Huy Nguyen and Rebekah Koon

Hawkins Public Library

CS 340

Summary

Library database for Hawkins Public Library that records the information of library customers, books present at the library, and upcoming events.  
Link to website: https://hawkins-library.now.sh

Table of Contents

[Design Changes 2](#_Toc33444228)

[Upgrades from Step 3 Final to Step 4 Draft 2](#_Toc33444229)

[Actions Based on Feedback—Step 4 Draft 2](#_Toc33444230)

[Upgrades from Step 3 Draft to Step 3 Final 2](#_Toc33444231)

[Actions Based on Feedback—Step 3 Final 2](#_Toc33444232)

[Upgrades from Step 3 Draft to Step 3 Final 3](#_Toc33444233)

[Actions Based on Feedback—Step 3 Draft 3](#_Toc33444234)

[Upgrades from Step 2 to Step 3 Draft 3](#_Toc33444235)

[Actions Based on Feedback—Step 2 4](#_Toc33444236)

[Upgrades from Step 1 to Step 2 4](#_Toc33444237)

[Overview 5](#_Toc33444238)

[Outline 6](#_Toc33444239)

[customers 6](#_Toc33444240)

[books 6](#_Toc33444241)

[checkoutOrders 6](#_Toc33444242)

[events 7](#_Toc33444243)

[eventRegistrations 7](#_Toc33444244)

[Feedback from Peer Reviewers—Step 3 8](#_Toc33444245)

[Feedback from Peer Reviewers—Step 2 10](#_Toc33444246)

[Entity-Relationship Diagram 13](#_Toc33444247)

[Schema 14](#_Toc33444248)

# Design Changes

## Upgrades from Step 3 Final to Step 4 Draft

* We initially used bootstrap to style our pages. After looking at other projects and seeing that a lot of groups also used bootstrap, we felt like we needed to stand out. Huy had experience with React and web development prior to enrolling in this program, so we decided to take advantage of that. We decided to overhaul our website using React, only keeping our library concept. Our new tech stack uses React on the frontend and Next.js to handle routing.
* We also added our ddl.sql and dml.sql files that include code for creating tables; insert and select queries for all tables; delete queries for events, customers, and events; and update queries for events and books.
* Our website now displays books, customers, library events, checkouts, and event registrations using data from our database.
* Implemented database Read functionality for all tables.
* Implemented database Create functionality for books and events.
* We decided to change the phone attribute in customers from an int to a varchar. This is because we were hitting the maximum value for integers with some phone numbers, so we decided to solve this by changing the data type to a varchar.
* We added an imgUrl attribute to the books entity so that we could add pictures of the books to our website’s library page.
* We added imgUrl and description attributes to the events entity. This allows us to add images as well as descriptions for events on our website’s events page.
* We changed the email attribute in customers in order to make it unique to each customer.
* Changed events table attributes:
  + guest and description are no longer required attributes

## Actions Based on Feedback—Step 4 Draft

* We did not receive feedback from our grading TA for our Step 3 Final Version.

## Upgrades from Step 3 Draft to Step 3 Final

* We decided to change the names of the foreign keys in each table, so orderNumber in Books became oid, customerID in CheckoutOrders became cid, cusomerID in EventRegistrations became cid, and eventID in EventRegistrations became eid.
* We also decided to uncapitalize the names of the entities to reflect what we named our tables.
* In the ID attributes for each entity, we changed ID to Id. We also changed orderNumber to orderId.

## Actions Based on Feedback—Step 3 Final

* We received peer feedback that we did not have delete and update functionality. This was already implemented on the admin > manage books page as well as the admin > manage events.
* Peer feedback also suggested that we should remove the tables in the Library and Return Books pages. Due to time constraints, we decided to leave the tables as-is for now. In the future, we plan on creating a new website with a friendlier user interface that gets rid of the tables on the user side of the website. However, from an updating, deleting, and adding standpoint, it still makes sense for the admin to be able to view everything in a table. Therefore, everything in the admin page will be displayed as such.
* We did not receive any grader feedback at the time of turning in our step 3 final version.

## Upgrades from Step 3 Draft to Step 3 Final

* We decided to add add-books.html, customers.html, and delete-events.html to our website. This will cover the missing insert and select functionalities as well as the delete for the many-to-many relationship. We also added links to these pages to the navigation bar.
* We originally had a “Manage events” page that dealt with adding events, but we decided to rename this to “Add an event” to better match what the page will be used for.
* Since we created the add-books.html page, we also got rid of the “Add a book” form on the “Manage books” page.

## Actions Based on Feedback—Step 3 Draft

* We decided to change the presentation of our file based on feedback from our grading TA. We added a cover page and table of contents in addition to adding headers for each section. We also put each section on a new page in order to make the document more presentable.
* We added rationale behind our changes in order to explain why we changed certain parts of our project.

## Upgrades from Step 2 to Step 3 Draft

* We decided to remove the Authors entity and add the Events entity. We felt that an Authors entity was not necessary for our database because it could simply be represented as an attribute in our Books entity. Libraries often have events such as author meet-and-greets and read-alouds for children, so adding an Events entity would allow us to keep track of upcoming events held at the library. We updated our ERD and schema to reflect this change.
* The Authors entity we removed had a many-to-many relationship with Books, so we needed to create a new many-to-many relationship. We decided to build a many-to-many relationship between Customers and Events in order to fulfill the many-to-many relationship requirement.
* Because we made changes to our entities, we changed our overview to reflect these changes.
* We created our website and added the following pages: add-events.html, admin.html, checkout-orders.html, events.html, index.html, join.html, library.html, manage-books.html, return.html, and view-event-registration.html. We also created style.css to add style to our webpage.

## Actions Based on Feedback—Step 2

* We received feedback from two reviewers that we needed a many-to-many relationship. We already had a many-to-many relationship in our original draft, so we decided not to add a second one. However, we did decide to change the many-to-many relationship as described in the following section.
* We received feedback to remove the Employees entity because it was essentially the same as the Customers entity. Therefore, we removed all attributes from other entities associated with Employees.
* We also decided to remove the Audiobooks entity based on feedback received due to its similarities with Books. We removed all attributes from other entities associated with Audiobooks as well as removed the AudiobookOrders relationship entity.

## Upgrades from Step 1 to Step 2

* Because we needed a new many-to-many relationship after removing the Audiobooks entity, we added the Authors entity to serve as a many-to-many relationship with Books. We felt that an Authors entity made sense in that it would enable us to keep track of each author within the library as well as the books they have written. With this addition, we needed to add a BooksAuthors relationship entity. We also updated our outline, ERD, and schema to reflect this addition.
* We changed the name of the city library, the population, number of books in the library, the average number of visitors per day, and the average number of books checked out. We originally had data associated with a larger city, but we decided to make the city smaller in order to simplify our database and the number of books held in the library. We also did not like the original name we picked for our city, so we decided to change its name. Therefore, our library changed from “Glenwood Public Library” to “Hawkins Public Library.”

# Overview

We will be creating a website that contains a database for Hawkins Public Library. The fictional city of Hawkins has a population of 30,000 people. With 100,000 Books available for checkout, the Hawkins Public Library has a wide variety of genres to choose from. An average of 150 visitors stop by the library each day and approximately 250 Books are checked out daily. Because of the large number of Books within the library, a database will be essential in order to store their information. The Books entity will be used to store the information of each book. The database will also be used to access information related to Customers of the library through the Customers entity. Customers can check out multiple Books in their CheckoutOrders. They can also have several CheckoutOrders, so having a database will be important in tracking each of the CheckoutOrders currently open. Additionally, events will be held at the library, which will be represented with the Events entity. Events include author meet and greets, book clubs, and children’s read-alouds. Customers will be able to sign up for multiple Events, and Events will be able to hold several Customers. In this way, the information stored within the database will allow the Hawkins Public Library to determine which Books have been checked, track the CheckoutOrders of Customers, and keep track of Events being held at the library.

# Outline

customers entity: Records the information of library Customers

* customerId: int, auto\_increment, unique, not NULL, primary key
* firstName: varchar, not NULL
* lastName: varchar, not NULL
* email: varchar, unique, not NULL
* phone: varchar, not NULL
* dateJoined: date, not NULL
* lateFee: int, not NULL; $1 will be added for each day a book is overdue. lateFee can have a value of 0 or more
* One-to-many relationship between the Customers and CheckoutOrders entities: Customers can have multiple CheckoutOrders, but each of the CheckoutOrders is only associated with at most one of the Customers. Customers can have 0 or more CheckoutOrders. customerID will be implemented as a foreign key within the CheckoutOrders entity.
* Many-to-many relationship between the Events and Customers entities: Customers can sign up for multiple Events and Events can have multiple Customers.

books entity: Records the information of library Books

* bookId: int, auto\_increment, unique, not NULL, primary key
* title: varchar, not NULL
* author: varchar, not NULL
* publisher: varchar, not NULL
* genre: varchar, not NULL
* imgUrl: varchar
* oid: int, foreign key from CheckoutOrders entity; used to keep track of what order the book belongs to. It will be NULL if the book does not belong to an order.
* One-to-many relationship between the Books and CheckoutOrders entities: CheckoutOrders can contain 0 or more Books, but each of the Books can only be in at most one of the CheckoutOrders. orderNumber from CheckoutOrders will be implemented as a foreign key within Books to track which of the CheckoutOrders it belongs to.

checkoutOrders entity: Records the checkout order of a Customer. CheckoutOrders can contain multiple Books and are associated with Customers.

* orderId: int, auto\_increment, unique, not NULL, primary key; used to identify a specific order
* checkoutDate: date, not NULL
* dueDate: date, not NULL
* cid: int, foreign key from Customers entity. Represents the ID of the customer who made the checkout order.
* One-to-many relationship between the Customers and CheckoutOrders entities: Customers can have multiple CheckoutOrders, but each of the CheckoutOrders is only associated with at most one of the Customers. Customers can have 0 or more CheckoutOrders. customerID will be implemented as a foreign key within the CheckoutOrders entity.
* One-to-many relationship between the Books and CheckoutOrders entities: CheckoutOrders can contain 0 or more Books, but each of the Books can only be in at most one of the CheckoutOrders. orderNumber from CheckoutOrders will be implemented as a foreign key within Books to track which of the CheckoutOrders it belongs to.

eventsentity: Records upcoming library events

* eventId: int, auto\_increment, unique, not NULL, primary key
* name: varchar, not NULL
* date: date, not NULL
* guest: varchar
* description: varchar
* imgUrl: varchar
* Many-to-many relationship between the Events and Customers entities: Customers can sign up for multiple Events and Events can have multiple Customers.

eventRegistrationsentity: Represents the relationship between Events and Customers

* registrationId: int, auto\_increment, unique, not NULL, primary key
* cid: int, not NULL, foreign key from the Customers entity
* eid: int, not NULL, foreign key from the Events entity

# Feedback from Peer Reviewers—Step 3

**Does the UI utilize a SELECT for every table in the schema? In other words, data from each table in the schema should be displayed on the UI. Note: it is generally not acceptable for just a single query to join all tables and displays them.**

**Karen Berba:** Yes, the UI utilizes a SELECT for every table in the schema (Books, CheckoutOrders, Customers, EventRegistrations, Events).

**Amy:** Yes, it does. It shows a select for checkout orders, event registrations, events, books, and customers

**Bowen Lin:** I think the UI contains data for each table, which is pretty comprehensive

**Does at least one SELECT utilize a search/filter with a dynamically populated list of properties?**

**Karen Berba:** Yes, there are 3 SELECT functions that utilize a search/filter with a dynamically populated list of properties.

**Amy:** Yes, under upcoming events

**Bowen Lin:** Yes, the web page contains eight lists to choose from.

**Does the UI implement an INSERT for every table in the schema? In other words, there should be UI input fields that correspond to each table and attribute in that table.**

**Karen Berba:** Yes, it does seem like the UI implements an INSERT for every table in the schema.

**Amy:** Yes, there's an insert query for every table

**Bowen Lin:** Yes, you can enter each of the corresponding properties in the UI framework.

**Does each INSERT also add the corresponding FK attributes, including at least one M:M relationship? In other words if there is a M:M relationship between Orders and Products, INSERTing a new Order (e.g. orderID, customerID, date, total), should also INSERT row(s) in the intersection table, e.g. OrderDetails (orderID, productID, qty, price and line\_total).**

**Karen Berba:** Yes, each INSERT function adds the corresponding FK attributes. And yes, the INSERT function adds the corresponding FK attributes to at least one M:M relationship (between the Events and Customers entities).

**Amy:** Yes, each insert add the corresponding FK attributes, and at least one M:M relationship

**Bowen Lin:** Yes, the project includes m:m relationships, such as libraries and books

**Is there at least one DELETE and does at least one DELETE remove things from a M:M relationship? In other words, if an order is deleted from the Orders table, it should also delete the corresponding rows from the OrderDetails table, BUT it should not delete any Products or Customers.**

**Karen Berba:** No, there doesn't seem to be a DELETE function.

**Amy:** have not seen a delete function

**Bowen Lin:** I did not find a place to delete, the web page can only upload data.

**Is there at least one UPDATE for any one entity? In other words, in the case of Products, can productName, listPrice, qtyOnHand, e.g. be updated for a single ProductID record?**

**Karen Berba:** From the UI side, there doesn't seem to be an UPDATE function, but it's possible it's implemented in the back-end.

**Amy:** I can't find one that allows this in the UI

**Bowen Lin:** Yes, the product can be updated

**Is at least one relationship NULLable? In other words, there should be at least one optional relationship, e.g. having an Employee might be optional for any Order. Thus it should be feasible to edit an Order and change the value of Employee to be empty.**

**Karen Berba:** Yes, there is at least one relationship that is NULLable -- the orderNumber in the Books entity can be NULL if the book does not belong to an order.

**Amy:** Yes, the order number is nullable

**Bowen Lin:** Yes, the page contains at least one relationship which can be NULLLable

**Do you have any other suggestions for the team to help with their HTML UI?**

**Karen Berba:** First of all, I want to say that the site is beautiful! I don't have any suggestions because the site seems easy and intuitive for users to navigate and use. Great job!

**Amy:** I think it's good! I am not totally sure why you're essentially displaying your database data in the UI, though. You should probably remove what is essentially every table as a display, and just make it something like "add book to checkout", list of books (or empty field), and user entry. It's a little overwhelming and confusing. Just try to think "what information do I need to know in my database, but the user doesn't necessarily need to know to perform this action or get information?" and go from there

**Bowen Lin:** I think this web page is designed to be very perfect, very much like a lending site that we can use in our lives. I think it would be nice to just add the delete button.

# Feedback from Peer Reviewers—Step 2

**Does the overview describe what problem is to be solved by a website with DB back end?**

**Ryan Lew:** Did not provide an answer.

**Anish Reddy:** Yes, the overview does describe the problem to be solved with a website and DB back end. They are trying to solve the problem of having a lot of books with a lot of people within a city at Glenwood Public Library.

**Karen Berba:** Yes, the overview clearly describes the problem to be solved by a website with a DB back end, which is to keep track of the information about the current selection of book / audiobook (e.g. availability / checked-out status, CheckoutOrders made by Customers and Employees)

**Rebecca:** Yes, the overview describes the problem to be solved and it’s back end portion of the website.

**Does the overview list specific facts?**

**Ryan Lew:** Yes, overview lists specifics about book numbers, customer numbers and usage.

**Anish Reddy:** Yes, the overview does list specific facts. It lists the amount of books available in the library, being 300,000, with the amount of people in the city being 200,000. They also list that there will be 2000 visitors each day, and approximately 2500 books and audiobooks are checked out daily.

**Karen Berba:** Yes, the overview lists specific facts about the population of the city, the number of books and audiobooks available, the average number of visitors, and the approximate number of books and audiobooks checked out each day

**Rebecca:** Yes, there are specific facts listing within the overview in relation to the entities and relationships.

**Are at least four entities described and does each one represent a single idea to be stored as a list?**

**Ryan Lew:** Yes. Lists entities, their relations and purposes.

**Anish Reddy:** Yes, there are 5 entities: Employees, Customers, Books, Audiobooks, CheckoutOrders. Maybe you don't need 5, you could cut out audiobooks to make the database easier to implement. Each entity does represent a single idea to be stored as a list.

**Karen Berba:** Yes, there are five entities (Customers, Employees, Books, Audiobooks, CheckoutOrders), and each one represents a single idea to be stored in a list

**Rebecca:** Yes, there are at least four entities. In this case there are more of which are represented as a single idea to be stored as a list.

**Does the outline of entity details describe the purpose of each, list attribute datatypes and constraints and describe relationships between entities?**

**Ryan Lew:** Yes. All details and relationships are defined and rationalized.

**Anish Reddy:** Yes, the outline does describe the list attribute datatypes and constraints and it does describe relationship between entities.

**Karen Berba:** Yes, each entity outline includes the purpose of the entity, the attribute datatypes and constraints, as well as clearly describes the relationships between the entities

**Rebecca:** Yes, the outline of the entity’s details does describe the purpose of each. They do list attribute datatypes. They do have constraints. They do describe their relationship between entities.

**Are 1:M relationships correctly formulated? Is there at least one M:M relationship?**

**Ryan Lew:** There is no many to many relationship, but the 1:M relationships are formulated correctly.

**Anish Reddy:** I did notice one problem with the relationship between CheckoutOrders and Employees. I understand that each employee can have 0 to many CheckoutOrders, but how can each CheckoutOrder have 0 employees? Wouldn't each CheckoutOrder need at least/most 1, and can't have 0? Maybe I am confusing myself here, but that is what I thought when I looked at the ER diagram. I think all the other relationships are correct, and there is one M:M relationship between Audiobooks and CheckoutOrders, which is formulated correctly.

**Karen Berba:** Yes, the 1:M relationships are correctly formulated. And yes, there is a M:M relationship between Audiobooks and CheckoutOrders. (The only question I have is: are Employees essentially the same as Customers, in that that they just check out books to read? Or, for each Customer that checks out a book, does an Employee need to be associated with processing that book check-out?)

**Rebecca:** Yes, there is a 1:M relationship which is correctly formulated. No, there is not least one M:M relationship within the current diagram.

**Is there consistency in a) naming between overview and entity/attributes b) entities plural, attributes singular c) use of capitalization for naming?**

**Ryan Lew:** Yes, across multiple entities naming schemes follow a pattern and is easy to understand.

**Anish Reddy:** Yes, entities are plural, and attributes are singular. The name is consistent between the overview and entity/attributes. Capitalization is used as the second word in an attribute name is capitalized and is consistent throughout.

**Karen Berba:** Yes, there is consistency in naming between overview and entity/attributes; the entities are all plural, while the attributes are all singular; there is also use of capitalization for naming

**Rebecca:** a. Yes, there is consistence in entity/attributes. With entities being plural and attributes singular.

b. Yes, entities are plural, and attributes are singular.

c. Yes, camel case style is used and naming is properly capitalized.

# Entity-Relationship Diagram

A close up of a sign

Description automatically generated

# Schema

A screenshot of a cell phone

Description automatically generated