BookMyShow High-Level Design - Summary & DB Schema

What’s Already Covered:

- ✔ Core Flow: City → Movie → Theater → ShowTime → Seats → Payment
- ✔ Traffic Estimations and Peak Load considered
- ✔ Booking, Payment, and Search services separated
- ✔ Transaction isolation (Serializable) for booking integrity
- ✔ Basic APIs and Entities like User, City, Movie, Show, Booking defined

Suggestions for Enhancements:

- 1. Add CDN (CloudFront) for static content
- 2. Use API Gateway for routing and rate limiting
- 3. Use Load Balancer and Redis for caching
- 4. Use SQS/Kafka for async operations
- 5. Object Storage (S3) for media
- 6. Add Notification, Auth, User Preferences, Analytics Services
- 7. Add APIs: showtimes, seats, payment status, search

Database Design Summary:

- SQL Tables: User, City, Cinema, Theater, Movie, Show, Booking, Seat, Show_Seat, Booking_Seat, Paym
- NoSQL (Redis): Seat locking (TTL), caching showtimes and movie lists
- Partitioning Strategy: BookingId (SQL), Sorted Sets for Redis Feed Cache

Recommended Database Schema

```
-- User
CREATE TABLE User (
id UUID PRIMARY KEY,
name VARCHAR(100),
email VARCHAR(100) UNIQUE,
phone VARCHAR(15),
password_hash TEXT
);

-- City
CREATE TABLE City (
id UUID PRIMARY KEY,
name VARCHAR(100),
state VARCHAR(100)
);

-- Cinema
CREATE TABLE Cinema (
id UUID PRIMARY KEY,
name VARCHAR(100),
location TEXT,
city_id UUID REFERENCES City(id)
);

-- Theater (Screen)
CREATE TABLE Theater (
id UUID PRIMARY KEY,
name VARCHAR(100),
seats INT,
cinema_id UUID REFERENCES Cinema(id)
);

-- Movie
CREATE TABLE Movie (
id UUID PRIMARY KEY,
name VARCHAR(100),
actors TEXT,
trailer_url TEXT
);

-- Show
CREATE TABLE Show (
id UUID PRIMARY KEY,
movie_id UUID REFERENCES Movie(id),
theater_id UUID REFERENCES Theater(id),
show_time TIMESTAMP,
price DECIMAL(10,2)
);

-- Seat
CREATE TABLE Seat (
id UUID PRIMARY KEY,
theater_id UUID REFERENCES Theater(id),
seat_number VARCHAR(10)
);
```