COMP 3005A Final Project (Fall 2021)

By: Hanna A. Kebedom (101143055)

Note: The image files for all schemas/diagrams displayed in this report will be made available in the project's GitHub Repository

2.1 Conceptual Design

To construct a database for the Look Inna Book bookstore application, we must critically examine the business requirements of the system. We will start this process by identifying the entities in the problem space and the relationships that exist between these entities. An entity can be defined as a "thing" or "object" in the real world that is distinguishable from all other objects. The entities necessary for the bookstore application, and the information we need about these entities are as follows:

Figure 1.1 - Entities

Book ● isbn ● title ● description ● pages ● price ● year_published ● quantity	**Publisher** ● publisher_id ● name ● address ● email ● phone_number	**Customer** ● customer_id ● name ● address ● card_number ● email ● phone
Owner ● owner_id ● name ● address	**Account** ● username ● password ● type (customer or owner)	**Order** ● order_id ● date ● status
Store ● store_id ● name ● address	**Warehouse** ● number ● address ● phone	**Author** ● author_id ● name ● bio
Cart ● cart_id	**Genre** ● name	**Bank Account** ● account_number ● balance

Next, we examine the relationships between the entities identified above. Listed below are the cardinalities and participation types of the relationships between the above entities:

Figure 1.2 - Relationships

Cardinalities
Customer/Owner An customer/owner can own multiple accounts, but each account is owned by a single customer/owner (**one-to-many**)
Cart A cart contains many books. A book can be placed in many carts. (**many-to-many**)
Order An order can be placed by one account. An account can place many orders. (**many-to-one**)
Book A book can have many genres. A genre can be assigned to many books. (**many-to-many**) A book can have many publishers. A publisher can publish many books. (**many-to-many**) A book can be stored in many warehouses/stores. A store/warehouse can store many books. (**many-to-many**) A book can be placed in many carts. Many carts can contain a specific book. (**many-to-many**)
Genre A genre describes many books. A book can be of many genres. (**many-to-many**)
Account An account manages one cart. A cart is managed by one account. (**one-to-one**) An account can place many orders. An order is placed by one account. (**one-to-many**)
Store A store stores many books. A book can be stored in many stores. (**many-to-many**)
Warehouse A warehouse stores many books. A book can be stored in many warehouses. (**many-to-many**)
Author An author writes many books. A book can be written by many authors. (**many-to-many**)
Publisher A publisher may publish many books. A book may be published by many publishers. (**many-to-many**) A publisher has a single bank account. A bank account may belong to a single publisher. (**one-to-one**)

Bank Account
A publisher has a single bank account. A bank account may belong to a single publisher. (**one-to-one**)

Participation Types

All accounts must be owned by a customer/owner
All carts must be managed by an account
All orders must be placed by an account
All books must be written by at least one author
All books must be published by at least one publisher

Using the information that we have gathered about the entities in this problem space and the relationships that exist between them we can create an Entity-Relationship (ER) Diagram. The ER diagram for the bookstore application is as follows:

Figure 1.3 - ER Diagram

View Bookstore ER Diagram

2.2 Reduction to Relation Schemas

In this section we will reduce our ER diagram into relation schemas. We can turn both our entity sets and relationship sets into relations that will eventually make up our database. For strong entity sets we use the primary key of the entity set to determine the primary key of the relation. All other attributes of an entity make up the other columns of the relation. The relations that result from strong entity sets in the bookstore database are as follows:

Figure 1.4 - Reduction to Relation Schemas (Strong Entities)

Relations of Strong Entities

- *owner(owner_id, name, address)*
- *customer(customer_id*, name, address, card_number, email, phone*)*
- *orders(order_id, date, status)*
- *account(username, password, type)*
- *cart(cart_id)*
- *book(isbn, title, description, pages, price, year_published)*
- *genre(name)*
- *author(author_id, name, bio)*
- *publisher(publisher_id, name, address, email, phone)*
- *bank_account(account_number, balance)*
- *warehouse(number, address, phone)*

- *store(store_id, name, address)*

We must also create relations from our relationship sets. We create a relation for the relationship that exists between two entities. The primary key of this new relationship set is the union of the primary keys of the two entities participating in the relationship. The relations that result from this are:

Figure 1.5 - Reduction to Relation Schemas (Relationships)

Relations of Relationships

- *owner_account(username, owner_id)*
- *customer_account(username, customer_id)*
- *places(order_id, username)*
- *manages(cart_id, username)*
- *contains(cart_id, isbn, quantity)*
- *writes(author_id,* isbn*)*
- *publishes(publisher_id,* isbn*)*
- *warehouse_book(*number*, isbn, quantity)*
- *store_book(store_id, isbn, quantity)*
- *book_genre(isbn, genre)*

2.3 Normalization of Relation Schemas

We will now normalize our database to simplify tables and minimize redundancy. We will start by identifying the functional dependencies in our schemas. A functional dependency X -> Y holds if each X value is associated with precisely one Y value. The following are the functional dependencies for the relations above:

Figure 1.6 - Functional Dependencies

Functional Dependencies for all Database Relations

owner(owner_id, name, address)
- owner_id -> name, address

customer(customer_id, name, address, card_number, email, phone*)*
- customer_id -> name, address, card_number, email, phone

order(order_id, date, status)
- order_id -> date, status

account(username, password, type)
- *username -> password, type*

cart(cart_id)

book(isbn, title, description, pages, price, year_published)
- *isbn -> title, description, pages, price, year_published*

genre(name)

author(author_id, name, bio)
- *author_id -> name, bio*

publisher(publisher_id, name, address, email, phone)
- *publisher_id -> name, address, email, phone*

bank_account(account_number, balance)
- *account_number -> balance*

warehouse(number, address, phone)
- *number -> address, phone*

store(store_id, name, address)
- *store_id -> name, address*

owner_account(username, owner_id)
- *username -> owner_id*

customer_account(username, customer_id)
- *username -> customer_id*

places(order_id, customer_id)
- *order_id -> customer_id*

manages(cart_id, customer_id)
- *cart_id -> customer_id*

contains(cart_id, isbn, quantity)
- *cart_id, isbn -> quantity*

writes(author_id, isbn*)*

publishes(publisher_id, isbn*)*

warehouse_book(number, isbn, quantity)
- number, isbn -> quantity

store_book(store_id, isbn, quantity)
- *store_id, isbn -> quantity*

book_genre(isbn, genre)

Our relations must be in Third Normal Form(3NF) and Boyce-Codd Normal Form (BCNF) for them to be considered in a good normal form. Using the functional dependencies listed above, we can test to see if these functional dependencies satisfy the requirements of 3NF & BCNF.

According to the course textbook (*Database System Concepts, Seventh Edition*), the requirements for 3NF and BCNF are as follows:

For a relation to be in **3NF**, every functional dependency **X -> Y** must meet one of the following requirements:
1. **X -> Y** is a trivial functional dependency
2. **X** is a superkey for **R**
3. Each attribute **A** in **Y - X** is contained in the candidate key for **R**

For a relation to be in **BCNF**, every functional dependency **X -> Y** must meet one of the following requirements:
1. **X -> Y** is a trivial functional dependency
2. **X** is a superkey for **R**

Below we test to see if the relations satisfy the requirements above:

Figure 1.7 - 3NF & BCNF Tests

BCNF and 3NF Tests
owner(owner_id, name, address) ● owner_id -> name, address The relation owner is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R
customer(customer_id, name, address, card_number, email, phone*)* ● customer_id -> name, address, card_number, email, phone

The relation customer is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R
order(order_id, date, status) ● order_id -> date, status The relation order is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R
account(username, password, type) ● *username -> password, type* The relation account is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.
book(isbn, title, description, pages, price, year_published) ● *isbn -> title, description, pages, price, year_published* The relation book is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.
author(author_id, name, bio) ● *author_id -> name, bio* The relation author is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.
publisher(publisher_id, name, address, email, phone) ● *publisher_id -> name, address, email, phone* The relation publisher is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.
bank_account(account_number, balance) ● *account_number -> balance* The relation bank_account is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.
warehouse(number, address, phone) ● *number -> address, phone* The relation warehouse is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.
store(store_id, name, address)

- *store_id -> name, address*

The relation store is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.

owner_account(username, owner_id)

- *username -> owner_id*

The relation owner_account is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.

customer_account(username, customer_id)

- *username -> customer_id*

The relation customer_account is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.

places(order_id, customer_id)

- *order_id -> customer_id*

The relation places is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.

manages(cart_id, customer_id)

- *cart_id -> customer_id*

The relation manages is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.

contains(cart_id, isbn, quantity)

- *cart_id, isbn -> quantity*

The relation contains is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.

warehouse_book(number, isbn, quantity)

- number, isbn -> quantity

The relation warehouse_book is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.

store_book(store_id, isbn, quantity)

- *store_id, isbn -> quantity*

The relation store_book is in 3NF & BCNF as the left hand side of the functional dependency is a super key for schema R.

The above tests show that the relation in our database are in good normal form and do not need to be decomposed further.

2.4 Database Schema Diagram

The final bookstore database schema diagram is linked below.

Figure 1.7 - Schema Diagram

View bookstore schema diagram

2.5 Implementation

I implemented the bookstore application as a command-line application with ruby as my main programming language, and postgresql as my relational database of choice. I used the ruby gem pg to interface with the relational database. The use of this gem allowed me easily execute SQL queries based on user input.

I designed my application using the model-view-controller (MVC) architecture pattern. My application consists of a model class (that handles all interactions with the database), a view class (which displays the appropriate output to the user) and a controller class (that controls the flow of the application and calls the model and view classes when necessary). This architecture pattern minimizes repetition and makes application code easy to understand and maintain.

Figure 1.8 - Model View Controller Architecture Pattern

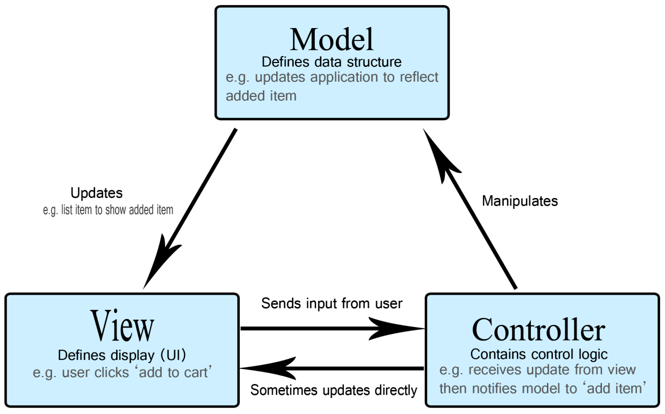
Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

Image Source: https://developer.mozilla.org/en-US/docs/Glossary/MVC

2.6 Bonus Features

Approximate Search

My bookstore does not only match exact strings in search results but also returns any books that contain your search query as a substring. This allows as many relevant results as possible to get to the end user.

Cart Persistence

A user's cart does not get cleared until they checkout. This means anyone with an account can add items to a cart, leave the app, and come back to have their cart items still there. This feature will be immensely valuable to store owners.

2.7 Github Repository

https://github.com/hannakebedom/LookInnaBook/

2.8 Appendix I

Availability for a demo:

9:00am
9:20am
9:40am