COMP 3005 Project Report

Name: Saurabh Gummaraj Kishore
Student ID: 101163778
Dec 11th, 2022

Table of Contents

2.1. Conceptual Design ... 3
2.1.1. User Stories/User Scope ... 3
2.1.2. Assumptions ... 3
2.1.3. Entity-Relationship Diagram ... 3
Cardinality ... 4
Participation ... 4
Assumptions based on Cardinality and Participation ... 4
2.2 Reduction to Relational Schemas ... 5
2.3 Normalization of Relational Schema ... 7
2.3.1. First Normal Form (1NF) ... 8
2.3.2. Second Normal Form (2NF) ... 8
2.3.3. Third Normal Form (3NF) ... 9
2.4 Database Schema Diagram ... 10
2.5 Implementation ... 12
2.6 Bonus Feature ... 13
2.7 GitHub Repository ... 13
2.8 Appendix I ... 13
References ... 14

2.1. Conceptual Design

This section of the report will address: the functionality that needs to be implemented into the database system. It includes any assumptions made; due to ambiguity in the scope or to increase efficiency of the system without compromising on functionality, as well as the database system's design as portrayed by an Entity-Relation Diagram using the UML-like notation.

2.1.1. User Stories/User Scope

Bookstore Owner

- As a bookstore owner, I need to be able to:
    - Add and remove books to my collection
    - Add customers details and see all existing customers
    - Add and remove publishers
    - Store publisher's information (name, address, email, banking details)
    - View sales analytics (units sold, revenue, expenses, etc.)
    - Automatically place orders for books that are low in stock

Customer

- As a customer at the bookstore, I need to be able to:
    - View books at the store
    - Add and remove books to my cart
    - Create a user profile (name, phone number, email, billing address)
    - Place an order
    - View status of my order

2.1.2. Assumptions

Some assumptions must be made to create a system that works efficiently and is not too taxing to build. One such assumption would be to assume that our bookstore will not keep track of any payment information due to security risks. Instead, our payment system will use a reliable third-party payment gateway. Additionally, the replenishment of low-stock books will happen at a fixed value. Once a book reaches 10 units or below, a fixed value of 100 books will be added to an order sent to the publisher by the owner using the automated system. Another assumption is that the customer will only be able to see what price each book is sold at and will not be able to see what price each book was bought at by the owner. Lastly, there can only be one owner for this store.

2.1.3. Entity-Relationship Diagram

This entity relationship will describe the overarching design for how each entity will interact and relate with other entities. This is an important aspect of the design of this project since from this diagram we can gain a better understanding as to how to structure each entity, what

type of interactions or relationships we will need to build, and lastly, we can gain a better understanding about the cardinality of each entity. It is important to keep in mind that the following ER diagram (Figure 1) will use the Unified Model Language notation or more commonly known as UML-notation.

Cardinality

The cardinality of an entity gives us information about the number of times an entity needs to associate with another entity. There are three types of cardinalities:

1. One to One (1:1) – for every one entity there is exactly one occurrence of another entity
2. One to Many (1:M) – for every one entity there are many occurrences of another entity
3. Many to Many (M:M) – for each occurrence of an entity, there are many occurrences of another entity

Participation

The participation of an entity tells us whether an entity must participate in a relationship with another entity. There are two types of participation:

- Total Participation (1)– this type of participation means that an entity must participate in a relationship to exist. In other words, every entity must be associated with another relationship.
- Partial Participation (0) – this type of participation means that an entity can exist without having to participate in the relationship with another entity.

Assumptions based on Cardinality and Participation

- The lower bound of the cardinality indicates the participation type. Hence, (1:M) or (1:1) would suggest that there is total participation between the two entities. Likewise, (0:M) or (0:1) suggests that participation is optional or that there is partial participation between the two entities.
- There can only be one publisher per book, but a publisher can publish many books.
- There can only be one owner.
- One owner can collect many books, but all books have only owner.
- One customer can place zero or many orders, but every order must have exactly 1 customer.
- Every order must contain a book and an ordered book must belong to exactly one order.

The following page contains the ER diagram.

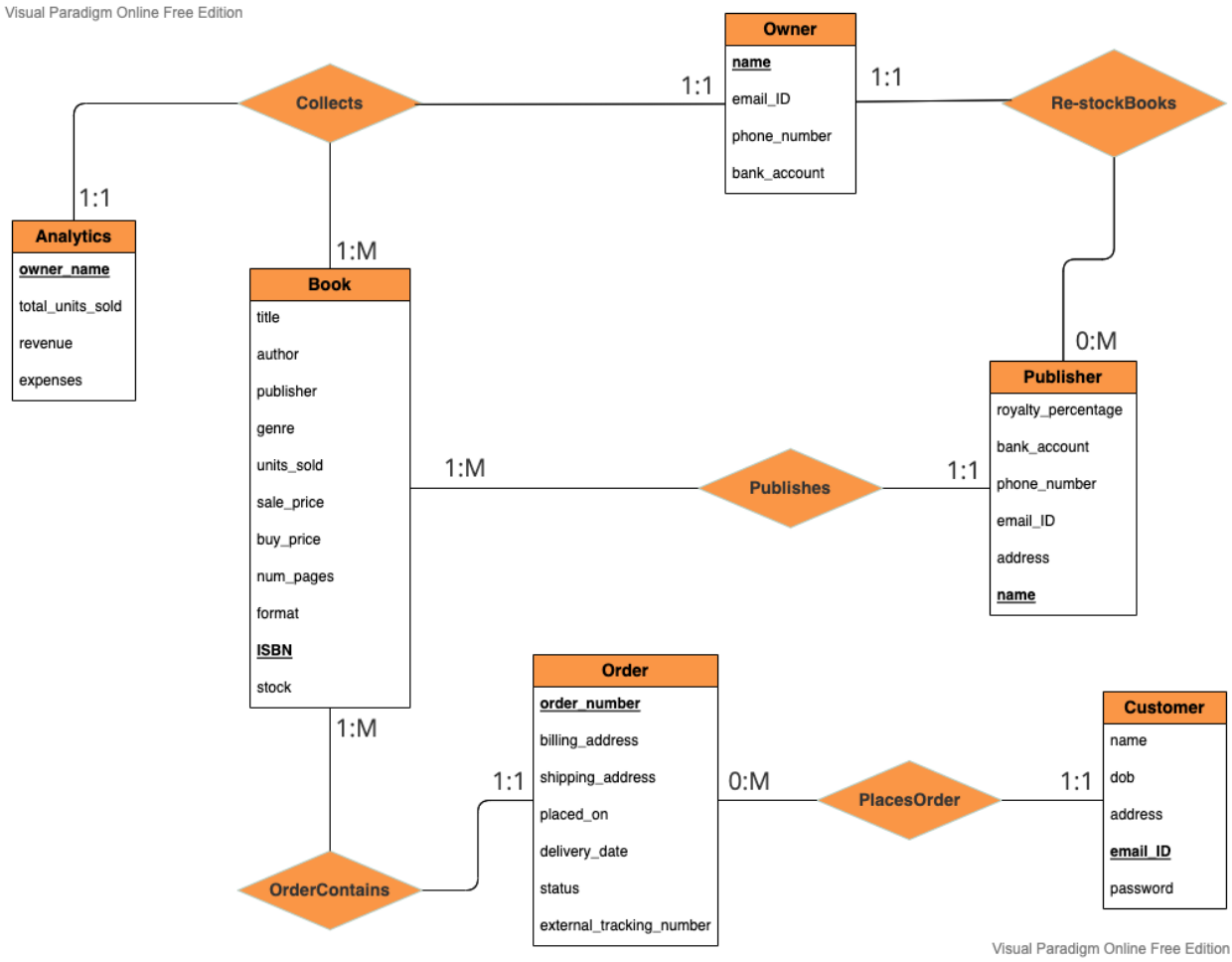


Figure 1 – Entity Relationship Diagram using UML-Notation.

2.2 Reduction to Relational Schemas

In order to better understand how each relation interacts with the entities, we can convert our ER diagram into what is known as relational schemas. A relational schema is a set of tables, where each table is known as a relation. Each relation tells us how items withing a table is related to one another. Relational schemas are heavily reliant on primary keys and foreign keys. A primary key is the unique identifier for each table; hence we cannot have duplicate values or null values take the roll of a primary key. Foreign keys on the other hand are a dependent value of another primary key that belongs to another table; we can have multiple foreign keys per table, and it can also contain duplicate and null values. In other words, the foreign key is the commonality between two or more tables, whereas the primary key is the unique identifier value of a table.

Before we can convert our conceptual model (ER diagram) into a logical model (Relational Schema), we need to understand the different types of entities and relations that exist in our conceptual model.

- Strong entities – strong entities are tables that have their own primary key and are not dependent on other tables for its existence.
- Weak entities – weak entities are tables that do not have their own primary key but rather relies on foreign keys from other tables for its existence.
- Binary relations – a relationship bound by two different entities.
- Unary relations – a relationship in which only a single entity participates in.
- Tertiary relations – a relationship bound by three different entities.

The first step to reducing an ER diagram to relational schema is to identify the strong and weak entities. Strong and weak entities can be reduced to a table when converting an ER diagram to a relational schema. However, weak entities need to be given a foreign key and strong entities need to have exactly one primary key. In our ER diagram, we have five strong entities:

BOOK (title, author, publisher, genre, units_sold, sale_price, buy_price, num_pages, format, ***ISBN****, stock)*

*OWNER (****owner_ID****, name, email_ID, phone_number, bank_account)*

*CUSTOMER (****customer_ID****, name, dob, email_ID, password)*

*PUBLISHER (****publisher_ID****, royalty_percentage, bank_account, phone_number, email_ID, address, name)*

*ORDER (****order_number,*** *billing_address, shipping_address, placed_on, delivery_date, status, external_tracking_number)*

We have one weak entity:

*ANALYTICS (****owner_ID****, total_units_sold, revenue, expenses)*

We have two binary relations, one of them with partial participation of one to many and the other one with total participation of one to many:

*PLACES_ORDER (****customer_ID****,* ***order_number****, name)*

*ORDER_CONTAINS (****order_number****,* ***ISBN****, units)*

Lastly, we have two tertiary relations:

*COLLECTS (****ISBN****, sale_price, buy_price, units_sold,* ***owner_ID****)*

*RESTOCK_BOOKS (**publisher_ID**, email_ID, **ISBN**, bank_account, buy_price, stock, **owner_ID**)*

Now that we have reduced all the entities and relations into tables, we can proceed to normalize them. Here is what the final reduced relational schema looks like.

*BOOK (title, author, publisher, genre, units_sold, sale_price, buy_price, num_pages, format, **ISBN**, stock)*

*OWNER (**owner_ID**, name, email_ID, phone_number, bank_account)*

*CUSTOMER (**customer_ID**, name, dob, email_ID, password)*

*PUBLISHER (**publisher_ID**, royalty_percentage, bank_account, phone_number, email_ID, address, name)*

*ORDER (**order_number,** billing_address, shipping_address, placed_on, delivery_date, status, external_tracking_number)*

*ANALYTICS (**owner_ID**, total_units_sold, revenue, expenses)*

*PLACES_ORDER (**customer_ID**, **order_number**, name)*

*ORDER_CONTAINS (**order_number**, **ISBN**, units)*

*COLLECTS (**ISBN**, sale_price, buy_price, units_sold, **owner_ID**)*

*RESTOCK_BOOKS (**publisher_ID**, email_ID, **ISBN**, bank_account, buy_price, stock, **owner_ID**)*

2.3 Normalization of Relational Schema

Normalization is a process by which we remove redundancy that may exist in our table. We do this by following the different forms of normalization and making sure that our table satisfies the conditions set out at each level. There are six categories in which every table must fit into. These are called normal forms, in total there are six normal forms however, for the sake of this project we will only by using the first three normal forms.

2.3.1. First Normal Form (1NF)

The first normal form permits only single values at each row and column. Hence, there is no room for repeats. This removes redundancy by allowing us to create a more efficient database design where, if we see any repeated values within the same table, we can safely assume that it would need its own table comprised of the repeated values.

To check if our design meets the first normal form parameters, we can objectively look at each table and see if there are any values that would be repeated as our table grows. This is a crucial step when it comes to scaling up our database to handle large amounts of data.

Since we have made use of primary keys and foreign keys in our relational schema, no repeated value is immediately evident, and if there are repeated values they come as a reference to a foreign key from another table. Hence, our repeated values are already in a different table. This means that our database design now satisfies the first normal form condition.

2.3.2. Second Normal Form (2NF)

Before we can check to see if our tables are in second normal form, we need to make sure that they all meet the requirement for the first normal form. The second condition for a table to be in second normal is based on functional dependency and composite keys. This means that all attributes of any table with more than one primary key must be fully dependant on the primary key. This can be demonstrated below:

BOOK (title, author, publisher, genre, units_sold, sale_price, buy_price, num_pages, format, ***ISBN****, stock)*

ISBN →title, author, publisher, genre, units_sold, sale_price, buy_price, num_pages, format, stock

*OWNER (****owner_ID****, name, email_ID, phone_number, bank_account)*

owner_ID → name, email_ID, phone_number, bank_account

*CUSTOMER (****customer_ID****, name, dob, email_ID, password)*

customer_ID → name, email_ID, dob, password

*PUBLISHER (****publisher_ID****, royalty_percentage, bank_account, phone_number, email_ID, address, name)*

publisher_ID → name, royalty_percentage, bank_account, phone_number, email_ID, address

*ANALYTICS (**owner_ID**, total_units_sold, revenue, expenses)*

owner_ID → total_units_sold, revenue, expenses

*ORDER (**order_number**, billing_address, shipping_address, placed_on, delivery_date, status, external_tracking_number)*

order_number → billing_address, shipping_address, placed_on, delivery_date, status, external_tracking_number

*PLACES_ORDER (**customer_ID**, **order_number**, name)*

customer_ID, order_number → name

*RESTOCK_BOOKS (**publisher_ID**, email_ID, **ISBN**, bank_account, buy_price, stock, **owner_ID**)*

publisher_ID, ISBN, owner_ID → email_ID, bank_account, buy_price, stock

*ORDER_CONTAINS (**order_number**, **ISBN**, units)*

order_number, ISBN → units

*COLLECTS (**ISBN**, sale_price, buy_price, units_sold, **owner_ID**)*

ISBN, owner_ID → sale_price, buy_price, units_sold

By looking at the above functional dependencies, we can see that each table has its own primary key which other attribute within that table rely on. Hence, our tables are now in second normal form.

2.3.3. Third Normal Form (3NF)

The third normal form removes what is known as transitive dependencies. However, before we do this, the first condition of the third normal form is to make sure that our tables are in second normal form. Now we can proceed. A transitive dependency is when a there is an indirect relationship between values within the same table. This causes a functional dependency since if one value updates the transitive value might still refer to the old value. To avoid this, we need to make sure that all our functional dependent values come from values that are primary keys within their own tables as well as removing any redundant values from our table.

Firstly, from our 2NF tables, we can remove a few tables since they are duplicates of a process and redundant. One such example is the RESTOCK_BOOKS table. This table possess the same functionality of an order. However, in this case it is the owner that needs to order from the publisher. Since this is an automatic process that is dependent on the number of units in inventory (stock) of each book, we can simply write a query for this and not use a table at all.

We can also remove the PLACES_ORDER table and ORDER_CONTAINS tables since both these tables contain duplicate values and the whole point of normalization is to eliminate any redundant tables. What we can do is directly relate the BOOK table with our ORDER table using the ISBN and relate the CUSTOMER table with the ORDER table by using the customer_ID attribute. However, we will now need to add a "units" column as well to our ORDER table to track the number of units requested.

We can eliminate the ANALYTICS table since the values within these tables are entirely dependent on other values such as the number of units sold for each book and the price of each unit and so on, having a static value initialized into the database would be bad design. This is because every time we need to calculate the revenue or expense, we need to access all the other values needed to calculate the revenue or expense, as well as update the column itself. This would be better suited as query. The same is true for every other value within this table.

After these changes, we have attempted to remove any redundancy within our design with the goal of creating a simpler database with fewer tables.

2.4 Database Schema Diagram

The database schema diagram gives us a great amount of detail on both the conceptual and logical design of our database. It does so by showing the interaction between each table, primary keys, and foreign keys. Primary keys are usually characterized by using a bold font with a line under them whereas foreign keys often have a reference arrow coming out of them. By taking the into account our relational schema and the normalization of them we can create a database schema diagram.

Keeping in mind that good database design is one that has fewer null values, less duplicates, and most importantly fewer tables. We can see how we attempted to achieve this by looking at our database schema diagram and comparing them between our normal forms.

In the next two pictures, we will see that we have greatly reduced the number of tables we used when we moved from second normal form to third normal form. We will also see how we eliminated redundant and duplicate values by using primary and foreign keys more efficiently, as well as taking advantage of the data query language to implement some of the functionality needed.

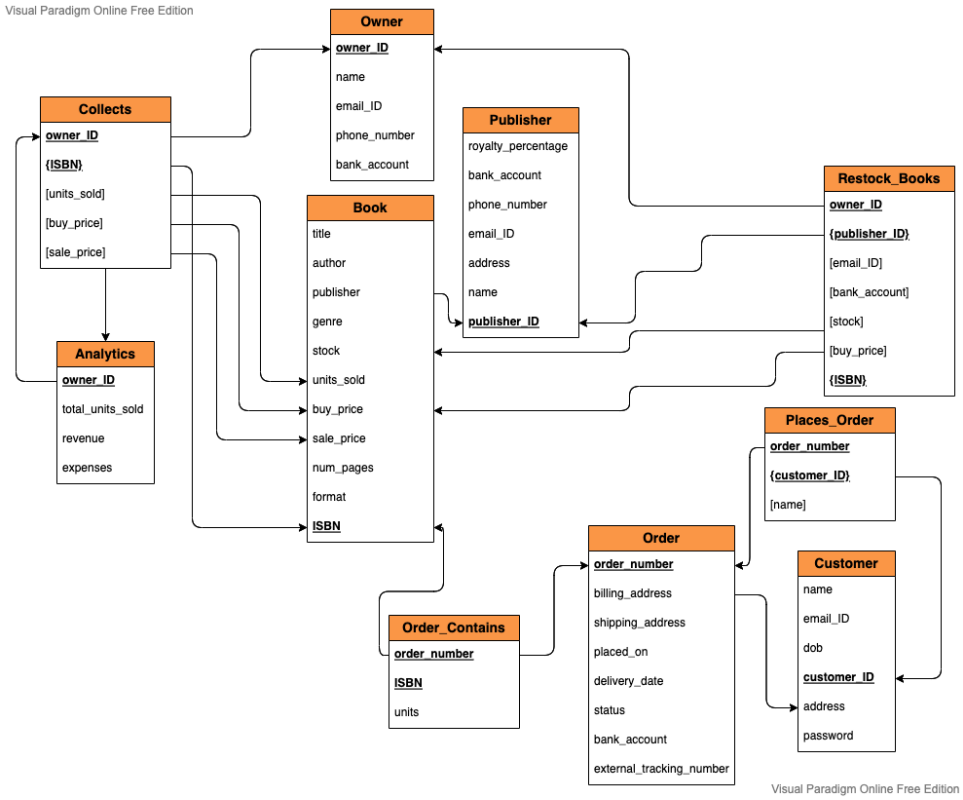


Figure 2 – Database schema diagram after second normal form (2NF)

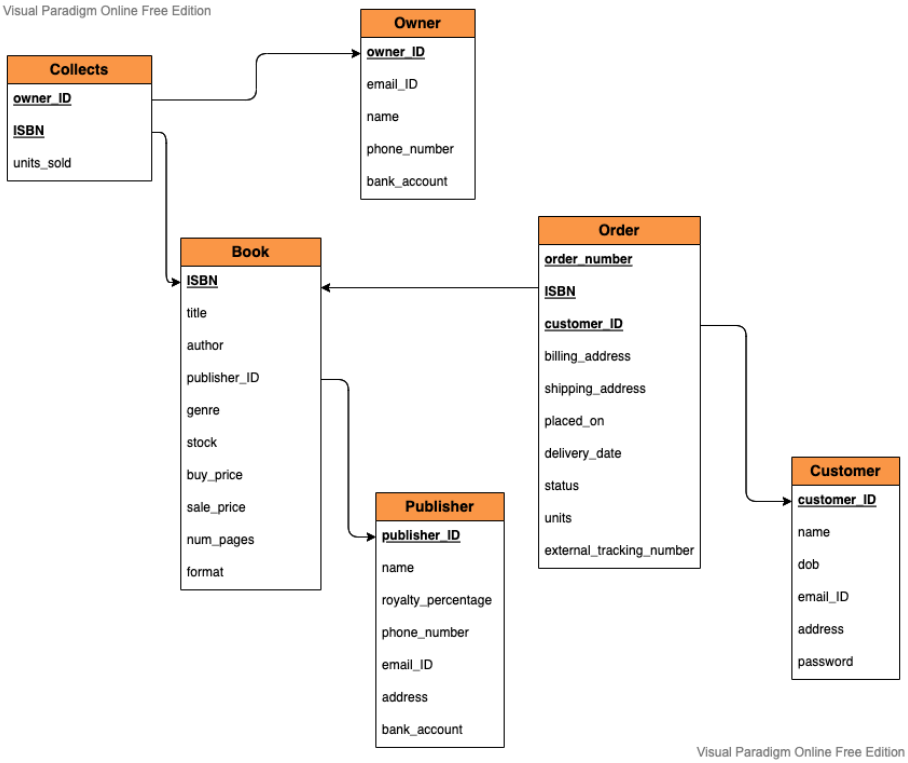


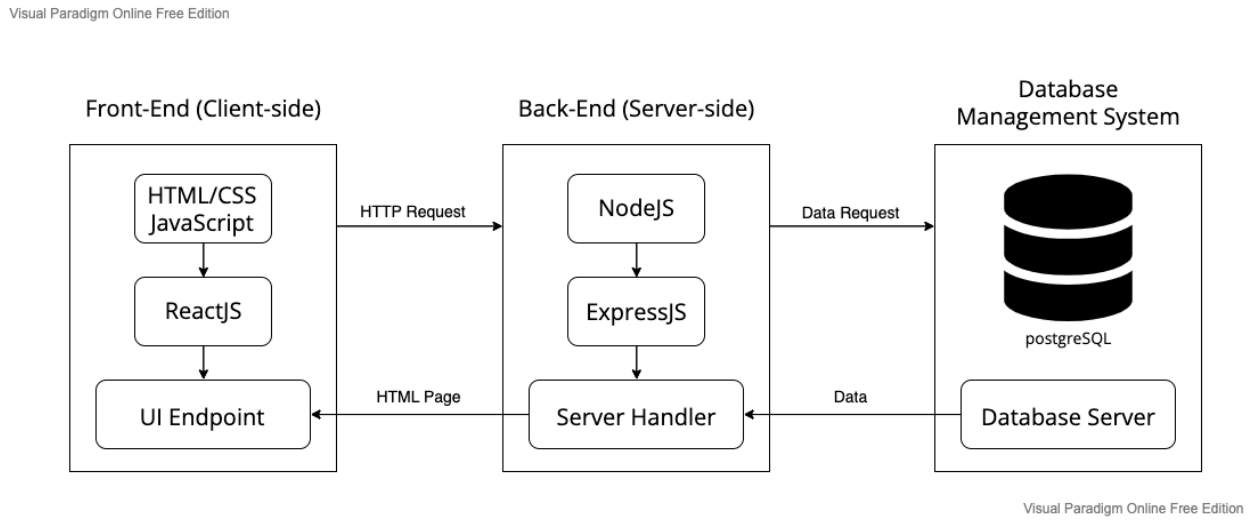
Figure 3 – Database schema diagram after third normal form (3NF)

The juxtaposition of the above figures (Figure 2 and Figure 3) clearly demonstrates the importance of normalization in order to work with a logical database that is scalable, non-redundant, and efficient.

2.5 Implementation

A general implementation for this would be to start of with a back-end framework that handles server requests based on the functionality needed. The backend will also be responsible to connect to an authentication service either built by the author or by connecting to an existing authentication API. The backend will also be responsible for connecting to the database and handling queries and extrapolating data from the database design. The frontend will can be built using frontend frameworks which will allow users to interact with the application in a more user friendly way by implementing concepts from user interface and user experience design. The backend and frontend will need to communicate in order for the entire application to function as per the needs of the author and user stories provided.

We may take a very common technology stack used to build web application. This technology stack uses SQL as our database query language, Express.js as our backend frame work, React.js as our frontend framework, and Node.js as the runtime environment. This tech-stack is commonly known as the SERN stack. It is important to keep in mind that we may also use scripting, markup, and styling languages such as JavaScript, HTML and CSS. The following image shows how the frameworks will communicate with each other.



2.6 Bonus Feature

Please note that this section is not included in this report.

2.7 GitHub Repository

The GitHub repository for this project can be found at https://github.com/gksaurabh/LibDB

2.8 Appendix I

I will be available on December 12th, 2022 during the following time:

- 10:00 am
- 11:00 am
- 1:00 pm

References

Eng, N., & Watt, A. (2014, October 24). *Chapter 12 Normalization*. Database Design 2nd Edition. Retrieved December 11, 2022, from https://opentextbc.ca/dbdesign01/chapter/chapter-12-normalization

Meena, S. (2022, July 20). *Web application architecture part-1 (guide to become full stack developer)*. Medium. Retrieved December 11, 2022, from https://engineering.csiworld.in/web-application-architecture-part-1-guide-to-become-full-stack-developer-cc9526a3519b

Nalimov, C. (2020, November 18). *Diagram maker for developers*. Gleek. Retrieved December 11, 2022, from https://www.gleek.io/blog/relational-schema.html