

Atividade Prática Supervisionada - APS

Aluno: Cristiano Koxne

RA: 1920251

1. Escolha de uma aplicação com dados correlacionados.

Aplicação desenvolvida em paralelo com disciplina de Gestão de Projeto de software, oriundo da necessidade de modelar um banco de dados para armazenar os dados oriundos da aplicação, graças a isso, decidi desenvolver as duas disciplinas em paralelo, se complementando, abrangendo assim as duas áreas: gestão de projetos e banco de dados.

A aplicação servirá como facilitador de reservas de salas e equipamentos da Universidade Tecnológica Federal do Paraná campus Pato Branco. A aplicação tem o intuito de facilitar os alunos e interessados a reserva de salas ou ambientes, bem como, diminuir a burocracia que atualmente existe no processo de reserva de ambientes.

A aplicação será desenvolvida para dispositivos móveis, com acesso irrestrito de qualquer pessoa com cadastro prévio, ou qualquer pessoa com algum tipo de vínculo com o campus. A aplicação irá possuir:

- Tela de login para controle de acesso,
- Seção para pesquisa de dia e horário,
- Mapa aéreo da universidade com indicação de cores para salas ocupadas no respectivo horário de consulta,
- Clique interativo na sala de desejo de reserva.
- Cadastro de reserva em caso de vaga, gerar comprovante de reserva para comprovação futura.

A título de curiosidade, o protótipo da aplicação que norteou o desenvolvimento do banco desenvolvida em paralelo porém na matéria de Gestão de Projeto de Software está disponível no link abaixo:

<https://www.figma.com/file/Te9OgavQUjKPP1DE6ZbbIn/prot%C3%B3tipo-UTFPR-Reservas?node-id=0%3A1>

2. Elaboração de um diagrama Entidade-Relacionamento para a aplicação.

Para a modelagem do diagrama Modelo entidade Relacionamento foi utilizado o software online diagrams.neto modelo conceitual criado é apresentado na figura 1.

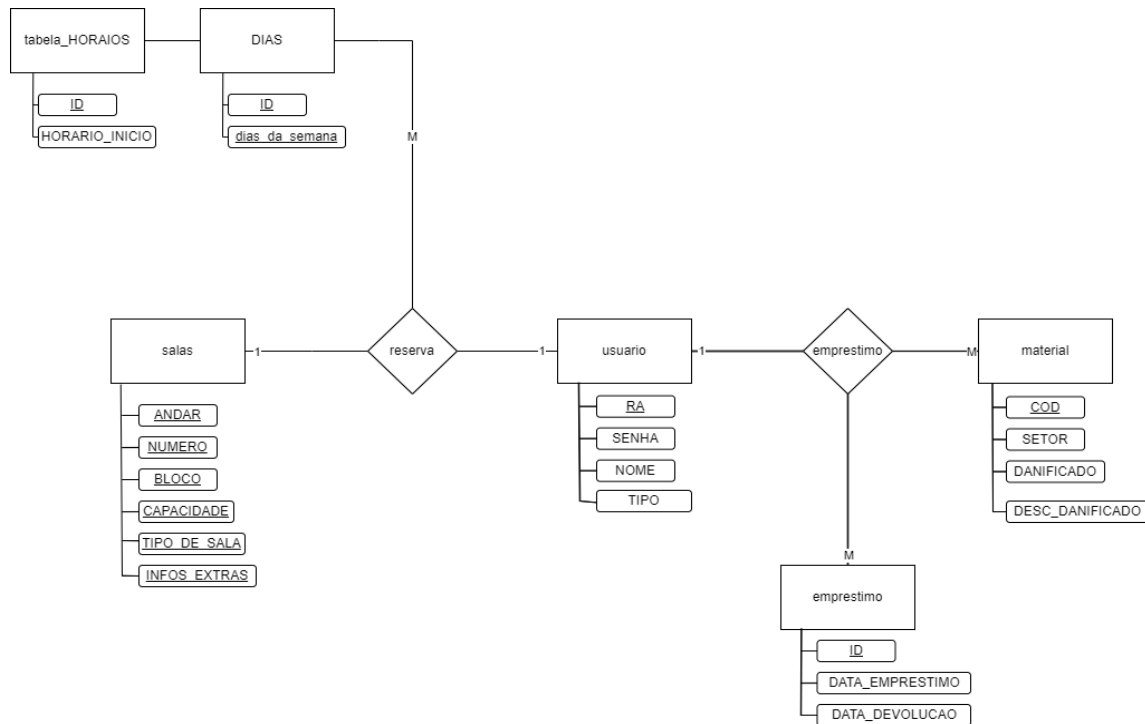


Figura 1 - Modelo Entidade relacionamento (MER) desenvolvido para o sistema

3. Mapeamento para o modelo relacional, indicando as restrições de integridade.

Em posse do modelo conceitual, foi elaborado o modelo relacional utilizando da aplicação online whimsical.com, respeitando a modelagem relacional proposta para a aplicação, e as restrições de entidade para cada tabela criada. Foi respeitado os conceitos da estrutura relacional onde certos dados não podem ser nulos e alguns precisam ser não nulos únicos, o modelo criado é representado na figura 2.

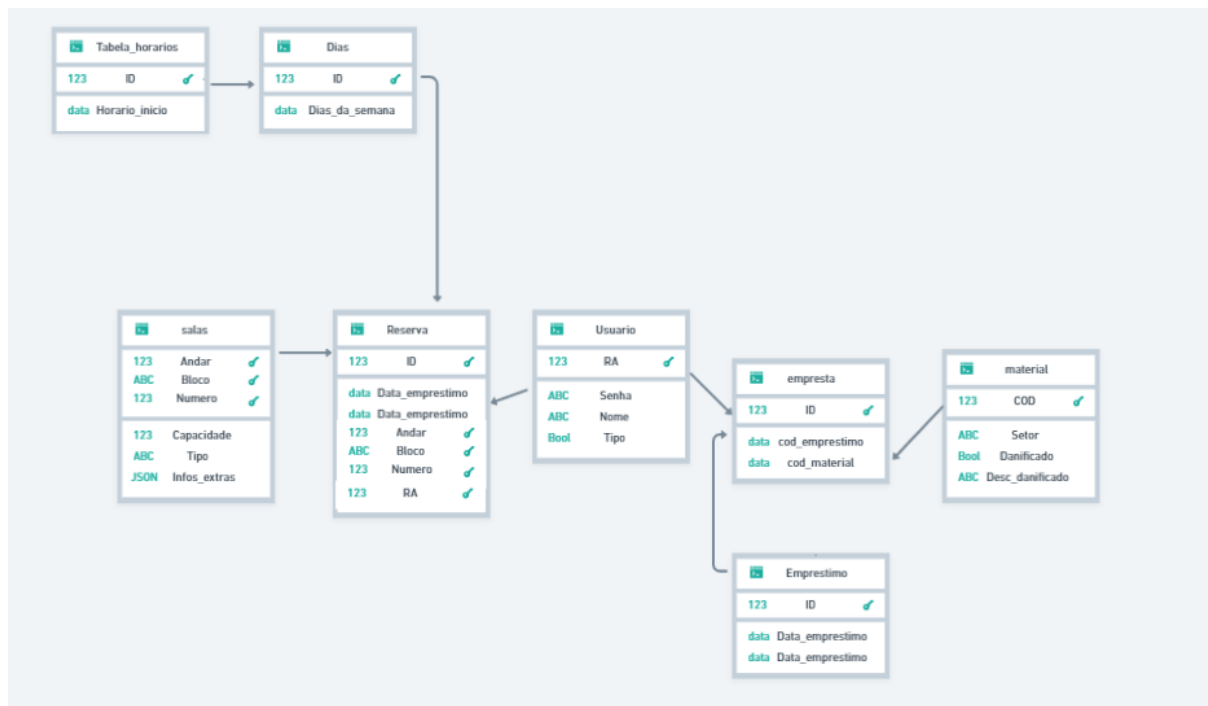


Figura 2 - Diagrama UML desenvolvido para representação do relacionamento entre as entidades

4. Script de criação de tabelas no PostgreSQL com as devidas restrições

```

CREATE TABLE Usuario(
    Nome VARCHAR(50) NOT NULL,
    RA DECIMAL(8) NOT NULL,
    senha varchar(20) Not NULL,
    Tipo boolean
);
  
```

```

CREATE TABLE Salas(
    Andar integer NOT NULL,
    numero DECIMAL NOT NULL,
    Bloco VARCHAR(1) NOT NULL,
    Capacidade integer,
    Tipo_sala VARCHAR(10),
    Infos_extras json
);
  
```

```
CREATE TABLE Emprestimo(  
    id_emprestimo Decimal(30) PRIMARY KEY,  
    Data_emprestimo DATE,  
    Data_devolucao DATE  
);
```

```
CREATE TABLE Material(  
    Cod DECIMAL(8) PRIMARY KEY,  
    Setor VARCHAR(5),  
    Danificado boolean  
);
```

```
CREATE TABLE Tabela_horario(  
    id_dia_sala varchar(30) PRIMARY KEY NOT NULL,  
    constraint dia foreign key (id_dia_sala) references dia,  
    horario_inicio VARCHAR(3)  
);
```

```
CREATE TABLE Dia(  
    id_dia varchar (30) PRIMARY KEY NOT NULL,  
    dias_da_semana varchar(3) NOT NULL  
);
```

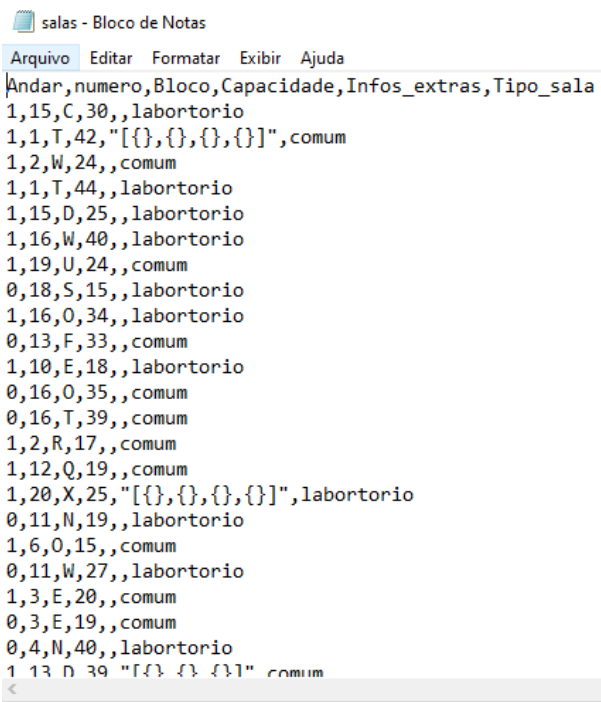
Foram criadas ainda tabelas de relacionamento que facilitam as consultas :

```
create table empresta(  
    ID decimal(8) not null,  
    cod_emprestimo numeric(30),  
    cod_material Decimal(8),  
    constraint RA primary key (ID),  
    constraint id_emprestimo foreign key (cod_emprestimo) references  
emprestimo,  
    constraint Cod foreign key (cod_material) references material  
);
```

```
create table reserva(
  Id_dia numeric(20) not null,
  ra_aluno DECIMAL(8) NOT NULL,
  diaDaSemana varchar(3),
  dataReserva DATE,
  numero_sala integer,
  RASala integer NOT NULL,
  constraint id_dia primary key (Id_dia),
  constraint RA foreign key (ra_aluno) references usuario,
  constraint dias_da_semana foreign key (diaDaSemana) references
dia,
  constraint idSala foreign key (Rasala) references salas
);
```

5. Popule as tabelas com alguns dados (preferência reais, se possível), suficiente para consultas.

Foi utilizado para popular as tabelas o software online mockroo, onde os dados criados se assemelham os reais, que poderiam vir a estar fazendo parte dos dados do sistema. após a geração desses dados, foi utilizado o comando COPY junto ao PostgreSQL para que ele inserisse nas tabelas criadas os dados gerados pela simulação, a figura 3 mostra alguns dos dados gerados pelo software online que foram posteriormente inseridos no sistema



salas - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

Andar,numero,Bloco,Capacidade,Infos_extras,Tipo_sala
 1,15,C,30,,labortorio
 1,1,T,42,"[{},{},{},{ }]",comum
 1,2,W,24,,comum
 1,1,T,44,,labortorio
 1,15,D,25,,labortorio
 1,16,W,40,,labortorio
 1,19,U,24,,comum
 0,18,S,15,,labortorio
 1,16,O,34,,labortorio
 0,13,F,33,,comum
 1,10,E,18,,labortorio
 0,16,O,35,,comum
 0,16,T,39,,comum
 1,2,R,17,,comum
 1,12,Q,19,,comum
 1,20,X,25,"[{},{},{},{ }]",labortorio
 0,11,N,19,,labortorio
 1,6,O,15,,comum
 0,11,W,27,,labortorio
 1,3,E,20,,comum
 0,3,E,19,,comum
 0,4,N,40,,labortorio
 1,13,D,39,"[{},{},{},{ }]",comum

Figura 1 - representação dos dados gerados pelo simulador online que foram inseridos ao sistema

A seguir demonstra-se um exemplo de script utilizado para inserir os dados junto às tabelas criadas:

```
COPY Salas
FROM 'C:\Users\cris\Desktop\Banco de dados 2\salas.csv'
DELIMITER ','
CSV HEADER;

table salas;
```

```
COPY Material
FROM 'C:\Users\cris\Desktop\Banco de dados 2\material.csv'
DELIMITER ','
CSV HEADER;
```

```
table usuario;
COPY usuario
FROM 'C:\Users\cris\Desktop\Banco de dados 2\usuario.csv'
DELIMITER ','
CSV HEADER;
```

```
table emprestimo;
COPY emprestimo
FROM 'C:\Users\cris\Desktop\Banco de dados 2\emprestimo.csv'
DELIMITER ','
CSV HEADER;
```

```
table empresta;
COPY empresta
FROM 'C:\Users\cris\Desktop\Banco de dados 2\empresta.txt'
DELIMITER ','
CSV HEADER;
```

```
COPY dia
FROM 'C:\Users\cris\Desktop\Banco de dados 2\dias_da_semana.csv'
DELIMITER ','
CSV HEADER;
```

```
COPY Tabela_horario
FROM 'C:\Users\cris\Desktop\Banco de dados 2\horarios.csv'
DELIMITER ','
CSV HEADER;
```

6. Criação de índices adequados às consultas.

alguns casos de utilização comum das consultas ao banco e seus respectivos índices criados para facilitar a busca:

1 - criação de índice para consulta de empréstimo de material por determinado usuário

exemplo de consulta:

```
SELECT cod_emprestimo from empresta
where ID = 5299051;
```

a. resultado da consulta

	cod_emprestimo [PK] numeric (30)
1	6706207125800818
2	6706207135800818
3	67062075800818
4	6706207125818
5	6207125800818
6	670620725800818
7	67062071818
8	6706203325800818
9	670620818
10	6706125800818

b. criação do índice:

```
CREATE UNIQUE INDEX IdxemprestaUser ON empresta (Id, cod_emprestimo,
cod_material);
```

c. Consulta após criação do índice, com explain analyse:

	QUERY PLAN
	text
1	Seq Scan on empresta (cost=0.00..1.50 rows=1 width=26) ...
2	[...] Filter: (id = '5299051'::numeric)
3	[...] Rows Removed by Filter: 30
4	Planning Time: 0.149 ms
5	Execution Time: 0.055 ms

2 - criação de índice para consulta de empréstimo de materiais em determinado dia

a. Exemplo de consulta:

```
SELECT id_emprestimo from emprestimo where data_emprestimo = '24/9/2021'
```

b. Resultado da consulta:

	id_emprestimo	
	[PK] numeric (30)	
1	3539865219793684	
2	4405492734971015	
3	5602239275523665	

c. Criação do índice:

```
CREATE UNIQUE INDEX IdxEmprestimo_data ON emprestimo (id_emprestimo);
```

d. Consulta após criação do índice, com explain analyse:

	QUERY PLAN	
	text	
1	Seq Scan on emprestimo (cost=0.00..19.50 rows=2 width=...	
2	[...] Filter: (data_emprestimo = '2021-09-24'::date)	
3	[...] Rows Removed by Filter: 997	
4	Planning Time: 0.133 ms	
5	Execution Time: 0.286 ms	

3 - data de empréstimo de determinado material:

a. exemplo de consulta:

```
select cod_material, data_emprestimo  
from empresta join emprestimo  
on emprestimo.id_emprestimo = empresta.cod_emprestimo  
where cod_material = 636294123;
```


b. Resultado da consulta:

	cod_material	data_emprestimo
	numeric (15)	date
1	636294123	2021-07-08

c. Criação do índice:

```
CREATE UNIQUE INDEX Idxdata ON empresta (id,cod_emprestimo,  
cod_material);
```

d. Consulta após criação do índice, com explain analyse:

	QUERY PLAN
	text
1	Nested Loop (cost=0.28..9.81 rows=1 width=22) (actual time=0.042..0.051 rows=1 loops=1)
2	[...] -> Seq Scan on empresta (cost=0.00..1.50 rows=1 width=44) (actual time=0.025..0.033 rows=1 loops=1)
3	[...] Filter: (cod_material = '636294123'::numeric)
4	[...] Rows Removed by Filter: 39
5	[...] -> Index Scan using idxemprestimo_data on emprestimo (cost=0.28..8.29 rows=1 width=15) (actual time=0.014..0.014 rows=1 loops=1)
6	[...] Index Cond: (id_emprestimo = empresta.cod_emprestimo)
7	Planning Time: 2.063 ms
8	Execution Time: 0.105 ms

7. Criação de funções para consultas corriqueiras ou tarefas nas tabelas.

1 - criação de função de consulta de empréstimo por determinado usuário

a. Código da função

```
CREATE OR REPLACE FUNCTION showEmprestimoUser(RA1 decimal(8))  
    returns table (nome varchar(100), RA Decimal(8), id_emprestimo  
Decimal(30),  
    Data_emprestimo DATE, codMaterial Decimal(30))  
as $$  
BEGIN  
    RETURN QUERY SELECT c.nome, c.RA,  
p.cod_emprestimo, v.Data_emprestimo, a.cod  
FROM usuario as c  
JOIN empresta as p ON p.id = c.RA  
JOIN emprestimo as v ON v.id_emprestimo = p.cod_emprestimo  
JOIN material as a ON  
a.cod=p.cod_material  
WHERE c.Ra = RA1;
```

```
END;  
$$ LANGUAGE plpgsql;
```

b. utilizando da função

```
select * from showEmprestimoUser('5299051');
```

c. retorno da função:

Explain

Data Output

Messages

Notifications

	nome character varying	ra numeric	id_emprestimo numeric	data_emprestimo date	codmaterial numeric
1	Kasper Piken	5299051	6706207125800818	2021-03-27	620110148

2 - função de listar usuários que emprestaram determinado material

a. Código da função:

```
CREATE OR REPLACE FUNCTION showMaterialUser(NroMaterial decimal(8))  
    returns table (nome varchar(100), RA Decimal(8), Data_emprestimo  
DATE,  
                    id_emprestimo Decimal(30))  
    as $$  
BEGIN  
    RETURN QUERY SELECT c.nome, c.RA,  
v.data_emprestimo, v.id_emprestimo  
    FROM material as a  
    JOIN empresta as p ON p.cod_material = a.cod  
    JOIN usuario c ON c.Ra = p.ID  
    JOIN emprestimo as v ON v.id_emprestimo = p.cod_emprestimo  
    WHERE a.cod = NroMaterial;  
END;  
$$ LANGUAGE plpgsql;
```

b. Utilizando da função:

```
select * from showMaterialUser('118221683');
```

c. retorno da função:

	nome character varying	ra numeric	data_emprestimo date	id_emprestimo numeric
1	Rosanna Michele	9209221	2021-07-11	5767338735089601555

3 - função para listar materiais emprestados por setor em determinada data

a. Código da função:

```
CREATE OR REPLACE FUNCTION MaterialSetor(SetorN varchar(15), data date)
    returns table (setor1 varchar(100), material decimal(100),data1
date )
    as $$
BEGIN
    RETURN QUERY SELECT a.setor,a.cod, v.data_emprestimo
    FROM material as a
    JOIN emprestimo as v ON v.data_emprestimo = data
    WHERE a.Setor = SetorN;
END;
$$ LANGUAGE plpgsql;
```

b. Utilizando da função:

```
select * from MaterialSetor('Support', '2021-09-05');
```

c. Retorno da Função:

	setor1 character varying	material numeric	data1 date
1	Support	69190258	2021-09-05
2	Support	69190258	2021-09-05
3	Support	69190258	2021-09-05
4	Support	69190258	2021-09-05
5	Support	69190258	2021-09-05
6	Support	932275	2021-09-05
7	Support	932275	2021-09-05
8	Support	932275	2021-09-05
9	Support	932275	2021-09-05
10	Support	932275	2021-09-05
11	Support	66184450	2021-09-05
12	Support	66184450	2021-09-05
13	Support	66184450	2021-09-05
14	Support	66184450	2021-09-05
15	Support	66184450	2021-09-05
16	Support	50458315	2021-09-05

8. Criação de visões

Como já citado a aplicação terá dois tipos de usuário, administrador e usuário comum, pela estrutura do banco a definição do tipo é dado pelo dado guardado dentro da tabela usuários, sendo 1 usuário do tipo administrador, e 0 usuário do tipo comum, as diferentes visões serão dadas a partir desse dado guardado.

Primeiramente faremos a criação da visão do tipo Administrador

a. Código de criação da visão do tipo usuario Administrador:

```
create view usuarios_administrador as
select nome, Ra, tipo
from usuario where tipo = true
order by nome desc;
```

b. Exemplo de uso da visão:

```
select * from usuarios_administrador
```

c. Exemplo de retorno da função:

	nome character varying (50) 🔒	ra numeric (8) 🔒	tipo boolean 🔒
1	Zahara MacCaughen	7775358	true
2	Yorgos Allicock	9820962	true
3	Yelena Beevers	5865968	true
4	Yancey Bristow	9370888	true
5	Xylina Violet	7087194	true
6	Xenia Backson	4495408	true
7	Wynn Theze	5609951	true
8	Winny Doswell	6185314	true
9	Willow Adicot	6721461	true
10	Wilie Kielt	7802585	true
11	Whitby Saphir	4686893	true
12	Westley Rubinov	8771526	true
13	Werner Peerless	3345331	true
14	Weber Tuff	2508822	true
15	Waylan Revill	3279449	true
16	Werner Titchener	8127000	true

Por fim, faremos a criação da visão do tipo usuário comum

a. Código de criação da visão do tipo usuario usuario comum:

```
create view usuarios_comum as
select nome, Ra, tipo
from usuario where tipo = false
order by nome asc;
```

b. Exemplo de uso da visão:

```
select * from usuarios_comum
```

c. Exemplo de retorno da função:

	Explain	Data Output	Messages	Notifications
		nome character varying (50)	ra numeric (8)	tipo boolean
1		Abbie Stearley	8559112	false
2		Abbye Tresvina	4122607	false
3		Abeu Elcoat	9746917	false
4		Ada Vanyakin	8534352	false
5		Addi Walton	4564355	false
6		Addy Sanderson	3255826	false
7		Adelheid Foad	4975536	false
8		Adelina Caddies	3236346	false
9		Ahmad McGilroy	8790579	false
10		Alaster Troni	7169457	false
11		Aldo Ledley	2387079	false
12		Alejoa Giacobbo	8851957	false
13		Alfi Sadgrove	7873429	false
14		Alaernon McGaffev	5591819	false

9 - Criação de Triggers

1. Criação de gatilho que verifica se o nome do usuário possui caracteres especiais antes da inserção ou atualização do campo para se assegurar que os nomes contenham apenas letras:

- a. Código de criação da trigger, ativada antes de cada inserção ou update na tabela usuario

```
CREATE OR REPLACE FUNCTION verificaNome()
RETURNS trigger AS $nome$
BEGIN
    IF
        NEW.nome ~ '[a-zA-Zàáâãäåąčćęēéëèìíîïłńóôõöøùúûüųÿýźżñ
çĉšžÀÁÂÃÄÅĄĆČĖĘÉÊËÌÍÎÏŁŃÓÔÕÖØÙÚÛÜŲŪỲÝŽŻÑẀṘÆČŠŽðø'-'-]+$'
    THEN
        RETURN NEW;
    END IF;
    RAISE EXCEPTION 'O nome não pode possuir caracteres especiais ';
END;
```

```
$nome$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER verificaUsuario
```

```
BEFORE INSERT OR UPDATE ON usuario
```

```
FOR EACH ROW EXECUTE PROCEDURE verificaNome();
```

- ### b. Exemplo de atualização

```
update usuario
set nome = 'CRÎSTÎANØ KØXNE'
where ra='4411261';
```

- ### c. Mensagem de retorno

Explain	Data Output	Messages	Notifications
		ERROR: 0 nome não pode possuir caracteres especiais CONTEXT: função PL/pgSQL verificanome() linha 9 em RAISE SQL state: P0001	

2. Criação de gatilho que verifica se o material de desejo pode ser emprestado, ou se ele já está emprestado

- a. Código de criação da trigger, ativada antes de cada inserção ou update na tabela emprestimo

```
CREATE OR REPLACE FUNCTION verificaMaterialEmprestado()  
RETURNS trigger AS $verifEmpres$  
    DECLARE  
    emprestimo1 emprestimo%ROWTYPE;  
BEGIN  
  
    SELECT * FROM emprestimo WHERE NEW.data_emprestimo =  
emprestimo.data emprestimo INTO
```

```

emprestimo1;

        IF(emprestimo1.data_emprestimo = new.data_emprestimo)
            THEN
                RAISE EXCEPTION 'Esta material não pode ser escolhida, pois
já esta emprestado!';
            end If;
        END;
$verifEmpres$ LANGUAGE plpgsql;

CREATE TRIGGER verifica_empresta
BEFORE INSERT OR UPDATE ON emprestimo
FOR EACH ROW EXECUTE PROCEDURE verificaMaterialEmprestado();

```

b. Exemplo de atualização

```

update emprestimo
set id_emprestimo = '3554003730600242'
where data_emprestimo ='2021-07-13';

```

c. Mensagem de retorno

```

ERROR:  Esta material não pode ser escolhido, pois já esta emprestado!
CONTEXT:  função PL/pgSQL verificamaterialemprestado() linha 12 em RAISE
SQL state: P0001

```

Em anexo os comandos criados em pg admin4 e desenvolvidos para presente atividade prática supervisionada