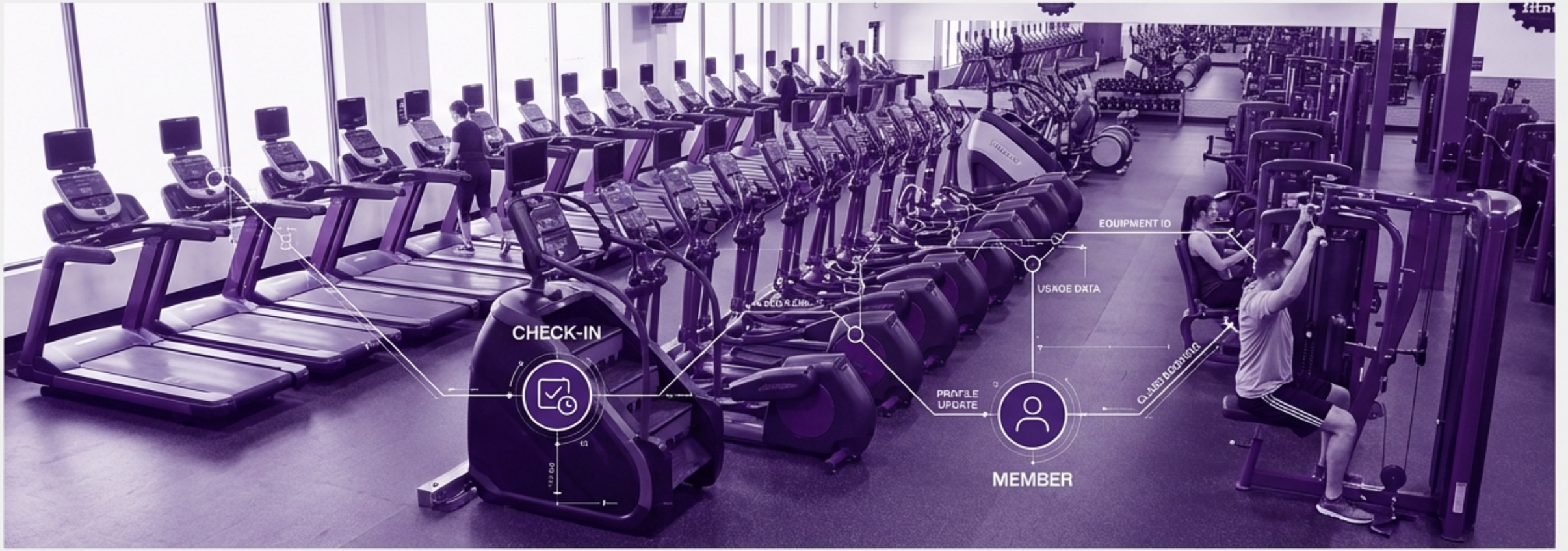


DATABASE DESIGN IN THE REAL WORLD

Architecting the Member Management System for a Planet Fitness Franchise

CASE STUDY: HARRISBURG FRANCHISE | SCENARIO: OPERATIONAL TURNAROUND

NotebookLM

The Crisis at the Harrisburg Franchise

Jen Martino, the new Regional Operations Manager, inherited an operational mess. The current “system” is a fragile mix of disconnected spreadsheets and paper logs.

Ghost Members

Who hasn't visited in 90 days? Impossible to answer. No engagement tracking means revenue leakage.

Guest Abuse

One non-member visited 11 times last month. Staff cannot cross-reference paper logs to enforce the 1-visit limit.

Blind Spots

Classic members ($15/mo) roaming to other gyms against policy. Front desk lacks data to validate "Home Gym" status.

Staffing Guesswork

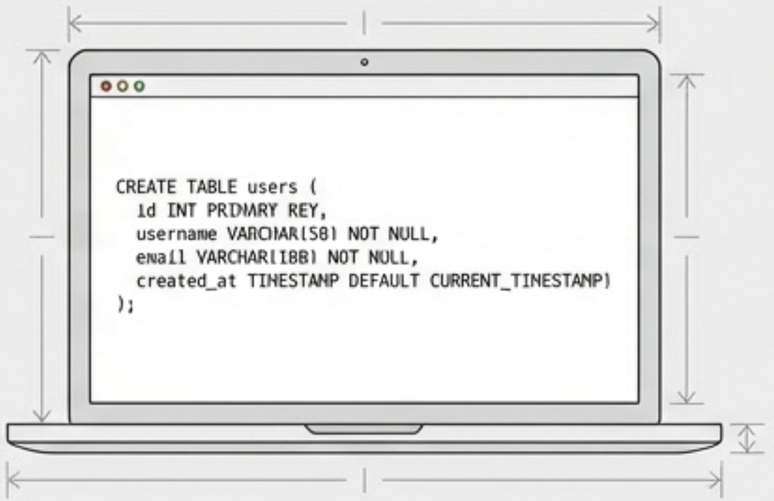
Scheduling based on “vibes” not volume. No check-in timestamp data to prove overstaffing.

NotebookLM

Stop. Don’t Open SQL Yet.

A database designer is a translator. You cannot translate business needs into data structures if you don’t speak the language of the business first.

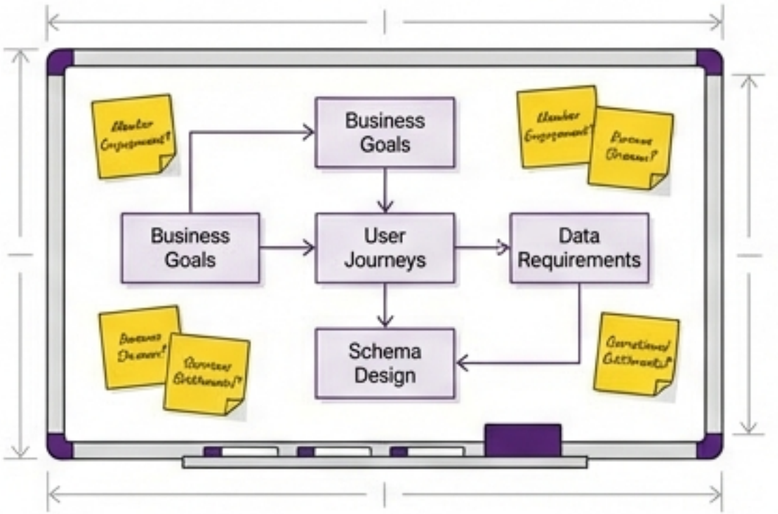
The Rookie Mindset



- Starts with Tables
- Focuses on Data Types

→ Result: Technically functional, Operationally broken.

The Pro Mindset



- Starts with Context
- Focuses on Business Rules

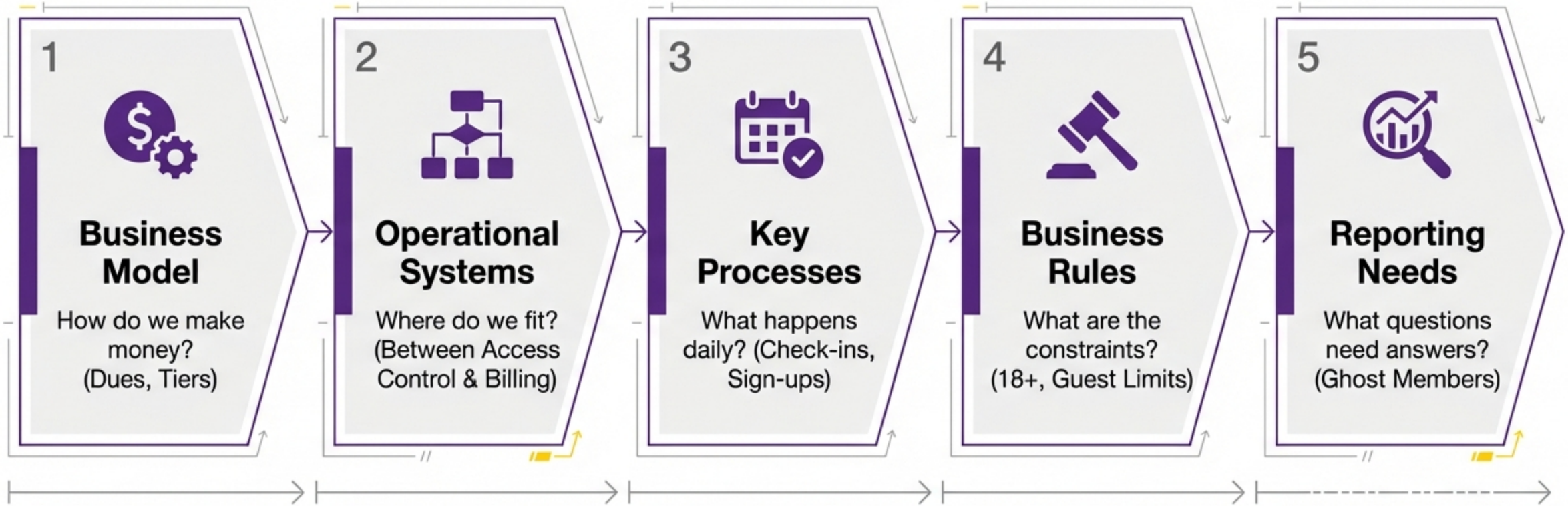
→ Result: Solves the human problem.

Insight: You don’t need to run the business, but you must understand how it operates.

NotebookLM

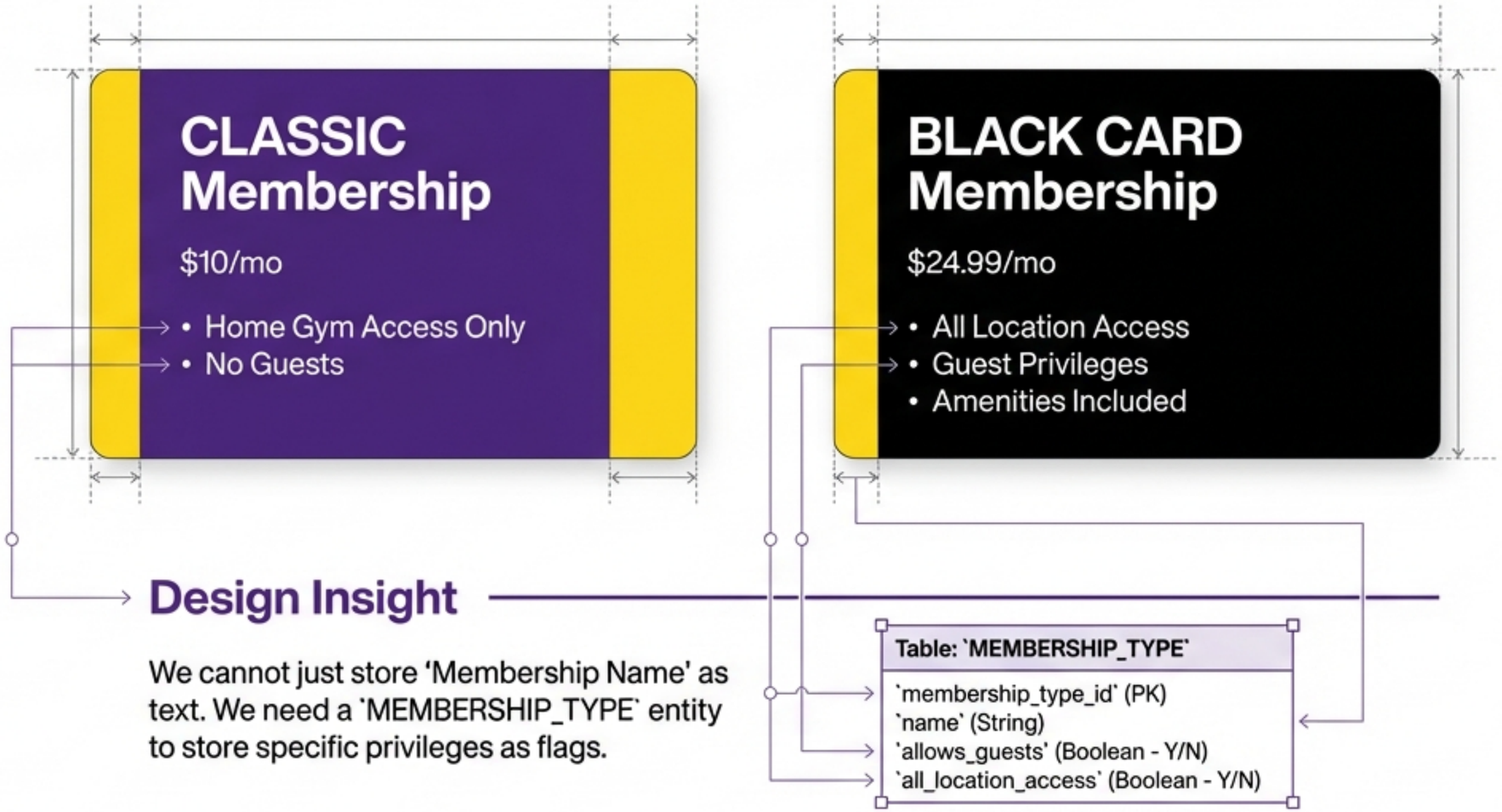
The Framework for Analysis

Before defining entities, we analyze these five areas of the Planet Fitness ecosystem:



NotebookLM

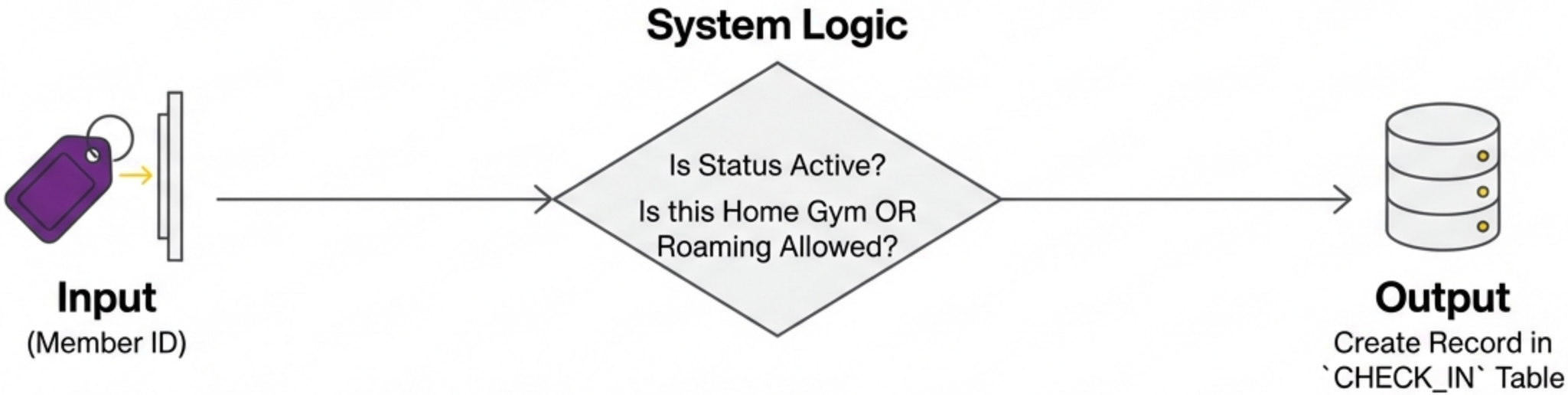
Decoding the Business Model



NotebookLM

The Heart of the System: The Check-In

The check-in is a transactional event that answers three questions: Who? Where? When?



Data Implication

This requires a transactional table.
It serves as the sole source of truth for ‘Ghost Member’ and ‘Peak Hours’ reports.

CHECK_IN			
🔑	PK	`check_in_id` (PK)	
🔑	FK	`member_id` (FK)	→ Who?
🔑	FK	`location_id` (FK)	→ Where?
		`check_in_timestamp`	→ When?

NotebookLM

Translating Reality into Relational Tables

We identify the core subjects (Entities) based on our analysis.

PEOPLE

Real World

Member Staff Guest

Database Entity

- MEMBER
- STAFF
- GUEST

PLACES

Real World

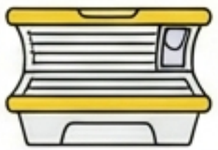
Gym Building

Database Entity

- LOCATION

THINGS

Real World



Tanning Bed

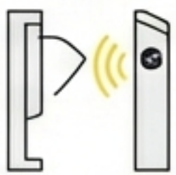
Membership Card

Database Entity

- AMENITY
- MEMBERSHIP_TYPE

EVENTS

Real World



Turnstile Beep

Guest Clipboard

Database Entity

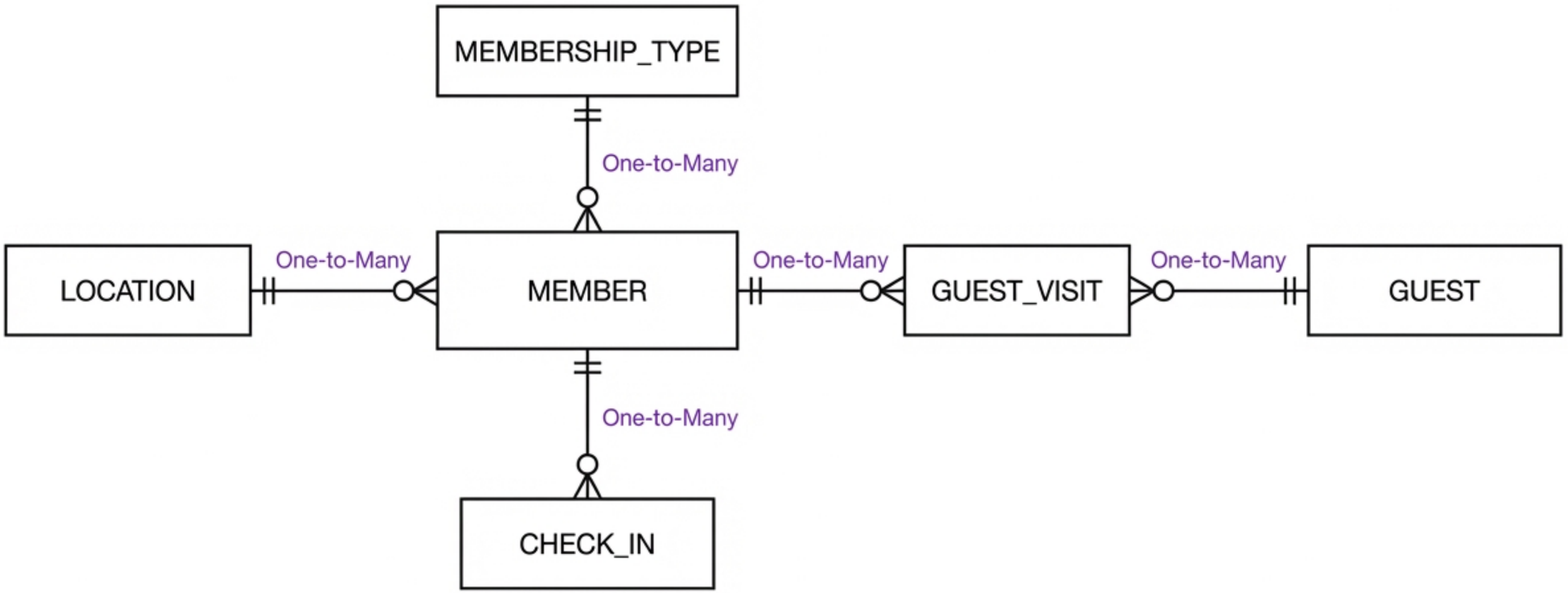
- CHECK_IN
- GUEST_VISIT

Crucial Design Choice: separating Identity (`GUEST`) from Activity (`GUEST_VISIT`).

NotebookLM

The Blueprint: High-Level Entity Relationship Diagram

A structural overview of the core entities and their relationships using Crow's Foot Notation.

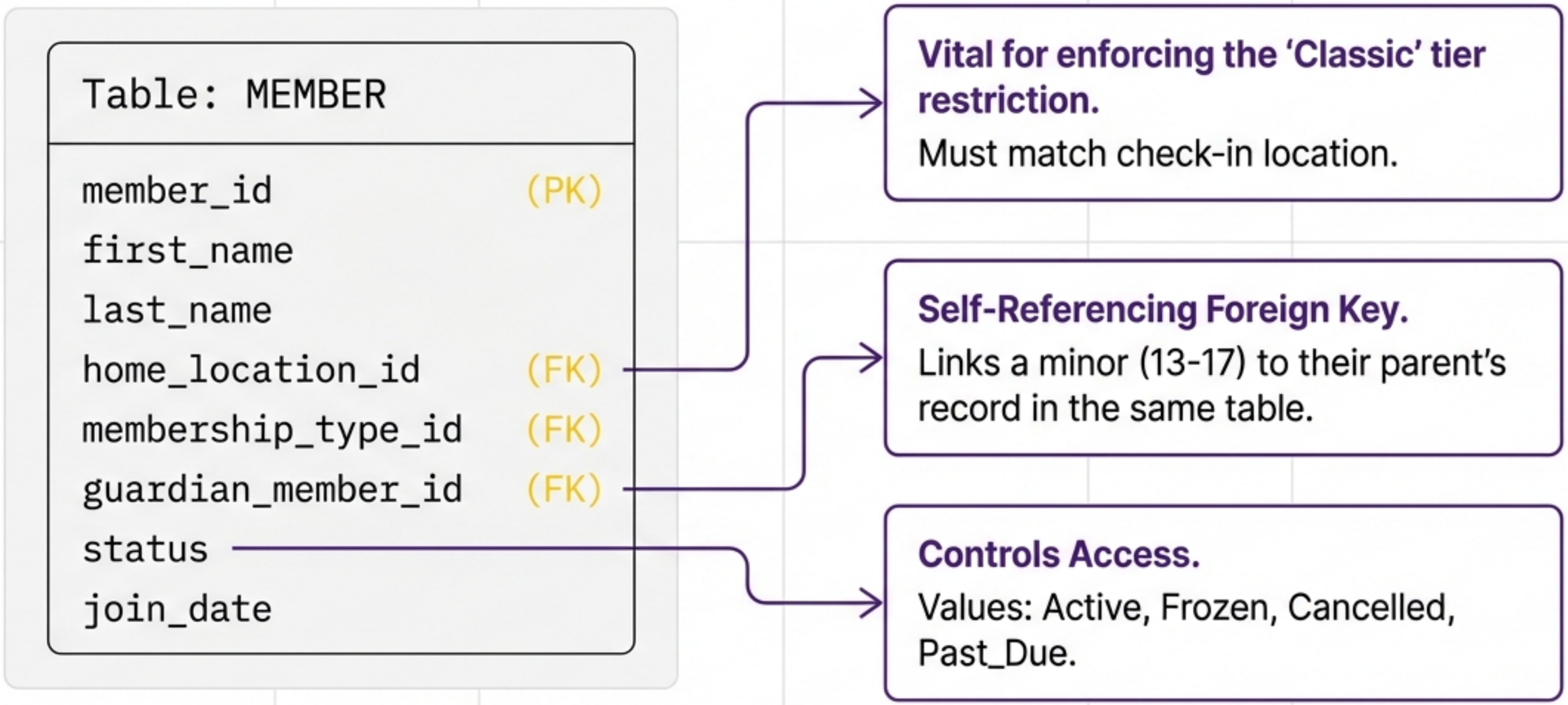


This structure creates a central record for every member while tracking their movement and guest activity independently.

NotebookLM

Design Deep Dive: The MEMBER Entity ⚙️

A detailed structural breakdown of the core `MEMBER` table specification and its key constraints.



NotebookLM

Solving the ‘Guest Abuse’ Problem

Business Rule: Guests can visit only once per location per month.

The Data Model

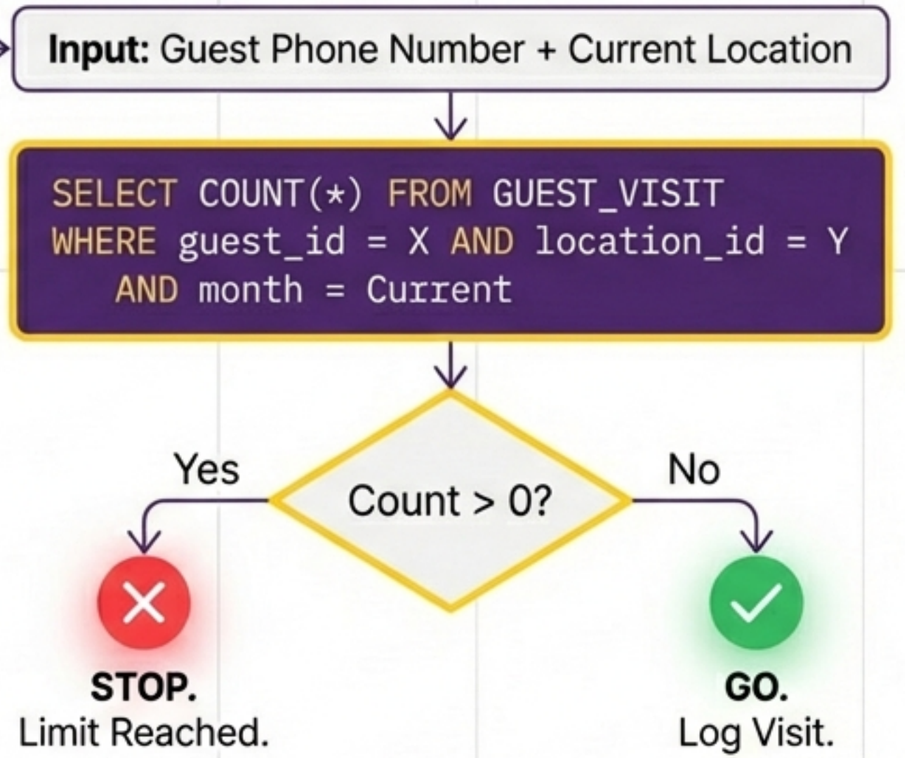
Table: GUEST

guest_id	(PK)
name	
phone_number (Unique ID)	

Table: GUEST_VISIT

visit_id	(PK)
guest_id	(FK)
location_id	(FK)
timestamp	
month_of_visit	

The Logic Gate

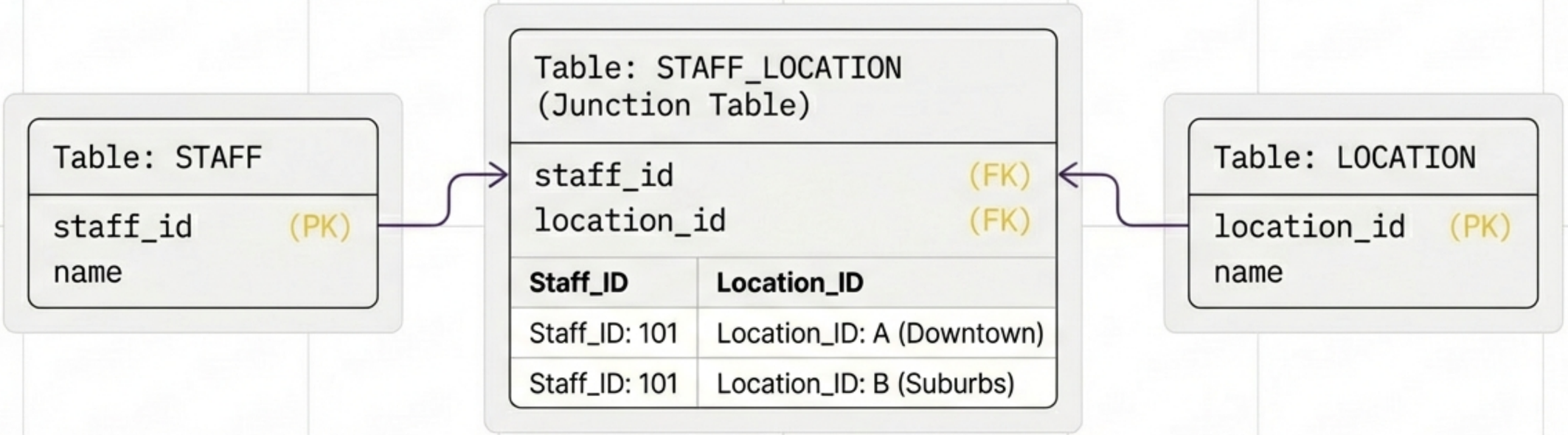


By decoupling identity from activity, we create a queryable history that enforces the limit mathematically.

NotebookLM

Handling Complexity: Many-to-Many Relationships

Scenario: Staff work at multiple locations. We cannot store a list of locations in the Staff table.



The Junction Table resolves the Many-to-Many relationship by storing pairs of IDs. This allows one staff member to be linked to infinite locations without duplicating staff data.

NotebookLM

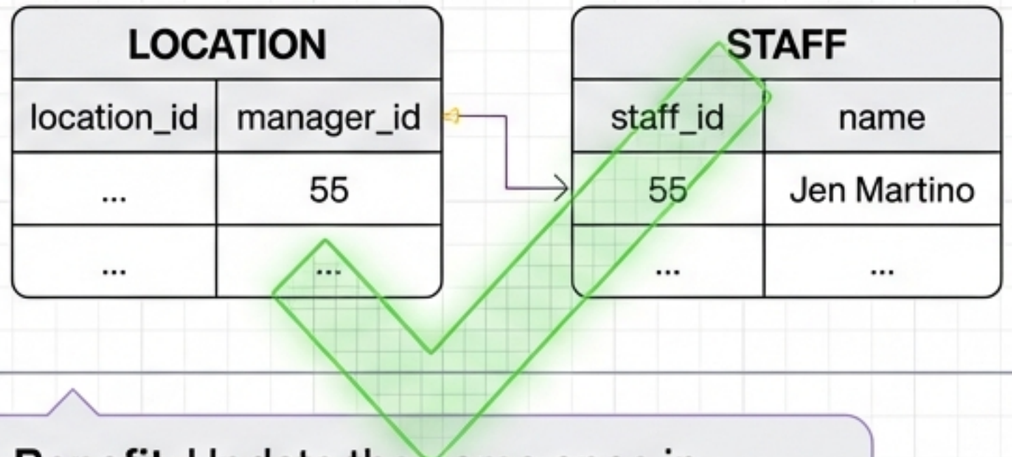
The Principle of Normalization

Single Source of Truth. We design to eliminate redundant data that causes errors.

The Wrong Way

Risk: If Jen gets married or changes her name, we must update this text string in every single table she appears in.

The Right Way



Benefit: Update the name once in the Staff table, and it automatically reflects everywhere.

"Duplicate information is bad; it wastes space and increases errors." — Microsoft Support

NotebookLM

The Return: Answering Jen’s Questions

Mapping the design back to the original operational crisis.

Problem

Who are the Ghost Members?

```
SELECT *
FROM Member
LEFT JOIN Check_In...
WHERE check_in IS NULL
```

Result

List of members with 0 visits in 90 days.

Problem

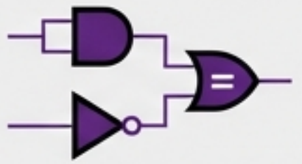
Is the Downtown gym overstaffed?

Result

Hourly check-in volume report generated from timestamp data.

Problem

Who are the Black Card Roamers?



Result

Comparison of `home_location_id` vs `check_in.location_id` reveals usage patterns.

NotebookLM

Designing for Tomorrow

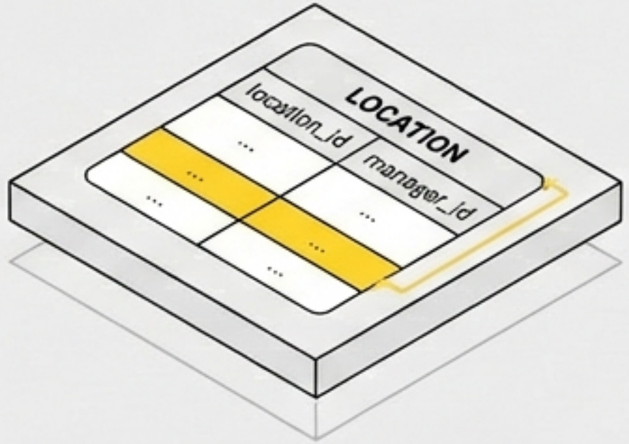
A robust relational design accommodates business changes without breaking the code.

Price Change

Prices increase?
Update `monthly_rate` in `MEMBERSHIP_TYPE`.
No code change needed.

New Tier

Adding 'Platinum' tier?
Insert 1 row into `MEMBERSHIP_TYPE`. System adapts automatically.



Expansion

Opening 4th Location?
Insert 1 row into `LOCATION`.
All logic scales instantly.

Normalization = Scalability.

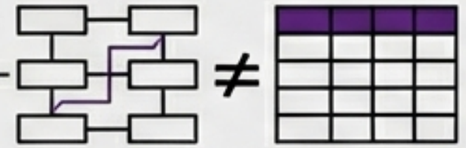
NotebookLM

Principles of World-Class Design

Context is King

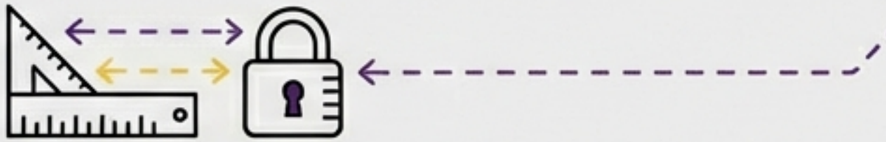
Understand the business model before defining the data structures.

Entities != Spreadsheets



Break data into logical, normalized subjects (People, Places, Events).

Rules become Constraints



Business policies (like Age 18+) dictate table structure (Guardian FK).

Design for Questions

Ensure your structure answers the reporting needs of the user.

NotebookLM