HOTEL BOOKING SYSTEM

LUXURYSTAY – DATABASE DESIGN SPECIFICATION

CREATED BY – ASWINNATH TE

VERSION – 1.0

DATE – 10/10/2025

Hotel Booking System - DDS

1. INTRODUCTION

1.1 Purpose

This **Database Design Specification (DDS)** document outlines the detailed structure and design considerations for the **Hotel Booking System (HBS)**, an online platform for managing hotel operations, reservations, and customer engagement. The platform supports user registration, authentication, room and booking management, payments, refunds, issue tracking, feedback collection, and real-time notifications.

The system's **primary datastore is PostgreSQL (80% of data/workload)**, handling all core transactional, relational, and analytical use cases, while **MongoDB (20%)** is used for logs, content management, and backup metadata.

1.2 Scope

The HBS database supports:

- Secure management of user accounts (customers and administrators)
- Room type and amenity management
- Booking creation, modification, and cancellation workflows
- Payment and refund processing with full audit trails
- Issue reporting, tracking, and chat-based resolution
- Verified review management and admin response handling
- Centralized notification system for customers and admins
- Role-based access control and session management
- Reporting and analytics for hotel performance and user activity
- Metadata storage for enumerations, reference values, and audit logs

---

1.3 Definitions and Acronyms

DDS: Database Design Specification

PK: Primary Key

FK: Foreign Key (PostgreSQL)

_id: MongoDB Object ID (ObjectID)

TTL: Time To Live

RDBMS: Relational Database Management System

CMS: Content Management System

API: Application Programming Interface

ACID: Atomicity, Consistency, Isolation, Durability

---

2. DATABASE ARCHITECTURE

21 Database Management Systems Used

- **Architecture:** Polyglot Persistence
- **Primary DBMS:** PostgreSQL 15+ *(☐80% of data/workload)*
- **Secondary DBMS:** MongoDB 7+ *(☐20% of data/workload)*
- **Character Set (PostgreSQL):** UTF-8
- **Collation (PostgreSQL):** `en_US.UTF-8` or suitable locale-collation depending on deployment

---

22 Why Hybrid?

- **PostgreSQL** is the backbone of the system, responsible for maintaining **ACID compliance**, **relational integrity**, and **transactional operations** involving users, rooms, bookings, payments, refunds, issues, reviews, and notifications.
- **MongoDB** complements the relational core by handling **high-volume, unstructured, or dynamic data** such as logs, CMS content (offers, banners, FAQs, policies), and backup metadata.
- This **hybrid model** enables the platform to:
  - Scale efficiently for read-heavy, document-centric workloads.
  - Maintain referential consistency for mission-critical transactional data.

- Support analytics and reporting through PostgreSQL's advanced SQL capabilities.
- Enable low-latency data ingestion and flexible schema evolution via MongoDB.

---

23 Naming Conventions

- **Tables/Collections:** lowercase with underscores (e.g., `bookings`, `room_types`, `refund_logs`)
- **Columns/Fields:** lowercase with underscores (e.g., `booking_id`, `user_id`, `created_at`)
- **Primary Keys:**
  - PostgreSQL * `id` or `<table_name>_id` (usually SERIAL or UUID)
  - MongoDB *`_id` (ObjectID)
- **Foreign Keys:** `<referenced_table>_id` (e.g., `customer_id`, `room_id`, `booking_id`)
- **Indexes:** `idx_<table>_<column>` (e.g., `idx_bookings_customer_id`)
- **Constraints:** `fk_<table>_<column>`, `chk_<table>_<condition>`

---

24 Operational Considerations

- **Encryption & Security:**
  - Enable **column-level encryption** in PostgreSQL for sensitive data (e.g., payment details, credentials).
  - Enable **field-level encryption** in MongoDB for PII and security logs.
  - All connections between services and databases are secured via **TLS/SSL** with **VPC network isolation** and **role-based least-privilege access**.
- **Performance Optimization:**
  - Use **TTL indexes** in MongoDB for temporary collections (e.g., backup sessions, transient logs).
  - Apply **composite indexes** in PostgreSQL on high-frequency queries (e.g., `(customer_id, status)` in `bookings`).

    - Implement **descending indexes** on timestamp fields for fast retrieval of recent data.
    - Partition large PostgreSQL tables (like `bookings`, `payments`, `audit_log`) by year for scalability.
- **Audit & Recovery:**
    - Every transactional mutation in PostgreSQL triggers an **audit log entry**.
    - MongoDB stores immutable operational logs to support **forensic traceability** and **backup verification**.

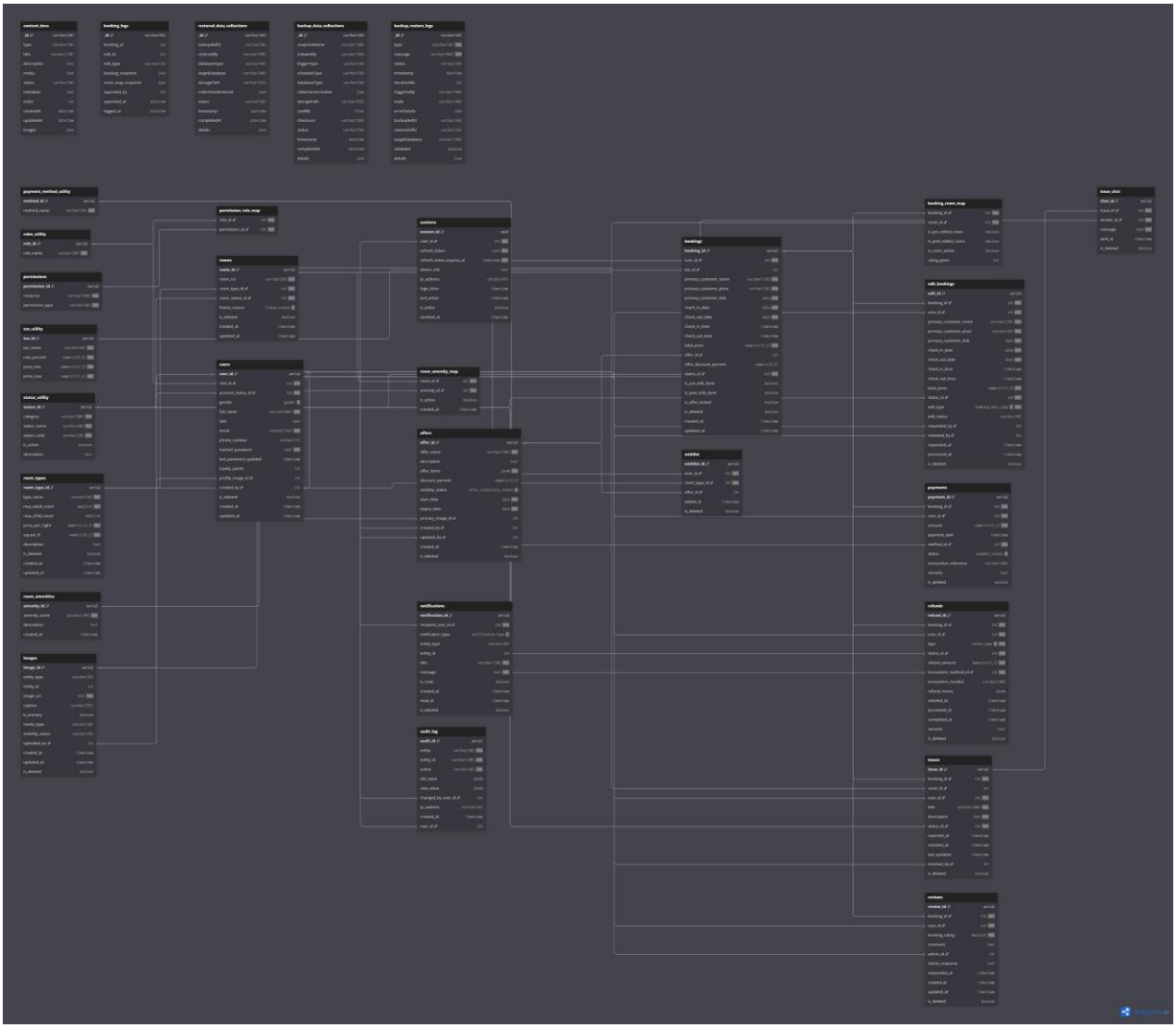
---

25 Additional Information

- PostgreSQL manages **core entities**: users, admins, rooms, bookings, payments, refunds, issues, reviews, notifications, and audit trails.
- MongoDB manages **supporting entities**: content documents (offers, banners, FAQs, hotel info), logs (API, backup, and admin activity), and backup metadata.
- All **cross-database references** use shared canonical keys (e.g., `booking_id`, `room_type_id`) for seamless correlation between SQL and NoSQL layers.
- Every major foreign key in PostgreSQL has a corresponding **B-tree index** for optimized joins and reporting.
- Application-level validation ensures synchronization between PostgreSQL and MongoDB entities, maintaining **data consistency across the hybrid layer**.

---

3. Entity Relationship Diagram



Link: https://dbdiagram.io/d/68e58561d2b621e422bd9b3f

3.1 Overview

The **Hotel Booking System (HBS)** implements a **polyglot persistence architecture**, integrating both **relational (PostgreSQL)** and **document-oriented (MongoDB)** data models.

This unified ERD consolidates **transactional data (PostgreSQL)**, **contextual extensions (MongoDB)**, and **event-driven operational logs**, enabling both **ACID**

reliability and **scalable flexibility** across hotel management workflows.

PostgreSQL anchors all mission-critical operations — users, rooms, bookings, payments, issues, and refunds — while MongoDB augments it with high-velocity, schema-flexible datasets such as logs, content documents, and backup metadata.

3.2 Core PostgreSQL Entities and Relationships

Entity	Description
users	Central identity registry for both customers and administrators.
users_utility	Reference table defining user roles and types (e.g., `CUSTOMER`, `ADMIN`).
customers	Stores customer profiles, personal info, contact details, and loyalty points.
customers_credentials	Contains encrypted password hashes for customers.
admins	Administrative accounts with hierarchical privileges.
admin_credentials	Stores encrypted admin passwords.
admin_permissions / admin_permission_map	Defines granular access rights and maps them to admins.
sessions	Tracks authentication sessions with JWT tokens and device metadata.
room_types	Defines room categories, pricing, and occupancy constraints.
rooms	Represents individual room inventory tied to a room type.
room_amenities / room_amenity_map	Defines amenities and many-to-many mappings between rooms and amenities.
bookings	Captures all reservation records, including duration, price, and customer linkage.
booking_room_map	Resolves many-to-many relationships between bookings and rooms.
booking_edit_history	Maintains revision history for booking modifications and reschedules.
payments	Tracks payment transactions, methods, and statuses.
refunds	Manages refund requests and workflow states.

Entity	Description
refund_room_map	Maps partial refunds to individual rooms for multi-room bookings.
issues	Customer-reported issues linked to specific bookings or rooms.
issue_chat	Message threads between customers and admins for resolving issues.
reviews / review_response	Captures customer reviews and corresponding single admin replies.
offers / offer_room_map	Manages promotional campaigns and their room-type associations.
notifications	Handles real-time and scheduled notifications for users and admins.
wishlist	Stores user-saved or favorited room types.
images	Centralized table storing URLs and metadata for images related to rooms, issues, users, and offers.
audit_log	Immutable record of all data-change events across SQL entities, mirrored in Mongo logs for compliance.

3.3 MongoDB Collections (Extensions)

Collection	Purpose	SQL References
api_logs	Captures all API request and response logs with metadata like latency, device, and user.	users , admins
content_docs	CMS-driven dynamic content such as banners, promotions, announcements, and testimonials.	admins
facilities_docs	Details on hotel facilities, categorized by type (wellness, dining, fitness, etc.).	None
hotel_info_docs	Contains informational sections like check-in/out policies and general hotel info.	None
policies_docs	Stores compliance policies — privacy, cancellation, and terms of service.	None
faqs_docs	Manages frequently asked questions and responses for frontend display.	None

3.4 Interaction Flow Summary

1. **Transactional Layer (PostgreSQL):**

   Manages all critical operations — user authentication, bookings, payments, refunds, issues, reviews, and notifications — under strict ACID constraints.

2. **Document Layer (MongoDB):**

   Extends the system with append-only logs, CMS content, and metadata for operational insights, backup audits, and non-transactional content delivery.

3. **API Bridge (Cross-Sync):**

   - PostgreSQL triggers push structured events to MongoDB (e.g., new booking → insert booking log).
   - MongoDB enriches API responses with contextual overlays (e.g., content, facilities, logs).

4. **Administrative Orchestration:**

   Admins operate as control nodes — governing content updates, issue resolution, and audit validation — bridging both databases for a unified operational experience.

4. TABLE SPECIFICATIONS

4.1 UTILITY TABLES and TYPES

`roles_utility`

Purpose:

Defines user role types such as `ADMIN`, `SUPERADMIN`, `CUSTOMER`, or system roles used for access segregation and RBAC enforcement.

Column	Type	Constraints	Description
role_id	SERIAL	**PK**	Unique role identifier
role_name	VARCHAR(50)	**UNIQUE NOT NULL**	Role label (`ADMIN`, `CUSTOMER`, etc.)

payment_method_utility

Purpose:

Lists all supported payment methods available within the system.

Column	Type	Constraints	Description
method_id	SERIAL	**PK**	Unique payment method identifier
method_name	VARCHAR(50)	**UNIQUE NOT NULL**	Name of payment method (`Credit Card` , `UPI` , `Wallet` , etc.)

status_utility

Purpose:

Acts as a **universal status registry** that maintains **state definitions** across modules like bookings, refunds, rooms, and users.

Instead of having multiple domain-specific status tables, all status types are unified under one master table, with a `category` field used for logical grouping.

Column	Type	Constraints	Description
status_id	SERIAL	**PK**	Unique identifier for each status
category	VARCHAR(100)	**NOT NULL**	Functional category (e.g., `BOOKING` , `ROOM` , `USER` , `REFUND`)
status_name	VARCHAR(50)	**UNIQUE NOT NULL**	Canonical readable name (e.g., `CONFIRMED` , `PENDING` , `ACTIVE`)
status_code	VARCHAR(20)	**UNIQUE NOT NULL**	Symbolic status code (`BK_CONFIRMED` , `RM_OCCUPIED` , etc.)
is_active	BOOLEAN	**DEFAULT TRUE**	Marks if the status is currently used in the system
description	TEXT	NULL	Optional context or note for administrators

permissions

Purpose:

Defines individual permission tokens that control access to specific modules or actions within the system.

Used for **Role-Based Access Control (RBAC)** in combination with `permission_role_map`.

Column	Type	Constraints	Description
permission_id	SERIAL	**PK**	Unique identifier for the permission
resource	VARCHAR(100)	**UNIQUE NOT NULL**	The protected module or API (e.g., `MANAGE_BOOKINGS`, `VIEW_PAYMENTS`)
permission_type	VARCHAR(50)	**NOT NULL**	Access type (`READ`, `WRITE`, `MANAGE`, `DELETE`)

permission_role_map

Purpose:

Maps **roles** to **permissions** — determining which actions are allowed for each role type.

Supports one-to-many and many-to-many relationships between roles and permissions.

Column	Type	Constraints	Description
role_id	INT	**NOT NULL**, FK * `roles_utility(role_id)`	Role being granted the permission
permission_id	INT	**NOT NULL**, FK * `permissions(permission_id)`	Linked permission assigned to the role

tax_utility

Purpose:

Stores configurable tax slabs based on price range and rate.

Used to compute taxes dynamically for bookings during checkout and refund processes.

Column	**Type**	**Constraints**	**Description**
tax_id	SERIAL	**PK**	Unique tax identifier
tax_name	VARCHAR(50)	**UNIQUE NOT NULL**	Name of the tax slab (`Standard Tax`, `Luxury Tax`, etc.)

Column	Type	Constraints	Description
rate_percent	NUMERIC(5,2)	**NOT NULL**	Percentage rate applied to total amount
price_min	NUMERIC(12,2)	**NOT NULL**	Lower bound for applicable price range
price_max	NUMERIC(12,2)	**NOT NULL**	Upper bound for applicable price range

ENUM TYPE DEFINITIONS

These enumerations represent **system-level static types** used across multiple entities in the database.

They ensure strict domain integrity and reduce dependency on auxiliary lookup tables for fixed categorical data.

```
--  ========================================================
===
-- ENUM TYPE DEFINITIONS (As per DBML v3.0)
--  ========================================================
===

CREATE TYPE freeze_reason AS ENUM ('MAINTENANCE', 'ADMIN_LOCK');
CREATE TYPE payment_status AS ENUM ('SUCCESS', 'PENDING', 'FAILED');
CREATE TYPE refund_type AS ENUM ('FULL_BOOKING', 'PARTIAL_ROOM');
CREATE TYPE booking_edit_type AS ENUM ('PRE', 'POST');
CREATE TYPE offer_visibility_status AS ENUM ('ACTIVE', 'INACTIVE', 'SCHEDULE
D', 'EXPIRED');
CREATE TYPE notification_type AS ENUM ('SYSTEM', 'PROMOTIONAL', 'REMIND
ER', 'BOOKING_EVENT', 'OTHER');
CREATE TYPE gender AS ENUM ('Male', 'Female', 'other');
```

4.2 USER and AUTH TABLES

users

Purpose:

Central identity table consolidating personal details, roles, authentication data, and lifecycle states for all system users.

Column	Type	Constraints	Description
user_id	SERIAL	**PK**	Unique identifier for each user.
role_id	INT	**NOT NULL**, FK * `roles_utility(role_id)`	Defines user role (`CUSTOMER` , `ADMIN` , `SUPERADMIN` , etc.).
account_status_id	INT	**NOT NULL**, FK * `status_utility(status_id)`	Current lifecycle state of the user account.
gender	gender	NULL	ENUM representing user's gender (`Male` , `Female` , `Other`).
full_name	VARCHAR(200)	**NOT NULL**	Full legal name of the user.
dob	DATE	NULL	Date of birth.
email	VARCHAR(150)	**UNIQUE NOT NULL**	Primary email for login and communication.
phone_number	VARCHAR(15)	**UNIQUE**	Contact number.
hashed_password	TEXT	**NOT NULL**	Securely hashed password.
last_password_updated	TIMESTAMP	DEFAULT now()	Timestamp of last password change.
loyalty_points	INT	DEFAULT 0	Accumulated reward points for customers.
profile_image_id	INT	FK * `images(image_id)` ON DELETE SET NULL	Reference to user's profile image.
created_by	INT	FK * `users(user_id)` ON DELETE SET NULL	References user who created the record.
is_deleted	BOOLEAN	DEFAULT FALSE	Logical flag for soft deletion.
created_at	TIMESTAMP	DEFAULT now()	Record creation timestamp.

Column	Type	Constraints	Description
updated_at	TIMESTAMP	DEFAULT now()	Last record update timestamp.

sessions

Purpose:

Tracks all active and historical login sessions for every user.

Column	Type	Constraints	Description
session_id	UUID	**PK**	Unique session identifier.
user_id	INT	**NOT NULL**, FK * users(user_id)	Associated user account.
refresh_token	TEXT	**UNIQUE NOT NULL**	Refresh token used for reauthentication.
refresh_token_expires_at	TIMESTAMP	**NOT NULL**	Expiry timestamp of the refresh token.
device_info	TEXT	NULL	Metadata about the login device (browser, OS, etc.).
ip_address	VARCHAR(45)	NULL	IPv4 or IPv6 address.
login_time	TIMESTAMP	DEFAULT now()	Session creation timestamp.
last_active	TIMESTAMP	DEFAULT now()	Last recorded activity timestamp.
is_active	BOOLEAN	DEFAULT TRUE	Session validity flag.
revoked_at	TIMESTAMP	NULL	Timestamp of session revocation.

4.3 ROOMS MANAGEMENT

room_types

Purpose:

Defines standardized categories of rooms with shared pricing, capacity, and specifications.

Column	Type	Constraints	Description
room_type_id	SERIAL	**PK**	Unique identifier for each room type.
type_name	VARCHAR(50)	**UNIQUE NOT NULL**	Canonical name of the room type.
max_adult_count	SMALLINT	**NOT NULL CHECK (max_adult_count > 1)**	Maximum number of adults allowed.
max_child_count	SMALLINT	**DEFAULT 0 CHECK (max_child_count > 0)**	Maximum number of children allowed.
price_per_night	NUMERIC(12,2)	**NOT NULL CHECK (price_per_night > 0)**	Base nightly rate.
square_ft	NUMERIC(8,2)	**NOT NULL**	Physical size of the room.
description	TEXT	NULL	Descriptive details about the room type.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Marks record as logically deleted.
created_at	TIMESTAMP	**DEFAULT now()**	Record creation timestamp.
updated_at	TIMESTAMP	**DEFAULT now()**	Last update timestamp.

room_amenities

Purpose:

Defines all available amenities that can be associated with one or more rooms.

Column	Type	Constraints	Description
amenity_id	SERIAL	**PK**	Unique identifier for each amenity.
amenity_name	VARCHAR(100)	**UNIQUE NOT NULL**	Name of the amenity.

rooms

Purpose:

Represents the physical room inventory linked to predefined room types.

Column	Type	Constraints	Description
room_id	SERIAL	**PK**	Unique identifier for each room.
room_no	VARCHAR(20)	**UNIQUE NOT NULL**	Physical or display room number.
room_type_id	INT	**NOT NULL, FK ➔ room_types(room_type_id)**	Reference to defined room type.
room_status_id	INT	**NOT NULL, FK ➔ status_utility(status_id)**	Current room lifecycle state.
freeze_reason	freeze_reason	NULL	ENUM reason for maintenance or lock.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Soft delete flag.
created_at	TIMESTAMP	**DEFAULT now()**	Record creation timestamp.
updated_at	TIMESTAMP	**DEFAULT now()**	Record update timestamp.

room_amenity_map

Purpose:

Defines a many-to-many relationship between rooms and amenities.

Column	Type	Constraints	Description
room_id	INT	**NOT NULL, FK ➔ rooms(room_id)**	Linked room.
amenity_id	INT	**NOT NULL, FK ➔ room_amenities(amenity_id)**	Linked amenity.
is_active	BOOLEAN	**DEFAULT TRUE**	Active status of the room-amenity mapping.
created_at	TIMESTAMP	**DEFAULT now()**	Mapping creation timestamp.

4.4 BOOKING MANAGEMENT

bookings

Purpose:

Stores finalized booking details and serves as the master record for all confirmed reservations.

Column	Type	Constraints	Description
booking_id	SERIAL	**PK**	Unique booking record.
user_id	INT	**NOT NULL**, FK * users(user_id)	Customer who made the booking.
tax_id	INT	FK * tax_utility(tax_id)	Applied tax slab reference.
primary_customer_name	VARCHAR(150)	**NOT NULL**	Primary adult guest's full name.
primary_customer_phno	VARCHAR(20)	**NOT NULL**	Registered mobile number of the primary adult guest.
primary_customer_dob	DATE	**NOT NULL CHECK (primary_customer_dob ›› CURRENT_DATE - INTERVAL '18 years')**	Date of birth of primary guest; must be 18+.
check_in_date	DATE	**NOT NULL**	Scheduled check-in date.
check_out_date	DATE	**NOT NULL**	Scheduled check-out date.
check_in_time	TIMESTAMP	NULL	Actual check-in timestamp.
check_out_time	TIMESTAMP	NULL	Actual check-out timestamp.
total_price	NUMERIC(12,2)	**NOT NULL CHECK (total_price ›› 0)**	Total payable amount after

Column	Type	Constraints	Description
			taxes and discounts.
offer_id	INT	FK * `offers(offer_id)` ON DELETE SET NULL	Linked promotional offer, if any.
offer_discount_percent	NUMERIC(5,2)	NULL	Discount percentage applied to the booking.
status_id	INT	**NOT NULL**, FK * `status_utility(status_id)`	Current booking lifecycle status.
is_pre_edit_done	BOOLEAN	DEFAULT FALSE	Indicates if pre-check-in edit was done.
is_post_edit_done	BOOLEAN	DEFAULT FALSE	Indicates if post-check-out edit was done.
is_offer_locked	BOOLEAN	DEFAULT FALSE	Marks booking as linked to an offer.
is_deleted	BOOLEAN	DEFAULT FALSE	Logical deletion flag.
created_at	TIMESTAMP	DEFAULT now()	Booking creation timestamp.
updated_at	TIMESTAMP	DEFAULT now()	Last modification timestamp.

`booking_room_map`

Purpose:

Defines the mapping between bookings and their assigned rooms, supporting pre- and post-edit scenarios.

Column	Type	Constraints	Description
booking_id	INT	**NOT NULL**, FK * `bookings(booking_id)` ON DELETE CASCADE	Parent booking reference.

Column	Type	Constraints	Description
room_id	INT	**NOT NULL**, FK * rooms(room_id)	Linked room reference.
is_pre_edited_room	BOOLEAN	DEFAULT FALSE	Marks room as modified in pre-check-in edit.
is_post_edited_room	BOOLEAN	DEFAULT FALSE	Marks room as modified in post-check-out edit.
is_room_active	BOOLEAN	DEFAULT TRUE	Indicates if the room is active under this booking.
rating_given	INT	DEFAULT 0	Captures post-stay user rating at room level.

edit_bookings

Purpose:

Captures proposed booking edits before or after check-in, mirroring the structure of the main booking record for administrative approval.

Column	Type	Constraints	Description
edit_id	SERIAL	**PK**	Unique edit record.
booking_id	INT	**NOT NULL**, FK * bookings(booking_id)	Linked booking being edited.
user_id	INT	**NOT NULL**, FK * users(user_id)	Customer requesting the edit.
primary_customer_name	VARCHAR(150)	**NOT NULL**	Updated name of primary adult guest.
primary_customer_phno	VARCHAR(20)	**NOT NULL**	Updated contact number.
primary_customer_dob	DATE	**NOT NULL**	Updated date of birth of primary guest.

Column	Type	Constraints	Description
check_in_date	DATE	**NOT NULL**	Proposed new check-in date.
check_out_date	DATE	**NOT NULL**	Proposed new check-out date.
check_in_time	TIMESTAMP	NULL	Proposed check-in time.
check_out_time	TIMESTAMP	NULL	Proposed check-out time.
total_price	NUMERIC(12,2)	**NOT NULL CHECK (total_price ‹ › 0)**	Proposed total price after adjustments.
status_id	INT	**NOT NULL**, FK * `status_utility(status_id)`	Updated booking status.
edit_type	booking_edit_type	**NOT NULL**	Edit type ENUM: `PRE` or `POST`.
edit_status	VARCHAR(50)	DEFAULT 'PENDING'	Edit processing state: `PENDING`, `APPROVED`, `REJECTED`.
requested_by	INT	FK * `users(user_id)`	User initiating the edit.
reviewed_by	INT	FK * `users(user_id)` ON DELETE SET NULL	Admin who reviewed the edit.
requested_at	TIMESTAMP	DEFAULT now()	Timestamp of edit request.
processed_at	TIMESTAMP	NULL	Timestamp of admin processing.
is_deleted	BOOLEAN	DEFAULT FALSE	Logical deletion flag.

4.5 FINANCIALS

payments

Purpose:

Logs all payment transactions associated with bookings, capturing payment method, lifecycle state, and external transaction details.

Column	Type	Constraints	Description
payment_id	SERIAL	**PK**	Unique payment identifier.
booking_id	INT	**NOT NULL**, FK * `bookings(booking_id)` ON DELETE RESTRICT	Linked booking reference.
user_id	INT	**NOT NULL**, FK * `users(user_id)`	User who made the payment.
amount	NUMERIC(12,2)	**NOT NULL CHECK (amount > 0)**	Payment amount.
payment_date	TIMESTAMP	**DEFAULT now()**	Timestamp of payment.
method_id	INT	**NOT NULL**, FK * `payment_method_utility(method_id)`	Payment method used.
status	payment_status	**DEFAULT 'PENDING'**	Payment lifecycle state (`SUCCESS`, `PENDING`, `FAILED`).
transaction_reference	VARCHAR(100)	**UNIQUE**	External transaction or gateway reference.
remarks	TEXT	NULL	Additional payment notes or comments.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Logical deletion flag.

`refunds`

Purpose:

Tracks refund requests, approvals, and completions linked to bookings and users, supporting both full and partial refund workflows.

Column	Type	Constraints	Description
refund_id	SERIAL	**PK**	Unique refund identifier.
booking_id	INT	**NOT NULL**, FK * bookings(booking_id) ON DELETE RESTRICT	Linked booking reference.
user_id	INT	**NOT NULL**, FK * users(user_id)	User requesting the refund.
type	refund_type	**NOT NULL**	Refund classification (FULL_BOOKING , PARTIAL_ROOM).
status_id	INT	**NOT NULL**, FK * status_utility(status_id)	Current refund lifecycle status.
refund_amount	NUMERIC(12,2)	**NOT NULL CHECK (refund_amount > 0)**	Total refunded amount.
transaction_method_id	INT	**NOT NULL**, FK * payment_method_utility(method_id)	Refund transaction method.
transaction_number	VARCHAR(100)	NULL	Admin or gateway transaction reference.
refund_rooms	JSONB	NULL	List of refunded rooms and amounts (for partial refunds).
initiated_at	TIMESTAMP	**DEFAULT now()**	Timestamp of refund initiation.
processed_at	TIMESTAMP	NULL	Timestamp of refund processing.
completed_at	TIMESTAMP	NULL	Timestamp of refund completion.
remarks	TEXT	NULL	Administrative notes or

Column	Type	Constraints	Description
			comments.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Logical deletion flag.

4.6 ISSUES MANAGEMENT

issues

Purpose:

Tracks all user-reported issues or complaints related to bookings or rooms.

Supports both customer-raised and staff-raised issues with complete lifecycle traceability.

Column	Type	Constraints	Description
issue_id	SERIAL	**PK**	Unique issue identifier.
booking_id	INT	**NOT NULL**, FK * `bookings(booking_id)` ON DELETE RESTRICT	Associated booking reference.
room_id	INT	FK * `rooms(room_id)`	Linked room reference, if applicable.
user_id	INT	**NOT NULL**, FK * `users(user_id)`	User who reported the issue.
title	VARCHAR(200)	**NOT NULL**	Short descriptive title for the issue.
description	TEXT	**NOT NULL**	Detailed issue description.
status_id	INT	**NOT NULL**, FK * `status_utility(status_id)`	Current issue lifecycle status.
reported_at	TIMESTAMP	**DEFAULT now()**	Timestamp when the issue was reported.
resolved_at	TIMESTAMP	NULL	Timestamp when the issue was resolved.

Column	Type	Constraints	Description
last_updated	TIMESTAMP	**DEFAULT now()**	Last modification timestamp.
resolved_by	INT	FK * `users(user_id)` ON DELETE SET NULL	User who resolved or closed the issue.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Logical deletion flag.

issue_chat

Purpose:

Stores threaded communication between users (e.g., customer ↔ admin) within an issue for contextual discussion and resolution tracking.

Column	Type	Constraints	Description
chat_id	SERIAL	**PK**	Unique chat identifier.
issue_id	INT	**NOT NULL**, FK * `issues(issue_id)` ON DELETE CASCADE	Linked issue reference.
sender_id	INT	**NOT NULL**, FK * `users(user_id)`	User who sent the message.
message	TEXT	**NOT NULL**	Chat message content.
sent_at	TIMESTAMP	**DEFAULT now()**	Message sent timestamp.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Logical deletion flag.

4.7 IMAGES MANAGEMENT

images

Purpose:

Generic image registry managing media references across multiple entities in the system.

Column	Type	Constraints	Description
image_id	SERIAL	**PK**	Unique image identifier.
entity_type	VARCHAR(50)	NULL	Entity type this image belongs to (`ROOM_TYPE` , `OFFER` , `REVIEW` , `BOOKING` , `USER` , etc.).
entity_id	INT	NULL	Linked entity record ID.
image_url	TEXT	**UNIQUE NOT NULL**	Path or URL of the stored image.
caption	VARCHAR(255)	NULL	Descriptive caption for the image.
is_primary	BOOLEAN	**DEFAULT FALSE**	Marks this image as the primary display image.
media_type	VARCHAR(20)	**DEFAULT 'image' CHECK (media_type IN ('image','video','thumbnail'))**	Media classification.
visibility_status	VARCHAR(20)	**DEFAULT 'VISIBLE' CHECK (visibility_status IN ('VISIBLE','HIDDEN','ARCHIVED'))**	Controls visibility status.
uploaded_by	INT	FK * `users(user_id)` ON DELETE SET NULL	User who uploaded the image.
created_at	TIMESTAMP	**DEFAULT now()**	Creation timestamp.
updated_at	TIMESTAMP	**DEFAULT now()**	Last modification timestamp.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Soft delete flag.

4.8 REVIEWS SYSTEM

reviews

Purpose:

Captures both public reviews for room types and private feedback for completed bookings.

Column	Type	Constraints	Description
review_id	SERIAL	**PK**	Unique review identifier.
booking_id	INT	**NOT NULL**, FK * bookings(booking_id) ON DELETE RESTRICT	Associated booking.
user_id	INT	**NOT NULL**, FK * users(user_id)	Customer submitting the review.
room_type_id	INT	FK * room_types(room_type_id)	Reviewed room type reference (nullable for booking-only reviews).
booking_rating	SMALLINT	**NOT NULL CHECK (booking_rating BETWEEN 1 AND 5)**	Rating score (1–5).
comment	TEXT	NULL	User review or feedback content.
admin_id	INT	FK * users(user_id)	Admin who responded.
admin_response	TEXT	NULL	Admin reply text.
responded_at	TIMESTAMP	NULL	Timestamp when admin responded.
created_at	TIMESTAMP	**DEFAULT now()**	Review creation timestamp.
updated_at	TIMESTAMP	**DEFAULT now()**	Last modification timestamp.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Logical deletion flag.

4.9 OFFERS MANAGEMENT

offers

Purpose:

Defines promotional offers and discount campaigns applicable to one or more room types.

Column	Type	Constraints	Description
offer_id	SERIAL	**PK**	Unique offer identifier.
offer_name	VARCHAR(100)	**UNIQUE NOT NULL**	Offer title.
description	TEXT	NULL	Description of the offer.
offer_items	JSONB	**NOT NULL**	Array of linked room types and pricing rules.
discount_percent	NUMERIC(5,2)	**CHECK (discount_percent BETWEEN 0 AND 100)**	Discount percentage.
visibility_status	offer_visibility_status	**DEFAULT 'SCHEDULED'**	Offer visibility lifecycle.
start_date	DATE	**NOT NULL**	Offer start date.
expiry_date	DATE	**NOT NULL CHECK (start_date <= expiry_date)**	Offer expiry date.
primary_image_id	INT	FK * `images(image_id)` ON DELETE SET NULL	Primary offer image.
created_by	INT	FK * `users(user_id)` ON DELETE SET NULL	Admin who created the offer.
updated_by	INT	FK * `users(user_id)` ON DELETE SET NULL	Admin who last updated the offer.
created_at	TIMESTAMP	**DEFAULT now()**	Offer creation timestamp.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Soft delete flag.

4.10 NOTIFICATIONS SYSTEM

`notifications`

Purpose:

Handles both real-time and scheduled notifications for users and admins, linked to key system events.

Column	Type	Constraints	Description
notification_id	SERIAL	**PK**	Unique notification identifier.
recipient_user_id	INT	**NOT NULL**, FK * `users(user_id)` ON DELETE CASCADE	User receiving the notification.
notification_type	notification_type	**DEFAULT 'OTHER'**	Type of notification (`SYSTEM` , `PROMOTIONAL` , `REMINDER` , etc.).
entity_type	VARCHAR(50)	NULL	Related entity type (e.g., `BOOKING` , `PAYMENT` , `REFUND`).
entity_id	INT	NULL	Linked entity record.
title	VARCHAR(150)	**NOT NULL**	Notification title.
message	TEXT	**NOT NULL**	Notification content.
is_read	BOOLEAN	**DEFAULT FALSE**	Marks if notification is read.
created_at	TIMESTAMP	**DEFAULT now()**	Notification creation timestamp.
read_at	TIMESTAMP	NULL	When notification was read.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Logical deletion flag.

4.11 WISHLIST SYSTEM

`wishlist`

Purpose:

Stores room types that users have favorited or saved for later bookings, optionally linked to offers.

Column	Type	Constraints	Description
wishlist_id	SERIAL	**PK**	Unique wishlist record.

Column	Type	Constraints	Description
user_id	INT	**NOT NULL**, FK * `users(user_id)` ON DELETE CASCADE	User who saved the room type.
room_type_id	INT	**NOT NULL**, FK * `room_types(room_type_id)`	Saved room type.
offer_id	INT	FK * `offers(offer_id)` ON DELETE SET NULL	Linked promotional offer.
added_at	TIMESTAMP	**DEFAULT now()**	Timestamp when added to wishlist.
is_deleted	BOOLEAN	**DEFAULT FALSE**	Logical deletion flag.

4.12 Indexes of tables

```
-------------------------------------------
-- USER & AUTH SUBSYSTEM
-------------------------------------------

CREATE INDEX idx_users_role_status ON users (role_id, account_status_id);
CREATE INDEX idx_users_email ON users (email);
CREATE INDEX idx_users_phone ON users (phone_number);
CREATE INDEX idx_users_is_deleted ON users (is_deleted) WHERE is_deleted ›
FALSE;

CREATE INDEX idx_sessions_user_active ON sessions (user_id, is_active);
CREATE INDEX idx_sessions_last_active ON sessions (last_active DESC);

-------------------------------------------
-- PERMISSIONS & RBAC MANAGEMENT
-------------------------------------------

CREATE INDEX idx_permissions_resource_type ON permissions (resource, permi
ssion_type);
CREATE INDEX idx_permission_role_map_role_perm ON permission_role_map (r
ole_id, permission_id);
```

```
-------------------------------------------
-- ROOMS & AMENITIES
-------------------------------------------

CREATE INDEX idx_room_types_name ON room_types (type_name);
CREATE INDEX idx_room_types_price ON room_types (price_per_night);

CREATE INDEX idx_rooms_type_status ON rooms (room_type_id, room_status_i
d);
CREATE INDEX idx_rooms_is_deleted ON rooms (is_deleted) WHERE is_deleted ›
FALSE;

CREATE INDEX idx_room_amenities_name ON room_amenities (amenity_name);
CREATE INDEX idx_room_amenity_map_room_amenity ON room_amenity_map (r
oom_id, amenity_id);

-------------------------------------------
-- BOOKINGS & EDIT MANAGEMENT
-------------------------------------------

CREATE INDEX idx_bookings_user_status ON bookings (user_id, status_id);
CREATE INDEX idx_bookings_offer ON bookings (offer_id);
CREATE INDEX idx_bookings_created_at ON bookings (created_at DESC);
CREATE INDEX idx_bookings_is_deleted ON bookings (is_deleted) WHERE is_del
eted ›  FALSE;

CREATE INDEX idx_booking_room_map_booking ON booking_room_map (bookin
g_id);
CREATE INDEX idx_booking_room_map_room ON booking_room_map (room_id);
CREATE INDEX idx_booking_room_map_flags
   ON booking_room_map (is_pre_edited_room, is_post_edited_room, is_room_ac
tive);

CREATE INDEX idx_edit_bookings_booking_type ON edit_bookings (booking_id,
edit_type);
CREATE INDEX idx_edit_bookings_status ON edit_bookings (edit_status);
CREATE INDEX idx_edit_bookings_requested_at ON edit_bookings (requested_at
```

```
DESC);
CREATE INDEX idx_edit_bookings_is_deleted ON edit_bookings (is_deleted) WHE
RE is_deleted > FALSE;

-- ------------------------------------------
-- PAYMENTS & REFUNDS
-- ------------------------------------------

CREATE INDEX idx_payments_booking_status ON payments (booking_id, status);
CREATE INDEX idx_payments_method ON payments (method_id);
CREATE INDEX idx_payments_date ON payments (payment_date DESC);
CREATE INDEX idx_payments_is_deleted ON payments (is_deleted) WHERE is_d
eleted > FALSE;

CREATE INDEX idx_refunds_booking_status ON refunds (booking_id, status_id);
CREATE INDEX idx_refunds_user ON refunds (user_id);
CREATE INDEX idx_refunds_initiated_at ON refunds (initiated_at DESC);
CREATE INDEX idx_refunds_is_deleted ON refunds (is_deleted) WHERE is_delete
d > FALSE;

-- ------------------------------------------
-- ISSUES MANAGEMENT
-- ------------------------------------------

CREATE INDEX idx_issues_booking_status ON issues (booking_id, status_id);
CREATE INDEX idx_issues_user ON issues (user_id);
CREATE INDEX idx_issues_reported_at ON issues (reported_at DESC);
CREATE INDEX idx_issues_is_deleted ON issues (is_deleted) WHERE is_deleted
> FALSE;

CREATE INDEX idx_issue_chat_issue ON issue_chat (issue_id);
CREATE INDEX idx_issue_chat_sender ON issue_chat (sender_id);
CREATE INDEX idx_issue_chat_sent_at ON issue_chat (sent_at DESC);

-- ------------------------------------------
-- CENTRALIZED IMAGES MANAGEMENT
-- ------------------------------------------
```

```
CREATE INDEX idx_images_entity ON images (entity_type, entity_id);
CREATE INDEX idx_images_uploaded_by ON images (uploaded_by);
CREATE INDEX idx_images_is_primary ON images (is_primary);
CREATE INDEX idx_images_visibility ON images (visibility_status);
CREATE INDEX idx_images_is_deleted ON images (is_deleted) WHERE is_deleted
› FALSE;

--------------------------------------------
-- REVIEWS SYSTEM
--------------------------------------------

CREATE INDEX idx_reviews_booking_user ON reviews (booking_id, user_id);
CREATE INDEX idx_reviews_room_type ON reviews (room_type_id);
CREATE INDEX idx_reviews_rating ON reviews (booking_rating);
CREATE INDEX idx_reviews_created_at ON reviews (created_at DESC);
CREATE INDEX idx_reviews_is_deleted ON reviews (is_deleted) WHERE is_delete
d › FALSE;

--------------------------------------------
-- OFFERS MANAGEMENT
--------------------------------------------

CREATE INDEX idx_offers_name ON offers (offer_name);
CREATE INDEX idx_offers_date_range ON offers (start_date, expiry_date);
CREATE INDEX idx_offers_created_by ON offers (created_by);
CREATE INDEX idx_offers_visibility_status ON offers (visibility_status);
CREATE INDEX idx_offers_is_deleted ON offers (is_deleted) WHERE is_deleted ›
FALSE;

--------------------------------------------
-- NOTIFICATIONS SYSTEM
--------------------------------------------

CREATE INDEX idx_notifications_user_read ON notifications (recipient_user_id, is
_read);
CREATE INDEX idx_notifications_created_at ON notifications (created_at DESC);
```

```
CREATE INDEX idx_notifications_is_deleted ON notifications (is_deleted) WHERE is_deleted > FALSE;

-------------------------------------------
-- WISHLIST SYSTEM
-------------------------------------------

CREATE INDEX idx_wishlist_user_room ON wishlist (user_id, room_type_id);
CREATE INDEX idx_wishlist_offer ON wishlist (offer_id);
CREATE INDEX idx_wishlist_added_at ON wishlist (added_at DESC);
CREATE INDEX idx_wishlist_is_deleted ON wishlist (is_deleted) WHERE is_deleted > FALSE;

-------------------------------------------
-- AUDIT & LOG MANAGEMENT
-------------------------------------------

CREATE INDEX idx_audit_entity ON audit_log (entity, entity_id);
CREATE INDEX idx_audit_changed_by ON audit_log (changed_by_user_id);
CREATE INDEX idx_audit_created_at ON audit_log (created_at DESC);
```

5. MONGO COLLECTIONS

5.1 LOG COLLECTIONS

audit_log

Purpose:

Even though MongoDB holds the majority of logs, this minimal **Postgres table** ensures **compliance and audit integrity** by maintaining pointers to Mongo log entries.

Column	Type	Constraints	Description
audit_id	SERIAL	PK	Unique audit identifier

Column	Type	Constraints	Description
entity	VARCHAR(50)	NOT NULL	e.g., `booking`, `room`
entity_id	VARCHAR(100)	NOT NULL	Canonical record reference
action	VARCHAR(20)	NOT NULL	`INSERT`, `UPDATE`, `DELETE`
old_value	JSONB	NULL	Previous state
new_value	JSONB	NULL	Updated state
changed_by_user_id	INT	FK * users(user_id) ON DELETE SET NULL	User/admin ID who done the change
ip_address	VARCHAR(45)	NULL	Origin IP
created_at	TIMESTAMP	DEFAULT now()	Timestamp of change
user_id	INT	NOT NULL REFERENCES `users(user_id)` ON DELETE set NULL	Receiver of the notification

Indexes:

```
CREATE INDEX idx_audit_entity ON audit_log(entity, entity_id);
```

backup_restore_logs

Purpose:

The `backup_restore_logs` collection serves as a **centralized logging repository** for all **backup** and **restore** process-level events.

Each document represents a **discrete operational event**, recording execution details, timing metrics, and system feedback from automated or manual backup/restore workflows.

By consolidating both processes into a unified schema, this collection enhances **log traceability**, **index efficiency**, and **monitoring clarity** across the HBS infrastructure.

Field	**Type**	**Constraints**	**Description**
_id	`ObjectId / string`	**Primary Key**	Unique identifier for each log record.
type	`string`	**NOT NULL**	Defines log category: `backup` or `restore`.

Field	Type	Constraints	Description
message	string	**NOT NULL**	Descriptive event or task message.
status	string	NULLABLE	Log outcome: info, success, warning, or error.
timestamp	datetime	**DEFAULT** › **current timestamp**	Event creation time in UTC.
durationMs	int	NULLABLE	Duration in milliseconds for the event or operation.
triggeredBy	string	NULLABLE	Source initiating the operation — e.g., system_scheduler, admin_102.
node	string	NULLABLE	Node, instance, or server name executing the process.
errorDetails	object / json	NULLABLE	Structured error or failure trace (stack, message, response codes, etc.).
backupRefId	string	NULLABLE	Reference to the parent backup job (backup_data_collections._id), if applicable.
restoreRefId	string	NULLABLE	Reference to the restore job (restored_data_collections._id), if applicable.
targetDatabase	string	NULLABLE	For restore operations — target DB environment (prod_clone, staging_env, etc.).
validated	boolean	NULLABLE	Indicates if post-restore validation or checksum passed.
details	object / json	NULLABLE	Extended event metadata (compression, restoreMode, checksumVerified, etc.).

booking_logs

Purpose:

Stores immutable snapshots of bookings and their room mappings whenever an edit is approved.

Managed in MongoDB for scalability and long-term audit retention.

Field	Type	Description
_id	ObjectId	MongoDB document ID
booking_id	INT	References bookings.booking_id
edit_id	INT	References edit_bookings.edit_id
edit_type	ENUM ⌫'PRE', 'POST']	Type of edit that generated the log
booking_snapshot	OBJECT	Booking record snapshot before update
room_map_snapshot	ARRAY› OBJECT‹	Room mapping snapshot before update
approved_by	INT	Admin who approved edit
approved_at	DATETIME	Timestamp of approval
logged_at	DATETIME	Log creation timestamp

5.2 CMS COLLECTIONS

content_docs

Field	Type	Description
_id	ObjectId / string	PK
type	string	announcement , banner , offer , testimonial , promotion
title	string	Title or headline
description	string	Body content
media	object	{ url: string, type: "image" }
status	string	used , unused , draft , published
metadata	object	Additional info (CTA, discount %)
order	int	Sorting priority
createdAt	datetime	Creation time
updatedAt	datetime	Last modified time
images	[object]	Optional list of images { url: string, caption: string }

Indexes:

{ type: 1, status: 1 ›

{ order: 1 ›

backup_data_collections

Purpose:

Central registry for all backup jobs — both manual and scheduled (daily, weekly, monthly, custom).

Tracks metadata, execution details, and validation state.

Field	Type	Description
_id	ObjectId / string	Primary key; used as backup reference in restore logs
snapshotName	string	Logical name (e.g., `snapshot_2025_10_08_weekly`)
initiatedBy	string	ID of the user/admin/system who triggered the backup
triggerType	string	`manual` or `scheduled`
scheduleType	string (nullable)	`daily`, `weekly`, `monthly`, `custom` (only for scheduled)
databaseType	string	`postgres` or `mongodb`
collectionsIncluded	[string]	Collections/tables included in this backup
storagePath	string	Backup file location (local/cloud)
sizeMB	number	Backup file size (MB)
checksum	string	Hash (SHA256/MD5) for integrity
status	string	`pending`, `in_progress`, `completed`, `failed`
timestamp	datetime	Backup start time
completedAt	datetime	Completion time
details	object	`{ durationSec, compression, retentionDays, verified }`

Indexes

```
{ snapshotName: 1 ›
{ timestamp: -1 ›
{ triggerType: 1 ›
{ status: 1 ›
```

Example

```
{
 "_id": "671f60b334a5e3b9c07df9e4",
 "snapshotName": "snapshot_2025_10_08_weekly",
 "initiatedBy": "system_scheduler",
 "triggerType": "scheduled",
 "scheduleType": "weekly",
 "databaseType": "postgres",
 "collectionsIncluded": ["bookings", "payments", "refunds"],
 "storagePath":    "s3://hbs-backups/weekly/postgres_2025_10_08.tar.gz",
 "sizeMB": 482.75,
 "checksum": "e0aaf12f45b87e4c1a23b5e9c7dd0a7a",
 "status": "completed",
 "timestamp": "2025-10-08T00:00:00Z",
 "completedAt": "2025-10-08T00:05:30Z",
 "details": {
  "durationSec": 330,
  "compression": "gzip",
  "retentionDays": 30,
  "verified": true}
}
```


restored_data_collections

Purpose:

Logs all restoration operations.

Each restore references the **backup's** `_id` — no separate restoreId required.

Captures when, who, and where the restore occurred, and post-restore validation.

Field	Type	Description
_id	ObjectId / string	Primary key for the restore record
backupRefId	ObjectId / string	References `_id` from `backup_data_collections`
restoredBy	string	User/admin/system who triggered the restore

Field	Type	Description
databaseType	string	`postgres` or `mongodb`
targetDatabase	string	Destination environment (e.g., `prod_clone`, `test_env`)
storagePath	string	Location of backup used for restore
collectionsRestored	[string]	List of restored entities
status	string	`initiated`, `in_progress`, `completed`, `failed`
timestamp	datetime	Restore start time
completedAt	datetime	Completion time
details	object	`{ checksumVerified, durationSec, restoreMode, validationPassed }`

Indexes

```
{ backupRefId: 1 ›
{ timestamp: -1 ›
{ status: 1 ›
```

Example


```
{
 "_id":   "671f618487e9a8a3d3f8b4aa",
 "backupRefId": "671f60b334a5e3b9c07df9e4",
 "restoredBy": "admin_105",
 "databaseType": "postgres",
 "targetDatabase":  "prod_clone",
 "storagePath":    "s3://hbs-backups/weekly/postgres_2025_10_08.tar.gz",
 "collectionsRestored": ["bookings", "payments", "refunds"],
 "status": "completed",
 "timestamp": "2025-10-09T03:00:00Z",
 "completedAt": "2025-10-09T03:04:30Z",
 "details": {
  "checksumVerified": true,
  "durationSec": 265,
  "restoreMode": "full",
  "validationPassed": true}
```

```
}
```

6. RELATIONSHIPS

```
--------------------------------------------
-- USER & AUTHENTICATION
--------------------------------------------

ALTER TABLE users
ADD CONSTRAINT fk_users_roles
FOREIGN KEY (role_id) REFERENCES roles_utility(role_id)
ON DELETE RESTRICT;

ALTER TABLE users
ADD CONSTRAINT fk_users_account_status
FOREIGN KEY (account_status_id) REFERENCES status_utility(status_id)
ON DELETE RESTRICT;

ALTER TABLE users
ADD CONSTRAINT fk_users_profile_image
FOREIGN KEY (profile_image_id) REFERENCES images(image_id)
ON DELETE SET NULL;

ALTER TABLE users
ADD CONSTRAINT fk_users_created_by
FOREIGN KEY (created_by) REFERENCES users(user_id)
ON DELETE SET NULL;

ALTER TABLE sessions
ADD CONSTRAINT fk_sessions_users
FOREIGN KEY (user_id) REFERENCES users(user_id)
ON DELETE CASCADE;

--------------------------------------------
```

```
-- ROOMS & AMENITIES
-------------------------------------------

ALTER TABLE rooms
ADD CONSTRAINT fk_rooms_room_types
FOREIGN KEY (room_type_id) REFERENCES room_types(room_type_id)
ON DELETE RESTRICT;

ALTER TABLE rooms
ADD CONSTRAINT fk_rooms_status
FOREIGN KEY (room_status_id) REFERENCES status_utility(status_id)
ON DELETE RESTRICT;

ALTER TABLE room_amenity_map
ADD CONSTRAINT fk_room_amenity_map_rooms
FOREIGN KEY (room_id) REFERENCES rooms(room_id)
ON DELETE CASCADE;

ALTER TABLE room_amenity_map
ADD CONSTRAINT fk_room_amenity_map_amenities
FOREIGN KEY (amenity_id) REFERENCES room_amenities(amenity_id)
ON DELETE CASCADE;

-------------------------------------------
-- BOOKINGS & EDIT MANAGEMENT
-------------------------------------------

ALTER TABLE bookings
ADD CONSTRAINT fk_bookings_users
FOREIGN KEY (user_id) REFERENCES users(user_id)
ON DELETE RESTRICT;

ALTER TABLE bookings
ADD CONSTRAINT fk_bookings_offers
FOREIGN KEY (offer_id) REFERENCES offers(offer_id)
ON DELETE SET NULL;
```

```
ALTER TABLE bookings
ADD CONSTRAINT fk_bookings_status
FOREIGN KEY (status_id) REFERENCES status_utility(status_id)
ON DELETE RESTRICT;

ALTER TABLE booking_room_map
ADD CONSTRAINT fk_booking_room_map_bookings
FOREIGN KEY (booking_id) REFERENCES bookings(booking_id)
ON DELETE CASCADE;

ALTER TABLE booking_room_map
ADD CONSTRAINT fk_booking_room_map_rooms
FOREIGN KEY (room_id) REFERENCES rooms(room_id)
ON DELETE CASCADE;

-------------------------------------------
-- BOOKING EDIT MANAGEMENT
-------------------------------------------

ALTER TABLE edit_bookings
ADD CONSTRAINT fk_edit_bookings_bookings
FOREIGN KEY (booking_id) REFERENCES bookings(booking_id)
ON DELETE CASCADE;

ALTER TABLE edit_bookings
ADD CONSTRAINT fk_edit_bookings_user
FOREIGN KEY (user_id) REFERENCES users(user_id)
ON DELETE RESTRICT;

ALTER TABLE edit_bookings
ADD CONSTRAINT fk_edit_bookings_status
FOREIGN KEY (status_id) REFERENCES status_utility(status_id)
ON DELETE RESTRICT;

ALTER TABLE edit_bookings
ADD CONSTRAINT fk_edit_bookings_requested_by
FOREIGN KEY (requested_by) REFERENCES users(user_id)
```

```
ON DELETE SET NULL;

ALTER TABLE edit_bookings
ADD CONSTRAINT fk_edit_bookings_reviewed_by
FOREIGN KEY (reviewed_by) REFERENCES users(user_id)
ON DELETE SET NULL;

--------------------------------------------
-- PAYMENTS & REFUNDS
--------------------------------------------

ALTER TABLE payments
ADD CONSTRAINT fk_payments_bookings
FOREIGN KEY (booking_id) REFERENCES bookings(booking_id)
ON DELETE RESTRICT;

ALTER TABLE payments
ADD CONSTRAINT fk_payments_users
FOREIGN KEY (user_id) REFERENCES users(user_id)
ON DELETE RESTRICT;

ALTER TABLE payments
ADD CONSTRAINT fk_payments_method
FOREIGN KEY (method_id) REFERENCES payment_method_utility(method_id)
ON DELETE RESTRICT;

ALTER TABLE refunds
ADD CONSTRAINT fk_refunds_bookings
FOREIGN KEY (booking_id) REFERENCES bookings(booking_id)
ON DELETE RESTRICT;

ALTER TABLE refunds
ADD CONSTRAINT fk_refunds_users
FOREIGN KEY (user_id) REFERENCES users(user_id)
ON DELETE RESTRICT;

ALTER TABLE refunds
```

```
ADD CONSTRAINT fk_refunds_status
FOREIGN KEY (status_id) REFERENCES status_utility(status_id)
ON DELETE RESTRICT;

ALTER TABLE refunds
ADD CONSTRAINT fk_refunds_method
FOREIGN KEY (transaction_method_id) REFERENCES payment_method_utility(m
ethod_id)
ON DELETE RESTRICT;

-------------------------------------------
-- ISSUES & CHAT
-------------------------------------------

ALTER TABLE issues
ADD CONSTRAINT fk_issues_bookings
FOREIGN KEY (booking_id) REFERENCES bookings(booking_id)
ON DELETE RESTRICT;

ALTER TABLE issues
ADD CONSTRAINT fk_issues_rooms
FOREIGN KEY (room_id) REFERENCES rooms(room_id)
ON DELETE SET NULL;

ALTER TABLE issues
ADD CONSTRAINT fk_issues_users
FOREIGN KEY (user_id) REFERENCES users(user_id)
ON DELETE RESTRICT;

ALTER TABLE issues
ADD CONSTRAINT fk_issues_resolved_by
FOREIGN KEY (resolved_by) REFERENCES users(user_id)
ON DELETE SET NULL;

ALTER TABLE issues
ADD CONSTRAINT fk_issues_status
FOREIGN KEY (status_id) REFERENCES status_utility(status_id)
```

```
ON DELETE RESTRICT;

ALTER TABLE issue_chat
ADD CONSTRAINT fk_issue_chat_issues
FOREIGN KEY (issue_id) REFERENCES issues(issue_id)
ON DELETE CASCADE;

ALTER TABLE issue_chat
ADD CONSTRAINT fk_issue_chat_users
FOREIGN KEY (sender_id) REFERENCES users(user_id)
ON DELETE RESTRICT;

-------------------------------------------
-- IMAGES MANAGEMENT
-------------------------------------------

ALTER TABLE images
ADD CONSTRAINT fk_images_uploaded_by
FOREIGN KEY (uploaded_by) REFERENCES users(user_id)
ON DELETE SET NULL;

-------------------------------------------
-- REVIEWS SYSTEM
-------------------------------------------

ALTER TABLE reviews
ADD CONSTRAINT fk_reviews_bookings
FOREIGN KEY (booking_id) REFERENCES bookings(booking_id)
ON DELETE RESTRICT;

ALTER TABLE reviews
ADD CONSTRAINT fk_reviews_users
FOREIGN KEY (user_id) REFERENCES users(user_id)
ON DELETE RESTRICT;

ALTER TABLE reviews
ADD CONSTRAINT fk_reviews_room_types
```

```
FOREIGN KEY (room_type_id) REFERENCES room_types(room_type_id)
ON DELETE RESTRICT;

ALTER TABLE reviews
ADD CONSTRAINT fk_reviews_admin
FOREIGN KEY (admin_id) REFERENCES users(user_id)
ON DELETE SET NULL;

-- ------------------------------------------
-- OFFERS MANAGEMENT
-- ------------------------------------------

ALTER TABLE offers
ADD CONSTRAINT fk_offers_created_by
FOREIGN KEY (created_by) REFERENCES users(user_id)
ON DELETE SET NULL;

ALTER TABLE offers
ADD CONSTRAINT fk_offers_updated_by
FOREIGN KEY (updated_by) REFERENCES users(user_id)
ON DELETE SET NULL;

ALTER TABLE offers
ADD CONSTRAINT fk_offers_primary_image
FOREIGN KEY (primary_image_id) REFERENCES images(image_id)
ON DELETE SET NULL;

-- ------------------------------------------
-- NOTIFICATIONS SYSTEM
-- ------------------------------------------

ALTER TABLE notifications
ADD CONSTRAINT fk_notifications_recipient_user
FOREIGN KEY (recipient_user_id) REFERENCES users(user_id)
ON DELETE CASCADE;

-- ------------------------------------------
```

```
-- WISHLIST SYSTEM
-------------------------------------------

ALTER TABLE wishlist
ADD CONSTRAINT fk_wishlist_users
FOREIGN KEY (user_id) REFERENCES users(user_id)
ON DELETE CASCADE;

ALTER TABLE wishlist
ADD CONSTRAINT fk_wishlist_room_types
FOREIGN KEY (room_type_id) REFERENCES room_types(room_type_id)
ON DELETE RESTRICT;

ALTER TABLE wishlist
ADD CONSTRAINT fk_wishlist_offers
FOREIGN KEY (offer_id) REFERENCES offers(offer_id)
ON DELETE SET NULL;

-------------------------------------------
-- AUDIT LOGS
-------------------------------------------

ALTER TABLE audit_log
ADD CONSTRAINT fk_audit_log_users
FOREIGN KEY (changed_by_user_id) REFERENCES users(user_id)
ON DELETE SET NULL;
```

7. VIEWS

Customer Booking Summary View

Aggregates customer bookings with associated room information in JSON format.

```
CREATE OR REPLACE VIEW vw_customer_bookings AS
SELECT
    b.booking_id,
    b.user_id,
```

```
    u.email AS customer_email,
    b.primary_customer_name,
    b.check_in_date,
    b.check_out_date,
    b.total_price,
    json_agg(
        json_build_object(
            'room_id', r.room_id,
            'room_no', r.room_no,
            'is_pre_edited_room', brm.is_pre_edited_room,
            'is_post_edited_room', brm.is_post_edited_room
        ) ORDER BY r.room_id
    ) AS assigned_rooms
FROM bookings b
JOIN users u ON b.user_id = u.user_id
LEFT JOIN booking_room_map brm ON b.booking_id = brm.booking_id
LEFT JOIN rooms r ON brm.room_id = r.room_id
WHERE b.is_deleted > FALSE
GROUP BY
    b.booking_id,
    b.user_id,
    u.email,
    b.primary_customer_name,
    b.check_in_date,
    b.check_out_date,
    b.total_price;
```

8. STORED PROCEDURES

sp_create_bookings()

Creates a new booking, maps rooms, and calculates the total amount.

```
CREATE OR REPLACE FUNCTION sp_create_bookings(...)
RETURNS INT AS $$
BEGIN
```

```
    INSERT INTO bookings (...);
    FOREACH v_room_id IN ARRAY p_room_ids LOOP
        INSERT INTO booking_room_map (...);
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

9. TRIGGERS

Booking Validation Trigger

Ensures valid check-in and check-out dates before inserting a booking record.

```
CREATE OR REPLACE FUNCTION trg_booking_before_insert_validate()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.check_out > > NEW.check_in THEN
        RAISE EXCEPTION 'Invalid booking dates';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

10. SECURITY CONSIDERATIONS

10.1 User Roles and Privileges

Role	Access Level	Capabilities
SuperAdmin	Full unrestricted system access	Manage admins, permissions, and configurations. Has global override authority (Postgres + MongoDB).
Admin	Operational CRUD across key entities	Manage users, rooms, offers, bookings, payments, issues, reviews, and notifications.

Role	Access Level	Capabilities
Customer	Restricted, self-scoped access	Manage own bookings, issues, reviews, and wishlists. Receives notifications and offers.
System (Internal Service)	Trusted automation agent	Executes audit triggers, syncs Postgres–MongoDB deltas, runs DRD, and performs backup verification.

Role Enforcement:

- Centralized in `roles_utility` and linked via `permission_role_map`.
- Application-level RBAC middleware validates JWT claims (`role_id` , `permissions`).
- Admin/API operations trigger `audit_log` (Postgres) and `api_logs` (MongoDB).
- Role-based stored procedures enforce privilege isolation.

10.2 Data Protection and Compliance

- **Credential Security:**
  - Passwords hashed with bcrypt (< 12 rounds).
  - Password resets and 2FA tokens expire in 15 min.
- **Network Security:**
  - Databases in isolated VPCs, accessible only via TLS 1.3.
  - IP whitelisting and mTLS for inter-service communication.
- **Encryption:**
  - Column-level encryption for `hashed_password` , `transaction_reference` .
  - Field-level encryption for sensitive MongoDB fields.
- **Audit & Forensics:**
  - `audit_log` (Postgres): immutable ledger table.
  - MongoDB: append-only `api_logs` , `security_logs` , `backup_restore_logs` .
  - Retention: 90 days with SHA-256 checksum verification.

10.3 Backup, Retention & Disaster Recovery

Database	Backup Strategy	Frequency	Retention	Notes
PostgreSQL	Base + WAL archiving	Full daily, WAL every 15 min	30 days	AES-256 encrypted, checksum verified
MongoDB	Cluster snapshots + Oplog replay	Daily	30 days	Point-in-time restore supported
Audit Logs	Cold archive to S3/Glacier	Weekly	90 days	Compressed append-only

- Backup verifications logged in `backup_restore_logs`.
- DRD replays missing WAL/oplogs automatically.
- Real-time monitoring through `system_health_logs`.

---

11. PERFORMANCE OPTIMIZATION

11.1 Query & Indexing Strategy

- Prepared statements enforced across ORM/API.
- Composite indexes:
  - `bookings`: `(user_id, status_id, check_in_date)`
  - `rooms`: `(room_type_id, room_status_id)`
  - `payments`: `(booking_id, status)`
  - `issues`: `(booking_id, status_id)`
  - `refunds`: `(booking_id, status_id)`
- Partial indexes on `WHERE is_deleted = FALSE`.
- Descending indexes on `created_at`, `updated_at`.
- Materialized views: `vw_customer_bookings`, `vw_refund_report`, `vw_issue_tracker`.

11.2 Scalability Architecture

- Table partitioning: `bookings`, `payments`, `audit_log` by year.

- Connection pooling: via pgBouncer.
- Caching: Redis for room listings, offers, availability.
- Async jobs: RabbitMQ/Kafka pipelines.
- Read replicas: analytics offload.

11.3 Data Aggregation & Analytics

- Nightly sync to `analytics_summary`.
- Real-time ETL: PostgreSQL * MongoDB * Data Lake.
- Weekly VACUUM & ANALYZE.
- Query profiling via `pg_stat_statements`.

12. DATA MIGRATION

12.1 Initial Data Load Sequence

Step	Table / Collection	Description
1	`users`	Core identity base
2	`roles_utility`, `permissions`, `permission_role_map`	Role and permission setup
3	`room_types`	Room categories
4	`rooms`	Room inventory
5	`room_amenities`, `room_amenity_map`	Amenities and mapping
6	`offers`	Promotional offers
7	`bookings`	Booking records
8	`booking_room_map`	Room allocations
9	`edit_bookings`	Edit workflows
10	`payments`, `refunds`	Financial records
11	`issues`, `issue_chat`	Complaints and communication
12	`reviews`	Feedback
13	`notifications`, `wishlist`, `images`	Engagement data

Step	Table / Collection	Description
14	`audit_log`, `backup_restore_logs`	System audits
15	`booking_logs`, `api_logs`, `security_logs`, `content_docs`	MongoDB collections

12.2 Data Consistency Rules

- Validate FKs before dependent inserts.
- Skip rows where `is_deleted = TRUE`.
- ENUMs preloaded before dependent data.
- MongoDB records linked post-import via relational IDs.
- Referential integrity verified post-load.

13. APPENDIX

13.1 Sample Data: Room Types

```
INSERT INTO room_types (type_name, max_adult_count, max_child_count, price
_per_night, description)
VALUES
('Standard', 2, 0, 80.00, 'Essential comfort with all basics'),
('Deluxe', 3, 1, 120.00, 'Spacious, premium-finish room'),
('Suite', 4, 2, 250.00, 'Luxury suite with lounge & exclusive service');
```

13.2 Sample Data: Room Amenities

```
INSERT INTO room_amenities (amenity_name, description)
VALUES
('WiFi', 'High-speed wireless internet'),
('AC', 'Air-conditioned environment'),
('TV', 'Smart LED television'),
```

```
('Breakfast', 'Complimentary breakfast service');
```

13.3 Sample Admin Permissions

```
INSERT INTO permissions (resource, permission_type)
VALUES
('MANAGE_BOOKINGS', 'WRITE'),
('MANAGE_ROOMS', 'WRITE'),
('VIEW_REPORTS', 'READ'),
('HANDLE_ISSUES', 'MANAGE'),
('APPROVE_EDITS', 'MANAGE');
```

13.4 Sample MongoDB Collections

booking_logs

```
{
 "booking_id": 101,
 "edit_id": 15,
 "edit_type": "PRE",
 "booking_snapshot": {
  "primary_customer_name": "John Doe",
  "check_in_date": "2025-10-01",
  "check_out_date":  "2025-10-05",
  "total_price": 480.00
 },
 "room_map_snapshot": [
  { "room_id": 12, "is_pre_edited_room": false }
 ],
 "approved_by": 2,
 "approved_at": "2025-09-30T14:22:00Z",
 "logged_at": "2025-09-30T14:23:00Z"
```

```
}
```

api_logs

```
{
  "endpoint": "/api/bookings/create",
  "method": "POST",
  "user_id": 302,
  "role": "CUSTOMER",
  "response_code": 201,
  "status": "success",
  "latency_ms": 120,
  "timestamp":    "2025-10-08T10:45:00Z"
}
```

content_docs

```
{
  "type": "banner",
  "title": "Winter Fest 2025 Deals!",
  "description": "Save up to 25% on Suites & Deluxe Rooms this festive season.",
  "status": "published",
  "metadata": { "discount_percent": 25, "valid_till": "2025-12-31" },
  "createdAt": "2025-10-01T00:00:00Z"
}
```

security_logs

```
{
  "action": "ADMIN_LOGIN",
  "user_id": 2,
  "ip_address":  "192.168.1.14",
  "status": "SUCCESS",
```

```
  "timestamp":    "2025-10-08T08:15:00Z"
}
```

14. DOCUMENT VERSION HISTORY

VERSION	DATE	AUTHOR	DESCRIPTION
1.0	10/10/2025	Aswinnath TE	Initial draft