

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Gabriel Junges Baratto
Wesley Augusto Catuzzo de Bona

ATIVIDADE PRÁTICA SUPERVISIONADA -APS
Agência de viagens aéreas

PATO BRANCO
2021

Gabriel Junges Baratto
Wesley Augusto Catuzzo de Bona

ATIVIDADE PRÁTICA SUPERVISIONADA - APS

Atividade Prática Supervisionada de Banco de Dados 2 realizada no curso de graduação de Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Prof: Ives Pola

PATO BRANCO
2021

SUMÁRIO

1. Escolha de uma aplicação com dados correlacionados.	4
2. Elaboração de um diagrama Entidade-Relacionamento para a aplicação.	4
3. Mapeamento para o modelo relacional, indicando as restrições de integridade.	4
4. Script de criação de tabelas no PostgreSQL com as devidas restrições.	5
5. Popule as tabelas com alguns dados (preferência reais, se possível), suficiente para consultas.	7
6. Criação de índices adequados às consultas.	11
7. Criação de funções para consultas corriqueiras ou tarefas nas tabelas.	17
8. Criação de visões	19
9. Criação de uma tabela de auditoria.	22
10. Criação de triggers.	24
ANEXO - Links para os arquivos csv e sql	28

1. Escolha de uma aplicação com dados correlacionados.

Para colocar em prática os conceitos discutidos na disciplina de Banco de Dados 2, é proposto uma atividade prática a ser desenvolvida seguindo algumas especificações. O projeto a ser desenvolvido será um banco de dados para o controle de uma agência de viagens aéreas. O banco será desenvolvido em PostgreSQL, cumprindo alguns requisitos estabelecidos no projeto.

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

Pensando na aplicação escolhida pelo grupo, foi elaborado o *modelo conceitual* da Figura 1, utilizando-se da software de modelagem *brModelo*.

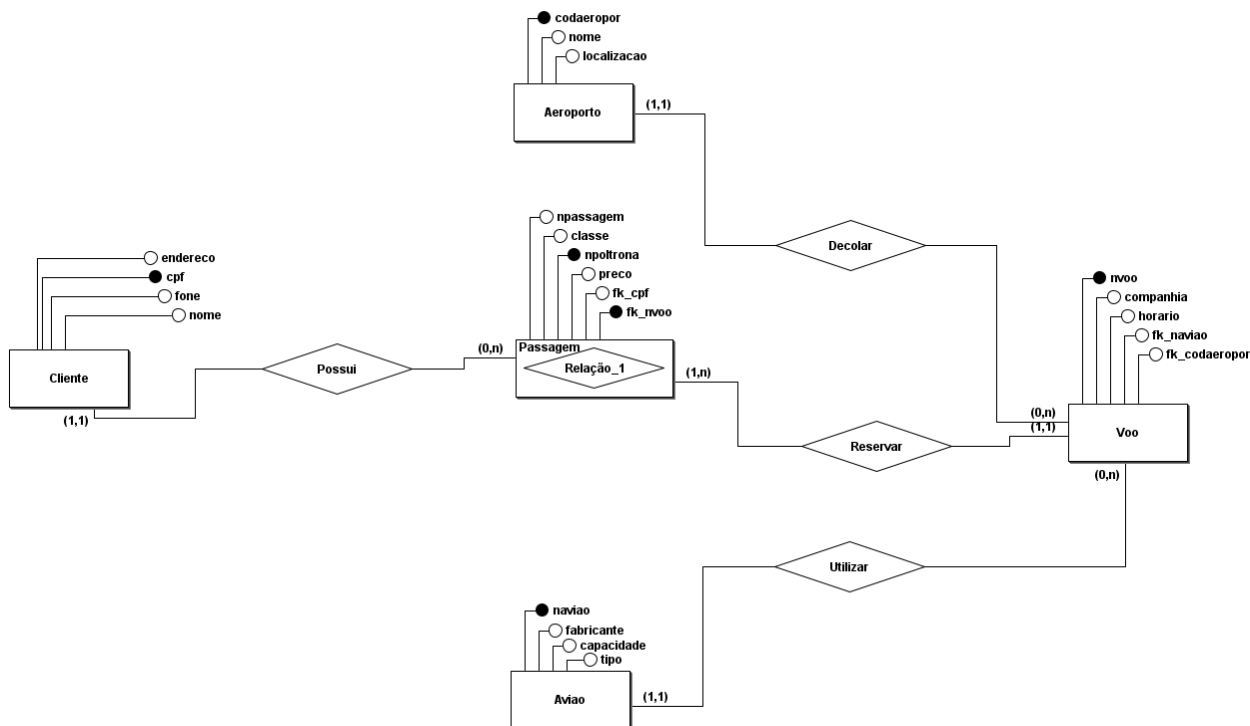


Figura 1 - Construção do modelo conceitual do sistema proposto.

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

A partir do modelo conceitual apresentado na tópicos anterior, conseguimos criar um mapeamento para o modelo relacional utilizando o software DBeaver (Figura 2). Em relação às restrições de integridade atribuímos NOT NULL para todos os campos exceto endereço e o fone do cliente, por conta da importância de todos

os campos. Além disso, fizemos a checagem de algumas variáveis que necessitavam ser maiores do que zero como o número de telefone, número de poltronas, capacidade dos aviões e o número das passagens. Também, além das chaves primárias, que precisam no PostgreSQL obrigatoriamente ser únicas, o campo que armazena o número da passagem, que não é chave primária, precisou da restrição UNIQUE, para garantir que ela seja única.

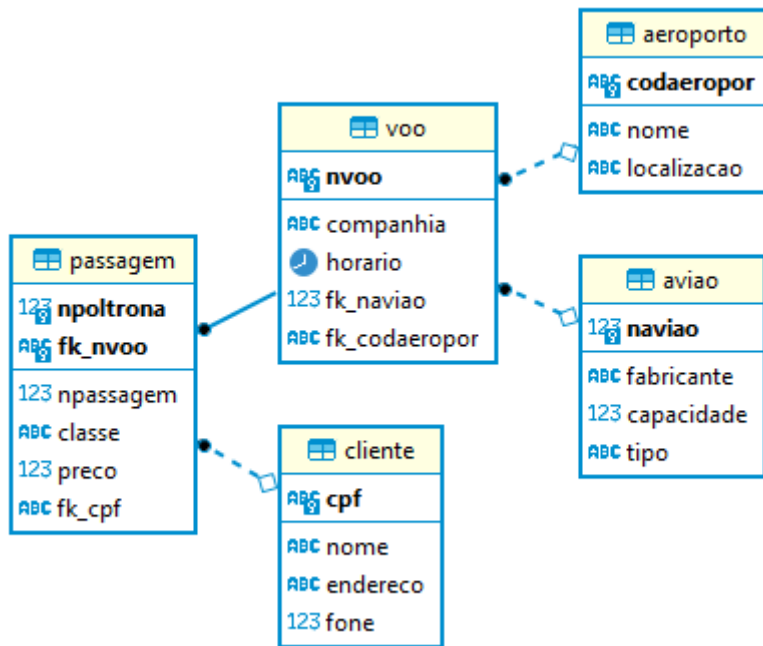


Figura 2 - Diagrama Entidade-Relacionamento construído a partir do modelo conceitual

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

```
CREATE TABLE IF NOT EXISTS public.cliente
(
    cpf character varying(100) NOT NULL,
    nome character varying(100) NOT NULL,
    endereco text,
    fone bigint,
    PRIMARY KEY (cpf),
    CONSTRAINT check_person CHECK(fone>0)
);

CREATE TABLE IF NOT EXISTS public.aviao
```

```

(
    naviao integer NOT NULL,
    fabricante character varying(100) NOT NULL,
    capacidade integer NOT NULL,
    tipo character varying(100) NOT NULL,
    PRIMARY KEY (naviao),
    CONSTRAINT check_plane CHECK(capacidade>0)
);

CREATE TABLE IF NOT EXISTS public.aeroporto
(
    codaeropor character varying(100) NOT NULL,
    nome character varying(100) NOT NULL,
    localizacao text NOT NULL,
    PRIMARY KEY (codaeropor)
);

CREATE TABLE IF NOT EXISTS public.voo
(
    nvoo character varying(100) NOT NULL,
    companhia character varying(100) NOT NULL,
    horario date NOT NULL,
    fk_naviao integer NOT NULL,
    fk_codaeropor character varying(100) NOT NULL,
    PRIMARY KEY (nvoo)
);

CREATE TABLE IF NOT EXISTS public.passagem
(
    npassagem bigint NOT NULL UNIQUE,
    classe character varying(20) NOT NULL,
    npoltrona integer NOT NULL,
    preco double precision NOT NULL,
    fk_cpf character varying(100) NOT NULL,
    fk_nvoo character varying(100) NOT NULL,
    PRIMARY KEY (npoltrona, fk_nvoo),
    CONSTRAINT check_passage CHECK(npoltrona>0 AND preco>=0 AND
                                     npassagem>0)
);

ALTER TABLE public.voo
    ADD FOREIGN KEY (fk_naviao)

```

```
REFERENCES public.aviao (naviao)
NOT VALID;

ALTER TABLE public.voo
ADD FOREIGN KEY (fk_codaeropor)
REFERENCES public.aeroporto (codaeropor)
NOT VALID;

ALTER TABLE public.passagem
ADD FOREIGN KEY (fk_cpf)
REFERENCES public.cliente (cpf)
NOT VALID;

ALTER TABLE public.passagem
ADD FOREIGN KEY (fk_nvoo)
REFERENCES public.voo (nvoo)
NOT VALID;
```

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

A geração de dados para população das tabelas foi realizada através do site *mockaroo*, em conjunto com scripts implementados no próprio PostgreSQL. Foram adicionadas 1000 tuplas por tabela. Para a inserção dos dados, antes armazenados em arquivos de texto (Figura 3), foi utilizado o comando COPY, que utiliza os dados do arquivo para a população da tabela. Os dados gerados foram utilizados para realizar testes nas consultas e a análise do funcionamento das tabelas, e estão disponíveis no [repositório do GitHub](#).



```
aeroporto - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
codaerop, nome, localizacao
R222-61:343, Kings, 79 Cambridge Center
L222-37:924, Raven, 157 Northridge Point
D222-72:334, Everett, 00589 Jenna Circle
Q222-28:997, Logan, 303 Kedzie Place
S222-73:598, Schmedeman, 32368 Menomonie Road
I222-45:326, Mendota, 43 Little Fleur Alley
M222-34:513, South, 701 Northport Circle
A222-23:267, Veith, 9094 Scofield Pass
A222-61:115, Hanover, 035 La Follette Junction
Y222-67:014, Crownhardt, 52147 Stoughton Road
Y222-45:669, Farragut, 505 Tony Crossing
Z222-78:496, Pawling, 279 Lukken Road
Y222-39:041, Bunting, 542 Oriole Center
W222-18:032, Buena Vista, 95 Buena Vista Plaza
P222-26:597, Fordem, 7993 Blue Bill Park Crossing
A222-43:939, Linden, 60421 Melrose Street
F222-82:324, Buhler, 04812 Sundown Hill
K222-90:537, Charing Cross, 7294 Scott Circle
```

Figura 3 - Arquivo gerado pelo *mockaroo* para população das tabelas

O código abaixo mostra o script implementado para a importação dos dados na tabela *aeroporto*. As tabelas *voo* e *passagem*, que possuem campos dependentes, necessitaram de scripts especiais, para a geração aleatória de *foreign keys* e campos dependentes válidos.

```
COPY cliente
FROM 'C:\Users\Public\Downloads\pessoas.csv'
DELIMITER ','
CSV HEADER;
```

```
--select * from cliente
```

```
COPY aviao
FROM 'C:\Users\Public\Downloads\avioes.csv'
DELIMITER ','
CSV HEADER;
```

```
--select * from aviao
```

```
COPY aeroporto
```



```

FROM 'C:\Users\Public\Downloads\aeroporto.csv'
DELIMITER ','
CSV HEADER;

CREATE TEMPORARY TABLE IF NOT EXISTS temp1
(
    nvoo character varying(100) NOT NULL,
    companhia character varying(100) NOT NULL,
    horario date NOT NULL,
    PRIMARY KEY (nvoo)
);

COPY temp1
FROM 'C:\Users\Public\Downloads\voo.csv'
DELIMITER ','
CSV HEADER;
alter table temp1
add column id1 SERIAL NOT NULL;

--select * from temp1
--select * from aeroporto
--(select cpf from cliente order by RANDOM() limit 1;)

do $$
declare
aa temp1%ROWTYPE;
aviaoo aviao.naviao%TYPE;
aero aeroporto.codaeropor%TYPE;
begin
    for i in 1..1000 loop
        select * into aa from temp1 where id1=i;
        select naviao into aviaoo from aviao order by RANDOM()
limit 1;
        select codaeropor into aero from aeroporto order by
RANDOM() limit 1;
        insert into voo values
(aa.nvoo,aa.companhia,aa.horario,aviaoo,aero);
    end loop;
end; $$
language plpgsql;
DROP TABLE temp1;
--select * from voo join aviao on voo.fk_naviao=aviao.naviao

```

```

CREATE TEMPORARY TABLE IF NOT EXISTS temp2
(
    npassagem bigint NOT NULL,
    data date NOT NULL,
    classe character varying(20) NOT NULL,
    preco double precision,
    PRIMARY KEY (npassagem)
);

COPY temp2
FROM 'C:\Users\Public\Downloads\passagem.csv'
DELIMITER ','
CSV HEADER;
--select * from temp2
alter table temp2 add column id1 serial;

do $$
declare
pass1 temp2%ROWTYPE;
cpff cliente.cpf%TYPE;
vooo voo%ROWTYPE;
cap aviao.capacidade%TYPE;
polt passagem.npoltrona%TYPE;
begin
    for i in 1..1000 loop
        select * into pass1 from temp2 where id1=i;
        select * into vooo from voo order by RANDOM() limit 1;
        select cpf into cpff from cliente order by RANDOM()
limit 1;
        select capacidade into cap from aviao where
aviao.naviao=vooo.fk_naviao;
        select npoltrona into polt from passagem where
passagem.fk_nvoo=vooo.nvoo order by npoltrona desc limit 1;
        if(polt IS NULL) then
            polt=0;
        elsif(polt=cap) then
            i:=i-1;
            continue;
        end if;
        insert into passagem values
(pass1.npassagem,pass1.classe,polt+1,pass1.preco,cpff,vooo.nvoo);

```

```

        end loop;
end; $$
language plpgsql;
--select * from passagem
DROP TABLE temp2;

```

6. Criação de índices adequados às consultas.

A seguir encontram-se alguns exemplos de consultas que seriam provavelmente frequentes durante a utilização do sistema.

1. Quantidade de voos diários durante um ano (pode ser também por aeroporto).
 - a. Criação de índice para voo.horario (ver melhor tipo de índice para tipo data)
 - b. Exemplo de consulta:

```

SELECT count(nvoo) as numero_voos
FROM voo JOIN aeroporto
ON aeroporto.codaeropor=voo.fk_codaeropor
WHERE (voo.horario BETWEEN '2024-01-01' AND
'2025-01-01') and aeroporto.nome ='Talisman';

```

- c. Índices criados para otimização:

```

CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE INDEX IF NOT EXISTS voo_idx_horario on
voo(horario);
CREATE INDEX IF NOT EXISTS aeroporto_idx_nome on
aeroporto(nome);
CREATE INDEX IF NOT EXISTS aeroporto_idx_trgm_nome on
aeroporto using GIN(nome gin_trgm_ops);

```

- d. Resultados da consulta e explain analyze:

	numero_voos bigint	aeroporto character varying (100)
1	6	1st
2	1	3rd
3	3	5th
4	3	8th
5	1	Aberg
6	2	Acker
7	1	Alpine
8	2	American

(a) Resultado da consulta

	QUERY PLAN text
1	Sort (cost=85.09..86.16 rows=42 / width=16) (actual time=1.605..1.615 rows=334 loops=1)
2	[...] Sort Key: aeroporto.nome
3	[...] Sort Method: quicksort Memory: 45kB
4	[...] -> HashAggregate (cost=62.17..66.44 rows=427 width=16) (actual time=0.893..0.926 rows=334 loops=1)
5	[...] Group Key: aeroporto.nome
6	[...] Batches: 1 Memory Usage: 61kB
7	[...] -> Hash Join (cost=31.50..58.46 rows=742 width=20) (actual time=0.319..0.613 rows=740 loops=1)
8	[...] Hash Cond: ((voo.fk_codaeropor)::text = (aeroporto.codaeropor)::text)
9	[...] -> Seq Scan on voo (cost=0.00..25.00 rows=742 width=24) (actual time=0.017..0.173 rows=740 loops=1)
10	[...] Filter: ((horario >= '2020-01-01'::date) AND (horario <= '2025-01-01'::date))
11	[...] Rows Removed by Filter: 260
12	[...] -> Hash (cost=19.00..19.00 rows=1000 width=20) (actual time=0.290..0.291 rows=1000 loops=1)
13	[...] Buckets: 1024 Batches: 1 Memory Usage: 60kB
14	[...] -> Seq Scan on aeroporto (cost=0.00..19.00 rows=1000 width=20) (actual time=0.009..0.092 rows=1000 loops=1)
15	Planning Time: 0.489 ms
16	Execution Time: 1.669 ms

(b) Explain analyze antes dos índices

	QUERY PLAN
	text
1	Aggregate (cost=32.45..32.46 rows=1 width=8) (actual time=0.076..0.077 rows=1 loops=1)
2	[...] -> Hash Join (cost=12.68..32.45 rows=1 width=12) (actual time=0.064..0.074 rows=1 loops=1)
3	[...] Hash Cond: ((voo.fk_codaeropor)::text = (aeroporto.codaeropor)::text)
4	[...] -> Index Scan using voo_idx_horario on voo (cost=0.28..19.38 rows=255 width=24) (actual time=0.011..0.035 rows=254 loops=1)
5	[...] Index Cond: ((horario >= '2024-01-01'::date) AND (horario <= '2025-01-01'::date))
6	[...] -> Hash (cost=12.36..12.36 rows=4 width=12) (actual time=0.013..0.014 rows=4 loops=1)
7	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	[...] -> Bitmap Heap Scan on aeroporto (cost=4.31..12.36 rows=4 width=12) (actual time=0.009..0.011 rows=4 loops=1)
9	[...] Recheck Cond: ((nome)::text = 'Talisman'::text)
10	[...] Heap Blocks: exact=3
11	[...] -> Bitmap Index Scan on aeroporto_idx_nome (cost=0.00..4.31 rows=4 width=0) (actual time=0.007..0.007 rows=4 loops=1)
12	[...] Index Cond: ((nome)::text = 'Talisman'::text)
13	Planning Time: 0.333 ms
14	Execution Time: 0.106 ms

(c) Explain analyze utilizando índices

Figura 4 - Quantidade de voos por aeroporto

2. Quantidade de voos por empresa num intervalo de tempo

a. Exemplo de consulta:

```
SELECT count(nvoo) as numero_voos
FROM voo WHERE companhia='GOL'
AND (voo.horario between '2024-01-01' AND '2025-01-01');
```

b. Índices criados:

```
CREATE INDEX IF NOT EXISTS voo_idx_companhia on
voo(companhia);
```

c. Resultados e explain analyze:

	numero_voos bigint	companhia character varying (100)
1	96	Azul
2	92	GOL
3	88	Copa Airlines
4	86	Aeromexico
5	81	Emirates Airlines
6	79	Delta Air Lines
7	78	Avianca
8	71	Air France
9	69	LATAM

(a) Resultado da consulta

	QUERY PLAN text
1	Sort (cost=28.94..28.97 rows=9 width=18) (actual time=0.247..0.248 rows=9 loops=1)
2	[...] Sort Key: (count(nvoo)) DESC
3	[...] Sort Method: quicksort Memory: 25kB
4	[...] -> HashAggregate (cost=28.71..28.80 rows=9 width=18) (actual time=0.239..0.240 rows=9 loops=1)
5	[...] Group Key: companhia
6	[...] Batches: 1 Memory Usage: 24kB
7	[...] -> Seq Scan on voo (cost=0.00..25.00 rows=742 width=22) (actual time=0.015..0.116 rows=740 loops=1)
8	[...] Filter: ((horario >= '2020-01-01'::date) AND (horario <= '2025-01-01'::date))
9	[...] Rows Removed by Filter: 260
10	Planning Time: 0.148 ms
11	Execution Time: 0.278 ms

(b) Explain analyze antes dos índices

	QUERY PLAN text
1	Aggregate (cost=16.13..16.14 rows=1 width=8) (actual time=0.046..0.046 rows=1 loops=1)
2	[...] -> Bitmap Heap Scan on voo (cost=5.03..16.06 rows=30 width=12) (actual time=0.027..0.042 rows=28 loops=1)
3	[...] Recheck Cond: ((companhia)::text = 'GOL'::text)
4	[...] Filter: ((horario >= '2024-01-01'::date) AND (horario <= '2025-01-01'::date))
5	[...] Rows Removed by Filter: 88
6	[...] Heap Blocks: exact=9
7	[...] -> Bitmap Index Scan on voo_idx_companhia (cost=0.00..5.02 rows=116 width=0) (actual time=0.015..0.015 rows=11...
8	[...] Index Cond: ((companhia)::text = 'GOL'::text)
9	Planning Time: 0.090 ms
10	Execution Time: 0.079 ms

(c) Explain analyze utilizando índices

Figura 5 - Quantidade de voos por empresa num intervalo de tempo

3. Exibição do nome dos passageiros que compraram passagens em uma determinada faixa de preços.

a. Exemplo de consulta:

```
SELECT cliente.nome, passagem.preco
FROM cliente JOIN passagem
ON cliente.cpf=passagem.fk_cpf
WHERE passagem.preco BETWEEN 1499.99 AND 2000.00
ORDER BY passagem.preco DESC;
```

b. Índices criados:

```
CREATE INDEX IF NOT EXISTS passagem_idx_preco on
passagem(preco);
```

c. Resultados e explain analyze:

	nome character varying (100)	preco double precision
1	Lorin Offill	1996.18
2	Durand Patching	1987.01
3	Leilah Boak	1982.78
4	Natividad Winterbottom	1981.45
5	Sean Leason	1977.52
6	Avie Langabeer	1974.68
7	Allix Wanden	1974.33
8	Sinclair Isaksson	1974.21
9	Deana Lempertz	1972.28
10	Lauri MacIlriach	1972.12
11	Alexina Ridners	1969.91

(a) Resultado da consulta

	QUERY PLAN	
	text	
1	Sort (cost=67.21..67.64 rows=175 width=22) (actual time=0.338..0.346 rows=174 loops=1)	
2	[...] Sort Key: passagem.preco DESC	
3	[...] Sort Method: quicksort Memory: 38kB	
4	[...] -> Hash Join (cost=29.19..60.69 rows=175 width=22) (actual time=0.149..0.295 rows=174 loops=1)	
5	[...] Hash Cond: ((cliente.cpf)::text = (passagem.fk_cpf)::text)	
6	[...] -> Seq Scan on cliente (cost=0.00..21.00 rows=1000 width=26) (actual time=0.011..0.068 rows=1000 loops=1)	
7	[...] -> Hash (cost=27.00..27.00 rows=175 width=20) (actual time=0.131..0.132 rows=174 loops=1)	
8	[...] Buckets: 1024 Batches: 1 Memory Usage: 17kB	
9	[...] -> Seq Scan on passagem (cost=0.00..27.00 rows=175 width=20) (actual time=0.009..0.111 rows=174 loops=1)	
10	[...] Filter: ((preco >= '1499.99'::double precision) AND (preco <= '2000'::double precision))	
11	[...] Rows Removed by Filter: 826	
12	Planning Time: 0.834 ms	
13	Execution Time: 0.363 ms	

(b) Explain analyze antes dos índices

	QUERY PLAN	
	text	
1	Sort (cost=60.90..61.34 rows=175 width=22) (actual time=0.288..0.294 rows=174 loops=1)	
2	[...] Sort Key: passagem.preco DESC	
3	[...] Sort Method: quicksort Memory: 38kB	
4	[...] -> Hash Join (cost=22.88..54.38 rows=175 width=22) (actual time=0.095..0.244 rows=174 loops=1)	
5	[...] Hash Cond: ((cliente.cpf)::text = (passagem.fk_cpf)::text)	
6	[...] -> Seq Scan on cliente (cost=0.00..21.00 rows=1000 width=26) (actual time=0.008..0.063 rows=1000 loops=1)	
7	[...] -> Hash (cost=20.69..20.69 rows=175 width=20) (actual time=0.081..0.081 rows=174 loops=1)	
8	[...] Buckets: 1024 Batches: 1 Memory Usage: 17kB	
9	[...] -> Bitmap Heap Scan on passagem (cost=6.07..20.69 rows=175 width=20) (actual time=0.030..0.061 rows=174...	
10	[...] Recheck Cond: ((preco >= '1499.99'::double precision) AND (preco <= '2000'::double precision))	
11	[...] Heap Blocks: exact=12	
12	[...] -> Bitmap Index Scan on passagem_idx_preco (cost=0.00..6.03 rows=175 width=0) (actual time=0.025..0.025 r...	
13	[...] Index Cond: ((preco >= '1499.99'::double precision) AND (preco <= '2000'::double precision))	
14	Planning Time: 0.314 ms	
15	Execution Time: 0.324 ms	

(c) Explain analyze utilizando índices

Figura 6 - Classes de voo mais contratada pelos clientes, em uma dada companhia aérea, em um intervalo de tempo.

Comparando as consultas antes (Figuras 4b, 5b e 6b) e depois da criação dos índices (Figuras 4c, 5c e 6c), observa-se que os índices conseguiram reduzir os tempos para a realização das mesmas.

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

1. Função para listar as passagens de um cliente pelo seu cpf

a. Código da função:

```
CREATE OR REPLACE FUNCTION
public.getPassagensCliente(cpf1 varchar(100))
RETURNS TABLE (poltrona integer,num_voo varchar(100),
                horario date,preco double
precision,companhia varchar(100),
                cod_aviao integer,nome_aeroporto
varchar(100)) as $$
BEGIN
    RETURN QUERY SELECT p.npoltrona, v.nvoo,
v.horario, p.preco, v.companhia, v.fk_naviao as
id_aviao, a.nome as aeroporto
    FROM cliente as c
    JOIN passagem as p ON p.fk_cpf = c.cpf
    JOIN voo as v ON v.nvoo = p.fk_nvoo
    JOIN aeroporto as a ON
a.codaeropor=v.fk_codaeropor
    WHERE cpf = cpf1;
END;
$$ LANGUAGE plpgsql;
```

b. Exemplo de uso:

```
select * from getPassagensCliente('317-10-0451');
```

c. Exemplo de retorno:

	poltrona integer	num_voo character varying	horario date	preco double precision	companhia character varying	cod_aviao integer	nome_aeroporto character varying
1	1	94-718-0667	2023-05-29	386.98	Aeromexico	523	Cambridge
2	2	73-119-7802	2025-04-09	2793.38	Azul	687	Schmedeman

Figura 7 - Informações sobre as passagens de um cliente listadas através do cpf.

2. Função para verificar os voos pela data

a. Código da função:

```

CREATE OR REPLACE FUNCTION public.getDecolagensData(data
date)
RETURNS TABLE (num_voo varchar(100), companhia
varchar(100), nome_aeroporto varchar(100), capacidade
integer, tipo_aviao varchar(100)) as $$
BEGIN
    RETURN QUERY SELECT v.nvoo, v.companhia, ae.nome,
av.capacidade, av.tipo
    FROM voo as v
    JOIN aeroporto as ae ON
v.fk_codaeropor=ae.codaeropor
    JOIN aviao as av ON v.fk_naviao=av.naviao
    WHERE horario = data;
END;
$$ LANGUAGE plpgsql;

```

b. Exemplo de chamada da função:

```
select * from getDecolagensData('2025-12-29')
```

c. Exemplo de retorno da função:

	num_voo character varying	companhia character varying	nome_aeroporto character varying	capacidade integer	tipo_aviao character varying
1	75-789-6234	Aeromexico	Kingsford	586	Embraer
2	60-781-3268	Air France	Shoshone	44	Embraer

Figura 8 - Informações de voos em determinado dia.

3. Função para retornar a primeira poltrona livre que encontrar em determinado voo

a. Código da função:

```

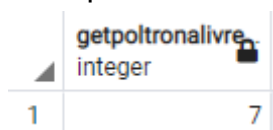
CREATE OR REPLACE FUNCTION getPoltronaLivre(nvoo1
varchar(100))
RETURNS integer AS $$
DECLARE
    voo1 VOO%ROWTYPE;
    cap AVIAO.capacidade%TYPE;
    i integer;
BEGIN
    SELECT * FROM voo INTO voo1 WHERE nvoo=nvoo1;
    IF voo1 IS NULL THEN
        RAISE EXCEPTION 'O voo informado não
existe!';
    END IF;
    SELECT capacidade INTO cap FROM aviao WHERE
aviao.naviao=voo1.fk_naviao;
    FOR i IN 1..cap LOOP
        IF EXISTS(SELECT 1 FROM passagem WHERE
npoltrona=i) THEN
            CONTINUE;
        ELSE
            RETURN i;
        END IF;
    END LOOP;
    RAISE EXCEPTION 'Não há poltronas livres!';
END;
$$ LANGUAGE plpgsql;

```

b. Exemplo de chamada da função:

```
select getPoltronaLivre('60-781-3268');
```

c. Exemplo de retorno da função:



getpoltronalivre
integer
1
7

Figura 9 - Primeiro número de poltrona livre encontrada para o voo informado.

8. Criação de visões

1. Para o primeiro exemplo de visão, foi escolhida uma consulta select que retorna o nome do cliente, cpf e valor total gasto em passagens que mais comprem passagens da agência de viagens. Esta consulta poderia ser útil

para a agência de viagens no levantamento de quais clientes investem mais dinheiro em sua empresa, de forma que poderiam receber um tratamento personalizado, por exemplo.

a. Código de criação da visão:

```
create view clientes_mais_valiosos as
select c.nome,c.cpf,sum(p.preco) as valor_gasto
from passagem p
join cliente c on p.fk_cpf=c.cpf
group by c.cpf
having sum(p.preco)>8000
order by valor_gasto desc;
```

b. Exemplo de uso da visão:

```
select * from clientes_mais_valiosos
```

c. Exemplo de retorno da função:

	nome character varying (100)	cpf character varying (100)	valor_gasto double precision
1	Wenonah Nursey	684-53-8691	11881.1
2	Garrott Jobes	368-66-8223	10295.750000000002
3	Barde Thorowgood	688-98-8179	9478.51
4	Conn Mileham	584-72-7113	8904.95
5	Illa Van Cassel	732-34-6880	8143.89
6	Arabel Russilll	739-49-3748	8139.860000000001
7	Elyse Bischof	170-27-1961	8114.300000000001
8	Fiss Pentelow	386-67-4074	8041.400000000001

Figura 10 - Retorno da seleção. Listagem de nomes e cpfs dos clientes que gastaram mais de oito mil reais.

2. Para o segundo exemplo de visão, foi criado um código para consultar os aviões disponíveis para uma dada companhia desejada, mostrando o nome do fabricante, tipo e também a capacidade de passageiros de cada aeronave. Essas informações poderiam ser úteis para os administradores que desejam criar pacotes de viagens de acordo com os orçamentos dos clientes, podendo optar por aeronaves que eles considerem mais confortáveis e melhor

atendimento, ou por aquelas mais simples e com maior capacidade, por exemplo.

a. Código de criação da visão:

```
create view avioes_da_gol as
select a.fabricante,a.tipo,a.capacidade from aviao a
join voo v
on a.naviao=v.fk_naviao
where v.companhia='GOL'
order by a.fabricante,a.tipo;
```

b. Exemplo de uso da visão:

```
select * from avioes_da_gol;
```

c. Exemplo de retorno da função:

	fabricante character varying (100)	tipo character varying (100)	capacidade integer
1	Bartell LLC	Airbus	91
2	Bashirian-Romaguera	Embraer	63
3	Baumbach, Klein and Adams	Bombardier	213
4	Beahan, Koepp and Gerhold	Embraer	513
5	Beatty-Konopelski	Embraer	364
6	Bednar-Koss	Boeing	395
7	Bins-Lind	Embraer	350
8	Brown, Shanahan and Sanford	Bombardier	214

Figura 11 - Retorno da seleção. Listagem de aviões da GOL e suas informações.

3. Para o terceiro exemplo de visão, foi criado um código especificamente para geração de um relatório para a empresa. A consulta realizada pela visão retorna o número de passagens vendidas, o número de voos e o lucro bruto, tudo isso de forma anual. Com isso, a agência e seus administradores poderiam gerir melhor suas despesas e lucros, e obteriam mais informações para a tomada de decisões, verificando se há uma tendência de crescimento no último ano, em relação aos anos anteriores.

a. Código de criação da visão:

```
create view relatorioanual as
select extract(YEAR from horario) as ano,
       count(p.npoltrona) as passagens_vendidas,
       count(v.nvoo) as numero_voos,
       round(sum(p.preco)::numeric,2) as lucro_bruto
from voo v join passagem p on p.fk_nvoo=v.nvoo
group by ano;
```

b. Exemplo de uso da visão:

```
select * from relatorioanual
```

c. Exemplo de retorno da função:

	ano double precision	passagens_vendidas bigint	numero_voos bigint	lucro_bruto numeric
1	2025	265	265	431595.89
2	2023	265	265	393054.11
3	2022	231	231	370120.76
4	2024	239	239	387105.73

Figura 12 - Retorno da seleção. Listagem do número de passagens vendidas, número de voos e o lucro bruto por ano.

9. Criação de uma tabela de auditoria.

Por envolver registro de valores financeiros e ser a tabela mais crítica do banco de dados, a tabela *passagem* foi escolhida para receber uma tabela de auditoria. Com ela, pode-se monitorar entradas e saídas de dados na tabela *passagem*, bem como identificar os responsáveis por elas.

- Código de criação da tabela de auditoria e da trigger, ativada a cada *inserção*, *update* ou *delete* na tabela *passagem*:

```
create table audit_passagem (operacao char, data timestamp,
usuario varchar, npassagem bigint, classe varchar(20),
npoltrona integer, preco double precision, fk_cpf
varchar(100), fk_nvoo varchar(100));
```

```
CREATE OR REPLACE FUNCTION process_passagem_audit()
RETURNS TRIGGER AS $passagem_audit$
BEGIN
    IF (TG_OP = 'DELETE') THEN
        INSERT INTO audit_passagem SELECT 'D', now(),
user, OLD.*;
        RETURN OLD;
    ELSIF (TG_OP = 'UPDATE') THEN
        INSERT INTO audit_passagem SELECT 'U', now(),
user, NEW.*;
        RETURN NEW;
    ELSIF (TG_OP = 'INSERT') THEN
        INSERT INTO audit_passagem SELECT 'I', now(),
user, NEW.*;
        RETURN NEW;
    END IF;
    RETURN NULL;
END;
$passagem_audit$ LANGUAGE plpgsql;
CREATE TRIGGER passagem_audit
AFTER INSERT OR UPDATE OR DELETE ON passagem
FOR EACH ROW EXECUTE PROCEDURE process_passagem_audit();
```

b. Exemplo de comando que ativaria a trigger da tabela de auditoria:

```
update passagem
set npoltrona=10
where fk_nvoo='06-051-0074' AND
npassagem=6463689017;
```

c. Tabela de auditoria após 6 execuções do comando acima:

	operacao character (1)	data timestamp without time zone	usuario character varying	npassagem bigint	classe character varying (20)	npoltrona integer	preco double precision	fk_cpf character varying (100)
1	U	2021-12-16 20:23:52.353602	postgres	6463689017	Primeira Classe	1	2344.52	115-51-627
2	U	2021-12-16 20:24:14.323283	postgres	6463689017	Primeira Classe	1	2344.52	115-51-627
3	U	2021-12-16 20:24:14.541324	postgres	6463689017	Primeira Classe	1	2344.52	115-51-627
4	U	2021-12-16 20:24:14.778203	postgres	6463689017	Primeira Classe	1	2344.52	115-51-627
5	U	2021-12-16 20:24:14.87298	postgres	6463689017	Primeira Classe	1	2344.52	115-51-627
6	U	2021-12-16 20:24:15.000365	postgres	6463689017	Primeira Classe	1	2344.52	115-51-627

Figura 13 - Estado da tabela de auditoria para a tabela *passagem* após esta receber 6 atualizações de campos.

10. Criação de triggers.

1. Para o primeiro gatilho criado, decidiu-se monitorar inserções e atualizações na tabela *passagem*, verificando se informações como cpf e o número do voo são válidas, bem como se o número da poltrona escolhida está livre e é também um número válido. Caso alguma dessas condições seja violada, a função do gatilho retorna uma mensagem de erro, avisando o usuário a informação que precisa ser revisada.
 - a. Código de criação da trigger, ativada antes de um *INSERT* ou *UPDATE* na tabela *passagem*:

```
CREATE OR REPLACE FUNCTION conferePassagem()  
RETURNS trigger AS $$  
DECLARE  
    voo1 VOO%ROWTYPE;  
    cap AVIAO.capacidade%TYPE;  
BEGIN  
    SELECT * FROM voo WHERE NEW.fk_nvoo=voo.nvoo INTO  
voo1;  
    -- Caso o voo inserido não exista:  
    IF(voo1 IS NULL) THEN  
        RAISE EXCEPTION 'O voo informado não  
existe!';  
    END IF;  
    IF(NOT EXISTS(SELECT 1 FROM cliente c WHERE  
NEW.fk_cpf=c.cpf)) THEN  
        RAISE EXCEPTION 'O cpf informado não pertence  
a nenhum cliente!';  
    END IF;  
  
    SELECT capacidade FROM aviao WHERE  
aviao.naviao=voo1.fk_naviao INTO cap;  
    -- Caso o número da poltrona seja maior que a  
capacidade do avião:  
    IF(NEW.npoltrona > cap) THEN  
        RAISE EXCEPTION 'Numero de poltrona  
inválido!';  
    END IF;
```



```

-- Caso o novo lugar escolhido esteja ocupado:
IF(EXISTS(SELECT 1 FROM passagem p WHERE
NEW.npoltrona=p.npoltrona AND p.fk_nvoo=NEW.fk_nvoo)
AND (TG_OP='INSERT' OR TG_OP='UPDATE' AND
OLD.npoltrona!=NEW.npoltrona)) THEN
    RAISE EXCEPTION 'Esta poltrona não pode ser
escolhida, pois já esta ocupada!';
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TrigPoltrona BEFORE INSERT OR UPDATE ON
PASSAGEM
FOR EACH ROW EXECUTE PROCEDURE conferePassagem();

```

- b. Exemplo de comando que ativaria a trigger (por conta da poltrona escolhida já estar ocupada):

```

update passagem
set npoltrona=2
where fk_nvoo='06-051-0074' AND
npassagem=6463689017;

```

- c. Mensagem de erro gerada pela trigger:

```

ERROR: Esta poltrona não pode ser escolhida, pois já esta ocupada!
CONTEXT: função PL/pgSQL conferepassagem() linha 23 em RAISE
SQL state: P0001

```

Figura 14 - Mensagem de erro gerada pela trigger ao tentar escolher uma poltrona já ocupada.

2. Para o segundo gatilho, decidiu-se criar mais uma tabela de auditoria, agora para monitorar a tabela voo, que também é uma tabela crítica (por conta de outras tabelas dependerem dela). Com essa tabela de auditoria, são registradas entradas e saídas de dados na tabela voo, bem como seus responsáveis.
 - a. Código de criação da tabela de auditoria e da trigger, ativada a cada *inserção*, *update* ou *delete* na tabela voo:

```
create table audit_voo (operacao char, data timestamp,
usuario varchar, nvoo varchar(100), companhia
varchar(100), horario date, fk_naviao integer,
fk_codaeropor varchar(100));
```

```
CREATE OR REPLACE FUNCTION process_voo_audit()
RETURNS TRIGGER AS $voo_audit$
BEGIN
    IF (TG_OP = 'DELETE') THEN
        INSERT INTO audit_voo SELECT 'D', now(),
user, OLD.*;
        RETURN OLD;
    ELSIF (TG_OP = 'UPDATE') THEN
        INSERT INTO audit_voo SELECT 'U', now(),
user, NEW.*;
        RETURN NEW;
    ELSIF (TG_OP = 'INSERT') THEN
        INSERT INTO audit_voo SELECT 'I', now(),
user, NEW.*;
        RETURN NEW;
    END IF;
    RETURN NULL;
END;
$voo_audit$ LANGUAGE plpgsql;
CREATE TRIGGER voo_audit
AFTER INSERT OR UPDATE OR DELETE ON voo
FOR EACH ROW EXECUTE PROCEDURE process_voo_audit();
```

- b. Exemplo de comando que ativaria a trigger (por alterar um dado na tabela voo):

```
update voo
set companhia='LATAM'
where nvoo='60-781-3268';
select * from audit_voo;
```

- c. Tabela de auditoria da tabela voo após 10 execuções do comando acima:

Data Output		Explain	Messages	Notifications		
	operacao character (1)	data timestamp without time zone	usuario character varying	nvoo character varying (100)	companhia character varying (100)	horar date
1	U	2021-12-17 11:26:46.845209	postgres	60-781-3268	LATAM	2025-
2	U	2021-12-17 11:26:47.406982	postgres	60-781-3268	LATAM	2025-
3	U	2021-12-17 11:26:47.592092	postgres	60-781-3268	LATAM	2025-
4	U	2021-12-17 11:26:47.703437	postgres	60-781-3268	LATAM	2025-
5	U	2021-12-17 11:26:47.822871	postgres	60-781-3268	LATAM	2025-
6	U	2021-12-17 11:26:47.945201	postgres	60-781-3268	LATAM	2025-
7	U	2021-12-17 11:26:48.084561	postgres	60-781-3268	LATAM	2025-
8	U	2021-12-17 11:26:48.31963	postgres	60-781-3268	LATAM	2025-
9	U	2021-12-17 11:26:48.440292	postgres	60-781-3268	LATAM	2025-
10	U	2021-12-17 11:26:48.547959	postgres	60-781-3268	LATAM	2025-

3. Para o terceiro gatilho, decidiu-se criar uma função que verifique o nome do cliente, antes de uma inserção ou atualização de tuple. Com essa função, deseja-se assegurar-se que os nomes contenham apenas letras (possuindo acento ou não) ou símbolos como apóstrofo e hífen.
 - a. Código de criação da trigger, ativada antes de cada *inserção* ou *update* na tabela *cliente*:

- b. Exemplo de atualização de tupla que ativaria a trigger:

```
update cliente
set nome='J04051NH0'
where cpf='245-24-2123';
```

c. Mensagem retornada pela trigger:

Data Output	Explain	Messages	Notifications
ERROR: O nome não deve conter números e símbolos desnecessários! CONTEXT: função PL/pgSQL verificanomecliente() linha 6 em RAISE SQL state: P0001			

Figura 16 -.Mensagem de erro gerada pela trigger ao tentar inserir números ou símbolos especiais.

ANEXO - Links para os arquivos csv e sql

Os [comandos sql](#) utilizados na construção do banco, bem como os [dados utilizados para a população das tabelas](#), em csv, estão disponíveis no repositório do GitHub:

<https://github.com/GJunges1/APS-Banco-2.git>