NOME: Mateus da Cruz Esposte **NUSP**: 13862650

NOME: Lucca Ishikawa Ribeiro **NUSP**: 14760172

NOME: Rodrigo Kenji Sagara Nishimi **NUSP**: 14749001

ACH2004 – BANCOS DE DADOS 1 – 2024

ESPECIFICAÇÃO DA PARTE FINAL TRABALHO

Consultas no SQL:

Cálculo dos ganhos: calcula o valor total recebido pela pizzaria, mas sem levar em conta qualquer tipo de desconto. Possui filtro que realiza as operações separadas por mês e ano.

```
def get_ganho(self, mes: str, ano: str):
    query = """
            SELECT
                COALESCE(SUM(pp.Quantidade_PIZZA * p.PRECO_PIZZA), 0) +
                COALESCE(SUM(ap.Quantidade_ACOMP * a.PRECO_ACOMP), 0) AS ganho_total
            FROM PEDIDO ped
            LEFT JOIN PIZZA_PEDIDO pp ON ped.ID_PEDIDO = pp.ID_PEDIDO
            LEFT JOIN PIZZA p ON pp.ID_PIZZA = p.ID_PIZZA
            LEFT JOIN ACOMPANHAMENTO_PEDIDO ap ON ped.ID_PEDIDO = ap.ID_PEDIDO
            LEFT JOIN ACOMPANHAMENTO a ON ap.ID_ACOMP = a.ID_ACOMP
            WHERE ped.status_pedido != 'Cancelado'
            """

    if mes:
        query += f"AND EXTRACT(MONTH FROM ped.data_pedido) = '{mes}'"

    if ano:
        query += f"AND EXTRACT(YEAR FROM ped.data_pedido) = '{ano}'"

    resultado =  self.db.execute_select_all(query)
    ganho_dict = resultado[0]
    return float(ganho_dict['ganho_total'])
```

Cálculo dos Gastos com Salário: Calcula quanto foi gasto com o salário dos entregadores.

```
def get_gastos_salarios(self, mes: str, ano: str):
    query = """
            SELECT COALESCE(SUM(e.SALARIO), 0) AS gastos_salario

            FROM ENTREGADOR e
```

Cálculo dos Gastos com Acompanhamento: calcula a soma dos gastos com cada acompanhamento presente nos pedidos de um determinado mês/ano.

```
def get_gastos_acomp(self, mes: str, ano: str):
    query = """
            SELECT COALESCE(SUM(fa.PRECO_FORNECEDOR * acp.Quantidade_ACOMP), 0) AS gasto_acomp

            FROM PEDIDO p

            JOIN ACOMPANHAMENTO_PEDIDO acp ON p.ID_PEDIDO = acp.ID_PEDIDO
            JOIN ACOMPANHAMENTO ap ON acp.ID_ACOMP = ap.ID_ACOMP
            JOIN FORNECE_ACOMP fa ON fa.ID_ACOMP = ap.ID_ACOMP

            WHERE p.status_pedido != 'Cancelado'
            """

    if mes:
        query += f"AND EXTRACT(MONTH FROM p.data_pedido) = '{mes}'"

    if ano:
        query += f"AND EXTRACT(YEAR FROM p.data_pedido) = '{ano}'"

    resultado =  self.db.execute_select_all(query)
    gasto_dict = resultado[0]
    return float(gasto_dict['gasto_acomp'])
```

Cálculo dos Gastos com Ingredientes: calcula a soma dos gastos com cada ingrediente utilizados nas pizzas dos pedidos de determinado mês/ano.

```
def get_gastos_ing(self, mes: str, ano: str):
    query = """
            SELECT COALESCE(SUM(pi.QUANTIDADE_INGREDIENTE * i.PRECO_UNIDADE * pp.Quantidade_PIZZA), 0) AS gastos_ingrediente

            FROM PEDIDO p

            JOIN PIZZA_PEDIDO pp ON pp.ID_PEDIDO = p.ID_PEDIDO
            JOIN PIZZA_INGREDIENTE pi ON pp.ID_PIZZA = pi.ID_PIZZA
            JOIN INGREDIENTE i ON pi.ID_INGREDIENTE = i.ID_INGREDIENTE

            WHERE p.status_pedido != 'Cancelado'
            """

    if mes:
        query += f"AND EXTRACT(MONTH FROM p.data_pedido) = '{mes}'"

    if ano:
        query += f"AND EXTRACT(YEAR FROM p.data_pedido) = '{ano}'"

    resultado = self.db.execute_select_all(query)
    gasto_dict = resultado[0]
    return float(gasto_dict['gastos_ingrediente'])
```

Total de Clientes: Retorna a quantidade total de clientes que fizeram pedidos não cancelados, podendo filtrar essas quantidades por meses ou anos.

```
def get_nclientes(self, mes: str, ano: str):
    query = """
            SELECT COUNT(DISTINCT CPF_CLIENTE) AS num_clientes
            FROM PEDIDO ped
            WHERE ped.status_pedido != 'Cancelado'
            """
    if mes:
        query += f"AND EXTRACT(MONTH FROM ped.data_pedido) = '{mes}'"

    if ano:
        query += f"AND EXTRACT(YEAR FROM ped.data_pedido) = '{ano}'"

    resultado =  self.db.execute_select_all(query)
    gasto_dict = resultado[0]
    return int(gasto_dict['num_clientes'])
```

Total de Pedidos: Retorna a quantidade total de pedidos que foram feitos e não cancelados, podendo filtrar essas quantidades por meses ou anos.

```
def get_npedidos(self, mes: str, ano: str):
    query = """
            SELECT COUNT(*) AS num_pedidos
            FROM PEDIDO ped
            WHERE ped.status_pedido != 'Cancelado'
            """
    if mes:
        query += f"AND EXTRACT(MONTH FROM ped.data_pedido) = '{mes}'"

    if ano:
        query += f"AND EXTRACT(YEAR FROM ped.data_pedido) = '{ano}'"

    resultado =  self.db.execute_select_all(query)
    gasto_dict = resultado[0]
    return int(gasto_dict['num_pedidos'])
```

Tabela dos Pedidos: Retorna os dados de todos os pedidos e dados dos clientes. Possui filtros de busca por id, cpf, status do pedido e ordena por data do pedido.

```
def get_pedido(self, idpedido : str, cliente : str, status : str, order : str, count : str):
    query = f"""
            SELECT p.id_pedido, c.cpf_cliente, c.nome_cliente, c.telefone_cliente,
            p.endereco_pedido, p.data_pedido, p.status_pedido, p.cpf_entregador
            FROM pedido p LEFT JOIN cliente c on c.cpf_cliente = p.cpf_cliente
            """
    if idpedido:
        query+= f"WHERE p.id_pedido = {idpedido}\n"
    if cliente:
        if "WHERE" in query:
            query+= f"AND p.cpf_cliente = '{cliente}'\n"
        else:
            query+= f"WHERE p.cpf_cliente = '{cliente}'\n"
    if status:
        if "WHERE" in query:
            query+= f"AND p.status_pedido = '{status}'\n"
        else:
            query+= f"WHERE p.status_pedido = '{status}'\n"
    if order:
        query+= f"ORDER BY p.data_pedido {order}\n"

    return self.db.execute_select_all(query)
```

Detalhes dos Acompanhamento de um Pedido: Retorna os acompanhamentos de um pedido de id específico.

```
# Acompanhamentos do pedido
def get_detalhe_pedido_a(self, idpedido : str):
    query = """
            SELECT nome_acomp, tipo_acomp, quantidade_acomp, (preco_acomp * quantidade_acomp) AS preco_acomps
            FROM pedido p
            LEFT JOIN acompanhamento_pedido ap on ap.id_pedido = p.id_pedido
            LEFT JOIN acompanhamento a on a.id_acomp = ap.id_acomp
            """
    if idpedido:
        query+= f"WHERE p.id_pedido = {idpedido}\n"

    return (self.db.execute_select_all(query))
```

Detalhes das Pizzas de um Pedido: Retorna as pizzas de um pedido de id específico.

```
# Pizzas do pedido
def get_detalhe_pedido_p(self, idpedido : str):
    query = """
            SELECT nome_pizza, quantidade_pizza, (preco_pizza * quantidade_pizza) AS preco_pizzas
            FROM pedido p
            LEFT JOIN pizza_pedido pzp on pzp.id_pedido = p.id_pedido
            LEFT JOIN pizza pz on pz.id_pizza = pzp.id_pizza
            """
    if idpedido:
        query+= f"WHERE p.id_pedido = {idpedido}\n"

    return (self.db.execute_select_all(query))
```

Calcula o Preço do Pedido: Retorna o preço total de todos os pedidos individualmente.

```python
#Gráfico de vendas
def get_preco_pedido(self):
    query = """
            SELECT p.id_pedido, data_pedido,
            COALESCE(SUM(pp.quantidade_pizza * pi.preco_pizza), 0) + COALESCE(SUM(ap.quantidade_acomp * ac.preco_acomp), 0) AS preco_pedido
            FROM pedido p
            LEFT JOIN pizza_pedido pp ON pp.id_pedido = p.id_pedido
            LEFT JOIN pizza pi ON pi.id_pizza = pp.id_pizza
            LEFT JOIN acompanhamento_pedido ap ON ap.id_pedido = p.id_pedido
            LEFT JOIN acompanhamento ac ON ac.id_acomp = ap.id_acomp
            GROUP BY p.id_pedido;
            """
    return self.db.execute_select_all(query)
```

Dados do Fornecedor: Devolve uma tabela com os dados do fornecedor e o número de ingredientes e acompanhamentos que ele fornece.

```python
def get_fornecedor(self, nome: str, cnpj: str):
    query = f"""
            SELECT f.*, (COUNT(fi.id_ingrediente) + COUNT(fa.id_acomp)) AS num_produtos
            FROM fornecedor f
            LEFT JOIN fornece_ingrediente fi ON fi.cnpj_fornecedor = f.cnpj_fornecedor
            LEFT JOIN fornece_acomp fa ON fa.cnpj_fornecedor = f.cnpj_fornecedor
            GROUP BY f.cnpj_fornecedor
            """
    if nome:
        query+= f"HAVING f.nome_fornecedor LIKE '%{nome}%'\n"
    if cnpj:
        if 'HAVING' in query:
            query+= f"AND f.cnpj_fornecedor = '{cnpj}'\n"
        else:
            query+= f"HAVING f.cnpj_fornecedor = '{cnpj}'\n"

    return self.db.execute_select_all(query)
```

Estoque de Acompanhamento: Faz uma tabela com todos os Acompanhamentos e suas quantidades em estoque. Além de aplicar filtros por nome ou ordenar por preço/quantidade.

```python
#Estoque de Acompanhamentos
def get_estoque_a(self, nome: str, preco: str, qtde: str):
    query = f"""
            SELECT a.*, (SUM(quantidade_fornecedor) - SUM(ap.quantidade_acomp)) AS quantidade_total
            FROM acompanhamento a
            JOIN fornece_acomp fa ON fa.id_acomp = a.id_acomp
            JOIN acompanhamento_pedido ap ON ap.id_acomp = a.id_acomp
            GROUP BY a.id_acomp, a.nome_acomp
            """
    if nome:
        query += f"HAVING a.nome_acomp ILIKE '%{nome}%'\n"
    if preco:
        query += f"ORDER BY a.preco_acomp {preco}\n"
    if qtde:
        query += f"ORDER BY quantidade_total {qtde}\n"

    return self.db.execute_select_all(query)
```

Estoque de Ingredientes: Faz uma tabela com todos os Ingredientes e suas quantidades em estoque. Além de aplicar filtros por nome ou ordenar por preço/quantidade.

```python
#Estoque de Ingredientes
def get_estoque_i(self, nome: str, preco: str, qtde: str):
    query = f"""
            SELECT i.*, (SUM(quantidade_fornecedor) - (SELECT SUM(pp.quantidade_pizza * pi.quantidade_ingrediente)
            FROM pizza_pedido pp
            JOIN pizza_ingrediente pi ON pp.id_pizza = pi.id_pizza
            WHERE pi.id_ingrediente = i.id_ingrediente)) AS quantidade_total
            FROM ingrediente i
            LEFT JOIN fornece_ingrediente fi ON fi.id_ingrediente = i.id_ingrediente
            GROUP BY i.id_ingrediente, i.nome_ingrediente
            """
    if nome:
        query += f"HAVING i.nome_ingrediente ILIKE '%{nome}%'\n"
    if preco:
        query += f"ORDER BY i.preco_unidade {preco}\n"
    if qtde:
        query += f"ORDER BY quantidade_total {qtde}\n"

    return self.db.execute_select_all(query)
```

Fornecedores de Acompanhamento: Retorna uma tabela com todos os fornecedores de um Acompanhamento específico, filtrado por id. Além disso, os fornecedores podem ser filtrados pelo CNPJ e ordenados pelos preços que fornecem o acompanhamento.

```python
def get_fornecedores_acomp(self, id_acomp: str, preco: str, cnpj: str):
    query = f"""
            SELECT a.id_acomp, a.nome_acomp, a.tipo_acomp, f.*, fa.quantidade_fornecedor,
            fa.preco_fornecedor, (fa.preco_fornecedor * fa.quantidade_fornecedor) AS preco_total
            FROM acompanhamento a
            JOIN fornece_acomp fa ON fa.id_acomp = a.id_acomp
            JOIN fornecedor f ON f.cnpj_fornecedor = fa.cnpj_fornecedor
            """
    if id_acomp:
        query += f"WHERE a.id_acomp = {id_acomp}\n"
    if cnpj:
        if 'WHERE' in query:
            query+= f"AND fa.cnpj_fornecedor = '{cnpj}'\n"
        else:
            query+= f"WHERE fa.cnpj_fornecedor = '{cnpj}'\n"
    if preco:
        query += f"ORDER BY preco_fornecedor {preco}\n"

    return self.db.execute_select_all(query)
```

Fornecedores de Ingrediente: Retorna uma tabela com todos os fornecedores de um Ingrediente específico, filtrado por id. Além disso, os fornecedores podem ser filtrados pelo CNPJ e ordenados pelos preços que fornecem o ingrediente.

```
def get_fornecedores_ingred(self, id_ingrediente: str, preco: str, cnpj: str):
    query = f"""
            SELECT i.id_ingrediente, i.nome_ingrediente, f.*, fi.quantidade_fornecedor,
            fi.preco_fornecedor, (fi.preco_fornecedor * fi.quantidade_fornecedor) AS preco_total
            FROM ingrediente i
            JOIN fornece_ingrediente fi ON fi.id_ingrediente = i.id_ingrediente
            JOIN fornecedor f ON f.cnpj_fornecedor = fi.cnpj_fornecedor
            """
    if id_ingrediente:
        query += f"WHERE i.id_ingrediente = {id_ingrediente}\n"
    if cnpj:
        if 'WHERE' in query:
            query+= f"AND fi.cnpj_fornecedor = '{cnpj}'\n"
        else:
            query+= f"WHERE fi.cnpj_fornecedor = '{cnpj}'\n"
    if preco:
        query += f"ORDER BY preco_fornecedor {preco}\n"

    return self.db.execute_select_all(query)
```

Ordena e Filtra os Entregadores: Cria uma tabela com todos os entregadores empregados, podendo os ordenar pelos seus salários ou buscar pelos seus CPF.

```
def get_entregador(self, cpf: str, salario: str):
    query = f"""
            SELECT e.*, COUNT(p.cpf_entregador) AS num_entregas
            FROM entregador e
            JOIN pedido p ON e.cpf_entregador = p.cpf_entregador
            GROUP BY e.cpf_entregador
            """
    if cpf:
        query += f"HAVING e.cpf_entregador = '{cpf}'"
    if salario:
        query += f"ORDER BY e.salario {salario}\n"

    return self.db.execute_select_all(query)
```

Associa Pedidos a Entregadores: Retorna os pedidos entregados por um entregador específico, filtrado por cpf.

```python
def get_pedidos_entregador(self, cpf: str):
    query = f"""
            SELECT p.id_pedido, p.cpf_cliente, p.data_pedido, p.endereco_pedido, p.status_pedido
            FROM pedido p, entregador e
            WHERE p.cpf_entregador = e.cpf_entregador
            """
    if cpf:
        query += f"AND e.cpf_entregador = '{cpf}'"

    return self.db.execute_select_all(query)
```

Tabela de Clientes: Cria uma tabela com as informações dos clientes cadastrados e seu número de pedidos realizados.

```python
# TABELA CLIENTE
def get_cliente(self, nome : str, cpf : str, order : str):
    query = f"""
            SELECT c.*, COUNT(p.id_pedido) as num_pedidos
            FROM cliente c LEFT JOIN pedido p on c.cpf_cliente = p.cpf_cliente
            GROUP BY c.cpf_cliente
            """
    if nome:
        query+= f"HAVING c.nome_cliente ILIKE '%{nome}%'\n"
    if cpf:
        if "HAVING" in query:
            query+= f"AND c.cpf_cliente = '{cpf}'\n"
        else:
            query+= f"HAVING c.cpf_cliente = '{cpf}'\n"
    if order:
        query+= f"ORDER BY num_pedidos {order}\n"

    return self.db.execute_select_all(query)
```

Detalhes dos Clientes: Retorna os pedidos de um determinado cliente, filtrado por CPF e os pedidos podem ser ordenados por data do pedido.

```python
def get_detalhe_cliente(self, cpf : str, order : str):
    query = f"""
            SELECT c.nome_cliente, p.id_pedido, p.endereco_pedido, p.data_pedido, p.status_pedido, p.cpf_entregador
            FROM pedido p LEFT JOIN cliente c on c.cpf_cliente = p.cpf_cliente
            """
    if cpf:
        query+= f"WHERE c.cpf_cliente = {cpf}\n"
    if order:
        query+= f"ORDER BY p.data_pedido {order}\n"

    return self.db.execute_select_all(query)
```

Descontos dos clientes: Atualiza o número de cupons de desconto que um cliente possui. A cada 5 pedidos, o cliente recebe 1 cupom. A consulta retorna o número de pedidos do cliente dividido por 5, que é o valor do número de cupons de desconto que o cliente tem. Essa alteração ocorre apenas nos clientes que possuem 5 ou mais pedidos.

```
def add_cupom(self):
    statement = f"""
            UPDATE cliente SET cupon_desconto = (SELECT (COUNT(*) / 5)
            FROM pedido WHERE pedido.cpf_cliente = cliente.cpf_cliente)
            WHERE cpf_cliente IN (SELECT cpf_cliente FROM pedido GROUP BY cpf_cliente HAVING COUNT(*) >= 5)
            """
    return self.db.execute_statement(statement)
```

Dados das pizzas: Retorna os dados de todas as pizzas, quantidade vendida e ganho. As pizzas podem ser filtradas por nome e ordenadas por quantidade vendida ou pelas mais lucrativas.

```
def get_pizza(self, nome:str, qtde: str, vendas: str):
    query = f"""
            SELECT p.*, SUM(pp.quantidade_pizza) AS quantidade_vendida,
            (SUM(pp.quantidade_pizza) * p.preco_pizza) AS total_vendido
            FROM pizza p
            LEFT JOIN pizza_pedido pp ON pp.id_pizza = p.id_pizza
            JOIN pedido pe ON pe.id_pedido = pp.id_pedido
            WHERE pe.status_pedido != 'Cancelado'
            GROUP BY p.id_pizza, p.nome_pizza
            """
    if nome:
        query += f"HAVING p.nome_pizza ILIKE '%{nome}%'\n"
    if qtde:
        query += f"ORDER BY quantidade_vendida {qtde}\n"
    if vendas:
        query += f"ORDER BY total_vendido {vendas}\n"

    return self.db.execute_select_all(query)
```

Detalhes da Pizza: Retorna os ingredientes de uma determinada pizza e seus detalhes, como preço e quantidade.

```
def get_detalhe_pizza(self, id_pizza:str):
    query = f"""
            SELECT p.nome_pizza, p.preco_pizza, i.nome_ingrediente, pi.quantidade_ingrediente,
            i.preco_unidade, (pi.quantidade_ingrediente * i.preco_unidade) AS preco_total
            FROM pizza p
            JOIN pizza_ingrediente pi ON pi.id_pizza = p.id_pizza
            JOIN ingrediente i ON i.id_ingrediente = pi.id_ingrediente
            """
    if id_pizza:
        query += f"WHERE p.id_pizza = {id_pizza}\n"

    return self.db.execute_select_all(query)
```