UNIVERSIDADE FEDERAL DA BAHIA

Tarcisio de Assis Pereira Brito

Projeto I - Elaboração, implantação, governança e uso de banco de dados em estudo de caso: Caso Pizzaria da Vivi

SALVADOR

2024

1. Introdução

Este relatório descreve o desenvolvimento de um banco de dados relacional projetado para gerenciar as operações de uma empresa de delivery de pizzas do interior da Bahia, chamada "Vivi Pizzas", compreendendo desde o mini-mundo até as últimas queries. O sistema de banco de dados foi desenvolvido para suportar diversas funções essenciais da pizzaria que a dona estaria disposta a automatizar, incluindo o gerenciamento de clientes, pedidos, produtos, ingredientes e funcionários. O objetivo principal é potencializar e otimizar processos como o controle de estoque, o processamento de pedidos e a alocação de recursos humanos de forma eficiente. A estrutura do banco de dados foi modelada para garantir a integridade e consistência dos dados, além de permitir consultas analíticas que possibilitam um melhor entendimento do desempenho da pizzaria, como a análise do histórico de compras dos clientes e o controle de vendas. As implementações das tabelas e relacionamentos envolvem não apenas a organização dos dados, mas também a criação de funcionalidades dinâmicas, como triggers e stored procedures, para garantir o bom funcionamento das operações diárias. Todos documentos, códigos e conteúdos referentes a esse banco de dados podem ser encontrado tanto no github como no google drive, cujos links seguem:

https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi

https://drive.google.com/drive/folders/1t9SqgngQzD_wzpUQpAkdwOoxsZFXQ9SZ?usp=drive_link

Link do vídeo explicativo:

2. Modelando a base de dados

2.1. Descrição do Mini-mundo:

Vivi Pizzas é uma empresa de delivery de pizzas que funciona através do whatsapp, se destaca pela qualidade de suas pizzas e velocidade de atendimento, possível pela preparação prévia do estoque. Frente à uma expansão, Vivi precisa de ajuda. Para gerenciar suas operações,o sistema de informação precisa ser capaz de controlar diversos aspectos da pizzaria, como o gerenciamento de funcionários, o processamento de pedidos e o controle de estoque de produtos.

Primeiro de tudo, o sistema deve gerenciar o relacionamento com os clientes, listando para eles apenas produtos disponíveis e lhes trazer informações sobre os ingredientes das pizzas quando necessário, algo importante para quem não come carne ou tem intolerância à lactose. Cada cliente pode realizar múltiplos pedidos, e o sistema deve ser capaz de associar os pedidos aos clientes e gerar um histórico de compras. Isso permite um atendimento personalizado e a possibilidade de oferecer promoções.

A gestão de pedidos é o principal ponto do sistema. Quando um cliente faz um pedido, o sistema deve registrar os produtos solicitados.O sistema deve garantir que o pedido só seja aceito se possuir os produtos necessários. Também deve associar o pedido aos funcionários responsáveis por realizá-lo, desde que estejam disponíveis.

O sistema deve manter um registro detalhado dos funcionários da pizzaria. Cada funcionário está associado a um cargo específico, como atendente, entregador, entre outros, e o sistema deve registrar também as informações relacionadas à remuneração e à carga horária de cada um. Além disso, o sistema deve permitir o cadastro de novos funcionários e a atualização das informações de acordo com mudanças ocorridas no mundo real.

Os produtos vendidos pela pizzaria, como pizzas, bebidas e outros itens do cardápio, devem ser registrados no sistema. O controle de estoque é fundamental para garantir que a pizzaria não venda algo que não tem. O sistema deve permitir o registro de vendas e o controle da quantidade de produtos disponíveis, evitando falta de estoque e desperdício. Também deve guardar as quantidades de ingredientes necessários para cada receita, garantindo a replicabilidade dos pratos.

2.2 Requisitos Funcionais:

RF1 Cadastro e Gerenciamento de Clientes: O sistema deve permitir o cadastro e atualização de clientes, registrando informações como nome, telefone, endereço e CPF, além de manter um histórico de compras.

RF2 Gestão de Produtos e Controle de Estoque: O sistema deve registrar produtos e controlar o estoque, atualizando automaticamente a quantidade disponível quando um pedido é realizado.

RF3 Gestão de Funcionários: O sistema deve permitir o cadastro de funcionários com dados como nome, cargo, salário e disponibilidade, além de controlar sua alocação nos pedidos conforme a disponibilidade.

RF4 Gerenciamento de Pedidos: O sistema deve permitir a realização de pedidos pelos clientes, associando-os aos produtos disponíveis em estoque e registrando informações como valor total, data e status do pedido.

RF5 Atribuição de Funcionários a Pedidos: O sistema deve alocar automaticamente funcionários disponíveis para cada pedido, levando em consideração a disponibilidade e os cargos necessários.

RF6 Controle de receitas, Ingredientes e Alérgenos: O sistema deve gerenciar os ingredientes de cada produto, fornecendo informações sobre alergênicos e intolerâncias alimentares para que os clientes possam escolher de acordo com suas necessidades, além de salvar as receitas dos pratos.

RF7 Promoções e reajuste de valores de produtos: O sistema deve permitir promoções em produtos específicos ou no geral e permitir reajuste dos produtos em determinada %, importante por causa da inflação. Também deve colocar os clientes em três categorias de acordo

RF8 Relatórios de Vendas e Estoque: O sistema deve gerar relatórios periódicos sobre as vendas, produtos vendidos e controle de estoque, auxiliando na gestão eficiente da pizzaria.

2.3 Delimitação do mini-mundo para o banco de dados

O banco de dados relacional desenvolvido para suportar as operações deste sistema de informação deve contar com tabelas e atributos coerentes ao mini-mundo. Abaixo, segue:

1. Tabela Cliente:

Descrição: Armazena informações dos clientes da pizzaria.

- **cp_id_cliente (SERIAL)**: Identificador único do cliente. Chave primária
- **nm_cliente (VARCHAR(100))**: Nome do cliente. Não nulo
- **tel_cliente (VARCHAR(15))**: Telefone do cliente. Pode ser nulo
- **endereco (VARCHAR(255))**: Endereço do cliente. Pode ser nulo
- **cpf_cliente (VARCHAR(11))**: CPF do cliente. Não nulo, Único
- **qtd_compras (INT)**: Quantidade de compras realizadas pelo cliente. Valor padrão: 0
- **categoria_cliente (VARCHAR(10))**: Categoria do cliente (bronze, prata, ouro). Valor padrão: 'bronze'

---

2. Tabela Pedido:

Descrição: Armazena informações dos pedidos realizados pelos clientes.

- **cp_id_pedido (SERIAL)**: Identificador único do pedido. Chave primária
- **ce_cliente (INT)**: Identificador do cliente que fez o pedido. Não nulo, Chave estrangeira que referencia Cliente(cp_id_cliente)
- **data_pedido (DATE)**: Data em que o pedido foi feito. Valor padrão: CURRENT_TIMESTAMP
- **observações (VARCHAR(255))**: Observações adicionais sobre o pedido. Pode ser nulo
- **status_pedido (VARCHAR(20))**: Status do pedido (Não-confirmado, Em aberto, Concluído, Cancelado). Não nulo, Valor padrão: 'Não-confirmado'
- **valor_total (DECIMAL(10, 2))**: Valor total do pedido. Pode ser nulo

---

3. Tabela Cargo:

Descrição: Armazena os cargos dos funcionários da pizzaria.

- **cp_id_cargo (SERIAL)**: Identificador único do cargo. Chave primária
- **nm_cargo (VARCHAR(100))**: Nome do cargo (ex: atendente, cozinheiro, entregador). Não nulo

- **salario (DECIMAL(10, 2))**: Salário do cargo. Pode ser nulo
- **horas_semanais (INT)**: Carga horária semanal para o cargo. Pode ser nulo

---

4. Tabela Funcionario:

Descrição: Armazena informações dos funcionários da pizzaria.

- **cp_id_funcionario (SERIAL)**: Identificador único do funcionário. Chave primária
- **ce_cargo (INT)**: Identificador do cargo do funcionário. Não nulo, Chave estrangeira que referencia Cargo(cp_id_cargo)
- **nm_funcionario (VARCHAR(100))**: Nome do funcionário. Não nulo
- **cpf_funcionario (VARCHAR(11))**: CPF do funcionário. Pode ser nulo, Único
- **email_funcionario (VARCHAR(100))**: Email do funcionário. Pode ser nulo
- **tel_funcionario (VARCHAR(15))**: Telefone do funcionário. Pode ser nulo
- **tarefas_ativas (INT)**: Número de tarefas/pedidos atualmente atribuídos ao funcionário. Pode ser nulo
- **status_funcionario (VARCHAR(15))**: Status do funcionário (Indisponível, Disponível). Não nulo, Valor padrão: 'Indisponível'

---

5. Tabela Produto:

Descrição: Armazena informações sobre os produtos vendidos na pizzaria.

- **cp_id_produto (SERIAL)**: Identificador único do produto. Chave primária
- **tipo_produto (VARCHAR(100))**: Tipo do produto (pizza, aperitivo, sobremesa, bebida, etc.). Não nulo
- **nm_produto (VARCHAR(100))**: Nome do produto. Não nulo
- **preco_produto (DECIMAL(10, 2))**: Preço do produto. Não pode ser nulo
- **descricao (VARCHAR(255))**: Descrição do produto. Pode ser nulo
- **tamanho_pizza (VARCHAR(20))**: Tamanho da pizza, caso o produto seja pizza. Pode ser nulo
- **volume_bebida (VARCHAR(20))**: Volume da bebida, caso o produto seja bebida. Pode ser nulo
- **qtd_disponivel (INT)**: Quantidade disponível do produto. Pode ser nulo
- **porcentagem_promoção (DECIMAL(10, 2))**: Porcentagem de desconto do produto. Valor padrão: 0

---

6. Tabela Ingrediente:

Descrição: Armazena informações sobre os ingredientes usados nos produtos.

- **cp_id_ingrediente (SERIAL)**: Identificador único do ingrediente. Chave primária
- **nm_ingrediente (VARCHAR(100))**: Nome do ingrediente. Não nulo
- **tipo (VARCHAR(20))**: Tipo de ingrediente (animal, vegetal, farináceo, lácteo, outro). Não nulo

---

7. Tabela Funcionario_pedido:

Descrição: Armazena a associação de um funcionário a um pedido específico, com o papel que o funcionário desempenha.

- **cp_id_pedido_id_funcionario (SERIAL)**: Identificador único da associação entre pedido e funcionário. Chave primária
- **ce_funcionario (INT)**: Identificador do funcionário atribuído ao pedido. Não nulo, Chave estrangeira que referencia Funcionario(cp_id_funcionario)
- **ce_pedido (INT)**: Identificador do pedido associado. Não nulo, Chave estrangeira que referencia Pedido(cp_id_pedido)
- **papel (VARCHAR(50))**: Papel do funcionário no pedido (ex: Entregador, Cozinheiro). Pode ser nulo
- **observação_funcionário (VARCHAR(100))**: Observações do funcionário sobre o pedido. Pode ser nulo

---

8. Tabela Produto_pedido:

Descrição: Armazena a relação entre produtos e pedidos realizados.

- **cp_id_pedido_id_produto (SERIAL)**: Identificador único da associação entre produto e pedido. Chave primária
- **ce_produto (INT)**: Identificador do produto no pedido. Não nulo, Chave estrangeira que referencia Produto(cp_id_produto)
- **ce_pedido (INT)**: Identificador do pedido em que o produto foi incluído. Não nulo, Chave estrangeira que referencia Pedido(cp_id_pedido)
- **qtd_compradas (INT)**: Quantidade do produto comprada no pedido. Não nulo

---

9. Tabela Produto_ingrediente:

Descrição: Armazena a relação entre os produtos e os ingredientes utilizados.

- **cp_id_produto_id_ingrediente (SERIAL)**: Identificador único da associação entre produto e ingrediente. Chave primária
- **ce_produto (INT)**: Identificador do produto. Não nulo, Chave estrangeira que referencia Produto(cp_id_produto)
- **ce_ingrediente (INT)**: Identificador do ingrediente. Não nulo, Chave estrangeira que referencia Ingrediente(cp_id_ingrediente)
- **gramas_ingrediente (INT)**: Quantidade de gramas do ingrediente utilizado no produto. Valor padrão: 1

2.4. Relacionamentos entre tabelas principais

1. **Relacionamento entre Cliente e Pedido**:

    - **Tipo**: Um-para-muitos
    - **Descrição**: Um cliente pode fazer de zero a vários pedidos, mas cada pedido é realizado por apenas um cliente. Por exemplo, o cliente "João" pode realizar diversos pedidos ao longo do tempo, enquanto cada pedido está associado a apenas um cliente.
2. **Relacionamento entre Pedido e Produto**:

    - **Tipo**: Muitos-para-muitos
    - **Descrição**: Um pedido pode conter de um vários produtos, e um produto pode aparecer em zero ou múltiplos pedidos. Por exemplo, um cliente pode pedir uma pizza e uma bebida no mesmo pedido, e o mesmo produto (como uma pizza) pode ser pedido por vários clientes em pedidos diferentes.
3. **Relacionamento entre Produto e Ingrediente**:

    - **Tipo**: Muitos-para-muitos
    - **Descrição**: Um produto pode ter zero (produtos prontos) ou múltiplos ingredientes (produtos da casa), e um ingrediente pode ser utilizado em zero ou vários produtos. Por exemplo, a pizza de "calabresa" pode ter vários ingredientes como queijo, molho de tomate, calabresa e outros. Da mesma forma, o ingrediente "queijo" pode ser utilizado em várias pizzas e outros produtos.
    -
4. **Relacionamento entre Pedido e Funcionario**

    - **Tipo**: Muitos-para-muitos
    - **Descrição**: Um funcionário pode ser responsável por zero a múltiplos pedidos e, ao mesmo tempo, um pedido pode ter de 1 a múltiplos funcionários envolvidos, como cozinheiro e entregador. Por exemplo,

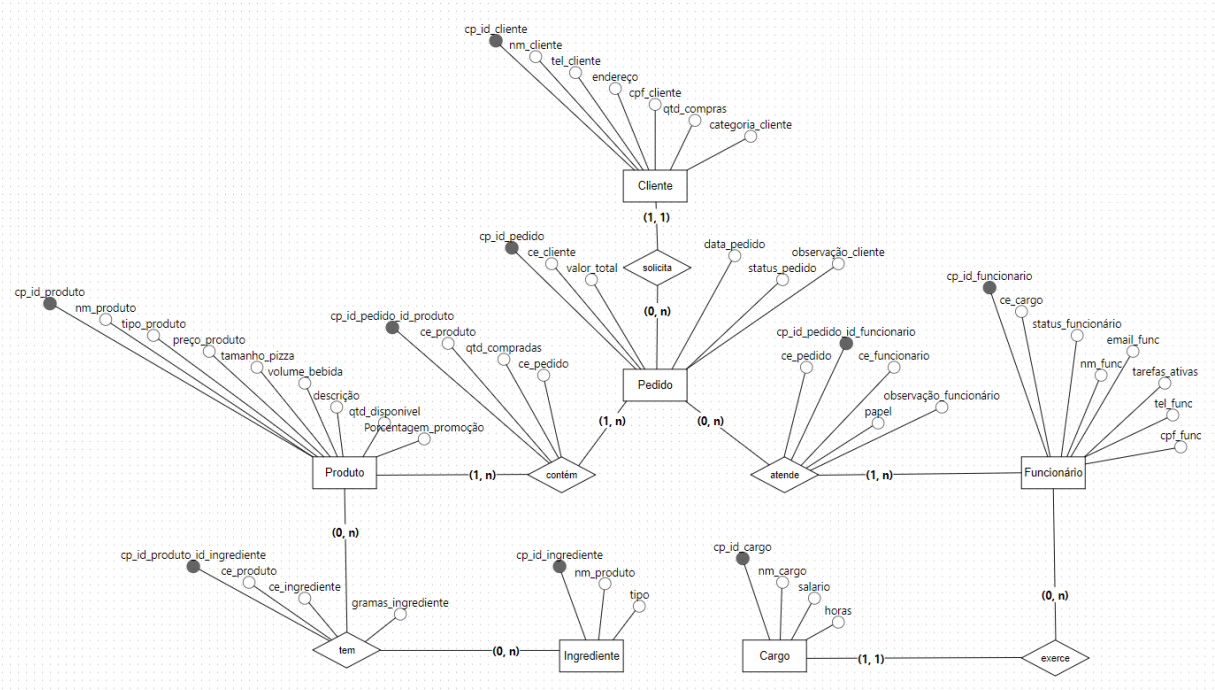
um cozinheiro pode preparar várias pizzas, enquanto o entregador pode ser responsável por entregar todas elas durante a noite.

5. **Relacionamento entre Funcionario e Cargo**:

    - **Tipo**: Muitos-para-um
    - **Descrição**: Vários funcionários podem ter o mesmo cargo, mas cada funcionário está associado a apenas um cargo. Por exemplo, vários atendentes podem estar trabalhando na pizzaria, mas eles não acumulam funções.

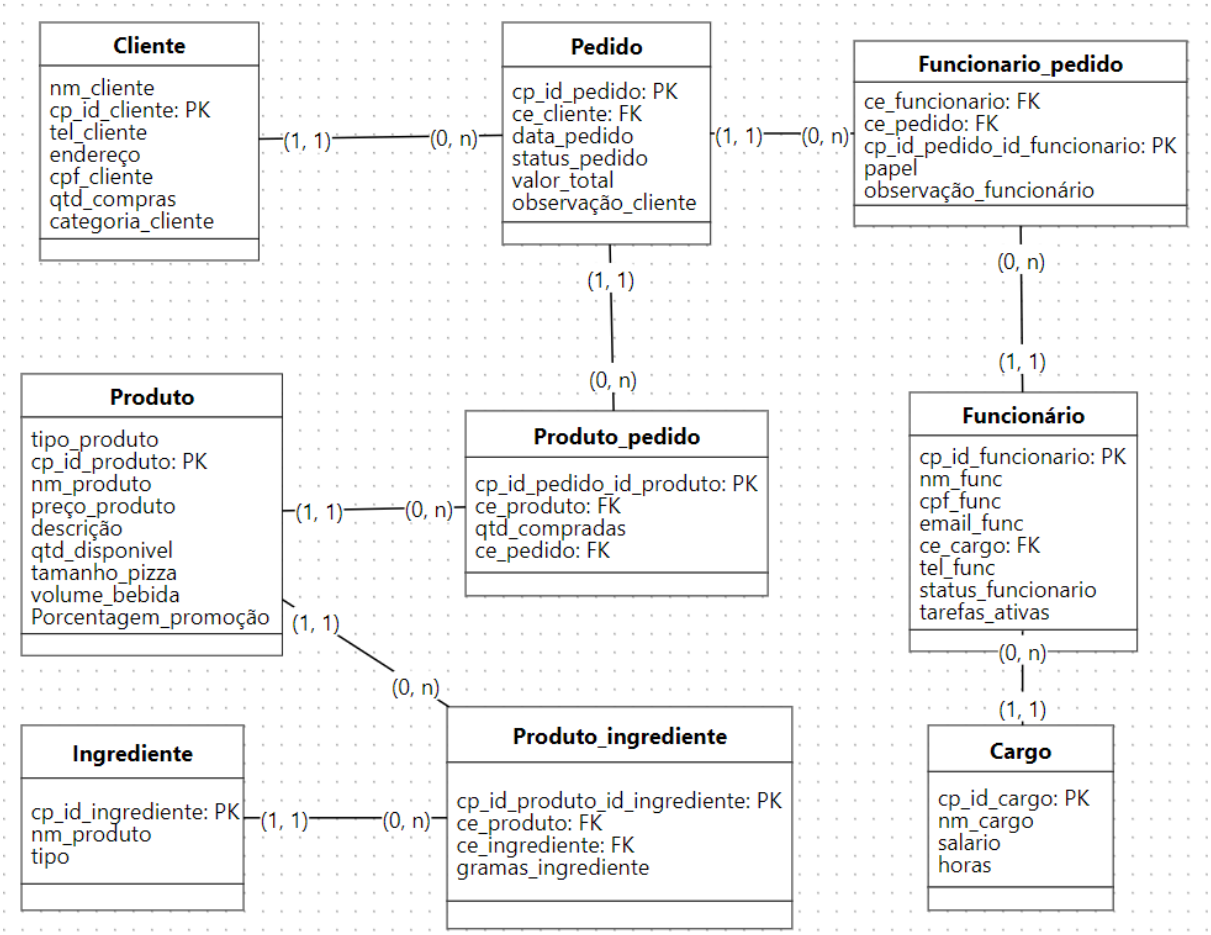
2.5. Modelo Conceitual e lógico

2.5.1. Modelo conceitual



Link do arquivo no GitHub com mais definição e detalhes:
https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/blob/main/Diagramas/Projeto%20conceitual.png

2.5.2. Modelo lógico



Link do arquivo no GitHub com mais definição e detalhes:
https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/blob/main/Diagramas/Projeto%20lógico.png

3. Tabelas

3.1. Confecção das tabelas

A seguir estão os comandos para criar as tabelas, disponivel em:

https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/blob/main/queries/criar_tabelas.sql

```
-- Drop the existing ENUM type if it exists

DROP TYPE IF EXISTS status_funcionario_enum;

CREATE TYPE status_funcionario_enum AS ENUM ('Indisponível', 'Disponível');
```

```sql
CREATE TABLE IF NOT EXISTS Cliente

(

    cp_id_cliente SERIAL,

    nm_cliente VARCHAR(100) NOT NULL,

    tel_cliente VARCHAR(15),

    endereco VARCHAR(255),

    cpf_cliente VARCHAR(11) NOT NULL,

    qtd_compras INT DEFAULT 0,

    categoria_cliente VARCHAR(10) DEFAULT 'bronze'

);



CREATE TABLE IF NOT EXISTS Pedido

(

    cp_id_pedido SERIAL,

    ce_cliente INT NOT NULL,

    data_pedido DATE DEFAULT CURRENT_TIMESTAMP,

    observações VARCHAR(255),

    status_pedido VARCHAR(20) NOT NULL DEFAULT 'Em aberto',

    valor_total DECIMAL(10, 2)

);



CREATE TABLE IF NOT EXISTS Cargo

(

    cp_id_cargo SERIAL,

    nm_cargo VARCHAR(100) NOT NULL,

    salario DECIMAL(10, 2),

    horas_semanais INT

);
```

```sql
CREATE TABLE IF NOT EXISTS Funcionario

(

    cp_id_funcionario SERIAL,

    ce_cargo INT NOT NULL,

    nm_funcionario VARCHAR(100) NOT NULL,

    cpf_funcionario VARCHAR(11),

    email_funcionario VARCHAR(100),

    tel_funcionario VARCHAR(15),

    tarefas_ativas INT,

    status_funcionario status_funcionario_enum NOT NULL DEFAULT 'Indisponível'

);



CREATE TABLE IF NOT EXISTS Produto

(

    cp_id_produto SERIAL,

    tipo_produto VARCHAR(100) NOT NULL,

    nm_produto VARCHAR(100) NOT NULL,

    preco_produto DECIMAL(10, 2),

    descricao VARCHAR(255),

    tamanho_pizza VARCHAR(20),

    volume_bebida VARCHAR(20),

    qtd_disponivel INT,

    porcentagem_promoção DECIMAL(10, 2) DEFAULT 0

);



CREATE TABLE IF NOT EXISTS Ingrediente

(

    cp_id_ingrediente SERIAL,

    nm_ingrediente VARCHAR(100) NOT NULL,
```

```
    tipo VARCHAR(20) NOT NULL
);

CREATE TABLE IF NOT EXISTS Funcionario_pedido
(
    cp_id_pedido_id_funcionario SERIAL,
    ce_funcionario INT NOT NULL,
    ce_pedido INT NOT NULL,
    papel VARCHAR(50),
    observação_funcionário VARCHAR(100)

);

CREATE TABLE IF NOT EXISTS Produto_pedido
(
    cp_id_pedido_id_produto SERIAL,
    ce_produto INT NOT NULL,
    ce_pedido INT NOT NULL,
    qtd_compradas INT NOT NULL DEFAULT 1
);

CREATE TABLE IF NOT EXISTS Produto_ingrediente
(
    cp_id_produto_id_ingrediente SERIAL,
    ce_produto INT NOT NULL,
    ce_ingrediente INT NOT NULL,
    gramas_ingrediente INT DEFAULT 1
);
```

3.2. Constraints das tabelas

A seguir estão os comandos para gerar as constraints, disponível em: https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/blob/main/queries/criar_constraints.sql

```
-- Constraints para a tabela Cliente

ALTER TABLE Cliente

    ADD PRIMARY KEY (cp_id_cliente),

    ADD UNIQUE (cpf_cliente),

    ADD CHECK (categoria_cliente IN ('bronze', 'prata', 'ouro')),

    ADD CHECK (qtd_compras>=0);



-- Constraints para a tabela Pedido

ALTER TABLE Pedido

    ADD PRIMARY KEY (cp_id_pedido),

    ADD CHECK (status_pedido IN ('Não-concluido','Em aberto', 'Concluído', 'Cancelado')),

    ADD CHECK (valor_total > 0),

    ADD FOREIGN KEY (ce_cliente) REFERENCES Cliente(cp_id_cliente) ON DELETE CASCADE;



-- Constraints para a tabela Cargo

ALTER TABLE Cargo

    ADD PRIMARY KEY (cp_id_cargo),

    ADD CHECK (salario > 0);



-- Constraints para a tabela Funcionario

ALTER TABLE Funcionario

    ADD PRIMARY KEY (cp_id_funcionario),

    ADD UNIQUE (cpf_funcionario),

    ADD CHECK ( tarefas_ativas>=0 ),
```

```
    ADD FOREIGN KEY (ce_cargo) REFERENCES Cargo(cp_id_cargo) ON DELETE CASCADE;



-- Constraints para a tabela Produto

ALTER TABLE Produto

    ADD PRIMARY KEY (cp_id_produto),

      ADD CHECK (tipo_produto IN ('pizza', 'aperitivo', 'sobremesa', 'bebida', 'bebida
alcoolica', 'utensilios')),

    ADD CHECK (qtd_disponivel > 0),

    ADD CHECK (porcentagem_promoção < 100);



-- Constraints para a tabela Ingrediente

ALTER TABLE Ingrediente

    ADD PRIMARY KEY (cp_id_ingrediente),

    ADD CHECK (tipo IN ('animal', 'vegetal', 'farinha', 'lacticinio', 'outro'));



-- Constraints para a tabela Funcionario_pedido

ALTER TABLE Funcionario_pedido

    ADD PRIMARY KEY (cp_id_pedido_id_funcionario),

     ADD FOREIGN KEY (ce_funcionario) REFERENCES Funcionario(cp_id_funcionario) ON DELETE
CASCADE,

    ADD FOREIGN KEY (ce_pedido) REFERENCES Pedido(cp_id_pedido) ON DELETE CASCADE;



-- Constraints para a tabela Produto_pedido

ALTER TABLE Produto_pedido

    ADD PRIMARY KEY (cp_id_pedido_id_produto),

    ADD CHECK (qtd_compradas > 0),

    ADD FOREIGN KEY (ce_produto) REFERENCES Produto(cp_id_produto) ON DELETE CASCADE,

    ADD FOREIGN KEY (ce_pedido) REFERENCES Pedido(cp_id_pedido) ON DELETE CASCADE;



-- Constraints para a tabela Produto_ingrediente
```

```
ALTER TABLE Produto_ingrediente

    ADD PRIMARY KEY (cp_id_produto_id_ingrediente),

    ADD CHECK (gramas_ingrediente > 0),

    ADD FOREIGN KEY (ce_produto) REFERENCES Produto(cp_id_produto) ON DELETE CASCADE,

     ADD FOREIGN KEY (ce_ingrediente) REFERENCES Ingrediente(cp_id_ingrediente) ON DELETE
CASCADE;
```

3.3. Populando as tabelas

As populações foram feitas através do chatGPT e corrigidas manualmente em casos que ignoravam constraints. Elas estão disponiveis em:

https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/tree/main/tabelas

O arquivo all_tables_populate.sql possui todas as populações em um único arquivo.

4. Protótipos de telas, códigos Sql associados, sub-rotinas de suporte e perguntas analiticas.

É recomendado utilizar os códigos provenientes do github:

As queries se encontram no link:
https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/tree/main/queries

As Materialized views se encontram no link:

https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/tree/main/Materialized_views

As stored procedures se encontram no link:
https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/tree/main/stored%20procedures

4.1. Rotinas OLTP (Online Transaction Processing)

A seguir estão disponibilizados os comandos para Inclusão, Alteração e Exclusão de registro.

Inclusão:

Descrição de tela	Queries
Tela de cadastro de usuário em que tem os campos de nome e CPF obrigatórios, além de telefone, endereço, cpf, e um botão de concluir. Ao clicar em concluir, a query é chamada para adicionar o novo cliente.	`--Query para cadastro de um novo cliente:` `INSERT INTO Cliente (nm_cliente, tel_cliente, endereco, cpf_cliente, qtd_compras, categoria_cliente)` `VALUES ('João Silva', '999999999', 'Rua A, 123', '12345678901', 0, 'bronze');`
Tela de cadastro de funcionário acessada apenas por usuários com role admin, com campos obrigatórios de nome e CPF, além de email, telefone e um botão de concluir que quando clicado, essa query é chamada.	`--Query para cadastro de um novo funcionário:` `INSERT INTO Funcionario (cp_id_funcionario, ce_cargo, nm_funcionario, cpf_funcionario, email_funcionario, tel_funcionario, tarefas_ativas, status_funcionario) VALUES(1, 4, 'Robespierre', '93715678901', 'ocara@vivi.com', '(11) 98765-4321', 0, 'Disponível');`
Tela de cadastro de cargo acessada apenas por usuários com role admin, com campo obrigatório de nome, além de salário, horas semanais e um botão de concluir que quando clicado, essa query é chamada.	`--Query pra adicionar um cargo à tabela de cargos` `INSERT INTO Cargo (nm_cargo, salario, horas_semanais) VALUES ('Degustador', 15000.00, 40);`
Tela de cadastro de ingrediente acessada apenas por usuários com role funcionário ou superior, com campo obrigatório de nome,	`--Query para cadastrar um ingrediente à tabela de ingrediente:`

além de tipo e um botão de concluir que quando clicado, essa query é chamada pra dar o insert.	`INSERT INTO Ingrediente (nm_ingrediente, tipo) VALUES ('Queijo', 'lacticinio');`
Tela de display de produtos. Quando um usuário clica no botão de adicionar no carrinho ou de comprar essa query é chamada, criando um pedido Não-confirmado vinculado ao produto desejado através da tabela produto-pedido. Uma das mais importantes telas/querys, pois permite a análise de quais pedidos não foram confirmados, por quem, quando e quanto.	`--Query para registrar um novo pedido:` `WITH novo_pedido AS (` `INSERT INTO Pedido (ce_cliente, status_pedido, valor_total)` `VALUES (2, 'Não-confirmado', <valor_total>)` `RETURNING cp_id_pedido` `)` `INSERT INTO Produto_pedido (ce_produto, ce_pedido, qtd_compradas)` `SELECT 3, cp_id_pedido, 3` `FROM novo_pedido;`
Tela de cadastro de produto acessada apenas por usuários com role funcionário ou superior, com campos obrigatórios de nome, tipo e preço além de outros e um botão de concluir que quando clicado, essa query é chamada pra fazer o insert.	`--Query para cadastro de um novo produto:` `INSERT INTO Produto (tipo_produto, nm_produto, preco_produto, descricao, tamanho_pizza, volume_bebida, qtd_disponivel, porcentagem_promoção) VALUES ('pizza', 'Pizza de ovo', 25.50, 'Pizza com molho de tomate, mussarela e manjericão', 'Grande', NULL, 100, 10.0);`

Alteração de registros:

Descrição de tela	Queries
Tela de edição de cargo acessada apenas por usuários com a role de admin. Exibe dois campos editáveis: "Nome do Cargo" e "Salário". O nome e salário do cargo são preenchidos automaticamente com valores resultantes de uma consulta anterior. Após a edição, o usuário clica no botão de "Enviar", que chama a query para atualizar os dados do cargo.	`--Query para alterar nome ou salário de um cargo específico:` `UPDATE Cargo` `SET nm_cargo = 'Coordenador',` `salario = 3500.00` `WHERE cp_id_cargo = 1;`
Tela de edição de cliente acessivel apenas por ele mesmo ou administrador que permite a edição de dados de um cliente, como o número de telefone. O usuário pode alterar o telefone do cliente através de um campo editável. Após a alteração, ao clicar no botão "Salvar", a query será chamada para atualizar o telefone do cliente.	`--Query para atualizar dados do cliente (exemplo de atualização de telefone):` `UPDATE Cliente` `SET tel_cliente = '71988888888'` `WHERE cp_id_cliente = 1;`
Tela de registro de ponto acessivel apenas poronde o status de disponibilidade de um funcionário pode ser alterado. Quando o ponto é registrado, a respectiva query é chamada, de acordo se o funcionário está entrando ou saindo.	`--Query para atualizar a disponibilidade de um funcionário` `UPDATE Funcionario` `SET status_funcionario = 'Disponível'` `WHERE cp_id_funcionario = 1;`

	`--Query oposta` `UPDATE Funcionario` `SET status_funcionario = 'Disponível'` `WHERE cp_id_funcionario = 1;`
	`--Alterar o nome ou tipo de um ingrediente específico` `UPDATE Ingrediente` `SET nm_ingrediente = 'Queijo Mozarela',` `tipo = 'lacticinio'` `WHERE cp_id_ingrediente = 3;`
Tela de confirmação de entrega do pedido, acessivel apenas pelo funcionário atribuido à ele ou administrador. Quando clica no botão de entregue, essa query é chamada para fazer o update.	`--Query para atualizar o status de um pedido:` `UPDATE Pedido` `SET status_pedido = 'Concluído'` `WHERE cp_id_pedido = 1;`
Tela para editar o estoque de um produto, acessivel apenas por um funcionário ou administrador, onde o campo "Quantidade Disponível" é mostrado e pode ser ajustado. O usuário pode aumentar ou diminuir a quantidade de produtos disponíveis. Após a alteração, ao clicar em "Atualizar Estoque", a query será chamada para atualizar o estoque do produto.	`--Query para atualizar o estoque de um produto:` `UPDATE Produto` `SET qtd_disponivel = qtd_disponivel + 10` `WHERE cp_id_produto = 1;`

Exclusão de registros:

Descrição de tela	Queries
Esta tela permite que o usuário com permissões de administrador edite ou remova um cargo da plataforma.	`--Excluir um cargo específico` `DELETE FROM Cargo` `WHERE cp_id_cargo = 2;`
Tela de demissão que permite ao administrador alterar a disponibilidade de um funcionário ou removê-lo da plataforma e ao clicar no botão de demitir essa querie é chamada.	`--Query para excluir um funcionário` `DELETE FROM Funcionario` `WHERE cp_id_funcionario = 1;`
Tela para editar ingredientes inclui o simbolo de uma lixeira. Ao clicar neste botão essa querie é chamada.	`--Para excluir um ingrediente específico da tabela Ingrediente:` `DELETE FROM Ingrediente` `WHERE cp_id_ingrediente = 3;`
Tela para editar produtos inclui o simbolo de uma lixeira. Ao clicar neste botão essa querie é chamada.	`--Excluir um produto específico` `DELETE FROM Produto` `WHERE cp_id_produto = 10;`

4.2. Rotinas OLAP (Online Analytical Processing)

Consultas simples

Descrição de tela/ Pergunta analitica	Queries
Tela de cadastro de ingrediente acessada apenas por usuários com role funcionário ou superior, com campo obrigatório de nome, além de tipo. Quando o campo de tipo é clicado, aparece uma lista de tipos possíveis, que vem do resultado dessa query.	`--Query para consultar os tipos possíveis de ingrediente` `SELECT DISTINCT tipo` `FROM Ingrediente;`
Quando o administrador acessar a tela de salários dos cargos a query é chamada e o que aparece é fruto dessa querie.	`--Cargos e seus salários` `SELECT cp_id_cargo, nm_cargo, salario FROM Cargo`
Quando o administrado acessar o dashboard na aba de pedidos e selecionar o botão “análise de horários de pico” essa query é chamada e seu resultado é exibido. PA1: Quais são os horários com maior quantidade de pedidos? Sugestão de gráfico de barras ou linha para visualiza a evolução da quantidade de pedidos.	`--Query pra determinar os horários de pico dos pedidos no ultimo mes` `SELECT` `EXTRACT(HOUR FROM p.data_pedido) AS hora,` `COUNT(p.cp_id_pedido) AS total_pedidos` `FROM` `Pedido p` `WHERE` `p.status_pedido = 'Concluído' -- Considera apenas pedidos concluídos`

	`` AND p.data_pedido >= CURRENT_DATE - INTERVAL '1 month' -- Filtra os pedidos do último mês GROUP BY hora ORDER BY total_pedidos DESC; ```
Quando um funcionário ou administrador acessar a tela de funcionários disponiveis essa querie é chamada e o resultado dela é mostrado.	``` --Query para ver funcionários disponíveis SELECT nm_funcionario, cpf_funcionario, ce_cargo, status_funcionario FROM Funcionario WHERE status_funcionario = 'Disponível'; ```
Quando um administrador ou um cliente acessar sua própria tela de perfil, essa querie é chamada e o resultado é mostrado nas informações do cliente	``` --query para consultar os dados de um cliente SELECT cp_id_cliente, nm_cliente, tel_cliente, endereco, cpf_cliente, qtd_compras, categoria_cliente FROM Cliente ```

	`` WHERE cp_id_cliente = 1; -- substitua o '1' pelo ID do cliente que deseja consultar ```
Ao acessar a tela de um produto essa query é chamada e o resultado é apresentado na parte de preço, considerando o desconto.	``` --Consultar o preço de um produto considerando promoção SELECT cp_id_produto, nm_produto, preco_produto, porcentagem_promoção, preco_produto * (1 - porcentagem_promoção / 100) AS preco_com_desconto FROM Produto WHERE cp_id_produto = 10; ```

Consultas intermediárias:

Descrição de tela/ Pergunta analitica	Queries
Quando um administrador ou funcionário acessar a tela de	``` --Query para ver funcionários do cargo "entregador" disponíveis ```

funcionários e clicar em “visualizar entregadores disponíveis no momento”, essa query será chamada e o resultado dela será mostrado. PA2: Quais funcionários entregadores estão disponíveis no momento?	SELECT f.nm_funcionario, f.cpf_funcionario, f.status_funcionario FROM Funcionario f JOIN Cargo c ON f.ce_cargo = c.cp_id_cargo WHERE c.nm_cargo = 'entregador' AND f.status_funcionario = 'Disponível';
Quando um administrador acessar a tela de funcionários para entrega, essa query será chamada e o resultado dela será mostrado. PA3: Quais funcionários atenderam mais pedidos no último mês? Recomendada a visualização utilizando gráfico de barras, ordenando pelos mais produtivos.	--Query pra determinar quais funcionários atenderam mais pedidos no último mês SELECT f.nm_funcionario, COUNT(fp.cp_id_pedido) AS total_pedidos_atendidos FROM Funcionario_pedido fp JOIN Funcionario f ON fp.ce_funcionario = f.cp_id_funcionario JOIN Pedido p ON fp.ce_pedido = p.cp_id_pedido WHERE

	p.status_pedido = 'Concluído' -- Considera apenas pedidos concluídos AND p.data_pedido >= CURRENT_DATE - INTERVAL '1 month' -- Filtra os pedidos do último mês GROUP BY f.nm_funcionario ORDER BY total_pedidos_atendidos DESC;
Quando um administrador ou funcionário acessar a tela de funcionários entregadores disponíveis para entrega, essa query será chamada e o resultado dela será mostrado. PA4: Quais produtos de cada categoria foram mais vendidos?	--Consultar quais são os produtos mais vendidos em cada categoria SELECT pr.tipo_produto, pr.nm_produto, SUM(pp.qtd_compradas) AS total_vendido FROM Produto pr JOIN Produto_pedido pp ON pr.cp_id_produto = pp.ce_produto JOIN Pedido p ON pp.ce_pedido = p.cp_id_pedido WHERE

	`p.status_pedido = 'Concluído' -- Considera apenas pedidos concluídos` `GROUP BY` `pr.tipo_produto, pr.nm_produto` `ORDER BY` `pr.tipo_produto, total_vendido DESC;`
Quando um administrador acessar a tela de dashboard na parte de funcionários e preencher o campo de “salário acima de” e der enter, essa query é chamada e os resultados são listados. PA5: Quais funcionários recebem acima de “x”?	`--Query para ver funcionários com salário acima de 2000 reais:` `SELECT f.nm_funcionario, f.cpf_funcionario, c.nm_cargo, c.salario` `FROM Funcionario f` `JOIN Cargo c ON f.ce_cargo = c.cp_id_cargo` `WHERE c.salario > x;`
Quando um administrador acessar a tela de dashboard na parte de pedidos e clicar no botão de “promoção x não-promoção” essa querie é chamada e a comparação é exibida. PA6: O negócio tem vendido mais produtos com ou sem promoção?	`--Query para comparar a quantidade de compras de produtos com promoção e sem promoção` `SELECT` `CASE` `WHEN p.porcentagem_promoção > 0 THEN 'Com Promoção'` `ELSE 'Sem Promoção'`

	`` END AS tipo_promocao, COUNT(pp.cp_id_pedido) AS total_compras FROM Produto p JOIN Produto_pedido pp ON p.cp_id_produto = pp.ce_produto JOIN Pedido ped ON pp.ce_pedido = ped.cp_id_pedido WHERE ped.status_pedido = 'Concluído' -- Considera apenas pedidos concluídos GROUP BY tipo_promocao; ```
Quando um usuário selecionar o botão de “sem lactose” na tela de pizza e aperitivos e der enter, essa query é chamada e o resultado é mostrado na tela.	``` ---Query para ver pizzas e aperitivos sem ingrediente do tipo "lacticinio" SELECT p.nm_produto, p.tipo_produto FROM Produto p LEFT JOIN Produto_ingrediente pi ON p.cp_id_produto = pi.ce_produto ```

	LEFT JOIN Ingrediente i ON pi.ce_ingrediente = i.cp_id_ingrediente WHERE (p.tipo_produto IN ('pizza', 'aperitivo')) AND (i.tipo != 'lacticinio' OR i.tipo IS NULL);

Consultas avançadas

Pergunta analitica	Relatórios e Dashboard
PA7: Quais funcionários estiveram envolvidos em mais pedidos cancelados?	A query traz os funcionários em ordem de número de cancelamentos, então um dashboard com gráfico de barras permitiria a visualização.

Query avançada equivalente:

```
--Resumo para contenção de danos:

--Query que define quais funcionários e de que cargos estiveram envolvidos em pedidos cancelados

SELECT

    f.nm_funcionario,

    c.nm_cargo,

    COUNT(p.cp_id_pedido) AS numero_pedidos_cancelados

FROM

    Funcionario_pedido fp

JOIN

    Funcionario f ON fp.ce_funcionario = f.cp_id_funcionario
```

```
JOIN

    Cargo c ON f.ce_cargo = c.cp_id_cargo

JOIN

    Pedido p ON fp.ce_pedido = p.cp_id_pedido

WHERE

    p.status_pedido = 'Cancelado'

GROUP BY

    f.nm_funcionario, c.nm_cargo

ORDER BY

    numero_pedidos_cancelados DESC, c.nm_cargo, f.nm_funcionario;
```

Pergunta analitica	Relatórios e Dashboard
PA8: Quantas pessoas que não pedem produtos de fonte animal o negócio tem atraido nos últimos meses?	Sugerido gráfico de linha ou barras para mostrar a evolução da quantidade de pessoas que fazem pedidos sem produtos de fonte animal ao longo do tempo

Query avançada equivalente:

```
-- Resumo do número de clientes que fizeram pedidos em cada mês e nunca compraram produtos com ingredientes do tipo "animal"

SELECT

       TO_CHAR(p.data_pedido, 'YYYY-MM') AS mes,   -- Agrupa por mês (ano-mês)

    COUNT(DISTINCT p.ce_cliente) AS numero_de_clientes

FROM
```

```
    Pedido p
JOIN
    Produto_pedido pp ON p.cp_id_pedido = pp.ce_pedido
JOIN
    Produto_ingrediente pi ON pp.ce_produto = pi.ce_produto
JOIN
    Ingrediente i ON pi.ce_ingrediente = i.cp_id_ingrediente
WHERE
    p.status_pedido = 'Concluído'  -- Apenas pedidos concluídos
       AND p.data_pedido >= CURRENT_DATE - INTERVAL '1 month'    -- Considera o último mês
     AND NOT EXISTS (  -- Verifica se o cliente NUNCA comprou produtos com ingredientes "animal"
        SELECT 1
        FROM Produto_pedido pp2
        JOIN Produto_ingrediente pi2 ON pp2.ce_produto = pi2.ce_produto
                    JOIN Ingrediente i2 ON pi2.ce_ingrediente = i2.cp_id_ingrediente
        WHERE pp2.ce_pedido = p.cp_id_pedido
        AND i2.tipo = 'animal'
    )
GROUP BY
    TO_CHAR(p.data_pedido, 'YYYY-MM')  -- Agrupa por mês
ORDER BY
    mes DESC;  -- Ordena para mostrar o mês mais recente primeiro
```

Pergunta analitica	Relatórios e Dashboard
PA9: Quantas pessoas que não pedem produtos com lacticinios o negócio tem atraído nos últimos meses?	Sugerido gráfico de linha ou barras para mostrar a evolução da quantidade de pessoas que fazem pedidos sem produtos de fonte animal ao longo do tempo

Query avançada equivalente:

```
--Contar os clientes que compraram pizza ou aperitivo mas nunca
    compraram produtos com ingredientes do tipo "lacticínio"
                                SELECT
 TO_CHAR(p.data_pedido, 'YYYY-MM') AS mes,  -- Agrupa por ano e mês
         COUNT(DISTINCT p.ce_cliente) AS numero_de_clientes
                                FROM
                                Pedido p
                                JOIN
         Produto_pedido pp ON p.cp_id_pedido = pp.ce_pedido
                                JOIN
           Produto pr ON pp.ce_produto = pr.cp_id_produto
                                JOIN
     Produto_ingrediente pi ON pr.cp_id_produto = pi.ce_produto
                                JOIN
      Ingrediente i ON pi.ce_ingrediente = i.cp_id_ingrediente
                                WHERE
     p.status_pedido = 'Concluído'  -- Considera apenas pedidos
                              concluídos
 AND pr.tipo_produto IN ('pizza', 'aperitivo')  -- Filtro para pizza
                             ou aperitivo
     AND i.tipo != 'lacticinio'  -- Exclui ingredientes do tipo
                            "lacticinio"
                              GROUP BY
        TO_CHAR(p.data_pedido, 'YYYY-MM')  -- Agrupa por mês
                                HAVING
  NOT EXISTS (  -- Filtra clientes que nunca compraram "lacticinio"
                                  SELECT 1
                            FROM Produto_pedido pp2
          JOIN Produto pr2 ON pp2.ce_produto = pr2.cp_id_produto
```

```
            JOIN Produto_ingrediente pi2 ON pr2.cp_id_produto =
                                pi2.ce_produto
                        JOIN Ingrediente i2 ON pi2.ce_ingrediente =
                                i2.cp_id_ingrediente
                            WHERE pp2.ce_pedido = p.cp_id_pedido
                                AND i2.tipo = 'lacticinio'
                                        )
                                    ORDER BY
    mes DESC;  -- Ordena por mês em ordem decrescente (últimos meses
                                    primeiro)
```

Pergunta analitica	Relatórios e Dashboard
PA10: Quais são os produtos e clientes pelos quais determinado funcionário está responsável agora mesmo?	Sugerida exibição em tabela para visualizar os dados dos produtos e clientes.

Query avançada equivalente:

```
--Query para definir dados dos produtos, endereço e nome dos clientes
para cada pedido em aberto que um funcionário especifico está
responsável

SELECT

    p.cp_id_pedido,

    p.data_pedido,

    pr.nm_produto,

    pp.qtd_compradas,

    pr.preco_produto,

    c.nm_cliente,

    c.endereco,
```

```
    f.nm_funcionario,

    p.status_pedido

FROM Pedido p

JOIN Produto_pedido pp ON p.cp_id_pedido = pp.ce_pedido

JOIN Produto pr ON pp.ce_produto = pr.cp_id_produto

JOIN Cliente c ON p.ce_cliente = c.cp_id_cliente

JOIN Funcionario_pedido fp ON p.cp_id_pedido = fp.ce_pedido

JOIN Funcionario f ON fp.ce_funcionario = f.cp_id_funcionario

WHERE p.status_pedido = 'Em aberto'

  AND f.nm_funcionario = 'Robespierre';
```

4.3 Stored Procedures

```
--Procedure a ser utilizada sempre que adicionarem algo no carrinho

CREATE OR REPLACE PROCEDURE sp_atualizar_valor_total_pedido(

    p_id_pedido INT

)

LANGUAGE plpgsql

AS $$

BEGIN

-- Atualizar o valor_total do pedido com base no preço, quantidade
comprada e porcentagem de promoção dos produtos

    WITH TotalPedido AS (

        SELECT pp.ce_pedido,

                    SUM((pr.preco_produto * pp.qtd_compradas) * (1 -
COALESCE(pr.porcentagem_promoção, 0) / 100)) AS novo_valor_total
```

```
        FROM Produto_pedido pp
        JOIN Produto pr ON pp.ce_produto = pr.cp_id_produto
        WHERE pp.ce_pedido = p_id_pedido
        GROUP BY pp.ce_pedido
    )
    UPDATE Pedido p
    SET valor_total = tp.novo_valor_total
    FROM TotalPedido tp
    WHERE p.cp_id_pedido = tp.ce_pedido;

END;
$$;
--CALL sp_atualizar_valor_total_pedido(<id_do_pedido>);
```

```
--Aqui tem a procedure que o funcionário mais precisa para exercer seu trabalho
--Procedure para definir dados dos produtos, endereço e nome dos clientes para cada pedido em aberto que um funcionário específico está responsável
CREATE OR REPLACE PROCEDURE sp_pedidos_em_aberto_por_funcionario(
    p_id_funcionario INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Consulta para listar os pedidos em aberto que o funcionário está responsável
```

```
    SELECT
        p.cp_id_pedido,
        p.data_pedido,
        pr.nm_produto,
        pp.qtd_compradas,
        pr.preco_produto,
        c.nm_cliente,
        c.endereco,
        f.nm_funcionario,
        p.status_pedido
    FROM Pedido p
    JOIN Produto_pedido pp ON p.cp_id_pedido = pp.ce_pedido
    JOIN Produto pr ON pp.ce_produto = pr.cp_id_produto
    JOIN Cliente c ON p.ce_cliente = c.cp_id_cliente
    JOIN Funcionario_pedido fp ON p.cp_id_pedido = fp.ce_pedido
    JOIN Funcionario f ON fp.ce_funcionario = f.cp_id_funcionario
    WHERE p.status_pedido = 'Em aberto'
      AND f.cp_id_funcionario = p_id_funcionario;
END;
$$;
--CALL sp_pedidos_em_aberto_por_funcionario(<id_do_funcionario>);
```

4.4.Materialized views

```sql
-- Esta view pode ser usada para ver rapidamente o total de compras e o valor total gasto por cada cliente.

CREATE MATERIALIZED VIEW mv_historico_compras_cliente AS

SELECT

    c.cp_id_cliente,

    c.nm_cliente,

    COUNT(p.cp_id_pedido) AS total_pedidos,

    SUM(p.valor_total) AS total_gasto

FROM

    Cliente c

JOIN

    Pedido p ON c.cp_id_cliente = p.ce_cliente

WHERE

        p.status_pedido  =  'Concluído'    --  Considera  apenas  pedidos concluídos

GROUP BY

    c.cp_id_cliente, c.nm_cliente;



--REFRESH MATERIALIZED VIEW mv_historico_compras_cliente;
```

```sql
--Materialized View para visualizar todos os pedidos realizados, seus produtos associados, e o status do pedido e entrega

CREATE MATERIALIZED VIEW mv_pedidos_status_entrega AS

SELECT

    p.cp_id_pedido,

    p.data_pedido,
```

```
    p.status_pedido,

    p.valor_total,

    c.nm_cliente,

    STRING_AGG(pr.nm_produto, ', ') AS produtos,

    f1.nm_funcionario AS entregador,

    f2.nm_funcionario AS cozinheiro,

    CASE

        WHEN p.status_pedido = 'Em aberto' THEN 'Aguardando'

        WHEN p.status_pedido = 'Concluído' THEN 'Finalizado'

        WHEN p.status_pedido = 'Cancelado' THEN 'Cancelado'

                    WHEN   f1.status_funcionario   =   'Disponível'   AND
f2.status_funcionario = 'Disponível' THEN 'Em Preparação'

        ELSE 'Em Entrega'

    END AS status_entrega

FROM

    Pedido p

JOIN

    Cliente c ON p.ce_cliente = c.cp_id_cliente

JOIN

    Produto_pedido pp ON pp.ce_pedido = p.cp_id_pedido

JOIN

    Produto pr ON pp.ce_produto = pr.cp_id_produto

LEFT JOIN

       Funcionario_pedido  fp1  ON  fp1.ce_pedido  =  p.cp_id_pedido  AND
fp1.papel = 'Entregador'

LEFT JOIN
```

```
    Funcionario f1 ON f1.cp_id_funcionario = fp1.ce_funcionario

LEFT JOIN

       Funcionario_pedido fp2 ON fp2.ce_pedido = p.cp_id_pedido AND
fp2.papel = 'Cozinheiro'

LEFT JOIN

    Funcionario f2 ON f2.cp_id_funcionario = fp2.ce_funcionario

GROUP BY

       p.cp_id_pedido, p.data_pedido, p.status_pedido, p.valor_total,
c.nm_cliente, f1.nm_funcionario, f2.nm_funcionario;

--REFRESH MATERIALIZED VIEW mv_pedidos_status_entrega;
```

5. Política de backup

Esta política visa garantir que o banco de dados esteja adequadamente protegido e que os dados possam ser recuperados em caso de falhas, com backups regulares realizados de forma automatizada. Serão utilizados backups completos e incrementais para otimizar o uso de armazenamento e o tempo de execução.

Plano de Backup

- **Tipo de backup**:
  - **Full Backup (semanal)**: Uma cópia completa do banco de dados será realizada semanalmente.
  - **Incremental Backup (diário)**: Cópias dos dados modificados desde o último backup completo ou incremental serão realizadas diariamente (de segunda a sábado).
- **Frequência**:
  - **Full Backup**: Realizado aos **domingos às 2h da manhã**.
  - **Incremental Backup**: Realizado de **segunda a sábado às 2h da manhã**.
- **Local de Armazenamento**:
  - Armazenamento local em **disco rígido externo** ou um diretório dedicado no servidor, preferencialmente em um disco diferente daquele onde o banco de dados é hospedado.
- **Ferramentas**:
  - **pg_dump**: Usado para realizar o backup completo.

- **pg_basebackup**: Usado para realizar o backup incremental.

Scripts de Backup

Os scripts de backup em python podem ser encontrados no link abaixo:

https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/tree/main/Politica%20de%20backup

Para agendar o backup devem ser utilizados o Cron no linux ou o agendador de tarefas no Windows.

6. Política de acesso e privacidade

A politica de acesso e privacidade pode ser encontrada no link abaixo:

https://github.com/tarcioee/MATA60---Pizzaria-da-Vivi/tree/main/Acesso%20e%20privacidade