Minhas anotações do projeto:

API Application Interface - é o que fica exposta para WEB. é a porta de entrada do seu back-end.

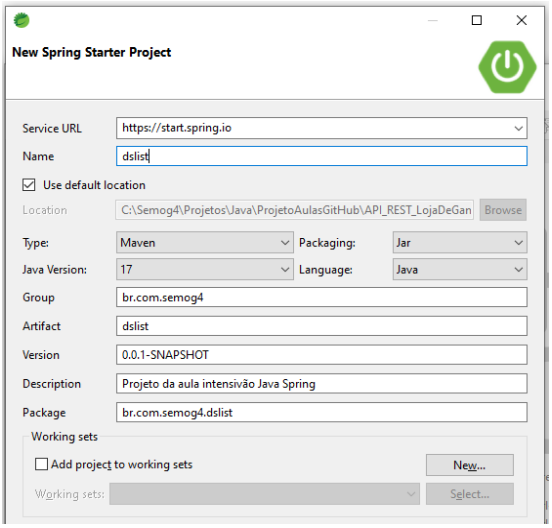
REST - é um padrão que especifica como uma API deve ser implementada para ser considerada REST. Seguindo os padrões do REST você terá uma API REST :

1 - Tem que ser uma aplicação cliente/servidor;
2 - Comunicação é stateless (não guarda status, ou seja o resultado da requisição não depende de algo que o sistema deva guardar para retornar o resultado. Ex. o retorno não depende de algo que precisa estar armazenado previamente na sessão do usuário para retornar)
3 - Interface uniforme com formato padronizado utilizando os verbos http(utilização dos verbos http para realização das operações)
GET - obter
POST - gravar
PUT - alterar (verbo idempotente significa que uma requisição realizada 1 vez ou 10 vez o resultado será o mesmo)
DELETE - deletar
4 - Utilização de Cache
5 - Sistema em camadas
6 - Código sob demanda (opcional)

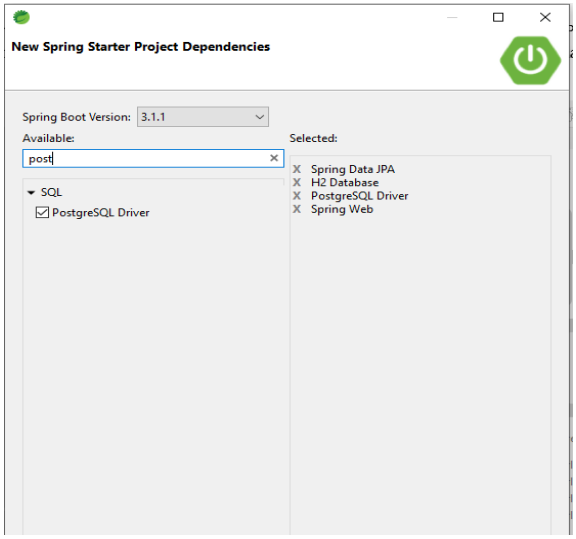
Uma porta com maçaneta podemos atribuir a maçaneta como sendo a API (a interface)
a porta de entrada para o seu back-end (que podemos considerar sendo a porta)

Camadas:
- Front End
- Back End
  - Controladores REST (Os controladores são as portas de entrada/interface = API)
    - Camada do Serviços (realizada as transações)
    - Camada de Acesso a Dados (realizada as transações)

Configuração do projeto dslist (pasta: API_REST_LojaDeGames)



Dependências do projeto: pom.xml



Como solução para resolver alguns casos de incompatibilidade do maven, devemos indicar a versão do maven que desejamos utilizar no pom.xml: (opcional, caso aconteça o problema segue a solução)

```
<plugin>
	<groupId>org.apache.maven.plugins</groupId>
	<artifactId>maven-resources-plugin</artifactId>
	<version>3.1.0</version>
</plugin>
```

Configurar o arquivo application.properties para ter vários perfis de ambiente: teste, homologação ...

1 – no application.properties (perfil de teste)
#criando um perfil de teste
spring.profiles.active=${APP_PROFILE:test}
spring.jpa.open-in-view=false

cors.origins=${CORS_ORIGINS:http://localhost:5173,http://localhost:3000}

2 – criar um outro application.properties com o nome application-test.properties

```
#H2 Connection - dados de conexao com o banco de dados
spring.datasource.url=jdbc:h2:mem:testedb
spring.datasource.username=sa
spring.datasource.password=

#H2 Client - habilita o console para ser utilizado no browse
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

#Show H2 - habilita a exibicao das queries SQL no console do STS
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

3 – Seed no tabela de game. Criar o arquivo import.sql na pasta resource.

Primeiro insert na tabela tb_game:

```
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('Mass Effect Trilogy', 4.8, 2012, 'Role-
playing (RPG), Shooter', 'XBox, Playstation, PC',
'https://raw.githubusercontent.com/devsuperior/java-spring-
dslist/main/resources/1.png', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil itaque
ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus dolorum
illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti dicta
odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('Red Dead Redemption 2', 4.7, 2018,
'Role-playing (RPG), Adventure', 'XBox, Playstation, PC',
'https://raw.githubusercontent.com/devsuperior/java-spring-
dslist/main/resources/2.png', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil itaque
ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus dolorum
illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti dicta
odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('The Witcher 3: Wild Hunt', 4.7, 2014,
'Role-playing (RPG), Adventure', 'XBox, Playstation, PC',
'https://raw.githubusercontent.com/devsuperior/java-spring-
dslist/main/resources/3.png', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil itaque
ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus dolorum
illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti dicta
odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('Sekiro: Shadows Die Twice', 3.8, 2019,
'Role-playing (RPG), Adventure', 'XBox, Playstation, PC',
'https://raw.githubusercontent.com/devsuperior/java-spring-
dslist/main/resources/4.png', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil itaque
ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus dolorum
illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti dicta
odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('Ghost of Tsushima', 4.6, 2012, 'Role-
playing (RPG), Adventure', 'XBox, Playstation, PC',
'https://raw.githubusercontent.com/devsuperior/java-spring-
dslist/main/resources/5.png', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil itaque
ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus dolorum
illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti dicta
odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('Super Mario World', 4.7, 1990,
'Platform', 'Super Ness, PC', 'https://raw.githubusercontent.com/devsuperior/java-
spring-dslist/main/resources/6.png', 'Lorem ipsum dolor sit amet consectetur
adipisicing elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil
itaque ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus
```

```
dolorum illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti
dicta odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('Hollow Knight', 4.6, 2017, 'Platform',
'XBox, Playstation, PC', 'https://raw.githubusercontent.com/devsuperior/java-spring-
dslist/main/resources/7.png', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil itaque
ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus dolorum
illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti dicta
odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('Ori and the Blind Forest', 4, 2015,
'Platform', 'XBox, Playstation, PC',
'https://raw.githubusercontent.com/devsuperior/java-spring-
dslist/main/resources/8.png', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil itaque
ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus dolorum
illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti dicta
odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('Cuphead', 4.6, 2017, 'Platform', 'XBox,
Playstation, PC', 'https://raw.githubusercontent.com/devsuperior/java-spring-
dslist/main/resources/9.png', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil itaque
ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus dolorum
illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti dicta
odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
INSERT INTO tb_game (title, score, game_year, genre, platforms, img_url,
short_description, long_description) VALUES ('Sonic CD', 4, 1993, 'Platform', 'Sega
CD, PC', 'https://raw.githubusercontent.com/devsuperior/java-spring-
dslist/main/resources/10.png', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Odit esse officiis corrupti unde repellat non quibusdam! Id nihil itaque
ipsum!', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Delectus dolorum
illum placeat eligendi, quis maiores veniam. Incidunt dolorum, nisi deleniti dicta
odit voluptatem nam provident temporibus reprehenderit blanditiis consectetur
tenetur. Dignissimos blanditiis quod corporis iste, aliquid perspiciatis architecto
quasi tempore ipsam voluptates ea ad distinctio, sapiente qui, amet quidem culpa.');
```

Concluímos a branch Aula_01

Criação da branch Aula_02:

1. Criamos a classe Belonging (tabela de relacionamento entre as tabelas Game e GameList)
2. Aprendemos que é necessário criar a classe BelongingPK para criar uma única chave quando existe uma tabela de relacionamento.
3. Seed na tabela tb_game_list. Adicionar os inserts ao arquivo import.sql na pasta resource.

```
INSERT INTO tb_game_list (name) VALUES ('Aventura e RPG');
INSERT INTO tb_game_list (name) VALUES ('Jogos de plataforma');
```

4. Seed no tabela tb_belonging. Adicionar os inserts ao arquivo import.sql na pasta resource.

```
INSERT INTO tb_belonging (list_id, game_id, position) VALUES (1, 1, 0);
INSERT INTO tb_belonging (list_id, game_id, position) VALUES (1, 2, 1);
INSERT INTO tb_belonging (list_id, game_id, position) VALUES (1, 3, 2);
INSERT INTO tb_belonging (list_id, game_id, position) VALUES (1, 4, 3);
```

```
INSERT INTO tb_belonging (list_id, game_id, position) VALUES (1, 5, 4);

INSERT INTO tb_belonging (list_id, game_id, position) VALUES (2, 6, 0);
INSERT INTO tb_belonging (list_id, game_id, position) VALUES (2, 7, 1);
INSERT INTO tb_belonging (list_id, game_id, position) VALUES (2, 8, 2);
INSERT INTO tb_belonging (list_id, game_id, position) VALUES (2, 9, 3);
INSERT INTO tb_belonging (list_id, game_id, position) VALUES (2, 10, 4);
```

5. Fizemos uso do método copyProperties da classe BeanUtils na classe GameDTO. Esse método atribui/copia todos os atributos de uma classe para outra, desde que ambas tenham os mesmos atributos e tipos. Ex. BeanUtils.copyProperties(entityGame, this);
   Obs.: Se utilizar este "macete" a classe DTO deverá ter todos os get e set, e não somente os get.

6. No GameService utilizamos a anotação @Transaction(readOnly=true) para garantir que ocorra a transação e o atributo informe ao banco que é uma transação de leitura e não de escrita, tornando o processo mais rápido.

7. Criando consulta SQL nativa:
   Query / método adicionado na GameRepository
   Sempre que utilizar a query nativa, o resultado da query vai derivar da interface Projection, assim para o nosso projeto criamos a interface GameMinProjection.

```
@Query(nativeQuery = true, value = """
  SELECT tb_game.id, tb_game.title, tb_game.game_year AS `year`,
         tb_game.img_url AS imgUrl,
         tb_game.short_description AS shortDescription, tb_belonging.position
  FROM tb_game
           INNER JOIN tb_belonging ON tb_game.id = tb_belonging.game_id
  WHERE tb_belonging.list_id = :listId
  ORDER BY tb_belonging.position
                    """)
List<GameMinProjection> searchByList(Long listId);
```

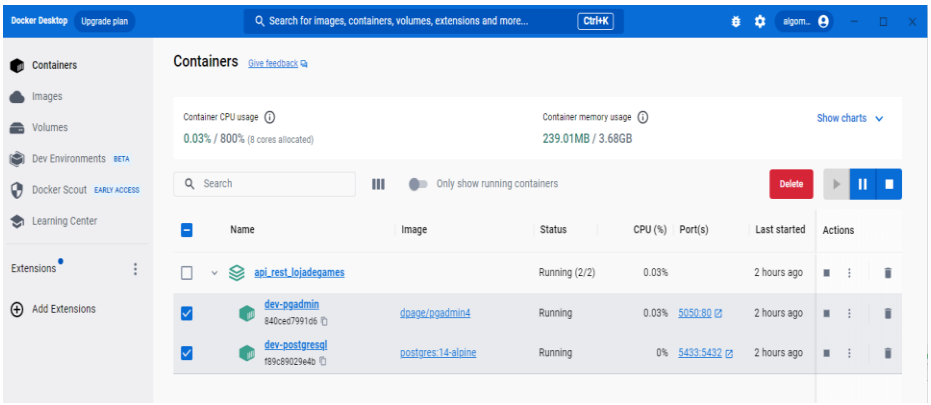
8. Na interface Projection deve ser declarada um método get para cada campo a ser retornado pela query.
   Ex.: tb_game.id = Long getId();
   tb_game.title = String getTitle();
9. Criação do endpoint http://localhost:8080/lists/2/games (leitura: traga a lista da categoria 2 dos games)

Concluímos a branch Aula_02

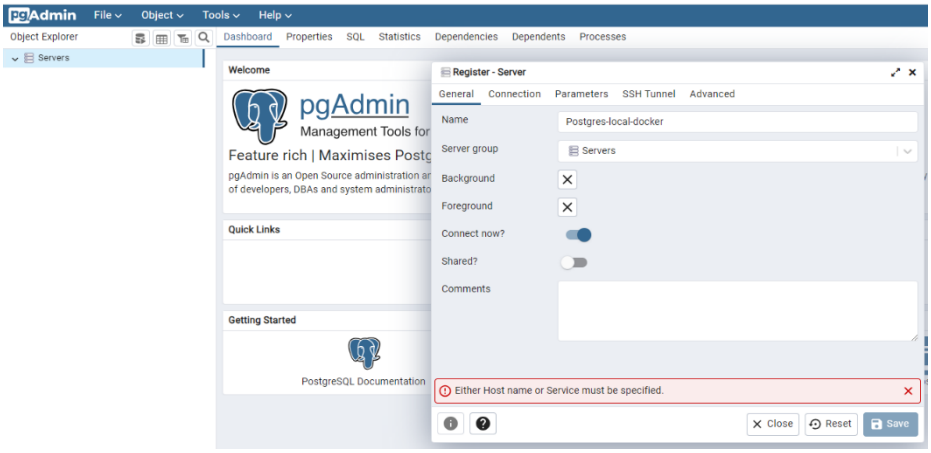
Criação da branch Aula_03:

Preparando a aplicação para rodar no banco de Postgresql:

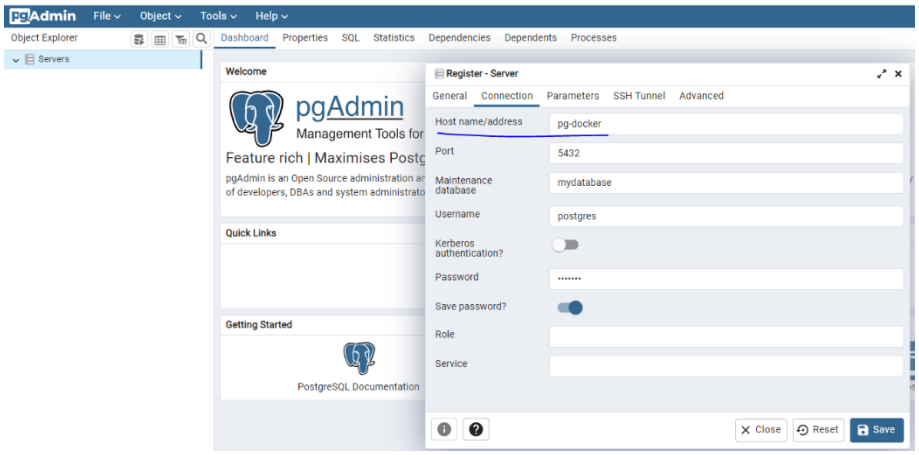
1. Podemos executar a aplicação de duas formas: instalando o Postgresql e o cliente pgAdmin na máquina local junto com a aplicação, ou instalando através do Docker o Postgresql e o PgAdmin em containers e executando a aplicação local. Utilizaremos a segunda opção com o Docker Compose:
    a. Baixar o script do docker-compose.yml (o script possui a configuração para criar dois containeres: Servidor do Postgresql e do PgAdmin).
    b. Executar a aplicação Docker Desktop. É possível visualizar os dois containers inicializados: dev-pgadmin e dev-postgresql:



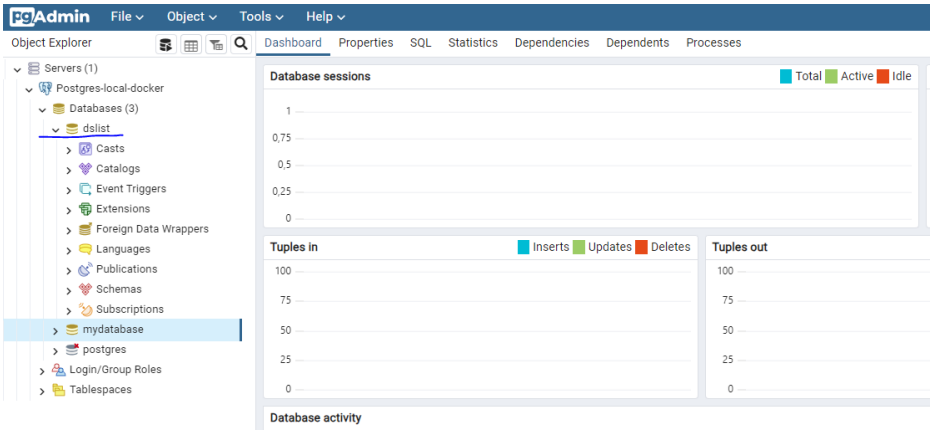
    c. Abrir o terminal do powerShell como administrador na mesma pasta onde encontra-se o script docker-compose.yml e executar o comando: docker-compose up -d (caso algum erro apareça execute o aplicativo docker).
    d. O processo anterior irar criar e iniciar os dois containeres.
    e. Confira se os containers estão executando: docker ps
    f. Testando: abra o navegador http://localhost:5050
    g. Logue com o usuário e senha do PgAdmin definido no script docker-compose.yml
    h. Configurando o Server local no Postgresql através do PgAdmin:



i. Através das informações que estão no docker-compose.yml configurar a Connection. O host name é o nome do container do Postgresql que está no script.



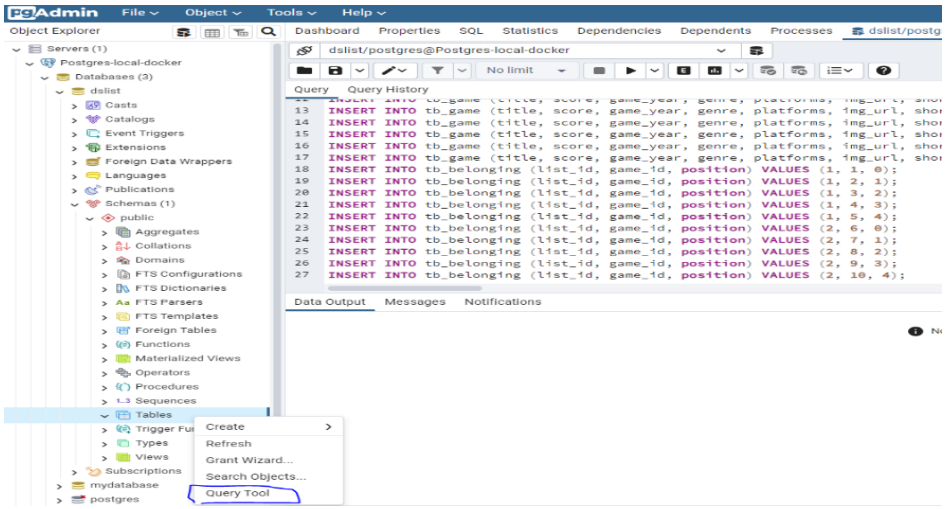
j. Criando o banco de dados dslist conforme abaixo:



k. Para popular o banco dslist com as tabelas e os seeds necessários, vamos configurar o arquivo application-dev.properties para que a aplicação ao ser inicializada gere um script com todas as informações necessárias para rodar no banco de dados. O arquivo sql gerado será criado na pasta local do projeto como o nome create.sql. Após a criação do arquivo create.sql as configurações realizadas no arquivo application-dev.properties devem ser comentadas.

l. Arquivo docker-compose.yml: analisar os comentários dentro do arquivo. Um ponto importante é quanto ao parâmetro: volumes que cria uma pasta data, essa pasta guarda o status e as informações atuais do banco, logo ao desligar a máquina as informações referente ao banco de dados não são perdidas junto com o container. Ao levantar o container novamente, as informações do banco serão recuperadas através da pasta data.

m. Abra o arquivo create.sql e execute no banco dslist conforme abaixo:



2. Criação dos perfis do projeto.
   a. Conforme configurado o arquivo application.properties, podemos indicar para a aplicação em qual ambiente desejamos executar, ou seja :
      i. application-test.properties : test (desenvolvimento utilizando o H2)
      ii. application-hom.properties: hom (ambiente de homologação utilizando Postgresql)
      iii. application-prod.properties: prod (ambiente de produção já publicado na nuvem)

   Definimos em qual ambiente queremos trabalhar, definindo no arquivo application.properties o parâmetro spring.profile.active=${APP_PROFILE:hom}. Neste caso o sistema irá utilizar as configurações definidas no arquivo: application-hom.properties.
   Cada arquivo application tem suas configurações de ambientes a serem utilizadas.

   b. Configurações do application conforme os ambientes:
      i. application-test.properties (desenvolvimento utilizando o H2)

```
#H2 Connection - dados de conexao com o banco de dados
spring.datasource.url=jdbc:h2:mem:testedb
spring.datasource.username=sa
spring.datasource.password=

#H2 Client - habilita o console para ser utilizado no browse
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

#Show H2 - habilita a exibicao das queries SQL no console do STS
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

      ii. application-hom.properties (perfil de homologação/desenvolvimento acessando Postegresql)

```
#Postgresql Connection - dados de conexao da aplicação com o
 banco de dados LOCAL no container Docker.
#As quatros linhas abaixo irão gerar um arquivo create.sql para
rodar no banco, contendo a criação das tabelas e os seeds, já
que temos que replicar as tabelas que foram criadas no H2.
#Depois de criar as tabelas no banco podemos comentar as essas
quatros linhas, já que as tabelas foram criadas e populadas no
Postgresql.
```

```
#spring.jpa.properties.jakarta.persistence.schema-
generation.create-source=metadata
#spring.jpa.properties.jakarta.persistence.schema-
generation.scripts.action=create
#spring.jpa.properties.jakarta.persistence.schema-
generation.scripts.create-target=create.sql
#spring.jpa.properties.hibernate.hbm2ddl.delimiter=;

spring.datasource.url=jdbc:postgresql://localhost:5433/dslist
spring.datasource.username=postgres
spring.datasource.password=1234567

spring.jpa.database-
platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation
=true
spring.jpa.hibernate.ddl-auto=none
```

iii. application-prod.properties (perfil de produção acessando Postegresql): Neste caso as informações referente ao banco são configuradas como parâmetros que devem ser informados na cloud.

```
#Postgresql Connection - dados de conexao com o banco de dados
PRODUCAO / NUVEM
spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}

spring.jpa.database-
platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation
=true
spring.jpa.hibernate.ddl-auto=none
```

3. Arquivo system.properties criado na raiz do projeto, devido algumas plataformas cloud exigirem este arquivo com a versão do java.

4. Urls para executar a aplicação:
   Game: http://localhost:8080/games: retorna a listagem de todos os games por categoria.
   Game by ID: http://localhost:8080/games/2: retorna dados do game 2.
   Game lists: http://localhost:8080/lists : retorna as categorias existentes.
   Game by lists: http://localhost:8080/lists/1/games: retorna todos os games da categoria 1.