

Introdução ao banco de dados relacional PostgreSQL. JDBC

Professor Isaac

Banco de Dados



Bancos de Dados



- Um banco de dados pode ser definido como um conjunto de dados devidamente relacionados.
- São **coleções organizadas** de dados que se **relacionam** de forma a criar algum sentido e dar mais eficiência durante uma pesquisa.

Bancos de Dados



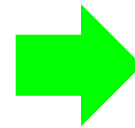
- Antigamente as empresas armazenavam informações em arquivos físicos.



Bancos de Dados



- O surgimento e evolução dos computadores possibilitaram o **armazenamento de dados de modo digital**.



Bancos de Dados



- Os BDs são de **vital importância para empresas** e se tornaram uma das principais peças dos sistemas de informação.
- Banco de Dados trata do problema de armazenamento e como recuperar eficientemente uma informação quando ela for necessária.

Modelos de Bancos de Dados



Relacional

The diagram consists of a dark blue rectangular container. Inside this container, there are two light green rectangular boxes stacked vertically. The top box contains the word 'Relacional' in blue text, and the bottom box contains the phrase 'Não relacional' in blue text.

Não relacional

Modelos de Bancos de Dados

□ Relacional

- Um banco de dados relacional organiza os dados em tabelas ou relações.
- As linhas da tabela são chamadas de registros ou tuplas.
- As colunas são chamadas de campo ou atributo.
- SQL (Structured Query Language) Linguagem de Consulta Estruturada.

Modelos de Bancos de Dados

❑ Relacional – Limitações

- O modelo relacional foi desenvolvido em torno de 1970 por Edgar F. Codd (pesquisador da IBM).
- Contudo, a quantidade de informações que precisam ser armazenadas tem crescido muito (principalmente em aplicações Web) e os bancos relacionais podem sofrer com problemas de escalabilidade*

**estar preparado para crescer / funcionar da mesma forma com mais trabalho*

Modelos de Bancos de Dados

□ Não Relacional

- A principal motivação para banco de dados não relacionais é a busca para resolução do problema de escalabilidade.
- São modelados de formas diferentes das relações tabulares usadas nos bancos de dados relacionais.
- NoSQL (Not only SQL).

Sistemas de Gerenciamento de Banco de Dados (SGBDs)

- Um **SGBD** (Data Base Management System - DBMS) é o conjunto de softwares responsáveis pelo gerenciamento de um banco de dados.
- O **SGBD** disponibiliza uma interface para que seus clientes possam incluir, alterar ou consultar dados previamente armazenados.

Sistemas de Gerenciamento de Banco de Dados (SGBDs)

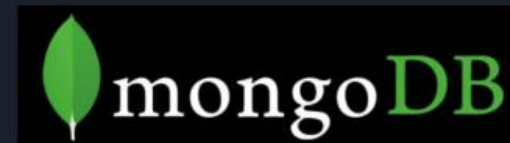
Relacionais



Oracle Database



Não Relacionais



Modelo Relacional para Bancos de Dados

- Bancos de dados Relacionais são compostos por tabelas:
 - Linhas são chamadas de registros
 - Colunas são os campos ou atributos
 - O primeiro campo é geralmente uma chave primária:
 - Única para cada registro
 - Pode ser mais de um campo ou nem ser necessária

Exemplo de Modelo Relacional para B.D.

Table: **Employee**

Number	Name	Department	Salary	Location
23603	JONES, A.	413	1100	NEW JERSEY
24568	KERWIN, R.	413	2000	NEW JERSEY
34589	LARSON, P.	642	1800	LOS ANGELES
35761	MYERS, B.	611	1400	ORLANDO
47132	NEUMANN, C.	413	9000	NEW JERSEY
78321	STEPHENS, T.	611	8000	ORLANDO

registro →

Chave primária ↑

Campo ↑

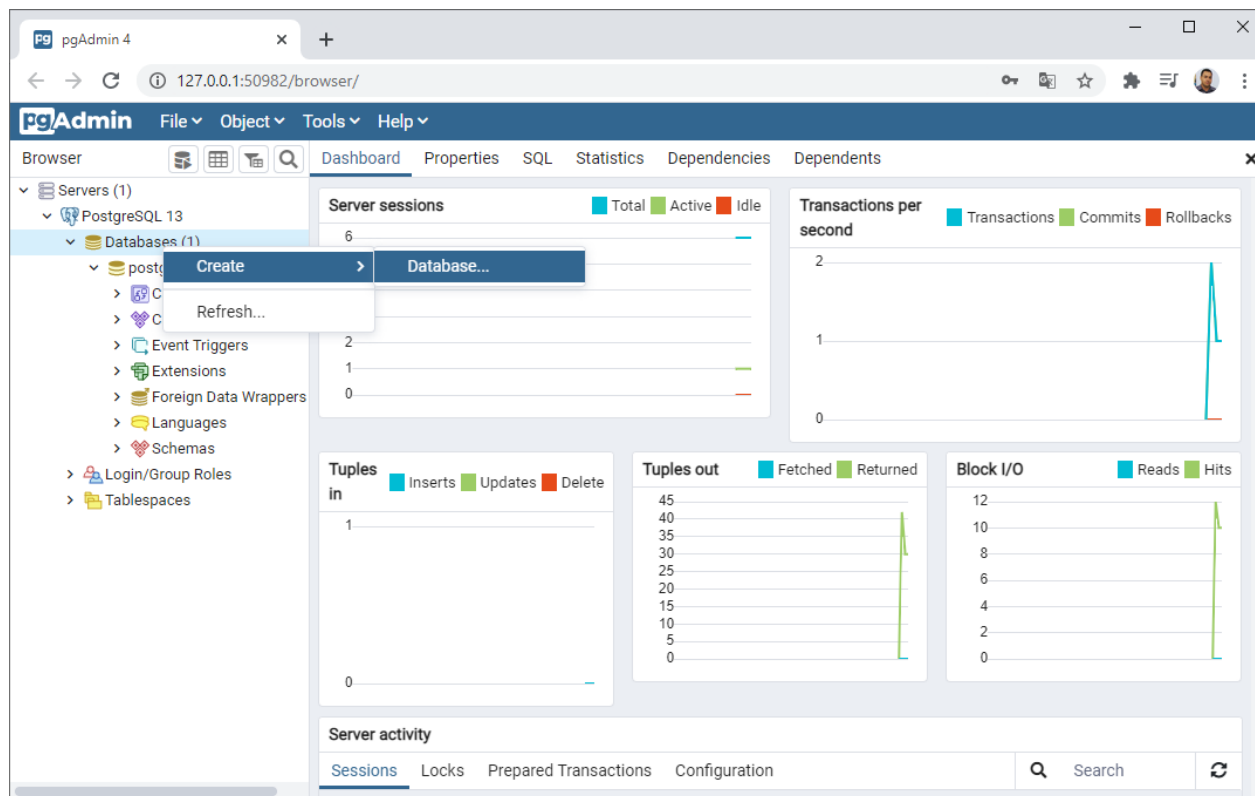
Relação entre BD e POO

- Tabela \leftrightarrow Classe
- Registro \leftrightarrow Objeto
- Campo \leftrightarrow Atributo

Criando um Banco de Dados

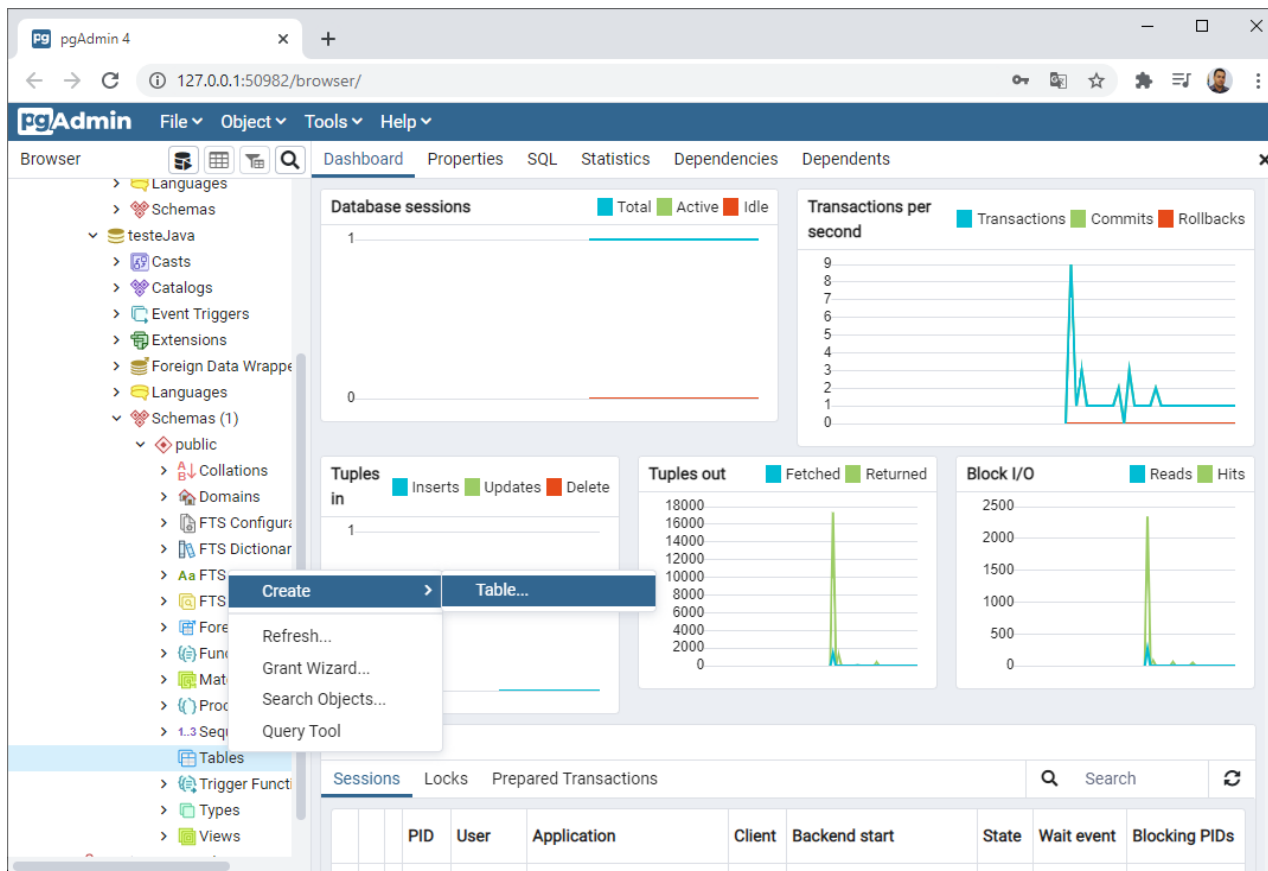
Criando um Banco de Dados

- Criaremos o DB usando a interface do PostgreSQL



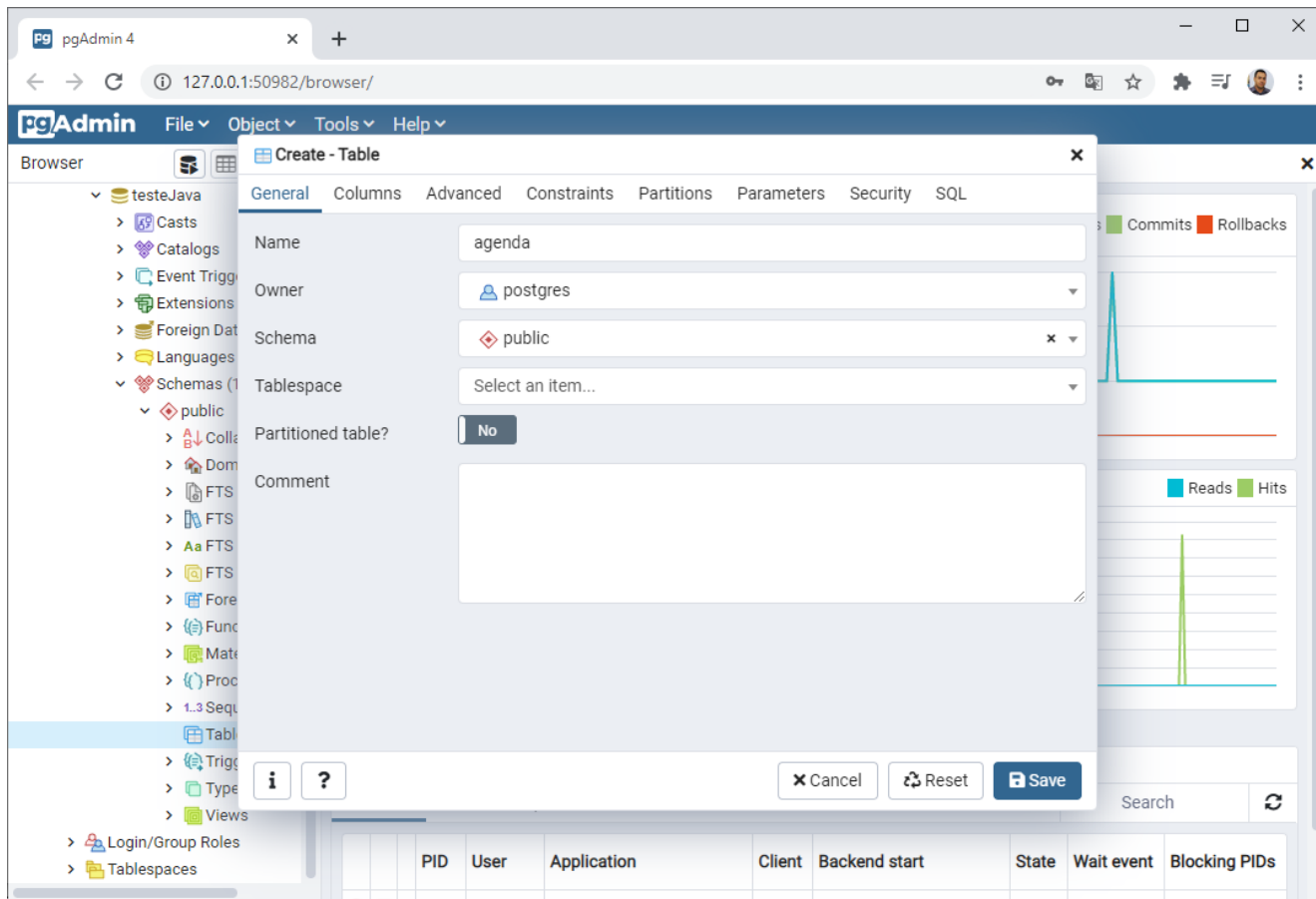
Criando uma Tabela no DB

- Crie uma Tabela no seu DB



Criando uma Tabela no DB

- Atribua um nome a sua Tabela

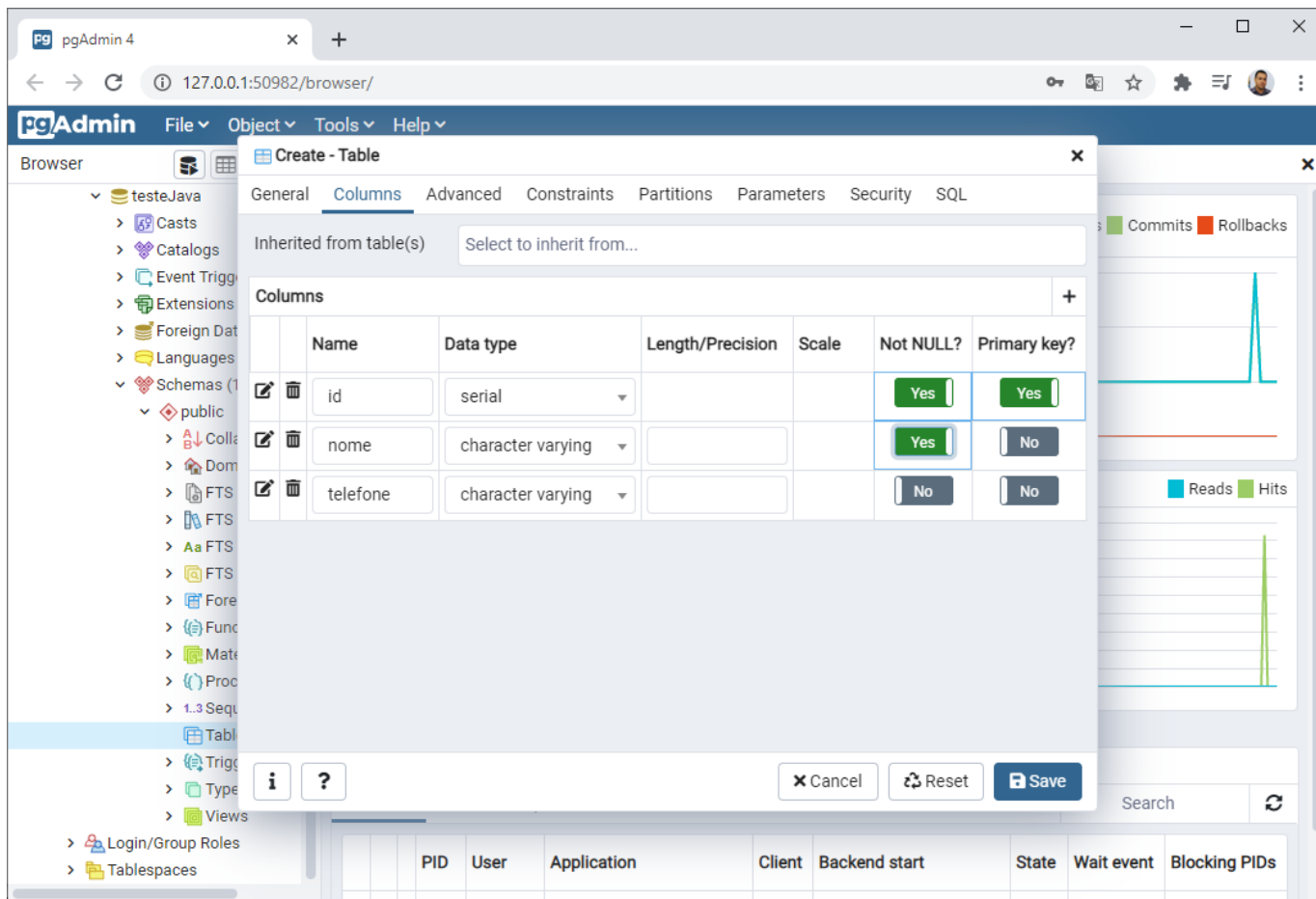


Criando uma Tabela no DB

- Crie as colunas da sua Tabela:
- Neste exemplo criaremos 3 colunas
 - Id
 - O id não pode ter valor nulo, nem repetição. Portanto será uma chave primária.
 - Nome
 - Não pode ser nulo.
 - Telefone
 - Pode ser nulo.

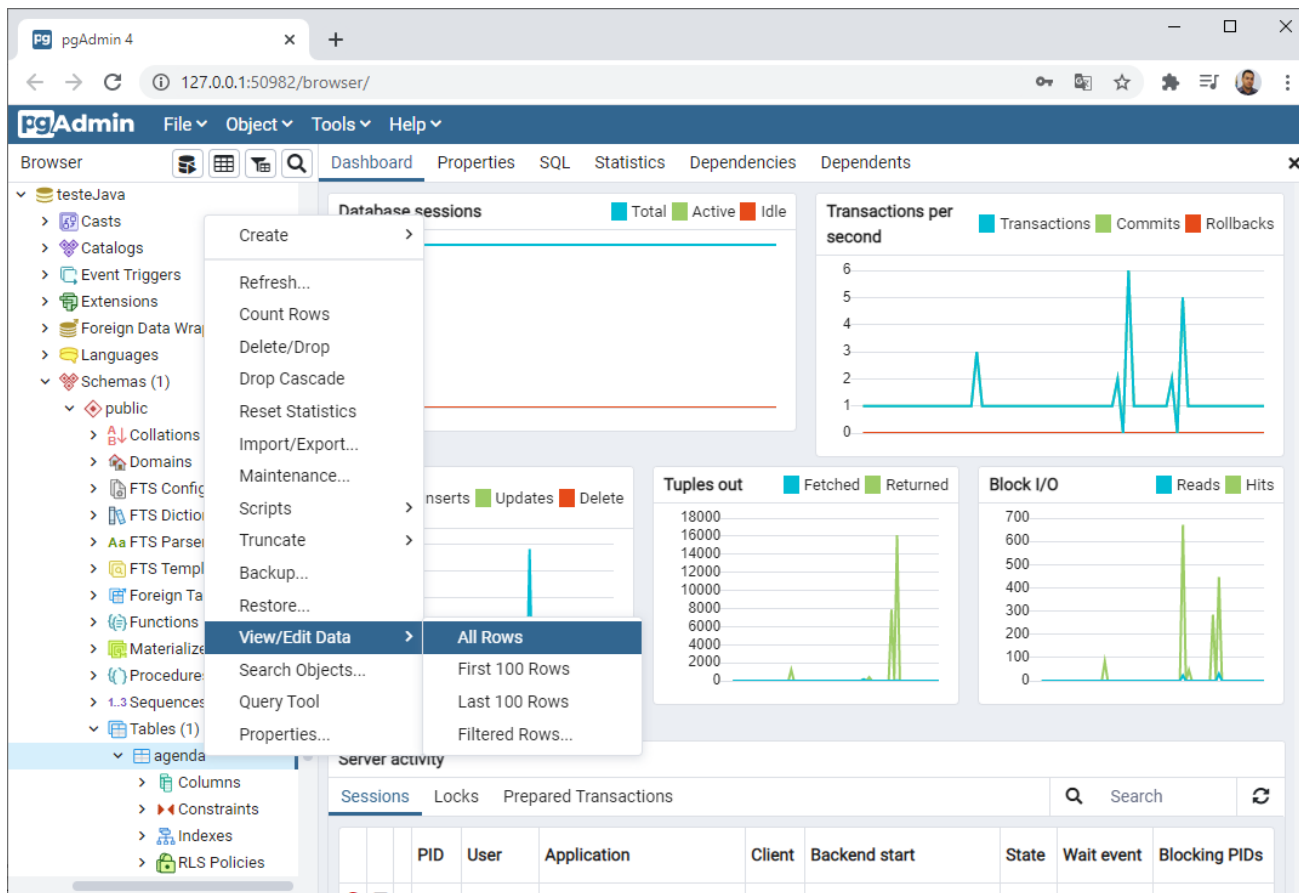
Criando uma Tabela no DB

- Crie as colunas da sua Tabela



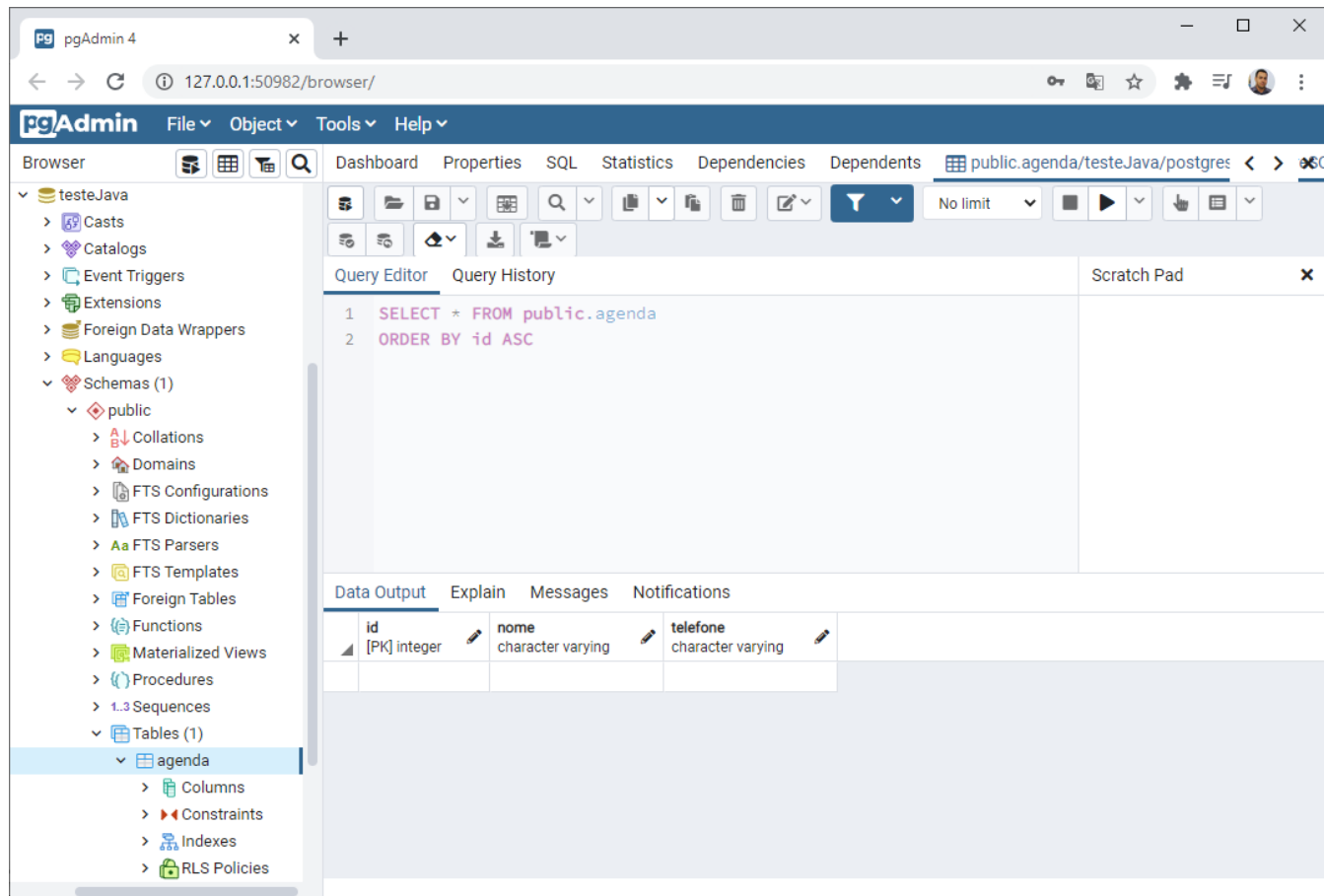
Criando uma Tabela no DB

- Agora verifique a sua Tabela



Criando uma Tabela no DB

- Como a Tabela acabou de ser criada, está vazia.

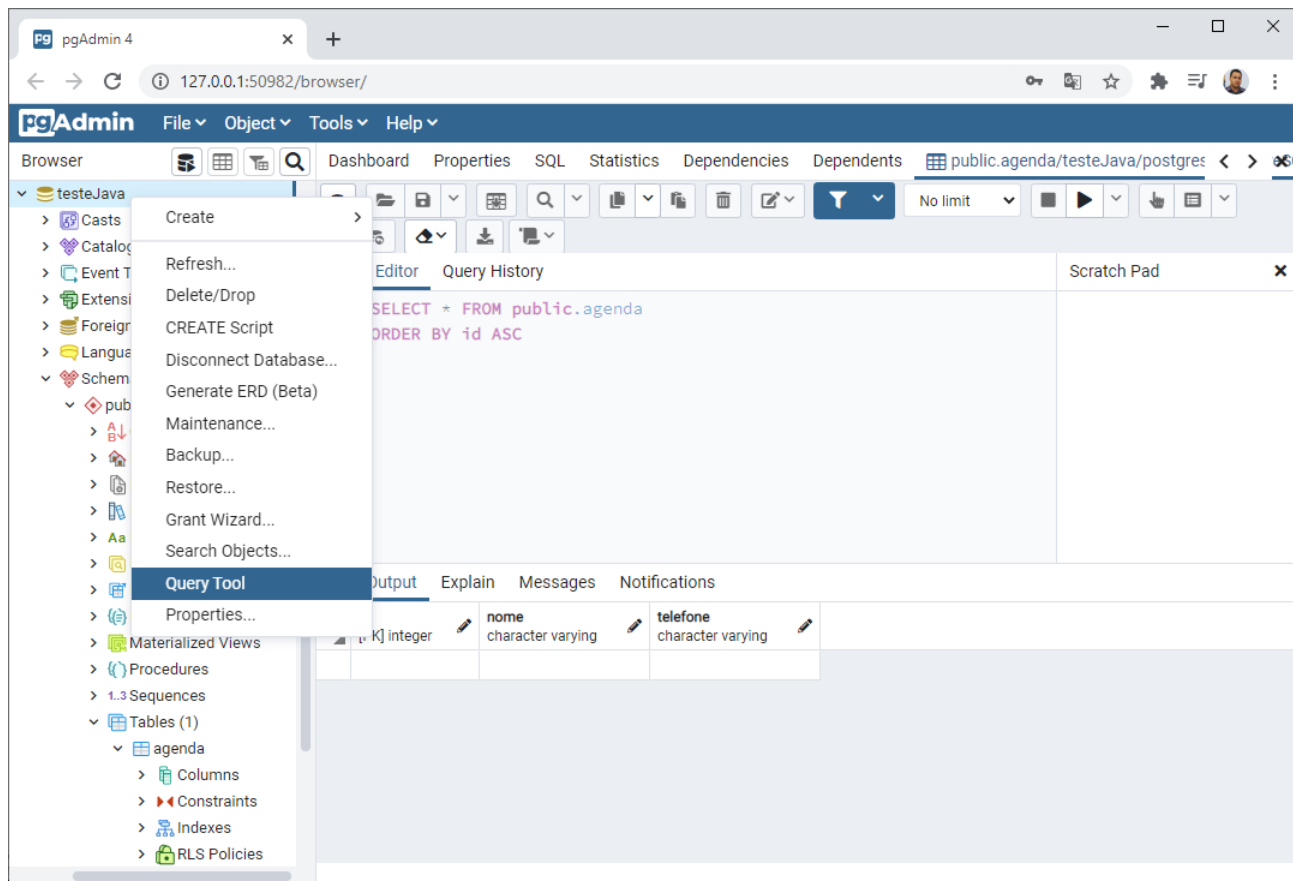


Inserindo dados no Banco de Dados



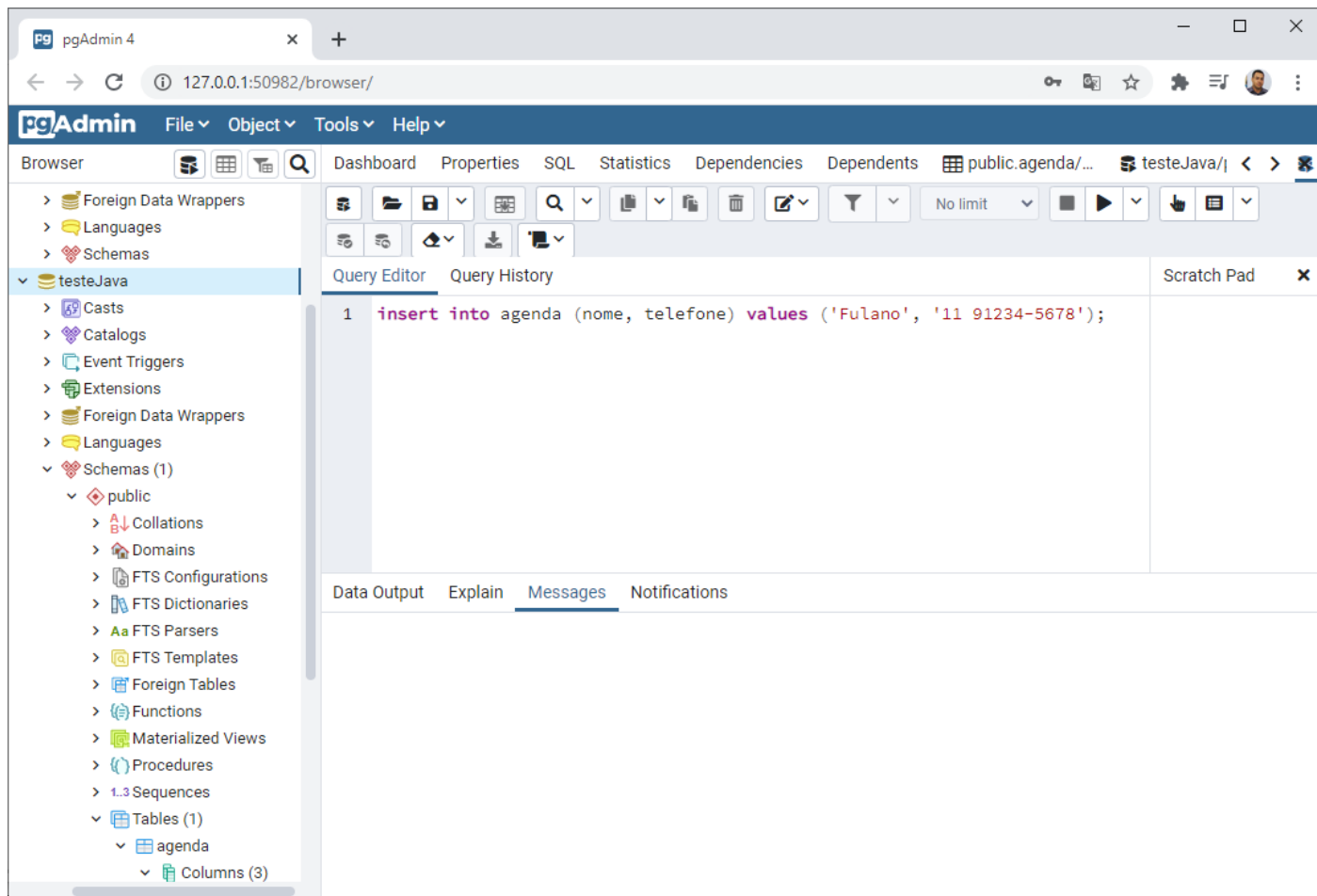
Inserindo dados no DB

- Para inserir dados nas suas tabela selecione o *Query Tool*.



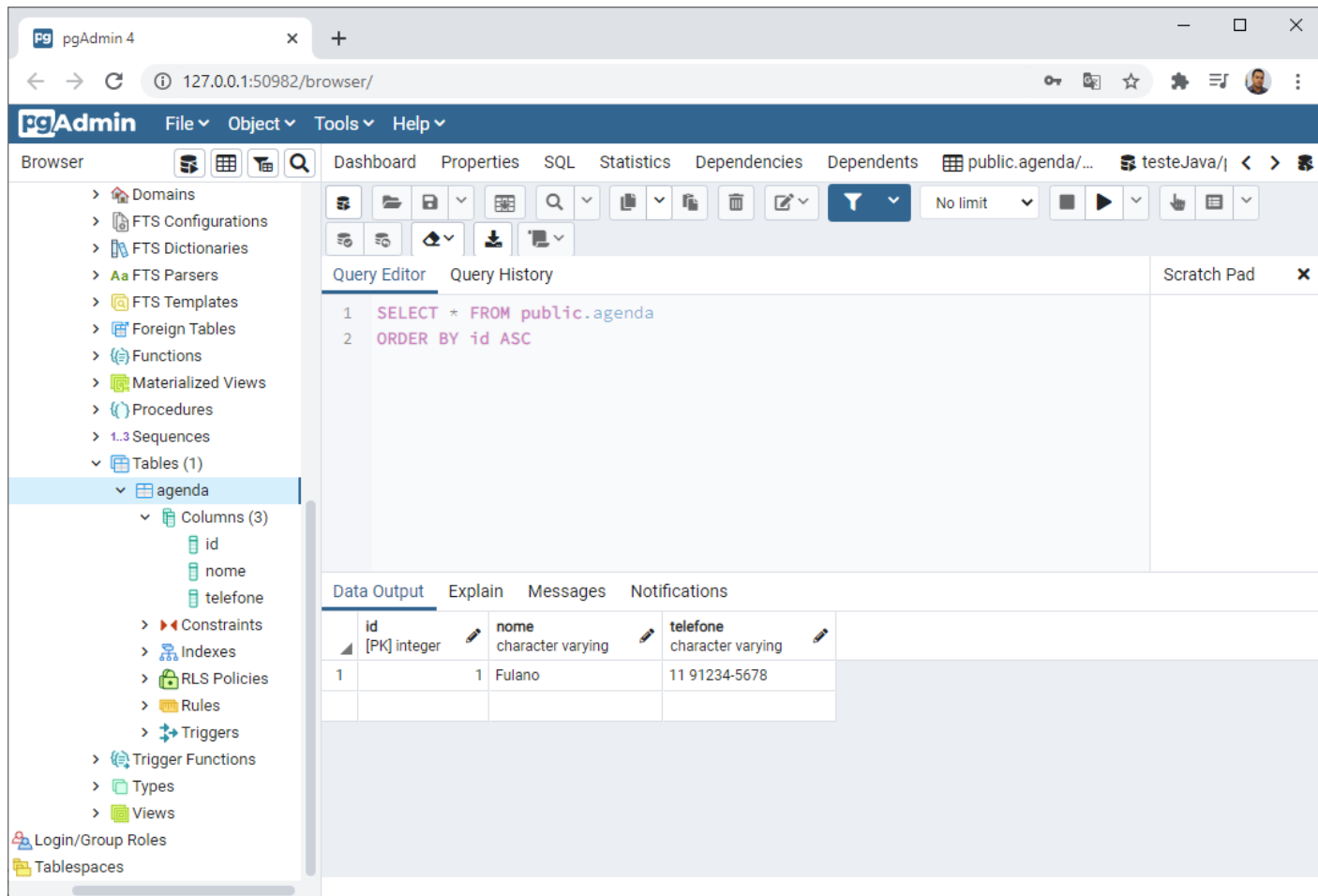
Inserindo dados no DB

- Escreva o comando em SQL para inserir um registro na Tabela, e execute o comando.



Inserindo dados no DB

- Verifique que agora já tem dados na sua Tabela.



The screenshot shows the pgAdmin 4 web interface in a browser. The left sidebar displays the database structure, with the 'agenda' table selected under 'Tables (1)'. The main panel shows the 'Query Editor' with the following SQL query:

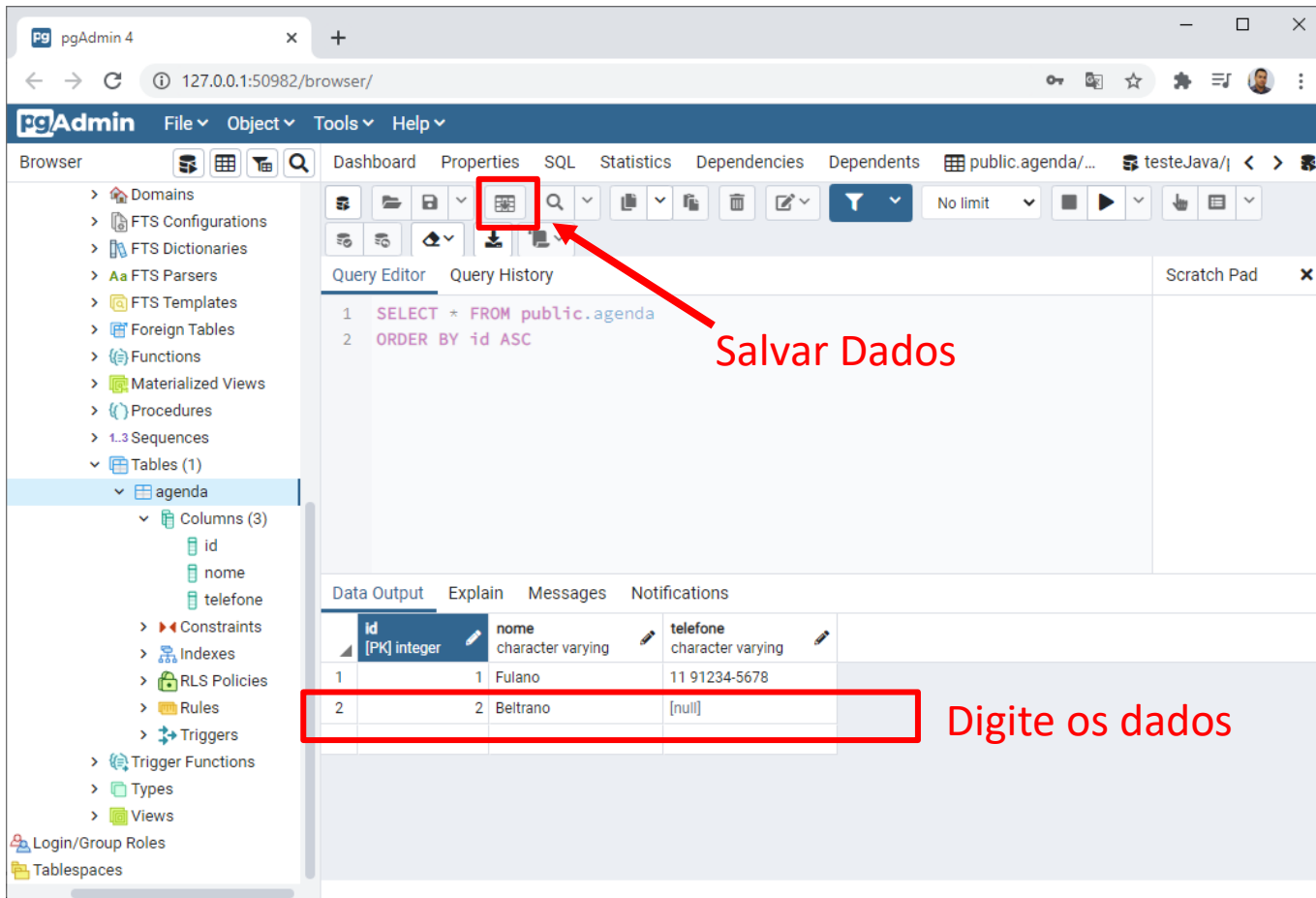
```
1 SELECT * FROM public.agenda
2 ORDER BY id ASC
```

Below the query editor, the 'Data Output' tab is active, displaying the results of the query in a table format:

id	nome	telefone
1	Fulano	11 91234-5678

Inserindo dados no DB

- Você também pode digitar dados na sua Tabela e clicar em Salvar Dados.



The screenshot shows the pgAdmin 4 web interface. The left sidebar displays a tree view of the database structure, with the 'agenda' table selected under 'Tables (1)'. The main panel shows the 'Query Editor' with a SQL query: `SELECT * FROM public.agenda ORDER BY id ASC`. A red arrow points from the text 'Salvar Dados' to a button in the toolbar. Below the query editor, the 'Data Output' tab is active, displaying a table with columns 'id', 'nome', and 'telefone'. The table has two rows: one with '1', 'Fulano', and '11 91234-5678', and another with '2', 'Beltrano', and '[null]'. A red box highlights the second row, and a red arrow points from the text 'Digite os dados' to the empty cells in that row.

pgAdmin 4

127.0.0.1:50982/browser/

pgAdmin File Object Tools Help

Browser Dashboard Properties SQL Statistics Dependencies Dependents public.agenda/... testeJava/...

Query Editor Query History Scratch Pad

```
1 SELECT * FROM public.agenda
2 ORDER BY id ASC
```

Salvar Dados

Data Output Explain Messages Notifications

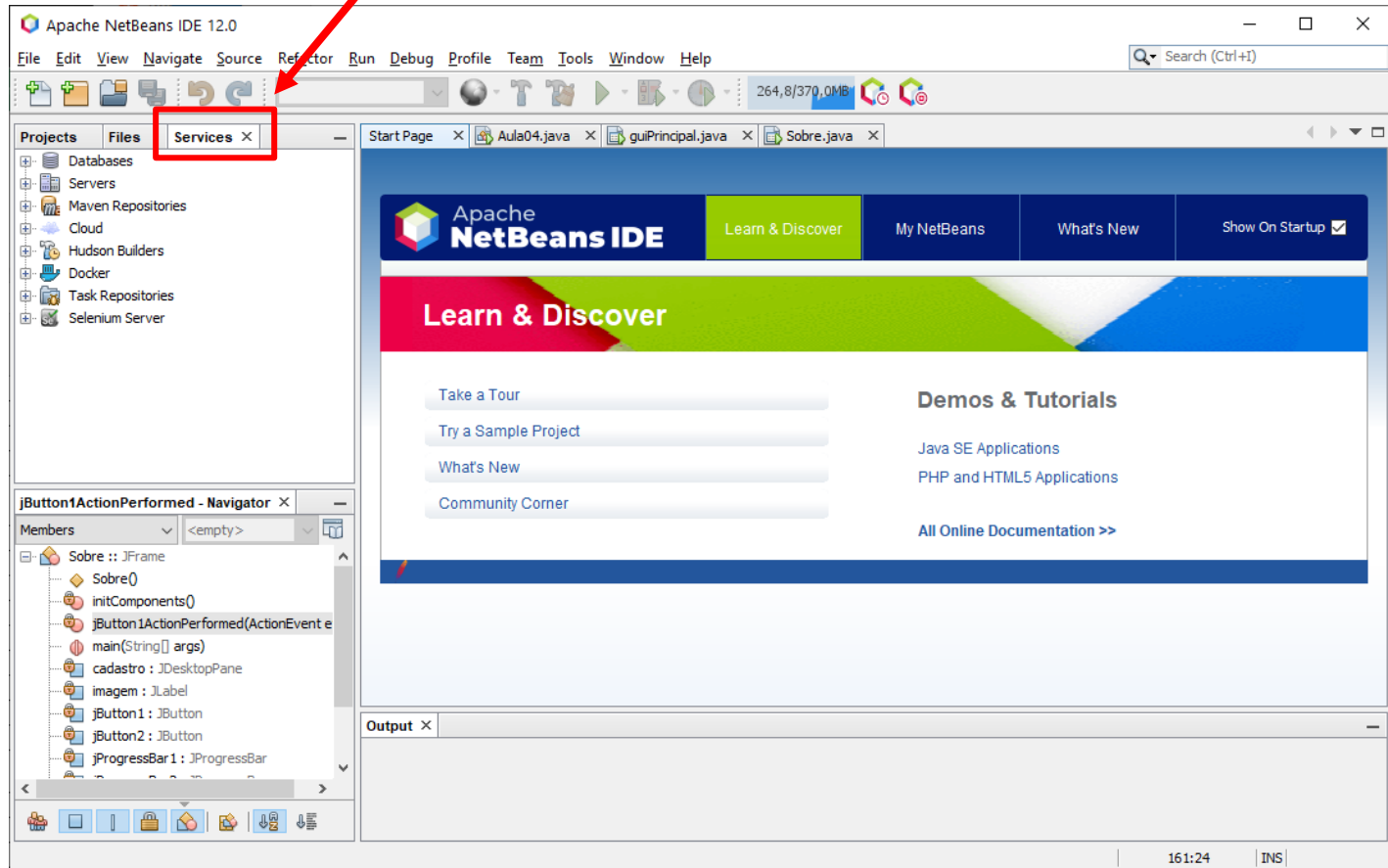
	id [PK] integer	nome character varying	telefone character varying
1	1	Fulano	11 91234-5678
2	2	Beltrano	[null]

Digite os dados

Conectando no DB com o NetBeans

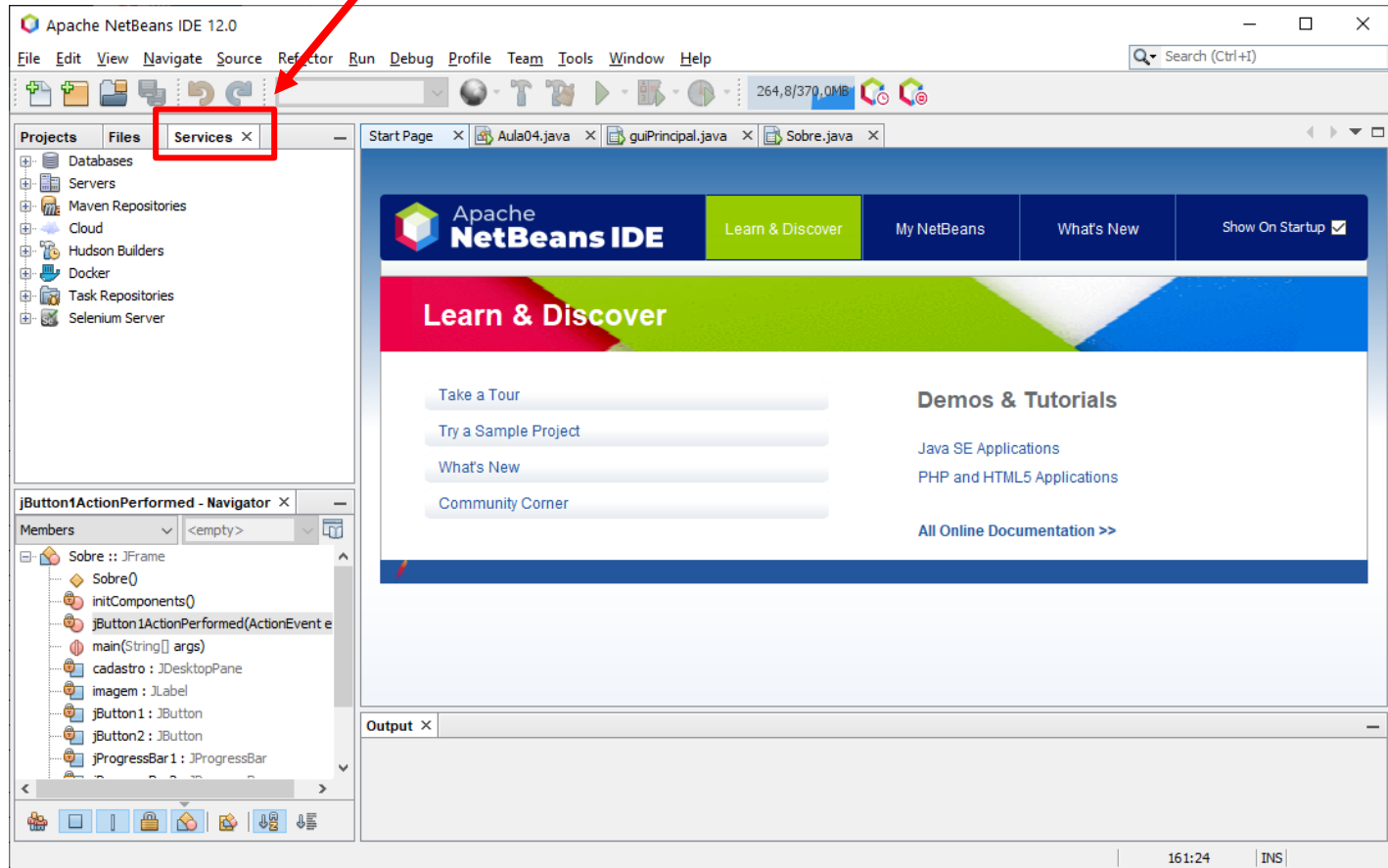
Conectando no DB com o NetBeans

- Selecione a Aba *Services*



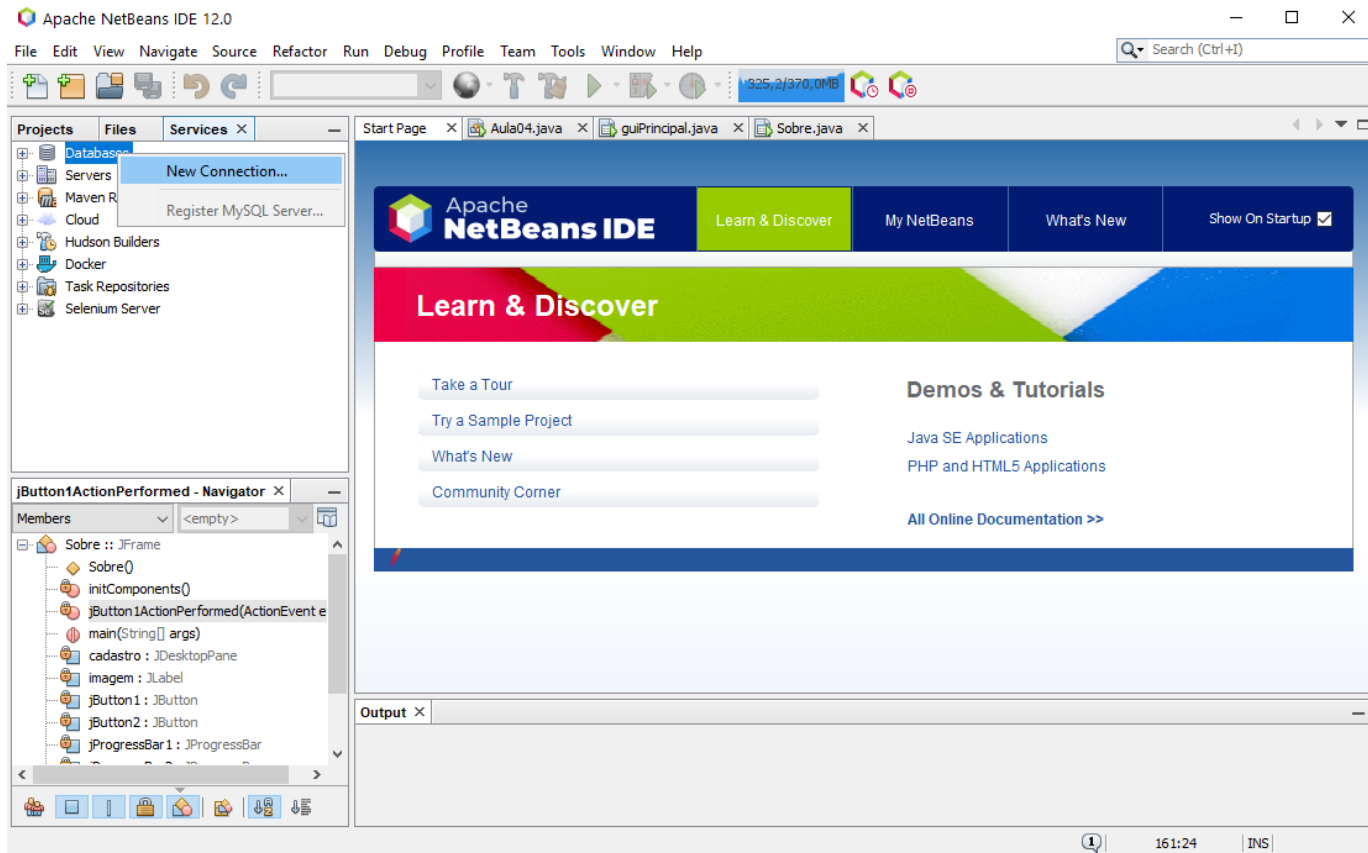
Conectando no DB com o NetBeans

- Selecione a Aba *Services*



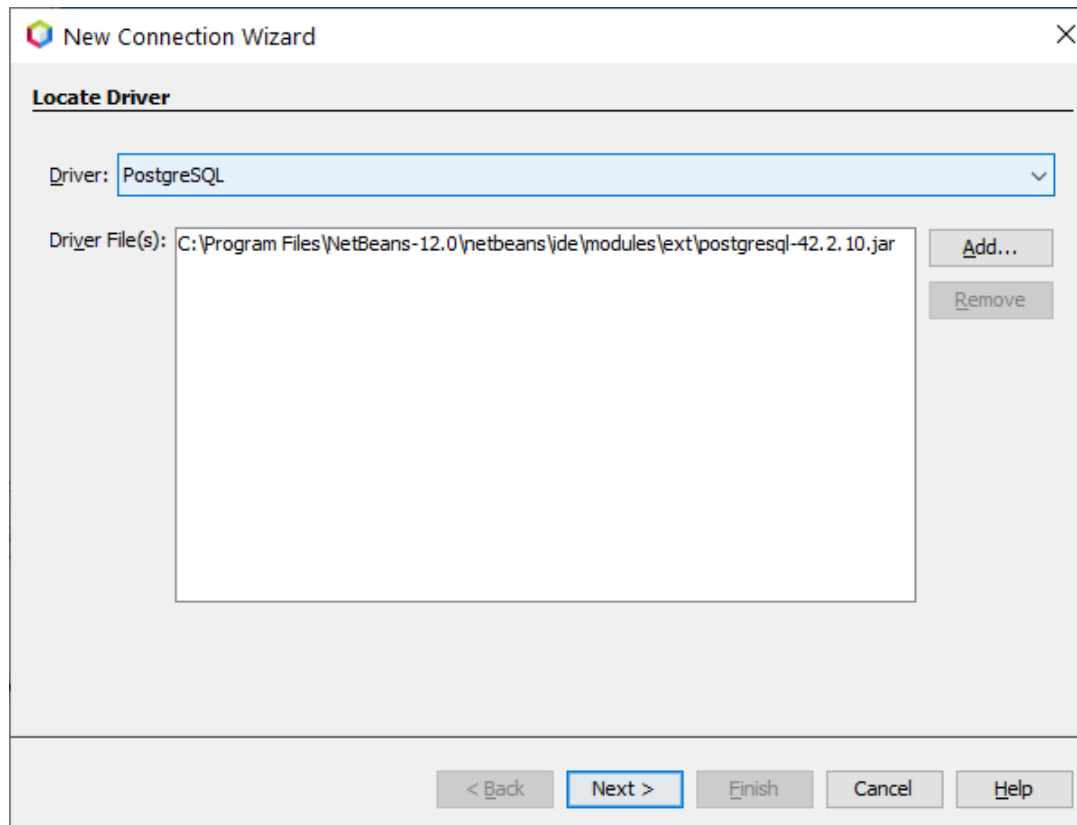
Conectando no DB com o NetBeans

- Clique com o botão Direito em Databases -> New Connection



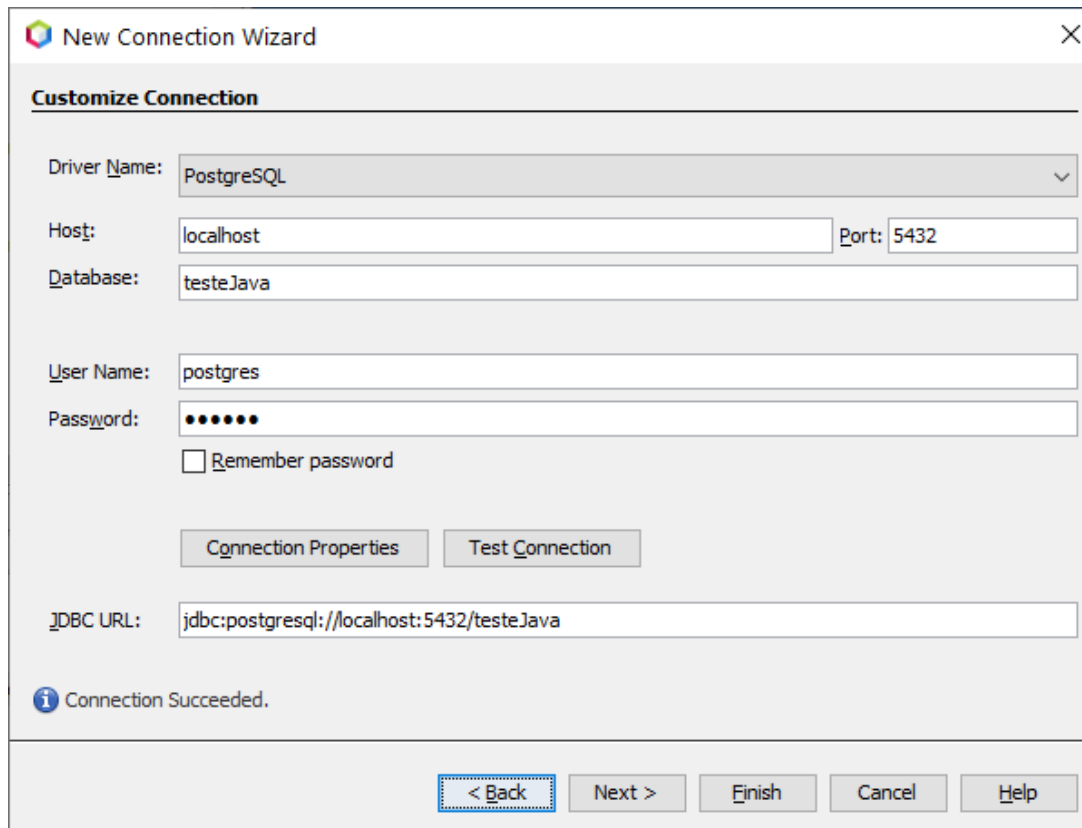
Conectando no DB com o NetBeans

- Seleccione o Driver PostgreSQL



Conectando no DB com o NetBeans

- Coloque a senha e o Database que você criou.
- Clique em Test Connection
 - *Veja que o teste conectou com sucesso*



The screenshot shows the 'New Connection Wizard' dialog box in NetBeans, specifically the 'Customize Connection' step. The dialog has a title bar with the NetBeans logo and a close button. The main area contains several input fields and buttons. The 'Driver Name' is set to 'PostgreSQL'. The 'Host' is 'localhost' and the 'Port' is '5432'. The 'Database' is 'testeJava'. The 'User Name' is 'postgres' and the 'Password' is masked with dots. There is a checkbox for 'Remember password' which is unchecked. Below these fields are two buttons: 'Connection Properties' and 'Test Connection'. At the bottom, the 'JDBC URL' is displayed as 'jdbc:postgresql://localhost:5432/testeJava'. A status bar at the bottom indicates 'Connection Succeeded.' with an information icon. The bottom of the dialog has a set of navigation buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The '< Back' button is highlighted with a blue dashed border.

New Connection Wizard

Customize Connection

Driver Name: PostgreSQL

Host: localhost Port: 5432

Database: testeJava

User Name: postgres

Password: •••••

☐ Remember password

Connection Properties Test Connection

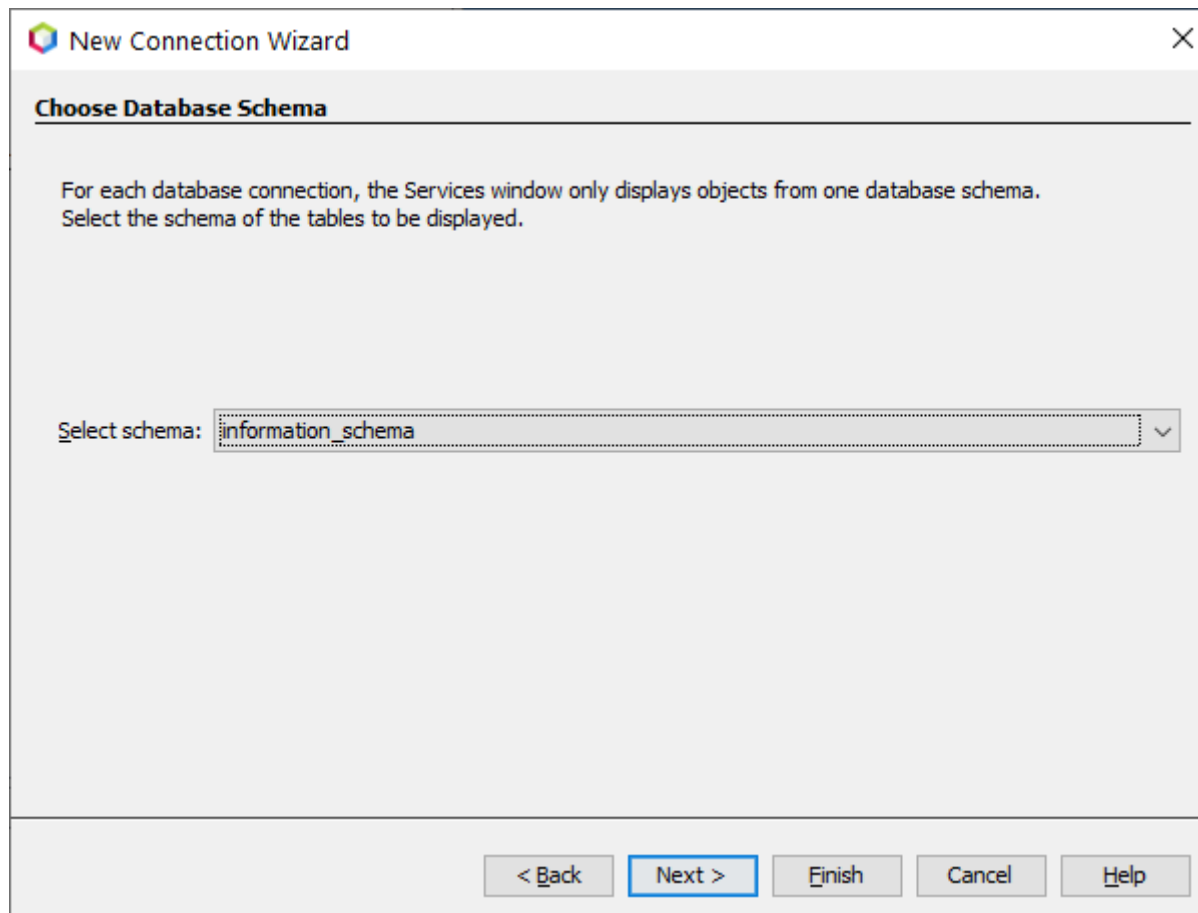
JDBC URL: jdbc:postgresql://localhost:5432/testeJava

Connection Succeeded.

< Back Next > Finish Cancel Help

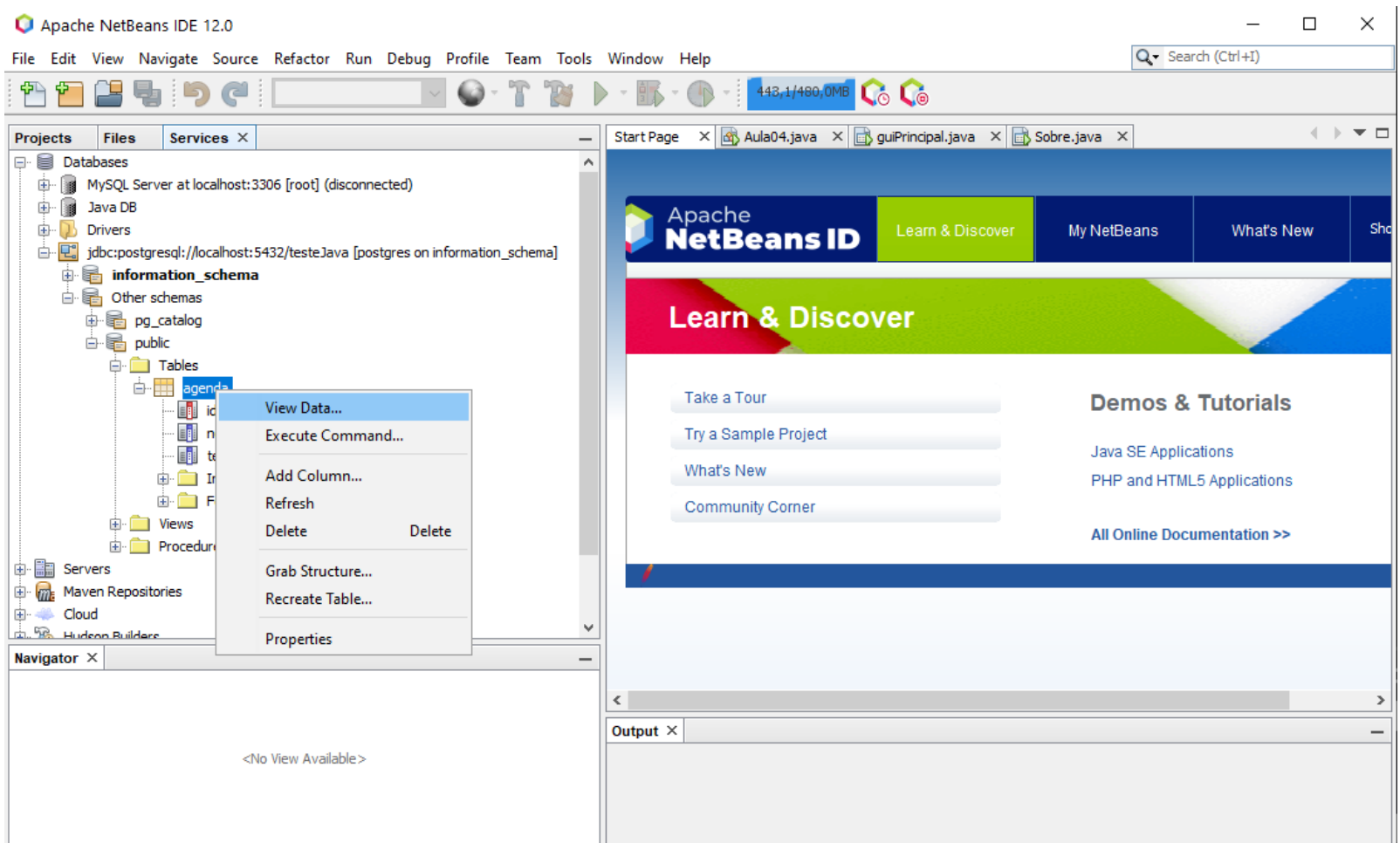
Conectando no DB com o NetBeans

- clique em *next* e depois *Finish*.



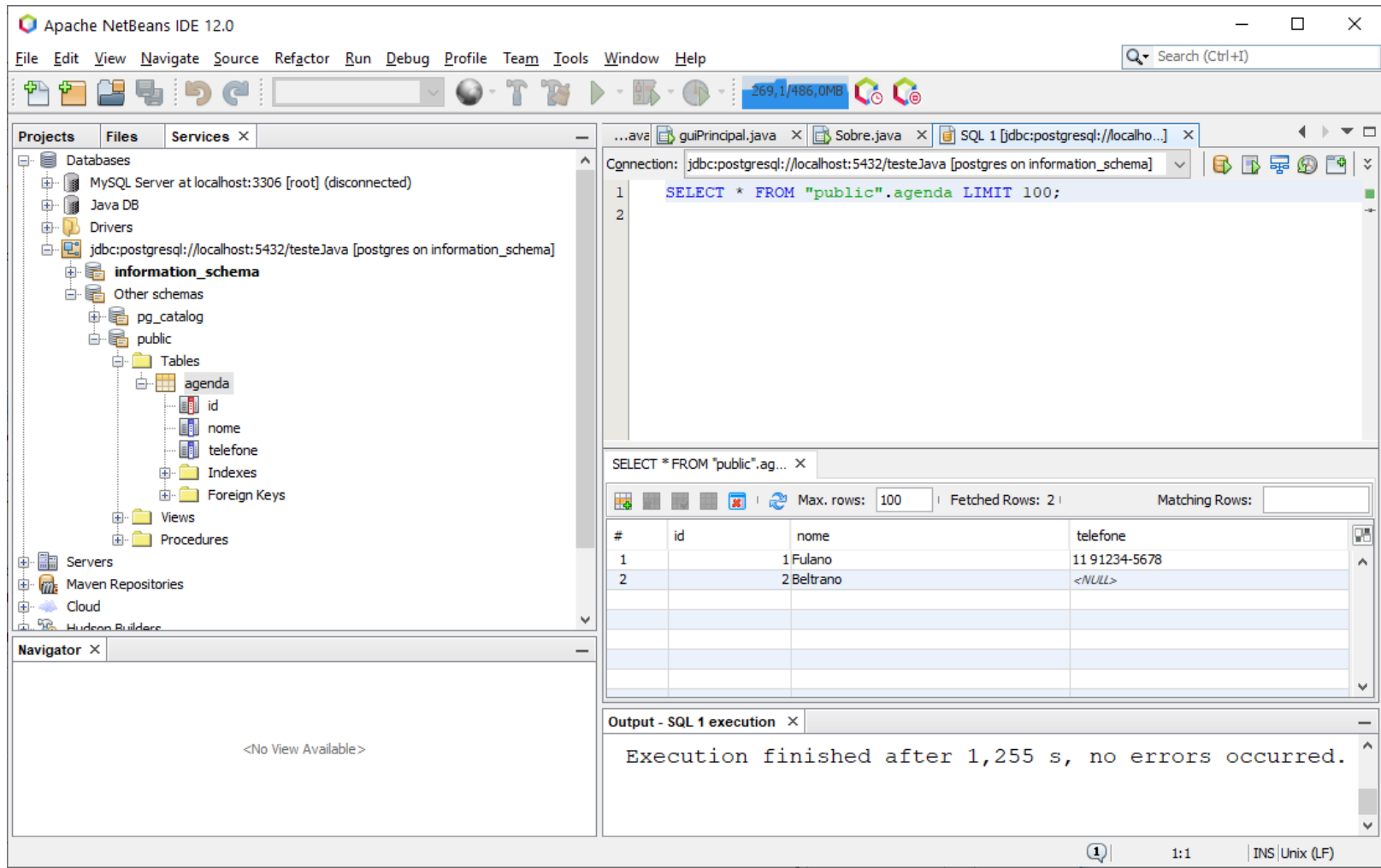
Conectando no DB com o NetBeans

- Agora você consegue visualizar suas Tabelas



Conectando no DB com o NetBeans

- Agora você consegue visualizar suas Tabelas



The screenshot shows the Apache NetBeans IDE 12.0 interface. The left sidebar displays the 'Projects' and 'Files' view, with the 'Databases' section expanded. Under 'Databases', there is a connection to 'jdbc:postgresql://localhost:5432/testeJava [postgres on information_schema]'. The 'information_schema' is expanded, showing 'pg_catalog' and 'public' schemas. The 'public' schema is further expanded, showing 'Tables' and 'Indexes'. The 'agenda' table is selected, showing its columns: 'id', 'nome', and 'telefone'.

The main editor area shows the SQL query: `SELECT * FROM "public".agenda LIMIT 100;`. The 'Output - SQL 1 execution' window at the bottom displays the results of the query:

#	id	nome	telefone
1	1	Fulano	11 91234-5678
2	2	Beltrano	<NULL>

The status bar at the bottom indicates '1:1' and 'INS | Unix (LF)'.

Acesso a Banco de Dados usando Java

Conexão direta com o Banco de Dados

- Protocolo é um conjunto de normas que estabelece a maneira de se comunicar com o Banco de Dados (BD).
- A conexão com o BD pode ser feita diretamente através do seu protocolo proprietário de conexão.
- Cada fabricante de BD tem o seu protocolo proprietário: Oracle, MySQL, SQL Server, etc.
- Nesse caso é necessário conhecer o protocolo de conexão de cada fabricante de BD.

Conexão com BD usando um “intermediário”

- Linguagens de programação fornecem bibliotecas que já tem os protocolos dos diferentes fabricantes de BD.
- Nesse caso não é necessário conhecer os diferentes protocolos, apenas as classes e seus métodos.
- No caso do Java vamos utilizar a biblioteca **JDBC**
(*Java DataBase Connectivity*)

Java DataBase Connectivity (JDBC)

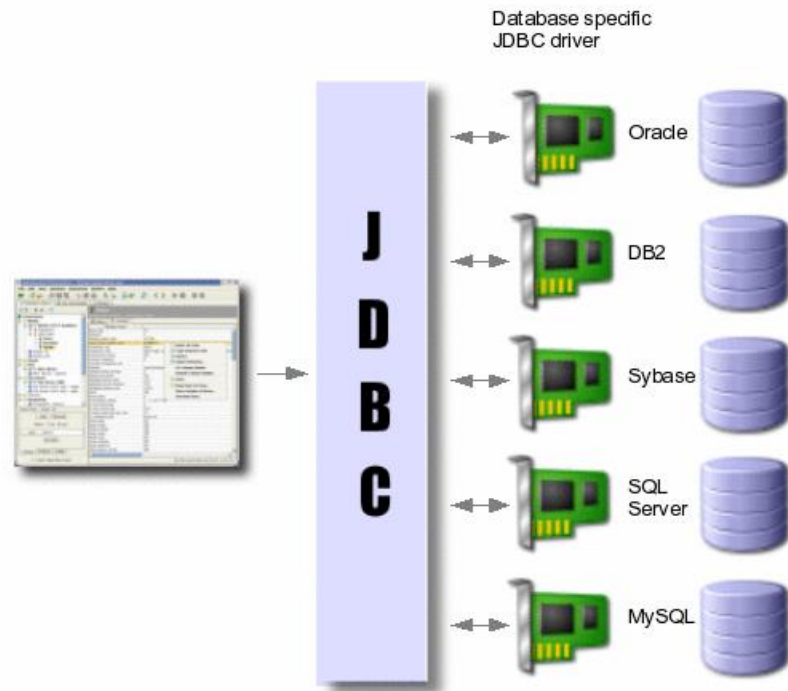
- *Application Programming Interface (API) que possibilita o acesso a BD através de um modelo de conexão uniforme.*
- Biblioteca implementada em Java que disponibiliza classes e interfaces que provê um padrão para tornar possível, aos desenvolvedores de aplicações e ferramentas a construção de software que acesse banco de dados.
- Com o JDBC, é possível que aplicações Java possam acessar dados mantidos por um Sistema de Gerenciador de Banco de Dados local ou remoto.

Java DataBase Connectivity (JDBC)

- Para cada BD existe uma implementação que leva em consideração as particularidades daquele BD.
- O Java suporta integração com diversos BD diferentes, cada fornecedor disponibiliza seu próprio driver JDBC.

Drivers JDBC

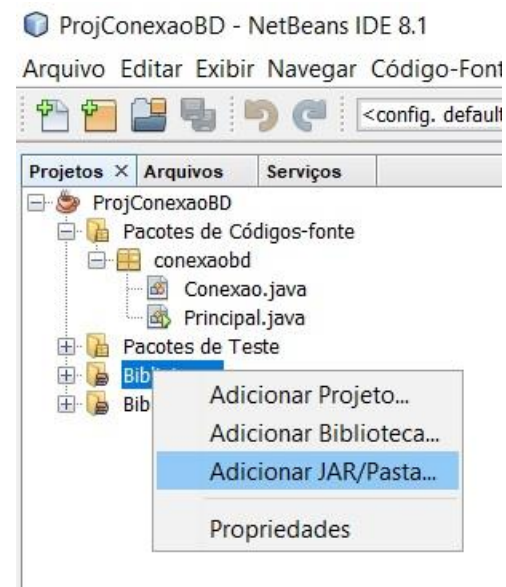
- Conjunto de classes que disponibilizam métodos para se comunicar com o BD
- Os principais BD possuem drivers JDBC que podem ser utilizados na linguagem JAVA



Adicionar um driver no projeto do NetBeans

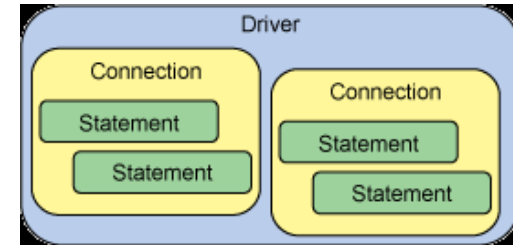
- Clique com o botão direito na categoria Biblioteca
- Clique no botão Adicionar JAR/Pasta
- Escolha o arquivo do driver disponibilizado pelo fabricante

No caso do Oracle iremos utilizar o arquivo ojdbc6.jar



Pacote java.sql

- Fornece a API para acesso e processamento de dados
- Principais classes e interfaces:
 - **DriverManager**: responsável por criar uma conexão com o banco de dados
 - **Connection**: classe responsável por manter uma conexão aberta com o banco
 - **Statement**: gerencia e executa instruções SQL
 - **PreparedStatement**: gerencia e executa instruções SQL, permitindo também a passagem de parâmetros em uma instrução
 - **ResultSet**: responsável por receber os dados obtidos em uma pesquisa ao banco



Importar Bibliotecas

- `import java.sql.Connection;`
- `import java.sql.DriverManager;`
- `import java.sql.Statement;`
- `import java.sql.PreparedStatement;`
- `import java.sql.ResultSet;`
- `import java.sql.SQLException;`

Java DataBase Connectivity (JDBC)

- Para se comunicar com o BD, deverá ser executado 5 passos:
 - Carregar o Driver do BD.
 - Criar uma conexão com o BD.
 - Criar o Statement.
 - Executar as queries.
 - Fechar a conexão com o BD.

Simple exemplo dos 5 passos.

Passo 1: Registrando o Driver

- Carrega um driver de banco de dados e cria a URL de conexão Oracle, através do método **forName()** da classe Class:

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

Passo 2: Criando Objeto de Conexão

- O método **getConnection()** da classe DriverManager é usado para estabelecer conexão com o BD:

```
conn=DriverManager.getConnection(  
"jdbc:postgresql://localhost:5432/testeJava", "postgres", "123456");
```

Passo 3: Criando o objeto Statement

- O método **prepareStatement()** da interface de conexão é usado para criar o Statement. O objeto conn é responsável por executar as queries no BD:

```
pstmt=conn.prepareStatement("SELECT * FROM DEPT ORDER BY DEPTNO");
```

Passo 4: Executando Query

- Os métodos `executeQuery()` ou `execute()` do objeto `pstmt` é usado para executar as queries no BD. Este método retorna um objeto do tipo `ResultSet` que pode ser usado para obter os registros de uma tabela DB.

```
rs=pstmt.executeQuery();
```

Passo 5: Fechando a conexão

- Ao fechar o objeto de conexão, o ResultSet será fechado automaticamente.
- O método `close()` da interface Connection é usado para fechar a conexão.

```
conn.close();
```

DriverManager

- Responsável pelo gerenciamento de drivers JDBC e quem estabelece conexões a bancos de dados
- Carrega um driver de banco de dados e cria a URL de conexão
- Oracle:
`Class.forName("org.postgresql.Driver");`
URL: `jdbc:postgresql://[servername]:[port]/database name]`
- Exemplo:
`DriverManager.getConnection("jdbc:postgresql://localhost:5432/testeJava",
"postgres", "123456");`

DriverManager – Outros Exemplos

- **MySQL:** `Class.forName("com.mysql.jdbc.Driver");`
URL: `jdbc:mysql://hostname/ databaseName`
- **Postgre SQL:** `Class.forName("org.postgresql.Driver");`
URL: `jdbc:postgresql://[servername]:[port]/database name]`
- **ODBC:** `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
URL: `jdbc:odbc:[DSN_NAME]`
- **DB2:** `Class.forName ("COM.ibm.db2.jdbc.net.DB2Driver");`
URL: `jdbc:db2:hostname:port Number/databaseName`
- **SyBase:** `Class.forName ("com.sybase.jdbc.SybDriver");`
URL: `jdbc:sybase:Tds:hostname: port Number/databaseName`

Connection

- Mantem aberta a conexão com o banco de dados
- Proporciona informações sobre as tabelas do banco através de transações
- Principais Métodos:
 - `commit()`: executa todas as alterações feitas com o banco de dados pela atual transação
 - `rollback()`: desfaz qualquer alteração feita com o banco de dados pela atual transação
 - `close()`: libera o recurso que estava sendo utilizado pelo objeto
 - `prepareStatement()`: recebe a string com o comando SQL que será executado
- Exemplo:

```
conn.prepareStatement("DELETE FROM DEPT WHERE DEPTNO = ?");
```


Statement

- Implementação de uma Interface que fornece métodos para executar uma instrução SQL
- Não aceita a passagem de parâmetros
- Principais métodos:
 - `executeUpdate()`: executa instruções SQL do tipo INSERT, UPDATE e DELETE
 - `execute()`: executa instruções SQL de busca de dados do tipo SELECT
 - `close()`: libera o recurso que estava sendo utilizado pelo objeto
- Exemplos:
 - `stmt.executeUpdate("DELETE FROM DEPT WHERE DEPTNO = 10");`
 - `stmt.execute ("SELECT * FROM DEPT ");`

PreparedStatement

- Possui todos os recursos da interface Statement acrescentando a utilização de parâmetros em uma instrução SQL
- Principais métodos:
 - setDate(): atribui um valor do tipo Data
 - setInt(): atribui valores do tipo inteiro
 - setString(): atribui valores do tipo String
 - execute(): executa instruções SQL do tipo INSERT, UPDATE e DELETE
 - executeQuery(): executa instruções SQL de busca de dados do tipo SELECT
- Exemplos:
 - pstmt.setString(1, "MANAGEMENT");
 - pstmt.setInt(2, 10);
 - pstmt.execute();

ResultSet

- Permite o recebimento e gerenciamento do conjunto de dados resultante de uma consulta SQL com métodos capazes de acessar os dados
- Principais métodos:
 - `next()`: move o cursor para a próxima linha de dados, já que o conjunto de dados retornados pela consulta SQL é armazenado em uma estrutura de lista
 - `getInt(CAMPO)`: utilizado para pegar valores do tipo inteiro
 - `getString(CAMPO)`: método utilizado para pegar valores do tipo String
 - `close()`: libera o recurso que estava sendo utilizado pelo objeto
- Exemplos:
 - `while (rs.next())`
 - `rs.getString("DNAME");`

Tipos de dados entre SQL e Java.

- Os tipos comuns do SQL são equivalentes a tipos primitivos do Java.
- Outros tipos tais como: DATE, TIME, TIMESTAMP; são suportados por classes específicas tais como:

`java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`.

Tipos de dados entre SQL e Java.

Tipo SQL	Tipo Java
BIGINT	long
BINARY	byte[]
BIT	boolean
BLOB	java.sql.Blob
CHAR	java.lang.String
CLOB	java.sql.Clob
DATE	java.sql.Date
DECIMAL	java.math.BigDecimal
DOUBLE	double
FLOAT	double
INT, INTEGER	int
LONGVARBINARY	byte[]
NUMBER	Int, double

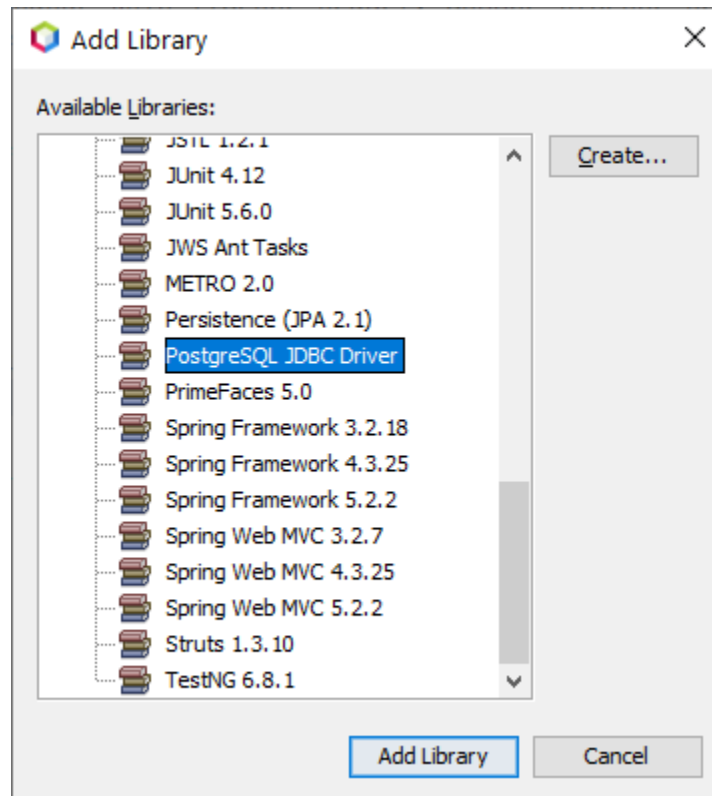
Tipo SQL	Tipo Java
LONGVARCHAR	java.lang.String
NUMERIC	java.math.BigDecimal
REAL	float
SMALLINT	short
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
TINYINT	byte
VARBINARY	Byte[]
VARCHAR	Java.lang.String

Exemplo de Conexão com o Banco



Conexão com o Banco

- Primeiro, adicione o JDBC Driver no seu projeto.



Importar bibliotecas

```
package conexaobd;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
```


Classe conexão

```
public class Conexao {
```

```
    private Connection conn;
```

Método para conectar ao BD

```
public Connection conectar() {  
    try {  
        // Informando qual driver de conexão será utilizado pelo DriverManager  
        Class.forName("org.postgresql.Driver");  
        // Criando a conexão com o BD  
        String url = "jdbc:postgresql://localhost:5432/testeJava";  
        String username = "postgres";  
        String password = "123456";  
        conn = DriverManager.getConnection(url, username, password);  
        System.out.println("Conectado com Sucesso");  
        return conn;  
    } catch (ClassNotFoundException | SQLException e) {  
        System.err.println("Erro ao conectar: "+e.getMessage());  
        return null;  
    }  
}
```

Método para desconectar do BD

```
public void desconectar() {  
    try {  
        if (conn != null && !conn.isClosed()) {  
            // Desconectando do BD  
            conn.close();  
        }  
    } catch (SQLException e) {  
    }  
}
```

Classe conexão

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

public class Conexao {
    private Connection conn;

    public Connection conectar() {
        try {
            // Informando qual driver de conexão será utilizado pelo DriverManager
            Class.forName("org.postgresql.Driver");
            // Criando a conexão com o BD
            String url = "jdbc:postgresql://localhost:5432/testeJava";
            String username = "postgres";
            String password = "123456";
            conn = DriverManager.getConnection(url, username, password);
            System.out.println("Conectado com Sucesso");
            return conn;
        } catch (ClassNotFoundException | SQLException e) {
            System.err.println("Erro ao conectar: " + e.getMessage());
            return null;
        }
    }

    public void desconectar() {
        try {
            if (conn != null && !conn.isClosed()) {
                // Desconectando do BD
                conn.close();
            }
        } catch (SQLException e) {
        }
    }
}
```

Testando a conexão com o banco

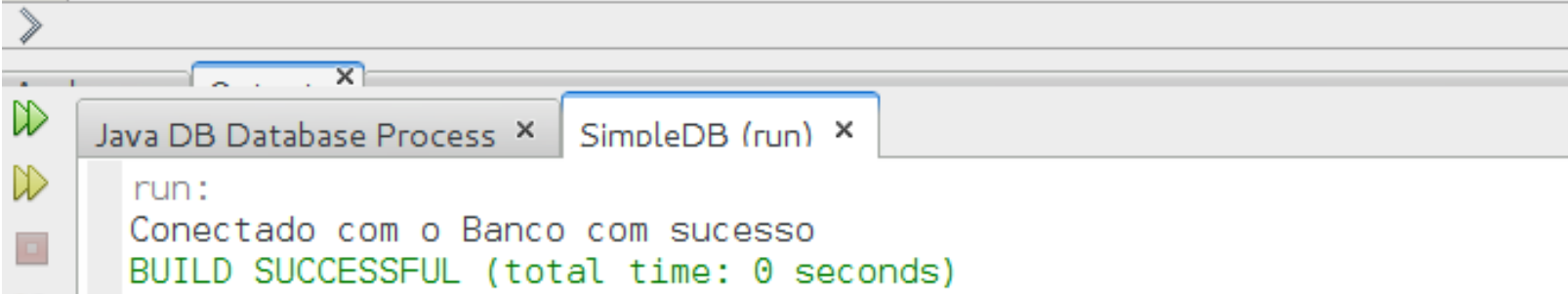
```
3 public class Principal {  
4     public static void main(String[] args) {  
5         // TODO code application logic here  
6         Conexao conexao = new Conexao();  
7         if(conexao.conectar()==null)  
8             System.err.println("Erro ao se conectar com o Banco");  
9         else  
10            System.out.println("Conectado com o Banco com sucesso");  
11     }  
12 }
```

run:

Erro ao conectar: IO Error: The Network Adapter could not establish the connection
Erro ao se conectar com o Banco
BUILD SUCCESSFUL (total time: 0 seconds)

Testando a conexão com o banco

```
3 public class Principal {  
4     public static void main(String[] args) {  
5         // TODO code application logic here  
6         Conexao conexao = new Conexao();  
7         if(conexao.conectar()==null)  
8             System.err.println("Erro ao se conectar com o Banco");  
9         else  
10            System.out.println("Conectado com o Banco com sucesso");  
11     }  
12 }
```



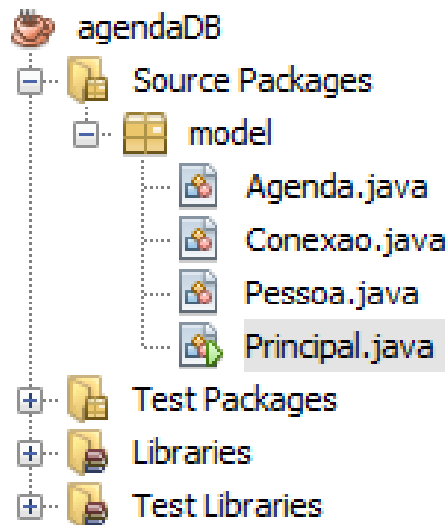
The screenshot shows an IDE interface. At the top, a code editor displays a Java class named `Principal` with a `main` method. The code attempts to connect to a database using a `Conexao` object. Below the code editor, there is a tabbed window titled "SimpleDB (run)". The output of the program is displayed in the console area, showing the message "Conectado com o Banco com sucesso" and a green "BUILD SUCCESSFUL" status.

run:
Conectado com o Banco com sucesso
BUILD SUCCESSFUL (total time: 0 seconds)

Exemplo de Inserindo dados no Banco

Testando a conexão com o banco

- Para esse projeto será criado as seguintes classes:



Classe conexão

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

public class Conexao {
    private Connection conn;

    public Connection conectar() {
        try {
            // Informando qual driver de conexão será utilizado pelo DriverManager
            Class.forName("org.postgresql.Driver");
            // Criando a conexão com o BD
            String url = "jdbc:postgresql://localhost:5432/testeJava";
            String username = "postgres";
            String password = "123456";
            conn = DriverManager.getConnection(url, username, password);
            System.out.println("Conectado com Sucesso");
            return conn;
        } catch (ClassNotFoundException | SQLException e) {
            System.err.println("Erro ao conectar: "+e.getMessage());
            return null;
        }
    }

    public void desconectar() {
        try {
            if (conn != null && !conn.isClosed()) {
                // Desconectando do BD
                conn.close();
            }
        } catch (SQLException e) {
        }
    }
}
```

Classe Pessoa

```
public class Pessoa {  
    private String nome;  
    private String telefone;  
  
    public Pessoa(String nome, String telefone) {  
        this.nome = nome;  
        this.telefone = telefone;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public String getTelefone() {  
        return telefone;  
    }  
}
```

Método Inserir - (INSERT)

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Agenda {
    private Connection conn;
    private PreparedStatement pstmt;
    private ResultSet rs;

    public void inserir(Pessoa pessoa) {
        Conexao conexao = new Conexao(); // Abrindo a conexão com o banco
        conn = conexao.conectar();
        try {
            String inserir = "INSERT INTO AGENDA (NOME, TELEFONE) VALUES ( ?, ?)";
            pstmt = conn.prepareStatement(inserir);
            // Setando o valor aos parâmetros
            pstmt.setString(1, pessoa.getNome());
            pstmt.setString(2, pessoa.getTelefone());
            pstmt.execute(); // Executando o comando sql do objeto preparedStatement
            System.out.println("Inserido com Sucesso");
            conexao.desconectar(); // Fechando a conexão com o banco
        } catch (SQLException e) {
            conexao.desconectar(); // Fechando a conexão com o banco
            System.err.println("Falha em Inserir no DB: " + e.getMessage());
        }
    }
}
```

Método main()

```
1
2 package model;
3
4 public class Principal {
5
6     public static void main(String[] args) {
7         Agenda agenda = new Agenda();
8
9         agenda.inserir(new Pessoa("Cicrano", "(11) 90000-1000"));
10    }
11 }
12
```

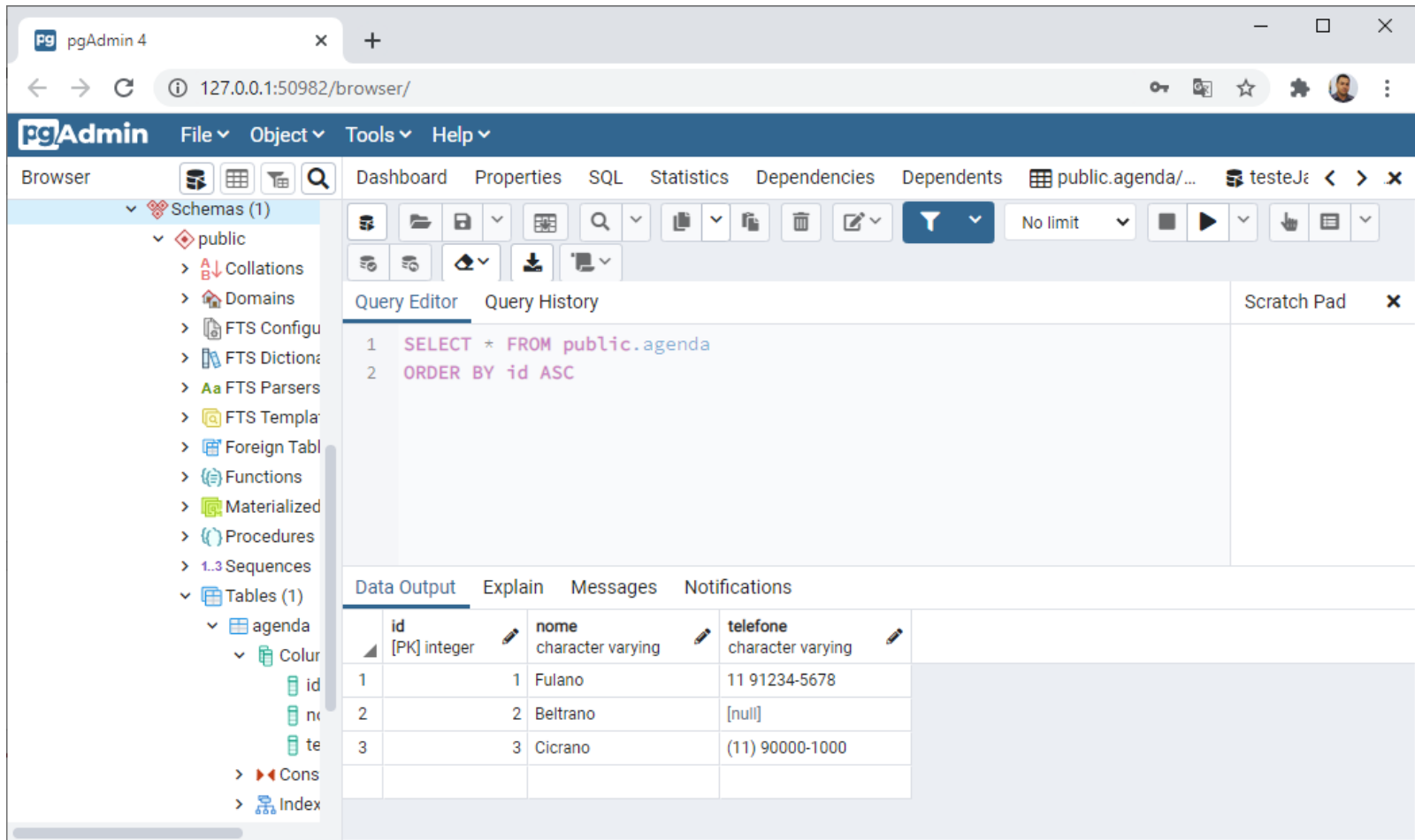
Output ×

SQL 1 execution × agendaDB (run) ×

run:
Conectado com Sucesso
Inserido com Sucesso
BUILD SUCCESSFUL (total time: 0 seconds)

Verificando o banco

- Verifique o valor inserido no banco.



The screenshot shows the pgAdmin 4 web interface in a browser. The left sidebar displays the database structure, with the 'public' schema expanded and the 'agenda' table selected. The main panel shows the 'Query Editor' with the following SQL query:

```
1 SELECT * FROM public.agenda
2 ORDER BY id ASC
```

Below the query editor, the 'Data Output' tab is active, displaying the results of the query in a table format:

id	nome	telefone
1	Fulano	11 91234-5678
2	Beltrano	[null]
3	Cicrano	(11) 90000-1000

Método remover - (DELETE)

```
public void remover(String nome) {  
    Conexao conexao = new Conexao(); // Abrindo a conexão com o banco  
    conn = conexao.conectar();  
    try {  
        String remove = "DELETE FROM AGENDA WHERE NOME = ?";  
        pstmt = conn.prepareStatement(remove);  
        pstmt.setString(1, nome); // Setando o valor aos parâmetros  
        pstmt.execute(); // Executando o comando sql do objeto preparedStatement  
        System.out.println("Removido com Sucesso");  
        conexao.desconectar(); // Fechando a conexão com o banco  
    } catch (SQLException e) {  
        conexao.desconectar(); // Fechando a conexão com o banco  
        System.err.println("Falha em remover no DB: " + e.getMessage());  
    }  
}
```

Método main()

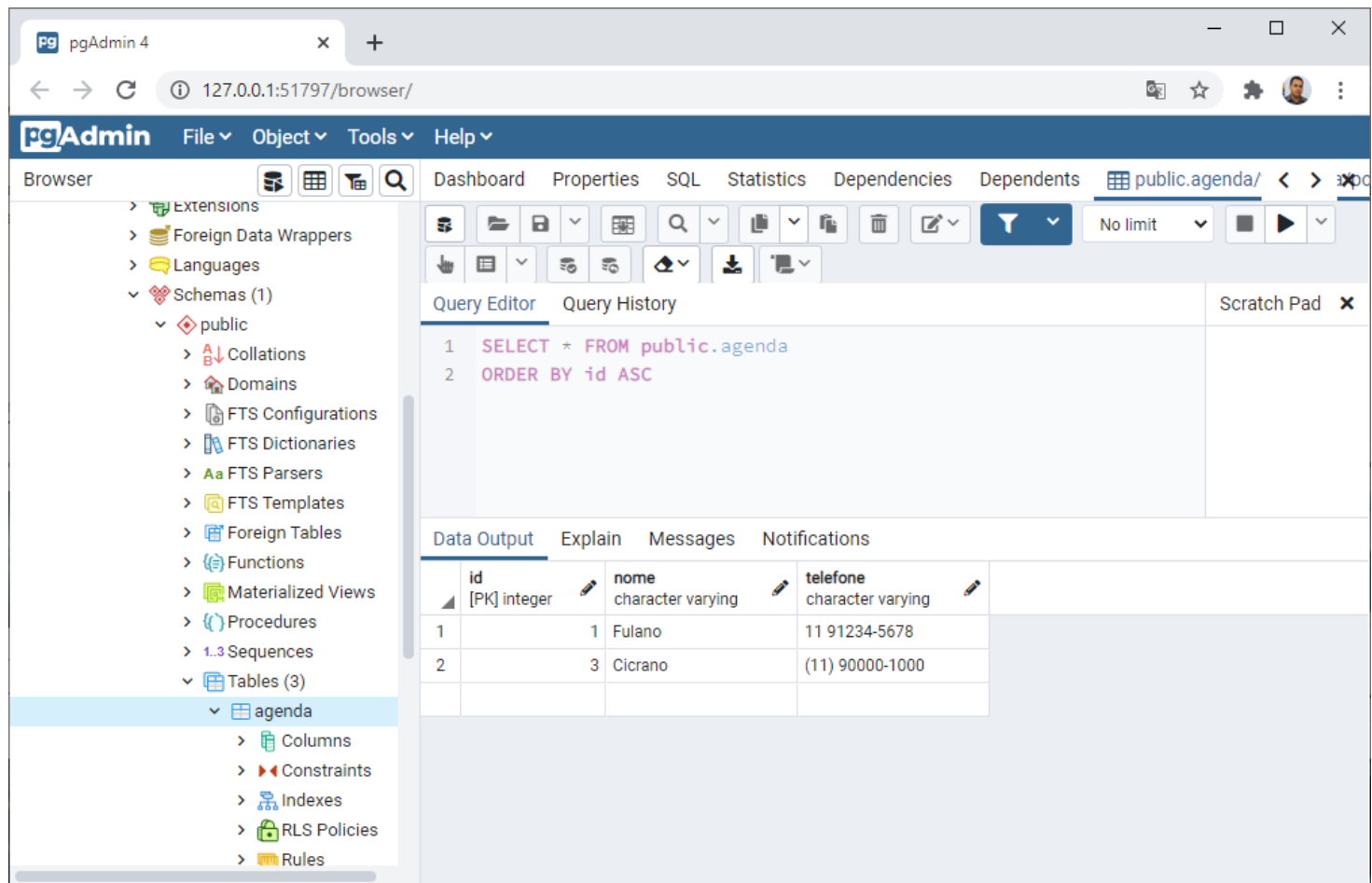
```
1
2 package model;
3
4 public class Principal {
5
6     public static void main(String[] args) {
7         Agenda agenda = new Agenda();
8
9         agenda.remover("Beltrano");
10    }
11 }
12
```

Output - agendaDB (run) ×

run:
Conectado com Sucesso
Removido com Sucesso
BUILD SUCCESSFUL (total time: 0 seconds)

Verificando o banco

- Verifique o valor removido do banco.



The screenshot shows the pgAdmin 4 web interface in a browser window. The address bar displays '127.0.0.1:51797/browser/'. The left sidebar shows the database structure, with 'public' expanded and 'agenda' selected. The main panel shows the 'Query Editor' with the following SQL query:

```
1 SELECT * FROM public.agenda
2 ORDER BY id ASC
```

Below the query editor, the 'Data Output' tab is active, displaying the results of the query in a table:

id	nome	telefone
1	Fulano	11 91234-5678
2	Cicrano	(11) 90000-1000