**Last Resort Hotel – Database Project Analysis Report**

Group Name: DataNotBasing

Group Members: Cicci, Lein, Skye, Sue

Date: October 14, 2025

### 1. Introduction

This project aims to design a relational database for Last Resort Hotel (LRH). The database will store information about customers, rooms, reservations, employees, and payments. The goal is to provide management with insights through queries and a simple web interface, including occupancy rates, revenue trends, and customer behavior.

### 2. Business Analysis

Last Resort Hotels has rapidly expanded, creating large complexes with multiple buildings, floors, wings, and room types, including sleeping rooms, suites, and meeting rooms. Customers make reservations through hotel staff, specifying preferences such as bed type, number of guests, smoking or nonsmoking, and room location.

Staff must manage room assignments dynamically, as rooms may be temporarily unavailable due to renovations, repairs, or cleaning. The system can track room relationships, status, and availability to ensure assignments meet guest preferences. The completion of cleaning and repair work is recorded when personnel use their magnetic cards to mark a room as available.

The hotel manages complex billing scenarios. Each reservation is linked to a responsible party, which may be an individual guest, a host organization, or a combination. Advance deposits may be required based on customer qualifications, such as prior history with the hotel. The hotel also supports different ways to split the bill based on the customer's idea. All charges are recorded promptly to ensure up-to-date bills are available.

Events are a core function, requiring the use of one or more rooms and having a designated host. The system tracks event duration, expected attendance, and affiliated guests. Sleeping rooms for event attendees are allocated based on their stated requirements and the hotel's room assignment policies.

Finally, the hotel relies on real-time information. Guests use PIN-based access cards, which, when used in readers throughout the complex, allow the system to track their location for service and security. Staff can access the current room status and guest information to respond to needs. The system also supports guest messaging so that staff can know their needs immediately.

**Due to incomplete or ambiguous information in the business description, several assumptions were made for database design:**

## **Location & identification**

* **wingName scope** → Unique within a **building**.
* **roomNumber scope** → Unique within a **wing**.
* **floorNumber scope** → Unique within a **wing**.
* **room.floorId + room.wingId** → Both stored; a constraint enforces room.wingId = floor.wingId.
* **Smoking default/override** → floor.isSmokingFloor is the default; room.isSmokingRoom overrides when not null.
* **currentStatus** → Enum: {available, cleaning, ooo, renovate, outOfService}. This single field blocks assignments; no separate block table.

## **Allocation policy**

* **Assignment order** → Filter by requirements, then sort by wingName (A→Z), then floorNumber (low→high), then roomNumber (low→high). Ties break by roomId.
* **Exact room vs request** → reservation holds preferences; room\_assignment is created only when a concrete room is chosen.

## **Room functions & beds**

* **Meeting→sleeping “in a pinch”** → Allowed only if meeting\_spec.hasToilet=true. Beds must be non-permanent (room\_has\_bed.isPermanent=false with isRollaway=true or isWallBed=true).
* **Permanent beds** → If any room\_has\_bed.isPermanent=true, that room **cannot** be assigned as MEETING via room\_has\_function.
* **suite\_spec** → A suite is one room with suite\_spec.hasAdjacentMeetingArea=true and a single corridor door; it is **not** two rooms.

## **Adjacency & divisibility**

* **room\_adjacency** → Two rooms are adjacent iff a door exists between them; store once with ordered pair (roomId1 < roomId2).
* **Adjacency limits** → A sleeping room may be adjacent to **at most one** meeting room; a meeting room may be adjacent to **up to two** sleeping rooms.
* **divisible\_oom / divided\_subroom** → Parent and any subroomId are **mutually exclusive** in room\_assignment time windows; subroomId ≠ parentRoomId and is unique across the table.  
  Here are the explicit **assumptions** I used to make your ERD executable. Format: area → assumption. Names match your attributes.

## **Capacities & ratings**

* **sleeping\_spec.sleepingCapacity** → Adults only, integer count.
* **meeting\_spec.seatingCapacity** → “Seated around tables” capacity; isOutdoor=true covers courtyard/patio treated as meeting rooms.

## **Time model (stays and slots)**

* **Stay window** → Sleeping rooms bill per day 16:00 → next-day 12:00 local time.
* **Grace rule** → Early/late ≤ 2h does not add a day; beyond that a surcharge is posted as a charge line.
* **useSlot catalog** → Fixed values: {breakfast, morning, lunch, afternoon, supper, evening, night}. Boundaries assumed:  
   breakfast 06:00–09:00, morning 09:00–12:00, lunch 12:00–14:00, afternoon 14:00–17:00, supper 17:00–19:00, evening 19:00–22:00, night 22:00–24:00.
* **Meeting assignment** → Each meeting\_slot\_assign row represents one slot for one assignmentId; no duplicates per assignment.

## **Pricing & discounts**

* **Half-rate rule** → service\_type drives pricing; non-eating slots charge 0.5× the eating rate.
* **Free non-eating per paid eating** → One free non-eating event\_facility\_use per paid eating slot; implemented by inserting a row with priceOverride=0 (or posting a $0 charge).
* **Event-linked waivers** → Discount logic is external; event\_facility\_use.priceOverride captures overrides when applied.

## **Parties, billing, splits**

* **Responsible party** → Every reservation and event must have a billable party reachable via billedPartyId or later via billing\_account.
* **Split billing scope** → Splits apply **per stay** in stay\_account\_split.percentage; percentages must sum to 100 for a given stayId.
* **Charge anchoring** → Each charge must reference **at least one** of roomId, stayId, or eventId. amount > 0.

## **Access cards & privacy**

* **reader attachment** → A reader attaches to either roomId **or** facilityCode (mutually exclusive).
* **direction field** → Stored in both reader.direction and access\_log.direction; a constraint/trigger enforces equality.
* **isActive** → Deactivating a card immediately prevents further access logs; historical logs remain.

## **Reservations & conflicts**

* **Advance window** → Accept reservations up to 2 years before startDate; major events are allowed by staff policy but use the same tables.
* **No overbooking** → The system does not overbook; conflicts are prevented at room\_assignment by disallowing overlapping (fromDateTime,toDateTime) per roomId.

## **Data types & codes**

* **IDs** → All …Id are surrogate keys (integers/UUIDs). Business codes like wingName, roomNumber, fixtureName, functionCode, serviceCode are natural keys within their parent scopes.
* **Names & phones** → person.phone and email are optional; at least one contact field is recommended but not enforced in schema.

Our database is designed to track room availability, room adjacency, events, reservations, and all charges associated with guests or hosts, ensuring that staff can access accurate and current information for managing hotel operations efficiently.

### 3. Allocation

Phase 1: ERD

Ver. 0: 10/5, a draft by Skye and Cicci

Ver. 1: 10/09, specific room part, by Cicci and Sue

Ver. 2: 10/13, Segmentation and more details, by Skye

Ver. 3: 10/14, formatted, and with crow’s foot notation, by Lein

**Milestone 1 Work Allocation by 10/16**

**Member A — Skye:** Created and managed the GitHub repository and Google shared folder; drafted ERD ver. 0; updated to ver. 2; assisted with final review.

**Member B — Cicci:** Developed ERD ver. 1; wrote the analysis report

**Member C — Lein:** Finalized ERD formatting and organization to ver.3 by adding the crow’s foot notation.

**Member D — Sue:** Developed ERD ver. 1; wrote the analysis report