

Esquema base para atividades

```
CREATE TABLE clientes (  
    id BIGINT PRIMARY KEY AUTO_INCREMENT,  
    nome VARCHAR(120) NOT NULL,  
    email VARCHAR(160) NOT NULL UNIQUE,  
    documento CHAR(11) NOT NULL,      -- CPF sem pontuação  
    cidade VARCHAR(80),  
    estado CHAR(2),  
    criado_em DATETIME NOT NULL,  
    INDEX idx_clientes_estado (estado)  
) ENGINE=InnoDB;
```

```
CREATE TABLE produtos (  
    id BIGINT PRIMARY KEY AUTO_INCREMENT,  
    sku VARCHAR(32) NOT NULL UNIQUE,  
    nome VARCHAR(160) NOT NULL,  
    categoria VARCHAR(80),  
    preco DECIMAL(10,2) NOT NULL,  
    ativo TINYINT(1) NOT NULL DEFAULT 1,  
    FULLTEXT INDEX ftx_produtos_nome (nome) -- se usar InnoDB 5.6+ / 8.0+  
) ENGINE=InnoDB;
```

```
CREATE TABLE pedidos (  
    id BIGINT PRIMARY KEY AUTO_INCREMENT,  
    cliente_id BIGINT NOT NULL,  
    status ENUM('NOVO','PAGO','ENVIADO','CANCELADO') NOT NULL,  
    criado_em DATETIME NOT NULL,  
    total DECIMAL(12,2) NOT NULL,  
    FOREIGN KEY (cliente_id) REFERENCES clientes(id),  
    INDEX idx_pedidos_cliente (cliente_id),  
    INDEX idx_pedidos_status_criado (status, criado_em)
```

) ENGINE=InnoDB;

```
CREATE TABLE itens_pedido (  
    pedido_id BIGINT NOT NULL,  
    produto_id BIGINT NOT NULL,  
    quantidade INT NOT NULL,  
    preco_unit DECIMAL(10,2) NOT NULL,  
    PRIMARY KEY (pedido_id, produto_id),  
    FOREIGN KEY (pedido_id) REFERENCES pedidos(id),  
    FOREIGN KEY (produto_id) REFERENCES produtos(id),  
    INDEX idx_item_produto (produto_id)  
) ENGINE=InnoDB;
```

1. Crie um índice para acelerar buscas por clientes.documento.

- Objetivo: usar EXPLAIN para confirmar uso do índice em `SELECT * FROM clientes WHERE documento = '...'`.

2. Avalie a necessidade de índice em clientes.cidade.

- Tarefa: rode `SELECT ... WHERE cidade = 'Porto Alegre'` antes/depois de criar `INDEX idx_cidade (cidade)` e compare rows no EXPLAIN.

3. Índice composto para range + igualdade.

- Crie `INDEX idx_pedidos_cliente_criado (cliente_id, criado_em)` e otimize consultas:
`SELECT * FROM pedidos WHERE cliente_id = ? AND criado_em BETWEEN ? AND ?;`

4. Cobertura de consulta (covering index).

- Crie `INDEX idx_produtos_categoria_preco_nome (categoria, preco, nome)` e execute:
`SELECT categoria, preco, nome FROM produtos WHERE categoria = 'Notebook' AND preco < 5000;`
- Objetivo: verificar Using index no EXPLAIN.

5. Unique vs. não-unique.

- Compare `UNIQUE(email)` (já existe) com um índice normal em nome.
- Explique impacto de unicidade em escrita/leitura.

6. Quando NÃO criar índice.

- Teste SELECT ... WHERE ativo = 1 em produtos.
- Crie INDEX idx_produtos_ativo (ativo), compare EXPLAIN. Discuta baixa seletividade.

7. Prefix index em colunas grandes.

- Crie INDEX idx_clientes_nome_prefix (nome(20)) e compare com índice completo em nome para consulta LIKE 'Mar%'.

8. Otimizar LIKE com curingas.

- Compare WHERE nome LIKE 'note%' (usa índice) vs. LIKE '%book' (não usa).
- Proponha alternativa com coluna gerada reversa para sufixo (ou FULLTEXT para buscas mais flexíveis).

9. FULLTEXT em produtos.nome.

- Use MATCH(nome) AGAINST ('+gaming -used' IN BOOLEAN MODE) e compare com LIKE '%gaming%'.
- Medir relevância e desempenho.

10. Índice para ORDER BY.

- Crie INDEX idx_pedidos_criado_total (criado_em, total) e optimize SELECT id, criado_em, total FROM pedidos WHERE criado_em >= CURDATE() - INTERVAL 30 DAY ORDER BY criado_em, total;
- Objetivo: evitar filesort.

11. Índice para GROUP BY.

- Crie INDEX idx_pedidos_status (status) e avalie SELECT status, COUNT(*) FROM pedidos GROUP BY status;
- Veja se o otimizador usa índice para agrupamento.

12. Escolha da ordem em índice composto.

- Compare (status, criado_em) vs (criado_em, status) para filtros comuns:
 - a) WHERE status='PAGO' AND criado_em BETWEEN ...
 - b) WHERE criado_em BETWEEN ... apenas.
- Documente qual ordem atende melhor seu padrão.

13. Chave primária e clustered index do InnoDB.

- Explique por que PRIMARY KEY em InnoDB é o índice clustered e como isso impacta índices secundários (que carregam a PK como ponteiro).

14. Índice em FK.

- Verifique se pedidos(cliente_id) e itens_pedido(pedido_id, produto_id) têm índices adequados.
- Mostre impactos em JOIN e DELETE/UPDATE com cascata.

15. Índices “invisíveis” (MySQL 8.0+).

- Torne idx_pedidos_status_criado invisível e compare planos de execução antes/depois sem dropar o índice.
- Objetivo: testar impacto sem afetar DDL.

16. Índice funcional (expressions).

- Crie índice em expressão INDEX idx_clientes_email_lower ((LOWER(email))) e otimize:
SELECT * FROM clientes WHERE LOWER(email) = 'joao@exemplo.com';

17. Seletividade e cardinalidade.

- Use SHOW INDEX FROM ... para ver Cardinality.
- Compare seletividade de estado (baixa) vs documento (alta) e discuta impacto.

18. Index Merge e combinações.

- Crie INDEX idx_produtos_categoria (categoria) e INDEX idx_produtos_preco (preco).
- Rode WHERE categoria='Notebook' AND preco BETWEEN ... e observe se o otimizador usa **Index Merge** ou prefere índice composto.
- Conclusão: quando preferir composto vs. múltiplos índices simples.

19. Manutenção e estatísticas.

- Execute ANALYZE TABLE e OPTIMIZE TABLE (em ambiente de teste).
- Observe mudanças em planos após grande carga/remoção de dados.

20. Trade-offs de escrita vs. leitura.

- Faça um benchmark simples:
 - Sem índices extras, insira 100k linhas em produtos.
 - Com vários índices novos, repita a carga.
- Compare tempos e explique o custo de manter muitos índices nas operações de INSERT/UPDATE/DELETE.

Desafios de modelagem/consulta

- **D1)** Reprojetar índices para a consulta:

- SELECT p.id, p.criado_em, c.estado, SUM(i.quantidade*i.preco_unit) AS total
- FROM pedidos p
- JOIN clientes c ON c.id = p.cliente_id
- JOIN itens_pedido i ON i.pedido_id = p.id
- WHERE c.estado = 'RS'
- AND p.criado_em >= CURDATE() - INTERVAL 90 DAY
- GROUP BY p.id, p.criado_em, c.estado
- ORDER BY p.criado_em DESC
- LIMIT 50;
 - Objetivo: propor índices mínimos para reduzir leituras e evitar filesort.
- **D2)** Planejar índices para relatórios por mês/ano.
 - Criar coluna gerada ano_mes (YYYYMM) e índice nela; comparar com função em coluna não indexada (DATE_FORMAT no WHERE).