MongoDB Associate Data Modeler

OEM: MongoDB • Duration: 3 Days (24 hrs) • Code: Associate-Data-Modeler

COURSE MODULES & TOPICS

Data Modeling Foundations

- What is data modeling and why it matters in MongoDB
- The document model vs. the relational model
- Flexible schema design and BSON document storage
- Constraints in data modeling: network, disk, RAM, security
- The data modeling methodology — 3 phases: workload, relationships, patterns
- Simplicity vs. performance trade-offs in schema design

Requirements Gathering (10%)

- Identifying write speed and scalability requirements
- Analytical computation: on-the-fly vs. pre-computed storage
- Translating business and technical requirements into schema decisions
- Documenting quantified workload for read and write operations

Entities (13%)

- Identifying and defining core entities in a data model
- Strong vs. weak entities and their attributes
- Cardinality and entity relationships (UML aggregation vs. composition)
- Has-a, Part-of, and Is-a relationships in document modeling
- Mapping entity attributes to MongoDB document fields

Relationships (8.5%)

- One-to-one relationships — embedding vs. referencing
- One-to-many and one-to-few relationships
- Many-to-many relationships with linking documents
- When to embed: data locality and read performance
- When to reference: large sub-documents, frequently updated data
- Data duplication trade-offs in denormalized models

Workload / Usage Analysis (10%)

- Identifying read and write operation patterns
- Using explain() and executionStats for query analysis
- Historical data management with TTL indexes
- Designing for scalability and high-throughput workloads
- Balancing read vs. write performance in schema design

Data Model Design — Schema Patterns (28%)

- Approximation Pattern — reduce write frequency for near-exact counts
- Attribute Pattern — handle fields with similar value types as key-value pairs
- Bucket Pattern — group time-series or IoT data into fixed-size buckets
- Computed Pattern — pre-compute and store aggregated values
- Document Versioning Pattern — keep multiple versions of a document
- Extended Reference Pattern — embed a subset of referenced document fields
- Inheritance Pattern — model shared and specialized fields across types
- Outlier Pattern — handle rare large documents without penalizing the common case
- Polymorphic Pattern — store different but related document types in one collection
- Schema Versioning Pattern — support multiple schema versions during migrations
- Subset Pattern — split large documents into frequently and infrequently accessed fields
- Tree Pattern — model hierarchical data (parent reference, child reference, materialized paths)

Schema Design Anti-patterns

- Massive arrays (unbounded array growth)
- Bloated documents exceeding 16 MB
- Unnecessary indexes increasing write overhead
- Data normalization causing excessive lookups ($lookup overhead)
- Case-insensitive queries without proper collation or indexes
- Separating data that is always accessed together

Modeling for Technical Requirements (10%)

- JSON Schema validation — defining schema rules with $jsonSchema
- validationLevel (strict vs. moderate) and validationAction (error vs. warn)
- Pattern limitations and when NOT to apply a design pattern
- Performance vs. storage efficiency trade-offs
- Application logic constraints influencing schema choices
- Handling time-series data and event-driven architectures

Indexing Strategies (13%)

- Compound indexes — field order and query selectivity
- Multi-key indexes — indexing array fields
- Partial indexes — indexing a subset of documents with a filter
- Sparse indexes — excluding documents missing the indexed field

- Clustered indexes — organizing collection storage by index key
- TTL indexes — automatic document expiration
- Hidden indexes — testing index removal without dropping
- Index intersection and covered queries
- Index optimization: ESR rule (Equality, Sort, Range)

Monitoring and Evolving Data Models (7.5%)

- Schema lifecycle management — planning for schema changes
- Using db.collection.stats() for storage and index analysis
- Using db.serverStatus() for operational health metrics
- Aggregation pipeline with $indexStats for index usage monitoring
- Schema evolution strategies — backward and forward compatibility
- Data governance — enforcing quality, consistency, and compliance
- Zero-downtime schema migrations with the Schema Versioning Pattern