EXERCÍCIO COMPLETO — NOSQL (GABARITO DETALHADO)

BLOCO 1 — Classificação de Modelos (Sugestões de Resposta)

1. Sistema de sessões de usuários.

→ Modelo: Chave-Valor (ex.: Redis). Justificativa: leituras e escritas muito rápidas, dados simples (id_sessao → dados da sessão).

2. Catálogo de produtos variados.

→ Modelo: Documento (ex.: MongoDB). Justificativa: diferentes produtos têm atributos distintos; JSON flexível evita tabelas enormes e muitos campos nulos.

3. Timeline global com bilhões de eventos.

→ Modelo: Colunar (ex.: Cassandra). Justificativa: escrita massiva distribuída, leitura por usuário ordenada por tempo, alta disponibilidade.

4. Rede social com conexões.

→ Modelo: Grafos (ex.: Neo4j). Justificativa: foco em relacionamentos (quem segue quem, graus de conexão, caminhos).

5. Logs distribuídos massivos.

→ Modelo: Colunar ou Documento. Preferencial: Colunar (HBase/Cassandra) para grandes volumes e leituras analíticas.

6. Notificações instantâneas.

→ Modelo: Chave-Valor. Justificativa: baixa latência, chave=usuário, valor=lista de notificações recentes.

7. Carrinho temporário.

→ Modelo: Chave-Valor. Justificativa: dados voláteis, pequenos, associados ao usuário, precisam de acesso rápido.

8. Histórico de transações.

→ Modelo: Colunar (para analytics) ou Relacional (para controle rígido). No contexto NoSQL, Colunar para relatórios massivos.

9. Chat com mensagens aninhadas.

→ Modelo: Documento. Justificativa: cada conversa pode ser um documento com array de mensagens, anexos, metadados.

10. Detecção de fraude conectando contas.

→ Modelo: Grafos. Justificativa: analisar conexões entre contas, dispositivos, IPs, padrões de fluxo.

BLOCO 2 — CAP + BASE vs ACID (Sugestões de Resposta)

Para cada cenário: escolha CP ou AP, e ACID ou BASE.

1. PIX.

→ CAP: CP (Consistência + Partição).

→ Modelo: ACID.

Justificativa: transações financeiras não podem divergir; é melhor recusar do que registrar valores inconsistentes.

2. Twitter Likes.

→ CAP: AP (Disponibilidade + Partição).

→ Modelo: BASE.

Justificativa: contadores podem estar levemente desatualizados, mas o sistema deve responder sempre.

3. Uber Maps (localização em tempo real).

→ CAP: AP.

→ Modelo: BASE.

Justificativa: localização pode ter pequena defasagem; prioridade é disponibilidade e baixa latência para navegação.

4. Checkout e-commerce.

→ CAP: CP (ou forte consistência).

→ Modelo: ACID.

Justificativa: fechamento de pedido não pode falhar ou gerar inconsistência de estoque/pagamento.

5. Feed do Instagram.

→ CAP: AP.

→ Modelo: BASE.

Justificativa: feed pode demorar alguns segundos para se atualizar; o mais importante é estar sempre disponível.

BLOCO 3 — Caso MongoDB (Gabarito Exemplo)

Modelo de documento JSON para restaurante (exemplo simplificado):


```
{
"nome": "Sushi da Casa",
"categoria": ["Japonesa", "Delivery"],
"endereco": {
"rua": "Av. Central, 123",
"cidade": "Porto Alegre",
"uf": "RS",
"cep": "90000-000"
},
"geoloc": {
"type": "Point",
"coordinates": [-51.2300, -30.0200]
},
"horarios": {
"segunda_sexta": "11:00-23:00",
"sabado_domingo": "12:00-00:00"
},
"cardapio": [
{ "prato": "Sashimi", "preco": 39.90, "disponivel": true },
{ "prato": "Temaki", "preco": 25.50, "disponivel": true },
```

{ "prato": "Combinado Família", "preco": 89.90, "disponivel": false }

],

"avaliacoes": [

{ "usuario": "user123", "nota": 5, "comentario": "Excelente!", "data": "2025-11-01" },

{ "usuario": "user456", "nota": 4, "comentario": "Muito bom, mas demorou.", "data": "2025-11-03" }

],

"historico_promocoes": [

{ "descricao": "Festival de Temaki", "inicio": "2025-10-01", "fim": "2025-10-15" }

],

"tags": ["japonês", "delivery", "sushi", "rodízio"]

}

Pergunta 1: Por que MongoDB facilita esse modelo?

→ Porque permite:

- documentos com estrutura rica e aninhada (subdocumentos e arrays);

- campos opcionais sem necessidade de alterar esquema global;

- evolução do modelo sem migrações complexas;

- representação natural do domínio (restaurante e seu cardápio em um único documento).

Pergunta 2: Quais campos complicariam no relacional?

→ Exemplos:

- cardapio (arrays de pratos) → exigiria tabela separada e JOIN;

- avaliacoes (lista de avaliações) → mais uma tabela;

- historico_promocoes → outra tabela;

- tags (lista de strings) → tabela associativa.

Isso multiplicaria o número de tabelas e JOINs.

Pergunta 3: Como aplicar sharding?

→ Possibilidades:

- Shard por cidade ou região (campo endereco.cidade ou geoloc), distribuindo restaurantes por localização;

- ou shard por restaurante_id, distribuindo a carga entre nós.

Para grandes sistemas, shardar por localização facilita buscas geográficas.

BLOCO 4 — Arquitetura Mini-Twitter (Sugestão de Gabarito)

Componente: Sessões de usuário

→ Modelo: Chave-Valor (Redis).

Justificativa: dados pequenos, voláteis, acesso rápido (token → dados de sessão).

Componente: Posts (tweets)

→ Modelo: Documento (MongoDB ou similar).

Justificativa: cada tweet pode ter texto, mídia, metadados, replies; JSON é natural para isso.

Componente: Comentários/Respostas

→ Modelo: Documento.

Justificativa: estrutura similar a posts; podem ser documentos próprios referenciando o tweet original.

Componente: Timeline

→ Modelo: Colunar (Cassandra).

Justificativa: escrita massiva ordenada por usuário+tempo, leitura sequencial rápida para feed.

Componente: Likes/retweets (contadores)

→ Modelo: Chave-Valor (Redis ou DynamoDB).

Justificativa: necessidade de atualização rápida de contadores, baixa latência.

Componente: Busca (por texto, hashtags, usuários)

→ Modelo: Documento orientado a índice invertido (Elasticsearch).

Justificativa: especializado em busca full-text, filtros, relevância.

Componente: Relações (quem segue quem)

→ Modelo: Grafos (Neo4j).

Justificativa: ideal para queries de relacionamento (seguidores em comum, recomendações, graus de conexão).

Observação:

As respostas podem variar, desde que o aluno:

- justifique o modelo escolhido;

- conecte o modelo às características (escala, flexibilidade, relacionamentos, performance) apresentadas nos slides.