The Physical Database Design

Version 1.4

EECS 447 Project

The Physical Database Design

EECS 447

Version 1.4

Revision History

Date	Version	Description	Authors
04/05/25	1.0	Document creation, role assignment, and initial division of document sections.	Fatima Avila, Siddh Bharucha, Bhavik Goplani, Vy Luu, Suhaan Syed, Alexis Vielma
04/09/25	1.2	Brainstorming, Supabase Creation	Fatima Avila, Siddh Bharucha, Bhavik Goplani, Vy Luu, Suhaan Syed, Alexis Vielma
04/18/25	1.3	Document finalization and database creation and sql coding.	Fatima Avila, Siddh Bharucha, Bhavik Goplani, Vy Luu, Suhaan Syed, Alexis Vielma
04/26/2025	1.4	Initial draft and database output verification	Fatima Avila, Siddh Bharucha, Bhavik Goplani, Vy Luu, Suhaan Syed, Alexis Vielma

Database Requirements Specifications

1. Introduction

Project Overview

Tech Titans is building a robust **Library Management System (LMS)** that efficiently organizes and manages physical and digital library content. The system simplifies core library operations such as borrowing, returning, fines processing, and reservation notifications. By leveraging a relational database, LMS ensures reliable data management for members and staff alike.

Scope

Building on our approved logical schema, the physical database will:

- maintain core entities—members, membership types, items (books, magazines, digital media), staff, transactions, reservations, payments, and notifications—exactly as mapped in the logical model;
- enforce business rules such as borrow-limit caps, automatic fine computation, and real-time item availability;
- support operational reporting (overdue items, circulation trends, revenue by membership type) and ad-hoc analytics demanded in the project brief.

Included: all data structures and functionality listed above, plus DDL-level integrity (PK/FK, domain and check constraints) and triggers for:

- Auto-updating: availability_status
- late-fee calculation,
- reservation/overdue notifications.

Excluded: room-booking, events, and self-service member profile edits (postponed beyond MVP). These remain out of scope to keep the schema focused and grading-ready.

Glossary:

- LMS (Library Management System): A software system designed to manage library resources, including books, digital media, memberships, and borrowing transactions.
- ISBN (International Standard Book Number): A unique numeric identifier assigned to books for classification and tracking.
- ISSN (International Standard Serial Number): A unique identifier for periodicals, such as magazines and journals.
- MariaDB: The specific DBMS selected for this project, compatible with MySQL, used to store and manage library records.
- Primary Key (PK): A unique identifier assigned to each record in a database table to ensure data integrity.
- Foreign Key (FK): A database field that links one table to another, maintaining relationships between entities.
- Authentication & Authorization: Security measures that control system access, ensuring that only authorized users can modify records or access sensitive data.
- Backup & Recovery: A strategy for storing copies of data to protect against data loss due to system failures or security breaches.
- Relational Schema: A logical structure that defines tables, attributes, keys, and constraints in a relational database.
- Functional Dependency (FD): A relationship where one attribute uniquely determines another.
- Normalization: The process of organizing data to reduce redundancy
- DDL (Data Definition Language): SQL commands that create or alter schema objects.
- Seed / Fixture: A repeatable script that bulk-inserts representative data for testing.
- Migration: A version-controlled DDL change, generated via the Supabase CLI.
- Supabase Project: A managed PostgreSQL instance with RESTful and realtime layers, Git-integrated migrations, and role-based security.

2. Platform - Supabase

At Tech Titans, we selected **Supabase (PostgreSQL)** because it gives us the best balance of productivity and rigor for a course-scale project. Four of our six developers have used Supabase in the past before, so we can move straight from the logical schema to DDL without a steep learning curve. Its GitHub-native migration workflow means every change to tables, triggers, or seed data is version-controlled and reviewed—exactly what the rubric expects for script traceability. Supabase exposes a full-fledged Postgres instance, so we still benefit from standard SQL, row-level security, and trigger support, yet the managed dashboard, REST/GraphQL APIs, and generous free tier eliminate the ops burden of self-hosting MariaDB on the EECS cycle servers. In short, Supabase lets us concentrate on enforcing library business rules and building automated tests rather than babysitting infrastructure, while still keeping our work portable to any vanilla Postgres environment later on.

Criterion	Supabase (PostgreSQL-as-a-Service)	Rationale
Team familiarity	4 / 6 members have prior Supabase + SQL experience	Minimizes onboarding time
GitHub integration	Native migration tracking → pull-request review	Matches course requirement for script traceability
Cost & scale	Generous free tier; sufficient for ~100 MB data & <1 M row writes	Fits class project size
Features used	Row-level security, realtime triggers, Swagger REST auto-docs	Mirrors modern production stacks
Limitations	Vendor lock-in; requires internet access for local tests	Acceptable for academic scope

3. Database Creation

3.1 We initialized a fresh Supabase Postgres project and pushed two migration files:

Migration	Purpose
20250427061226_init_schema.sql	Creates all enum types, tables, primary/foreign keys, checks, and the three business-rule triggers (availability, borrow-limit, fine-calc).
20250427061611_add_rls_and_comments.sql	Adds row-level-security stubs (disabled for grading) and column comments for automatic API docs.

Each migration is committed under `supabase/migrations/` and automatically applied with

`supabase db deploy` for the cloud option

```
/*=========================================================================
 Tech Titans  --  Library Management System
 Physical Schema  (Supabase/PostgreSQL 15)
 Author(s): Suhaan Syed, Siddh Bharucha, Bhavik Goplani, Alexis Vielma, Vy
Luu, Fatima Avila
 -------------------------------------------------------------------------
   • Creates all enum types
   • Creates tables with keys, checks, and comments
   • Adds triggers for
         - setting item availability
         - enforcing member borrow-limit
         - calculating late fees
 ========================================================================*/

-- ──────────────────────────────── ENUM TYPES
────────────────────────────────
CREATE TYPE membership_status_t      AS ENUM
('Active','Suspended','Overdue');
CREATE TYPE membership_type_name_t   AS ENUM ('Regular','Student','Senior
Citizen');
CREATE TYPE item_type_t              AS ENUM ('Book','Digital
Media','Magazine');
CREATE TYPE avail_status_t           AS ENUM ('Available','On
Loan','Reserved');
CREATE TYPE media_format_t           AS ENUM
('eBook','Audiobook','Video','Other');
CREATE TYPE staff_role_t             AS ENUM ('Librarian','Administrator');
CREATE TYPE notification_type_t      AS ENUM ('Reservation','Due Date
Alert','Overdue Alert');

-- ──────────────────────────────── LOOKUP TABLES
────────────────────────────
CREATE TABLE public.membership_types (
    type_id           BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    type_name         membership_type_name_t NOT NULL UNIQUE,
    max_borrow_limit  INT    NOT NULL CHECK (max_borrow_limit > 0),
    fine_rate         NUMERIC(5,2) NOT NULL CHECK (fine_rate >= 0)
);
```

```sql
-- ──────────────────────────────── CORE ENTITIES
────────────────────────────

CREATE TABLE public.members (
    member_id          BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    name               TEXT NOT NULL,
    contact_info       TEXT,
    membership_type_id BIGINT
        REFERENCES membership_types(type_id) ON UPDATE CASCADE,
    account_status     membership_status_t NOT NULL DEFAULT 'Active'
);

CREATE TABLE public.library_items (
    item_id            BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    title              TEXT NOT NULL,
    item_type          item_type_t NOT NULL,
    availability_status avail_status_t NOT NULL DEFAULT 'Available'
);

-- ────────────── "INHERITANCE" SUB-TYPE TABLES  ──────────────
CREATE TABLE public.books (
    book_id            BIGINT PRIMARY KEY
        REFERENCES library_items(item_id) ON DELETE CASCADE,
    isbn               TEXT UNIQUE NOT NULL,
    author             TEXT NOT NULL,
    genre              TEXT,
    publication_year   SMALLINT
);

CREATE TABLE public.digital_media (
    media_id           BIGINT PRIMARY KEY
        REFERENCES library_items(item_id) ON DELETE CASCADE,
    creator            TEXT NOT NULL,
    format             media_format_t NOT NULL
);

CREATE TABLE public.magazines (
    magazine_id        BIGINT PRIMARY KEY
        REFERENCES library_items(item_id) ON DELETE CASCADE,
    issue_number       INT  NOT NULL UNIQUE,
    publication_date  DATE NOT NULL
);

CREATE TABLE public.staff (
    staff_id          BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    name              TEXT NOT NULL,
    role              staff_role_t NOT NULL,
    contact_info      TEXT NOT NULL
);

-- ──────────────────────────────── TRANSACTIONS
──────────────────────────────

CREATE TABLE public.borrowing_transactions (
    borrow_id         BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    member_id         BIGINT NOT NULL
        REFERENCES members(member_id) ON UPDATE CASCADE ON DELETE RESTRICT,
    item_id           BIGINT NOT NULL
        REFERENCES library_items(item_id) ON UPDATE CASCADE ON DELETE
RESTRICT,
    staff_id          BIGINT
        REFERENCES staff(staff_id) ON UPDATE CASCADE ON DELETE SET NULL,
    borrow_date       DATE NOT NULL,
    due_date          DATE NOT NULL,
    return_date       DATE,
    fine_incurred     NUMERIC(5,2) CHECK (fine_incurred IS NULL OR
fine_incurred >= 0)
);

CREATE TABLE public.reservations (
    reservation_id    BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    member_id         BIGINT NOT NULL
        REFERENCES members(member_id),
    item_id           BIGINT NOT NULL
        REFERENCES library_items(item_id),
    reservation_date  DATE NOT NULL,
    expiry_date       DATE NOT NULL
        CHECK (expiry_date >= reservation_date)
);
```

```
CREATE TABLE public.payments (
    payment_id        BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    member_id         BIGINT NOT NULL
        REFERENCES members(member_id),
    amount_paid       NUMERIC(7,2) NOT NULL CHECK (amount_paid >= 0),
    payment_date      DATE NOT NULL
);

CREATE TABLE public.notifications (
    notification_id   BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    member_id         BIGINT NOT NULL
        REFERENCES members(member_id),
    notification_date TIMESTAMPTZ NOT NULL DEFAULT now(),
    notification_type notification_type_t NOT NULL
);
```

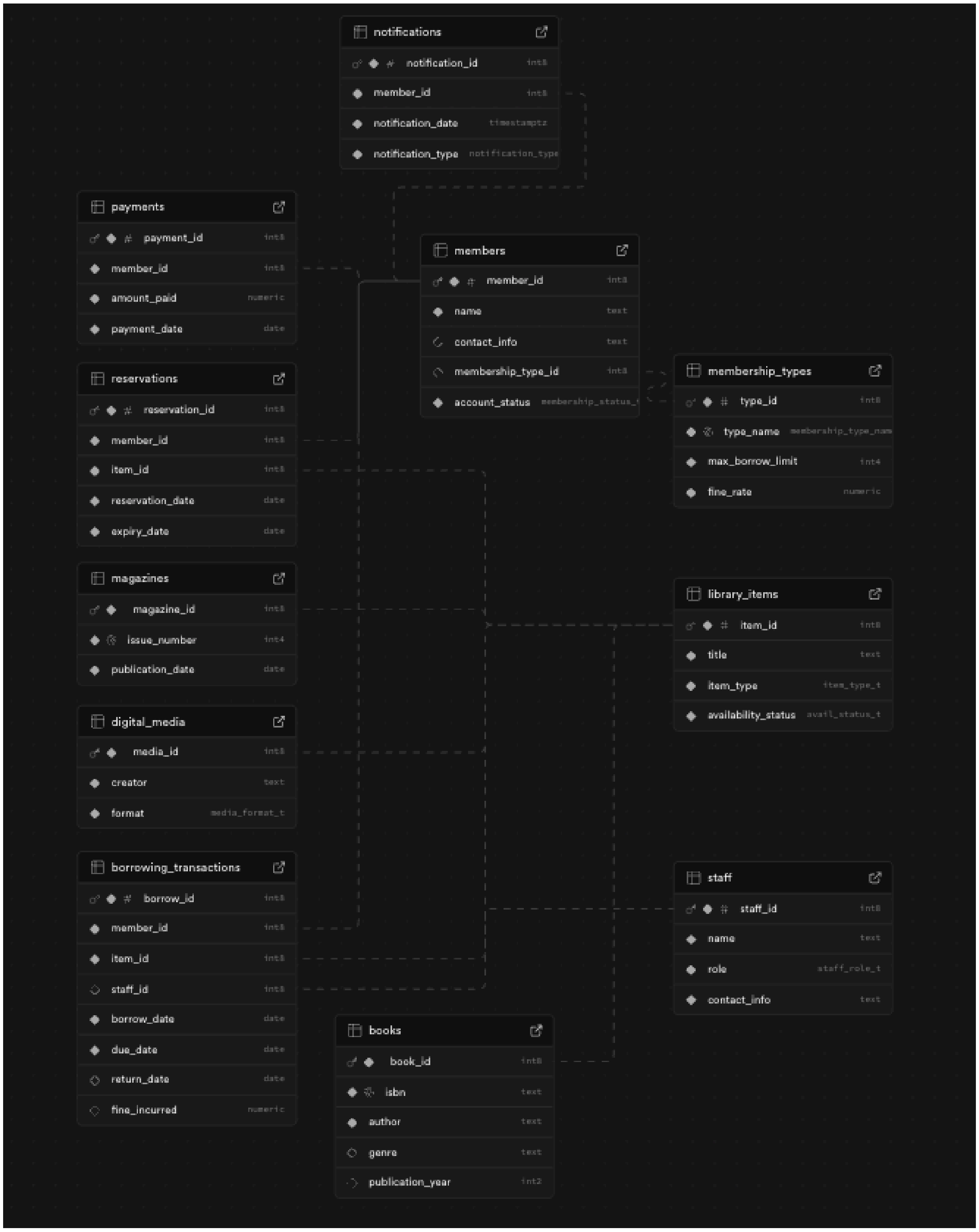
4. Physical Database Schema

The full schema (after both migrations) is captured in schema_dump.sql—generated with:

`supabase db dump --schema-only --file schema_dump.sql`

View the file here → [link-to-schema_dump.sql].

It includes all CREATE TYPE, CREATE TABLE, constraints, indexes, and trigger definitions for grader verification.


notifications
notification_id int8
member_id int8
notification_date timestamptz
notification_type notification_type
payments
payment_id int8
member_id int8
amount_paid numeric
payment_date date
members
member_id int8
name text
contact_info text
membership_type_id int8
account_status membership_status_t
membership_types
type_id int8
type_name membership_type_name
max_borrow_limit int4
fine_rate numeric
reservations
reservation_id int8
member_id int8
item_id int8
reservation_date date
expiry_date date
magazines
magazine_id int8
issue_number int4
publication_date date
library_items
item_id int8
title text
item_type item_type_t
availability_status avail_status_t
digital_media
media_id int8
creator text
format media_format_t
borrowing_transactions
borrow_id int8
member_id int8
item_id int8
staff_id int8
borrow_date date
due_date date
return_date date
fine_incurred numeric
staff
staff_id int8
name text
role staff_role_t
contact_info text
books
book_id int8
isbn text
author text
genre text
publication_year int2

5. Data Population

- **Lookup & staff rows** are inserted by **01_sample_data.sql** (executed via psycopg2 inside seed_db.py)
- **Bulk CSV fixtures** (generated once, version-controlled in **supabase/seed/**)

Table	CSV	Generated by
membership_types	membership_types.csv	manual
staff	staff.csv	manual
members	members.csv	seed_db.py
library_items	library_items.csv	seed_db.py
books	books.csv	seed_db.py
digital_media	digital_media.csv	seed_db.py
magazines	magazines.csv	seed_db.py
borrowing_transactions	borrowing_transactions.csv	seed_transactions.py
reservations	reservations.csv	seed_transactions.py
payments	payments.csv	seed_transactions.py
notifications	notifications.csv	seed_transactions.py

Loader scripts

Script	Role
01_sample_data.sql	Inserts the three membership‑type rows and two staff rows (fixed primary-key values).
seed_db.py	→ Truncates all tables → inserts lookup rows → COPYs the six core CSVs (members + catalog).

seed_transactions.py	⟶ Fabricates realistic circulation data, writes four CSVs, then COPYs them into the transactional tables.

6. Printing Table Contents

- print_tables.py executes SELECT * FROM <table> LIMIT 10; for every table and writes two deliverables to supabase/seed/output/:
    - row_counts.csv – a one-line summary of how many rows landed in each table.
    - <table>_preview.csv – the first 10 rows of every table (human-readable snapshot).

7. Github Repository

- Link: https://github.com/aelxxs/tech-titans