Trabalho Prático – Grau G2

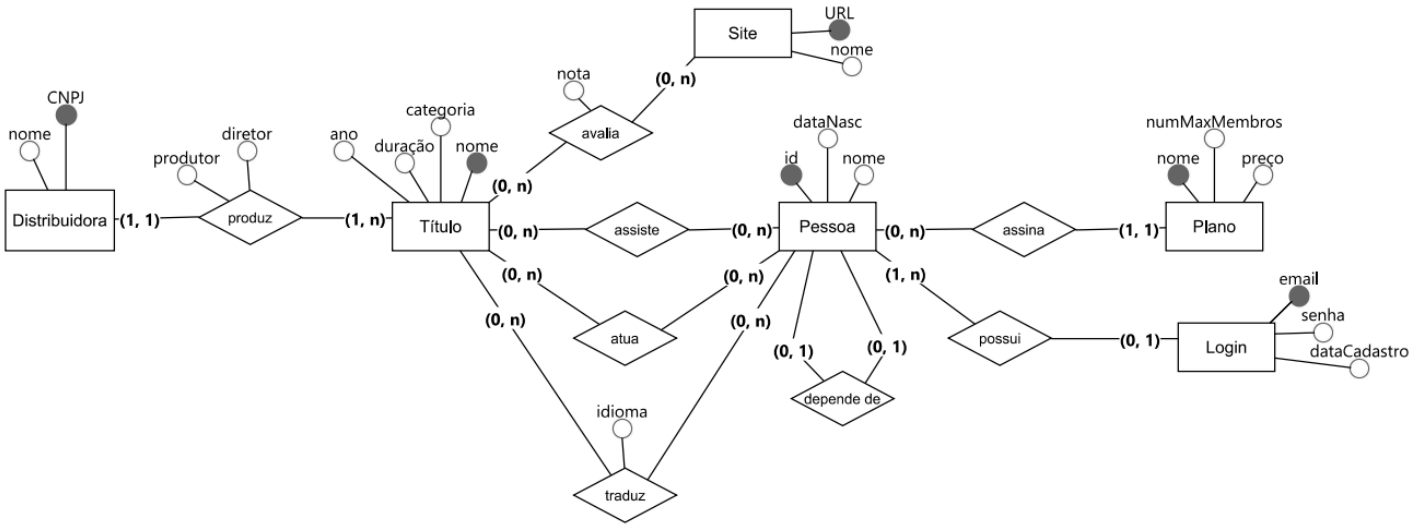
Integrantes:
- Gabriela Soares
- Louis Pottier
- Luísa Silveira
- Miguel Carneiro
- Vinícius Lucena

Questão 1

Um serviço de streaming é projetado para oferecer uma experiência de entretenimento personalizada para seus assinantes. O objetivo é disponibilizar uma vasta biblioteca de títulos, incluindo filmes, séries, documentários e outros tipos de conteúdo audiovisual, garantindo que os usuários tenham acesso a uma diversidade de opções de alta qualidade.
Os títulos disponíveis na plataforma são classificados por nome, ano de lançamento, duração, categoria e distribuidora. Para ter acesso a esses títulos, é necessário realizar um cadastro na plataforma e assinar algum dos planos existentes. Porém, para gerar uma maior flexibilidade aos clientes, o streaming permite que mais de uma pessoa assista aos títulos por assinatura, dependendo do plano escolhido. Assim, toda pessoa registrada na plataforma possui um número de identificação (id), nome, data de nascimento, além de poder ter, também, outra pessoa associada a sua conta (dependente), informação que deve ser mantida no banco de dados do streaming. Além disso, são armazenados os dados de login do cliente, sendo eles e-mail, senha e data de cadastro e, sabe-se que os dados de login de um dependente são os mesmos de seu titular. Dessa forma, um título pode ser assistido por múltiplas pessoas e cada pessoa pode assistir quantos títulos desejar.
Para ajudar os usuários a escolherem o que assistir, os títulos possuem avaliações de diversos sites de crítica, que atribuem notas variando de 0 a 10. Cada site pode ter apenas uma nota para cada filme e informações como o nome e a URL do site, o filme avaliado e sua nota são guardados no sistema para estarem disponíveis à visualização dos usuários.
A distribuidora de um título é responsável por coordenar todos os aspectos da produção, desde o financiamento até a contratação de talentos e a supervisão das filmagens e pós-produção. Assim, cada uma dessas empresas, que possui um nome e um CNPJ registrados no sistema, pode produzir vários títulos, mas cada título é produzido por uma única empresa. Na produção dos títulos, além da distribuidora responsável, participam também um produtor e um diretor.
Cada título possui uma enorme equipe de atuação e tradução, desempenhando um papel crucial na experiência com o público. Uma pessoa pode ser escalada para atuar em vários títulos, dando a vida a diferentes personagens. Os títulos também possuem registradas as pessoas responsáveis por traduzir a experiência do áudio original para outros idiomas, permitindo que um público global aprecie o conteúdo. Assim, tanto os atores quanto os dubladores são pessoas registradas na plataforma, além de também poderem assinar algum plano do streaming para assistir a diferentes títulos.

Com isso, ao registrar informações detalhadas sobre títulos, avaliações, produções, planos e pessoas, a plataforma de streaming proporciona um ecossistema de entretenimento digital voltado para a experiência do usuário.

Questão 2



Questão 3

Restrições de integridade semânticas (regras de negócio):

1. O plano tem um limite de 4 membros.
2. A nota da avaliação no site precisa estar entre 0 e 10.
3. O preço do plano precisa ser maior que zero, não pode ser de graça.
4. O email de um titular precisa ser único.

Questão 4

STREAMING

pessoa (nome, id, dataNasc, plano, email, dataCadastro, senha, titular);

planos (nome, preço, numMaxMembros);

titulos (nome, ano, duração, categoria, distribuidora, produtor, diretor);

site (nome, url);

distribuidora (empresa, cnpj);

assiste (nomeTitulo , pessoa);

avalia (site, nomeTitulo, nota);

atua (id, nomeTitulo);

traduz (id, nomeTitulo, idioma);

As chaves primárias (PK) de cada tabela estão sublinhadas.

Chaves estrangeiras:

1. Na tabela titular, a chave estrangeira “planos” faz referência ao atributo “nome” da tabela planos.

2. Na tabela titulos, o atributo “distribuidora” faz referência ao atributo “cnpj” da tabela distribuidora.

3. Na tabela assiste, o atributo “nomeTitulo” faz referência ao atributo “nome” da tabela titulos e o atributo “pessoa” faz referência ao atributo “id” da tabela pessoa.

4. Na tabela avalia, o atributo “nomeTitulo” faz referência ao atributo “nome” da tabela titulo e o atributo “site” faz referência ao atributo “urlj” da tabela site.

5. Na tabela atua, o atributo “nomeTitulo” faz referência ao atributo “nome” da tabela titulo e o atributo “id” faz referência ao atributo “id” da tabela pessoa.

6. Na tabela traduz, o atributo “nomeTitulo” faz referência ao atributo “nome” da tabela titulo e o atributo “id” faz referência ao atributo “id” da tabela pessoa.

Questão 5

1. **Tabela titular**

```
create table titular
(
 nome varchar(255) not null,
 id varchar(20),
 datanasc date not null,
 plano varchar(255) not null,
 email varchar(256) not null,
 datacadastro date not null,
 senha varchar(20) not null,

 constraint pk_titular primary key(id)
)

alter table titular
add constraint fk_titular_plano foreign key(plano) references planos(nome)
```

```
alter table titular
add constraint uq_email_titular unique (email)
```

Explicação: Para a escolha do tipo do nome, escolhemos varchar(255), pois o nome possui comprimento variável, podendo ocupar strings de até 255 caracteres - não imaginamos que um nome passará disso -, mas se a string armazenada tiver menos caracteres, apenas o espaço necessário será usado. Para o id usamos o varchar(20), pois possui comprimento variável, e consideramos que o id não passará de 20 caracteres. Para as datas, usamos o tipo data, mais especializado para a formatação das datas. Para o email, usamos varchar(256), pois, como dito anteriormente, possui comprimento variável e, de acordo com a RFC 5321 (seção 4.5.3), o máximo de caracteres que um email pode ter é 256. Por fim, escolhemos um tamanho limite de 20 caracteres para a senha. Além dos tipos, definimos o atributo “id” como chave primária, definimos a chave estrangeira “plano” para se relacionar com o atributo “nome” da tabela planos e adicionamos a restrição de que um email precisa ser único para cada titular.

2. **Tabela dependentes**

```
create table dependentes
(
 nome varchar(255) not null,
 datanasc date not null,
 id char(11),
 id_titular char(11) not null,

 constraint pk_dependente primary key(id),
 constraint fk_dependente_titular foreign key(id_titular) references titular(id)

)
```

Explicação: O atributo nome, definido como varchar(255), armazena o nome do dependente, permitindo até 255 caracteres para acomodar nomes de comprimentos variados. O atributo datanasc utiliza o tipo date, especializado para o armazenamento e manipulação de datas, garantindo a correta formatação e facilitando operações como cálculos de idade. O campo id é definido como varchar(20), adequado para armazenar o id do dependente, que é um identificador variável de até 20 caracteres de números. Ele é a chave primária, garantindo que cada dependente seja identificado de forma única na tabela. O atributo id_titular, também definido como varchar(20), armazena o id do titular ao qual o dependente está vinculado. A fim de garantir a integridade referencial, uma restrição de chave estrangeira (fk_dependente_titular) é estabelecida, vinculando id_titular ao campo id na tabela titular, assegurando que cada dependente esteja associado a um titular existente.

3. Tabela assiste

```
create table assiste
(
 nomeTitulo varchar(255),
 titular varchar(20),

 constraint pk_assite primary key(nomeTitulo, titular),
 constraint fk_assiste_titulo foreign key(nomeTitulo) references titulos(nome),
 constraint fk_assiste_titular foreign key(titular) references titular(id)

)
```

Explicação: O campo titular apresenta registros de id, e, como já explicado, apresentam tipo varchar(20). O campo nomeTitulo, apresenta o nome do título do filme, que foi imposto um limite de 255 caracteres com o tipo varchar para strings de comprimento variável. A chave primária composta (pk_assite) possui os campos nomeTitulo e titular. A restrição fk_assiste_titulo relaciona o atributo nomeTitulo ao campo nome da tabela titulos. A restrição fk_assiste_titular relaciona o atributo titular ao campo id na tabela titular.

4. Tabela distribuidora

```
create table distribuidora
(
 empresa varchar(255),
 cnpj char(14),
 constraint pk_distribuidora primary key(cnpj)

)
```

O campo empresa é definido com tipo varchar(255), permitindo o armazenamento do nome da empresa com até 255 caracteres, podendo variar o comprimento. O campo cnpj é definido como char(14), adequado para armazenar o CNPJ, que é um identificador fixo de 14 caracteres numéricos no Brasil.O atributo cnpj é definido como chave primária por meio da restrição pk_distribuidora. Isso assegura que cada CNPJ registrado na tabela seja único e válido, evitando duplicidades.

5. **Tabela títulos**

```
create table títulos
(
 nome varchar(255),
 ano smallint not null,
 duracao time not null,
 categoria varchar(50) not null,
 distribuidora char(14) not null,
 produtor varchar(255),
 diretor varchar(255),

 constraint pk_titulo primary key(nome),
 constraint fk_titulo_distribuidora foreign key(distribuidora) references distribuidora(cnpj)

)
```

Explicação: O campo nome é definido como varchar(255), permitindo armazenar o nome do título com até 255 caracteres, e é também a chave primária da tabela (pk_titulo). O campo ano é definido como smallint e não permite valores nulos (not null), apropriado para armazenar anos, pois o intervalo do tipo smallint é suficiente para cobrir os anos de interesse para títulos. O campo duracao usa o tipo time e é também not null, garantindo o armazenamento preciso da duração do título no formato de tempo. O campo categoria é definido como varchar(50) e também é not null, permitindo o armazenamento de categorias com até 50 caracteres, o que é geralmente suficiente para descrever a maioria das categorias. O campo distribuidora é definido como char(14) e é not null, armazenando o CNPJ da empresa distribuidora, que é um identificador fixo de 14 caracteres. Uma chave estrangeira (fk_titulo_distribuidora) é definida, referenciando o campo cnpj na tabela distribuidora, garantindo que cada título esteja associado a uma distribuidora registrada. Os campos produtor e diretor são ambos definidos como varchar(255), permitindo o armazenamento de nomes de produtores e diretores com até 255 caracteres, com flexibilidade para nomes de diferentes tamanhos.

6. **Tabela planos**

```
create table planos
(
 nome varchar(255),
 preco decimal(5,2) not null,
 numMaxMembros int not null,

 constraint ck_membros check (numMaxMembros >= 1 and numMaxMembros<=4),
 constraint ck_preco check (preco >0)

)
```

```
alter table planos
add constraint pk_nome primary key(nome)
```

Explicação: O “nome” do plano é do tipo varchar e pode apresentar até 255 caracteres. Após a criação da tabela, há a alteração dela, colocando o nome como chave primária. O “preco” é definido com tipo decimal (5,2), podendo ir de -999,99 até 999,99. Porém é adicionado uma restrição, na qual os valores da coluna “preco” precisa ser maior que zero, ou seja os valores podem ir de 0 até 999,99. O tipo do “numMaxMembros” é int, ou seja, um número inteiro, e é feita uma restrição na qual o número de membros tem que ser de 1 a 4.

7. **Tabela site**

```
create table site
(
 nome varchar(2083) not null,
 cnpj char(14),

 constraint pk_site primary key(cnpj)
 )
```

Explicação: o nome do site é do tipo varchar(2083), já que é uma string de tamanho variável e o máximo de caracteres que um site pode possuir no internet explore são 2083 caracteres. Então, usamos esse limite, já que achamos que nenhum site vai passar deste mesmo. Como já dito, o cnpj apresenta 14 números, apresentando valor fixo e, consequentemente, o tipo char(14). O cnpj do site foi escolhido como chave primária da tabela.

8. **Tabela profissional**

```
create table profissional
(
 nome varchar(255) not null,
 id varchar(20),

 constraint pk_profissional primary key(id)

)
```

Explicação: O nome de um profissional apresenta tamanho variado, então colocamos o tipo do campo nome como varchar(255). O id, como já dito antes também,pode ser variável, colocando o tipo do campo id como varchar(20). Além disso, o campo id foi escolhido como chave primária.

9. **Tabela avalia**

```
create table avalia
(
```

```
 nomeTitulo varchar(255),
 site char(14),
 nota numeric(3,1),

 constraint pk_avalia primary key(nomeTitulo, site, nota),
 constraint fk_avalia_titulo foreign key(nomeTitulo) references titulos(nome),
 constraint fk_avalia_site foreign key(site) references site(cnpj),
 constraint ck_nota check (nota>=0 and nota<=10)

)
```

Explicação: O título de um filme apresenta tamanho variado. Dessa forma, como explicado anteriormente, escolhemos varchar(255). O campo site é o cnpj do site que faz a avaliação, dessa forma, escolhemos o tipo char(14), já que o cnpj apresenta 14 números. Já para o campo “nota”, escolhemos o tipo numeric(3,1). Dessa forma, os valores dessa coluna poderiam ir de -99.9 a 99.9. Adicionamos uma restrição “ck_nota” no qual a nota presa estar entre 0 e 10. A chave primária é composta pelos campos “nomeTitulo”, “site” e “nota”. A fk_avalia_titulo estabelece uma relação com a tabela titulos, garantindo que cada título avaliado exista na tabela titulos. Já a fk_avalia_site estabelece uma relação com a tabela site, garantindo que cada CNPJ de site referenciado exista na tabela site.

10. Tabela atua

```
create table atua
(
 nomeTitulo varchar(255),
 id varchar(20),

 constraint pk_atua primary key(nomeTitulo, id),
 constraint fk_atua_titulo foreign key(nomeTitulo) references titulos(nome),
 constraint fk_atua_profissional foreign key(id) references profissional(id)

)
```

Explicação: O campo nomeTitulo, definido como varchar(255), armazena o nome do título no qual um profissional atua. O campo id, definido como varchar(20), armazena o id do profissional que atua no título. Os dois campos fazem parte da chave primária. A restrição fk_atua_titulo vincula o campo nomeTitulo ao campo nome na tabela titulos. A restrição fk_atua_profissional vincula o campo id ao id na tabela profissional.

11. Tabela traduz

```
create table traduz
(
 nomeTitulo varchar(255),
```

```
id varchar(20),
idioma varchar(50) not null,

constraint pk_traduz primary key(nomeTitulo, id),
constraint fk_traduz_titulo foreign key(nomeTitulo) references titulos(nome),
constraint fk_traduz_profissional foreign key(id) references profissional(id)

)
```

Explicação: O campo nomeTitulo, definido como varchar(255), armazena o nome do título que foi traduzido. O campo id, definido como varchar(20), armazena o id do profissional que realiza a tradução. Ambos os campos fazem parte da chave primária. O campo idioma, definido como varchar(50), armazena o idioma para o qual o título está sendo traduzido.Ele é not null, garantindo que sempre tenha um idioma especificado para cada tradução. A restrição fk_traduz_titulo vincula o campo nomeTitulo ao campo nome na tabela titulos. Já a restrição fk_traduz_profissional vincula o campo id ao id na tabela profissional.

Questão 6

- **Tabela planos**

```
INSERT INTO planos(nome, preco, numMaxMembros)
VALUES (‘Básico’, 11.90, 1);

INSERT INTO planos(nome, preco, numMaxMembros)
VALUES ('Padrão com anúncios', 18.90, 2);

INSERT INTO planos(nome, preco, numMaxMembros)
VALUES ('Padrão', 39.90, 2);

INSERT INTO planos(nome, preco, numMaxMembros)
VALUES ('Premium', 55.90, 4);
```

- **Tabela distribuidora**

```
INSERT INTO distribuidora(empresa, cnpj)
VALUES ('Sony pictures', '33040767000101');

INSERT INTO distribuidora(empresa, cnpj)
VALUES ('Warner Bros Pictures', '06136283000180');

INSERT INTO distribuidora(empresa, cnpj)
VALUES ('Metro-Goldwyn-Mayer', '09595976000129');

INSERT INTO distribuidora(empresa, cnpj)
VALUES ('A24', '15909001000120’);
```

INSERT INTO distribuidora(empresa, cnpj)
VALUES ('Paramount Pictures', '27654722000116’);

INSERT INTO distribuidora(empresa, cnpj)
VALUES ('Universal Pictures', '04133240000204’);

INSERT INTO distribuidora(empresa, cnpj)
VALUES ('DOWNTOWN FILMES', '07616202000101’);

INSERT INTO distribuidora(empresa, cnpj)
VALUES (Fox film', '33110420000937’);

INSERT INTO distribuidora(empresa, cnpj)
VALUES ('Walt Disney Pictures ', '73042962000187’);

INSERT INTO distribuidora(empresa, cnpj)
VALUES ('Paris Filmes', '12580503000162’);

- **Tabela títulos**

INSERT INTO titulos(nome, ano, duracao, categoria, distribuidora, produtor, diretor)
VALUES ('Godzilla e Kong: O Novo Império', 2024, ‘01:55:00’, 'Ação/Ficção científica', ‘06136283000180’, ‘Mary Parent’ , ‘Adam Wingard’);

INSERT INTO titulos(nome, ano, duracao, categoria, distribuidora, produtor, diretor)
VALUES ('Duna', 2021, '02:35:00', 'Ficção científica/Aventura', '06136283000180', 'Mary Parent ', 'Denis Villeneuve');

INSERT INTO titulos(nome, ano, duracao, categoria, distribuidora, produtor, diretor)
VALUES ('Interestellar', 2014, '02:49:00', 'Ficção científica/Aventura', '06136283000180', 'Christopher Nolan', 'Christopher Nolan');

INSERT INTO titulos(nome, ano, duracao, categoria, distribuidora, produtor, diretor)
VALUES ('Oppenheimer', 2023, '03:01:00', 'Drama/Obra de Época', '04133240000204', 'Emma Thomas', 'Christopher Nolan');

- **Tabela titular:**

INSERT INTO titular(id, nome, datanasc, plano, email, datacadastro, senha) **VALUES** ('15104411311','SINVALDO SILVA DA GAMA','1989-08-27','Padrão com anúncios','fviegas@msn.com','2019-08-13','kupalisko');

INSERT INTO titular(id, nome, datanasc, plano, email, datacadastro, senha) **VALUES** ('15747008200','SOLANGE BESSA CAVALCANTI','1990-09-17','Padrão com anúncios','luebke@gmail.com','2015-06-02','dodooo');

INSERT INTO titular(id, nome, datanasc, plano, email, datacadastro, senha) **VALUES** ('07503329300','SOLANGE MARA DUARTE GAIA','2006-03-30','Padrão','sonnen@verizon.net','2015-08-10','nLf1HtRMD3');

…

- **Tabela profissional:**

INSERT INTO profissional(nome, id)
VALUES ('Tom Holland', '12345678901');

INSERT INTO profissional(nome, id)
VALUES ('Timothée Chalamet', '23456789012');

INSERT INTO profissional(nome, id)
VALUES ('Daniel Craig', '34567890123');
…

- **Tabela atua:**

INSERT INTO atua(nometitulo, id)
VALUES ('Uncharted', '12345678901');

INSERT INTO atua(nometitulo, id)
VALUES ('Tudo em Todo Lugar ao Mesmo Tempo', '45678901234');

INSERT INTO atua(nometitulo, id)
VALUES ('Top Gun: Maverick', '56789012345');

INSERT INTO atua(nometitulo, id)
VALUES ('Jurassic World: Domínio', '67890123456');
…

- **Tabela assiste:**

INSERT INTO assiste(nometitulo ,titular)
VALUES('A Bela e a Fera', '66393663391');

INSERT INTO assiste(nometitulo ,titular)
VALUES('Jojo Rabbit', '11483243214');

INSERT INTO assiste(nometitulo ,titular)
VALUES('Soul', '01376150514');
…

- **Tabela traduz:**

INSERT INTO traduz(nometitulo, id, idioma)
VALUES ('Stranger Things', '17296090073', 'pt-br');

INSERT INTO traduz(nometitulo, id, idioma)
VALUES ('Toy Story', '98141480081', 'pt-br');

INSERT INTO traduz(nometitulo, id, idioma)
VALUES ('Peaky Blinders', '54059598046', 'pt-br');
...

- **Tabela avalia:**

INSERT INTO avalia(nometitulo, site, nota)
VALUES ('Interstellar', '82662200000174', 8.6);

INSERT INTO avalia(nometitulo, site, nota)
VALUES ('Peaky Blinders', '82662200000174', 8.8);

INSERT INTO avalia(nometitulo, site, nota)
VALUES ('Dark', '33333333000100', 4.5);
...

Questão 7: Consultas em Álgebra Relacional

1. Qual o nome dos dependentes que estão vinculados a uma conta que assiste filmes da categoria ‘drama’?
Tabelas Chamadas: Títulos / Assiste / Dependente

$R1 \leftarrow \sigma_{(categoria = \text{'drama'})} (Títulos)$

$R2 \leftarrow Assiste \underset{(nomeTítulo = nome)}{\bowtie} R1$

$\delta_{nome \rightarrow nomeDependente} (Dependentes)$

$R3 \leftarrow Dependentes \underset{(id\_titular = titular)}{\bowtie} R2$

$Resposta \leftarrow \Pi_{(nomeDependente)} (R3)$

2. Quais os nomes dos titulares e os emails associados que assinaram o plano mais caro do Streaming no ano de 2023?
Tabelas chamadas: Titular / Plano

$R1(maiorPreço) \leftarrow \mathcal{G}_{max(preço)} (Planos)$

$R2 \leftarrow R1 \underset{(maiorPreço = preço)}{\bowtie} Planos$

$\delta_{\text{nome} \rightarrow \text{nomePlano}}(R2)$

$R3 \leftarrow R2 \underset{(\text{nomePlano} = \text{plano})}{\bowtie} \text{Titular}$

$R4 \leftarrow \sigma_{(\text{dataCadastro.ano} = \text{'2023'})}(R3)$

$\text{Resposta} \leftarrow \Pi_{(\text{nome, email})}(R4)$

3. Quais titulares (Nome e id) assistem todos os títulos com dublagem em português?

Tabelas chamadas: Traduz / Assiste / Titular

$\text{DublagemPortuguês} \leftarrow \Pi_{\text{nomeTítulo}}(\sigma_{(\text{idioma} = \text{'português'})}(\text{Traduz}))$

$\text{TitularesAssistem} \leftarrow \Pi_{(\text{nomeTítulo, titular})}(\text{Assiste})$

$R1 \leftarrow \text{TitularesAssistem} \div \text{DublagemPortuguês}$

$R2 \leftarrow R1 \underset{(\text{titular} = \text{id})}{\bowtie} \text{Titular}$

$\text{Resposta} \leftarrow \Pi_{(\text{id, nome})}(R2)$

4. Quais os nomes dos profissionais que atuaram apenas nos filmes da distribuidora "Sony"?

Tabelas chamadas: Distribuidora / Atua / Título / Profissional

$\text{Sony} \leftarrow \sigma_{(\text{empresa} = \text{'Sony'})}(\text{Distribuidora})$

$\delta_{\text{nome} \rightarrow \text{nomeTítulo}}(\text{Título})$

$\text{TitulosSony} \leftarrow \text{Sony} \underset{(\text{CNPJ} = \text{distribuidora})}{\bowtie} \text{Título}$

$\text{Outras} \leftarrow \sigma_{(\text{empresa} \neq \text{'Sony'})}(\text{Distribuidora})$

$\text{TitulosOutras} \leftarrow \text{Outras} \underset{(\text{CNPJ} = \text{distribuidora})}{\bowtie} \text{Titulo}$

$\text{AtoresSony} \leftarrow \text{TitulosSony} \bowtie \text{Atua}$

$\text{AtoresOutras} \leftarrow \text{TitulosOutras} \bowtie \text{Atua}$

$\text{ApenasSony} \leftarrow \text{AtoresSony} - \text{AtoresOutras}$

$\text{Aux} \leftarrow \text{ApenasSony} \bowtie \text{Profissional}$

$\text{Resposta} \leftarrow \Pi_{(\text{Nome})}(\text{Aux})$

5. Listar o nome de todos os títulos presentes no banco de dados. Para aqueles títulos que possuírem avaliação maior que 6, exibir ao lado de seu nome, sua nota e o nome do site que fez sua avaliação.

Tabelas chamadas: Site / Título / Avalia

$R1 \leftarrow \sigma_{(nota > 6)} (Avalia)$

$R2 \leftarrow R1 \underset{(site = CNPJ)}{\bowtie} Site$

$\delta_{nome \rightarrow nomeTítulo} (Título)$

$R3 \leftarrow Título \rightarrowtail R2$

$Resposta \leftarrow \Pi_{(nomeTítulo, nome, nota)} (R2)$

Questão 8: consultas em SQL

1. Quais os títulos que não têm avaliação nenhuma?

```
SELECT nome
FROM titulos
WHERE nome NOT IN
                    (SELECT nomeTitulo
                    FROM avalia)
```

2. Quais os produtores cujos títulos têm maior duração que todos os títulos produzidos pela Netflix?

```
SELECT produtor
FROM titulos
WHERE duracao > ALL(SELECT duracao
                    FROM titulos
                    WHERE distribuidora IN    (SELECT CNPJ
                                               FROM distribuidora
                                               WHERE nome = 'Netflix'
                                               )
                    )
```

3. Qual o nome dos atores assinados a um plano, cujo e-mail pertence ao Gmail? Ordene por ordem alfabética.

```
SELECT nome
FROM pessoa
WHERE email like '%@gmail%'
AND id IN
          (SELECT id
```

```
                FROM atua)
ORDER BY nome
```

Questão 9

Q1: Foram feitas mudanças na questão 1, de forma a deixar a apresentação da plataforma de Streaming com um contexto maior e um objetivo mais explicado. Antes a estrutura do texto estava baseada mais apenas nas definições dos relacionamentos para o modelo conceitual.

Q2: Foram feitas mudanças no modelo conceitual da questão 2:

- As entidades 'profissional' e 'dependente' deixaram de existir, pois continham os mesmos atributos da entidade 'titular'. Todos foram englobados em uma entidade 'pessoa'.
- A entidade 'pessoa' passou a ter um auto-relacionamento para representar os dependentes daquelas pessoas que assinaram o plano.
- O identificador da entidade 'site' mudou de CNPJ para URL, pois acreditamos que a maioria dos sites não tenha um CNPJ, e sim uma URL para diferenciá-lo dos demais sites.

Q3: Não foram feitas alterações nas restrições semânticas do nosso banco de dados.

Q4: Foram feitas mudanças na questão 4, pois como nosso modelo conceitual mudou, consequentemente nossas tabelas também tiveram alterações.

- Para alterar o nome da coluna CNPJ da tabela site para o nome URL, seguimos os seguintes passos:
    1. Removemos a constraint de fk que ligava a pk da tabela site com a coluna 'site' da tabela avalia e removemos a constraint de pk da tabela site.

```
alter table avalia drop constraint fk_avalia_site
alter table site drop constraint pk_site]
alter table site rename column cnpj to url
```

    2. Alteramos o tipo de dado da coluna url da tabela site e da coluna site da tabela avalia. Antes era um tipo char(14), pois todo cnpj tem 14 caracteres, mas mudamos para um varchar(255) para englobar diferentes urls.

```
alter table site alter column url type varchar(255)
alter table avalia alter column site type varchar(255)
```

    3. Atualizamos todos os dados da coluna url da tabela site e da coluna site da tabela avalia. Antes os dados eram números de cnpj e passaram a ser urls.

```
update avalia
set site = 'https://filmow.com/'
where site = '45269448000143'

update site
set url = 'https://filmow.com/'
where url = '45269448000143'
```

4. Adicionamos de volta as constraints de pk da tabela site e de fk da tabela avalia.

```
alter table site add constraint pk_site primary key(url)
alter table avalia add constraint fk_avalia_site foreign key(site) references site(url)
```

- Para eliminarmos a tabela dependentes e migrar seus dados para a tabela pessoa, sem perder sua referência com seus respectivos titulares, seguimos os seguintes passos:
    1. Alteramos o nome da tabela titular para pessoa.

```
alter table titular rename to pessoa
```

2. Adicionamos uma coluna ‘titular’ na tabela pessoa para referenciar o id dos respectivos titulares dos dependentes que vão ser adicionados à tabela. Aquelas pessoas que já são os titulares, ou seja, não possuem titulares, vão receber ‘null’ no novo atributo titular.

```
alter table pessoa add column titular varchar(20)
```

3. Adicionamos as tuplas dos dependentes na tabela pessoa. Para isso, migramos os dados já existentes na tabela dependentes. Os campos id, nome e datanasc, que já existiam na tabela dependentes foram mantidos para a tabela pessoa. O campo titular da tabela pessoa foi preenchido com o número do id do titular presente no campo id_titular de cada tupla da tabela dependentes. Os campos plano, email, datacadastro e senha, que não existiam na tabela dependentes, foram preenchidos com os dados dos respectivos titulares.

```
insert into pessoa (nome, id, datanasc, plano, email, datacadastro, senha, titular)
select d.nome, d.id, d.datanasc, t.plano, t.email, t.datacadastro, t.senha, d.id_titular
from dependentes d, pessoa t
where d.id_titular = t.id
```

4. Removemos a tabela dependentes do banco de dados.

```
drop table dependentes
```

- Para eliminarmos a tabela profissional e migrarmos seus dados para a tabela pessoa, seguimos os seguintes passos:
    1. Removemos as restrições de not null definidas para as colunas datanasc, plano, email, datacadastro e senha, pois nem todo profissional, que vai passar a fazer parte dessa tabela, possui assinatura na plataforma de streaming.

```
alter table pessoa alter column datanasc drop not null
alter table pessoa alter column plano drop not null
alter table pessoa alter column email drop not null
alter table pessoa alter column datacadastro drop not null
alter table pessoa alter column senha drop not null
```

    2. Migramos os dados dos profissionais que não eram assinantes da plataforma de streaming para a tabela pessoa, pois aqueles que já eram assinantes (além de serem profissionais de algum título) já tinham seu registro na tabela pessoa. Para os dados migrados dos não assinantes, definimos como null os valores das colunas plano, email, datacadastro, senha e titular, pois eles não são assinantes, logo, não possuem esses dados. No caso da coluna datanasc, como os profissionais não possuíam essa coluna em sua tabela anteriormente, não tivemos como migrar esses dados, portanto, também preenchemos a coluna datanasc como null, mas com a intenção de preenchê-la depois.

```
insert into pessoa (nome, id, datanasc, plano, email,
datacadastro, senha, titular)
select p.nome, p.id, null, null, null, null, null, null
from profissional p left join pessoa pe
on p.id = pe.id
where pe.id is null
```

    3. Inserimos a data de nascimento de todos os profissionais que estavam sem.

```
update pessoa
set datanasc = case
when id = '23456789012' then '1995/12/27'::date
when id = '34567890123' then '1968/03/02'::date
when id = '45678901234' then '1962/08/06'::date
.
.
.
when id = '32421124645' then '1960/01/19'::date
end
where id in ('23456789012', '34567890123',... , '32421124645')
```

4. Adicionamos novamente a restrição de not null para a coluna datanasc.

```
alter table pessoa alter column datanasc set not null
```

5. Removemos as constraints de fk da tabela atua e da tabela traduz que referenciavam a tabela profissional para conseguirmos removê-la.

```
alter table atua drop constraint fk_atua_profissional
alter table traduz drop constraint fk_traduz_profissional
```

6. Adicionamos as constraints de fk para a tabela atua e traduz de forma a referenciar a id dos profissionais presentes na tabela pessoa

```
alter table atua add constraint fk_atua_pessoa foreign key(id)
references pessoa(id)
alter table traduz add constraint fk_traduz_pessoa foreign
key(id) references pessoa(id)
```

7. Removemos a tabela profissional

```
drop table profissional
```

Q5 e Q6: Mantivemos as questões 5 e 6 do nosso trabalho da G1, pois foi como criamos e inserimos dados na tabela. Alterações na tabela foram mencionadas acima.

Q7: Mantivemos a questão 7 do nosso trabalho da G1, pois, apesar de as tabelas terem tido alterações, as questões de Álgebra desenvolvidas estavam corretas e de acordo com nossa estrutura antiga das tabelas, e o professor disse que poderíamos manter.

Q8: As consultas em SQL 1 e 2 propostas foram mantidas, pois elas usam tabelas que não foram alteradas, logo, elas funcionam corretamente para a nossa nova estrutura do banco de dados. Porém, alteramos a última consulta, pois ela utilizava a tabela profissional, que foi removida do nosso banco de dados. Portanto, alteramos seus comandos do SQL para continuar respondendo ao mesmo enunciado proposto inicialmente.

Questão 10

1. **Listar os nomes dos 3 filmes que tiveram mais idiomas traduzidos.**

```
SELECT nome, count(Idioma) AS numIdiomas
FROM Titulos JOIN Traduz ON nome = nomeTitulo
GROUP BY nome
ORDER BY numIdiomas DESC
LIMIT 3
```

2. **Listar os nomes dos títulos que possuem todas as suas notas acima de 7. Liste também o valor de suas respectivas menores notas para mostrar que de fato não existe nenhuma abaixo de 7.**

```
SELECT DISTINCT nomeTitulo, min(nota)as notamin
FROM Avalia
GROUP BY nomeTitulo
HAVING min(nota) > 7
```

3. **Listar os filmes produzidos por diretores que realizaram 2 filmes ou mais.**

```
SELECT nome, diretor
FROM titulos
WHERE diretor IN (
  SELECT diretor
  FROM titulos
  GROUP BY diretor
  HAVING COUNT(nome) >= 2
)
```

4. **Listar os planos e as receitas que cada um deles trouxe graças às assinaturas.**

```
WITH num_titular_por_plano AS (
  SELECT plano, COUNT(*) as num_titular
  FROM pessoa
  WHERE plano is not null AND titular is NULL
  GROUP BY plano
)

SELECT P.nome, SUM(P.preco * N.num_titular) as receita
FROM planos P JOIN num_titular_por_plano N ON P.nome = N.plano
GROUP BY P.nome;
```

5. **Mostrar o filme mais popular, ou, seja, o filme que teve maior número de assinantes assistindo-o na plataforma.**

```
WITH title_visua AS (
  SELECT nomeTitulo, COUNT(*) as num_visua
  FROM assiste
  GROUP BY nomeTitulo
)
SELECT nomeTitulo
FROM title_visua
ORDER BY num_visua DESC
LIMIT 1;
```

6. **Listar os atores que já atuaram em mais de um filme e que são titulares de uma assinatura. Mostrar também os planos deles.**

```
SELECT p.nome, p.plano, count(a.nomeTitulo) as numfilmes
FROM pessoa p, atua a
WHERE p.id = a.id
AND p.titular IS NULL
GROUP BY p.nome, p.plano
HAVING count(a.nomeTitulo) > 1
```

Questão 11A

Essa visão ajuda a identificar pessoas diferentes que estão usando a mesma conta, o que pode ser útil para vermos o número de membros de cada conta.

```
CREATE VIEW mesmaConta AS
SELECT p1.nome AS nome1, p2.nome AS nome2
FROM pessoa p1, pessoa p2
WHERE p1.email = p2.email
AND p1.nome <> p2.nome;
```

ENUNCIADO: Encontrar todas as pessoas que estão usando a mesma conta que "MIGUEL CARNEIRO".

CONSULTA:

```
SELECT nome2
FROM mesmaConta
WHERE nome1 = 'MIGUEL CARNEIRO'
```
> Facilita encontrar pessoas que estão na mesma conta

Resultados da consulta

nome2
VINICIUS LUCENA
LUIZ GUILHERME SILVA DUARTE

Essa visão seleciona as avaliações acima da média de diferentes sites de avaliação, considerando os critérios de nota específica de cada site.

```
CREATE VIEW acima_da_media AS
SELECT *
FROM avalia
```

```
WHERE
((site = 'https://www.adorocinema.com/' OR site =
'https://filmow.com/') AND nota > 3.5)
OR
((site = 'https://www.imdb.com/' OR site =
'https://www.rottentomatoes.com/' OR site =
'https://www.senscritique.com' OR site =
'https://www.allocine.fr') AND nota > 5.0)
```

ENUNCIADO: Encontrar títulos avaliados acima da média nos sites de acordo com o critério de cada um.

CONSULTA:

```
SELECT nometitulo
FROM acima_da_media a, titulos t
WHERE (t.ano > 2000 AND t.ano <2020)
AND
a.nometitulo = t.nome
GROUP BY (nometitulo)
```

>Facilita encontrar títulos acima da média presentes no banco de dados

Resultados da consulta

nometitulo
Legalmente Loira
Ford vs Ferrari
The Witcher
Peaky Blinders
1917
Stranger Things
High School Musical 2
Dark
Avengers: Endgame
Jojo Rabbit
Coringa
Rocketman
Interstellar
Pixels
Spider-man: Homecoming
Diário de um Banana
Sai de Baixo: O Filme

Para fazer essa view, foi necessário adicionar mais linhas para a tabela ‘avalia’, dado que nem todos os títulos presentes na tabela ‘titulos’ estavam presentes. Os comandos utilizados foram:

```
INSERT INTO avalia(nometitulo, site, nota)
VALUES
  ('The Witcher', 'https://www.imdb.com/', 8.1),
  ('Stranger Things', 'https://www.adorocinema.com/', 4.6),
  ('Sai de Baixo: O Filme', 'https://www.imdb.com/', 5.2),
  ('Toy Story', 'https://www.adorocinema.com/', 4.5),
```

```
('Esqueceram de Mim', 'https://www.imdb.com/', 7.7),
('Avatar 2', 'https://www.adorocinema.com/', 4.0),
('Clube da Luta', 'https://www.imdb.com/', 8.8),
('Gente Grande', 'https://www.adorocinema.com/', 3.2),
('Spider-Man: Homecoming', 'https://www.imdb.com/', 7.4),
('Avengers: Endgame', 'https://www.adorocinema.com/', 4.6),
('High School Musical 1', 'https://www.adorocinema.com/', 3.5),
('High School Musical 2', 'https://www.imdb.com/', 5.1),
('High School Musical 3: Ano da Formatura', 'https://www.adorocinema.com/', 3.3),
('Esqueceram de Mim 2: Perdido em Nova York', 'https://www.imdb.com/', 6.8),
('Esqueceram de Mim 3', 'https://www.adorocinema.com/', 2.8),
('Esqueceram de Mim 4', 'https://www.imdb.com/', 2.6),
('Legalmente Loira', 'https://www.adorocinema.com/', 3.6);
('Diário de um Banana', 'https://www.imdb.com/', 6.2),
('Um Álibi Perfeito', 'https://www.adorocinema.com/', 3.4),
('Pixels', 'https://www.imdb.com/', 5.5),
('O Rei Leão', 'https://www.adorocinema.com/', 4.7),
('Pulp Fiction', 'https://www.imdb.com/', 8.9),
('Forrest Gump', 'https://www.adorocinema.com/', 4.6),
('Os Sete Pecados Capitais', 'https://www.imdb.com/', 8.6),
('O Silêncio dos Inocentes', 'https://www.adorocinema.com/', 4.6),
('Jurassic Park', 'https://www.imdb.com/', 8.2),
('O Resgate do Soldado Ryan', 'https://www.adorocinema.com/', 4.6),
('Matrix', 'https://www.imdb.com/', 8.7),
('O Sexto Sentido', 'https://www.adorocinema.com/', 4.4),
('Titanic', 'https://www.imdb.com/', 7.9),
('Coração Valente', 'https://www.adorocinema.com/', 4.4),
('Apollo 13', 'https://www.imdb.com/', 7.7);
```

Questão 11B

1. Visão para pessoas cadastradas com plano premium.

   CÓDIGO SQL DDL:

   ```
   CREATE VIEW premium_users AS
   SELECT nome, plano
   FROM pessoa
   WHERE plano = 'Premium'
   WITH CHECK OPTION;
   ```

   ATUALIZAÇÃO COM SUCESSO:

   ```
   UPDATE premium_users
   SET plano = 'Premium'
   WHERE nome like 'A%'
   ```

   Resultados da consulta

   5 linha(s) afetadas.

ATUALIZAÇÃO COM FRACASSO:

```
UPDATE premium_users
SET plano = 'Padrão'
WHERE nome like 'A%'
```

Erro de SQL:

```
ERROR:  new row violates check option for view "premium_users"
DETAIL:  Failing row contains (ABEL GALINDO MARQUES, 34539771140, 1984-08-25, Padrão, abel.marques82@gmail.com, 2020-06-27, milanek22, null).
```

No bloco:

```
UPDATE premium_users
SET plano = 'Padrão'
WHERE nome like 'A%'
```

2. Visão para os filmes de ação.

```
CREATE VIEW filme_acao AS
SELECT nome, categoria, diretor
FROM titulos
WHERE categoria like '%Ação%'
WITH CHECK OPTION;
```

ATUALIZAÇÃO COM SUCESSO:

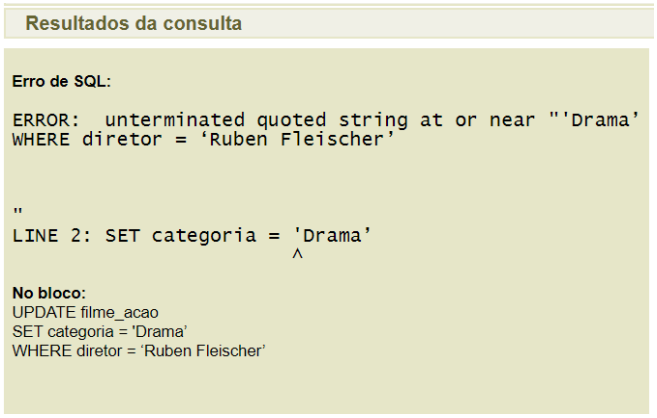
```
UPDATE filme_acao
SET categoria = 'Ação'
WHERE diretor = 'Ruben Fleischer'
```

Resultados da consulta

1 linha(s) afetadas.

ATUALIZAÇÃO COM FRACASSO:

```
UPDATE filme_acao
SET categoria = 'Drama'
WHERE diretor = 'Ruben Fleischer'
```



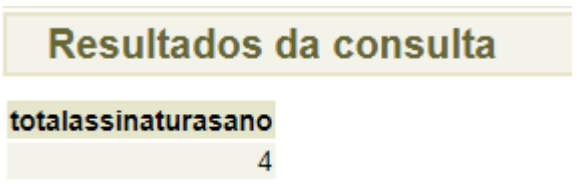
Questão 12A: Funções

1. PROPÓSITO: Definir uma função que receba como parâmetro um plano disponível na plataforma de streaming e um ano e retorne o total de assinaturas que a plataforma teve neste ano, para esse determinado plano.

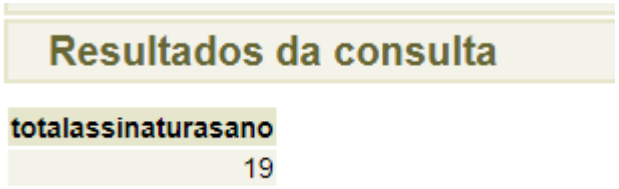
```
CREATE OR REPLACE FUNCTION totalAssinaturasAno(plano_nome VARCHAR,
ano INTEGER)
RETURNS INTEGER AS $$
DECLARE
    total_assinaturas INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO total_assinaturas
    FROM Pessoa
    WHERE plano = plano_nome
    AND EXTRACT(YEAR FROM datacadastro) = ano
    AND titular IS NULL;

    RETURN total_assinaturas;
END;
$$ LANGUAGE plpgsql;
```

Comandos em SQL para verificar que a função funciona de acordo com o esperado:
select totalAssinaturasAno ('Premium', 2021)



select totalAssinaturasAno ('Padrão', 2018)

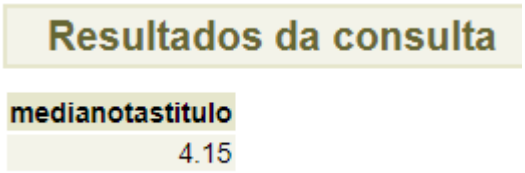


2. PROPÓSITO: Definir uma função que receba como parâmetro um título da plataforma de streaming e retorne a média de notas que esse título recebeu.

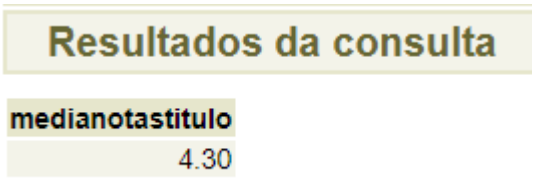
```
CREATE OR REPLACE FUNCTION mediaNotasTitulo(titulo_in VARCHAR)
RETURNS NUMERIC(10,2) AS $$
DECLARE
    media_notas NUMERIC(10,2);
BEGIN
    SELECT AVG(nota)::NUMERIC(10,2)
    INTO media_notas
    FROM Avalia
    WHERE nometitulo = titulo_in;

    RETURN media_notas;
END;
$$ LANGUAGE plpgsql;
```

Comandos em SQL para verificar que a função funciona de acordo com o esperado:
select medianotastitulo ('Jojo Rabbit')



select medianotastitulo ('1917')



Questão 12B

Essa questão foi feita após a questão 12C.

PROPÓSITO: O procedimento atualiza o plano de um usuário titular e, ao mesmo tempo, ajusta o plano de todos os seus dependentes para o mesmo plano. Isso é útil em um cenário onde um titular decide mudar o seu plano e deseja que todos os seus dependentes tenham o mesmo plano atualizado.

Além disso, vale considerar que a questão 12C interfere na 12B. Quando a função *update_plan* tenta atualizar o plano de todos os dependentes, o gatilho *check_dependents_trigger* é acionado para cada atualização de linha, verificando a integridade da quantidade de membros permitidos pelo plano.

A primeira tentativa para essa questão foi a criação do seguinte procedimento:

```
CREATE OR REPLACE FUNCTION update_plan(
    titular_id VARCHAR(20),
    new_plan VARCHAR(255)
)
RETURNS void AS $$
BEGIN
    UPDATE pessoa
    SET plano = new_plan
    WHERE id = titular_id OR titular = titular_id;

    RAISE NOTICE 'Plano atualizado para titular e dependentes';
END;
$$ LANGUAGE 'plpgsql';
```

Esse procedimento funciona corretamente em conjunto do trigger *check_dependents_trigger* quando o número de membros de um plano é maior ou menor que o titular mais a quantidade de dependentes dele. Ou seja, se um titular muda para um plano no qual o número máximo de membros é maior que a quantidade de dependentes que ele possui mais um (inclui o titular), há a atualização do plano na tabela pessoa tanto na tupla do titular quanto nas tuplas dos dependentes. Caso seja menor, não há a atualização.

Porém, se o número de dependentes mais o titular for igual ao número máximo de membros permitido pelo novo plano, a primeira atualização do plano pode passar, mas as atualizações subsequentes falharão, pois o gatilho será acionado novamente para cada dependente, levando a uma verificação que impede a mudança de plano devido ao número de dependentes.

Ou seja, na primeira atualização, a função *update_plan* tenta atualizar o plano do titular. O gatilho *check_dependents_trigger* é acionado e verifica a quantidade de dependentes, permitindo a mudança se a condição for satisfeita. Nas próximas atualizações, a função

update_plan tenta atualizar o plano de cada dependente. Cada vez que uma linha é atualizada, o gatilho *check_dependents_trigger* é acionado novamente, gerando um conflito, pois o gatilho verifica novamente o número de dependentes após cada atualização. Como o plano ainda não foi atualizado para todos, ele pode considerar que o número de dependentes excede o número máximo permitido pelo novo plano, bloqueando a atualização subsequente.

A fim de solucionar esse problema, desativamos temporariamente o gatilho durante a execução da função *update_plan*, e reativamos após a conclusão da atualização, garantindo que todas as atualizações sejam feitas sem interrupções, respeitando a integridade do número máximo de membros apenas uma vez, no início do processo. Dentro da função *update_plan*, incluímos a verificação se o número de dependentes mais um é menor ou igual ao número máximo de membros permitidos para realizar o procedimento.

Vale ressaltar que as funções para desativar e ativar o trigger foram procuradas na internet.

Função de desativar o trigger:

```
CREATE OR REPLACE FUNCTION disable_trigger(trigger_name TEXT, table_name TEXT)

RETURNS void AS $$

BEGIN

    EXECUTE 'ALTER TABLE ' || quote_ident(table_name) || ' DISABLE TRIGGER ' || quote_ident(trigger_name);

END;

$$ LANGUAGE plpgsql;
```

Função de ativar o trigger:

```
CREATE OR REPLACE FUNCTION enable_trigger(trigger_name TEXT, table_name TEXT)

RETURNS void AS $$

BEGIN

    EXECUTE 'ALTER TABLE ' || quote_ident(table_name) || ' ENABLE TRIGGER ' || quote_ident(trigger_name);

END;

$$ LANGUAGE plpgsql;
```

Procedimento para atualizar o plano:

```
CREATE OR REPLACE FUNCTION update_plan(
    titular_id VARCHAR(20),
    new_plan VARCHAR(255)
)
RETURNS void AS $$
DECLARE
    max_membros INTEGER;
    num_dependentes INTEGER;
BEGIN
    SELECT numMaxMembros INTO max_membros
    FROM planos
    WHERE nome = new_plan;

    SELECT COUNT(*) INTO num_dependentes
    FROM pessoa
    WHERE titular = titular_id;

    num_dependentes := num_dependentes + 1;

    IF num_dependentes <= max_membros THEN
                PERFORM  disable_trigger('check_dependents_trigger',
'pessoa');

        UPDATE pessoa
        SET plano = new_plan
```

```
        WHERE id = titular_id OR titular = titular_id;


                PERFORM  enable_trigger('check_dependents_trigger',
'pessoa');

    ELSE

          RAISE EXCEPTION 'O novo plano não permite o número atual
de membros (titular + dependentes)';

    END IF;

END;

$$ LANGUAGE plpgsql;
```

EXEMPLO TESTE

Verificando a quantidade de dependentes do titular de id '36641440664'. O total de membros, como visto abaixo, é dois. Ao atualizar o plano do titular para 'Padrão', há a atualização com sucesso do plano na tupla do titular e dos dependentes, já que o número máximo de membros do plano 'Padrão', três, não é menor que a quantidade de dependentes mais o titular.

```
select * from pessoa where id = '36641440664'
union
select * from pessoa where titular = '36641440664'
```

nome	id	datanasc	plano	email	datacadastro	senha	titular
AIDE SOARES TOJAL	36641440664	2004-06-09	Premium	aide.tojal@gmail.com	2017-03-03	PeMaKuHa2834@	*NULL*
ALEXANDRE MALTA DA COSTA MESSEDER	61459068122	1962-04-23	Premium	aide.tojal@gmail.com	2017-03-03	PeMaKuHa2834@	36641440664

2 linha(s)

- De plano Premium para plano Padrão

```
select * from update_plan( '36641440664','Padrão')
select * from pessoa where id = '36641440664'
union
select * from pessoa where titular = '36641440664'
```

nome	id	datanasc	plano	email	datacadastro	senha	titular
AIDE SOARES TOJAL	36641440664	2004-06-09	Padrão	aide.tojal@gmail.com	2017-03-03	PeMaKuHa2834@	*NULL*
ALEXANDRE MALTA DA COSTA MESSEDER	61459068122	1962-04-23	Padrão	aide.tojal@gmail.com	2017-03-03	PeMaKuHa2834@	36641440664

2 linha(s)

- Ao mudarmos o plano para ‘Padrão com anúncios’, que possui número máximo de membros igual a dois, a mesma quantidade da soma do titular com seus dependentes, o procedimento é feito com sucesso.

```
select * from update_plan( '36641440664','Padrão com anúncios’)

select * from pessoa where id = '36641440664'
union
select * from pessoa where titular = '36641440664'
```

nome	id	datanasc	plano	email	datacadastro	senha	titular
AIDE SOARES TOJAL	36641440664	2004-06-09	Padrão com anúncios	aide.tojal@gmail.com	2017-03-03	PeMaKuHa2834@	*NULL*
ALEXANDRE MALTA DA COSTA MESSEDER	61459068122	1962-04-23	Padrão com anúncios	aide.tojal@gmail.com	2017-03-03	PeMaKuHa2834@	36641440664

- Ao tentarmos mudar o plano para o ‘Básico’, que possui um como número máximo de membros, a atualização não é feita, pois o número de dependentes incluído com o titular, dois, é maior que um.

```
select * from update_plan( '36641440664','Básico')
```

Erro de SQL:

```
ERROR:  O novo plano não permite o número atual de membros (titular + dependentes)
CONTEXT:  PL/pgSQL function update_plan(character varying,character varying) line 25 at RAISE
```

No bloco:

```
select * from update_plan( '36641440664','Básico')
```

Questão 12C

PROPÓSITO: Definir uma função e um gatilho (trigger) no banco de dados que garantem a integridade semântica relacionada ao número de dependentes permitidos por plano na tabela pessoa.

A função check_num_dependents é uma função que verifica se o número de dependentes de um titular não excede o número máximo de membros permitido pelo plano.

O gatilho check_dependents_trigger associa a função check_num_dependents à tabela pessoa. Ele é executado antes de cada operação de inserção ou atualização na tabela.

```
CREATE OR REPLACE FUNCTION check_num_dependents()
RETURNS TRIGGER AS $$
DECLARE
    max_membros INTEGER;
    num_dependentes INTEGER;
BEGIN
```

```
    SELECT numMaxMembros INTO max_membros
    FROM planos
    WHERE nome = NEW.plano;

    SELECT COUNT(*) INTO num_dependentes
    FROM pessoa
    WHERE titular = NEW.titular;

    num_dependentes := num_dependentes + 1;

    IF num_dependentes > max_membros THEN
          RAISE EXCEPTION 'Número de dependentes excede o número
máximo de membros permitido pelo plano';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER check_dependents_trigger
BEFORE INSERT OR UPDATE ON pessoa
FOR EACH ROW
EXECUTE PROCEDURE check_num_dependents();
```

EXEMPLO TESTE:

É apenas um exemplo, onde é tentado inserir uma linha na tabela pessoa, no qual o id da pessoa é 1 e o tipo do plano é básico (possui apenas um membro, o próprio titular) e outra linha na tabela pessoa, no qual o id da pessoa é 2 e o id de seu titular 1. Porém, como o plano permite apenas um membro, **a inserção do bloco deu erro**. Logo após esse exemplo, houve uma remoção dessas linhas, pois os ids não correspondem à formatação utilizada.

```
INSERT INTO pessoa (nome, id, dataNasc, plano, email, dataCadastro, senha, titular)
VALUES ('João Silva', '1', '1980-01-01', 'Básico', 'joao@example.com',
'2024-06-24', 'senha123', NULL);

INSERT INTO pessoa (nome, id, dataNasc, plano, email, dataCadastro, senha, titular)
VALUES ('Maria Silva', '2', '2005-05-05', 'Básico', 'maria@example.com',
'2024-06-24', 'senha456', '1');
```

Erro de SQL:

```
ERROR:  Número de dependentes excede o número máximo de membros permitido pelo plano
CONTEXT:  PL/pgSQL function check_num_dependents() line 21 at RAISE
```

No bloco:

```
INSERT INTO pessoa (nome, id, dataNasc, plano, email, dataCadastro, senha, titular)
VALUES ('João Silva', '1', '1980-01-01', 'Básico', 'joao@example.com', '2024-06-24', 'senha123', NULL);

INSERT INTO pessoa (nome, id, dataNasc, plano, email, dataCadastro, senha, titular)
VALUES ('Maria Silva', '2', '2005-05-05', 'Básico', 'maria@example.com', '2024-06-24', 'senha456', '1');
```

EXEMPLO NO BANCO DE DADOS:

Ao executar o SQL a seguir, que permite ver os dependentes do titular que possui id '61592540902', notamos que o titular possui dois dependentes, o que corrobora com a integridade semântica de o titular possuir até dois dependentes, já que o plano é o Padrão (número máximo de membros é 3, incluindo o titular).

```
SELECT *
FROM pessoa
WHERE titular = '61592540902'
```

nome	id	datanasc	plano	email	datacadastro	senha	titular
LUIZ GUILHERME SILVA DUARTE	23024473360	2002-03-11	Padrão	vbitucortez@gmail.com	2024-06-04	SenhaForte	61592540902
MIGUEL CARNEIRO	65842154654	2004-04-15	Padrão	vbitucortez@gmail.com	2024-06-04	SenhaForte	61592540902

Ao tentar inserir mais uma tupla no qual a coluna titular possua o mesmo id '61592540902', notamos que deu erro, pois excedeu o número de membros permitidos no plano.

```
INSERT INTO pessoa (nome, id, dataNasc, plano, email, dataCadastro, senha, titular)
VALUES ('João Silva', '11882501756', '1980-01-01', 'Padrão', 'joao@example.com',
'2024-06-24', 'senha123', '61592540902');
```

Erro de SQL:

```
ERROR:  Número de dependentes excede o número máximo de membros permitido pelo plano
CONTEXT:  PL/pgSQL function check_num_dependents() line 21 at RAISE
```

No bloco:

```
INSERT INTO pessoa (nome, id, dataNasc, plano, email, dataCadastro, senha, titular)
VALUES ('João Silva', '11882501756', '1980-01-01', 'Padrão', 'joao@example.com', '2024-06-24', 'senha123', '61592540902');
```

Ao tentar atualizar uma tupla que já está na tabela, modificando o conteúdo da coluna titular para o id '61592540902', vemos que também ocorre o erro, pois excede o número de membros permitidos no plano do titular.

```
update pessoa
set titular = '61592540902'
where id = '16394613886'
```

Erro de SQL:

```
ERROR:  Número de dependentes excede o número máximo de membros permitido pelo plano
CONTEXT:  PL/pgSQL function check_num_dependents() line 21 at RAISE
```

No bloco:

```
update pessoa
set titular = '61592540902'
where id = '16394613886'
```

Um exemplo de sucesso é o caso da pessoa do titular de id ‘34539771140’, que possui plano Premium (máximo de 4 dependentes incluindo o titular). Com o SQL a seguir, notamos que ele possui apenas um dependente.

```
SELECT *
FROM pessoa
WHERE titular = '34539771140'
```

nome	id	datanasc	plano	email	datacadastro	senha	titular
DENISE BARATA	31902099010	1974-09-25	Premium	abel.marques82@gmail.com	2020-06-27	milanek22	34539771140

Ao inserir uma nova tupla no qual o conteúdo da coluna titular é o id ‘34539771140’, a inserção ocorreu sem problemas, pois ainda não atingiu o número máximo de membros do plano Premium.

```
INSERT INTO pessoa (nome, id, dataNasc, plano, email, dataCadastro, senha, titular)
VALUES ('João Silva', '11882501756', '1980-01-01', 'Padrão', 'joaosilva@gmail.com',
'2024-06-24', 'senha123', '34539771140');
```

1 linha(s) afetadas.

Tempo de execução total: 9.954 ms

SQL executado.

Editar SQL

Verificando a inserção:

```
SELECT *
FROM pessoa
WHERE titular = '34539771140'
```

nome	id	datanasc	plano	email	datacadastro	senha	titular
DENISE BARATA	31902099010	1974-09-25	Premium	abel.marques82@gmail.com	2020-06-27	milanek22	34539771140
João Silva	11882501756	1980-01-01	Premium	joaosilva@gmail.com	2024-06-24	senha123	34539771140

2 linha(s)

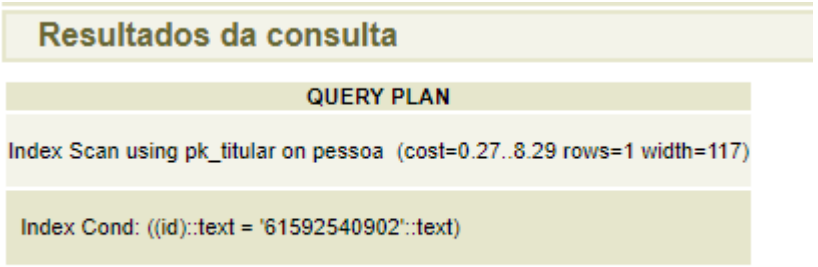
Questão 13A

Um índice primário é um tipo de índice que garante a unicidade e ordena fisicamente os dados na tabela. Ele também facilita a recuperação eficiente dos dados. Vamos considerar a tabela *pessoa*. Normalmente, um índice primário é criado sobre a coluna que identifica unicamente cada registro na tabela, como a coluna id em *pessoa*, que é chave primária.

O índice primário foi criado implicitamente ao definir a coluna id como PRIMARY KEY na definição da tabela *pessoa*. Ele é utilizado automaticamente pelo sistema de banco de dados para garantir a unicidade de valores na coluna id e para acelerar operações de busca, inserção, atualização e exclusão de registros com base nesta coluna.

Utilizando o explain em uma consulta SQL, podemos verificar que o índice está sendo utilizado.

```
EXPLAIN SELECT * FROM pessoa WHERE id = '61592540902'
```

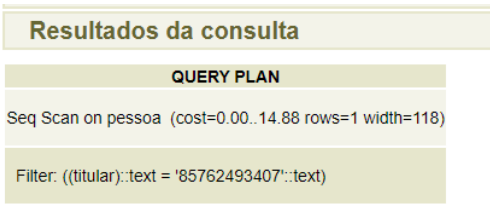


Questão 13B

1. PROPÓSITO: Criar um índice secundário na coluna ‘titular’ da tabela ‘pessoa’, uma vez que nosso banco de dados possui muitas operações que envolvem a verificação dos dependentes de um titular.

   Antes de criar o índice, vamos executar uma consulta na tabela pessoa, utilizando a coluna titular para observarmos o plano de execução gerado pelo comando **EXPLAIN** para acessar essa coluna e achar o id do titular especificado.

   ```
   EXPLAIN SELECT * FROM pessoa WHERE titular = '85762493407';
   ```

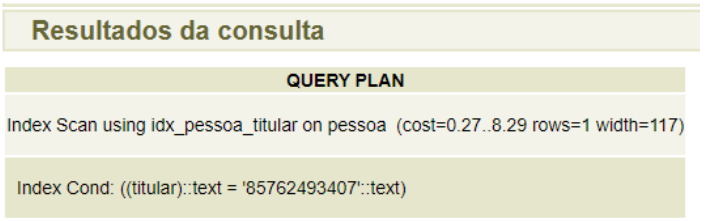


Criamos, então, o índice secundário na coluna ‘titular’:

```
CREATE INDEX idx_pessoa_titular ON pessoa(titular);
```

Vamos executar a mesma consulta definida acima, utilizando o comando **EXPLAIN**, para observarmos seu plano de execução após a criação do índice para a coluna ‘titular’.

```
EXPLAIN SELECT * FROM pessoa WHERE titular = '85762493407';
```



Comparando os resultados do plano de execução das consultas à coluna ‘titular’ da tabela ‘pessoa’ antes e depois da criação do índice, podemos analisar algumas diferenças.

Primeiramente, podemos garantir que o otimizador está utilizando o índice na segunda consulta, pois o plano de execução indica que está sendo realizada uma varredura de índice (‘*Index Scan*’), ao invés de uma varredura sequencial (‘*Seq Scan*’).
A varredura sequencial trata-se de uma leitura sequencial das linhas da tabela, começando pela primeira e lendo uma a uma até chegar na última. Esse método é utilizado quando não há índices disponíveis que possam ajudar na consulta ou quando o otimizador decide que a leitura sequencial é mais eficiente devido ao tamanho da tabela ou à natureza da consulta. Por isso, ele foi utilizado pelo otimizador antes de criarmos o índice.
Já através da varredura de índice, ao invés de ler todas as linhas da tabela o SGBD usa um índice para localizar rapidamente as linhas que atendem à condição da consulta. Portanto, após criarmos um índice para a coluna titular, o otimizador pode utilizar a varredura de índice para realizar a consulta solicitada nessa coluna.

Além disso, também podemos ver a diferença no *cost* entre as consultas sem índice e com índice, representando um maior desempenho para a realização das consultas trago pela utilização do índice. O *cost* no plano de execução é representado por dois valores: o custo inicial para começar a executar a consulta, e o custo total estimado para completar a execução da consulta.

Comparando os valores do cost:

Start-Up Cost:

- Sem indice: 0.00 - começa a leitura imediatamente

- Com índice: 0.27 - inclui o custo de acessar o índice e buscar a posição inicial.

Total Cost:

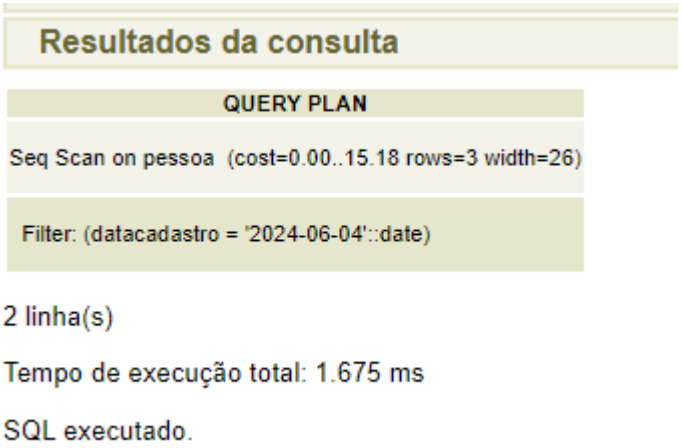
- Sem índice: 14.88 - custo total para ler todas as linhas da tabela e aplicar o filtro para retornar os resultados.
- Com índice: 8.29 - custo total para acessar o índice, buscar as linhas correspondentes e retornar os resultados.

A redução no total cost de 14.88 para 8.29 ao usar o índice *idx_pessoa_titular* mostra que o índice melhora a eficiência da consulta, tornando-a menos custosa, já que o índice permite que o SGBD localize rapidamente as linhas que satisfazem a condição titular = '85762493407' sem precisar ler todas as linhas da tabela. Embora haja um pequeno aumento no start-up cost de 0.27 ao utilizar o índice, o benefício da redução do *total cost* compensa isso.

2. PROPÓSITO: Criar um índice secundário para a tabela ‘pessoa’ na coluna ‘datacadastro’ tendo em vista o elevado número de tuplas na tabela pessoa, além de tornar menos custoso identificar os usuários que se cadastraram em uma determinada data.

   Utilizando o comando **EXPLAIN** podemos observar o plano de execução da seguinte consulta:

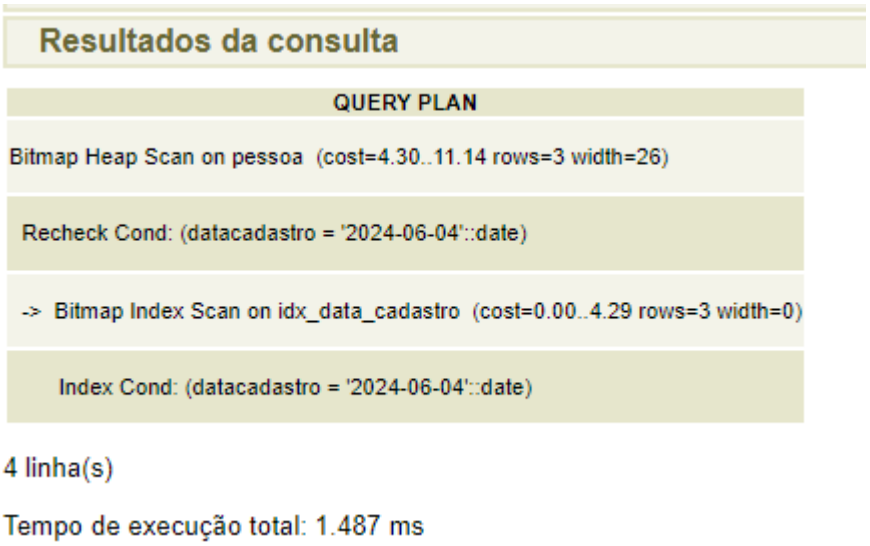
```
EXPLAIN SELECT nome
FROM pessoa
WHERE datacadastro = '2024-06-04';
```



Criação do índice secundário:

```
CREATE INDEX idx_data_cadastro ON pessoa(datacadastro);
```

Rodando novamente a consulta com o comando **EXPLAIN,** podemos visualizar a diferença entre rodar a consulta com e sem o índice. A execução desse comando nos gera um plano detalhado da consulta, o que torna visível algumas informações como o tipo do Scan utilizado, custo estimado da consulta, etc.



A criação do **idx_data_cadastro** melhora consideravelmente a eficiência da consulta, a tornando mais rápida e menos custosa. Apesar da utilização do índice ter adicionado um pequeno tempo inicial, no geral a consulta foi mais rápida com a sua utilização.

Start-Up Cost:

- Sem indice: 0.00 - começa a leitura imediatamente
- Com índice: 4.29 - inclui o custo de acessar o índice do Bitmap Index Scan

Total Cost:

- Sem índice: 15.18 - custo total para ler todas as linhas da tabela e aplicar o filtro para retornar os resultados.
- Com índice: 11.14 - custo total de execução para retornar os resultados.

De modo geral o índice melhora de forma significativa a eficiência da consulta, o que é ainda confirmado pelo tempo de execução (1.675ms sem o índice e 1.487ms com o índice). Embora possa parecer uma diferença pequena, ela demonstra uma boa otimização na busca que pode ser muito importante principalmente para tabelas que contenham uma quantidade muito grande de tuplas.