UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Universidad Distrital Francisco Jose de Caldas
Computer engineering program

Final Delivery

Juan Daniel Rodríguez Hurtado
Santiago Sanchez Moya
Faider Camilo Trujillo Olaya

Supervisor: Eng. Carlos Andrés Sierra, M.Sc.

December 11, 2025

Abstract

This report presents the design and architecture of Havenly, a scalable and data-driven property rental platform. Havenly manages high volumes of bookings, users, and properties while ensuring data integrity and fast response times. The platform separates transactional and analytical workloads, employing caching and distributed database techniques to optimize performance. A data warehouse is proposed to support reporting and decision-making without affecting operational tasks. This study emphasizes database design, system architecture, and information management as foundations for building reliable, high-performance digital marketplaces.

Keywords: Property Rental Platform, Database Architecture, Scalability, Data Warehouse, OLTP, OLAP, Caching, Distributed Systems.

Contents

List of Figures

List of Tables

List of Abbreviations

SMPCS	School of Mathematical, Physical and Computational Sciences

Chapter 1

Introduction

The rise of digital marketplaces for short-term accommodations has transformed the way people find, book, and manage properties. Platforms like Airbnb and Vrbo demonstrate the importance of data-driven systems capable of handling high concurrency, transactional integrity, and analytical workloads simultaneously.

Havenly is proposed as a scalable, reliable, and secure property rental platform designed to meet these demands at a regional level. The platform supports property listings, bookings, payments, and user interactions while ensuring strong data consistency and system performance.

Key challenges addressed by Havenly include:

- Concurrency Management: Ensuring that simultaneous booking requests do not result in double bookings or inconsistent states.
- Scalability: Supporting growing numbers of users, properties, and transactions without degrading performance.
- Data-Driven Insights: Maintaining a separation between transactional workloads (OLTP) and analytical workloads (OLAP) to allow reporting, dashboards, and decision support.
- Performance Optimization: Leveraging caching and distributed data techniques to minimize latency and maximize throughput.

This report focuses on the database-centric design of Havenly, emphasizing how careful architecture and modeling decisions enable a robust, maintainable, and high-performing platform. By detailing the system objectives, architecture, and information requirements, the study provides a foundation for implementation and future expansion.

1.1 Background

Havenly is conceived as a database-driven information system, where the core complexity lies not in the user interface but in the management of data, transactions, and information flows. The platform must handle highly structured data—such as users, properties, bookings, and payments—while also supporting analytical and administrative data used for reporting and decision-making.

From a database perspective, property rental platforms exhibit characteristics typical of high-concurrency transactional systems. Multiple users may attempt to reserve the same property within overlapping time windows, making data consistency and isolation critical to

system correctness. At the same time, the system must support read-intensive workloads generated by property search, filtering, and exploration.

As usage grows, centralized database architectures become insufficient to meet performance and availability requirements. Industry practices indicate the need for replication, data distribution, and caching to improve scalability and fault tolerance. Additionally, the increasing demand for business analytics requires separating operational data processing from analytical workloads to avoid performance degradation.

Within this context, Havenly serves as a case study for applying advanced database concepts—including transactional integrity, distributed data management, and OLTP/OLAP separation—within a realistic application domain. The background presented here justifies the architectural decisions explored in subsequent sections of this report.

Chapter 2

Literature Review

The design of modern property rental platforms has been widely studied in both academic literature and industry case analyses, particularly in the context of large-scale digital marketplaces. These systems are commonly characterized as two-sided platforms, where value is co-created through interactions between hosts and guests, mediated by a technological infrastructure that must ensure trust, availability, and performance.

2.1 Two-Sided Marketplaces and Data Management

Research on digital marketplaces emphasizes the importance of structured data models to represent users, listings, transactions, and reviews in a consistent manner. In accommodation platforms, inaccurate or inconsistent data—especially related to availability and pricing—directly affects user trust and platform credibility. As a result, relational databases remain a core component due to their support for strong consistency, integrity constraints, and transactional guarantees.

2.2 OLTP and OLAP Separation

Several studies highlight the necessity of separating Online Transaction Processing (OLTP) from Online Analytical Processing (OLAP) workloads. Transactional operations such as booking confirmations and payments require low latency and strict consistency, while analytical queries for reporting, trends, and performance evaluation often involve large data scans and complex aggregations. Mixing these workloads in a single database instance can significantly degrade system performance. Consequently, modern architectures propose dedicated analytical stores or data warehouses fed by transactional systems.

2.3 Concurrency and Consistency Challenges

Concurrency control is identified as a critical challenge in reservation-based systems. Literature on database systems stresses the need for atomic transactions and isolation mechanisms to prevent anomalies such as double booking, lost updates, or inconsistent reads. Techniques such as row-level locking, transaction isolation levels, and pessimistic or optimistic concurrency control are commonly applied to ensure correctness under high contention.

2.4 Scalability and Distributed Databases

Industry reports from large-scale platforms demonstrate a growing reliance on distributed database technologies to scale beyond the limits of single-node systems. Horizontal partitioning (sharding) and distributed query execution allow platforms to manage increasing volumes of data and concurrent users. Extensions to relational databases, rather than full migration to NoSQL solutions, are often preferred when strong consistency and relational modeling remain priorities.

2.5 Caching and Performance Optimization

Caching layers, particularly in-memory data stores, are frequently cited as effective mechanisms to reduce database load and response times for read-heavy workloads. Frequently accessed data such as property listings and search results can be served from cache, minimizing repeated queries to the primary database and improving overall system responsiveness.

2.6 Implications for Havenly

The concepts discussed in existing literature directly inform the design of Havenly. The choice of a relational database core, the separation of transactional and analytical workloads, the use of distributed tables, and the integration of caching mechanisms align with best practices identified in both academic studies and real-world systems. This literature provides the theoretical foundation for the architectural decisions presented in the following sections.

Chapter 3

Objectives

3.1 General Objective

To analyze, design, and document a database-oriented architecture for a property rental platform that ensures transactional integrity, scalability, and support for analytical processing.

3.2 Specific Objectives

- Identify and formalize functional and non-functional requirements relevant to a high-concurrency reservation system.
- Design a normalized relational data model that accurately represents business entities and relationships while ensuring referential integrity.
- Propose an architecture that integrates replication, distribution, and caching to enhance performance and availability.
- Analyze concurrency scenarios and database mechanisms used to prevent anomalies such as double booking.

Chapter 4

Scope

This report focuses on the analysis and design of the data architecture for the Havenly property rental platform. The scope includes:

- Definition of business and information requirements relevant to a rental marketplace.
- Design of a logical and conceptual relational database model, including key entities and relationships.
- Proposal of a layered database architecture addressing transactional (OLTP) and analytical (OLAP) workloads.
- Analysis of concurrency, scalability, availability, and performance from a database systems perspective.
- Identification of operational and analytical information outputs to support system functionality and decision-making.

The following aspects are explicitly out of scope:

- Detailed user interface or user experience (UI/UX) design.
- Full implementation of APIs, services, or frontend components.
- Deployment, containerization, and production monitoring strategies.
- Load, stress, or fault-injection testing using real production data.

4.1 Assumptions

The design and analysis presented in this report are based on the following assumptions:

- The platform initially serves a regional market with controlled but growing user concurrency.
- A relational database management system is suitable for the core transactional workload due to consistency and integrity requirements.
- Read-heavy operations significantly outnumber write operations, particularly in search and property exploration.

- Payment processing and identity verification are delegated to external, third-party services.
- The system operates in a cloud-hosted environment, enabling replication, distribution, and elastic scaling.

4.2 Limitations

Despite aiming for a realistic and robust design, this study presents the following limitations:

- Workload estimates are theoretical and not derived from real-world usage metrics.
- Architectural decisions are evaluated conceptually, without empirical benchmarking or stress testing.
- Certain advanced features—such as real-time recommendations, fraud detection, or machine learning-based pricing—are considered only at a high level.
- Security mechanisms are discussed from a design perspective, without detailed cryptographic or compliance-level analysis.

Chapter 5

Methodology

The methodology adopted in this study follows a systematic, incremental, and database-centered approach. It integrates principles of requirements engineering, conceptual data modeling, and architectural design to ensure transactional integrity, scalability, and analytical capability. The focus of this methodology is not on software implementation details, but on database design decisions supported by conceptual and practical validation.

5.1 Business and Context Modeling

The initial phase of the methodology focused on understanding the business domain of a property rental platform. A Business Model Canvas was used to identify key stakeholders, including guests, hosts, and administrators, as well as core business processes such as property publication, reservation management, payment processing, and review handling.

This step defined the semantic boundaries of the system and established a shared understanding of the data domain, which served as the foundation for subsequent requirements elicitation and data modeling activities.

5.2 Requirements Engineering

Functional and non-functional requirements were elicited and refined to be explicit, measurable, and verifiable. Functional requirements describe core system capabilities such as user registration, property management, booking workflows, payment recording, and review submission.

Non-functional requirements focus on performance, availability, scalability, security, maintainability, and, in particular, concurrency control. Special attention was given to preventing anomalies such as double booking and inconsistent payment states.

User stories were prioritized using the MoSCoW method (Must, Should, Could, Won't) and estimated collaboratively through Planning Poker, ensuring alignment between business priorities and architectural constraints.

5.3 Data Modeling and Normalization

Based on the validated requirements, a conceptual Entity–Relationship (ER) model was developed to represent the main entities, attributes, and relationships of the system. The model clearly identifies primary keys, foreign keys, cardinalities, and participation constraints.

The conceptual model was then transformed into a logical relational schema normalized up to Third Normal Form (3NF) in order to reduce redundancy and avoid insertion, update, and deletion anomalies. Referential integrity constraints were defined to ensure consistency across all transactional operations.

The resulting schema was implemented using PostgreSQL, providing ACID-compliant transactional support for the operational workload.

5.4 Transaction and Concurrency Analysis

Critical transactional scenarios were analyzed to identify potential concurrency conflicts, particularly during the booking and payment processes. Scenarios such as simultaneous reservation attempts for the same property and overlapping date ranges were examined to define transaction boundaries and isolation requirements.

This analysis ensured that all booking operations are executed atomically and consistently, preventing anomalies such as dirty reads, lost updates, and double booking, while maintaining acceptable system performance.

5.5 Distributed Database Architecture

To address scalability and regional data locality, the PostgreSQL database was partitioned using the Citus extension. The distributed architecture consists of a coordinator node and multiple worker nodes, each representing a geographical region: Argentina, Brazil, Colombia, Chile, Peru, and Ecuador.

A regional identifier (`region_id`) was defined as the distribution key, ensuring that geographically related data is colocated on the same worker node. This strategy improves query parallelism, reduces latency, and supports horizontal scalability. The correctness of the distribution strategy was validated through query execution plans and data placement inspection.

5.6 Caching Strategy

An in-memory caching layer based on Redis was introduced to optimize read-heavy workloads. Frequently accessed data such as property listings, search results, and availability checks are stored in the cache, reducing the load on the primary database and improving response times.

The effectiveness of this strategy was validated by comparing query execution paths with and without cache hits, confirming reduced access to the transactional database for repeated read operations.

5.7 Data Warehouse and BI Design

To support analytical and decision-support workloads, a data warehouse was designed using a star schema. The central fact table, `fact_booking`, captures booking events and includes foreign keys referencing multiple dimension tables, including `dim_date`, `dim_property`, `dim_guest`, `dim_host`, and `region_id`, which also serves as the data distribution key.

The fact table stores additive metrics such as `nights_booked`, `total_revenue`, `host_commission`, and `platform_commission`, as well as non-additive attributes such as booking and payment status. Dimension tables provide descriptive attributes that enable multidimensional analysis across time, geography, user types, and property characteristics.

5.7.1 Materialized Views and Analytical Optimization

To improve analytical query performance, a materialized view named `vm_regional_kpis` was created as a pre-aggregated data structure. This view stores key performance indicators such as revenue per available room (RevPAR) and occupancy rate, aggregated by region and date.

The materialized view is refreshed periodically by a dedicated BI service, providing near-instantaneous response times for dashboards and reports. This approach eliminates the need for frequent full-table scans on the large `fact_booking` table, preserving system performance.

5.8 Validation and Evidence

Validation of the proposed methodology was achieved through practical implementation evidence. This includes the successful deployment of a PostgreSQL database with Citus-based distribution, the integration of Redis caching, and the execution of analytical queries against the star schema and materialized views.

Detailed schemas, SQL definitions, and example queries are provided in the appendices to support reproducibility and verify the consistency between requirements, architecture, and results.

Chapter 6

Results

This section presents the results obtained from the design and partial implementation of the Havenly database architecture. The results validate the architectural and modeling decisions proposed in this study by demonstrating their feasibility and alignment with database engineering best practices.

6.1 Functional and Non-Functional Requirements Validation

The defined functional requirements were mapped to transactional database operations, including user management, property publication, booking creation, payment registration, and review storage. Non-functional requirements related to consistency, availability, scalability, and performance were validated through architectural choices such as replication, distribution, and caching.

Table 6.1 summarizes representative requirements and their architectural support.

Table 6.1: Representative Requirements and Architectural Support

Requirement	**Supporting Mechanism**
Prevent double bookings	ACID transactions, isolation control
High read performance	Redis caching, read replicas
Scalability by region	Citus horizontal distribution
Analytical reporting	Star schema, materialized views

—

6.2 High-Level Database Architecture

A layered, database-centric architecture was implemented, separating concerns between transactional processing, analytical workloads, and support services.

- **Presentation Layer**: Web and mobile clients.
- **Operational Layer (OLTP)**: PostgreSQL with Citus distribution.
- **Support Layer**: Redis caching and asynchronous BI service.
- **Analytical Layer (OLAP)**: Star schema and materialized views.

6.3 Distributed PostgreSQL Deployment with Citus

The transactional database was implemented using PostgreSQL and horizontally partitioned using the Citus extension. The architecture consists of a coordinator node and multiple worker nodes, each corresponding to a geographical region: Argentina, Brazil, Colombia, Chile, Peru, and Ecuador.

The attribute `region_id` was defined as the distribution key, ensuring data locality and parallel query execution across workers.

Figure 6.2 shows the logical distribution of data across regions running on Docker

Figure 6.1: Citus-Based Distributed Architecture by Region

6.4 Caching Layer with Redis

Redis was integrated as an in-memory caching layer to reduce database load for read-heavy operations such as property search and availability checks. Cached entries include frequently accessed query results and precomputed responses.

The effectiveness of caching was validated by observing reduced query execution paths to the primary database in repeated requests.

6.5 Data Warehouse Star Schema Design

To support analytical workloads, a data warehouse was designed using a star schema. The central table, `fact_booking`, records booking events and references multiple dimension tables.

6.5.1 Fact Table: `fact_booking`

The fact table includes:

- **Foreign Keys (Dimensions):** `dim_date_id`, `dim_host_id`, `dim_guest_id`, `dim_property_id`, `region_id`
- **Additive Metrics:** `nights_booked`, `total_revenue`, `host_commission`, `platform_commission`
- **Non-Additive Attributes:** `booking_status`, `payment_status`

The `region_id` attribute also functions as the distribution key, maintaining alignment between the OLTP and OLAP architectures.

6.6 Dimension Tables

The following dimensions were implemented:

- **dim_date**: Supports temporal analysis by day, week, month, quarter, and year.
- **dim_property**: Enables analysis by property type, price, and capacity.
- **dim_guest and dim_host**: Allow segmentation of revenue and commissions by user role.
- **dim_currency** (optional): Supports multi-currency financial reporting.

—

6.7 Materialized View for BI Optimization

A materialized view named `vm_regional_kpis` was created as the final aggregation layer for BI consumption.

Purpose: To store pre-aggregated KPIs such as RevPAR and occupancy rate, grouped by `region_name` and `full_date`.

Advantages:

- Near-instant query response times.
- Reduced computational load on the fact table.
- Isolation of analytical workloads from transactional operations.

Listing 6.2 shows a simplified definition of the materialized view.

```
CREATE MATERIALIZED VIEW vm_regional_kpis AS
SELECT
    region_name,
    full_date,
    SUM(total_revenue) AS total_revenue,
    SUM(nights_booked) AS nights_booked
FROM fact_booking
JOIN dim_date ON fact_booking.dim_date_id = dim_date.id
GROUP BY region_name, full_date;
```

Listing 6.1: Materialized View Definition

—

6.8 Example of data in DW

```
docker exec -it $(docker ps --filter "name=coordinator" --format "{{.Names
    }}") psql -U heavenly -d heavenly -c "SELECT COUNT(*) FROM fact_booking
    ;"

 count

-------

     2
```

```
8
9  (1 row)
10
11
12
13 ssant@Santy MINGW64 ~/Documents/U/Bases2/Havenly/Final-Project/Heavenly (
       main)
14
15 $ docker exec -it $(docker ps --filter "name=coordinator" --format "{{.
       Names}}") psql -U heavenly -d heavenly -c "SELECT region_name,
       full_date, total_revenue, nights_booked, revpar, occupancy_rate FROM
       vm_regional_kpis WHERE total_revenue > 0 ORDER BY full_date DESC;"
16
17  region_name | full_date  | total_revenue | nights_booked |        revpar
              |    occupancy_rate
18
19 --
       -----------+------------+---------------+---------------+----------------------+-------
20
21  Argentina   | 2025-02-15 |        200.00 |             2 |
       200.0000000000000000 | 200.0000000000000000
22
23  Argentina   | 2025-01-10 |        300.00 |             3 |
       300.0000000000000000 | 300.0000000000000000
24
25 (2 rows)
```

Listing 6.2: STAR MODEL RESULTS

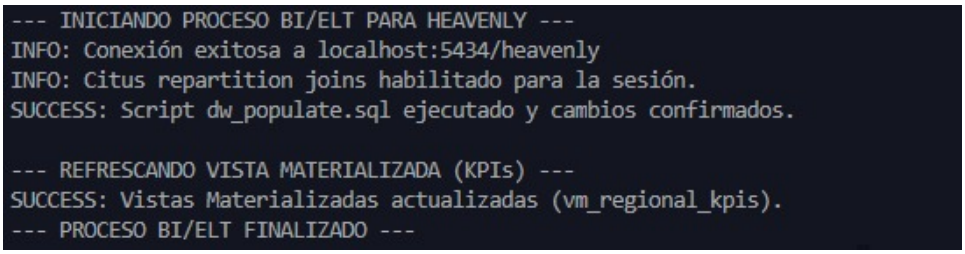

Figure 6.2: Python BI Service

Additional implementation details, SQL scripts, and execution examples are provided in the appendices to ensure traceability and reproducibility.

Chapter 7

Discussion

This chapter discusses the implications of the architectural and database design decisions presented in the Results chapter. The discussion focuses on scalability, consistency, performance, and analytical capabilities, evaluating how the proposed approach addresses common challenges in data-intensive marketplace systems.

7.1 Transactional Integrity and Concurrency Control

One of the most critical challenges in property rental platforms is the prevention of concurrency anomalies, particularly double booking. The use of PostgreSQL as the core transactional database ensures ACID compliance, providing atomicity and isolation guarantees for booking and payment transactions.

The analysis of transactional scenarios demonstrates that a relational database with well-defined transaction boundaries is well-suited for reservation-based systems. By relying on strict transaction handling and relational constraints, the system preserves data consistency even under concurrent access. Compared to eventual-consistency approaches, this design prioritizes correctness over relaxed consistency models, which is essential for financial and reservation data.

7.2 Scalability Through Distribution and Regional Partitioning

Horizontal scalability is achieved through the use of Citus, which enables distributed PostgreSQL deployments without sacrificing relational integrity. Partitioning data by geographical region using `region_id` as the distribution key improves data locality and allows queries to be executed in parallel across worker nodes.

This regional partitioning strategy aligns with real-world access patterns, where users typically interact with geographically nearby data. Compared to a single-node database or vertical scaling strategies, the distributed approach provides improved throughput and fault isolation. However, it also introduces operational complexity, such as distributed query planning and node coordination, which must be carefully managed.

7.3 Performance Optimization via Caching

The introduction of Redis as an in-memory caching layer significantly reduces load on the transactional database. Read-heavy operations, such as property searches and availability lookups, benefit from low-latency responses served directly from the cache.

This approach highlights the importance of separating read optimization from transactional correctness. While Redis improves performance, the database remains the single source of truth, ensuring consistency. The trade-off lies in cache invalidation and synchronization, which must be carefully designed to prevent stale data from being presented to users.

7.4 Analytical Workloads and OLTP–OLAP Separation

The separation of transactional (OLTP) and analytical (OLAP) workloads is a key outcome of the proposed design. By isolating analytical queries in a star-schema-based data warehouse, the system avoids performance degradation of critical transactional operations.

The star schema simplifies query formulation and supports efficient aggregation across multiple dimensions such as time, region, property type, and user role. Compared to normalized schemas, this dimensional approach offers superior performance for BI queries, at the cost of controlled redundancy. This trade-off is justified in analytical contexts where query speed is prioritized.

7.5 Materialized Views and BI Efficiency

The use of materialized views for pre-aggregated KPIs demonstrates a practical solution for achieving near-instantaneous BI query performance. By periodically refreshing analytical snapshots, the system eliminates the need for repeated full scans of the fact table.

While this approach improves performance, it introduces data freshness constraints. Analytical dashboards reflect the state of the system at the time of the last refresh rather than real-time data. In many business intelligence scenarios, this trade-off is acceptable, especially when balanced against improved performance and system stability.

7.6 Trade-Offs and Design Limitations

Although the proposed architecture provides strong guarantees in terms of consistency and scalability, it also presents trade-offs. Distributed databases introduce operational complexity, including node management and monitoring. Caching layers increase system performance but require careful invalidation strategies. Data warehouses and materialized views prioritize analytical efficiency over real-time accuracy.

These trade-offs reflect conscious design decisions aligned with the system's objectives and workload characteristics. Future work may explore automated cache invalidation, incremental materialized view refreshes, and dynamic repartitioning strategies to further enhance system adaptability.

7.7 Overall Evaluation

Overall, the results indicate that a relational, distributed, and cache-augmented architecture provides a robust foundation for data-intensive rental platforms. The design successfully balances transactional correctness, scalability, and analytical performance. The combination of PostgreSQL, Citus, Redis, and a dimensional data warehouse illustrates how traditional relational systems can be extended to support modern, high-demand applications without abandoning consistency guarantees.

Chapter 8

Conclusions

This study presented the analysis, design, and partial implementation of a database-centered architecture for Havenly, a property rental platform focused on transactional integrity, scalability, and analytical capabilities. The work demonstrates that a carefully designed relational architecture can effectively support both operational and analytical workloads in data-intensive marketplace systems.

The use of PostgreSQL as the core transactional database provided strong ACID guarantees, ensuring consistency and correctness in critical processes such as bookings and payments. Concurrency analysis confirmed that relational transaction management is well-suited to prevent anomalies such as double booking, which are common challenges in reservation-based systems.

Scalability requirements were addressed through horizontal distribution using the Citus extension. Partitioning data by geographical region enabled improved query performance, data locality, and parallel execution, while preserving the relational model and SQL compatibility. This approach demonstrates that distributed relational databases can scale effectively without requiring a full transition to non-relational technologies.

Performance optimization was further enhanced through the integration of Redis as an in-memory caching layer. By offloading read-heavy workloads from the transactional database, the system achieved reduced latency and improved responsiveness, while maintaining the database as the single source of truth.

From an analytical perspective, the separation of OLTP and OLAP workloads proved essential. The implementation of a star schema data warehouse enabled efficient multidimensional analysis, and the use of materialized views for pre-aggregated key performance indicators significantly improved query performance for business intelligence use cases. These design decisions illustrate the practical benefits of dimensional modeling and pre-aggregation in analytical systems.

Although the proposed architecture introduces additional operational complexity—particularly due to distribution, caching, and data synchronization—these trade-offs were justified by the gains in scalability, performance, and analytical efficiency. The results confirm that combining relational databases with distributed processing, caching mechanisms, and dedicated analytical structures provides a robust foundation for modern digital marketplaces.

Finally, this work establishes a solid conceptual and technical baseline for future extensions. Potential future work includes empirical performance benchmarking, automated cache invalidation strategies, incremental refresh of materialized views, and the integration of advanced analytics or machine learning techniques. Overall, the Havenly architecture demonstrates the continued relevance and adaptability of relational database systems in large-scale, data-driven applications.

8.1 Future work

Future work will focus on:

- Implementing frontend modules.
- Enhancing service orchestration and monitoring
- Expanding API.

Overall, Havenly demonstrates an academically grounded and technically feasible application of modern software architecture principles.

Acknowledgements

Juan Daniel Rodríguez Hurtado, Faider Camilo Trujillo Olaya and Santiago Sánchez wish to express sincere gratitude to Carlos Andrés Sierra and peers of the *Software Engineering Seminar* course at the Universidad Distrital Francisco José de Caldas for their guidance and collaboration during the development of this project. Special thanks are extended once again to the professor for his valuable feedback, which greatly contributed to the design and documentation of the *Havenly* platform.

Glossary of Terms

ACID Atomicity, Consistency, Isolation, Durability, the core properties of reliable transactional systems. 19

Business Intelligence (BI) Processes and tools used to analyze data and support decision-making. 19

Citus An extension for PostgreSQL that enables horizontal distribution of tables across multiple nodes, supporting scalability and parallel query execution. 19

Dimension Table Tables in a star schema that provide descriptive attributes for slicing and dicing fact metrics, such as date, property, or user information. 19

Fact Table The central table in a star schema that contains measurable metrics, such as bookings or revenue. 19

Materialized View A precomputed, stored view that aggregates data to accelerate analytical queries. 19

OLAP Online Analytical Processing, systems optimized for analytical queries, multidimensional analysis, and reporting, often using star or snowflake schemas. 19

OLTP Online Transaction Processing, a class of systems designed to manage transactional workloads, ensuring ACID compliance and supporting concurrent operations. 19

Redis An in-memory key-value store used for caching and fast data retrieval in read-heavy workloads. 19

RevPAR Revenue per Available Room, a key performance metric in hospitality calculated from total revenue and available room nights. 19

Slice-and-Dice Analytical operations that segment and filter data across different dimensions. 19

Star Schema A dimensional modeling design where a central fact table is connected to multiple dimension tables, optimized for analytical queries. 19

References

Airbnb (2025), 'Statistics and market analysis'. Available online: https://www.airbnb.com/press/news.

Data, C. (2025), *Distributed PostgreSQL Documentation*. Available online: https://www.citusdata.com/docs/.

Elmasri, R. and Navathe, S. B. (2017), *Fundamentals of Database Systems*, 7th edn, Pearson.

Institute, C. F. (2019), 'Business model canvas examples'. Available online: https://corporatefinanceinstitute.com/resources/knowledge/strategy/business-model-canvas/.

Kimball, R. and Ross, M. (2013), *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd edn, Wiley.

Labs, R. (2025), *Redis Documentation*. Available online: https://redis.io/documentation.

PostgreSQL Global Development Group (2025), *PostgreSQL Documentation*. Available online: https://www.postgresql.org/docs/.

Ramshaw, S. (2025), *FastAPI Documentation*. Available online: https://fastapi.tiangolo.com/.