

Módulo 10 – Servidores – Atividade 04

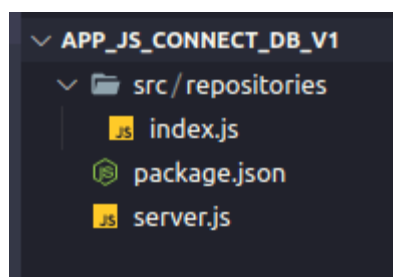
1. Crie um aplicativo Node.js, seguindo as seguintes condições:
 - a. No src do aplicativo, crie uma pasta e nomeie-a como “repositories”.
 - b. Crie um arquivo “index.js” dentro de “repositories”.
 - c. No arquivo “index.js”, crie um pool de clientes.
 - d. A título de teste, realize uma conexão ao banco de dados criado para o e-commerce e depois, libere-a.
2. Para o sistema de e-commerce, crie as seguintes queries:
 - a. Inserção de produtos.
 - b. Alteração dos dados do produto por seu ID.
 - c. Listagem de produtos, filtrando-os por categoria e marca, ordenados do menor ao maior preço.
 - d. Exclusão de produtos por seu ID.
3. Para cada query criada na questão anterior, crie uma função na pasta “repositories” que receba como parâmetros os valores necessários e execute a referida query.

Q1) A imagem a seguir ilustra o início da aplicação node.js, através do comando “npm init -y”:

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module10-database/Aula04/app_js_connect_db_v1$ npm init -y
Wrote to /home/letonio/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module10-database/Aula04/app_js_connect_db_v1/package.json:

{
  "name": "app_js_connect_db_v1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Conforme requisitado, adicionou-se um arquivo “index.js” no diretório “repositories”:



Adicionando o pacote (coleção de módulos) node.js para interface com o bancos de dados PostgreSQL: “npm install pg”

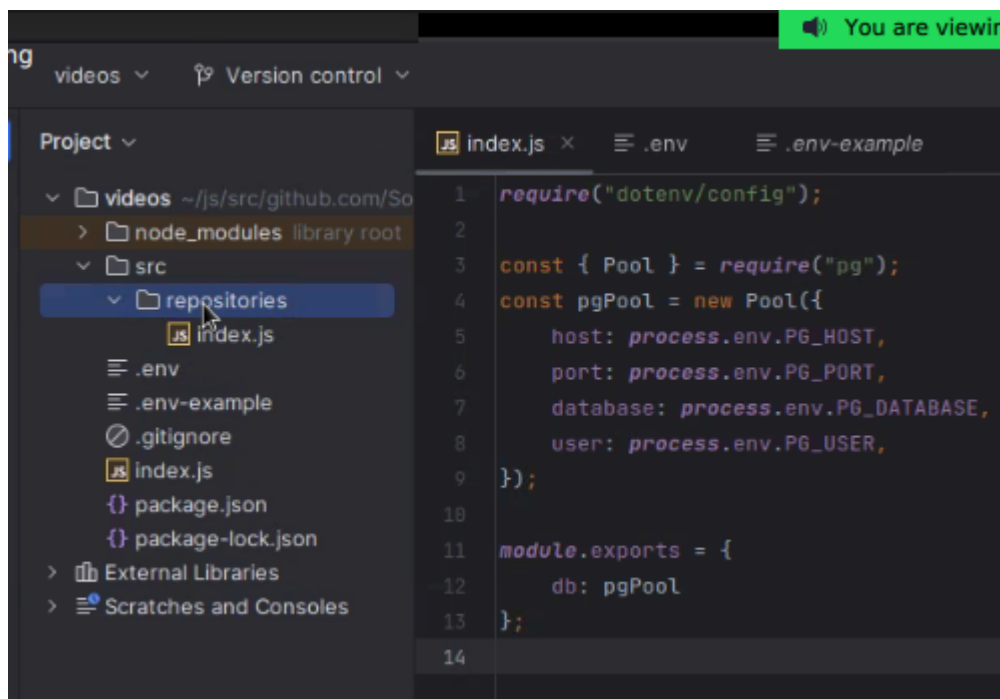
```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module10-database/Aula04/app_js_connect_db_v1$ npm install pg

added 15 packages, and audited 16 packages in 3s

found 0 vulnerabilities
```

Como criar e conectar clientes não são ações instantâneas, procura-se diminuir os custos de criar clientes constantemente. A opção adotada é criar um “pool” de conexões, que mantém um número de clientes reutilizáveis que são utilizados, liberados e devolvidos para novas utilizações. Portanto, no arquivo “index.js” adicionamos um pool:

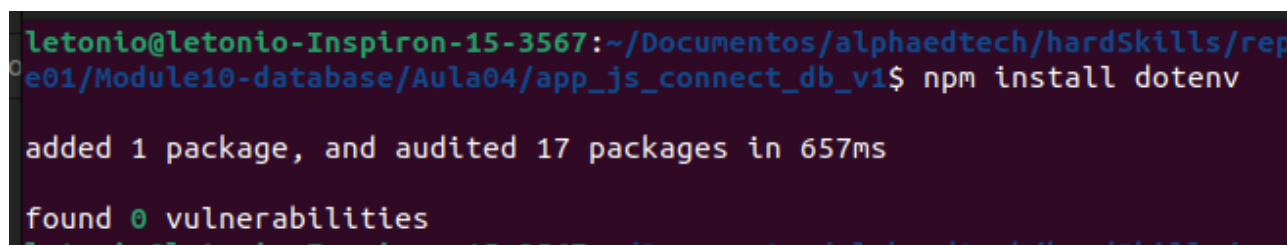
(COLOCAR UMA FOTO DO POOL)



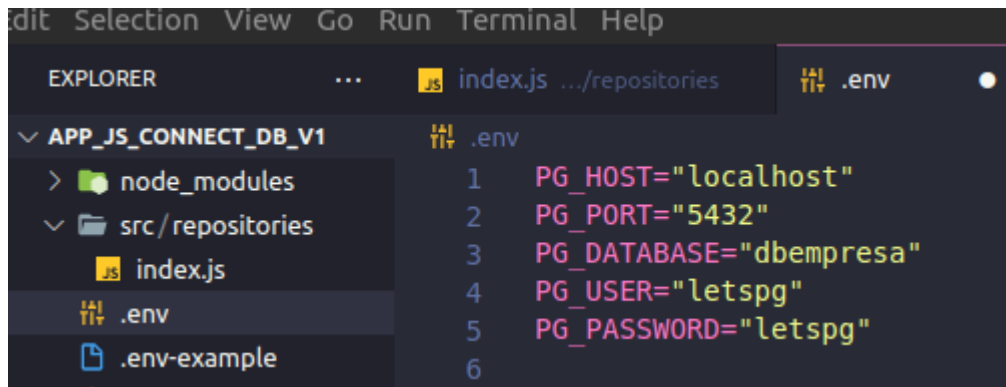
Em seguida, é necessário criar uma conexão de um cliente a partir do pool:

(COLOCAR UMA FOTO DA CONEXÃO)

Preparando-se para praticar já voltado para o projeto, adicionamos um pacote “dotenv” que permite habilitar variáveis de ambientes que são salvas em um arquivo “.env”. Assim você não fica com informações sensíveis expostas no código.



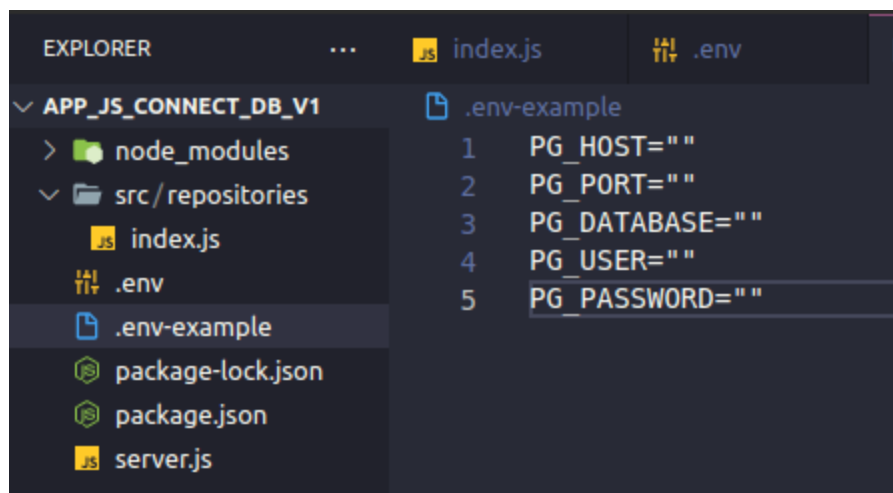
A próxima imagem ilustra o arquivo. É importante adicionar esse nome no “.gitignore” para que as informações restritas não sejam enviadas para o github. Você deixa um arquivo “.env-example” para que as pessoas que acessem o seu código saibam que precisam de um arquivo com as variáveis de ambiente para o programa funcionar.



The screenshot shows the VS Code interface with the Explorer view on the left and the Editor view on the right. The Explorer view shows the file structure of the project, including the `node_modules` directory, the `src/repositories` directory, and the `index.js` file. The Editor view shows the `.env` file with the following content:

```
1 PG_HOST="localhost"
2 PG_PORT="5432"
3 PG_DATABASE="dbempresa"
4 PG_USER="letspg"
5 PG_PASSWORD="letspg"
6
```

A próxima imagem ilustra um exemplo de “.env-example”, assim a pessoa que acessa saberá que esses 5 campos precisam ser preenchidos para o código funcionar e você não expõe os dados da sua empresa.



The screenshot shows the VS Code interface with the Explorer view on the left and the Editor view on the right. The Explorer view shows the file structure of the project, including the `node_modules` directory, the `src/repositories` directory, and the `index.js` file. The Editor view shows the `.env-example` file with the following content:

```
1 PG_HOST=""
2 PG_PORT=""
3 PG_DATABASE=""
4 PG_USER=""
5 PG_PASSWORD=""
```

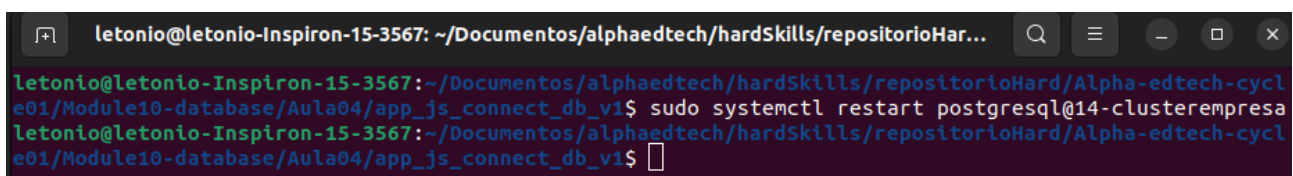
Criei um arquivo “.gitignore” e adicionei “node_modules” e “.env” nele, conforme está ilustrado a seguir:



The screenshot shows the VS Code interface with the Explorer view on the left and the Editor view on the right. The Explorer view shows the file structure of the project, including the `node_modules` directory, the `src/repositories` directory, and the `index.js` file. The Editor view shows the `.gitignore` file with the following content:

```
1 # node files
2 node_modules
3
4 # env files
5 .env
```

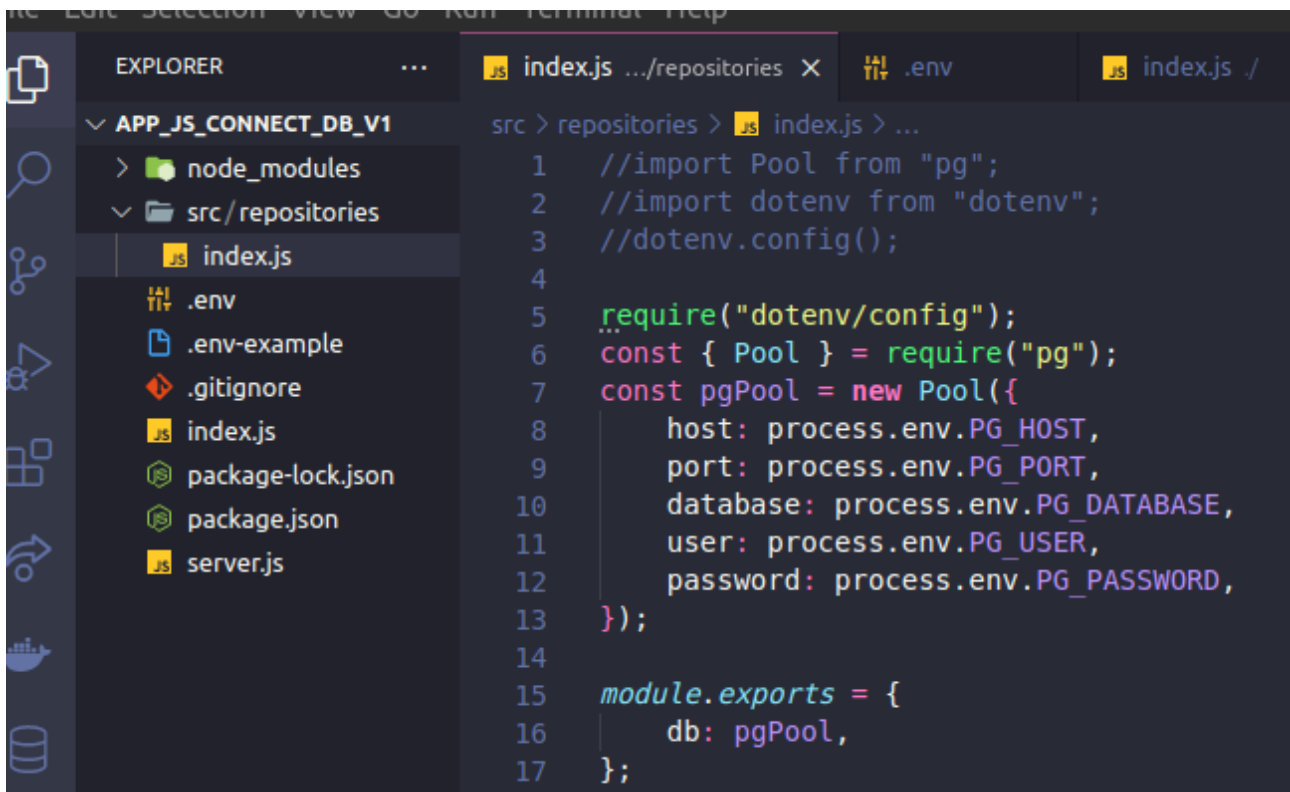
Segui o professor e reiniciei:



The screenshot shows a terminal window with the following command and output:

```
letonio@letonio-Inspiron-15-3567: ~/Documentos/alphaedtech/hardSkills/repositorioHar...
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycl
e01/Module10-database/Aula04/app_js_connect_db_v1$ sudo systemctl restart postgresql@14-clusterempresa
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycl
e01/Module10-database/Aula04/app_js_connect_db_v1$
```

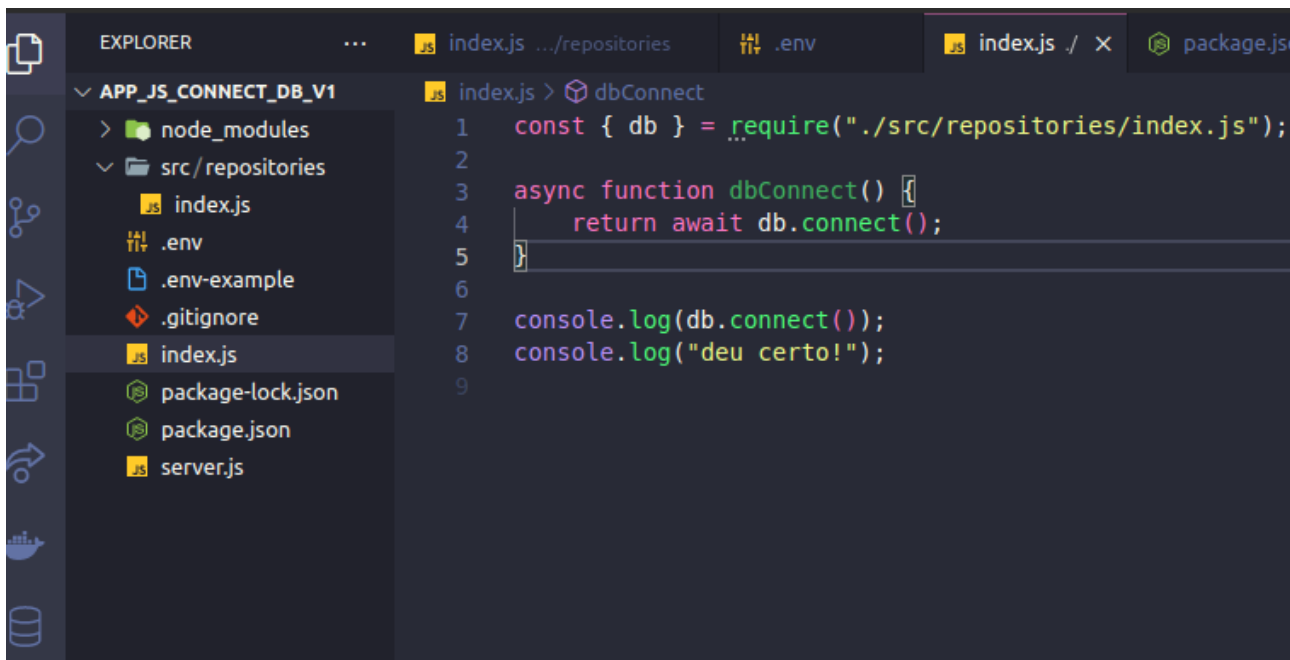
Deixando registrado o código como funcionou:



The screenshot shows the VS Code interface with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the project structure with the file `index.js` selected in the `src/repositories` directory. The Editor panel displays the code for `index.js`, which imports `pg` and `dotenv`, configures a database pool using `dotenv.config()` and `pg.Pool`, and exports the pool as `db`.

```
src > repositories > index.js > ...
1 //import Pool from "pg";
2 //import dotenv from "dotenv";
3 //dotenv.config();
4
5 require("dotenv/config");
6 const { Pool } = require("pg");
7 const pgPool = new Pool({
8   host: process.env.PG_HOST,
9   port: process.env.PG_PORT,
10  database: process.env.PG_DATABASE,
11  user: process.env.PG_USER,
12  password: process.env.PG_PASSWORD,
13 });
14
15 module.exports = {
16   db: pgPool,
17 };

```



The screenshot shows the VS Code interface with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the project structure with the file `index.js` selected in the `src` directory. The Editor panel displays the code for `dbConnect.js`, which imports `index.js` from `./src/repositories` and defines an `async function dbConnect()` that returns `await db.connect()`. It also includes `console.log` statements to verify the connection.

```
index.js > dbConnect
1 const { db } = require("../src/repositories/index.js");
2
3 async function dbConnect() {
4   return await db.connect();
5 }
6
7 console.log(db.connect());
8 console.log("deu certo!");
9

```

Executando “node index.js” a partir da raiz, o resultado foi esse:

```

letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/e01/Module10-database/Aula04/app_js_connect_db_v1$ node index.js
Promise { <pending> }
Deu certo!

```

Fiz umas modificações para fazer uma “query” parecida, só que pegando o horário, o código é mostrado a seguir:

```

index.js > ...
1  const { db } = require("../src/repositories/index.js");
2
3  async function dbConnect() {
4      return await db.connect();
5  }
6
7  //console.log(db.connect());
8  //console.log("Deu certo!");
9
10 (async function query() {
11     const conn = await dbConnect();
12
13     const sql = "SELECT NOW()";
14     const resp = await conn.query(sql);
15     console.log(resp);
16     console.log(resp.rows[0].now);
17 })();
18
19

```

O objeto de resposta “resp” contém um campo que mostra o horário atual.

```

letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/e01/Module10-database/Aula04/app_js_connect_db_v1$ node index.js
Result {
  command: 'SELECT',
  rowCount: 1,
  oid: null,
  rows: [ { now: 2023-02-11T14:28:29.701Z } ],
  fields: [
    Field {

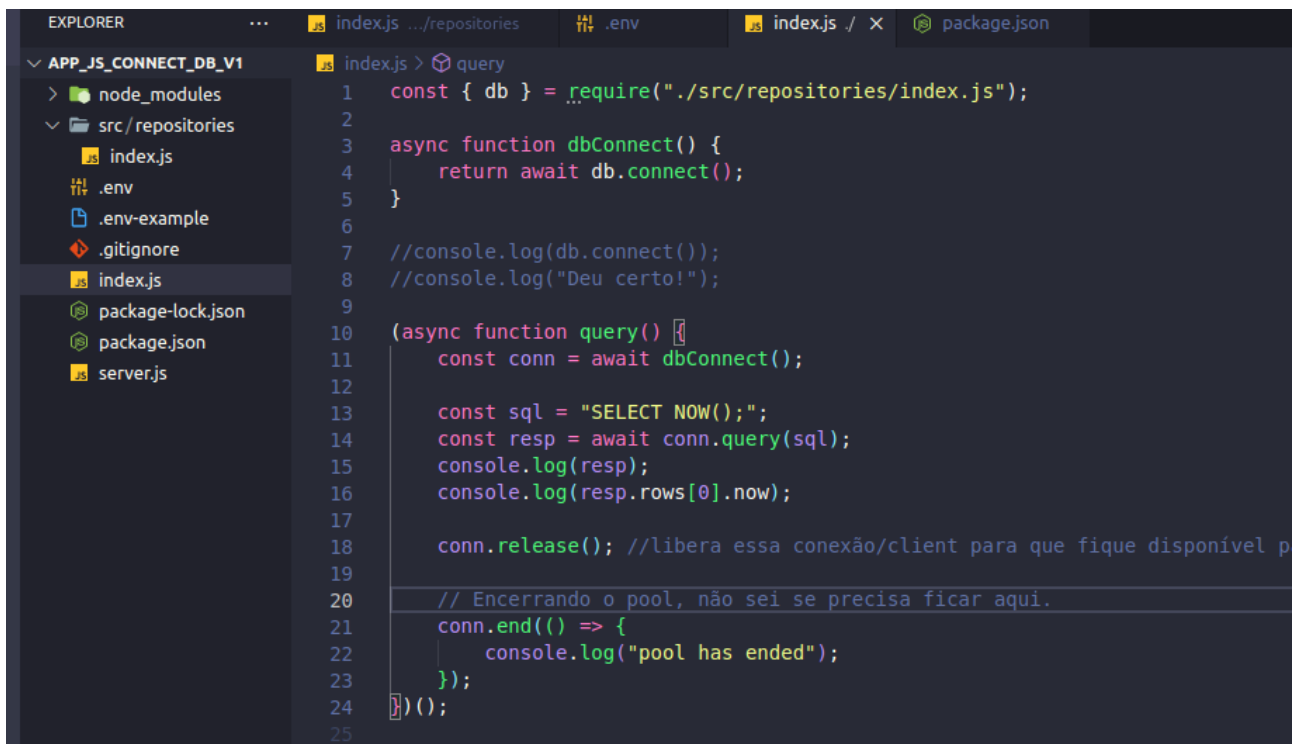
```

```

    rowAsArray: false
  ]
}
2023-02-11T14:28:29.701Z

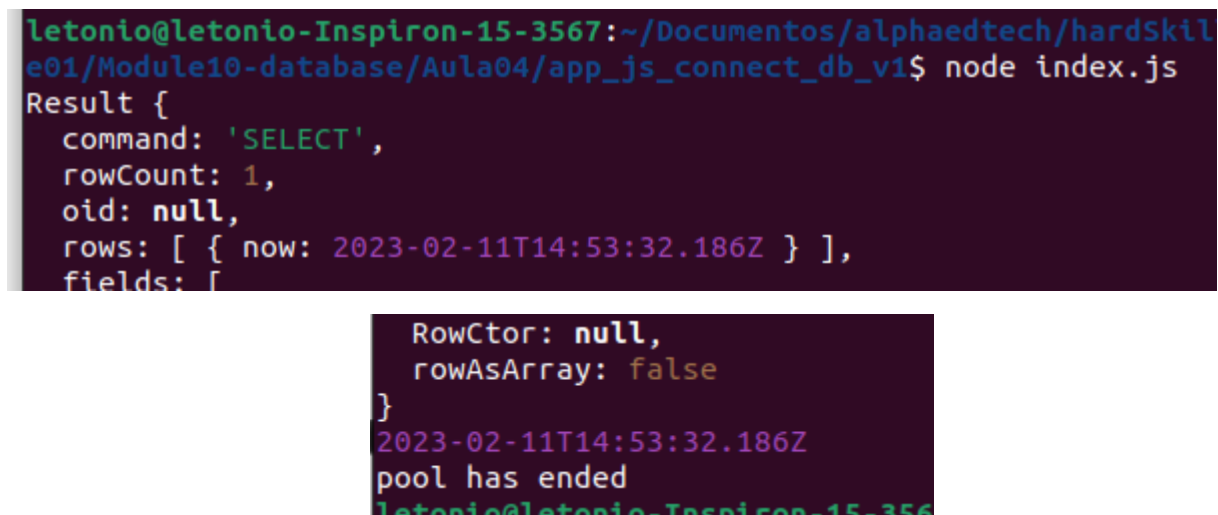
```

A conexão ainda fica ativa, adicionei algumas linhas no código para caso eu queira chamar a função, fazer a query, então liberar o cliente/pool para ficar disponível para novas conexões, adicionei também um código que libera o pool e envia uma mensagem de encerrou a conexão:



```
1  const { db } = require("../src/repositories/index.js");
2
3  async function dbConnect() {
4    return await db.connect();
5  }
6
7  //console.log(db.connect());
8  //console.log("Deu certo!");
9
10 (async function query() {
11   const conn = await dbConnect();
12
13   const sql = "SELECT NOW()";
14   const resp = await conn.query(sql);
15   console.log(resp);
16   console.log(resp.rows[0].now);
17
18   conn.release(); //libera essa conexão/client para que fique disponível p
19
20   // Encerrando o pool, não sei se precisa ficar aqui.
21   conn.end() => {
22     console.log("pool has ended");
23   });
24 })();
25
```

Executando, aparece no final do terminal:



```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkill
e01/Module10-database/Aula04/app_js_connect_db_v1$ node index.js
Result {
  command: 'SELECT',
  rowCount: 1,
  oid: null,
  rows: [ { now: 2023-02-11T14:53:32.186Z } ],
  fields: [
    RowCtor: null,
    rowAsArray: false
  ]
}
2023-02-11T14:53:32.186Z
pool has ended
letonio@letonio-Inspiron-15-3567
```

Pronto, fiz a conexão de teste, usando o pool depois liberei a conexão de teste, o que finaliza a questão Q1.

Q2) e Q3) Fiz as queries e já adicionei em funções, cada função contida dentro de um arquivo “.js”

A seguir apresenta-se imagens mostrando as funções e ela sendo aplicadas na tabela “products”, do banco de dados do “dbempresa”, usando “letspg” de usuário.

Item a) Inserção de produtos:

```
1  async function insertProduct(  
2    conn,  
3    { productName, categoryId, brandId, productPrice, productQuant }  
4  ) {  
5    const sql = `  
6      INSERT INTO products (id, name,category_id,brand_id,price,quant_available,created_at)  
7      VALUES (gen_random_uuid(), $1,$2,$3,$4,$5, now())  
8      RETURNING id;  
9    `;  
10   //verdadeir implica que a query não tinha n
```

```
11   const resp = await conn.query(sql, [  
12     productName,  
13     categoryId,  
14     brandId,  
15     productPrice,  
16     productQuant,  
17   ]);  
18  
19   // verdadeiro implica que a query não tinha n  
20   if (resp.rows.length === 0) {  
21     throw "Saving new product";  
22   }  
23  
24   return {  
25     productId: resp.rows[0].id,  
26   };  
27 }
```

A tabela a seguir mostra a função dentro de index.js que chama a função para adicionar um novo produto.

```
53  //FUNCTION INSERT NEW PRODUCT  
54  (async function query() {  
55    const conn = await dbConnect();  
56    const resp = await insertProduct(conn, {  
57      productName: "Roupa adicionada",  
58      categoryId: "c451d080-493e-4796-b7f9-770d0b40e3f5",  
59      brandId: "97df59bf-4d9d-4ec2-bace-6e474e75a218",  
60      productPrice: 333,  
61      productQuant: 5,  
62    });  
63  
64    console.log("Resposta da query realizada:", resp);  
65  
66    conn.release(); //libera essa conexão/client para que fique  
67    console.log("Conexão liberada para ser reaproveitada");
```


Para executar, basta abrir o terminal na raiz do projeto e usar “node index.js”.
O resultado obtido foi:

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioH...  
e01/Module10-database/Aula04/app_js_connect_db_v1$ node index.js  
Resposta da query realizada: { productId: 'a97669d2-de8c-4f81-ac86-9e28c2b83e78' }  
Conexão liberada para ser reaproveitada
```

O Id mostrado acima é o id do produto que acabou de ser adicionado. A próxima imagem confirma a inserção: usei o comando sql “SELECT * FROM products”

10	c4500132-e0a6-4a69-a3d4-db4c6d7007...	Ben10 novo	c451d080-493e-4796-b7f9-770d0b40e3f5	9
11	a8565d15-2808-4961-8f95-c5e1826a926e	zzzBen10 novo	c451d080-493e-4796-b7f9-770d0b40e3f5	9
12	a97669d2-de8c-4f81-ac86-9e28c2b83e78	Roupa adicionada	c451d080-493e-4796-b7f9-770d0b40e3f5	9

Item b) Alteração dos dados do produto por seu ID.

A próxima imagem mostra o código que foi inserido numa novo arquivo “.js”.

```
1  async function updateProduct(  
2    conn,  
3    { productId, productName, categoryId, brandId, productPrice, productQuant }  
4  ) {  
5    const sql = `  
6      UPDATE products SET name = $2,category_id=$3,brand_id=$4, price=$5, quant_available=$6,  
7      update_at = now()      WHERE id = $1 RETURNING *;  
8    `;  
9  }
```

```
11    const resp = await conn.query(sql, [  
12      productId,  
13      productName,  
14      categoryId,  
15      brandId,  
16      productPrice,  
17      productQuant,  
18    ]);
```

```
21    if (resp.rows.length === 0) {  
22      console.log(resp, resp.rows);  
23      throw "Update product";  
24    }  
25  
26    return {  
27      productId: resp.rows[0].id,  
28    };
```

A próxima imagem mostra o conteúdo no index.js, para fazer esse tipo de requisição:


```

71
72 //FUNCTION UPDATE PRODUCT BY ID
73 (async function query() {
74     const conn = await dbConnect();
75     const resp = await updateProduct(conn, {
76         productId: "a97669d2-de8c-4f81-ac86-9e28c2b83e78",
77         productName: "Roupa atualizada prod",
78         categoryId: "c451d080-493e-4796-b7f9-770d0b40e3f5",
79         brandId: "97df59bf-4d9d-4ec2-bace-6e474e75a218",
80         productPrice: 5000,
81         productQuant: 24,
82     });
83
84     console.log("Resposta da query realizada:", resp);
85
86     conn.release(); //libera essa conexão/client para que fique
87     console.log("Conexão liberada para ser reaproveitada");
88 })();
89

```

Usei o mesmo ID do produto que foi recém-criado (a97669d2-de8c-4f81-ac86-9e28c2b83e78)
Para execução é o mesmo procedimento: node index.js na raiz do projeto.

```

letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHar
e01/Module10-database/Aula04/app_js_connect_db_v1$ node index.js
Resposta da query realizada: { productId: 'a97669d2-de8c-4f81-ac86-9e28c2b83e78' }
Conexão liberada para ser reaproveitada

```

Conferindo com o pgAdmin:

SELECT id,name,update_at FROM products WHERE update_at is NOT NULL;

Esse comando mostra apenas algumas colunas e mostra apenas os produtos em que a coluna “update_at” não tem valor nulo, logo o resultado foi:

	Id [PK] uuid	name text	update_at timestamp without time zone
1	f0215ed3-7c67-48c0-bf89-5af90e31e5ac	Vest Royal atualizado	2023-02-11 14:40:54.006168
2	a97669d2-de8c-4f81-ac86-9e28c2b83e78	Roupa atualizada prod	2023-02-11 20:16:12.523731

Nota-se que realmente foi atualizado.

Item c) Listagem de produtos, filtrando-os por categoria e marca, ordenados do menor ao maior preço.

Para esse item, decidi fazer dois filtros. O primeiro filtra pela categoria e ordena pelo preço; no segundo, seleciona-se apenas os produtos que possuem uma marca específica, em seguida, ordena-se pelo preço, sempre do mais barato para o mais caro.

Como o código é similar ao que foi apresentado nas questões anteriores, deixarei apenas a entrada sql que foi implementada:

```

1  async function selectProductByCategory(conn, { categoryName }) {
2      const sql = `
3          SELECT products.id, products.name, products.price,
4             categories.name AS category_name, product_brands.name AS brand_name
5          FROM products
6          JOIN categories ON products.category_id = categories.id
7          JOIN product_brands ON products.brand_id = product_brands.id
8          WHERE categories.name = $1
9          ORDER BY products.price;
10     `;
11
12     //quando o sql tem parâmetro ($1,$2), query() recebe um segundo argumento
13     const resp = await conn.query(sql, [categoryName]);

```

index.js na raiz:

```

92  //FUNCTION SELECT PRODUCTS BY CATEGORY AND ORDER BY PRICE
93  (async function query() {
94      const conn = await dbConnect();
95      const resp = await selectProductByCategory(conn, {
96          categoryName: "Camisa",
97      });
98
99      console.log("Resposta da query realizada:", resp);
100
101      conn.release(); //libera essa conexão/client para que f
102      console.log("Conexão liberada para ser reaproveitada");
103  })();

```

O próximo print mostra a saída do terminal, somente produtos com categoria “Camisa” foram retornados e a ordem está do mais barato para o mais caro:

```

letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkill
e01/Module10-database/Aula04/app_js_connect_db_v1$ node index.js
Resposta da query realizada: {
  productFiltered: [
    {
      id: 'e8ece67b-f036-4f38-8e02-fa34677b0333',
      name: 'Camisa do ben 10000',
      price: '250.00',
      category_name: 'Camisa',
      brand_name: 'Prada'
    },
    {
      id: 'c14ea5c7-514c-4233-807a-a75d7bcc61ed',
      name: 'Camisa top top',
      price: '300.00',
      category_name: 'Camisa',
      brand_name: 'Prada'
    }
  ]
}
Conexão liberada para ser reaproveitada

```

As próximas imagens são referentes ao filtro seguindo a marca “Gucci”:

```
1  async function selectProductByBrand(conn, { brandName }) {
2      const sql = `
3          SELECT products.id, products.name, products.price,
4          categories.name AS category_name, product_brands.name AS brand_name
5          FROM products
6          JOIN categories ON products.category_id = categories.id
7          JOIN product_brands ON products.brand_id = product_brands.id
8          WHERE product_brands.name = $1
9          ORDER BY products.price;
10     `;
11
12     //quando o sql tem parâmetro ($1,$2), query() recebe um segundo argument
13     const resp = await conn.query(sql, [brandName]);
```

Index.js:

```
106  //FUNCTION SELECT PRODUCTS BY BRAND AND ORDER BY PRICE
107  (async function query() {
108      const conn = await dbConnect();
109      const resp = await selectProductByBrand(conn, {
110          brandName: "Gucci",
111      });
112
113      console.log("Resposta da query realizada:", resp);
114
115      conn.release(); //libera essa conexão/client para que f
116      console.log("Conexão liberada para ser reaproveitada");
117  })();
118
```

```
Module10-database/Aula04/app_js_connect_db_v1$ node index.js
Resposta da query realizada: {
  productFiltered: [
    {
      id: '55c28f24-e213-4a73-817d-2f00f8b77cb5',
      name: 'Pele sintetica',
      price: '320.00',
      category_name: 'Casaco',
      brand_name: 'Gucci'
    },
    {
      id: '0382fef8-7b1e-44f8-94fb-f4b6af77bd3d',
      name: 'Curta verão',
      price: '450.00',
      category_name: 'Saia',
      brand_name: 'Gucci'
    }
  ]
}
```

```

    {
      id: 'f0215ed3-7c67-48c0-bf89-5af90e31e5ac',
      name: 'Vest Royal atualizado',
      price: '450.00',
      category_name: 'Vestido',
      brand_name: 'Gucci'
    },
    {
      id: 'db8fb2d1-6c0a-4ff0-ba75-db50e2b18b7f',
      name: 'Leve croche',
      price: '540.00',
      category_name: 'Casaco',
      brand_name: 'Gucci'
    }
  ]
}

```

Nota-se que apenas marca Gucci foi retornada e em ordem crescente de preço.

Item d) Exclusão de produtos por seu ID.

Por fim, mostra-se a exclusão, vamos excluir o elemento que foi inserido no início dessa questão, cujo id é: "a97669d2-de8c-4f81-ac86-9e28c2b83e78"

```

1  async function deleteProductById(conn, { productId }) {
2    const sql = `
3      DELETE FROM products WHERE id = $1 RETURNING id;
4    `;
5    //quando o sql tem parâmetro ($1,$2), query() recebe u
6    const resp = await conn.query(sql, [productId]);
7
8    // verdadeiro implica que a query não tinha nenhum com
9    if (resp.rows.length === 0) {
10     throw "Delete product";
11   }
12
13   return {
14     productId: resp.rows[0].id,
15   };
16 }

```

index.js:

```

120
121 //FUNCTION DELETE PRODUCT BY ID
122 (async function query() {
123     const conn = await dbConnect();
124     try {
125         const resp = await deleteProductById(conn, {
126             productId: "a97669d2-de8c-4f81-ac86-9e28c2b83e78",
127         });
128
129         console.log("Resposta da query realizada:", resp);
130     } catch (err) {
131         console.error(`Error deleting product: ${err.message}`);
132     } finally {
133         conn.release(); //libera essa conexão/client para que fique
134     }
135
136     console.log("Conexão liberada para ser reaproveitada");
137 })();
138

```

Se usarmos novamente aquele filtro do item b, nota-se que o elemento não está mais lá após a execução do “node index.js” no terminal:

```

letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/A
e01/Module10-database/Aula04/app_js_connect_db_v1$ node index.js
Resposta da query realizada: { productId: 'a97669d2-de8c-4f81-ac86-9e28c2b83e78' }
Conexão liberada para ser reaproveitada

```

	Id [PK] uuid	name text	update_at timestamp without time zone
1	f0215ed3-7c67-48c0-bf89-5af90e31e5ac	Vest Royal atualizado	2023-02-11 14:40:54.006168

Aqui fizemos o hard delete, porém uma opção seria usar o campo “delete_at” que permite um soft delete. O dado não é apagado realmente do sistema. Mas nessa atividade, optamos por deletar.