**1. Estrutura proposta para o banco de dados:**

**1.1 Tabela Pedidos**

* + **Campos: id\_pedido (chave primária), id\_cliente (chave estrangeira), data\_pedido, status.**
  + **Relacionamento: Um cliente pode fazer vários pedidos (1:N entre Clientes e Pedidos).**
  + **Justificativa: O campo id\_cliente conecta os pedidos ao cliente correspondente, garantindo rastreabilidade. Índices nas chaves estrangeiras são utilizados para melhorar o desempenho das consultas.**

**1.2 Tabela Produtos**

* + **Campos: id\_produto (chave primária), nome, descricao, preco, estoque.**
  + **Justificativa: O id\_produto é único para cada item, assegurando a integridade e a identificação dos produtos.**
  1. **Tabela Clientes**
  + **Campos: id\_cliente (chave primária), nome, email, telefone, data\_criacao.**
  + **Descrição: Armazena informações detalhadas e exclusivas de cada cliente.**
  + **Justificativa: A chave primária id\_cliente é essencial para identificar cada cliente no sistema.**

**1.4 Tabela Categorias**

* + **Campos: id\_categoria (chave primária), nome.**
  + **Justificativa: O campo id\_categoria permite classificar os produtos em diferentes categorias de forma organizada e reutilizável.**

**1.5 Tabela de relação Produtos\_Categorias**

* + **Campos: id\_produto (chave estrangeira), id\_categoria (chave estrangeira).**
  + **Relacionamento: Permite uma relação de muitos para muitos entre Produtos e Categorias.**
  + **Justificativa: Esta tabela intermediária é necessária para associar múltiplas categorias a um único produto ou vice-versa.**
  1. **Tabela Pagamentos**
  + **Campos: id\_pagamento (chave primária), id\_pedido (chave estrangeira), valor, metodo\_pagamento, data\_pagamento.**
  + **Relacionamento: Cada pedido tem um único pagamento associado (1:1 entre Pedidos e Pagamentos).**
  + **Justificativa: O campo id\_pedido conecta os pagamentos ao pedido correspondente, garantindo que os registros financeiros sejam rastreados corretamente.**

Essa estrutura foi pensada para garantir integridade relacional, desempenho em consultas e flexibilidade para atender às necessidades de um sistema de e-commerce.

**2. Construção de uma API RESTful com Laravel** O código foi implementado conforme solicitado e se encontra disponível para teste. Para criação foi utilizado PHP 7.4.6 e Laravel Installer 4.5.1 e com ambos instalados basta rodar o comando php artisan migrate para criação do banco e php artisan db:seed para popular o banco com dados de teste.

Ao utilizar a rota <http://localhost:8000/api/produtos> pode-se omitir a passagem dos parâmetros, mas também pode se usar os parâmetros de preco\_min e preco\_max para filtrar entre esses preços e o parâmetro categoria passando o nome da categoria para filtrar dentre todos os produtos.

Segue abaixo um exemplo a ser utilizado na requisição utilizando a rota <http://localhost:8000/api/pedidos> para a criação de um novo pedido:

{

    "id\_cliente": 1,

    "produtos": [

        {

            "id\_produto": 101,

            "quantidade": 2

        },

        {

            "id\_produto": 102,

            "quantidade": 1

        }

    ],

    "status": "pendente"

}

Dessa forma será criado um pedido com o id do cliente e o status, sendo referenciado os produtos desse pedido na tabela pedido\_produto.

Para utilizar a rota [http://localhost:8000/api/pedidos/{id}](http://localhost:8000/api/pedidos/%7bid%7d) que faz a atualização do pedido basta enviar uma requisição igual a mencionada no item anterior porem fazendo as alterações desejadas em um pedido existente.

**3. Lógica e Otimização de Consultas SQL**

Segue algumas consultas propostas otimizadas:

**3.1 Total de pedidos e receita de cada cliente no último ano:**

SELECT c.id\_cliente, c.nome, COUNT(p.id\_pedido) AS total\_pedidos, SUM(pg.valor) AS receita\_total FROM clientes c

JOIN pedidos p ON c.id\_cliente = p.id\_cliente

JOIN pagamentos pg ON p.id\_pedido = pg.id\_pedido

WHERE p.data\_pedido >= DATE\_SUB(CURDATE(), INTERVAL 1 YEAR)

GROUP BY c.id\_cliente, c.nome

ORDER BY receita\_total DESC;

**3.2 Produtos mais vendidos por categoria:**

SELECT cat.nome AS categoria, prod.nome AS produto, SUM(pp.quantidade) AS quantidade\_total FROM categorias cat

JOIN produtos\_categorias pc ON cat.id\_categoria = pc.id\_categoria

JOIN produtos prod ON pc.id\_produto = prod.id\_produto

JOIN pedido\_produtos pp ON prod.id\_produto = pp.id\_produto

GROUP BY cat.nome, prod.nome

ORDER BY cat.nome, quantidade\_total DESC;

**3.3 Otimizações:**

**1. Índices**

Criar índices nas chaves estrangeiras (id\_cliente, id\_pedido, id\_produto) acelera consultas e junções, tornando a busca por dados mais eficiente, especialmente em tabelas grandes.

**2. Particionamento**

Particionar grandes tabelas (como pedidos e pagamentos) por data melhora a performance ao dividir os dados em partes menores, facilitando buscas rápidas nas partições relevantes.

**3. Cache**

Armazenar os resultados de consultas frequentes em memória (usando Redis ou Memcached) evita acessos repetidos ao banco de dados, acelerando o retorno dos dados.

**4. Análise com EXPLAIN**

O comando EXPLAIN permite analisar o plano de execução das consultas, ajudando a identificar gargalos e ajustar índices ou a consulta para melhorar o desempenho.