://www.cis.temple.edu/~vasilis/Courses/CS33/Documentation/tutorial.pdf>> Acesso em: 05 nov. 2012.
- SALIBA JÚNIOR, Edwar. **Exemplo: Conexão do NetBeans com PostgreSQL!** Disponível em: <<http://javafree.uol.com.br/artigo/877101/Exemplo-Conexao-do-NetBeans-com-Postgre-SQL.html>> Acesso em: 16 nov. 2012.
- Wikipedia. **Singleton.** Disponível em:
- <<https://pt.wikipedia.org/wiki/Singleton>>. Acesso em: 29 Set. 2021.