**Ch 2**

**Data Models and Query Languages**

**1. Core Ideas**

* A **data model** shapes how software is written & how we think about the problem.
* Apps usually **layer multiple models**:
  + Real-world entities → app-level data structures → database representation (JSON, SQL tables, graph) → bytes on disk.
* Choice of data model affects:
  + Which queries are fast/slow.
  + Simplicity vs complexity of transformations.
  + Flexibility in evolving schemas.

**2. Main Models**

**Relational Model**

* Organizes data into **relations (tables)**, each row = tuple, each column = attribute.
* Strength: powerful **joins**, well-understood theory.
* Weakness: impedance mismatch with nested/JSON-like app data.

**Document Model**

* Data stored as **self-contained documents** (JSON, BSON).
* Pros: schema flexibility, locality (all related fields in one place).
* Cons: poor for many-to-many relationships (joins weak).

**Graph Model**

* **Vertices + edges** (nodes + relationships).
* Useful when “anything may be related to everything” (social networks, knowledge graphs).
* Pros: natural for connected data.
* Cons: performance challenges on large traversals.

⚖️ Takeaway: **No single best model** — pick per workload.

**3. Query Languages**

* **Imperative vs Declarative**:
  + Imperative (CODASYL, MapReduce snippets): specify how to compute.
  + Declarative (SQL, Cypher, SPARQL): specify what you want, optimizer decides how.
* SQL: declarative, hides access paths, benefits from query optimizer.
* MongoDB aggregation pipeline: declarative but JSON-flavored.
* Graph queries: Cypher (property graphs), SPARQL (triple stores), Datalog (logic-based).

**4. Schema Flexibility**

* Relational DBs: **explicit schema enforced on write**.
* NoSQL (doc/graph): **implicit schema validated on read**.
* Trade-off:
  + Schema-on-write = consistency + safety.
  + Schema-on-read = flexibility + evolvability.

**5. Mappings to GenAI Pyramid / LexiFlow**

| **DDIA Concept** | **Pyramid Layer** | **LexiFlow Application** |
| --- | --- | --- |
| Relational model | Infra (metadata DB) | DynamoDB for doc metadata (stable joins, consistency). |
| Document model | RAG (storage) | S3 JSON docs, retrieval chunks as documents. |
| Graph model | RAG + Orchestration | Knowledge graphs for citations, agent-to-agent links. |
| Declarative queries | RAG retrieval | FAISS/Weaviate ANN queries as declarative retrieval. |
| Schema-on-read | Eval + Guardrails | Adapt to new doc structures w/o breaking pipeline. |

**6. LexiFlow Example**

* Document model → PDFs stored as JSON chunks in S3.
* Relational model → DynamoDB for tracking doc ingestion + metadata (doc\_id, tenant, version).
* Graph model → extend LexiFlow to build **citation graph** (doc → section → references).
* Query language design: RAG queries = declarative (retriever decides ANN path), not imperative.

**7. Interview-Style Q&A**

* **Q:** Why can’t one model serve all needs?  
  **A:** Trade-offs: relational excels at joins, document at locality, graph at relationships. Each adds overhead if misused.
* **Q:** Which model fits RAG pipelines best?  
  **A:** Hybrid: documents for chunks, relational for metadata, optional graph for knowledge grounding.
* **Q:** How do declarative queries matter in GenAI?  
  **A:** Let retrievers optimize ANN access paths (HNSW, IVF-PQ) instead of hardcoding search steps.

**8. Key Takeaways**

* **Data model choice = system design choice** (affects performance & maintainability).
* For GenAI/RAG: hybrid storage is the norm.
* Schema flexibility matters for evolving doc ingestion pipelines.
* Declarative queries hide infra complexity → essential for scalable retrieval.