📖Apêndices – Seven Databases in Seven Weeks

A1 – Database Overview Tables

Este apêndice resume os bancos de dados apresentados no livro, destacando suas características principais.

Banco de Dados	Gênero	Pontos Fortes	Casos de Uso Ideais
PostgreSQL	Relacional	Consistência, integridade, consultas complexas	ERPs, finanças, sistemas corporativos
HBase	Colunar	Escala massiva, integração com Hadoop	Big Data, logs, telemetria
MongoDB	Documentos	Flexível, consultas ricas, escalabilidade	Web, APIs, aplicações modernas
CouchDB	Documentos	Replicação, sincronização offline	Mobile, redes instáveis, sistemas distribuídos
Neo4j	Grafos	Relacionamentos nativos, consultas rápidas	Redes sociais, recomendação, logística
DynamoDB	Chave-valor/ Documento	Escalabilidade serverless, integração AWS	E-commerce, IoT, cloud-native apps
Redis	In-memory/ estruturas de dados	Altíssima performance, cache, Pub/Sub	Sessões, ranking, mensageria, cache

📌 Essa tabela mostra como **cada banco é otimizado para um tipo de problema**, justificando a diversidade do ecossistema NoSQL.

A2 – O Teorema CAP

O **Teorema CAP**, proposto por Eric Brewer, afirma que em sistemas distribuídos é impossível garantir simultaneamente as três propriedades abaixo:

1. **Consistência (C)** → todos os nós veem os mesmos dados ao mesmo tempo.
2. **Disponibilidade (A)** → o sistema sempre responde, mesmo em caso de falha.
3. **Tolerância a Partição (P)** → o sistema continua funcionando mesmo com falhas de rede.

⚖️Trade-off

- É possível garantir **apenas dois** dos três atributos ao mesmo tempo.
- Exemplo: em caso de partição de rede, deve-se escolher entre consistência ou disponibilidade.

Eventual Consistency

- Muitos bancos (como DynamoDB, CouchDB, Cassandra) oferecem **consistência eventual**.
- Isso significa que, após algum tempo, todos os nós convergem para o mesmo estado.

CAP na Prática

- **PostgreSQL** → favorece consistência (C) e disponibilidade (A), mas não é tolerante a partições por si só.
- **HBase** → prioriza consistência (C) e tolerância a partição (P).
- **MongoDB** → em **replica sets**, tende a **CP (Consistência + Partição)**: apenas o primário aceita escritas; sob partição sem maioria, o cluster recusa escritas até nova eleição, sacrificando disponibilidade. Leituras em secundários são **eventualmente consistentes**, mas é possível exigir consistência forte no primário com `writeConcern: { w: "majority" }` e `readConcern: "linearizable"`. Em **sharding**, cada shard mantém as mesmas propriedades (CP por shard).
- **CouchDB/DynamoDB** → favorecem disponibilidade (A) e tolerância a partição (P), com consistência eventual.
- **Redis Cluster** → pode ser configurado para diferentes equilíbrios entre C, A e P.

Latência e CAP

- Um aspecto importante é a **latência**. Quanto mais consistência exigida, maior pode ser a latência.
- Em contrapartida, sistemas que aceitam consistência eventual conseguem respostas mais rápidas e maior disponibilidade.

🔗Conclusão dos Apêndices

- O **Apêndice A1** mostrou uma visão panorâmica das forças e usos ideais de cada banco.
- O **Apêndice A2** reforçou que todo sistema distribuído exige **escolhas conscientes** entre consistência, disponibilidade e tolerância a partição.

📌O aprendizado final: **não existe um banco perfeito, mas sim ferramentas certas para diferentes contextos**.