Z2004: DATABASE MANAGEMENT SYSTEMS

FABLE,

AN ONLINE BOOKSTORE.

"Database Management Systems; Submitted on 20th June, 2025"
- Anonymous.

BY KUDRAAT ABOUD

ZDA23B021

Table Of Contents

INTRODUCTION

Project Overview.

The Fable Database Management System, which will, from here on out, be referred to as the FDBMS, is one designed to act as the backbone of Fable, an E-commerce business where users can find and purchase books, review them, borrow them from the Fable library, and more. The aim of the FDBMS project is to store and handle the copious amounts of data held on Fable, managing entities such as books, their authors, user reviews, orders and carts.

The Objectives.

The FDBMS project had the following objectives:

- *Book Catalogue Management*: An important thing during the development of the FDBMS was to have a large variety of books to fill the inventory with, and to allow users to be able to filter the books based on author, genre, price and more. This goal has been expanded upon with further additions such as the ability for those running the store to see insights into popular novels and user sentiments.

- *Review And Rating Handling*: A goal of the FDBMS' was to store user reviews and their ratings, so that users can make informed decisions on their purchases, as well as leave some of their own.

- *User Account Management*: This includes storing information on the user, such as their login credentials, email, name, address and more. An added step to this objective is the ability of storing all the above, as well as the kinds of literature they interact and engage with, which is information that can be used to create targeted and strongly backed recommendations.

- *Order Processing System*: As an E-commerce website, Fable users can place orders on the website receive their books in either a physical or digital form. An objective of the FDBMS is to hold information on what orders have been placed, their status, the quantity of units sold and returned, and other pieces of information to have a comprehensive image of the websites ordering system.

- *Ensuring Efficient Data Storage and Retrieval*: Another aim was for both the users end as well as those running the ship in the back, to be able to quickly and efficiently retrieve information on books, orders, customers and more with the use of optimised database queries implemented into the system.

- *Inventory And Stock Management*: The FDBMS keeps track of the quantity of books in stock, their current prices and more. The FDBMS is meant to enforce basic constraints such as item quantities being capped out at the number of units available on hand.

- *Membership System*: Like other E-commerce sites, an aspect of Fable is the ability for users to subscribe to a membership tier for additional perks. One of the goals of the FDBMS is to track which users are members and which ones aren't, which benefits a member can access and attributes such as how many credit points they have left.

The Scope.

The completed FDBMS serves as the foundational backend required to deliver a seamless and scalable user experience across all aspects of the Fable website; It includes the development and integration of systems for book catalogue management, review and rating storage, user account handling, order processing, inventory tracking, and a membership tier framework. The FDBMS also supports advanced functionalities such as search filtering, sentiment analysis, and user engagement tracking to inform recommendations and business insights. The system ensures data integrity, enforces key business

constraints, and is optimized for efficient retrieval and updates, benefiting both front end users and administrators.

DATA DICTIONARY

Table	Attribute	Data Type	Constraints	Description
Addresses	ID	INT	PK, NOT NULL	Unique identifier for each stored address.
Addresses	UserID	INT	FK, References Users	Foreign key that links the stored addresses to their respective user.
Addresses	Town	NVAR CHAR (MAX)	NOT NULL	Attribute that stores the town the user lives in.
Addresses	Address	NVAR CHAR (MAX)	NOT NULL	Attribute that stores user's street name and house.
Authors	ID	INT	PK, NOT NULL	Unique identifier for each author.
Authors	First_Name	NVAR CHAR (MAX)	NOT NULL	First name of the author.
Authors	Last_Name	NVAR CHAR (MAX)	NULL	Last name of the author.
Authors	About_ Author	NVAR CHAR (MAX)	NULL	Brief description of the author.
Books	ID	INT	PK, NOT NULL	Unique identifier for each author.
Books	Title	NVAR CHAR	NOT NULL	Name of the book.

		(MAX)		
Books	Author_s	NVAR CHAR (MAX)	NOT NULL	Name(s) of the author(s) of the book.
Books	Rating	FLOAT	NULL	Aggregate rating of the book based on user ratings.
Books	Rating_count	FLOAT	NULL	Number of ratings left by users.
Books	Review_ count	FLOAT	NULL	Number of reviews left by users.
Books	No_Of_ Pages	FLOAT	NULL	Number of pages in the book.
Books	Format	NVAR CHAR (50)	NULL	Format of the book i.e. Paperback, Audiobook, etc.
Books	Awards	NVAR CHAR (1350)	NULL	Name(s) of award(s) awarded to the book.
Books	Genres	NVAR CHAR (1350)	NULL	Name(s) of the genre(s) the book belongs to.
Genres	ID	SMALL INT	PK, NOT NULL	Unique identifier for each genre.
Genres	Genre	NVAR CHAR (50)	NOT NULL	Name of the genre.
Genres	Description	NVAR CHAR (3900)	NULL	Brief description on the nature of the genre.
Inventory	ID	INT	PK, NOT NULL	Unique identifier for each product.
Inventory	BookID	INT	FK, References Books, NOT NULL	Foreign key that links each stored product to a book in the Books catalogue.

Inventory	Price	FLOAT	NOT NULL	Selling cost of each product.
Inventory	Stock	INT	NOT NULL, Qnt_Available_ Check >= 0	Number of units of the product available.
Members	ID	INT	PK, NOT NULL	Unique identifier for each Fable member.
Members	UserID	INT	FK, References Users, NOT NULL	Foreign key that links together the Members and Users table on the Users.ID value.
Members	Username	NVAR CHAR (100)	NOT NULL	Username of the Fable member.
Members	Membership_ Tier	NVAR CHAR (50)	FK, References Memberships, NOT NULL	Foreign key that links the Members to the Memberships table for information on the members perks.
Members	Credit_ Balance	SMALL INT	NOT NULL	Number of credits the user has left in their account.
Member ships	Tier	NVAR CHAR (50)	PK, NOT NULL	Name of the Fable membership subscription tier.
Member ships	Description	NVAR CHAR (300)	NOT NULL	Brief description on the perks provided by each tier.
Member ships	Credits_mont h	TINY INT	NOT NULL	Number of credits provided per month for the tier.
Member ships	Price	FLOAT	NOT NULL	Cost of the membership tier per month.
Order_ Items	OrderID	INT	FK, References Orders, NOT NULL	Foreign key that links the item ordered to the order and user it belongs to.

Order_ Items	Product_ID	INT	FK, References Inventory, NOT NULL	Foreign key that links the item ordered to the product pricing and other details.
Order_ Items	Quantity	SMALL INT	NOT NULL	Number of that item that have been ordered.
Orders	ID	INT	PK, NOT NULL	Unique identifier for each placed order.
Orders	UserID	INT	FK, References Users, NOT NULL	Foreign key that links the order and the user it belongs to.
Orders	OrderStatus	NVAR CHAR (50)	NOT NULL	Describes the current state of the order.
Orders	PaymentMeth od	INT	FK, References Payment_ Methods, NOT NULL	Foreign key that links together the order and the payment method used in the transaction.
Orders	CartID	INT	FK, References Shopping_Carts, NOT NULL	Foreign key that references the cart from which the order came.
Orders	OrderDate	DATE	NOT NULL	The stored date on which the order was placed.
Orders	TotalAmount	FLOAT	NOT NULL	The price paid (or refunded) for the order.
Payment_ Methods	ID	INT	PK, NOT NULL	Unique identifier for each stored payment method.
Payment_ Methods	UserID	INT	FK, References Users, NOT NULL	Foreign key that links the stored payment method and the user to which it belongs to.
Payment_ Methods	Card Numbers	BIG INT	NOT NULL	Card details.
Payment_ Methods	CardType	NVAR CHAR (50)	NOT NULL	Information on the card distributor.

Reviewers	ReviewerID	INT	PK, NOT NULL	Unique identifier for each reviewer.
Reviewers	Username	NVARCHAR(MAX)	NOT NULL	The reviewer's username, can link to the users table.
Reviews	ID	INT	PK, NOT NULL	Unique identifier for each review left on Fable.
Reviews	ReviewerID	INT	FK, References Reviewers, NOT NULL	Foreign key that links each review with the user that wrote it.
Reviews	Review	NVARCHAR(50)	NULL	The content of the review.
Reviews	BookID	INT	FK, References Books, NOT NULL	Foreign key that references to which book the review belongs.
Reviews	Date	DATE	NOT NULL	Describes the date the review was published.
Reviews	Rating	FLOAT	NULL	The rating, out of 5, the reviewer gives the book in the review.
Shopping_Cart_Items	CartID	INT	FK, References Shopping_Carts	Foreign key that links the item to the cart where it belongs.
Shopping_Cart_Items	ProductID	INT	FK, References Inventory	Foreign key that links the item to the inventory for price fetching and etc.
Shopping_Carts	ID	INT	PK, NOT NULL	Unique identifier for each shopping cart.
Shopping_Carts	UserID	INT	FK, References Users, NOT NULL	Foreign key that links the shopping cart to the user it belongs to.
Shopping_Carts	TotalAmount	FLOAT	NOT NULL	Total amount of all items within the cart.
Users	ID	INT	PK, NOT NULL	Unique identifier for each user.

Users	Username	NVARCHAR (MAX)	NOT NULL	The users screen-name for Fable.
Users	First_Name	NVARCHAR (50)	NOT NULL	The users first name.
Users	Last_Name	NVARCHAR (50)	NULL	The users last name.
Users	EmailAddress	NVARCHAR (50)	NOT NULL	The users email address associated with the account.
Users	DOB	DATE	NOT NULL	Date of birth.
Users	AddressID	INT	FK, References Addresses, NOT NULL	Foreign key that links the user to their stored address in Addresses.
Wishlist	ID	INT	PK, NOT NULL	Unique identifier for each wishlist.
Wishlist	UserID	INT	FK, References Users, NOT NULL	Foreign key that links each wishlist to its respective user.
Wishlist_Items	WishlistID	INT	FK, NOT NULL	Foreign key that links the wishlisted item to the wishlist it belongs to.
Wishlist_Items	ProductID	INT	FK, NOT NULL	Foreign key that links the item to the Inventory table for price checking and more.

Table 1: Fable Database Management System Data Dictionary.

Conceptual Design

Entity Relationship Rationale & Choices.

Before diving into the Entity-Relationship diagram, let us first discuss the entity choices we will observe within the ER diagram that were briefly touched upon in the data dictionary section earlier.

As Fable is an E-commerce website that primarily sells books across different mediums, the main entities are appropriately book-related; Books, Genres, Authors to name a few, while also having entities that are from the business side of things such as Orders, Shopping_Carts and Wishlists. Weak entities such as Order_Items, Shopping_Cart_Items and Wishlist_Items were used to help with normalisations, and depend on their respective name counterparts.

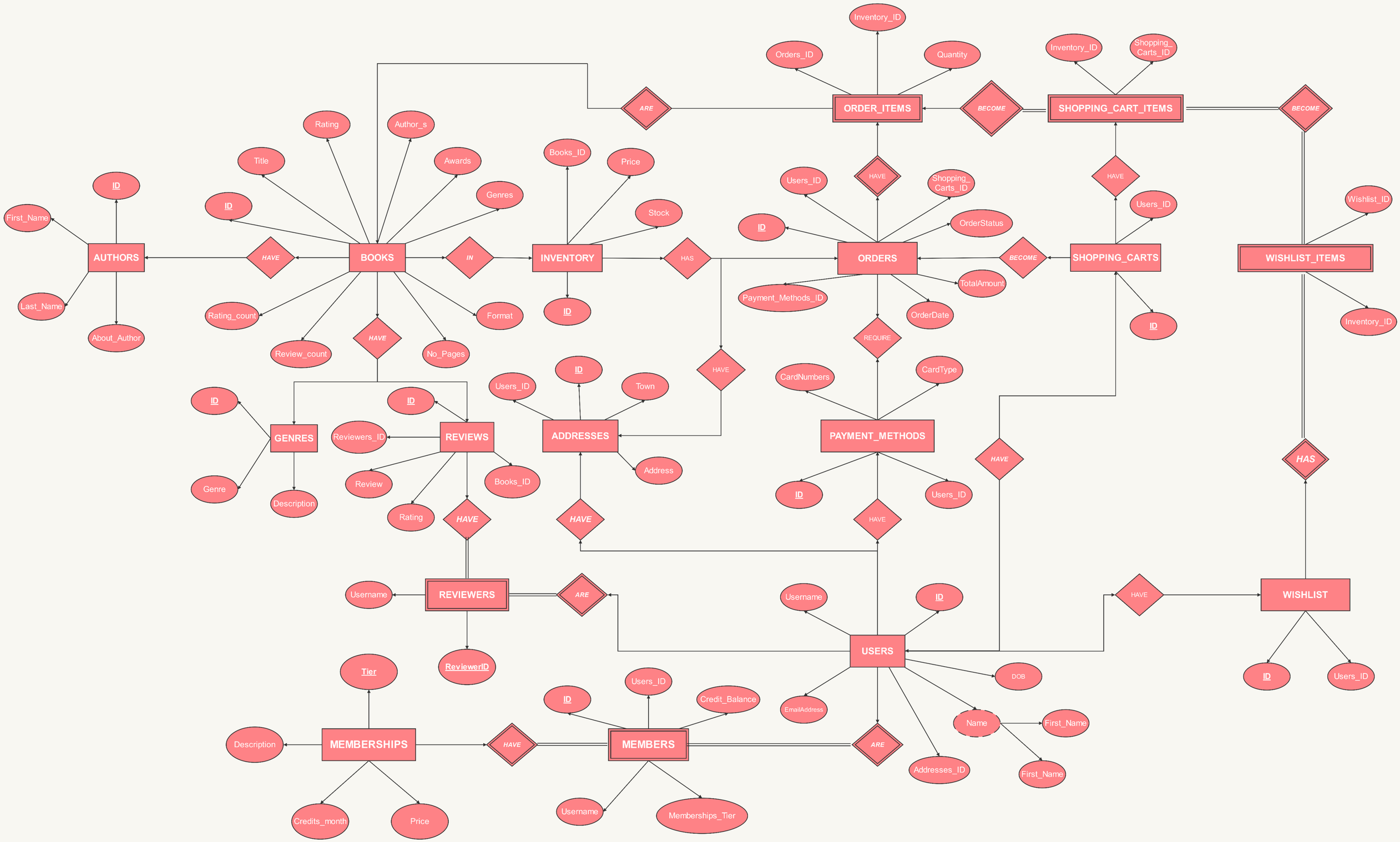
A few assumptions made during the conceptual design phase are:

i. Every users has a single shopping cart and wishlist: This one is relatively self-explanatory. Shopping carts and wishlists are essentially temporary stores for users to place items, those they are (relatively) sure they want to purchase and those they are not.
ii. Customers can place multiple orders, and each order can have multiple books. While generating the data for how many books each order would have, it was decided that a suitable range would be between 1 and 10 books; any more seemed a bit overkill.
iii. Not all users leave reviews; those who do were stored in a separate table called reviewers. This was done in part to store the reviewer information obtained while scraping the books on Goodreads as well as to simplify the querying process while trying to track user preferences based on their review history.

The above are just a few of the assumptions made; in the next section is the ER diagram for the Fable Database Management System.

Entity-Relationship Diagram.

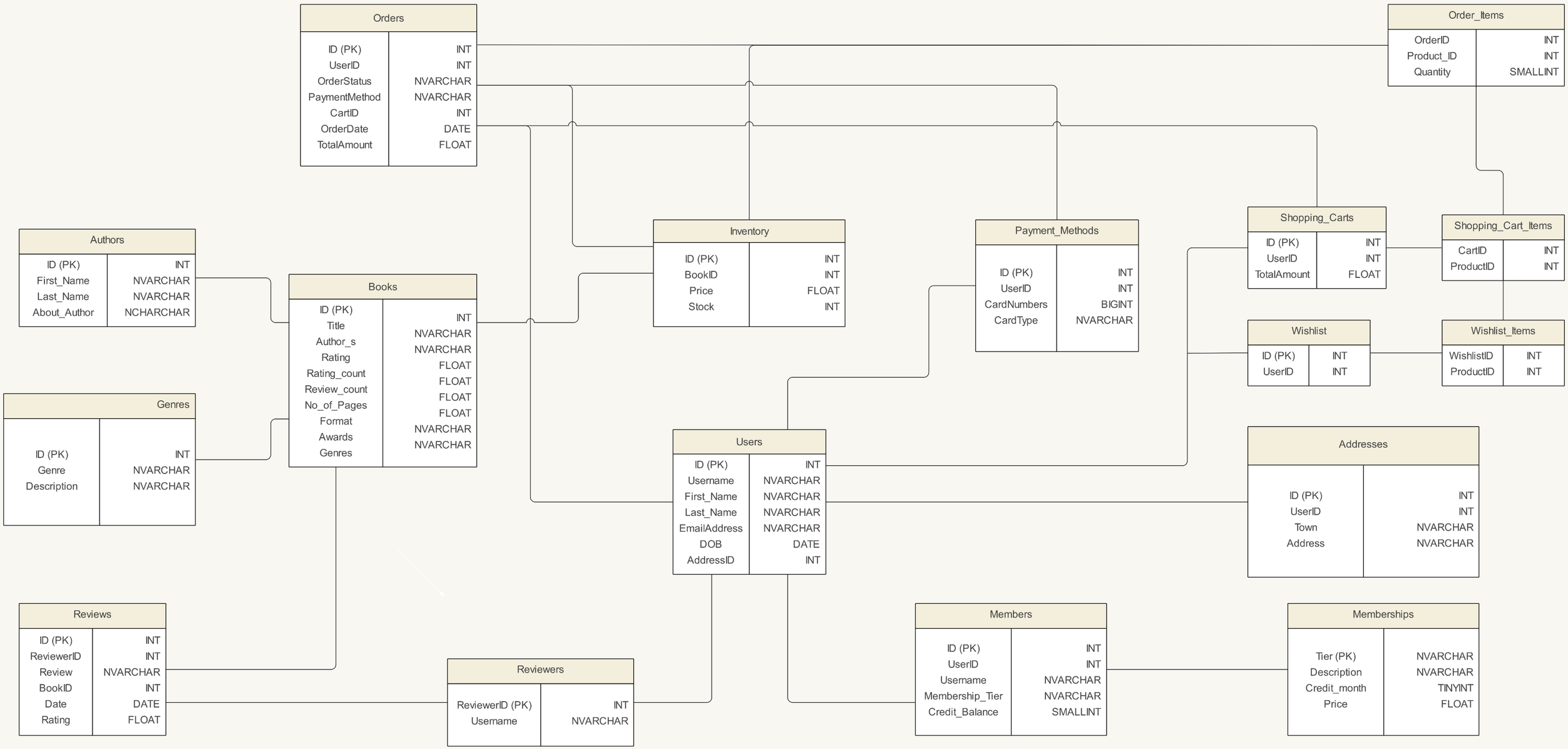
Figure 1: Final Entity Relationship Diagram For the FDBMS.



Relational Schema

Relational Schema Diagram.

Figure 1: Relational Schema Diagram For the FDBMS.



Detailed Relational Schema.

Addresses (ID: INT, UserID: INT, Town: NVARCHAR(MAX), Address: NVARCHAR(MAX))

PK: ID

FK: USerID

Authors (ID: INT, First_Name: NVARCHAR(MAX), Last_Name: NVARCHAR(MAX), About_About: NVARCHAR(MAX))

PK: ID

FK: None

Books (ID: INT, Title: NVARCHAR(MAX), Author_s: NVARCHAR(MAX), Rating: FLOAT, Rating_count: FLOAT, Review_count: FLOAT, No_of_Pages: FLOAT, Format: NVARCHAR(50), Awards: NVARCHAR(1350), Genres: (1350))

PK: ID

FK: None

Genres (ID: INT, Genre: NVARCHAR(50), Descriptions: NVARCHAR(3900))

PK: ID

FK: None

Inventory (ID: INT, BookID: INT, Price: FLOAT, Stock: INT)

PK: ID

FK: BookID

Members (ID: INT, UserID: INT, Username: NVARCHAR(100), Membership_Tier: NVARCHAR(50), Credit_Balance: INT)

PK: ID

FK: UserID, Membership_Tier

Memberships (Tier: INT, Description: NVARCHAR(300), Credits_month: TINY INT, Price: FLOAT)

PK: ID

FK: None

Order_Items (OrderID: INT, ProductID: INT, Quantity: SMALLINT)

PK: None

FK: None

Orders (ID: INT, UserID: INT, OrderStatus: NVARCHAR(50), PaymentMethod: INT, CartID: INT, OrderDate: DATE, TotalAmount: FLOAT)

PK: ID

FK: UserID, PaymentMethod, CartID

Payment_Methods (ID: INT, UserID: INT, CardNumbers: BIGINT, CardType: NVARCHAR(50))

PK: ID

FK: UserID

Reviewers (ReviewerID: INT, Username: NVARCHAR(MAX))

PK: ReviewerID

FK: None

Reviews (ID: INT, ReviewerID: INT, Review: NVARCHAR(MAX), BookID: INT, Date: DATE, Rating: FLOAT)

PK: ID

FK: ReviewerID, BookID

Shopping_Cart_Items (CartID: INT, ProductID: INT)

PK: None

FK: CartID, ProductID

Shopping_Carts (ID: INT, UserID: INT, TotalAmount: FLOAT)

PK: ID

FK: UserID

Users (ID: INT, Username: NVARCHAR(MAX), First_Name: NVARCHAR(50), Last_Name: NVARCHAR(50), EmailAddress: NVARCHAR(50), DOB: DATE, AddressID: INT)

PK: ID

FK: AddressID

Wishlist (ID: INT, UserID: INT)

PK: ID

FK: UserID

Wishlist_Items (WishlistID: INT, ProductID: INT)

PK: None

FK: WishlistID, ProductID

Normalisation.

Most of the tables within the database are in the Third Normal Form (3NF), notably Genres, Authors, Addresses, Members, Memberships, PaymentMethods, Users and a few more. They satisfy the criterion of:

i. Atomic values across all columns.
ii. No functional dependencies.
iii. No transitive dependencies.

Tables such as Books, while having not even achieved 1NF, are normalised within the query scripts during the creation of select views. Thus, the current state of Books can be described as denormalised as it first underwent normalisation which is how we obtained the Genres and Authors tables, before being denormalised back to having non-atomic values in it's Author_s and Genres columns respectively for the purpose of simplifying querying.

Orders, Shopping_Carts and Wishlist are in the BCNF state; the non-primary key columns (one each in each table) depend only on the primary key of their respective tables.

SQL Implementation

SQL DDL Scripts.

For the FDBMS, there is technically no one schema.sql script; all tables were created using Python. All tables in the database are csv files obtained by converting Pandas data-frames. All IPYNB files used to create the data are in the submitted folder DB Code.

Below is a snippet of the code used to create what inevitably becomes our Payment_Methods table in the database:

```
#Creating our payment method data-frame:
payment_df = {'UserID':user_ids, 'CardNumbers': card_numbers, 'CardType': card_types}
payment_df = pd.DataFrame(payment_df, index = payment_ids)

#Displaying our payment_df:
display(payment_df)

#Saving our data-frame to a csv:
payment_df.to_csv('payment-methods.csv')
```

Figure 3: Code snippet from Tables_Part_02.ipynb.

SQL DML Scripts.

Similarly to the process of creating the tables, the tables were also populated via Python. This is because the aim while filling the tables was to populate them with as much information as possible; thus, the data-collection process was automated, and the Goodreads website was scraped to obtain Book information, their reviews, ratings and more. Below is a snippet of the code used to get the raw, scraped data from Goodreads.

```
#Iterating over 2000 books on Goodreads:
for i in range(1,2000):
  url = f'https://www.goodreads.com/book/show/{i}'
  page = urlopen(url)

  html_bytes = page.read()
  html = html_bytes.decode('utf-8')

  my_soup = BeautifulSoup(html,'html.parser')

  #Obtaining information on the book such as name, author, rating:
  content = my_soup.find_all("script", type = "application/ld+json")

  #Obtaining information on the books genres:
  genres =  my_soup.find_all("span", class_= "BookPageMetadataSection__genreButton")
  genres = [g.getText() for g in genres]

  #Obtaining the book reviews information:
  reviews = my_soup.find_all("section",class_ = "ReviewText__content")
  review_date = my_soup.find_all("section",class_ = "ReviewCard__row")
  reviewers = my_soup.find_all("div",class_ = "ReviewerProfile__name")
  ratings = my_soup.find_all("div",class_ = "ShelfStatus")

  reviews = [r.getText() for r in reviews]
  review_date = [rd.getText() for rd in review_date]
  reviewers = [rv.getText() for rv in reviewers]

  #Taking a break every 200 books to avoid getting blocked by the website:
  if count == 200:
    time.sleep(6)
    count = 0

  if len(content)== 0:
    continue
  book_info += [[content[0].getText(),genres,reviews,review_date,reviewers,ratings]]

```

Figure 4: Code snipped of book info being scraped from Goodreads from Tables_Part_01.

There is however also a DML script submitted separately. Within this, the table dependencies were added via foreign key constraints, certain columns and rows were added or removed and other important factors for building an RDBMS were implemented.

Within this DML script, any errors in the test that may have arisen during the importing process were dealt with; an example of one such error is the common ' error that occurs when data is transferred from one form to another, such as HTML text/tags to text. Furthermore, after forgetting to add

a column that tracks the order date to the order column, the table was updated with this important information here.

SQL Query Scripts.

Within the folder __, you will find the SQL query scripts; there are two, one showcasing queries that are user-based i.e. front-end related queries, while the other script contains back-end focused queries. Within these scripts, views have been created to best answer what are the most important questions the Fable DBMS should be able to answer. Below are some examples.

i. What were the sales like in 2024?

```
Create View Sales2024 As
Select TOP 12 Datename(Month, OrderDate) As Month, Count(*) As
BooksSold, Sum(TotalAmount) As TotalSales
From Orders
Where OrderDate Between '2024' And '2025'
Group By Datename(Month, OrderDate)
Order By Min(OrderDate) Asc;
```

The query sifts through the Orders table and calculates the sum of the number of books sold each month, and the amount made from these sales in 2024; this information is then grouped by month, and ordered in an ascending fashion by date, starting from the beginning of the year and moving month-by-month.

ii. What are some of the most ordered books on Fable?

```
Create View MostOrderedItems As
Select Top 100 ProductID, Sum(Quantity) As Total_Pieces_Orders From
Order_Items
Group By ProductID
Order By Sum(Quantity) Desc;
```

We create a view that stores the ProductID and the quantity of that product sold on the website, grouping the information by ProductID and ordering them by total pieces sold; as it is a view, to use the order by command, we decided to keep the top one hundred rows.

With these queries, business insights can be obtained; for example, knowing which books are the most ordered lets us know which books to keep ordering stock for. Keeping track of which users are not subscribed the Fable membership program as well as user preferences, the website can better target these users by appealing to their interests and so forth.

Implementation Details & System Usage

The Tools.

For the FDBMS project, the following tools were used:

i. SSMS:
   To implement the RDBMS and write all SQL scripts.
ii. EDrawMax:
   To create the ER diagram and Relational Schema for the database.
iii. Google Colaboratory:
   To implement python scripts to scrape the Goodreads website for data.

Challenges Faced.

During the project, there were a few challenges along the way in bringing all the ideas to life.

i. Obtaining the data.

Currently, there are no pre-existing datasets that completely encompass the needs of the Fable database. Thus, the data for the database was obtained from the Goodreads website, one of the worlds largest websites for everything books. This was done via web-scraping, which involved tediously scouring through the pages source code for the right tags and key-words.

Furthermore, due to the large number of books being scraped (the final number of books scraped is 1931), it was a time-taking process, with each execution taking an average of 45 minutes to complete.

By using the BeautifulSoup documentation, following a lot of YouTube tutorials, and being incredibly patient, the process eventually became less and less taxing and more fun.

ii. Implementing constraints.

The aforementioned constraints include both checks and foreign key constraints. Due to the tables being imported into SSMS rather than created there, all constraints were added later within the DML script. It was time-taking to add in the foreign key constraints one-by-one, making sure that the data types between the columns acting as foreign keys had the same data-types and keeping track of the details.

Being focused and taking each constraint one step at a time was important to making sure all constraints and other alter statements were properly implemented without any issues. In the end, the database is now well-linked and the pay-off was worth it in the end.

The Limitations.

The current inventory and book catalogue is not as expansive as it can be, with just 1931 books. Furthermore, when it comes to the orders, shopping carts and wishlists, the data is mostly synthetic; that is, they are not based on any real date, nor do they accurately reflect real users purchasing behaviours. Essentially, while the manner in which we answer questions through our queries is correct, the results are not accurate to what they would be in real life.

Additionally, while the database is well-linked, there are still more constraints that could be added between tables to further strengthen their inter-connectedness, such as exploring how to better link shopping carts and wish lists, whether refunds w

The Future Of The FDBMS.

While it is overall well-rounded and implements most of the initial objectives, there are still a few tidbits that can be added. For one, keeping track of when items were added to the users wishlist and shopping cart can help further track user activity, as well as when users signed up for the website.

Observing relationships such as followings between users can also lead to insights such as which books they are likely to read, what books to recommend to them and more. Users can also follow authors they like; thats more untapped data.

Furthermore, expanding the book catalogue is another important enhancement to consider, as it broadens the available genres, author pool and other related information stores.

And finally, implementing and examining all possible relationships within the entities currently in the database is the next step, and on a similar note, adding more entities that are relevant to the system to further enhance its capabilities, for both the user and the site owners experience.

CONCLUSION

In conclusion, the Fable Database Management System has been designed to efficiently support the operational needs of Fable through a well-structured relational schema. The design process prioritized data integrity, minimal redundancy, and future scalability. With up to Third Normal Form (3NF) and, in some cases, BCNF, key entities such as Users, Books, Orders, and shopping carts are represented in a way that ensures consistency and gets rid of anomalies.

Overall, FDBMS is a reliable, maintainable, and extensible system. It serves as a strong foundation for managing the dynamic and data-intensive requirements

of Fable, while also remaining flexible enough to adapt to future enhancements and business growth, enhancements that I believe would bring the website to new frontiers.