Distributed Property Rental Platform

Database Architecture for Concurrent, Scalable Accommodation Systems

Team: Santiago Sanchez Moya, Juan Daniel Rodriguez Hurtado, Faider Camilo Trujillo Olaya

Business Context

A distributed property rental platform connects hosts offering short-term accommodation with guests seeking stays worldwide. The system must handle high-concurrency booking operations, real-time search across thousands of properties, and analytical workloads for business intelligence.

Core challenge: balance transactional integrity (preventing double-bookings) with analytical performance (host dashboards, search ranking) whilst supporting global scale and fault tolerance.

Critical Requirements

Performance Target

Support 100 concurrent users generating 300 read operations per minute during peak periods

Data Integrity

Prevent double-booking scenarios under concurrent write operations with ACID guarantees

Scalability Goal

Manage 1,000 property listings with 50 new additions monthly, processing 500-1,000 bookings per month

Four-Layer Architecture Strategy

Diagram placeholder: Complete architecture diagram showing four distinct layers with data flows between components

01

Presentation Layer

CDN for static assets and load balancers distribute traffic globally, reducing latency and ensuring fault tolerance

02

Operational OLTP Layer

PostgreSQL handles transactional operations with ACID compliance; object storage manages binary media separately

03

Analytical OLAP Layer

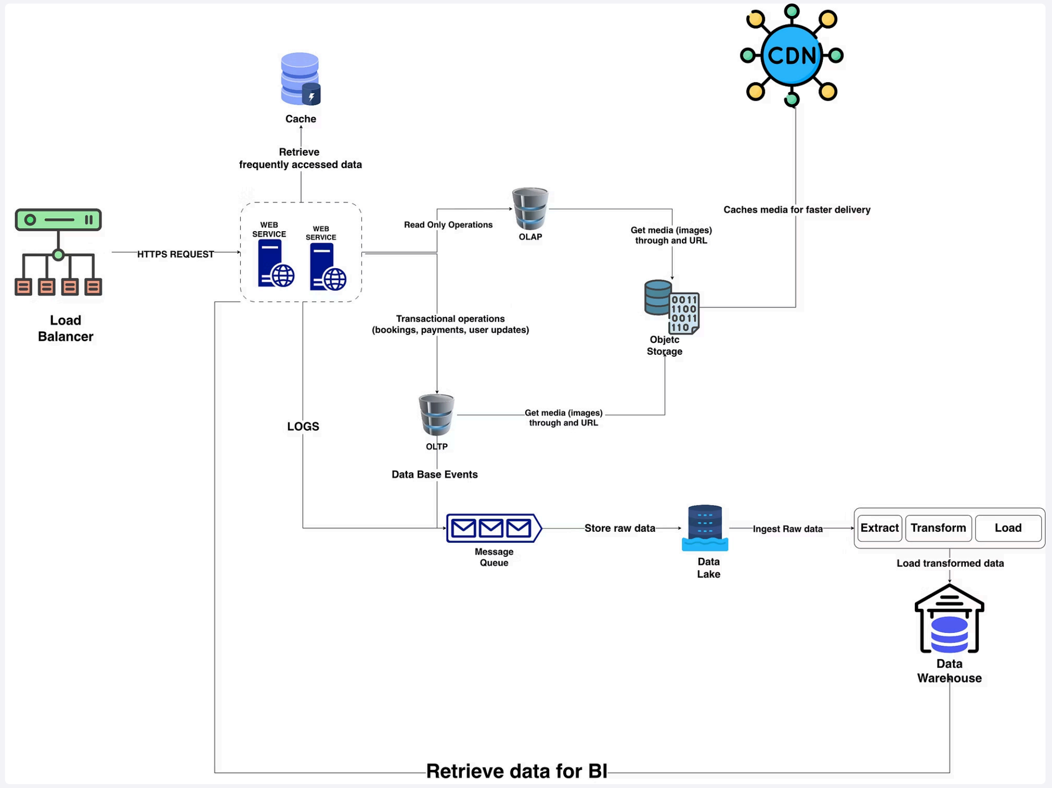
Read replicas, data warehouse, and ETL pipelines support business intelligence without impacting operational performance

04

Support Layer

Redis cache and message queues optimize performance and enable asynchronous background processing

Hight Architecture Diagram



Data Layer Decisions

PostgreSQL (Relational)

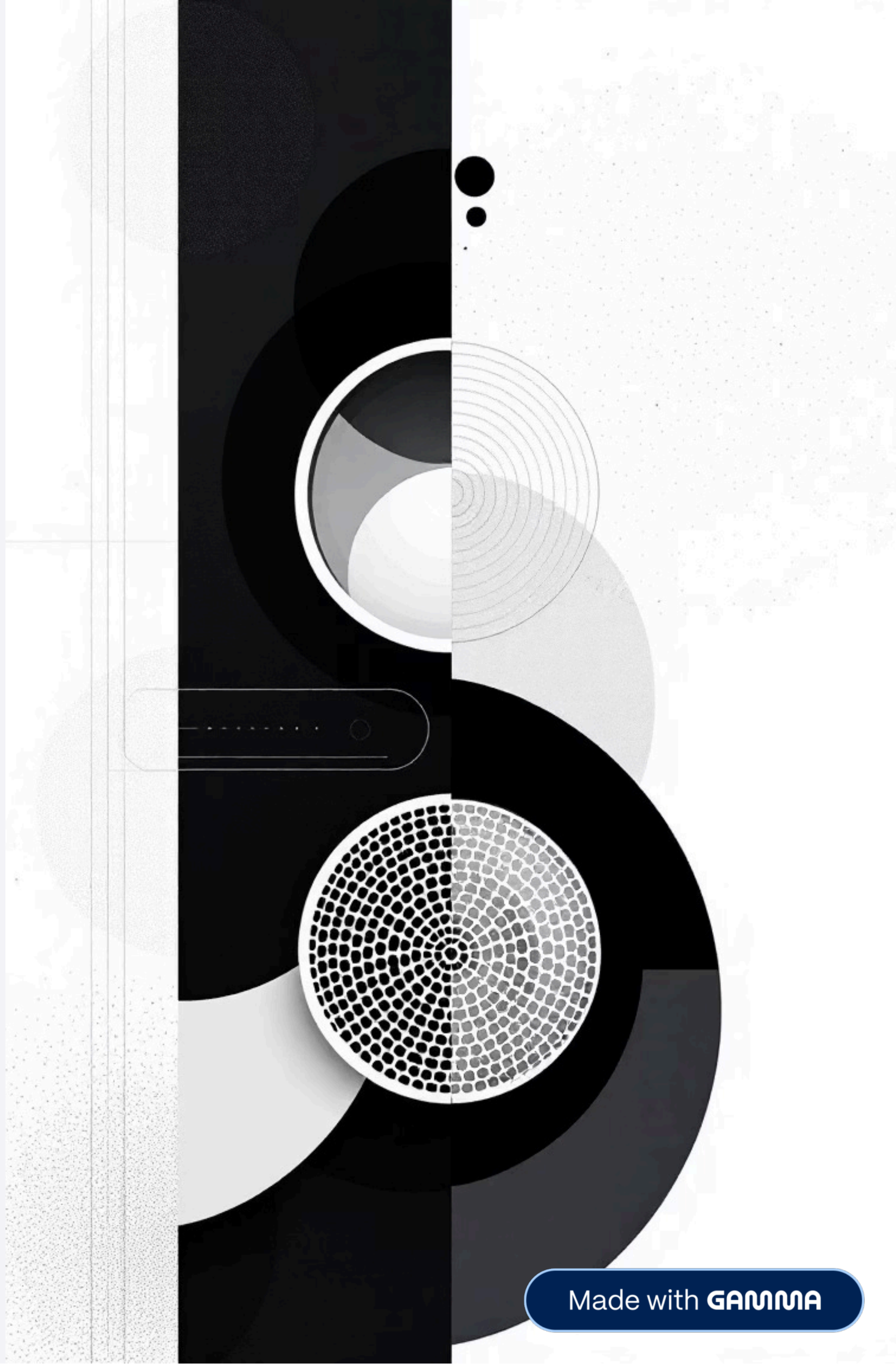
- User accounts and authentication
- Property listings and metadata
- Booking transactions and status
- Payment records and audit trails
- Reviews linked to bookings

Rationale: Complex relationships require referential integrity, foreign key constraints, and transactional consistency

Redis (NoSQL Cache)

- Session management and tokens
- Pre-computed search results
- Real-time availability indices
- Property ranking scores
- Notification queues

Rationale: Sub-millisecond access for high-frequency reads without database load



Concurrency Control Strategy

1

High Isolation Transactions

SERIALIZABLE level for booking workflows prevents phantom reads during availability checks

2

Atomic Conditional Updates

SQL conditions ensure updates only succeed when resource state is valid, avoiding lost updates

3

Advisory & Distributed Locks

PostgreSQL advisory locks and Redis Redlock pattern provide mutual exclusion for critical operations

Double-booking prevention: Atomic UPDATE statements with WHERE clauses validate availability state before modification, ensuring only one transaction succeeds when multiple users attempt simultaneous bookings

Distributed Database Design

Diagram placeholder: Distributed architecture showing regional partitions, replication strategy, and data flow between nodes

1

Horizontal Partitioning

Composite keys combine regional identifiers with property IDs, enabling natural data segmentation by geography

2

Regional Fault Containment

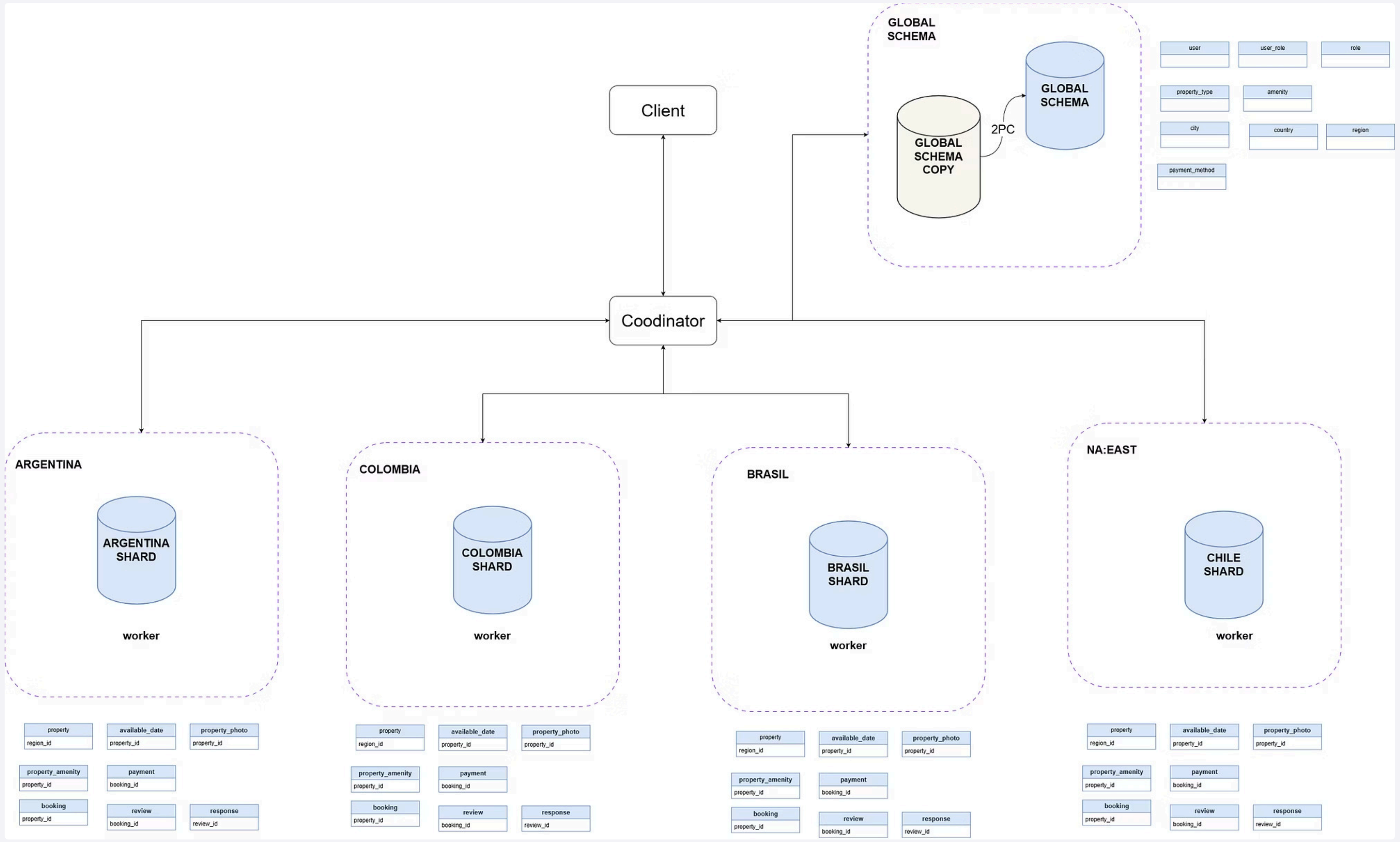
Synchronous replication within partitions ensures disruptions in one zone don't compromise global availability

3

Workload Separation

Specialized clusters for transactional and analytical operations prevent reporting queries from degrading booking performance

Distributed DB Design



Performance Optimisation Strategies

Data Partitioning

Bookings and payments tables partitioned by month and region reduce index size and enable parallel scans

Distributed Caching

Redis cluster handles popular property queries with sub-millisecond latency, reducing database load by 70-80%

Event-Driven Architecture

Message queues process emails, notifications, and ETL triggers asynchronously without blocking user operations

Read Replica Strategy

Dedicated replicas for search and analytics isolate read-heavy workloads from write-critical transaction processing

Expected Performance Outcomes

Query Latency

Cache-hit property searches: **<10ms**

Complex filtered queries: **<500ms**

Booking transactions: **<1 second**

Throughput Capacity

Concurrent users: **100+**

Queries per minute: **300+**

Monthly bookings: **1,000+**

Data Freshness

Replication lag: **<1ms**

Cache TTL: **60-300s**

Analytics refresh: **Real-time**

Performance basis: Redis sorted sets enable O(log N) range queries for price-based search. PostgreSQL B-tree indices support efficient filtered searches. Partitioning maintains query performance as data volume grows linearly.

Architectural Outcomes & Integration Path

The proposed architecture delivers a production-ready blueprint for concurrent, distributed property rental systems. Key achievements include proven concurrency control patterns preventing data anomalies, clear separation between OLTP and OLAP workloads, and hybrid SQL/NoSQL strategy balancing consistency with performance.

Business Integration

System supports immediate operational needs whilst establishing foundations for global scaling as market demands evolve

Next Phase

Prototype implementation with load testing, multi-region deployment validation, and observability infrastructure for production monitoring