**Universidade Federal de São Carlos – Campus Sorocaba**

**Bacharelado em Ciência da Computação**

**Sistemas de Bancos de Dados**

Profa. Sahudy Montenegro

**Projeto Integrado – Bases de dados brasileiras**

Tema 6 – Cadastro brasileiro de cursos universitários

Grupo 4:

José Vitor Novaes dos Santos RA: 743556

Marcus Vinícius Natrielli Garcia RA: 743578

Victor Fernandes de Oliveira Brayner RA: 743600

William Giacometti Dutra de Oliveira RA: 743606

**Fase Final: 12/06/2019**

Sumário

[1. Descrição do minimundo 3](#_Toc11005212)

[2. Definição de esquema de BD 3](#_Toc11005213)

[3. Especificação do banco e das consultas em SQL 4](#_Toc11005214)

[3.1 Código SQL do banco de dados: 4](#_Toc11005215)

[3.2 Código SQL das consultas (sem otimização): 5](#_Toc11005216)

[4. Amostragem das consultas sem otimização 7](#_Toc11005217)

[5. Populando o Banco de Dados 7](#_Toc11005218)

[6. Indexação e otimização das consultas 8](#_Toc11005219)

[7. Programação com Banco de Dados 12](#_Toc11005220)

[8. Controle de acesso de usuários 12](#_Toc11005221)

[9. Informações sobre o projeto 14](#_Toc11005222)

[10. Considerações finais 14](#_Toc11005223)

1. Descrição do minimundo

São muitos os cursos universitários existentes no Brasil, e, a partir disso, torna-se necessária a existência de um sistema para armazenar e organizar eles e suas informações. As Instituições de ensino contam com uma sigla identificadora única, seu nome completo e a sua natureza administrativa (pública ou privada) e sua nota no índice Geral de Cursos (IGC, mas informalmente chamado de “Nota do Mec”) pelo Ministério da Educação como atributos.

Os campi são identificados pela sigla de sua instituição e pelo seu nome, e possuem como informações o estado e a cidade em que se localizam. Por fim, os cursos possuem nome, área do conhecimento (ciências exatas e da terra, ciências biológicas, engenharias, ciências da saúde, ciências agrárias, ciências sociais aplicadas, ciências humanas, linguística, letras e artes, e outros), nota no ENADE, grau do curso (bacharelado, licenciatura, mestrado, doutorado ou tecnólogo), o turno (matutino, noturno, integral ou vespertino), sua duração em semestres, o ano em que foi criado.

As notas IGC e ENADE variam de 0 a 5. Um curso é unicamente identificado a partir da sigla da instituição e do nome do campus ao qual pertence e do seu nome, de maneira que o campus de uma instituição só pode possuir um curso com um determinado nome. O ano de criação do curso deve ser maior que 1980, e não pode ultrapassar o ano atual.

Uma instituição pode ter vários campi, porém um campus só pertence a uma instituição. Além disso, um campus pode ter vários cursos, e de um campus para o outro, um certo curso pode ter informações diferentes.

1. Definição de esquema de BD

Modelo relacional do esquema do banco de dados proposto, baseado no código SQL presente no apêndice:

instituicao(sigla, nome\_instituicao, natureza\_administrativa, nota\_igc)

campus(nome\_campus, estado, cidade, sigla)

sigla referencia instituição (sigla)

curso(nome\_curso, area, nota\_enade, grau, turno, duracao, ano\_criacao, campus, sigla)

campus referencia campus(nome\_campus)

sigla referencia campus (sigla)

1. Especificação do banco e das consultas em SQL

Nesta seção é especificado o código SQL do banco de dados e das consultas, estas que também contam com seus respectivos enunciados, campos de busca, operadores de comparação e campos de visualização dos resultados.

* 1. Código SQL do banco de dados:

CREATE TABLE instituicao (

sigla varchar(10),

nome\_instituicao varchar(55) NOT NULL,

natureza\_administrativa varchar(7) NOT NULL CHECK (natureza\_administrativa IN ('Pública', 'Privada')),

nota\_IGC smallint NOT NULL CHECK (nota\_IGC >= 0 AND nota\_IGC <= 5),

PRIMARY KEY(sigla)

);

CREATE TABLE campus(

nome\_campus varchar(55) NOT NULL,

estado char(2) NOT NULL CHECK (estado IN ('AC', 'AL', 'AP', 'AM', 'BA', 'CE', 'DF', 'ES', 'GO', 'MA', 'MT', 'MS', 'MG', 'PA', 'PB', 'PR', 'PE', 'PI', 'RJ', 'RN', 'RS', 'RO', 'RR', 'SC', 'SP', 'SE', 'TO')),

cidade varchar(55) NOT NULL,

sigla varchar(10),

PRIMARY KEY(sigla, nome\_campus),

FOREIGN KEY(sigla) REFERENCES instituicao(sigla) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE curso(

nome\_curso varchar(55) NOT NULL,

area varchar(30) NOT NULL CHECK (area IN ('Ciências Exatas e da Terra', 'Ciências Biológicas', 'Engenharias', 'Ciências da Saúde', 'Ciências Agrárias', 'Ciências Sociais Aplicadas', 'Ciências Humanas', 'Linguística, Letras e Artes', 'Outros')),

nota\_enade smallint NOT NULL CHECK (nota\_enade >= 0 AND nota\_enade <= 5),

grau varchar(12) NOT NULL CHECK (grau IN ('Bacharelado', 'Licenciatura', 'Mestrado', 'Doutorado', 'Tecnólogo')),

turno varchar(10) NOT NULL CHECK (turno IN ('Matutino', 'Integral', 'Noturno', 'Vespertino')),

duracao smallint NOT NULL CHECK (duracao > 0 AND duracao <= 12),

ano\_criacao smallint NOT NULL CHECK (ano\_criacao <= DATE\_PART('YEAR', CURRENT\_DATE) AND ano\_criacao > 1980),

campus varchar(55) NOT NULL,

sigla varchar(10) NOT NULL,

PRIMARY KEY(sigla, campus, nome\_curso),

FOREIGN KEY(sigla, campus) REFERENCES campus(sigla, nome\_campus) ON DELETE CASCADE ON UPDATE CASCADE

);

## Código SQL das consultas (sem otimização):

1ª Consulta: Listar sigla da instituição, nome do campus, estado, cidade, nome, grau de especialização, turno e duração em semestres e nota no ENADE de cursos em instituições de uma certa natureza administrativa e que possuam um determinado termo no nome.

Campos de visualização do resultado: curso.sigla, campus, estado, cidade nome\_curso, grau, turno, duracao e nota\_enade.

Campos de busca (ou das condições): nome\_curso (relativa) e natureza\_administrativa (absoluta).

Operadores de condições: nome\_curso (LIKE), natureza\_administrativa (=).

SQL:

SELECT curso.sigla, campus, estado, cidade, nome\_curso, grau, turno, duracao, nota\_enade

FROM instituicao, campus, curso

WHERE instituicao.sigla = campus.sigla AND campus.nome\_campus = curso.campus AND campus.sigla = curso.sigla AND natureza\_administrativa = <natureza\_fornecida> AND nome\_curso LIKE '%<termo\_fornecido>%'

ORDER BY curso.sigla, estado, cidade;

2ª Consulta: Ranquear campi que criaram mais cursos entre dois determinados anos, fornecendo a sigla, o nome e a natureza administrativa da instituição atrelada a eles, seus nomes, estados, cidades e quantidade de cursos.

Campos de visualização do resultado: a.sigla, nome\_instituicao, natureza\_administrativa, a.nome\_campus, estado, cidade e cont.

Campos de busca (ou das condições): ano\_criacao (absoluta).

Operadores de condições: ano\_criacao (<=, >=).

SQL:

SELECT a.sigla, nome\_instituicao, natureza\_administrativa, a.nome\_campus, estado, cidade, cont

FROM

(SELECT campus.sigla, campus.nome\_campus, estado, cidade, COUNT(\*) AS cont

FROM campus, curso WHERE campus.sigla = curso.sigla AND campus.nome\_campus = curso.campus AND ano\_criacao <= <ano\_fornecido1> AND ano\_criacao >= <ano\_fornecido2>

GROUP BY campus.sigla, campus.nome\_campus) AS a,

instituicao

WHERE instituicao.sigla = a.sigla

ORDER BY cont DESC, sigla;

# Populando o Banco de Dados

Foram enviados os arquivos de Backup do banco de dados, com os dados das tabelas propostas para o trabalho integrado, junto da tabela registro\_backup, que possui as informações solicitadas sobre quantidade de dados em cada tabela e seus tamanhos em bytes; e o *script*, de autoria própria do grupo, que gerou a grande maioria dos dados do banco.

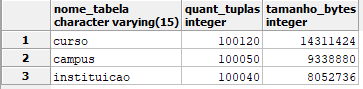


Figura 1. Tabela registro\_backup, com a quantidade de registros e o tamanho das tabelas do banco de dados

Os dados com nomes reais e formatados para serem apresentados posteriormente na aplicação a ser desenvolvida foram criados sem uso de *scripts*, usando a inserção manual. Os dados genéricos foram gerados por um script desenvolvido na linguagem C++.

# Indexação e otimização das consultas

Versão do PostgreSQL, utilizando o comando select version(): "PostgreSQL 9.6.13, compiled by Visual C++ build 1800, 64-bit".

Dadas as duas consultas propostas, seguem abaixo as tabelas 1 e 2 com a execução delas sem utilizar técnicas de indexação e otimização.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1ª exec. | 2ª exec. | 3ª exec. | 4ª exec. | 5ª execução |
| Tempo utilizado | 6.2 segs. | 5.8 segs. | 5.7 segs. | 5.7 segs. | 6.1 segundos |

*Tabela 1. Apresentação de resultados de execução da primeira consulta*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1ª exec. | 2ª exec. | 3ª exec. | 4ª exec. | 5ª execução |
| Tempo utilizado | 10.7 segs. | 10.5 segs. | 10.6 segs. | 10.4 segs. | 10.5 segundos |

*Tabela 2. Apresentação de resultados de execução da segunda consulta*

Para a primeira consulta, na Tabela 1, foi buscado o termo “Ciência” no nome dos cursos e a natureza administrativa escolhida foi “Pública”. Para a segunda consulta, na Tabela 2, o intervalo de tempo, em anos, utilizado foi de 1980 até 2019.

As médias de tempo de execução das consultas são:

1ª Consulta: 5.9 segundos

2ª Consulta: 10.54 segundos

Para a otimização das consultas, o grupo utilizou duas técnicas principais: a indexação (criação de índices com árvore B e índice de texto completo (GIN)) e a sintonização das consultas, reescrevendo as mesmas para evitar operações com muitos acessos a disco.

Para a primeira consulta, reescrevendo a mesma, obtivemos:

SELECT curso.sigla, campus, estado, cidade, nome\_curso, grau, turno, duracao, nota\_enade

FROM

(SELECT sigla FROM instituicao WHERE natureza\_administrativa = <natureza\_fornecida>) AS instituicao,

campus,

(SELECT sigla, campus, nome\_curso, grau, turno, duracao, nota\_enade

FROM curso

WHERE nome\_curso LIKE '%<termo\_fornecido>%') AS curso

WHERE instituicao.sigla = campus.sigla AND campus.nome\_campus = curso.campus AND campus.sigla = curso.sigla

ORDER BY curso.sigla, estado, cidade;

A segunda consulta, após ser reescrita, ficou:

SELECT a.sigla, nome\_instituicao, natureza\_administrativa, a.nome\_campus, estado, cidade, cont

FROM

(SELECT campus.sigla, campus.nome\_campus, estado, cidade, COUNT(\*) AS cont

FROM campus inner join curso on (campus.sigla = curso.sigla AND campus.nome\_campus = curso.campus AND ano\_criacao <= <ano\_fornecido1> AND ano\_criacao >= <ano\_fornecido2>)

GROUP BY campus.sigla, campus.nome\_campus) AS a,

instituicao

WHERE instituicao.sigla = a.sigla

ORDER BY cont DESC, sigla;

Os índices criados para ambas as consultas foram:

CREATE EXTENSION pg\_trgm;

CREATE UNIQUE INDEX natureza\_insti ON instituicao (sigla, natureza\_administrativa);

CREATE INDEX gin\_nome\_curso ON curso USING gin(sigla gin\_trgm\_ops, campus gin\_trgm\_ops, nome\_curso gin\_trgm\_ops);

CREATE INDEX fk\_curso ON curso(sigla, campus);

CREATE INDEX fk\_campus ON campus(sigla);

CREATE INDEX ano ON curso (sigla, campus, ano\_criacao);

Sendo que pg\_trgm não é um índice e sim uma opção para ativar uma estrutura de dados trigrama, que é usada no índice GIN.

Após a avaliação e os conselhos da professora Sahudy sobre as técnicas de indexação e otimização que o grupo utilizou para as consultas, pudemos reavaliar os métodos empregados e compreender melhor as técnicas que deveríamos utilizar.

Figura 2. Explain analyse da segunda consulta sem índices

A execução das consultas otimizadas é expressa pelas tabelas 3 e 4:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1ª exec. | 2ª exec. | 3ª exec. | 4ª exec. | 5ª execução |
| Tempo utilizado | 14 mseg. | 15 mseg. | 14 mseg. | 15 mseg. | 16 milissegundos |

*Tabela 3. Apresentação de resultados de execução da primeira consulta otimizada*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1ª exec. | 2ª exec. | 3ª exec. | 4ª exec. | 5ª execução |
| Tempo utilizado | 10.3 segs. | 10.3 segs. | 10.3 segs. | 10.4 segs. | 10.4 segundos |

*Tabela 4. Apresentação de resultados de execução da segunda consulta otimizada*

As médias de tempo de execução das consultas são:

1ª Consulta: 14.8 milissegundos

2ª Consulta: 10.34 segundos

As tabelas 5 e 6 comparam a média de execução entre as consultas:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Consulta inicial | Consulta otimizada | Diferença (%) |
| Tempo de execução | 5.9 segundos | 14.8 milissegundos | 99,7 |

*Tabela 5. Apresentação de resultados da primeira consulta*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Consulta inicial | Consulta otimizada | Diferença (%) |
| Tempo de execução | 10.54 segundos | 10.34 segundos | 1,9 |

*Tabela 6. Apresentação de resultados da segunda consulta*

Buscamos utilizar os métodos de indexação e de sintonização de consultas para otimizar a execução das consultas pois a criação de índices é a ferramenta mais conhecida na criação de banco de dados para aperfeiçoar o tempo de execução e busca de dados de uma consulta; enquanto a sintonização permite que uma mesma consulta seja executada de forma mais eficiente dependendo do contexto no qual ela é utilizada, utilizando operadores menos custosos e realizando menos acessos a disco.

Seguem os planos de consulta das seleções antes e depois de serem otimizadas, junto dos comentários referentes a cada consulta.

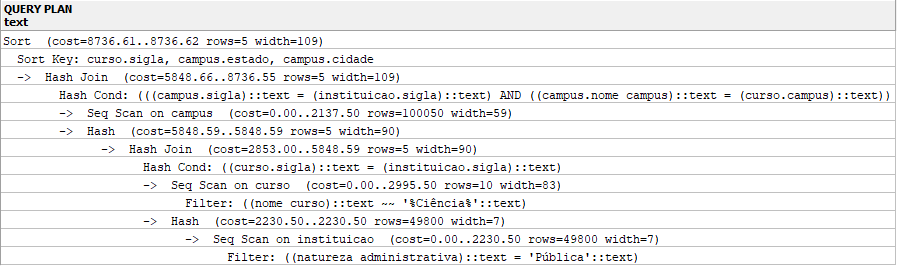


Figura 2. Plano de consulta da primeira consulta, sem otimização

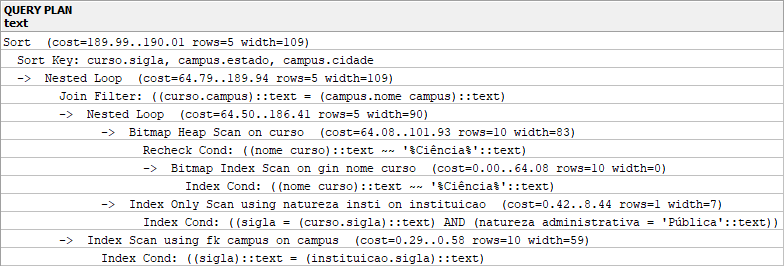


Figura 3. Plano de consulta da primeira consulta, com otimização

O grupo pôde observar que, na execução desta consulta, a busca, na tabela curso pelo termo fornecido era extremamente custosa, pois utiliza o operador LIKE ao mesmo tempo em que realiza uma busca sequencial no produto cartesiano das três tabelas presentes no banco de dados.

Para contornar esta situação, criamos índices de texto completo que permitissem uma busca inteligente pelos nomes dos cursos, diminuindo a quantidade de vezes em que o operador LIKE era utilizado, e índices para a natureza administrativa das instituições e para as chaves estrangeiras das tabelas. Também utilizamos, na sintonização das consultas, o uso de subconsultas, de maneira a juntar as tabelas gerando uma quantidade menor de tuplas do que o produto cartesiano geraria.

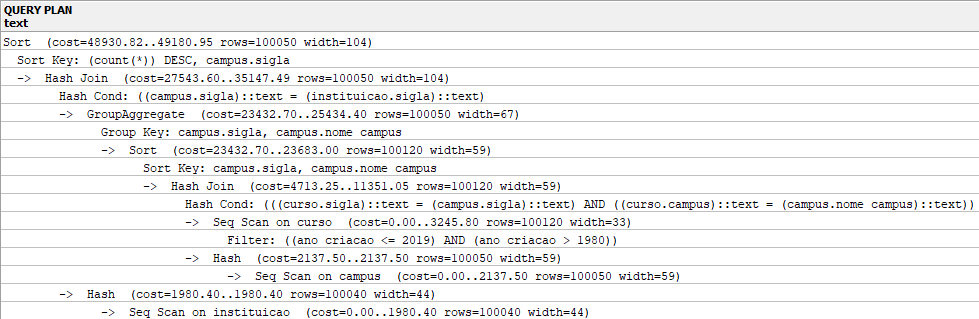


Figura 4. Plano de consulta da segunda consulta, sem otimização

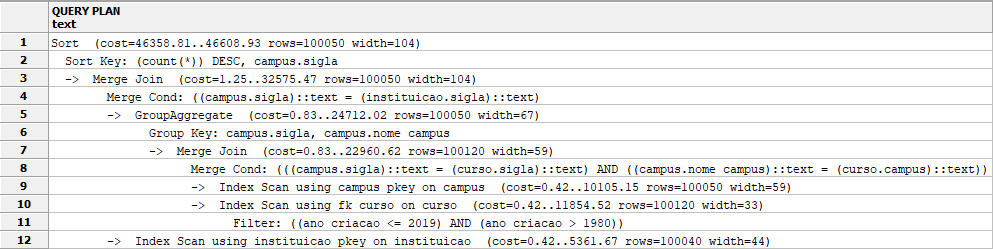


Figura 5. Plano de consulta da segunda consulta, com otimização

Nesta segunda consulta, o grupo pôde notar que uma possível indexação no atributo ano\_criacao poderia facilitar a busca pelos cursos criados dentro do período fornecido, além de os demais índices criados para a primeira consulta poderem ser empregados para aperfeiçoar as condições de junção das tabelas. Além disso, possíveis sintonizações para evitar o produto cartesiano, utilizando subconsultas como na primeira consulta, também seria de grande auxílio.

Porém, mesmo após a utilização das técnicas citadas, o tempo de execução da consulta permaneceu igual, tendo pouca ou nenhuma diferença para a consulta não otimizada, mesmo que o plano da consulta tenha sido modificado e tenha utilizado os índices propostos.

O grupo pediu auxílio para a professora Sahudy, testando outras possíveis dicas e métodos que pudessem aperfeiçoar a execução da consulta, como o uso do operador inner join. Contudo, isto não foi efetivo o bastante para melhorar o tempo da consulta.

1. Programação com Banco de Dados

Para a parte de funções e programação interna do banco de dados, foram criadas as seguintes *Stores Procedures*:

* busca\_nome\_natureza (nome, natureza)

CREATE OR REPLACE FUNCTION busca\_nome\_natureza(nome varchar(55), natureza varchar(7))

RETURNS TABLE(

sigla VARCHAR(10),

estado CHAR(2),

campus VARCHAR(55),

cidade VARCHAR(55),

nome\_curso VARCHAR(55),

grau VARCHAR(12),

turno VARCHAR(10),

duracao SMALLINT,

nota\_enade SMALLINT

)

AS $$

DECLARE

sql text;

BEGIN

sql:='SELECT curso.sigla, estado, campus, cidade, nome\_curso, grau, turno, duracao, nota\_enade

FROM (SELECT sigla FROM instituicao WHERE natureza\_administrativa = '''||natureza||''') AS instituicao, campus,

(SELECT sigla, campus, nome\_curso, grau, turno, duracao, nota\_enade FROM curso WHERE nome\_curso LIKE ''%'||nome||'%'') AS curso

WHERE instituicao.sigla = campus.sigla AND campus.nome\_campus = curso.campus AND campus.sigla = curso.sigla

ORDER BY curso.sigla, estado, cidade;';

RETURN QUERY EXECUTE sql;

END;

$$ LANGUAGE 'plpgsql';

Exemplo de utilização da *Stored Procedure*:

SELECT \* FROM busca\_nome\_natureza('C', 'Pública');

Esta função foi criada para realizar a execução da primeira consulta proposta para este trabalho, a busca de cursos a partir de um determinado termo presente em seu nome e da natureza administrativa da instituição ao qual ele está vinculado.

Como entrada, a função recebe o termo a ser buscado nos nomes dos cursos (nome) e a natureza administrativa a ser buscada na tabela instituicao (natureza). Como saída, a função retorna a nova tabela com os resultados da seleção.

* ranqueamento\_campi(ano\_inicial, ano\_final)

CREATE OR REPLACE FUNCTION ranqueamento\_campi(ano\_inicial INTEGER, ano\_final INTEGER)

RETURNS TABLE(

sigla VARCHAR(10),

nome\_instituicao VARCHAR(55),

natureza\_administrativa VARCHAR(55),

nome\_campus VARCHAR(55),

estado CHAR(2),

cidade VARCHAR(55),

cont BIGINT

)

AS $$

DECLARE

sql text;

BEGIN

sql:='SELECT a.sigla, nome\_instituicao, natureza\_administrativa, a.nome\_campus, estado, cidade, cont

FROM (SELECT campus.sigla, campus.nome\_campus, estado, cidade, COUNT(\*) AS cont

FROM campus INNER JOIN curso ON (campus.sigla = curso.sigla AND campus.nome\_campus = curso.campus AND ano\_criacao > '||ano\_inicial||'AND ano\_criacao <= '||ano\_final||')

GROUP BY campus.sigla, campus.nome\_campus) AS a, instituicao

WHERE instituicao.sigla = a.sigla ORDER BY cont DESC, sigla;';

RETURN QUERY EXECUTE sql;

END;

$$ LANGUAGE 'plpgsql';

Exemplo de utilização da *Stored Procedure*:

SELECT \* FROM ranqueamento\_campi(1980, 2019);

Esta função foi criada para realizar a execução da segunda consulta proposta para este trabalho, o ranqueamento de campus que criaram mais cursos em um determinado período, em anos.

Como entrada, a função recebe o ano inicial (ano\_inicial) e o ano final (ano\_final) que delimitam o período a ser considerado. Como saída, a função retorna a nova tabela com os resultados da seleção.

1. Controle de acesso de usuários

Essa seção descrever as diferentes visões de usuários presentes no banco de dados e suas relações com as tabelas, como mostrado abaixo na tabela 7.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Administrador | Gerente | Usuário |
| Campus | S, I, U, D | S, U | S |
| Curso | S, I, U, D | S, U | S |
| Instituição | S, I, U, D | S, U | S |

*Tabela 7. Perfis de usuários, seu papel e privilégios*

*(SELECT (S), INSERT (I), DELETE (D) e UPDATE (U))*

Script:  
--Cria um usuário com login e senha, que pode criar outros usuários e bancos de dados

CREATE USER administrador WITH NOINHERIT PASSWORD 'admin' CREATEDB CREATEROLE;

--Dá todos os privilégios nas 3 tabelas especificadas ao usuário administrador e permite que ele dê esses privilégios a usuários que ele criar

GRANT ALL PRIVILEGES ON campus, curso, instituicao TO administrador WITH GRANT OPTION;

--Da todos os privilégios do usuário postgres para o usuário administrador

GRANT postgres TO administrador;

--Seleciona o usuário administrador

SET ROLE administrador;

--Com o usuário administrador selecionado cria o usuário gerente com login e senha, que pode criar apenas outros usuários

CREATE USER gerente WITH NOINHERIT PASSWORD '12345678' CREATEROLE;

--Dá privilégios de leitura e atualização ao usuário gerente nas 3 tabelas especificadas e permite que ele dê esses privilégios a usuários que ele criar

GRANT SELECT, UPDATE ON campus, curso, instituicao TO gerente WITH GRANT OPTION;

--Seleciona o usuário gerente

SET ROLE gerente;

--Cria um papel chamado usuario

CREATE ROLE usuario WITH NOINHERIT;

--Garante privilégios de apenas leitura ao papel usuario

GRANT SELECT ON campus, curso, instituicao TO usuario;

A aplicação utiliza apenas o papel usuario para as consultas e para conectar com o banco de dados foi criada uma conexão com o JDBC em uma classe Java, seguindo o Padrão de projeto DAO. A partir daí, os métodos dessa classe representam as consultas que são utilizadas na aplicação.

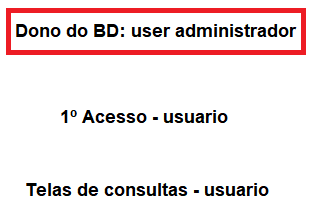


Figura 5. Conexão utilizando o perfil de usuário

1. Informações sobre o projeto
2. Considerações finais