Conectar a API ao BD Tipos de arquitetura



ATIVIDADES DE HOJE

Horário das atividades

- 13h30 Faísca
- 13h45 Prática SQL
- 15h00 Intervalo
- 15h15 Conectar a API ao BD e Tipos de arquitetura
- 16h40 Checkout
- 17h00 Chamada



Criar o banco

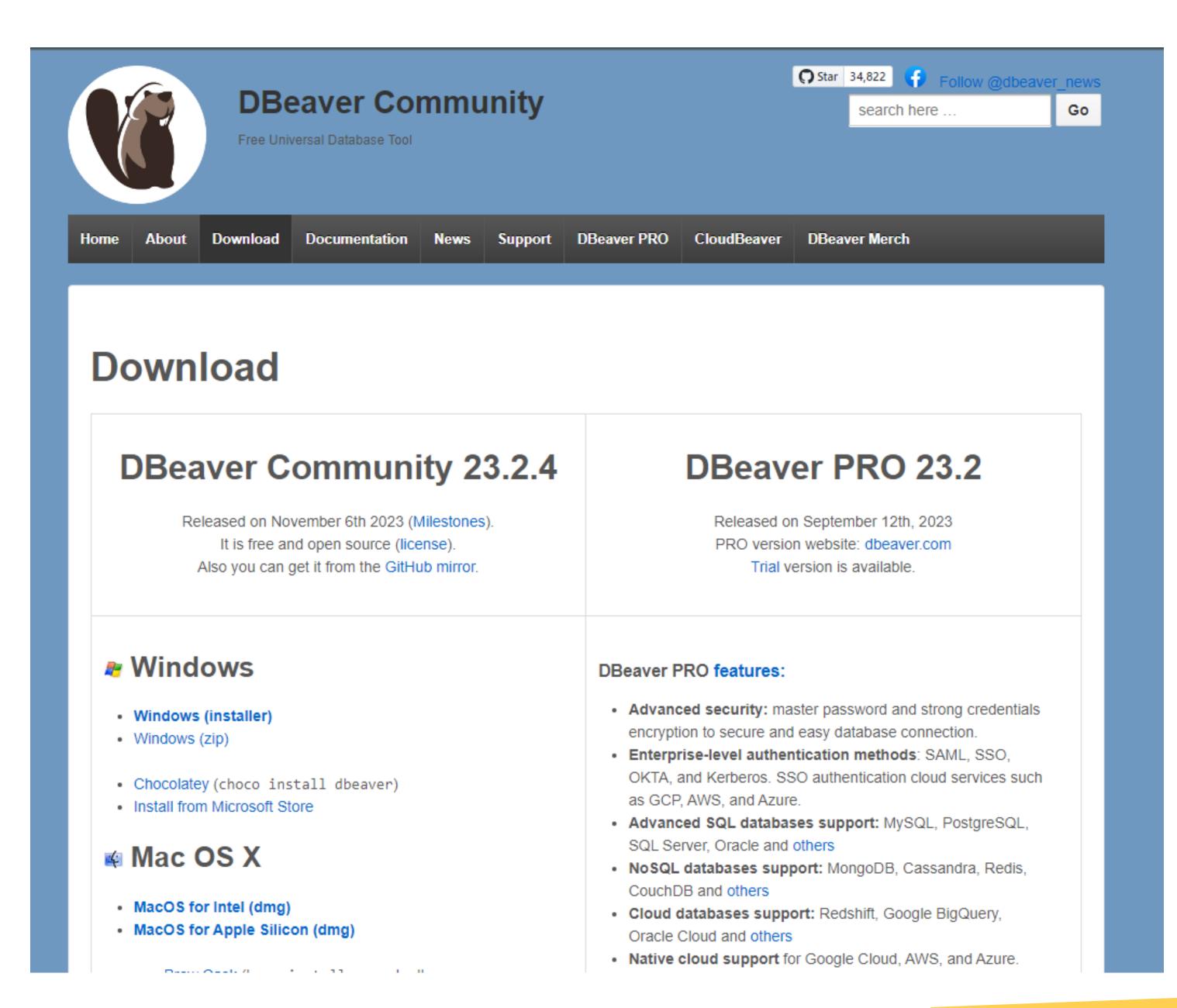


DBeaver

https://dbeaver.io/

- Clique em Download e execute;
- Marque o "Associate SQL Script with Database"

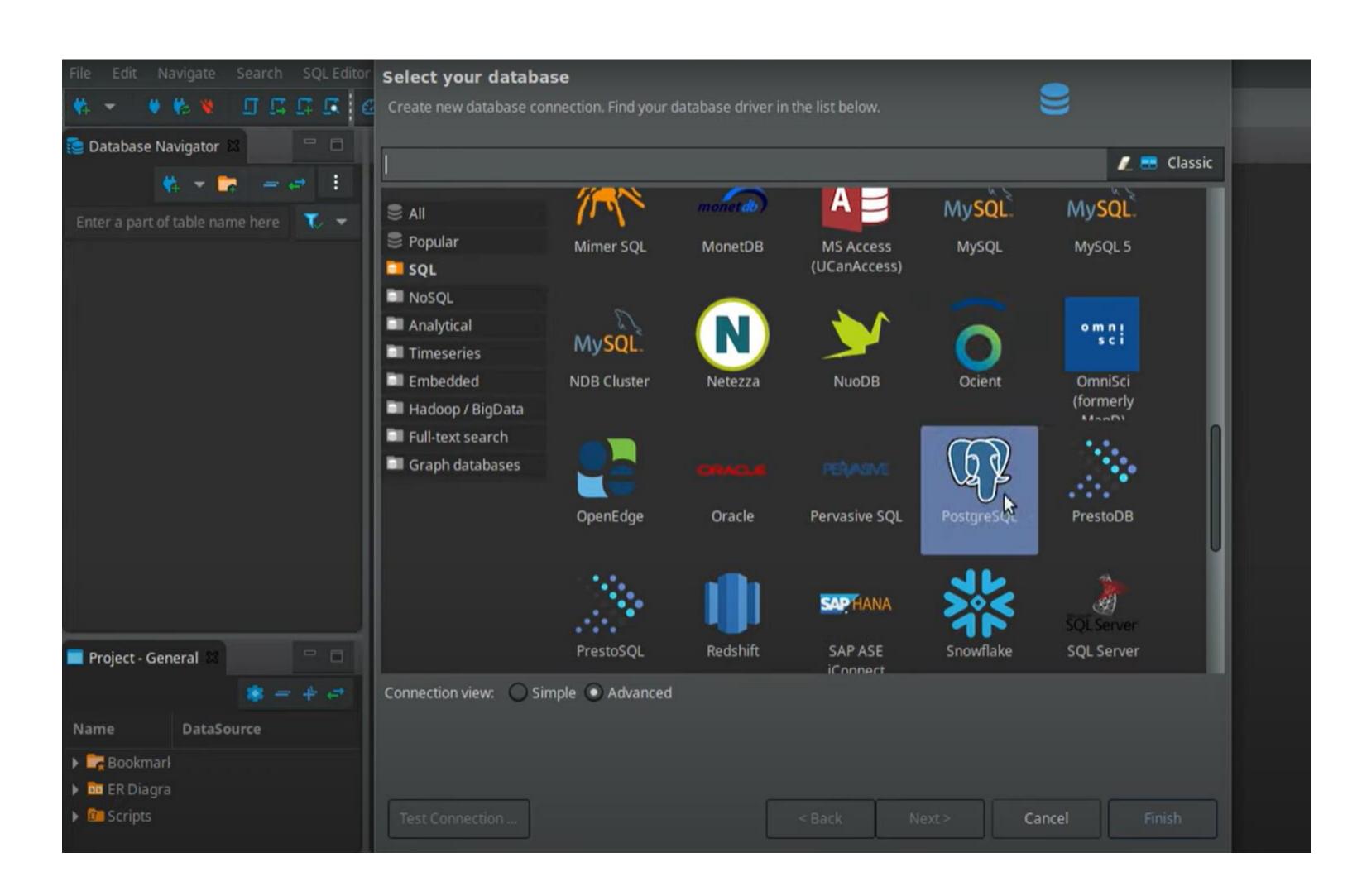






Adicionando uma conexão

- Clique na "tomada" no canto superior esquerdo;
- Selecione PostGreSQL;





Adicionando uma conexão

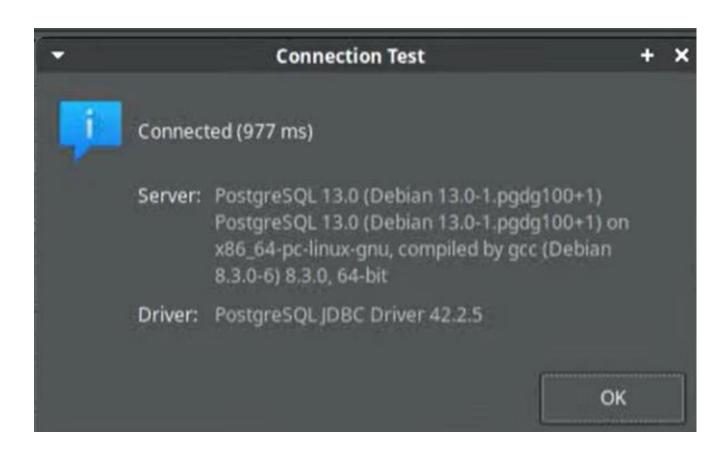
Configure as credenciais do teu banco;

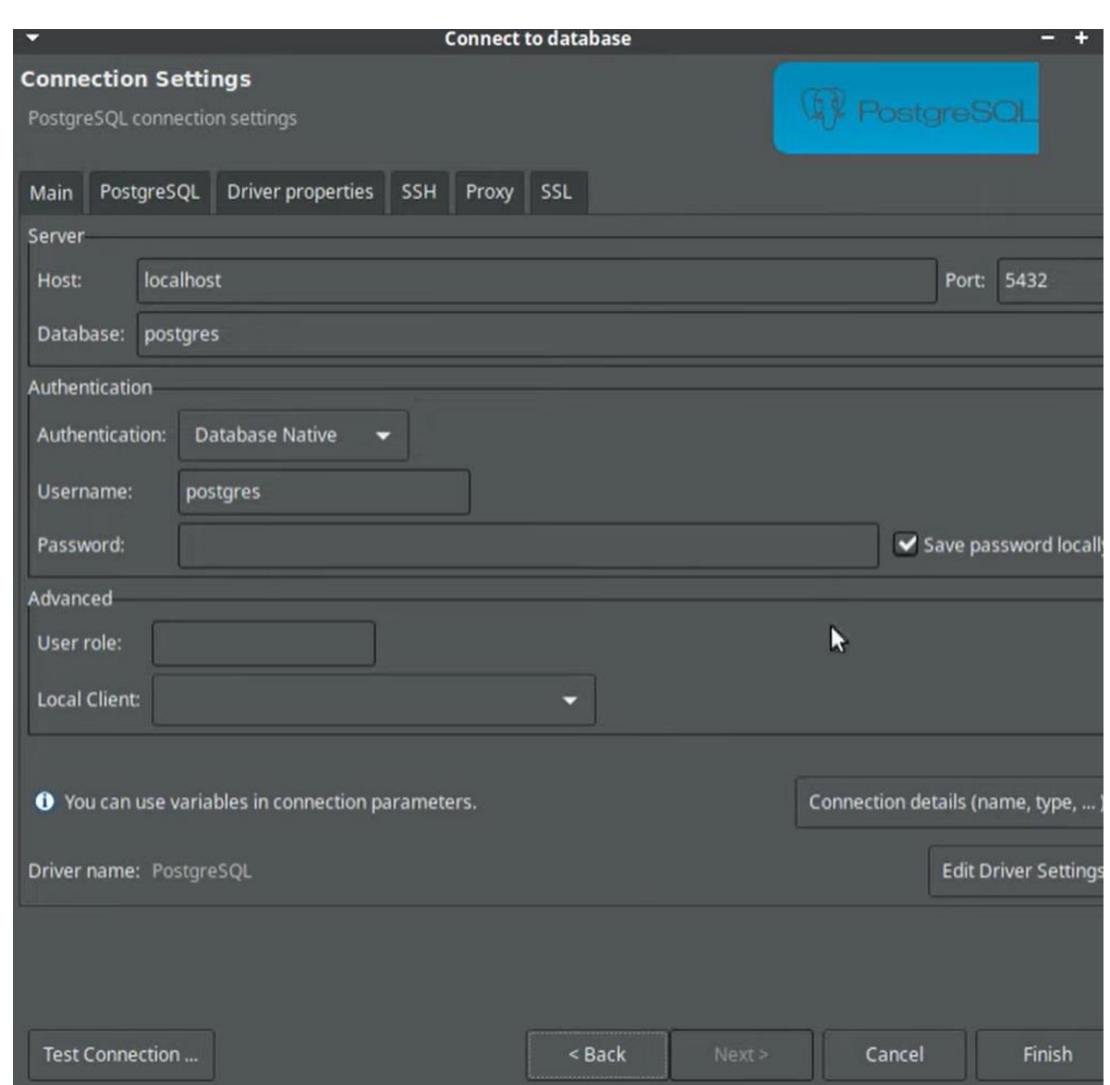
Database: postgres

Username: postgres

Password: postgres

E teste a conexão;







Criando uma table

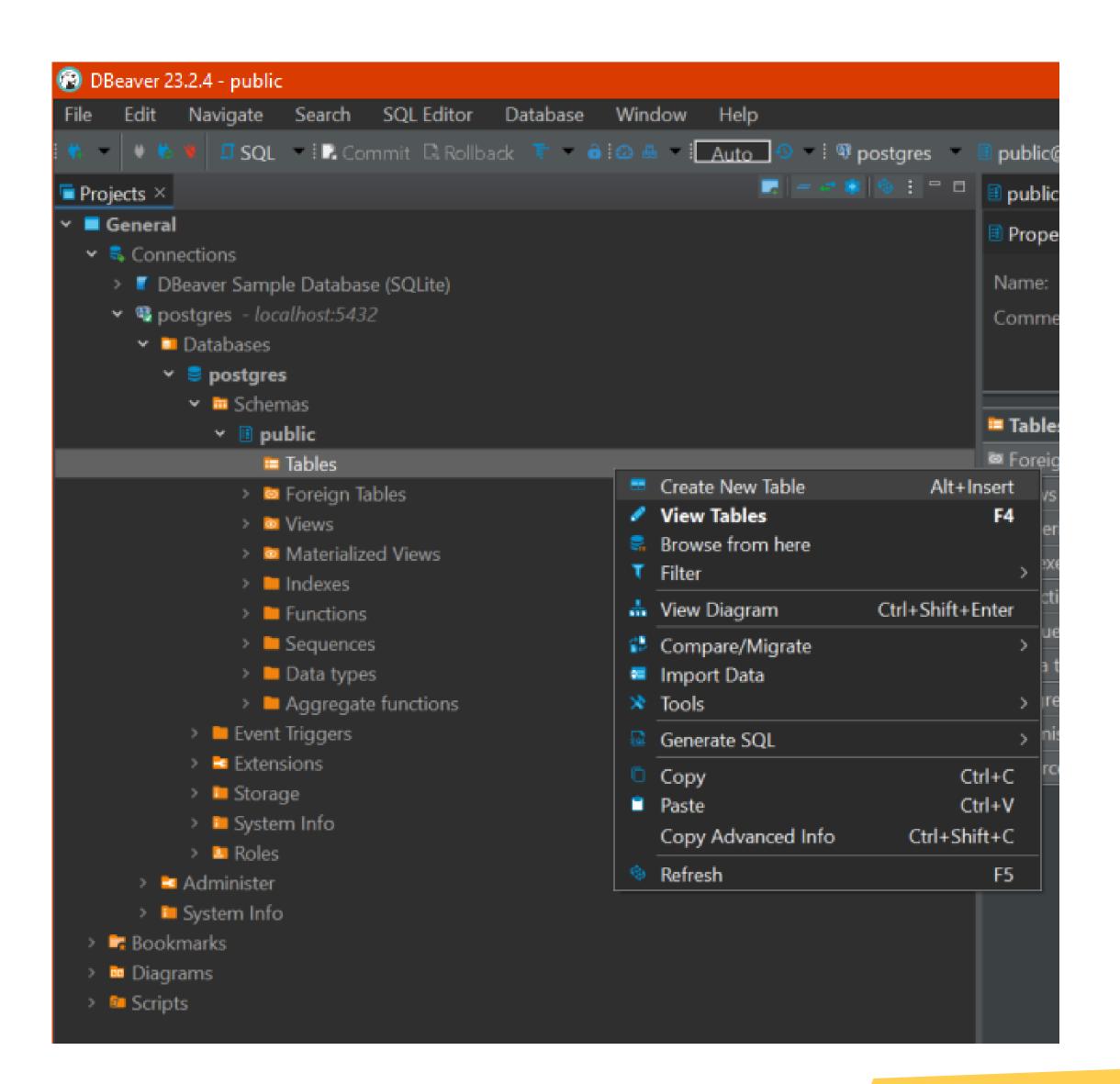
 Siga o caminho na imagem e clique com o botão direito e em Create New Table

Database: postgres

Username: postgres

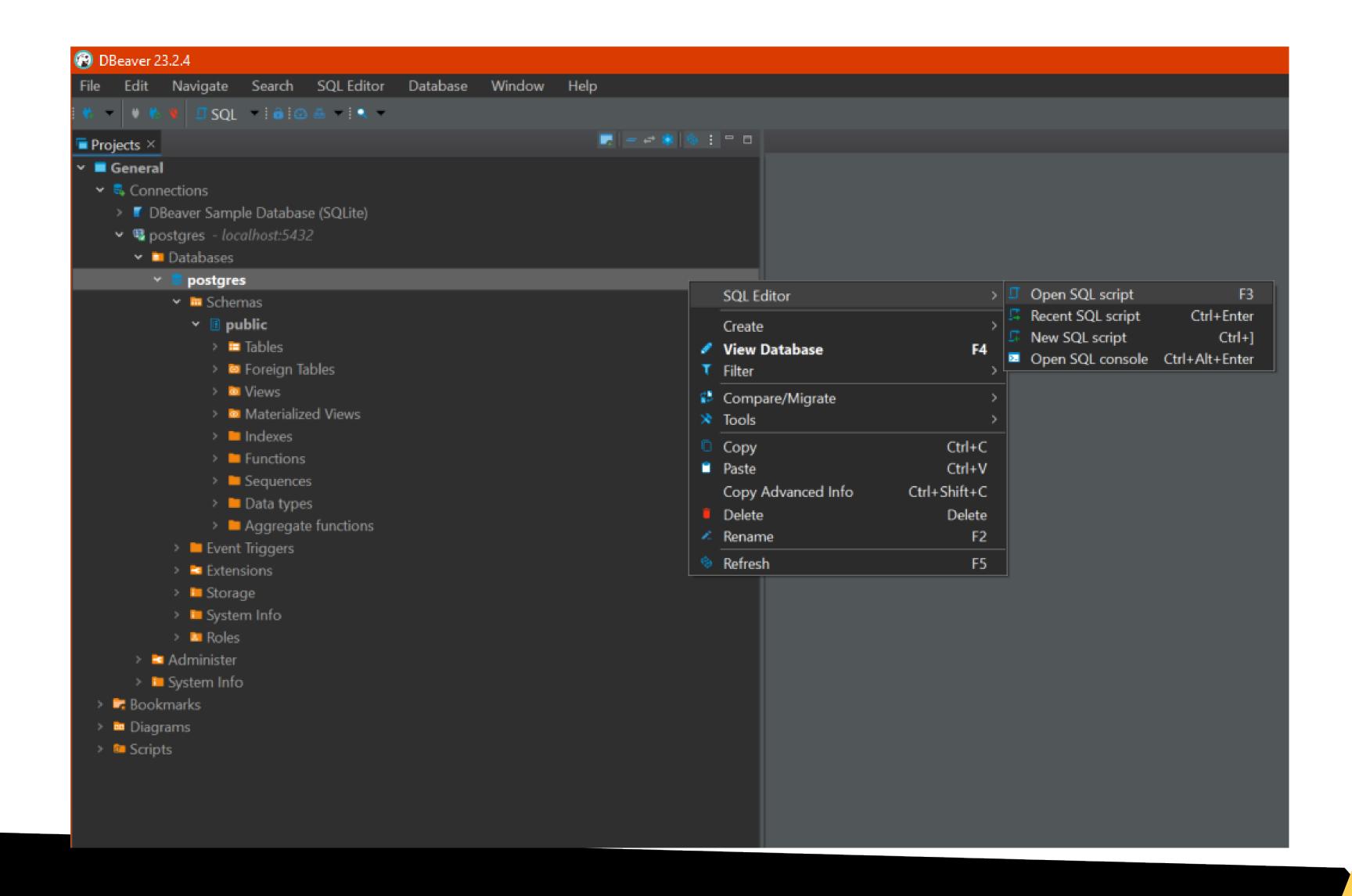
Password: postgres

• E teste a conexão;



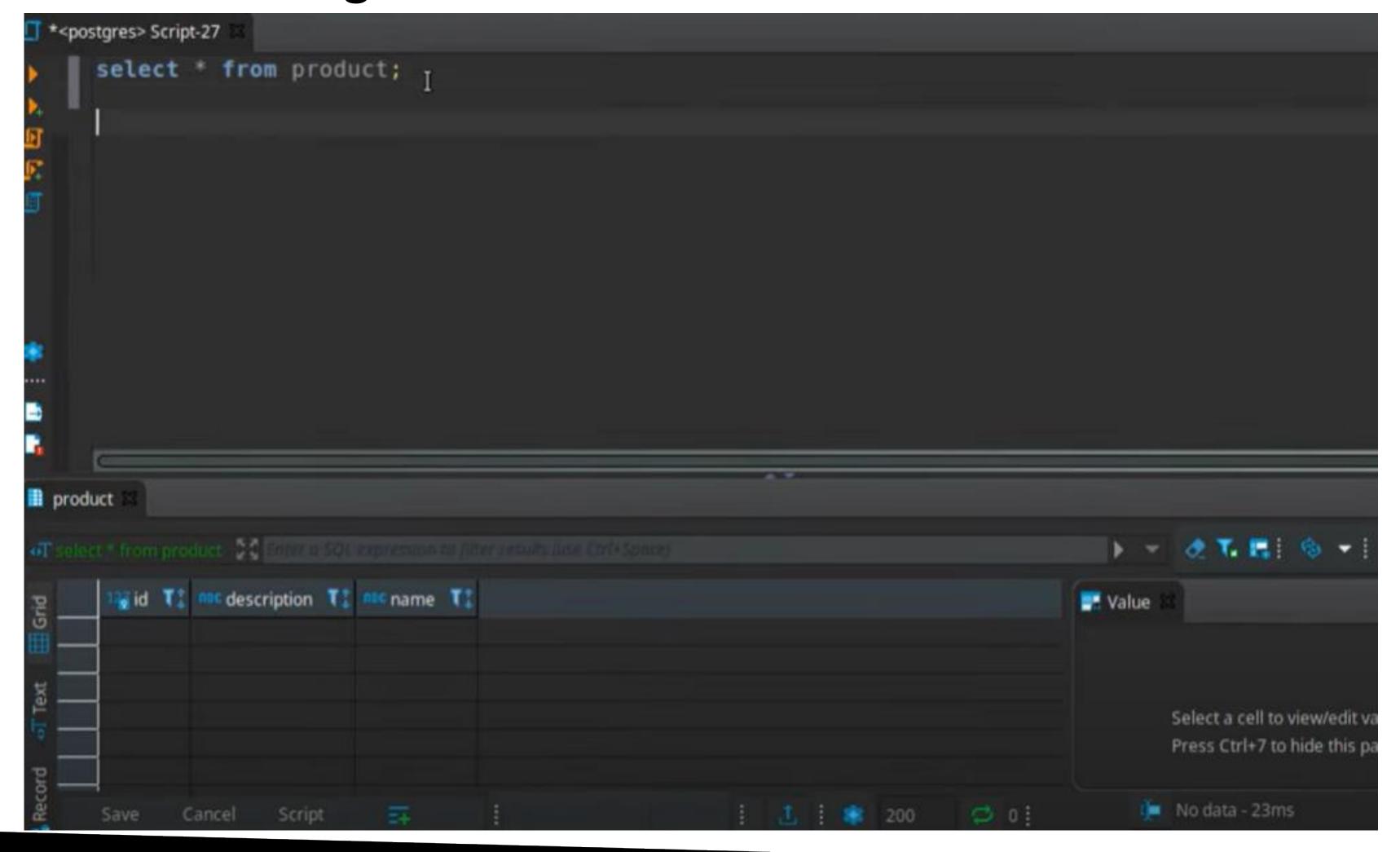


Criando uma querie



Criando uma table

Você pode rodar o código com CRTL + Enter





Conectar a API ao BD



Conectar a API ao BD

Criação de um projeto Spring

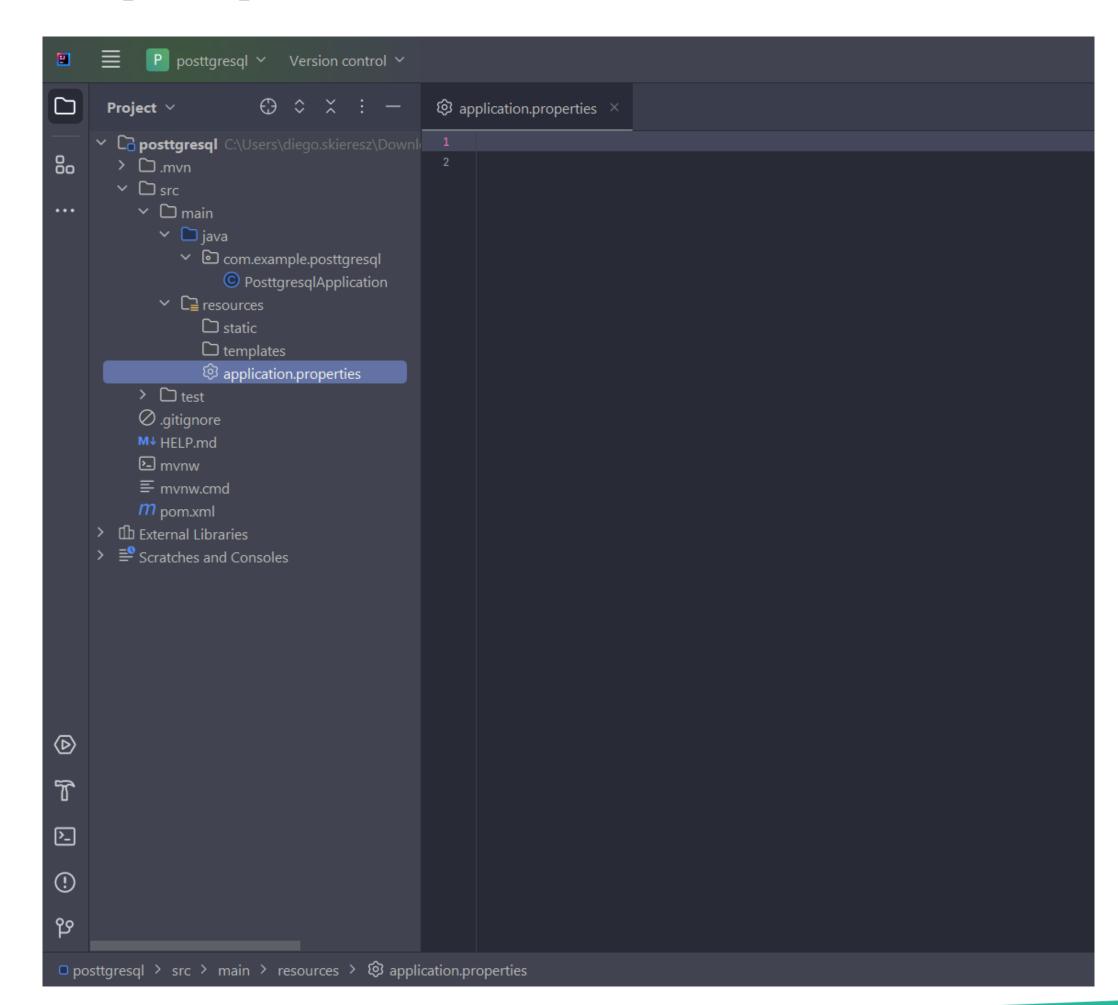
≡	spring initializr							
	Artifact Name Description Package name Packaging	HOT) O 3.2.0 (RC2) O 3.1.6 (SNASHOT) O 3.0.12 O 2.7.18 (SNAPS	APSHOT) • 3.1.5 SHOT) • 2.7.17		Spring Data JPA Persist data in SQL st PostgreSQL Drive A JDBC and R2DBC of	RESTful, applications using Spring stainer. SQL ores with Java Persistence API u	ısing Spriı	ng Data and Hibernate.
⊕			GENERATE CTRL+ ₫	EXPLO	ORE CTRL + SPACE	SHARE		



Estruturas de retrospectiva

Criar o database e editar o application.properties

- Neste aquivo especificamos diferentes propriedades do SpringBoot que queremos usar;
- Aqui onde iremos especificar a nossa string para conectar com o nosso database;





Configurando application.properties

Criar o database e editar o application.properties

- Primeiro definir a url de origem do banco, com a porta e o nome do database no final;
- Segundo, especificar o username;
- E também o password do "database";

```
(3) application.properties
Project ~
posttgresql C:\Users\diego.skieresz\Downl
                                                 = Set here configurations for the database connection
   .mvn
                                               # Connection url for the database "postgres_database"
                                                spring.datasource.url=jdbc:postgresql://localhost:5432/postgres_database
         © PosttgresqlApplication
                                               # Username and password

∨ □ resources

                                               spring.datasource.username = username
                                               spring.datasource.password = password
           static
                                          11
           templates
           application.properties
                                          13
   > 🗀 test
                                          14
   Ø .gitignore
```



Configurações adicionais

```
Project ~
                                              application.properties

▼ Coposttgresql C:\Users\diego.skieresz\Downl
                                                    # ==============
                                                   # = Set here configurations for the database connection
       > 🗀 .mvn

✓ ☐ src

✓ ☐ main

                                                   # Connection url for the database "postgres_database"

✓ □ java

                                                    spring.datasource.url=jdbc:postgresql://localhost:5432/postgres_database
               © PosttgresqlApplication
                                                   # Username and password

✓ ☐ resources

                                                    spring.datasource.username = username
                                                    spring.datasource.password = password
                 static
                                              11
                 templates
                                                   # Controls the maximum number of milliseconds that you will wait for setting up a connection from the pool
                 application.properties
                                                    spring.datasource.hikari.connection-timeout=20000
         > 🗀 test
                                                   # Specifies number of database connections between database and application.
         ② .gitignore
                                                    spring.datasource.hikari.maximum-pool-size=5
         M↓ HELP.md
         ⊡ mvnw
                                                    # ===============
         ≡ mvnw.cmd
                                                    # = JPA / HIBERNATE
                                                    m pom.xml
                                              20
      Libraries
                                                   # Show or not log for each sql query
    > 
Scratches and Consoles
                                                   spring.jpa.show-sql = true
                                              23
                                                   # Hibernate ddl auto (create, create-drop, update): with "update" the database
                                                   # schema will be automatically updated accordingly to java entities found in
                                                   # the project
                                                   spring.jpa.hibernate.ddl-auto = update
                                              28
                                                   # Allows Hibernate to generate SQL optimized for a particular DBMS
                                                   spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
D
                                                   # spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
                                              32
```



Configurando application.properties

Criar o database e editar o application.properties

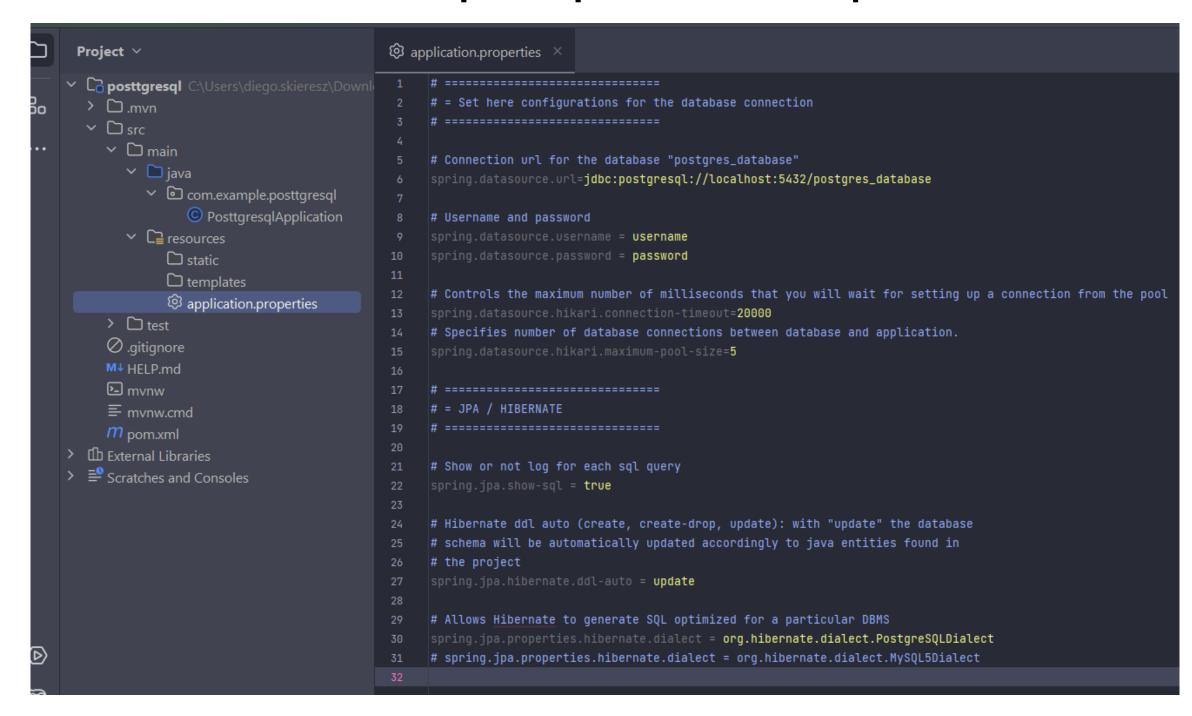
• Connection timeout = Quanto tempo para conexão deve disparar o timeout

• Poolsize = Máximo de conexões simultâneas ao database que queremos que o

spring crie;

 Dialect = Qual dialeto do SQL iremos utilizar. O hibernate vai gerar queries automaticamente para o database;

- ddl-auto = Ele vai atualizar o database com a mudança, por exemplo se você muda o model e conecta ao database ele ira ver que existe uma mudança no model e vai executar um update ao database;
- show-sql = Loga o sql no console





Run the application



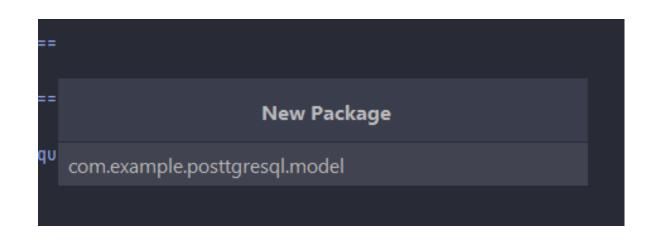


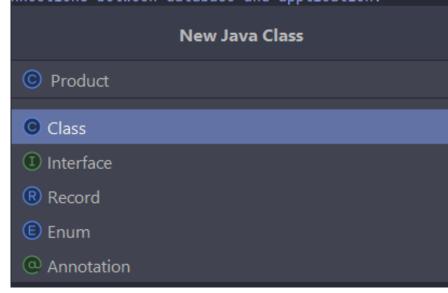
Model

- O que é? É como se fosse a "memória" da aplicação,
- onde guardamos informações importantes.
- Representa a camada de dados da aplicação.
- É responsável por gerenciar o estado e a lógica de negócios.
- No Spring, os objetos Model geralmente são POJOs (Plain Old Java Objects) que encapsulam os dados da aplicação.
- **Exemplo:** Se nossa aplicação é sobre uma lista de tarefas, o Model seria onde guardamos cada tarefa com suas informações, como nome, data, etc.



Criando o Model





@Entity Diz ao Spring que é uma entidade no database

@ld é usada para marcar o campo id como a chave primária da entidade;

@GeneratedValue especifica como o valor da chave primária é gerado;

@GeneratedValue(strategy = GenerationType.AUTO) indica que a estratégia de geração da chave primária deve ser determinada automaticamente

Usar no InteliJ o alt + insert para criar os construtores

```
Project 🗸
                                               C Product.java
 🗖 posttgresql
                                                    package com.example.posttgresql.model;
   .mvn
                                                     import jakarta.persistence.Entity;
   □ src
                                                     import jakarta.persistence.GeneratedValue;

✓ ☐ main

                                                     import jakarta.persistence.Id;

    Com.example.posttgresql

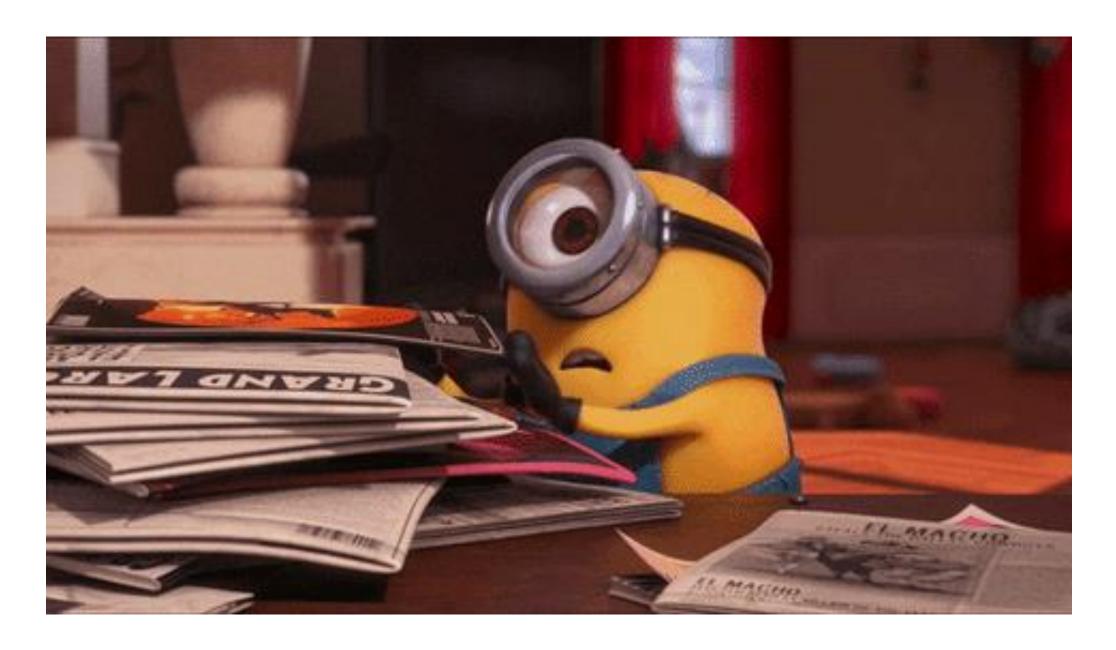
                                                     // Anotação 'Entity' indica que esta classe é uma entidade JPA que pode ser mapeada
             // para uma tabela de banco de dados
                  © Product
               © PosttgresglApplication
                                                    @Entity
                                                     public class Product {

→ □ resources

            static
                                                        // Anotação 'Id' indica que o campo 'id' é a chave primária da entidade
            templates
            application.properties
                                                        @Id
                                                        // Anotação 'GeneratedValue' especifica como o valor da chave primária é gerado
   > 🗀 test
   .gitignore
                                                        @GeneratedValue(strategy = GenerationType.AUTO)
   M↓ HELP.md
   <u>►</u> mvnw
                                                        // Campo que armazena o nome do produto
   ≡ mvnw.cmd
   m pom.xml
                                                        private String name;
 // Campo que armazena a descrição do produto
 Scratches and Consoles
                                                        private String description;
                                                        // Construtor padrão da classe Product
                                                        public Product() {
                                                        // Construtor que permite inicializar os campos da classe Product
                                                        public Product(Long id, String name, String description) {
                                                            this.id = id;
                                                            this.name = name;
                                                            this.description = description;
```

Repository

• O que é? É como um assistente pessoal para lidar com o armazenamento de informações. Ele cuida de salvar e buscar coisas no "armário" da aplicação (o banco de dados).



- É responsável pela interação com o banco de dados. Ele abstrai o acesso aos dados, permitindo que o Model se concentre na lógica de negócios.
- No Spring, os repositórios são frequentemente interfaces estendidas por interfaces do Spring Data JPA, que fornece métodos para realizar operações básicas de CRUD (Create, Read, Update, Delete) no banco de dados.
- **Exemplo:** Continuando com o exemplo, um repositório de tarefas poderia ter métodos como save(Task task), findById(Long id), etc.



Para criar um repositório e usar o model usaremos o JpaRepository

JpaRepository é uma interface fornecida pelo Spring Data JPA, possui funcionalidades predefinidas para realizar operações CRUD (Create, Read, Update e Delete) em entidades JPA.

Outras funcionalidades:

• **Métodos de Consulta Derivada**: O Spring Data JPA permite a criação de consultas personalizadas de forma declarativa, derivando o nome dos métodos na interface do repositório. Por exemplo, você pode criar métodos com nomes específicos, como findByFirstNameAndLastName, e o Spring Data JPA automaticamente gera a consulta SQL correspondente com base no nome do método..



Para criar um repositório e usar o model usaremos o JpaRepository

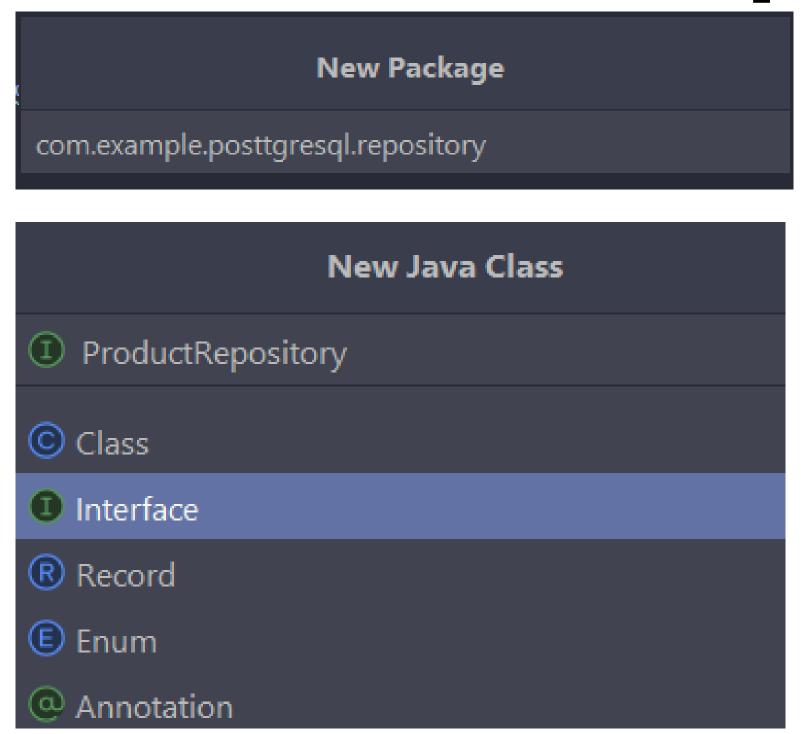
- Paginação e Classificação: O JpaRepository oferece suporte à paginação e classificação de resultados, facilitando a recuperação de grandes conjuntos de dados de forma eficiente.
- Consultas Nativas e JPQL: Além das consultas derivadas, o JpaRepository permite a execução de consultas SQL nativas e consultas JPQL (Java Persistence Query Language) personalizadas.
- **Métodos Customizados:** Você pode adicionar seus próprios métodos personalizados no repositório, para atender a requisitos específicos de consulta que não podem ser atendidos por consultas derivadas.



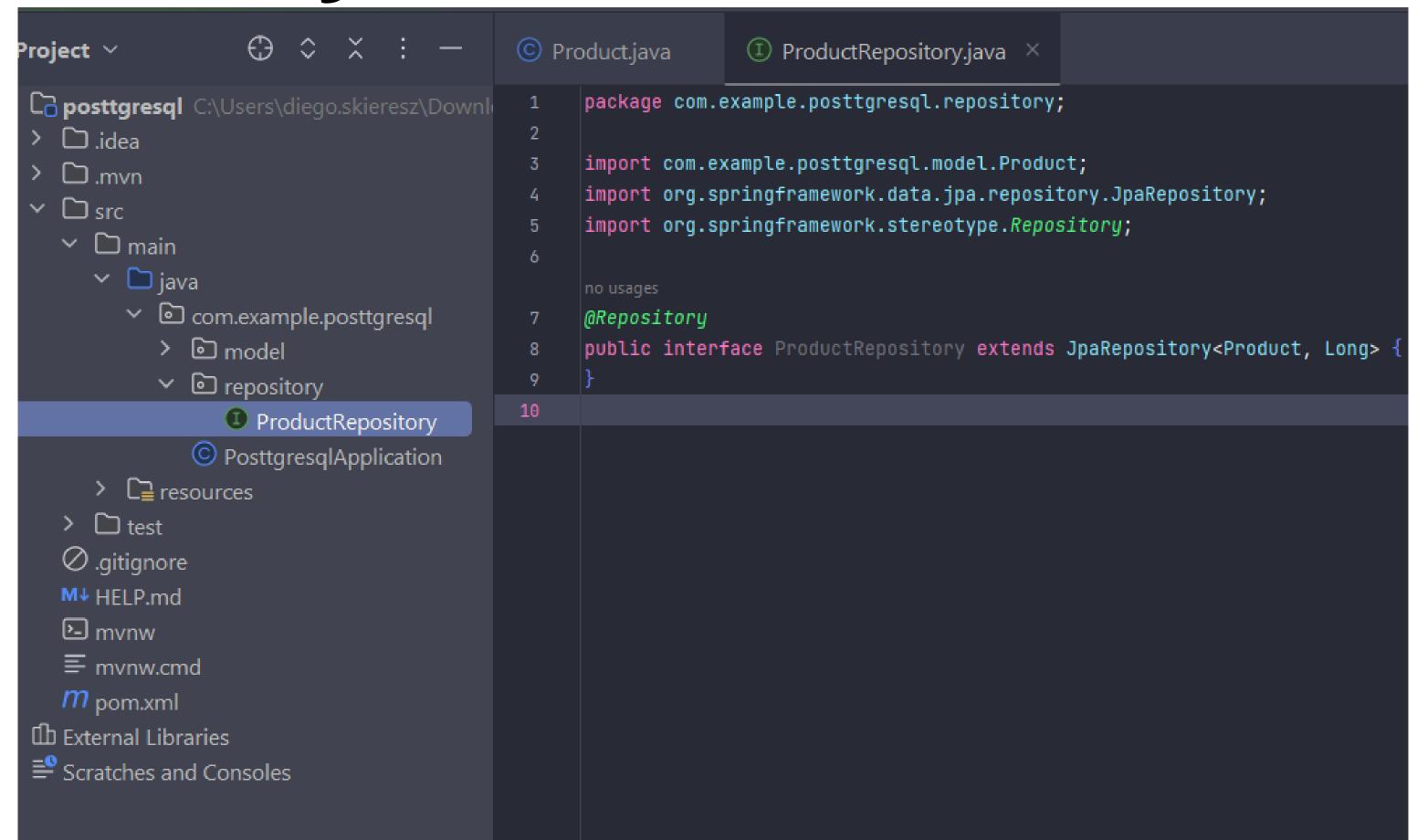
Existe outras maneiras de criar repositórios além do JpaRepository?

- CrudRepository: Esta é a interface mais básica e fornece métodos CRUD (Create, Read, Update e Delete) genéricos. É a interface pai do JpaRepository.
- PagingAndSortingRepository: Esta interface estende CrudRepository e adiciona suporte a paginação e classificação de resultados.
- QuerydslPredicateExecutor: Permite a execução de consultas complexas usando o Querydsl, que é uma biblioteca que fornece uma maneira tipada e segura de criar consultas dinâmicas.
- **KeyValueRepository**: Esta interface é usada quando você deseja trabalhar com armazenamento de chave-valor, em vez de um banco de dados relacional.
- Repositórios de grafo: O Spring Data também oferece suporte para repositórios orientados a gráficos, como Neo4jRepository, se você estiver usando um banco de dados de grafo como o Neo4j.





 Precisa configurar duas coisas, qual o tipo de dado tu esta armazenando no Repo (Product), e também qual o tipo do ID (Long)



Controller

- O que é? É o "cérebro" da aplicação, que recebe as ordens (requisições) e decide o que fazer com elas.
- Responsável por receber as requisições do cliente, processar essas requisições e chamar os métodos apropriados no Model.
- No Spring, os controladores são geralmente classes anotadas com @Controller. Eles contêm métodos mapeados para URLs específicas.
- **Exemplo:** Se alguém quer adicionar uma nova tarefa à lista, o Controller recebe essa ordem e avisa o Model para adicionar essa tarefa.



Criando o Controller

```
© ProductController.java ×
                  ¹roject ∨
                                                                   package com.example.posttgresql.controller;
                  posttgresql C:\Users\diego.skieresz\Downle
  com.example.pc >
                                                                   import com.example.posttgresql.repository.ProductRepository;
                     .mvn
                                                                   import org.springframework.http.ResponseEntity;

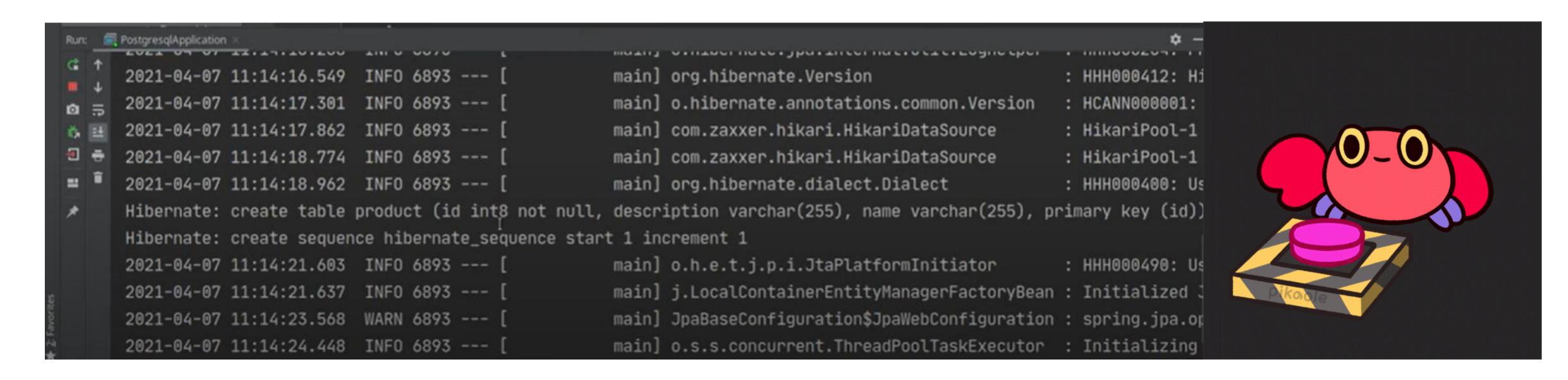
→ □ src

                                                                   import org.springframework.web.bind.annotation.GetMapping;
                    ∨ 🗀 main
                                                                   import org.springframework.web.bind.annotation.RequestMapping;
                                                                   import org.springframework.web.bind.annotation.RestController;

✓ ② com.example.posttgresql

                             // Define que esta classe é um controlador que lida com solicitações HTTP.
                                   ProductController
© ProductContr
                                                                   @RestController
                             > 🗈 model
                                                                   // Mapeia a raiz da URL para "/product" para este controlador.
                             @RequestMapping("/product")
Class
                                   ProductRepository
                                                                   public class ProductController {
                               © PosttgresqlApplication
Interface
                       > 🗀 resources
                                                                       // Declaração de uma variável para armazenar o repositório de produtos.
                     > 🗀 test
R Record
                                                                      private final ProductRepository productRepository;
                     ② .gitignore
E Enum
                                                            17
                     M↓ HELP.md
                                                                       // Construtor da classe que recebe uma instância do repositório de produtos como parâmetro.
                                                            18
                     ⊡ mvnw
Annotation
                     ≡ mvnw.cmd
                                                                       public ProductController(ProductRepository productRepository) {
                                                            19
                     m pom.xml
                                                                          // Injeta o repositório no controlador por meio da injeção de dependência do Spring.
                   file External Libraries
                                                                          this.productRepository = productRepository;
                                                            21
                   Scratches and Consoles
                                                            22
                                                            23
                                                                       // Método que responde a solicitações HTTP GET para "/product".
                                                            24
                                                                       @GetMapping
                                                                      public ResponseEntity getAllProducts() {
                                                                          // Retorna uma resposta HTTP encapsulada em uma ResponseEntity.
                                                            27
                                                                          // O conteúdo da resposta é obtido chamando o método "findAll" no repositório.
                                                                          return ResponseEntity.ok(this.productRepository.findAll());
```

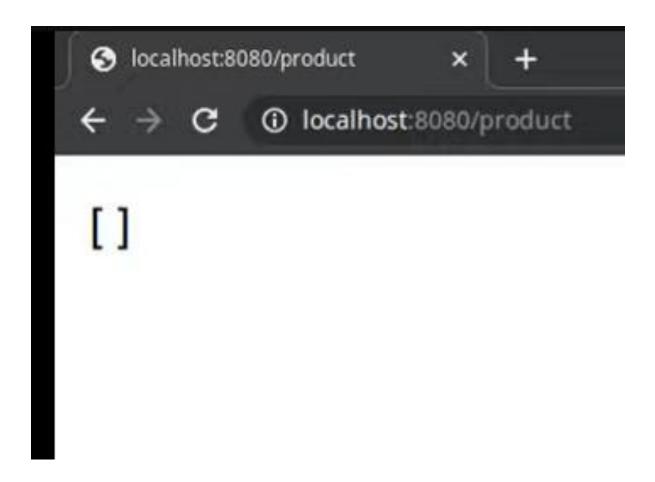
Start application

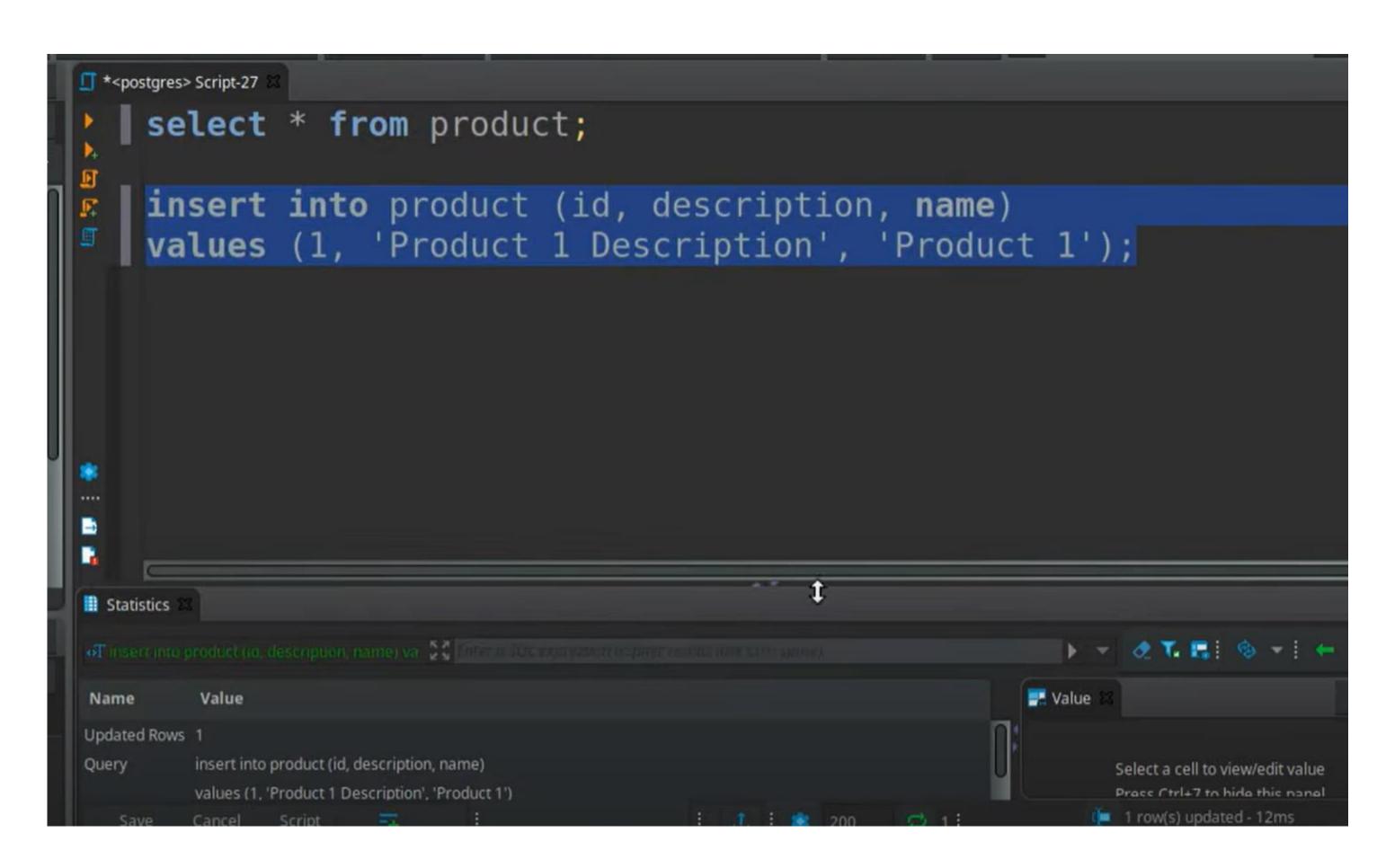


- Essa é a feature que utilizamos, **show-sql**, que mostra as queries que estamos executando no database;
- Como selecionamos o generatios type auto, ele esta criando a tabela "hibernate_sequence" onde, . Cada vez que uma nova entidade é persistida no banco de dados, o Hibernate consulta essa tabela para obter o próximo valor da sequência e o atribui à entidade antes de inseri-la no banco de dados.



Adding some data







Adding some data

```
*<postgres> Script-27
   select * from product;
    insert into product (id, description, name)
    values (1, 'Product 1 Description', 'Product 1');

    ∫ localhost:8080/product

                                                                         → C (i) localhost:8080/product
                                                                       [{"id":1, "name": "Product 1", "description": "Product 1 Description"}]
m product
             description 🚺 👊 name 🏋
         1 Product 1 Description Product 1
```

Agradecemos a sua atenção!

