CSCI 485 – MongoDB

1) Project Overview and Learning Goals

- **Domain**: Event Discovery and Check-In System with Geospatial Analytics
- **Primary DB**: MongoDB (document model, GeoJSON, aggregations, change streams?)
- **Course Learning Alignment**:
  - Modeling non-relational data for real applications (events, venues, check-ins)
  - Index design and query planning for performance at scale
  - Advanced MongoDB features: text search, geospatial, aggregations, transactions
  - Real-time data processing using change streams and WebSockets ?
  - Benchmarking, explain plans, and optimization

2) Data Model and Justification

- **Document Model Rationale**: Events have heterogeneous attributes, embedding provides efficient reads, references used for shared entities (venues, users). Demonstrates schema flexibility vs. rigid relational schemas.
- **Collections**: `events`, `venues`, `users`, `checkins`
- **Embedding vs Referencing**:
  - Embed: `tickets[]`, `attendees[]`
  - Reference: `venue_id`, `user_id`, `event_id`
- **Validation**: JSON Schema rules (required fields, GeoJSON structure, numeric bounds) to show controlled flexibility.
- **Identifiers**: `ObjectId` for distributed, timestamped IDs, enables cursor pagination and time-aware operations.

3) Query Workloads and Index Strategy

- **Discovery**:
  - Text search across `title`, `description`, `tags` using MongoDB text index
  - Category filtering + chronological sorting
  - Geospatial discovery via `$geoNear` on `location` (2dsphere)
- **Temporal**:
  - Upcoming events, weekend windows, date ranges
- **Analytics**:
  - Peak hours/days, category popularity, monthly trends (aggregation)
- **Pagination**:
  - Cursor-based pagination using `_id` to avoid `skip` penalties
- **Indexes**:

    - `location: "2dsphere"`
    - Text index on `title, description, category, tags`
    - `start_date`, `created_at`
    - Compound: `(category, start_date)`, `(location:2dsphere, start_date)`, `(organizer, start_date)`

4) Advanced MongoDB Features

- **Geospatial**: `$geoNear`, GeoJSON, 2dsphere index – supports "events near me this weekend" use case.
- **Text Search**: Multi-field index with relevance, explore `$meta: "textScore"` ordering.
- **Aggregations**: `$group`, `$sort`, time extraction operators.
- **Change Streams**: Real-time updates on `events` collection.
- **Transactions**: Multi-document workflow for ticket booking (seat decrement + check-in creation).

5) CAP & Consistency Trade-offs

- **Priority**: Availability, strong consistency for ticket booking.
- **Eventual Consistency** acceptable for: attendee counts, analytics, recommendations.
- **Strong Consistency** required for: seat inventory, payments (transactional section).

6) Real-Time Component ?

- **Design**: MongoDB Change Streams → Flask-SocketIO → browser clients
- **Use cases**: New events, updates, deletions, optional room subscriptions (by location/category)

7) Semester Plan & Milestones

- Week 1–2: Finalize schema/indexes, seed 10k dataset, baseline benchmarks
- Week 3–4: Implement analytics dashboards, capture explain plans, tune indexes
- Week 5–6: Real-time pipeline, measure update latency, optional room filters
- Week 7–8: Transactions for booking, contention tests on seat inventory
- Week 9–10: Scaling experiments (index size, memory, cache efficacy)
- Week 11–12: Write-up: performance results, lessons learned, trade-offs
- Week 13–14: Polish and present

8) To Think On Throughout The Semester

- What index changes had the biggest effect and why?
- Where did index intersection or wrong index choice hurt performance?
- How did cursor pagination affect user-perceived latency vs. offset?
- What trade-offs did you make between embedding and referencing?
- Where is strong consistency required, and how was it achieved?
- How would you shard this in production and what are risks?