Universidade Federal de Uberlândia

Felipe Alves Belisário
Breno Caldeira Nascimento
Adiel Pereira Prado

Projeto

Pizzaria com Entretenimento

Curso: Bacharelado em Ciência da Computação
Disciplina: Sistema de Banco de Dados (GBC043)
Professora: Profa. Maria Camila Nardini Barioni
Data de entrega: 28/06/2019

ÍNDICE

1 – Especificação do Problema / Esquema conceitual

A) Os tópicos a seguir descrevem o funcionamento do banco de dados utilizado no aplicativo.

B) Os usuários são identificados de forma única. Logo, usamos o CPF deste como chave identificadora. Também utilizamos data de registro, nome e endereço para armazenar no sistema.

C) Uma das opções que o usuário possui ao se cadastrar é de ser um Dono de Negócio. Para essa disjunção, registramos sua conta do LinkedIn.

D) Um dono de negócio pode possuir várias pizzarias, porém, uma pizzaria terá apenas um dono. Modelamos essa situação com uma cardinalidade 1 para n, entre “Dono de negócio”, “Possui” e “Pizzaria”.

E) A entidade “Pizzaria” possui os seguintes atributos: CEP, endereço, número de telefone, web site, e horários de funcionamento e nome. Criamos este último atributo e o usamos como chave, visto que é único para cada estabelecimento.

F) Criamos uma entidade fraca “Pizza”, cujos atributos são: nome e preço, onde aquele é uma chave fraca. A entidade é fraca pois duas pizzas podem possuir o mesmo sabor e o mesmo preço, porém, vindas de estabelecimentos diferentes, tornando-as únicas para cada pizzaria.

G) Uma pizza pertence a uma categoria, porém uma categoria possui várias pizzas. Logo, justifica-se a cardinalidade 1 para n, no relacionamento “Pertence” entre “Pizza” e “Categoria”. Este é uma entidade com os seguintes atributos: Descrição, e Código, usado como chave. Além disso, essa recém criada entidade “Categoria” faz uma relação consigo mesma já que dentro de uma categoria podem haver várias subcategorias e uma subcategoria pode estar contida em uma categoria.

H) Criamos uma entidade fraca “Acompanhamento”, que depende de uma pizzaria, seus atributos são: nome, descrição, tipo de acompanhamento e preço. O código é usado como chave para distingui-lo dentro de uma única pizzaria.

I) Como citado no item b), outra opção para o usuário é de ser um “Consumidor Faminto”, uma entidade que possui endereço de entrega como atributo.

J) Criamos uma entidade “Pedido”, com um atributo ID, usado como chave. A relação “Realiza”, entre “Pedido” e “Consumidor Faminto” possui data e horário como atributos. Adicionamos “Horário Posterior” e “Quantidade de Pessoas” como atributos de “Pedido”, para a escolha pessoal do consumidor.

K) Criamos uma relação “Possui”, entre “Pedido” e “Pizza”, com os seguintes atributos: massa, quantidade de molho e borda. Sua cardinalidade é n para m, visto que um pedido pode conter várias pizzas, e várias pizzas podem pertencer a um pedido. Os atributos estão na relação pois eles “modificam” uma pizza padrão já existente, para incluir na realização do pedido.

L) Criamos a relação “Possui” entre “Pedido” e “Acompanhamento”. Ela possui quantidade como atributo. A cardinalidade e o atributo dela possui a mesma justificativa do item anterior.

M) Fizemos uma agregação envolvendo a relação “Possui”, que se encontra entre “Pedido” e “Pizza”, nomeando-a de “Pizza_Pedido”, a mesma deve existir pois a entidade criada “Ingrediente Extra” se trata de apenas um adicional (opcional) para a pizza. Um ingrediente extra pode estar contido a vários pedidos de pizzas e um pedido de pizzas pode conter vários ingredientes extras.

N) Um “Ingrediente Extra” possui um código, nome e preço como atributos. O primeiro é tratado como chave. A existência desses atributos provam este ser uma entidade.

O) A entidade “Pedido com Entretenimento” é uma disjunção de “Pedido”, visto que herda os mesmos atributos. Além disso, possui tipo e duração como atributos.

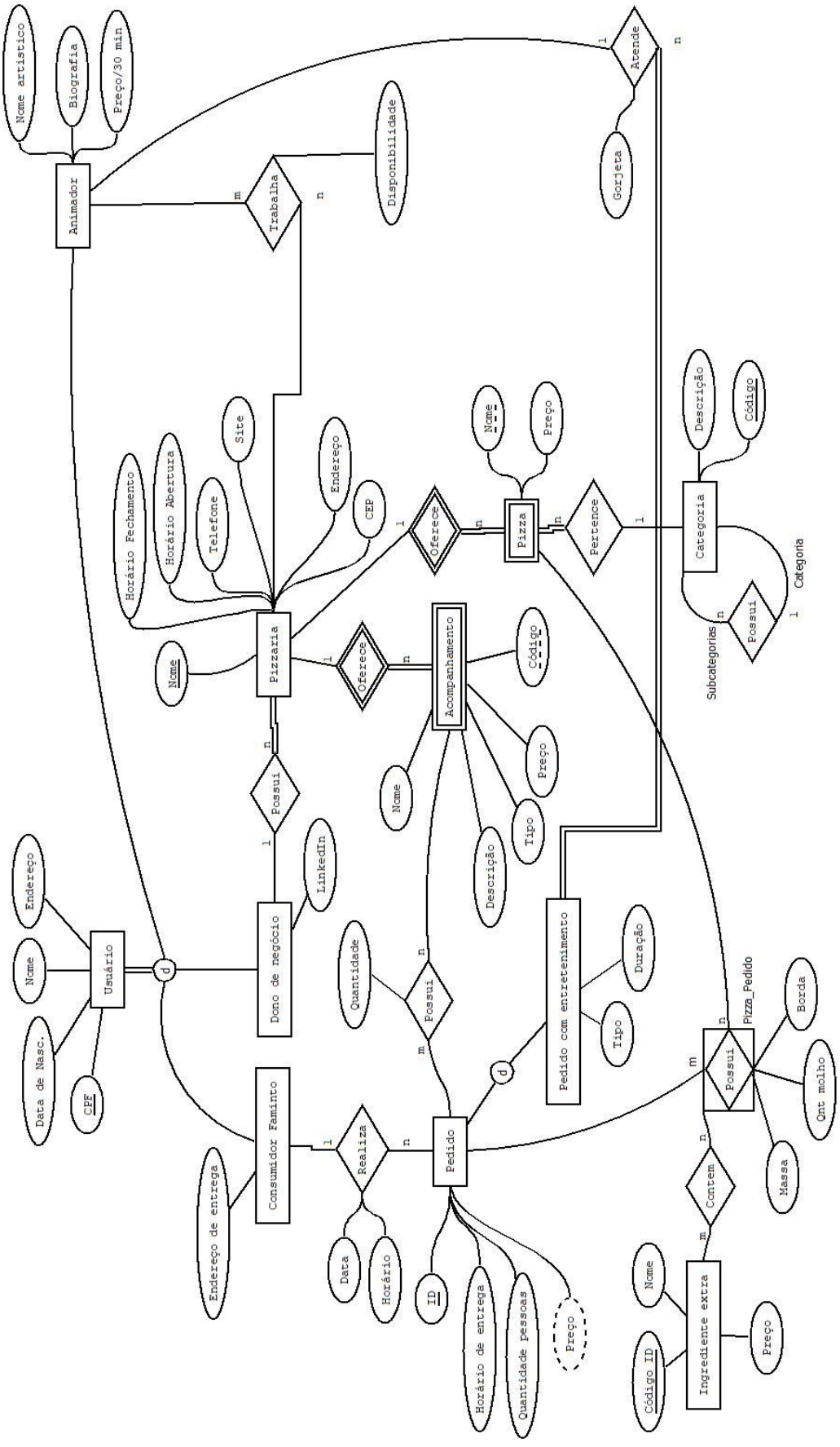
P) Como o preço é calculado a partir das pizzas escolhidas, ingredientes extras, acompanhamentos e escolha de entretenimento, ele torna-se um atributo derivado de “Pedido”.

Q) Um usuário, ao se cadastrar no sistema, pode ser um animador. Para isso, criamos uma disjunção de “Usuário” chamada “Animador”. Ela possui nome artístico, biografia e preço, para 30 minutos de entretenimento, como atributos.

R) Um “Pedido com Entretenimento” possui apenas um animador. Logo, justifica-se a cardinalidade 1 para n da relação “Atende” entre “Animador” e “Pedido com entretenimento”.

S) Criamos a relação “Trabalha” entre “Animador” e “Pizzaria”. Nela, o animador especifica os dias da semana que ele está disponível, logo, a disponibilidade dele se tornará um atributo da relação “Trabalha”.

Extra) Criamos um atributo “Gorjeta” que faz parte da relação “Atende” entre “Pedido com entretenimento” e “Animador” com o intuito de os animadores poderem receber um valor a mais sobre seus serviços vindos do consumidor faminto. E o outro requisito adicional que adicionamos é colocar um atributo chave “Nome” para cada pizzaria para que elas possam ser identificadas pelo seu nome.

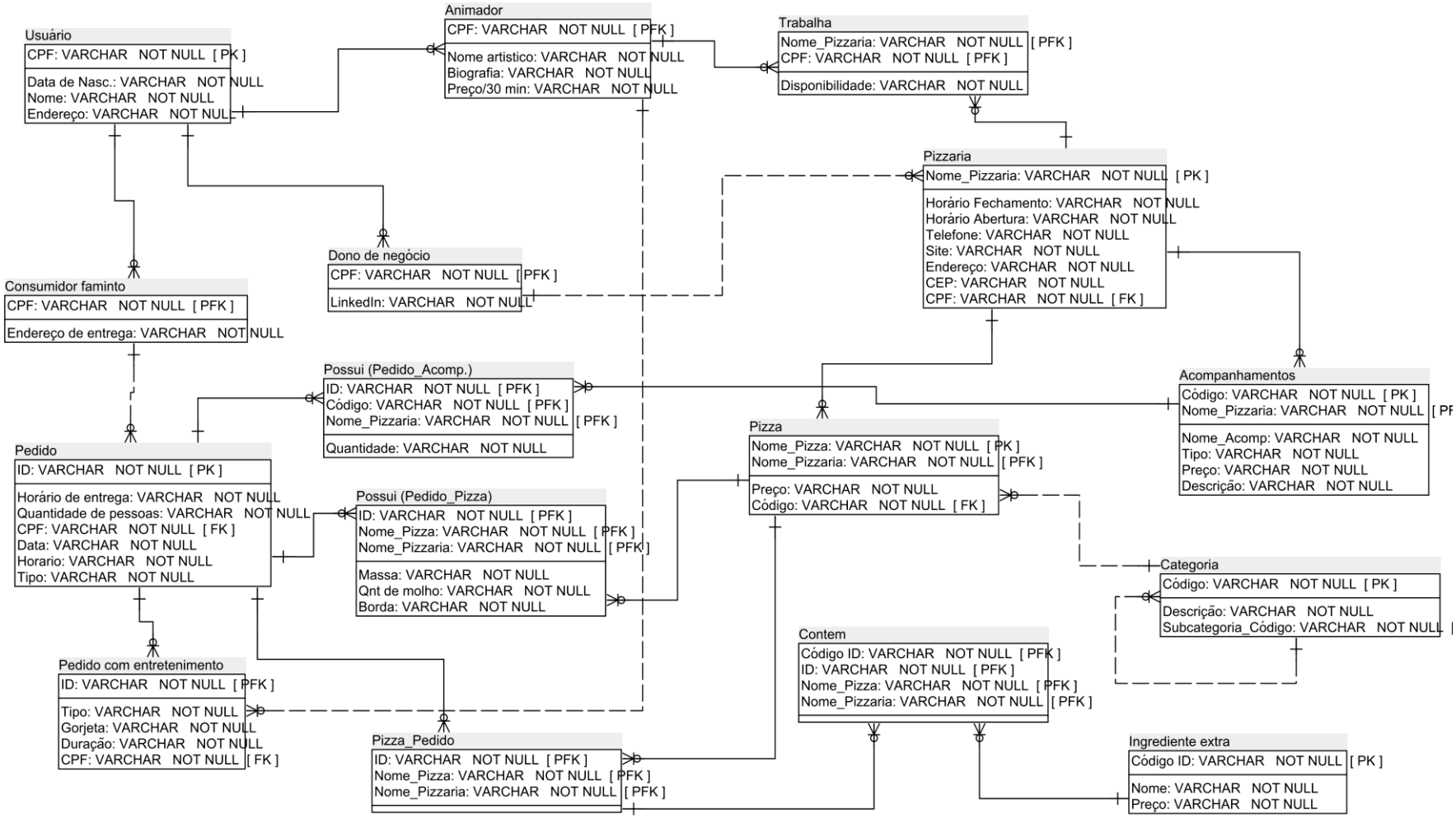

Nome artístico
Biografia
Preço/30 min
Animador
Trabalha
Disponibilidade
Atende
Gorjeta
Horário Fechamento
Horário Abertura
Telefone
Site
Endereço
CEP
Nome
Pizzaria
Ofereço
Nome
Preço
Pizza
Pertence
Descrição
Código
Categoria
Subcategorias
Possui
Categoria
Nome
Ofereço
Acompanhamento
Código
Preço
Nome
Descrição
Tipo
Possui
Endereço
Nome
Data de Nasc.
CPF
Usuário
d
Dono de negócio
LinkedIn
Quantidade
Possui
Pedido com entretenimento
Duração
Tipo
Endereço de entrega
Consumidor Faminto
Realiza
Pedido
Data
Horário
ID
Horário de entrega
Quantidade pessoas
Preço
Pizza_Pedido
Borda
Qnt molho
Massa
Possui
Contem
Código ID
Nome
Ingrediente extra
Preço

2 – Esquema Relacional

Para mapear o modelo ER feito na tarefa 1 seguimos o passo a passo e mapeamos primeiramente as entidades fortes e depois as fracas (Acompanhamento e Pizza) que não são subclasses, as quais só se foram passadas uma forma de serem implementadas em sala de aula. Logo após isso, foram mapeadas todas as relações 1:n, já que não há nenhuma de 1:1, (Realiza entre Consumidor Faminto e Pedido, Possui entre Dono de Negócio e Pizzaria, Atende entre Animador e Pedido com Entretenimento, Possui entre Categoria e ela mesma, Pertence entre Pizza e Categoria) utilizando o método no qual a entidade com a cardinalidade n recebe a chave estrangeira, que não será chave primária da mesma, vinda da entidade de cardinalidade 1. Depois disso vieram os relacionamentos n:m, menos o Contem entre Ingrediente e a agregação, (Possui entre Pedido e Acompanhamento, Trabalha entre Animador e Pizzaria, Possui entre Pedido e Pizza), onde foi criada uma tabela para cada um deles e neles foram recebidas as chaves estrangeiras vindas das duas entidades as quais faz parte, e estas se tornam também suas chaves primárias.

Após, foi a vez das subclasses, em que foi utilizado o método 8A devido à grande quantidade de atributos das superclasses, então as disjunções de Usuário foram separadas em 4 tabelas distintas e cada uma herdou a chave primária de Usuário. Por último foi mapeada a agregação Pizza_Pedido a qual se tornou uma tabela separada da do relacionamento Possui entre Pedido e Pizza pois os atributos desta última não podem ser repassados para a agregação já que a função dela é apenas tornar os ingredientes extras da pizza opcionais, e junto disso possibilitando finalmente o mapeamento da relação n:m Contem entre Ingrediente Extra e Pizza_Pedido que não foi implementada antes devido a inexistência da tabela da agregação.

OBS: Atributos derivados não são mapeados


Usuário
CPF: VARCHAR NOT NULL [PK]
Data de Nasc.: VARCHAR NOT NULL
Nome: VARCHAR NOT NULL
Endereço: VARCHAR NOT NULL
Animador
CPF: VARCHAR NOT NULL [PFK]
Nome artistico: VARCHAR NOT NULL
Biografia: VARCHAR NOT NULL
Preço/30 min: VARCHAR NOT NULL
Trabalha
Nome_Pizzaria: VARCHAR NOT NULL [PFK]
CPF: VARCHAR NOT NULL [PFK]
Disponibilidade: VARCHAR NOT NULL
Pizzaria
Nome_Pizzaria: VARCHAR NOT NULL [PK]
Horário Fechamento: VARCHAR NOT NULL
Horário Abertura: VARCHAR NOT NULL
Telefone: VARCHAR NOT NULL
Site: VARCHAR NOT NULL
Endereço: VARCHAR NOT NULL
CEP: VARCHAR NOT NULL
CPF: VARCHAR NOT NULL [FK]
Dono de negócio
CPF: VARCHAR NOT NULL [PFK]
LinkedIn: VARCHAR NOT NULL
Consumidor faminto
CPF: VARCHAR NOT NULL [PFK]
Endereço de entrega: VARCHAR NOT NULL
Possui (Pedido_Acomp.)
ID: VARCHAR NOT NULL [PFK]
Código: VARCHAR NOT NULL [PFK]
Nome_Pizzaria: VARCHAR NOT NULL [PFK]
Quantidade: VARCHAR NOT NULL
Acompanhamentos
Código: VARCHAR NOT NULL [PK]
Nome_Pizzaria: VARCHAR NOT NULL [PF
Nome_Acomp: VARCHAR NOT NULL
Tipo: VARCHAR NOT NULL
Preço: VARCHAR NOT NULL
Descrição: VARCHAR NOT NULL
Pizza
Nome_Pizza: VARCHAR NOT NULL [PK]
Nome_Pizzaria: VARCHAR NOT NULL [PFK]
Preço: VARCHAR NOT NULL
Código: VARCHAR NOT NULL [FK]
Pedido
ID: VARCHAR NOT NULL [PK]
Horário de entrega: VARCHAR NOT NULL
Quantidade de pessoas: VARCHAR NOT NULL
CPF: VARCHAR NOT NULL [FK]
Data: VARCHAR NOT NULL
Horario: VARCHAR NOT NULL
Tipo: VARCHAR NOT NULL
Possui (Pedido_Pizza)
ID: VARCHAR NOT NULL [PFK]
Nome_Pizza: VARCHAR NOT NULL [PFK]
Nome_Pizzaria: VARCHAR NOT NULL [PFK]
Massa: VARCHAR NOT NULL
Qnt de molho: VARCHAR NOT NULL
Borda: VARCHAR NOT NULL
Categoria
Código: VARCHAR NOT NULL [PK]
Descrição: VARCHAR NOT NULL
Subcategoria_Código: VARCHAR NOT NULL
Contem
Código ID: VARCHAR NOT NULL [PFK]
ID: VARCHAR NOT NULL [PFK]
Nome_Pizza: VARCHAR NOT NULL [PFK]
Nome_Pizzaria: VARCHAR NOT NULL [PFK]
Pedido com entretenimento
ID: VARCHAR NOT NULL [PFK]
Tipo: VARCHAR NOT NULL
Gorjeta: VARCHAR NOT NULL
Duração: VARCHAR NOT NULL
CPF: VARCHAR NOT NULL [FK]
Pizza_Pedido
ID: VARCHAR NOT NULL [PFK]
Nome_Pizza: VARCHAR NOT NULL [PFK]
Nome_Pizzaria: VARCHAR NOT NULL [PFK]
Ingrediente extra
Código ID: VARCHAR NOT NULL [PK]
Nome: VARCHAR NOT NULL
Preço: VARCHAR NOT NULL

3 – Criação Banco de Dados

Nessa etapa o projeto desenvolvido durante todo o semestre foi concluído, todo o código feito no pgAdmin está abaixo com todas as explicações sobre o mesmo em forma de comentários ao decorrer dele.

```
DROP SCHEMA pizzaria_entretenimento CASCADE;

CREATE SCHEMA pizzaria_entretenimento;

SET search_path TO pizzaria_entretenimento;

SET datestyle TO 'DMY';


CREATE TABLE USUARIO(

	cpf BIGINT NOT NULL,

	nome_usuario VARCHAR(50) NOT NULL,

	endereco VARCHAR(50) NOT NULL,

	data_nasc DATE,


	CONSTRAINT PK_USUARIO PRIMARY KEY (cpf)

);


CREATE TABLE ANIMADOR(

	cpf BIGINT NOT NULL,

	nome_art VARCHAR(50) NOT NULL,

	preco REAL NOT NULL,
```

```
    bio VARCHAR(150),

    CONSTRAINT PK_ANIMADOR PRIMARY KEY (cpf),
    CONSTRAINT FK_ANIMADOR FOREIGN KEY (cpf) REFERENCES USUARIO(cpf)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE CONSUMIDOR_FAMINTO(
    cpf BIGINT NOT NULL,
    end_entrega VARCHAR(50) NOT NULL,

    CONSTRAINT PK_CONSUMIDOR PRIMARY KEY (cpf),
    CONSTRAINT FK_CONSUMIDOR FOREIGN KEY (cpf) REFERENCES
USUARIO(cpf)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE DONO_DE_NEGOCIO(
    cpf BIGINT NOT NULL,
    linkedin VARCHAR(50),
```

```
        CONSTRAINT PK_DONO PRIMARY KEY (cpf),
        CONSTRAINT FK_DONO FOREIGN KEY (cpf) REFERENCES USUARIO(cpf)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

CREATE TABLE PEDIDO(
        id_pedido INTEGER NOT NULL,
        cpf_consumidor BIGINT NOT NULL,
        horario_entrega TIME,
        horario_realizado TIME,
        qnt_pessoas INTEGER,
        data_realizado DATE,
        tipo VARCHAR(50) NOT NULL,
      total REAL,

        CONSTRAINT PK_PEDIDO PRIMARY KEY (id_pedido),
        CONSTRAINT FK_PEDIDO FOREIGN KEY (cpf_consumidor) REFERENCES
CONSUMIDOR_FAMINTO(cpf)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);
```

```
CREATE TABLE PEDIDO_ENTRETENIMENTO(
        id_pedido INTEGER NOT NULL,
        cpf_animador BIGINT NOT NULL,
        duracao TIME NOT NULL,
        gorjeta REAL,
        tipo VARCHAR(50) NOT NULL,

        CONSTRAINT PK_PEDIDO_ENT PRIMARY KEY (id_pedido),
        CONSTRAINT FK_PEDIDO_ENT FOREIGN KEY (id_pedido) REFERENCES
PEDIDO(id_pedido)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

CREATE TABLE PIZZARIA(
        nome_pizzaria VARCHAR(50) NOT NULL,
        cpf_dono BIGINT NOT NULL,
        horario_fechamento TIME NOT NULL,
        horario_abertura TIME NOT NULL,
        telefone VARCHAR(50) NOT NULL,
        site VARCHAR(50),
        endereco VARCHAR(50) NOT NULL,

        cep VARCHAR(50),
```

```
	CONSTRAINT PK_PIZZARIA PRIMARY KEY (nome_pizzaria),

	CONSTRAINT FK_PIZZARIA FOREIGN KEY (cpf_dono) REFERENCES
DONO_DE_NEGOCIO(cpf)

		ON UPDATE CASCADE

		ON DELETE CASCADE

);


CREATE TABLE CATEGORIA(

	codigo INTEGER NOT NULL,

	descricao VARCHAR(150),

	subcategoria VARCHAR(50) NOT NULL,


	CONSTRAINT PK_CATEGORIA PRIMARY KEY (codigo)

);


CREATE TABLE PIZZA(

	nome_pizza VARCHAR(150) NOT NULL,

	nome_pizzaria VARCHAR(50) NOT NULL,

	preco REAL NOT NULL,

	codigo INTEGER NOT NULL,


	CONSTRAINT PK_PIZZA PRIMARY KEY (nome_pizza,nome_pizzaria),

	CONSTRAINT FK_PIZZA FOREIGN KEY (nome_pizzaria) REFERENCES
PIZZARIA(nome_pizzaria)
```

```
                ON UPDATE CASCADE
                ON DELETE CASCADE,
                FOREIGN KEY (codigo) REFERENCES CATEGORIA(codigo)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

CREATE TABLE ACOMPANHAMENTOS(
        codigo_acomp INTEGER NOT NULL,
        nome_pizzaria VARCHAR(50) NOT NULL,
        nome_acompanhamento VARCHAR(50) NOT NULL,
        preco REAL NOT NULL,
        tipo VARCHAR(50),
        descricao VARCHAR(150),

        CONSTRAINT PK_ACOMPANHAMENTOS PRIMARY KEY
(codigo_acomp,nome_pizzaria),
        CONSTRAINT FK_ACOMPANHAMENTOS FOREIGN KEY (nome_pizzaria)
REFERENCES PIZZARIA(nome_pizzaria)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);
```

```
CREATE TABLE INGREDIENTE_EXTRA(
        cod_id INTEGER NOT NULL,
        nome_ingrediente VARCHAR(50) NOT NULL,
        preco REAL NOT NULL,

        CONSTRAINT PK_INGREDIENTE PRIMARY KEY (cod_id)

);

CREATE TABLE TRABALHA(
        nome_pizzaria VARCHAR(50) NOT NULL,
        cpf_animador BIGINT NOT NULL,
        disponibilidade VARCHAR(50) NOT NULL,

        CONSTRAINT PK_TRABALHA PRIMARY KEY (nome_pizzaria,cpf_animador),
        CONSTRAINT FK_TRABALHA FOREIGN KEY (nome_pizzaria) REFERENCES
PIZZARIA
                ON UPDATE CASCADE
                ON DELETE CASCADE,
                FOREIGN KEY (cpf_animador) REFERENCES ANIMADOR(cpf)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);
```

```
CREATE TABLE POSSUI_PEDIDO_ACOMP(
        id_pedido INTEGER NOT NULL,
        codigo_acomp INTEGER NOT NULL,
        nome_pizzaria VARCHAR(50) NOT NULL,
        quantidade INTEGER NOT NULL,

        CONSTRAINT PK_POSSUI_PEDIDO_ACOMP PRIMARY KEY
(nome_pizzaria,id_pedido,codigo_acomp),
        CONSTRAINT FK_POSSUI_PEDIDO_ACOMP FOREIGN KEY (codigo_acomp,
nome_pizzaria) REFERENCES ACOMPANHAMENTOS(codigo_acomp, nome_pizzaria)
                ON UPDATE CASCADE
                ON DELETE CASCADE,
                FOREIGN KEY (id_pedido) REFERENCES PEDIDO(id_pedido)
                ON UPDATE CASCADE
                ON DELETE CASCADE
);

CREATE TABLE POSSUI_PEDIDO_PIZZA(
        id_pedido INTEGER NOT NULL,
        nome_pizza VARCHAR(50) NOT NULL,
        nome_pizzaria VARCHAR(50) NOT NULL,
        massa VARCHAR(50) NOT NULL,
        qnt_molho VARCHAR(50) NOT NULL,
```

```
	borda VARCHAR(50) NOT NULL,


	CONSTRAINT PK_POSSUI_PEDIDO_PIZZA PRIMARY KEY
(nome_pizzaria,id_pedido,nome_pizza),
	CONSTRAINT FK_POSSUI_PEDIDO_PIZZA FOREIGN KEY (nome_pizza,
nome_pizzaria) REFERENCES PIZZA(nome_pizza, nome_pizzaria)
		ON UPDATE CASCADE
		ON DELETE CASCADE,
		FOREIGN KEY (id_pedido) REFERENCES PEDIDO(id_pedido)
		ON UPDATE CASCADE
		ON DELETE CASCADE
);


CREATE TABLE PIZZA_PEDIDO(
	id_pedido INTEGER NOT NULL,
	nome_pizza VARCHAR(50) NOT NULL,
	nome_pizzaria VARCHAR(50) NOT NULL,


	CONSTRAINT PK_PIZZA_PEDIDO PRIMARY KEY
(nome_pizzaria,id_pedido,nome_pizza),
	CONSTRAINT FK_PIZZA_PEDIDO FOREIGN KEY (nome_pizza, nome_pizzaria)
REFERENCES PIZZA(nome_pizza, nome_pizzaria)
		ON UPDATE CASCADE
		ON DELETE CASCADE,
		FOREIGN KEY (id_pedido) REFERENCES PEDIDO
```

```
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE CONTEM(
    cod_id INTEGER NOT NULL,
    id_pedido INTEGER NOT NULL,
    nome_pizza VARCHAR(50) NOT NULL,
    nome_pizzaria VARCHAR(50) NOT NULL,

    CONSTRAINT PK_CONTEM PRIMARY KEY
(cod_id,nome_pizzaria,id_pedido,nome_pizza),
    CONSTRAINT FK_CONTEM FOREIGN KEY (nome_pizza, nome_pizzaria)
REFERENCES PIZZA(nome_pizza, nome_pizzaria)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
        FOREIGN KEY (id_pedido) REFERENCES PEDIDO(id_pedido)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
        FOREIGN KEY (cod_id) REFERENCES INGREDIENTE_EXTRA(cod_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
```

-- OBS: O procedimento armazenado e um dos triggers estão no meio das inserções devido à necessidade dos mesmos

-- serem executados antes dessas inserções para se tornarem válidos (INSERÇÕES DE POSSUI_PEDIDO_PIZZA, POSSUI_PEDIDO_ACOMP, CONTEM E PEDIDO_ENTRETENIMENTO DEVEM VIR APÓS ESSES TRIGGERS):

-- Função que converte tipo TIME para INTEGER (segundos):

```
CREATE OR REPLACE FUNCTION to_seconds(tempo TIME)
  RETURNS REAL AS $$
DECLARE
    hs INTEGER;
    ms INTEGER;
    s INTEGER;
          resultado REAL;
BEGIN
    SELECT (EXTRACT (HOUR FROM tempo) * 60*60) INTO hs;
    SELECT (EXTRACT (MINUTES FROM tempo) * 60) INTO ms;
    SELECT (EXTRACT (SECONDS FROM tempo)) INTO s;
    SELECT (hs + ms + s)::real INTO resultado;
    RETURN resultado;
END $$ LANGUAGE 'plpgsql';
```

-- Função que dado o nome do animador calcula o total que o mesmo recebeu dentre todos os pedidos que participou:

```
CREATE OR REPLACE FUNCTION calcula_dinheiro_animador(
IN nome_animador ANIMADOR.nome_art%TYPE,
OUT valor_recebido ANIMADOR.preco%TYPE
) AS $$
DECLARE tempo_duracao TIME;
DECLARE gorjeta_animador REAL;
BEGIN

        SELECT SUM(pe.duracao), SUM(pe.gorjeta)
        INTO  tempo_duracao, gorjeta_animador
        FROM ANIMADOR ani, PEDIDO_ENTRETENIMENTO pe
        WHERE nome_art = nome_animador and ani.cpf = pe.cpf_animador
        GROUP BY pe.cpf_animador;

        SELECT preco
        INTO  valor_recebido
        FROM ANIMADOR
        WHERE nome_art = nome_animador;

        valor_recebido := (((to_seconds(tempo_duracao) * valor_recebido) /
to_seconds('00:30:00')) + gorjeta_animador);

END $$ LANGUAGE 'plpgsql';
```

```
-- Testes para as funções:


SELECT to_seconds('01:40:50');

SELECT calcula_dinheiro_animador('Ricardinho Milos');


-- Conjunto de triggers para calcular preço total de cada pedido:


-- Trigger responsavel pelo preço total das pizzas:


CREATE OR REPLACE FUNCTION calcula_total_pizzas()

RETURNS TRIGGER AS $$

BEGIN


                UPDATE PEDIDO

                SET total = total + (    SELECT preco

                                                FROM POSSUI_PEDIDO_PIZZA
pp, PIZZA pi

                                                WHERE pp.nome_pizza =
new.nome_pizza and pp.nome_pizzaria = pi.nome_pizzaria

                                                        and pp.nome_pizzaria =
new.nome_pizzaria

                                                        and pp.id_pedido =
new.id_pedido and pi.nome_pizza = pp.nome_pizza

                                                )

                WHERE id_pedido = new.id_pedido;
```

```
        RETURN NULL;

END $$ LANGUAGE 'plpgsql';

CREATE TRIGGER calcula_total_pizzas_t
AFTER INSERT ON POSSUI_PEDIDO_PIZZA
FOR EACH ROW
EXECUTE PROCEDURE calcula_total_pizzas();

-- Trigger responsavel pelo preço total dos acompanhamentos:

CREATE OR REPLACE FUNCTION calcula_total_acomp()
RETURNS TRIGGER AS $$
BEGIN

        UPDATE PEDIDO
        SET total = total + (    SELECT preco
                                 FROM
POSSUI_PEDIDO_ACOMP pa, ACOMPANHAMENTOS ac
                                 WHERE pa.codigo_acomp =
new.codigo_acomp and pa.codigo_acomp = ac.codigo_acomp
```

```
and pa.id_pedido = new.id_pedido and pa.nome_pizzaria = new.nome_pizzaria
                                                        and pa.nome_pizzaria =
ac.nome_pizzaria
                                                    ) * new.quantidade
            WHERE id_pedido = new.id_pedido;

            RETURN NULL;

END $$ LANGUAGE 'plpgsql';

CREATE TRIGGER calcula_total_acomp_t
AFTER INSERT ON POSSUI_PEDIDO_ACOMP
FOR EACH ROW
EXECUTE PROCEDURE calcula_total_acomp();

-- Trigger responsavel pelo preço total dos ingredientes extras:

CREATE OR REPLACE FUNCTION calcula_total_ingr_extra()
RETURNS TRIGGER AS $$
DECLARE add_preco REAL;
BEGIN

            SELECT preco
```

```
        INTO add_preco
        FROM CONTEM cn, INGREDIENTE_EXTRA ie, POSSUI_PEDIDO_PIZZA pe
        WHERE cn.cod_id = ie.cod_id and cn.cod_id = new.cod_id
        and pe.id_pedido = cn.id_pedido and pe.id_pedido = new.id_pedido;

        UPDATE PEDIDO
        SET total = total + add_preco
        WHERE id_pedido = new.id_pedido;

        RETURN NULL;

END $$ LANGUAGE 'plpgsql';

CREATE TRIGGER calcula_total_ingr_extra_t
AFTER INSERT ON CONTEM
FOR EACH ROW
EXECUTE PROCEDURE calcula_total_ingr_extra();

-- Trigger responsavel pelo preço total pago para o animador:

CREATE OR REPLACE FUNCTION calcula_total_animador_preco()
RETURNS TRIGGER AS $$
```

```
DECLARE nome VARCHAR(50);

BEGIN

        SELECT nome_art

        INTO nome

        FROM ANIMADOR ani, PEDIDO_ENTRETENIMENTO pe, PEDIDO ped

        WHERE pe.id_pedido = ped.id_pedido and pe.id_pedido = new.id_pedido

                and pe.cpf_animador = ani.cpf and pe.cpf_animador =
new.cpf_animador;



        UPDATE PEDIDO

        SET total = total + calcula_dinheiro_animador(nome)

        WHERE id_pedido = new.id_pedido;



        RETURN NULL;


END $$ LANGUAGE 'plpgsql';


CREATE TRIGGER calcula_total_animador_preco_t

AFTER INSERT ON PEDIDO_ENTRETENIMENTO

FOR EACH ROW
```

```
EXECUTE PROCEDURE calcula_total_animador_preco();


-- INÍCIO DAS CONSULTAS!


-- Calcula a media dos preços das pizzas de cada pizzaria e mostra qual a menor:


SELECT pi.nome_pizzaria, AVG(pi.preco)
FROM PIZZA pi, PIZZARIA pi2
WHERE pi.nome_pizzaria = pi2.nome_pizzaria
GROUP BY pi.nome_pizzaria
HAVING AVG(pi.preco) <= ALL(SELECT AVG(pi.preco)
                            FROM PIZZA pi, PIZZARIA pi2
                            WHERE pi.nome_pizzaria = pi2.nome_pizzaria
                            GROUP BY pi.nome_pizzaria
                            );


-- Mostra nome e quantidade de pedidos da(s) pizzaria(s) que possuem maior número de pedidos:


SELECT nome_pizzaria, COUNT(id_pedido)
FROM PIZZA_PEDIDO
GROUP BY nome_pizzaria
HAVING COUNT(id_pedido) >= ALL(SELECT COUNT(id_pedido)
```

```
                                        FROM PIZZA_PEDIDO
                                        GROUP BY nome_pizzaria);

-- Mostra o nome de cada acompanhamento existente junto com sua quantidade total individual
-- e a quantidade total de acompanhamentos (ambos dentre todos os pedidos):

SELECT ac.nome_acompanhamento, SUM(pac.quantidade)
FROM ACOMPANHAMENTOS ac INNER JOIN POSSUI_PEDIDO_ACOMP pac ON
        pac.codigo_acomp = ac.codigo_acomp and pac.nome_pizzaria = ac.nome_pizzaria
GROUP BY ROLLUP(ac.nome_acompanhamento);

-- Mostra o id do pedido que demorou menos tempo para ser entregue juntamente com esse tempo:

SELECT id_pedido, horario_entrega - horario_realizado
FROM PEDIDO
WHERE horario_entrega - horario_realizado <= ALL(SELECT horario_entrega - horario_realizado
        FROM PEDIDO
                                                );
```

-- Mostra a(s) pizza(s) mais vendida(s) de cada pizzaria, seu preço e o nome dessas pizzarias:

```
SELECT F2.nome_pizzaria, MAX(F2.nome_pizza),F2.preco
FROM (
	SELECT COUNT(PIZZA.nome_pizza), PIZZA.nome_pizzaria, PIZZA.nome_pizza,PIZZA.preco
	FROM  PIZZA_PEDIDO, POSSUI_PEDIDO_PIZZA, PEDIDO, PIZZA
	WHERE POSSUI_PEDIDO_PIZZA.id_pedido = PIZZA_PEDIDO.id_pedido AND
		PEDIDO.id_pedido = PIZZA_PEDIDO.id_pedido AND
		PIZZA.nome_pizza = POSSUI_PEDIDO_PIZZA.nome_pizza and
		Pizza.nome_pizzaria = pizza_pedido.nome_pizzaria
	GROUP BY PIZZA.nome_pizza,PIZZA.preco, PIZZA.nome_pizzaria
) AS F2
group by F2.nome_pizzaria, F2.preco;
```

-- Mostra o animador que mais recebeu dentre todos (utilizando função criada):

```
SELECT nome_art, calcula_dinheiro_animador(nome_art)
FROM ANIMADOR
WHERE calcula_dinheiro_animador(nome_art) >= ALL(SELECT calcula_dinheiro_animador(nome_art)
```

```
FROM ANIMADOR

    );


-- Mostra os id's de todos os pedidos junto com a quantidade de pessoas de cada e por fim
-- a quantia de dinheiro que ficaria para cada pessoa caso forem dividir o preço total das pizzas


SELECT ped1.id_pedido, ped2.qnt_pessoas, SUM(preco) / qnt_pessoas
FROM POSSUI_PEDIDO_PIZZA ped1, PEDIDO ped2, PIZZA pi
WHERE ped1.id_pedido = ped2.id_pedido and ped1.nome_pizza = pi.nome_pizza and
        ped1.nome_pizzaria = pi.nome_pizzaria
GROUP BY ped1.id_pedido, qnt_pessoas;


-- Seleciona o animador mais contratado de cada pizzaria:


SELECT MAX(F1.COUNT), F1.nome_pizzaria, F1.nome_art
FROM POSSUI_PEDIDO_PIZZA p3, Animador a2, (
        SELECT COUNT(nome_art), p1.nome_pizzaria, nome_art, p1.id_pedido, cpf
        FROM POSSUI_PEDIDO_PIZZA p1, PEDIDO_ENTRETENIMENTO p2, ANIMADOR a1
        WHERE p1.id_pedido = p2.id_pedido AND
```

```
            a1.cpf = p2.cpf_animador
        GROUP BY p1.nome_pizzaria, nome_art, p1.id_pedido, cpf
) AS F1
WHERE p3.id_pedido = F1.id_pedido AND
    a2.cpf = F1.cpf
GROUP BY F1.nome_pizzaria, F1.nome_art;

-- Mostra qual(is) ingrediente(s) esta presente em cada pedido:

SELECT nome_ingrediente, cn.id_pedido
FROM INGREDIENTE_EXTRA ie, CONTEM cn
WHERE cn.cod_id = ie.cod_id
GROUP BY nome_ingrediente, cn.id_pedido;

-- Mostra todos os animadores que trabalham nos fins de semana:

SELECT nome_art
FROM ANIMADOR ani, TRABALHA tr
WHERE (tr.disponibilidade LIKE '%Sabado%' and tr.disponibilidade LIKE '%Domingo%')
        and tr.cpf_animador = ani.cpf;
```

```
-- FIM DAS CONSULTAS!


-- Tabela que o segundo trigger irá utilizar:


CREATE TABLE HISTORICO_PRECOS(
        nome_pizza VARCHAR(50) NOT NULL,
        nome_pizzaria VARCHAR(50) NOT NULL,
        preco_antigo REAL NOT NULL,
        preco_novo REAL NOT NULL,
        situacao VARCHAR(50),
        porcentagem FLOAT4,
        usuario CHAR(20),
        data_feita TIMESTAMP
);


-- Trigger que grava historico de cada alteração do preco de alguma pizza armazenando o
-- seu valor antigo, o novo, se o mesmo aumentou ou diminuiu ou ficou indiferente, a porcentagem
-- que aumentou ou diminuiu, o usuario que fez a alteração e o horário em que foi feita:


CREATE OR REPLACE FUNCTION att_preco()
RETURNS TRIGGER AS $$
DECLARE pc FLOAT4;
```

```
BEGIN

    IF new.preco > old.preco THEN

        pc := ((new.preco - old.preco) / old.preco) * 100;

        INSERT INTO HISTORICO_PRECOS VALUES (old.nome_pizza,
old.nome_pizzaria, old.preco, new.preco, 'Aumentou', pc, CURRENT_USER, NOW());

    ELSIF new.preco < old.preco THEN

        pc := ((old.preco - new.preco) / old.preco) * 100;

        INSERT INTO HISTORICO_PRECOS VALUES (old.nome_pizza,
old.nome_pizzaria, old.preco, new.preco, 'Diminuiu', pc, CURRENT_USER, NOW());

    ELSIF new.preco = old.preco THEN

        pc := 0;

        INSERT INTO HISTORICO_PRECOS VALUES (old.nome_pizza,
old.nome_pizzaria, old.preco, new.preco, 'Indiferente', pc, CURRENT_USER, NOW());
```

```
        END IF;


        RETURN NULL;


END $$ LANGUAGE 'plpgsql';


CREATE TRIGGER add_historico

AFTER UPDATE OF preco ON PIZZA

FOR EACH ROW

EXECUTE PROCEDURE att_preco();


-- Testes:


UPDATE PIZZA SET preco = 34.00 WHERE nome_pizza = 'Mussarela' and nome_pizzaria =
'Happy Pizza';

UPDATE PIZZA SET preco = 50.00 WHERE nome_pizza = 'Peperone' and nome_pizzaria =
'Que Delicia Pizzaria';

UPDATE PIZZA SET preco = 40.00 WHERE nome_pizza = 'Peperone' and nome_pizzaria =
'Que Delicia Pizzaria';

UPDATE PIZZA SET preco = 67.50 WHERE nome_pizza = 'Brigadeiro' and nome_pizzaria =
'Pizzaria Pizza Ria';

UPDATE PIZZA SET preco = 40.50 WHERE nome_pizza = 'Portuguesa' and nome_pizzaria
= 'Pizzaria Anima Tudo';


SELECT *
```

```
FROM HISTORICO_PRECOS;


SELECT *

FROM PIZZA;
```