Phase 1

Advanced Database Systems

Club Booking System

Name	ID
Hesham Mohamed Kamel El Afandi	21P0054
Mohamed Walid Helmy	21P0266
Youssef Habil Abou Elkheir	21P0187
Mazen Saeed Mohamed	21p0125
Youssef Ahmed Saad	21P0045

Table of Contents

1. Identify Related Data and Workload

Identify Entities and Relationships

#	Entity Name	Description (What data does it hold?)
1	Member	Stores member information such as name, contact details, membership level, and account status.
2	Membership Level	Defines privileges like maximum bookings per day, advance booking window, and accessible facilities.
3	Facility	Represents each club facility (court, room, pool, etc.) with attributes such as name, type, location, availability status, and maintenance notes.
4	Booking	Stores reservation details, including member, facility, date/time, duration, and booking status (confirmed/cancelled).
5	UsageLog	Records real-time facility usage — check-in/check-out times, duration, and member involved.
6	Payment	Logs financial transactions for bookings — amount, payment method, date, and payment status.
7	Staff	Contains staff data such as name, role, and assigned facilities for managing bookings or maintenance.
8	Notification	Holds messages or alerts sent to members (e.g., booking confirmation, cancellation, or maintenance updates).

Generate Workload Table

#	Action	Query Type (Read/ Write)	Entities Involved (Read/Returned)	Frequency (per day/hour)	Priority
1	Member Login & Profile View	Read	Member, MembershipLevel	~200/day	**High**
2	View Available Facilities (Real-time)	Read	Facility, Booking	~150/day	**High**
3	Create Booking Request	Write	Booking, Member, Facility	~80/day	**High**
4	Cancel Booking	Write	Booking, Facility	~20/day	**Medium**
5	List All Bookings for a Member	Read	Booking, Member, Facility	~100/day	**High**
6	Check-In to Facility (Usage Start)	Write	UsageLog, Member, Facility	~70/day	**High**
7	Check-Out of Facility (Usage End)	Write	UsageLog, Facility	~70/day	**High**
8	Send Notification to Member	Write	Notification, Member	~50/day	**Medium**
9	Process Payment for Booking	Write	Payment, Booking, Member	~40/day	**Medium**
10	Generate Usage Statistics for Admin	Read	UsageLog, Facility, Booking	~5/hour	**Medium**

11	Manage Membership Levels	Write	MembershipLevel	~5/day	**Low**
12	Update Facility Availability (Maintenance Only)	Write	Facility, Staff	~10/day	**Medium**
13	Add / Edit Facility	Write	Facility, Staff	~5/day	**Medium**
14	Register New Member	Write	Member, MembershipLevel	~10/day	**Medium**
15	Generate Monthly Usage & Revenue Report	Read	UsageLog, Payment, Facility	~1/day	**Medium**

2. Create a Schema Map (Embed vs. Reference)

Determine the Collections

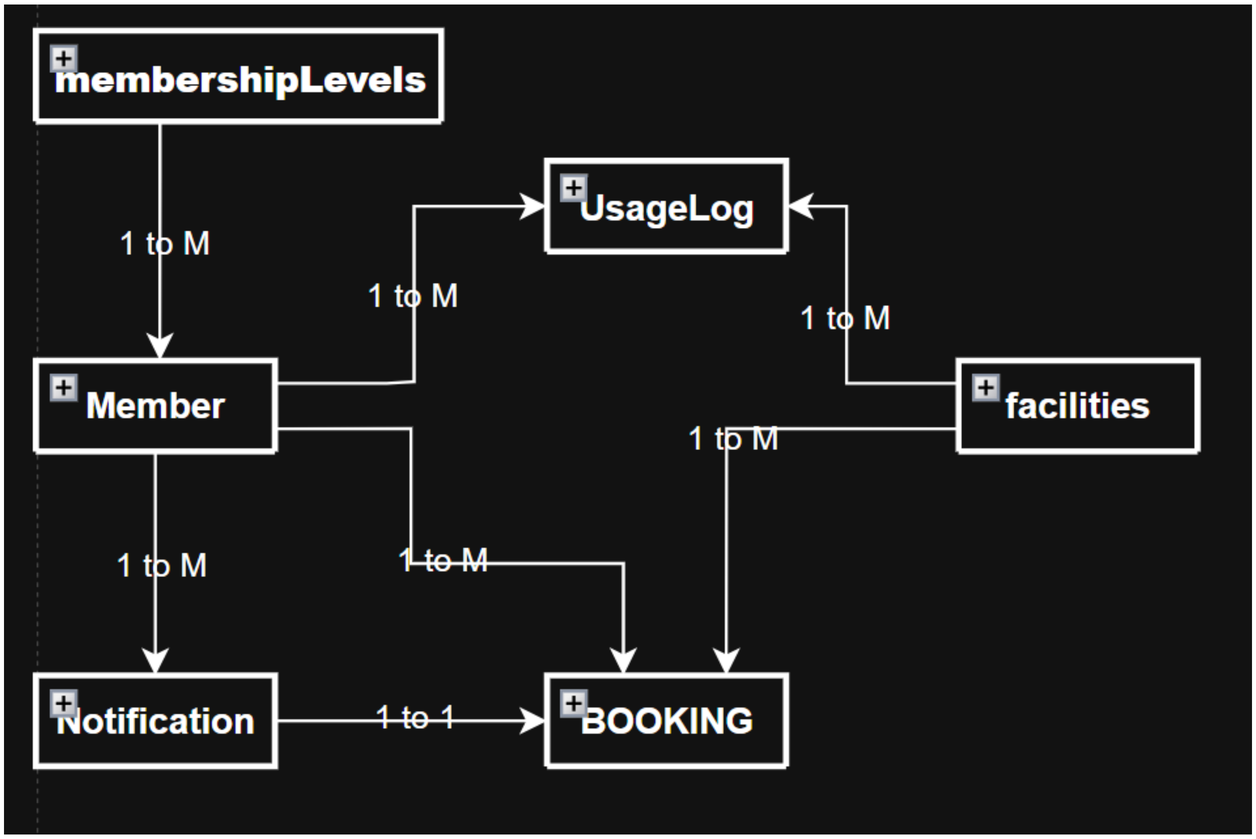
The set of collections below represents the final logical structure of the database, created based on identified entities and workload characteristics.

#	Collection Name	Purpose
1	members	Stores personal and contact information for all club members, along with their assigned membership level and current account status.
2	membershipLevels	Defines all membership tiers and their privileges, including booking limits, advance booking window, and accessible facility types.
3	facilities	Represents each facility (court, room, pool, etc.) with details like type, location, availability, and embedded staff responsible for it.
4	bookings	Stores all reservation details made by members, along with embedded payment information. Tracks facility, timing, and booking status.
5	usageLogs	Records real-time check-in and check-out activity for each facility usage to analyze utilization and generate reports.
6	notifications	Stores system-generated messages sent to members (e.g., booking confirmations, cancellations, or maintenance alerts).

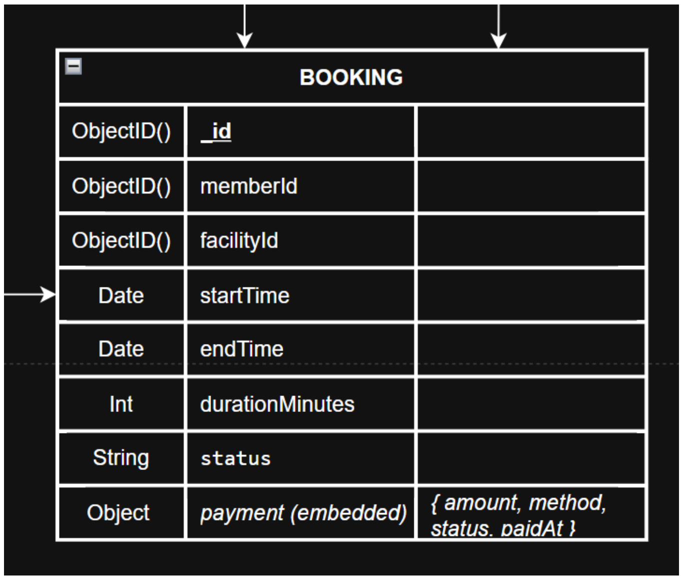
Map the Relations and Justify

Relationship (A to B)	Modeling Choice (Embed/Reference)	Justification (Why?)
Member to MembershipLevel	**Reference** (One-to-Many)	**Reason:** Since membership levels are shared across many members and don't change often, referencing makes more sense than embedding
Booking to Member	**Reference** (Many-to-One)	**Reason:** Members can have multiple bookings, and member information might change independently,
Booking to Facility	**Reference** (Many-to-One)	**Reason:** Each facility can have many bookings pointing to it. Since facilities get updated for maintenance or availability changes
Booking to Payment	**Embed** (One-to-One)	**Reason:** Every booking has exactly one payment associated with it, and payment details are always retrieved together with the booking
UsageLog to Member	**Reference** (Many-to-One)	**Reason:** Usage logs are created constantly throughout the day (~140 check-ins/outs), and members have many usage records over time.
UsageLog to Facility	**Reference** (Many-to-One)	**Reason:** Similar to the member relationship, each facility accumulates many usage logs. Referencing prevents data bloat
Facility to Staff	**Embed** (One-to-Few)	**Reason:** Each facility is managed by a small number of staff members (usually just a few people), and this information is typically viewed together with facility details.
Notification to Member	**Reference** (Many-to-One)	**Reason:** Notifications are sent regularly (~50/day) and pile up over time, while each member receives many notifications.

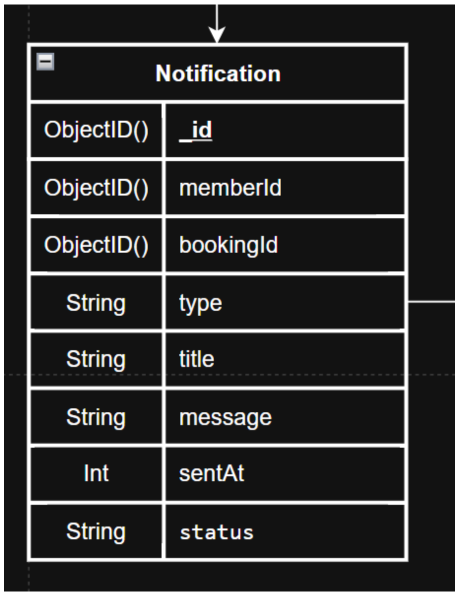
Schema Modeling Diagram



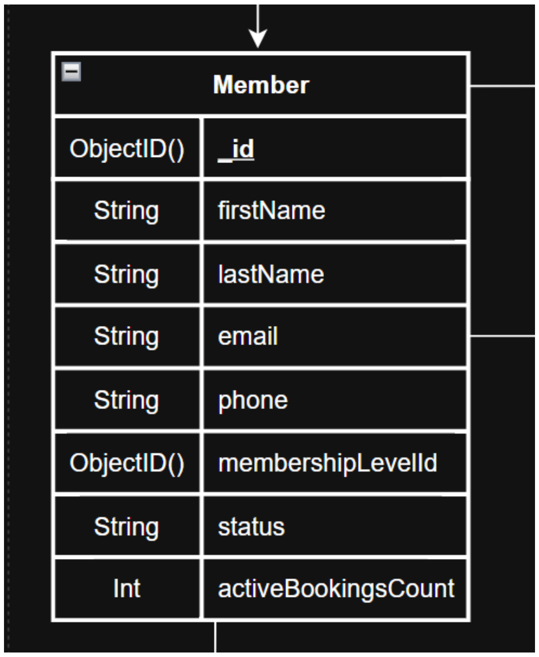
Tables



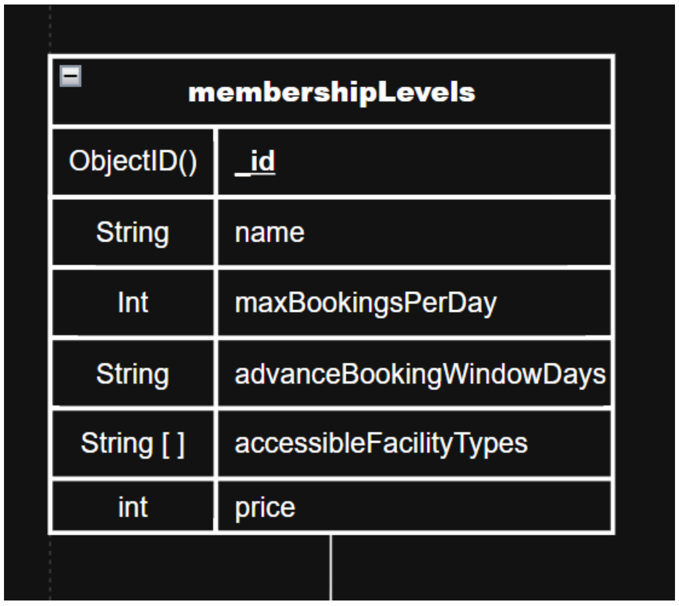
BOOKING		
ObjectID()	_id	
ObjectID()	memberId	
ObjectID()	facilityId	
Date	startTime	
Date	endTime	
Int	durationMinutes	
String	status	
Object	*payment (embedded)*	*{ amount, method, status, paidAt }*



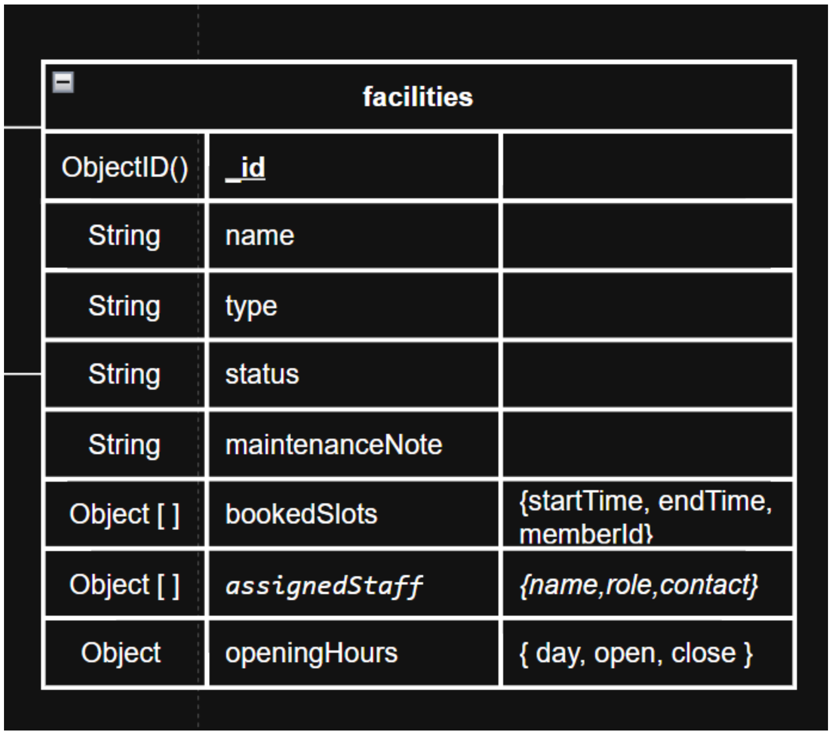
Notification	
ObjectID()	_id
ObjectID()	memberId
ObjectID()	bookingId
String	type
String	title
String	message
Int	sentAt
String	status



Member	
ObjectID()	_id
String	firstName
String	lastName
String	email
String	phone
ObjectID()	membershipLevelId
String	status
Int	activeBookingsCount



membershipLevels	
ObjectID()	_id
String	name
Int	maxBookingsPerDay
String	advanceBookingWindowDays
String [ ]	accessibleFacilityTypes
int	price



facilities		
ObjectID()	_id	
String	name	
String	type	
String	status	
String	maintenanceNote	
Object [ ]	bookedSlots	{startTime, endTime, memberId}
Object [ ]	*assignedStaff*	*{name,role,contact}*
Object	openingHours	{ day, open, close }

3. Conclusion

The final MongoDB schema provides a clean and efficient structure for the Clubs-Sports Booking System. It uses embedding where it improves performance and references where data grows or is shared. The design supports real-time availability, booking rules, and usage tracking, while keeping the system scalable and easy to maintain.