Documentation on a Database-backed Web Application for an Online Bookstore: Design and Technological Survey

**By Carson Forsyth**

For CMPSCI 431W @ PSU with Professor Dong Xie, SP 2021

# Table of Contents

Table of Contents 1

List of Figures 5

List of Tables 5

1. Introduction 6

Tasks in Phase 1 6

Task 1 6

Task 2 6

Task 3 6

Task 4 6

Tasks in Phase 2 and timing 6

1. Database Population 6

2. Functionality Implementation 6

3. Web GUI 6

4. Video Presentation / Reflection 6

5. Submission of Deliverables 7

2. Requirement Analysis 8

2.1. Analysis of Application Users 8

Public 8

User 8

Customer 8

Employee 8

Manager 8

2.2. Required Functionality 8

Customer Registration 8

Multiple Managers 8

Ordering 8

New book 8

Arrival of more copies 9

Comments 9

Usefulness ratings 9

Trust recordings 9

Book browsing 9

Useful Comments 9

Buying suggestions 9

Degrees of separation 10

Book Statistics 10

User awards 10

2.3. Proposed Functionality Implementations 10

Images 10

Sales 10

Coupons 10

Hold Employees Accountable 11

Outstanding Orders 11

Secondary Addresses 11

International Shipping 11

Suggest Similar Books 11

Show Trusted Commentors 11

Report Low Stock 11

Find Popular Destinations 13

Track Hard Workers 13

Allow Customer Deletion 13

Book Score 13

Hide Untrustworthy Users 13

3. Conceptual design 14

3.1. Entire ER diagram 14

3.2. Generalized ER diagram 16

3.3. Modules of the Database 17

User Module 17

User 17

Customer 18

Employee 18

Manager 18

Book Module 19

Author 19

Book 20

Sale 21

Comment Module 21

Comment 22

Rating 22

Ordering Module 23

Address 23

Order Item 24

Order 24

Coupon 26

3.4. Aggregations 27

Comment Rating 27

Customer Usefulness 27

Orders of Books 27

Orders to Addresses 27

4. Technological Survey 28

4.1. Database Management System Recommendations 28

4.2. Web Frontend Recommendations 29

4.3. Backend Recommendations 30

4.4. Final Recommendation 32

5. Logical database design and Boyce-Codd Normalization 33

5.1. User Module 33

User, Employee and Manager 33

Customer 33

5.2. Book Module 34

Author 34

Book 34

Sale 36

5.3. Comment Module 37

Comment 37

Rating 37

5.4. Ordering Module 37

Address 37

Order 37

Order Item 38

Coupon 38

5.5. Aggregations 39

Comment Rating 39

Customer Usefulness 39

Orders of Books 39

Orders to Addresses 40

Works Cited 41

# List of Figures

Figure 1 - Complete System ER Diagram 11

Figure 2 - Generalized ER Diagram with Aggregations 12

Figure 3 - Generalized ER Diagram without Aggregations 12

Figure 4 - Users Module of ER Diagram 14

Figure 5 - Book Module of ER Diagram 16

Figure 6 - Comment Module of ER Diagram 17

Figure 7 - Order Module of ER Diagram 20

Figure 8 - Comments Aggregations from ER Diagram 21

Figure 9 - Orders Aggregations from ER Diagram 21

# List of Tables

Table 1 - March 2021 Top Database System Rankings from DB-Engines 23

Table 2 - All Available Storage Classes from SQLite 24

# Introduction

## Tasks in Phase 1

Task 1 – The initial task, *Requirement Analysis*, intends to establish an abstract plan for possible implementations of requested services. The functionalities laid out in the overall requirements for this database must be analyzed in terms of group using the service, data required, data expected, and the effect the function will have in the database. Past the given minimal requirements, this analysis identifies needed information and some complicated parts of the system to implement.

Task 2 – *Conceptual Database Design* will elaborate the constraints and required data greatly. The section provides and walks through the entity-relationship (ER) diagram to rudimentarily model the system and its features and restrictions. The overall ER diagram is split up into a generalized form, and then more importantly the individual modules of the system. The modules Books, Users, and Comments are split to correspond to the minimal requirements, Book Data, Customer Data, and Comments; in addition to these 3, the Ordering Module is an obvious additional section to improve organization of Task 2.

Task 3 – To manage and understand current trends the *Technological Survey* requires an in depth look at trends today in web programming. The full stack from database to frontend to backend is examined to look at potential options and the best possible route to take for this system implementation.

Task 4 – *Logical Database Design and Normalization* finalizes the relationships using an SQL data definition language (DDL) to implement the identified constraints and requirements from the previous tasks. In addition, the normalization for schema is explained and justified.

## Tasks in Phase 2 and timing

1. Database Population – For this task, it is required to create and fill in the tables of the database with the appropriate data. While some initial test data is provided, this task will require additional analysis for what to expect in the database, and a scripting to automatically generate the desired tables. Complete by April 6
2. Functionality Implementation – This is the bulk of the work for this system; prototyping will require all functionality listed in this document to be implemented using Flask Web Framework, Python and SQLite. A read me will need to be implemented in this task to allow others to examine the prototype. Complete by April 22
3. Web GUI – This part of the project should not take as long as the backend development. It is vital to the design of a successful system to have a convenient and very comprehensive interface. Complete by April 26
4. Video Presentation / Reflection – A 5-minute video of the system and its functionality is required with reflections. This task is intended to examine changes and why changes had to occur in design, for improvements to future implementations of systems similar like this. Complete by April 28
5. Submission of Deliverables – Post all code, the csv files, the video presentation, and a readme to Canvas in a zip file. Complete by April 29

# 2. Requirement Analysis

## 2.1. Analysis of Application Users

Public – Someone that has not been logged in should still be able to see basic information regarding the webstore, as well as a method to become registered as a site user.

User – Users are clients that have been registered and signed in using their username and password.

Customer – Clients that intend to place an order will have to become a customer. Customers will need to have a phone number and will be able to make orders. Customers will need to be able to browse and make comments and ratings and orders for the website.

Employee – An Employee is a designated user by the managers. An employee may be able to modify stock amounts, or perhaps fulfill orders.

Manager – Managers will have near complete access to the system. A manager is expected to maintain records for stock items and keep the store up to date with new arrivals.

## 2.2. Required Functionality

Customer Registration - To implement registration for a customer:

* Client: Public
* Input: take in the user’s login name, password, first name, last name, and email.
* Output: alert the user that the new account was created successfully.
* Side effects: create a new entry in a table of Users if login name is unique.

Multiple Managers - To implement multiple managers that access full functionality:

* Client: Managers
* Input: the original superuser adds in manager login names to allow access.
* Output: alert the superuser that accounts have been successfully updated.
* Side effects: update the roles of the given users from the login names to have full access.

Ordering - To implement multiple orders of books by a customer:

* Client: Customers
* Input: the user browses and selects the books and quantities they want to order.
* Output: alert the customer that books have been ordered.
* Side effects: create a new order entry relating to this user and each book selected.

New book - To implement new arrivals to the warehouse:

* Client: Manager
* Input: for each new book; its ISBN, publisher, publish date, number of pages, language, price, stock, subject(s), keywords, author(s).
* Output: alert the manager that the book has been added successfully.
* Side effects: add a new entry to the book table, as well as the author(s) if not kept.

Arrival of more copies - To implement stock changes:

* Client: Manager
* Input: the book ISBN and stock to add or remove.
* Output: alert of success and new stock level.
* Side effects: the book entry must have its stock updated in the database.

Comments - To implement comments:

* Client: Customers
* Input: a comment text if desired and a score from 0-10 by a customer.
* Output: direct the customer to the comment.
* Side effects: a new entry will be created in the comments table that can only be modified by the associated customer.

Usefulness ratings - To implement usefulness ratings:

* Client: Users
* Input: the given score for a comment.
* Output: alert success to the user and updated usefulness rating.
* Side effect: create a new rating entry for the specified comment.

Trust recordings - To implement a user trust system:

* Client: Users
* Input: a username of the person to trust or not trust
* Output: alert user on success.
* Side effects: add a new entry to that users’ trust, holding value of trust or distrust in a Boolean.

Book browsing - To implement book browsing on attributes:

* Client: Users
* Input: a search query containing authors, publishes, title words, language, and preferred sort.
* Output: a result set of books containing all matching types from search, sorted as requested.
* Side effects: none.

Useful Comments - To implement usefulness:

* Client: Users
* Input: the number of comments to retrieve.
* Output: a set of comments sorted by usefulness, containing the number of comments desired.
* Side effects: none.

Buying suggestions - To implement suggestions for book buying:

* Client: Customers
* Input: the book A that was ordered.
* Output: return a list of related books sorted by decreasing sales that were also bought by those buying book A.
* Side effects: none.

Degrees of separation - To implement degrees of separation:

* Client: Users
* Input: The author X
* Output: the set of books that are co-authored by X, and the set of books that share a co-author with X.
* Side effects: none.

Book Statistics - To implement book statistics:

* Client: Manager
* Input: none.
* Output: the most popular books, authors, and publishers sold this quarter
* Side effects: none.

User awards - To implement user awards:

* Client: Manager
* Input: how to sort users.
* Output: users with the most trusted entries or with the most useful average comments.
* Side effects: none.

## 2.3. Proposed Functionality Implementations

Images – Images may be beneficial to include to help sell books, so add functionality to add images to a book.

* Client: Manager
* Input: image files to be referenced by the book entry.
* Output: the updated book entry.
* Side effects: the database will need to store the image file and have the book entry reference it.

Sales – Add functionality to discount certain books as prescribed by managers when orders are within the desired time frame.

* Client: Manager
* Input: the date the sale should start and end, the discount, the display name, and the books to include in the sale
* Output: alert success and show the created sale.
* Side effects: a sale entry should be created and stored in the database.

Coupons – Add coupon codes to allow managers to make promotions: while in a certain timeframe, and when the correct code is used, the manager may want a promotion to reduce the price for consumers’ orders.

* Client: Manager
* Input: the date the promotion should start and end, the discount, the display name, and the code needed to be entered to receive the discount.
* Output: alert success and show the created coupon.
* Side effects: a coupon entry should be created and stored in the database.

Hold Employees Accountable – The manager may wish to examine an order to view which employee fulfilled it in case there have been any issues.

* Client: Manager
* Input: Order ID to check.
* Output: The Employee that fulfilled the Order.
* Side effects: none.

Outstanding Orders – Maintain a time fulfilled for all orders and track outstanding orders (those that have not been shipped yet).

* Client: Manager and Employees
* Input: none.
* Output: Orders that have not yet been fulfilled.
* Side effects: none.

Secondary Addresses – Customers may have multiple residencies and wish to not retype them for every order.

* Client: Customer
* Input: Another address for the customer.
* Output: alert user on success.
* Side effects: A new address entry will be added to the address table associated with the customer.

International Shipping – store the country and the region of each address.

* Client: Customer
* Input: A country and region to ship to.
* Output: alert success.
* Side effects: store the country and region in the desired address associated with the customer.

Suggest Similar Books – return a list of books that were most purchased by the last book the user ordered.

* Client: Customer
* Input: the book B that the customer ordered.
* Output: books that were most purchased by customers who have also purchased book B.
* Side effects: none.

Show Trusted Commentors – Give the User a list of comments by trusted users that are on the given book.

* Client: User
* Input: the book B that the customer is examining.
* Output: comments made by trusted users of current user on the desired book
* Side effects: none.

Report Low Stock – The manager should be able to check which books are out of or running low on stock.

* Client: Manager
* Input: none.
* Output: books that have a low number in inventory, sorted by increasing stock
* Side effects: none.

Find Popular Destinations – If the bookstore wishes to expand, tracking cities, regions or countries that commonly order from the store may inform the decision.

* Client: Manager
* Input: Preferred filter for order destinations such as by country or by city.
* Output: The desired destination type, sorted by the number of orders going there, descending.
* Side effects: none.

Track Hard Workers – The manager may want to examine which employees have fulfilled the most orders.

* Client: Manager
* Input: none.
* Output: The employees that have fulfilled some orders, sorted by number of orders descending.
* Side effects: none.

Allow Customer Deletion – The system should let a customer remove their account. However, the system should keep addresses until all orders are fulfilled.

* Client: Customer
* Input: The customer that wants to be deleted.
* Output: alert success to user.
* Side effects: Remove associated user and addresses if and only if no outstanding orders exist.

Book Score – The store should display an average score of a book from all the comments made on it.

* Client: Customer
* Input: The book the customer wishes to view.
* Output: The score of the book, out of 10 as an average rating of the book.
* Side effects: none.

Hide Untrustworthy Users – Comments that are made by customers with very little trust should be hidden on the external book views, while the comments from trusted customers are displayed.

* Client: Customer
* Input: The book the customer wishes to view.
* Output: the comments on that book that were made by users that are very trusted, hiding others.
* Side effects: none.

# 3. Conceptual design

For the design of this system, in this task first the entire ER diagram is given before organizing the diagram into relevant modules. A generalized version exists to abstract attributes away and view the topical organization of the system. Each module is broken down into its entities, which are then further described.

For constraints on the attributes, a required attribute is a property of an entity that may not be null. The entity is described, then data, then the entities relationships within the system. Finally, any additional constraints should be listed.

From one entity A to another B, there may be a relationship mapping the two. If this is a bijection, where all members of A exactly correspond to a single member of B, then A relates to exactly one B. If the relationship is such that A is not required to have a relationship to B, but some A may relate to some members of B, then A relates to none or many B (0+ A may map to 0+ in B). If all members of A must correspond to a B, and A may correspond to many members of B, then A relates to one or many B. Finally, if a mapping from A only exists for a max of one B, then A relates to none or one member of B.

## 3.1. Entire ER diagram

Diagram

Description automatically generated

Figure - Complete System ER Diagram

## 3.2. Generalized ER diagram

Diagram, timeline

Description automatically generated

Figure - Generalized ER Diagram with Aggregations

#### Without Aggregations:

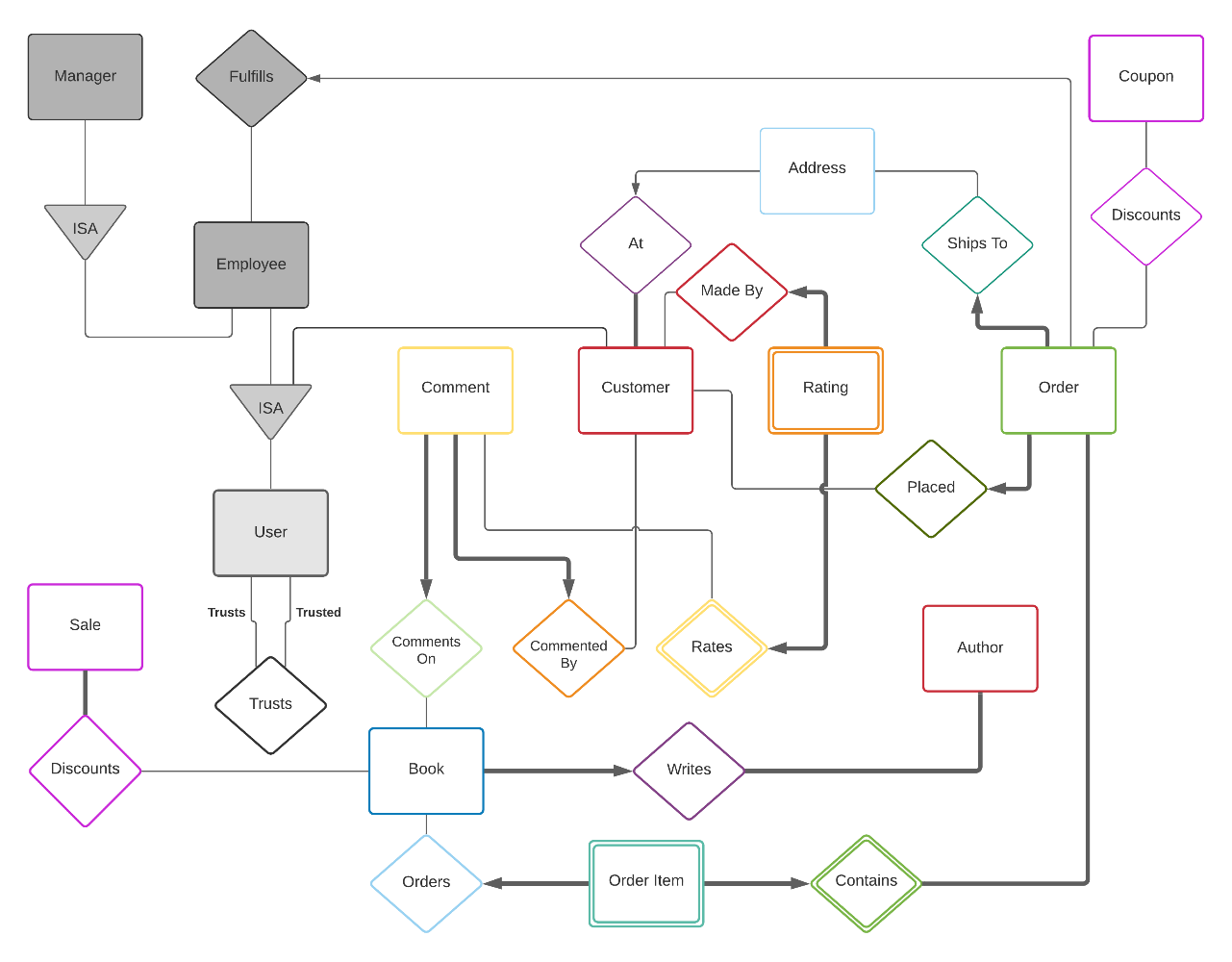


Figure - Generalized ER Diagram without Aggregations

## 3.3. Modules of the Database

## User Module

Diagram

Description automatically generated

Figure - Users Module of ER Diagram

User – Any host that connects and requests more than basic public entity views will be required to login. For a new user, store their email for password recovery and as a method to contact them, and store first name and last name.

* Data attributes needed:
  + Email
    - Required
    - Unique
  + User identification
    - Required
    - Unique
    - Generated by application.
  + Password
    - Required
  + Username
    - Required
    - Unique
* A User can *Trust* other Users. Each User may *Trust* none or many Users, with a Boolean to represent trusts or does not trust.

Generate an ID on creation to identify Users.

Usernames, emails, and user ID must be unique.

Customer – Users will be required to register as a customer entity when they want to order. Customers will inherit basic user permissions, and additionally be able to place an order. Due to internationally shipping, only require a first name.

* Data attributes needed:
  + Phone number
    - Required
  + First name
    - Required
  + Last name
* Customers *Trust* other Customers; customers may have one or many trusted customers.
* Customers may *Comment* on a book.
* A Customer may *Rate* a Comment.

Employee – A User may be registered as an employee by a manager. Employees are assigned an employee ID on creation. An Employee may be specialized into a Manager, in which case Employee *Is A* Manager.

* Employees *Fulfill* Orders. An Employee can *Fulfill* none or many Orders; every Order is *Fulfilled* by zero or one Employee.

Manager – One original manager will be hardcoded into the application. Additional managers may only be added by a manager. A manager may create, modify, or delete a book, inventory item, author, discount, or coupon.

* A Manager *Tracks* the Orders of all Books. Managers award customers with comments that have been rated as very useful, and customers that are trusted most.
* A Manager *Awards* Customers having the most trusts and best ratings.

## Book Module

Diagram

Description automatically generated

Figure - Book Module of ER Diagram

Author – Author information is split from the Book entity to provide easier searching of authors and allow for later author details to be added to the system while also reducing anomalies. Degrees of separation for authors may be implemented using this table. Only a first name is required.

* Data attributes needed:
  + First name
    - Required
  + Last name
  + Author ID
    - Required
    - Unique
    - Generated by application.
* An Author *Writes* a Book. Every Book must be *Written* by one or many Authors, and every Author *Writes* one or many Books.

Book – To implement book browsing, a book object is created to represent, correlate to, and find physical books in the bookstore inventory. These are entity-types that represent a specific inventory item of the store. ISBN for a book is unique, so the rest of the information contained can linked to the ISBN. In each

* Data attributes needed:
  + ISBN
    - Required
    - Unique
  + Title
    - Required
  + Publisher
    - Required
  + Language
    - Required
  + Publication Date
    - Required
  + Number of Pages
    - Required
  + Stock Level
    - Required
    - Default 0
  + Price
    - Required
  + Keywords
    - Multivalued
  + Subject Genre(s)
    - Multivalued
  + Images
    - Multivalued
  + Date added
    - Generated by application.
* A Book may be *Commented On* by a Comments. Every Comment must be made by a different Customer for the same Book. Every Comment *Comments On* a single book. A Book may be *Commented on* by none or many Comments.
* A Book may be *Ordered* by an Order Item. A Book may be *Ordered* none or many times. Every Order *Orders* a single Book.
* Every Book was *Written* by an Author. A Book is *Written* by at least one Author. An Author may *Write* one or many Books.
* A Book may be *Discounted* by a Sale. Every Sale *Discounts* one or many Books.

Generate an ID on creation to identify Books.

Sale – Functionality to create sales on specific books can be created in a new entity.

* Data attributes needed:
  + Name to display
    - Required
  + Discount amount
    - Required
  + Starting date
    - Required
  + End date
    - Required
  + Sale identification
    - Required
    - Unique
    - Generated by application.
* Every Sale *Discounts* one or many Books.

Generate an ID on creation to identify Sales.

## Comment Module

Diagram

Description automatically generated

Figure - Comment Module of ER Diagram

Comment – Comments provide a way for customers to leave a rating on a book, and an optional message for other potential buyers of the book. They should be viewed by other commentors but only modified by the original creator.

* Data attributes needed:
  + Score by comment
    - Required
  + Comment content
  + Time left
    - Required
    - Generated by application.
  + Comment identification
    - Required
    - Unique
    - Generated by application.
* A Comment *Comments On* a Book. Every Comment is *Commented By* exactly one Customer. Customers may *Comment On* none or many Books.
* A Comment is *Rated* by a Rating. A Comment may be *Rated* none or many times. Every Rating must *Rate* a single Comment.

Generate an ID on creation to identify Comments.

Additionally, comments should be restricted to books that the customer has received already.

Rating – A rating may be left on a comment by a customer to demonstrate its usefulness.

* Data attributes needed
  + Value of rating
    - Required
  + Time left
    - Required
    - Generated by application.
* A Rating *Rates* a Comment. Every Rating *Rates* a single Comment. Every Comment may be *Rated* by none or many Ratings.
* A Rating is *Made By* a Customer. Every Rating is *Made By* a single Customer, and a Customer may *Make* none or many Ratings.

A rating should not be allowable on comments the rater created.

## Ordering Module

Diagram

Description automatically generated

Figure - Order Module of ER Diagram

Address – An address is an entity storing all the necessary information to ship a book or send payroll information to the related user. Addresses should be separate from users to allow multiple. Having attributes of address split up will allow for easier querying of location data such as states with the most orders.

* Data attributes needed:
  + Street number
    - Required
  + Street name
    - Required
  + City
    - Required
  + State/region
    - Required
  + Apartment number
  + Zip code
  + Address identification
    - Required
    - Unique
    - Generated by application.
* A Customer is *At* an Address. Every Customer is *At* one or more Addresses. Every Address has none or many Customer *At* itself.
* An Order *Ships To* an Address. Every Order *Ships To* exactly one Address, and an Address may have none or many Orders.

Generate an ID on creation to identify Users.

Order Item – Orders for the bookstore foreseeably will contain copies of the same book. In addition, the manager of the bookstore will likely want to keep a history of income, while also allowing book prices to be updated. Creating a smaller Order Item entity allows for price at time of order to be stored.

* Data attributes needed:
  + Quantity
    - Required
  + Book price at order
    - Required
  + Item identification within the order
    - Required
    - Unique to the Order
    - Generated by application.
* An Order Item *Orders* a Book. Every Order *Orders* exactly one book, and a Book may be *Ordered* none or many times.

Generate an Item ID on creation to help identify the item within the Order.

Order Items will store the price of the book that it orders at that moment. Income can be reproduced from orders using this attribute.

Order – Customers will create orders of specific books to have shipped out to them.

* Data attributes needed:
  + Order identification
    - Required
    - Unique
    - Generated by application.
  + Time of order
    - Required
    - Generated by application.
  + Time of fulfillment
    - Default Null
* An Order may be *Discount* by a Coupon. An Order may be *Discounted* by none or many Coupons. A Coupon may *Discount* none or many Orders.
* Every Order *Contains* one or many Order Items.
* An Order is *Placed* by a Customer. Every Order is *Placed* by zero or one Customer.
* An Order may be *Fulfilled* by zero or one Employees.

Generate an ID on creation to identify Orders.

The total price may be derived attribute from all the Order Items contained within the Order.

Coupon – A new functionality can be implemented by creating Coupon entity-types to discount orders when the correct code is put in by the customer.

* Data attributes needed:
  + Name to display
    - Required
  + Code to input
    - Required
    - Unique
  + Discount amount
    - Required
  + Starting date
    - Required
  + End date
    - Required
  + Coupon identification
    - Required
    - Unique
    - Generated by application.
* Some Coupons may *Discount* none or many Orders. Coupons may only apply to an Order when the current time is within the allowed period.

Generate an ID on creation to identify Coupons.

## 3.4. Aggregations

Diagram

Description automatically generated

Figure - Comments Aggregations from ER Diagram

Diagram

Description automatically generated

Figure - Orders Aggregations from ER Diagram

Comment Rating – Every comment has a Comment Rating that is generated from an aggregation. Comments, Customers, and Ratings may be aggregated into an entity that will be called Comment Rating. This aggregation allows every Comment to be *Rated Useful* by all the Ratings left on that Comment by all Customers.

Customer Usefulness – Managers may *Award* Customers who leave very useful comments and who are very trusted. Aggregating Comment Ratings provides the necessary information, a usefulness rating for Comments and the trust from the Customers entity within.

Orders of Books – Managers may *Track* the number of sales of books each quarter, the authors with the most copies sold, and the publishers with the most copies sold. Within Orders, Order Items, and Books all this information is contained. An aggregation, called Orders of Books of these entities will allow Managers to record the desired statistics.

Orders to Addresses – Managers will want to track the number of orders being shipped to certain regions. An aggregation of Orders, Order Items, Books, and Addresses produces the required information. Then Amounts of orders can be tracked based off country, region, zip code, city, or even street.

# 4. Technological Survey

This section intends to compare existing options to implement the desired system and all its functionality. Trends in web development are examined, as of the current date, April 8, 2021, to analyze differences in frameworks. From the comparison of the full stack, database, frontend, and backend, the best fit for the system requirements will be substantiated by the unique attributes that are explored in the following.

## 4.1. Database Management System Recommendations

An initial survey of the most popular databases today results in the following list of top candidates: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, and MongoDB. Each of these are capable database services capable of handling the desired system especially if intended for a smaller bookstore. Examining trends overtime from DB-Engines’ Trend Popularity graph, each these of these database management systems (DBMS) are trending upwards or steady, perhaps excepting Microsoft SQL Server (solid IT, 2021). From these historical trends, MariaDB also seems to be growing rapidly relative to other services.

Table - March 2021 Top Database System Rankings from DB-Engines

|  |  |  |
| --- | --- | --- |
| **Mar 2021 Rank** | **DBMS** | **Database Model** |
| 1. | Oracle, 1979 | Relational, Multi-model |
| 2. | MySQL, 1995 | Relational, Multi-model |
| 3. | Microsoft SQL Server, 1989 | Relational, Multi-model |
| 4. | PostgreSQL, 1996 | Relational, Multi-model |
| 5. | MongoDB, 2009 | Document, Multi-model |

(solid IT, 2021)

Of the common choices, MongoDB stands out as a nonrelational model, and is considered a NoSQL database. MongoDB is the newest of the top 5 DBMS and shows steady growth over the years. Rather than storage of data in pages of tables and rows, MongoDB stores information in JSON-like documents (MongoDB, 2021). MongoDB documents very easily map to object-oriented programming designs, and those that hold related data may be fetched immediately rather than requiring to join tables like SQL languages. JSON document support has a good deal of potential especially when used with a JavaScript backend.

Oracle has support for many languages and is a commercial option for DBMS. Oracle is the most common DBMS now (solid IT, 2021) and many consider it the most scalable DBMS offering (AltexSoft, 2019). Oracle will have a high cost associated with the license, as well as a difficult learning curve; however, due to its popularity and Oracle’s support, a great deal of documentation can be found.

MySQL is an alternative to Oracle, also owned by Oracle but with a free version offered. Rather than Oracle, MySQL can restrict access of certain users to a desired hostname/domain (Oracle, 2005). MySQL additionally grants per user rather than using roles as Oracle can, as MySQL does not have roles. Both DBMS share a great deal of similarities in their schema. MySQL does not scale as well as Oracle or MongoDB (AltexSoft, 2019).

PostgreSQL is a completely open-sourced option that is dense with features and adheres closer to SQL standards than MySQL or Oracle; comparing MySQL to PostgreSQL, “[performance] will be good enough in either case, even if you consider expected future growth” (Hristozov, 2019). PostgreSQL may use inheritance and handle concurrency better than MySQL; PostgreSQL is object relational. The DBMS is very vertically scalable, with good support for many data types, and even some NoSQL features (AltexSoft, 2019)

MariaDB is an optional fork of MySQL that is increasing in popularity. This option is completely open source rather than MySQL itself; the MariaDB project is free, highly scalable, and supports the SQL standard strongly, while also supporting JSON document storage (MariaDB, 2021). Some small modifications to MySQL have resulted in some speed improvements.

SQLite is a good option to rapidly develop a working prototype. The DBMS is open source and support the SQL standard, and does not require a server or cloud connection. The DBMS does not support as many types of data as more popular options. For prototyping this self-contained database will allow portability and to transfer a working database for evaluation. SQLite will not be largely scalable as other options. The SQLite DB could be useful as a local cache in warehouses/stores perhaps to reduce load and some latency on a separate DBMS. SQLite would not be the best option for a full implementation; “SQLite competes with fopen()”, in other words SQLite is another method for keeping files locally; this DBMS is recommended for only a demo of an enterprise database, or for education of students or for training (SQLite, 2010). In addition, data types are somewhat limiting, especially when compared to SQL Standards, as shown in Table 2. While all other primitive types can be emulated, built in support would reduce some minor development time.

Table - All Available Storage Classes from SQLite

|  |  |
| --- | --- |
| Type | Storage class contains values: |
| NULL | The value is a NULL value. |
| INTEGER | The value is a signed integer, stored in 1-4, 6, or 8 bytes depending on the size of the value. |
| REAL | The value is a floating-point value, stored as an 8-byte IEEE floating point number |
| TEXT | The value is a text string, stored in the database encoding (UTF-8, UTF-16BE or UTF-16LE). |
| BLOB | The value is a blob of data, stored exactly as it was input. |

(SQLite, 2021)

SQLite is the selection for this project due to time and budgets constraints. It is recommended to use MariaDB for a complete implementation of the bookstore system, due to MariaDB’s scalability, and slight performance gains from MySQL, while remaining completely open source.

## 4.2. Web Frontend Recommendations

Today, there is a “triad of technologies that all Web developers must learn: HTML …, CSS … and JavaScript …” (Flanagan, 2011). These technologies make up a huge majority of all web pages in existence today, and a system for purchasing books from this bookstore will run off the same structure as no easily viable alternatives exist. Within JavaScript, the open-source language has advanced by many groups who have released projects publicly.

Angular, React, and TypeScript are three trending projects to examine. The Angular framework for JavaScript (JS) extends HTML and abstracts a great deal of DOM and AJAX code (AngularJS, 2020). The Google project would be a great benefit to this project by abstracting the common CRUD calls and simplifying development.

React by Facebook is an open-source JS library. React uses a virtual DOM to reduce the number of modifications to the real DOM, which improves performance greatly. In addition, downward data flow from the parent to child elements improves the stability of code and reduce updates (AltexSoft, 2020). Tree reconciliation offers a large performance improvement to many web apps using JS. Also, React would be conducive to highly reusable code and faster development due to its implementation of user interfaces as components (Buna, 2017).

Typescript is a very noteworthy trend as of today; JavaScript is extended by TypeScript to have static type definitions, reducing many of JavaScript’s largest complaints (TypeScript, 2021). The TypeScript language will produce native JavaScript, with the use of typing to verify code. Typescript reduces development time with code completion and may reduce type errors.

These JavaScript extensions could provide a great deal of benefit to the website with a larger customer base. Even in a basic implementation, use of Angular to rapidly produce CRUD functionality may be beneficial to the development time. Scaling with either React or Angular would likely be easier than with vanilla JS. While an initial implementation may only include vanilla JavaScript, it is recommended to consider Angular for a maintainable and more easily developed front end service.

## 4.3. Backend Recommendations

In this section, four options for a backend are chosen for their current rising popularity, and viability: Express for Node.js, Apache, Django for Python, and Flask for Python.

Express is a minimalistic framework for Node.js with great flexibility, scalability, performance, and a huge community (MDN Contributors, 2021). The framework is very lightweight, and therefore as a benefit has huge support from third party middleware and libraries. Use of JavaScript for the backend as well as frontend could have a benefit to development costs.

Apache is also an open-sourced web server software that provides efficient hosting for HTTP servers. A huge amount of today’s web is based off apache for its efficiency and ease of configuration (B, 2019). Apache is free and has the most deployments in the web today. Apache integrates perfectly with PHP and has been a staple since 1995.

Django is a Python framework that is known for its scalability and features (Django, 2021). Django has outstanding documentation and feature support as well as intelligently implemented security built into the base framework (MDN Constributors, 2021). This framework would be extremely robust and maintainable. A drawback for Django is relatively large amount of unused bloat for a smaller system as this would be.

Flask is also a viable trending option that also builds on Python. This “micro” web framework has limited functionality implemented by itself; Flask relies on other libraries to extend its abilities (Flask, 2010). This framework is intended to implement simple projects rapidly and is very pythonic. The framework has extremely limited features by itself, but this allows a large amount of flexibility in decisions on how to implement certain features, as extensions are necessary for even the basics. Scalability may suffer from choosing Flask.

The Flask Python framework will be used, to reduce costs in time of development. For the future implementation of a fully scalable version of the bookstore system, it is recommended to consider Node.js, as this framework is highly flexible and performant with a huge amount of third-party support as well as documentation.

## 4.4. Final Recommendation

The MEAN (MongoDB, Express.js, Angular, and Node.js) stack is considered the most popular web stack today, due to its great amount of flexibility, scalability, and ease of development. The LAMP (Linux, Apache, MariaDB and PHP) stack provide a great deal of competition to MEAN, however, LAMP has been losing popularity recently as JavaScript continues to dominate. Currently, Node has more support than PHP, and MEAN may have benefits in code reusability and refactoring due to being the same language frontend and backend. However, PHP has some slight performance improvements, and a great versatility in DBMS (PHP, 2020). SQL will be a convenient option for this system, and as such for a full implementation, LAMP is recommended for reconsideration.

However, for the time constraints and low developer support, to create a working prototype, the system will be built off the Flask Web Framework, Python, and SQLite. This is conducive to a rapid development time due to more rigidly defined methods, and with no investment in software.

# 5. Logical database design and Boyce-Codd Normalization

## 5.1. User Module

User, Employee and Manager – Users that are employees or managers may all be stored the same. The passwords will be kept encrypted on the database in a blob. Values for trust may be 1 for trust or -1 for distrust. Users and Trusts must be split. Emails and Usernames are unique and therefore must be split into a separate table to maintain the 3rd Normal Form.

CREATE TABLE Users

(

user\_id INTEGER CONSTRAINT PRIMARY KEY UNIQUE,

password BLOB NOT NULL,

role TEXT

);

CREATE TABLE Trusts

(

user\_id INTEGER FOREIGN KEY REFERENCES Users(user\_id),

trusts INTEGER FOREIGN KEY REFERENCES Users(user\_id),

value INTEGER CHECK (value = -1 OR value = 1)

);

CREATE TABLE Emails

(

user\_id INTEGER FOREIGN KEY REFERENCES Users(user\_id),

email TEXT NOT NULL UNIQUE

);

CREATE TABLE UserNames

(

user\_id INTEGER FOREIGN KEY REFERENCES Users(user\_id),

username TEXT NOT NULL UNIQUE

);

Customer – Customer attributes are atomic, and no bad dependencies exist.

CREATE TABLE Customers

(

user\_id INTEGER FOREIGN KEY REFERENCES Users(userID)

ON DELETE CASCADE,

phone INTEGER,

lastname TEXT NOT NULL,

firstname TEXT NOT NULL

);

## 5.2. Book Module

Author – Authors must have at least one name. Authors are split from Books to improve searchability and allow additional author information to be kept. No bad functional dependencies exist.

CREATE TABLE Authors

(

author\_id INTEGER PRIMARY KEY,

lastname TEXT,

firstname TEXT NOT NULL

);

Book – Redundancies have been removed from Books by splitting subjects, keywords, and images into separate tables, validating the 3rd Normal Form.

CREATE TABLE Books

(

isbn INTEGER PRIMARY KEY,

language TEXT NOT NULL,

pages INTEGER NOT NULL,

timestamp INTEGER NOT NULL,

publisher TEXT NOT NULL,

price REAL NOT NULL,

publish\_date INTEGER NOT NULL,

stock INTEGER DEFAULT 1 NOT NULL,

CHECK (pages > 0 and price >= 0)

);

CREATE TABLE AuthorWrites

(

author\_id INTEGER FOREIGN KEY REFERENCES Authors(author\_id)

ON DELETE CASCADE,

ISBN INTEGER FOREIGN KEY REFERENCES Books(isbn)

ON DELETE CASCADE

);

CREATE TABLE Subjects

(

subject\_id INTEGER PRIMARY KEY,

subject TEXT NOT NULL

);

CREATE TABLE BookHasSubject

(

keyword\_id INTEGER FOREIGN KEY REFERENCES Subjects(subject\_id)

ON DELETE CASCADE,

ISBN INTEGER FOREIGN KEY REFERENCES Books(isbn)

ON DELETE CASCADE

);

CREATE TABLE Keywords

(

keyword\_id INTEGER PRIMARY KEY,

keyword TEXT NOT NULL

);

CREATE TABLE BookRelatesTo

(

keyword\_id INTEGER FOREIGN KEY REFERENCES Keywords(keyword\_id),

ON DELETE CASCADE,

ISBN INTEGER FOREIGN KEY REFERENCES Books(isbn)

ON DELETE CASCADE

);

CREATE TABLE Images

(

image\_id INTEGER PRIMARY KEY,

isbn INTEGER FOREIGN KEY REFERENCES Books(isbn)

image BLOB NOT NULL

);

Sale – Sales have no functional dependences that are not trivial or not from the super key. SalesDiscounts must be separate from Sales to uphold Normalcy

CREATE TABLE Sales

(

sale\_id INTEGER PRIMARY KEY,

name TEXT NOT NULL,

discount REAL NOT NULL CHECK (discount > 0 AND discount <= 1),

date\_start INTEGER NOT NULL,

date\_end INTEGER NOT NULL

);

CREATE TABLE SalesDiscounts

(

sale\_id INTEGER FOREIGN KEY REFERENCES Sales(sale\_id),

isbn INTEGER FOREIGN KEY REFERENCES Books(isbn)

);

## 5.3. Comment Module

Comment –Comments do not have bad functional dependencies. Scores are constrained for books from 0-10, and comments should be removed without a book or customer who commented.

CREATE TABLE Comments

(

comment\_id INTEGER PRIMARY KEY,

content TEXT,

score INTEGER NOT NULL CHECK (score >= 0 and score <= 10),

timestamp INTEGER NOT NULL

isbn INTEGER FOREIGN KEY REFERENCES Books(isbn),

ON DELETE CASCADE,

user\_id INTEGER FOREIGN KEY REFERENCES Customers(user\_id)

ON DELETE CASCADE

);

Rating – Ratings may only be 0, 1, or 2 to represent useless, useful, or very useful respectively. Without a comment Ratings should be deleted.

CREATE TABLE Ratings

(

comment\_id INTEGER FOREIGN KEY REFERENCES Comments(comments\_id)

ON DELETE CASCADE,

value INTEGER NOT NULL CHECK (value >= 0 AND value <= 2),

timestamp INTEGER NOT NULL

);

## 5.4. Ordering Module

Address – An address will have attributes separated to allow easier statistics by location. Addresses are in Boyce-Codd Normal form because no functional dependencies exist beside the superkey.

CREATE TABLE Addresses

(

address\_id INTEGER PRIMARY KEY,

street\_number INTEGER,

street\_name TEXT NOT NULL,

apt\_number INTEGER,

city TEXT NOT NULL,

zip\_code INTEGER,

region TEXT NOT NULL,

country TEXT NOT NULL,

user\_id INTEGER FOREIGN KEY REFERENCES Customers(user\_id)

ON DELETE NO ACTION

);

Order – Orders must remain in the system until they are fulfilled. Additionally, orders should have a fulfilment attribute that is default unfulfilled. No functional dependencies exist beside the superkey.

CREATE TABLE Orders

(

order\_id INTEGER PRIMARY KEY,

timestamp INTEGER NOT NULL,

order\_quarter INTEGER,

time\_fulfilled INTEGER DEFAULT 0,

ships\_to INTEGER FOREIGN KEY REFERENCES Addresses(address\_id)

ON DELETE NO ACTION,

placed\_by INTEGER FOREIGN KEY REFERENCES Customers(user\_id)

ON DELETE NO ACTION

);

Order Item – Order Items have no bad functional dependencies.

CREATE TABLE OrderItems

(

item\_id INTEGER PRIMARY KEY,

price REAL NOT NULL CHECK (price >= 0),

quantity INTEGER DEFAULT 1 NOT NULL,

isbn INTEGER FOREIGN KEY REFERENCES Books(isbn),

ON DELETE CASCADE,

order\_id INTEGER FOREIGN KEY REFERENCES Orders(order\_id)

ON DELETE CASCADE

);

Coupon – Discounts must be a proportion off, so must be from 0-1. CouponDiscounts is separated out to allow multiple books to be discounted by one sale in Boyce-Codd Form. No functional dependencies exist beside the superkey, so Boyce-Codd Form Holds.

CREATE TABLE Coupons

(

coupon\_id INTEGER PRIMARY KEY,

discount REAL NOT NULL CHECK (discount > 0 and discount <= 1),

name TEXT NOT NULL,

code TEXT NOT NULL,

date\_start INTEGER NOT NULL,

date\_end INTEGER NOT NULL

);

CREATE TABLE CouponDiscounts

(

order\_id INTEGER FOREIGN KEY REFERENCES Orders(order\_id)

ON DELETE CASCADE,

coupon\_id INTEGER FOREIGN KEY REFERENCES Coupons(coupon\_id)

ON DELETE NO ACTION

);

## 5.5. Aggregations

Comment Rating – CommentRatings provides a view for customers to access the rating of comments.

CREATE VIEW CommentRating AS SELECT

(

Users.user\_id AS user\_id,

Comments.comment\_id AS comment\_id

avg\_rating,

) FROM (

SELECT user\_id, Comments.comment\_id, AVG(value) AS avg\_rating FROM Comments INNER JOIN Ratings GROUP BY user\_id

);

Customer Usefulness – Managers will have access to a view that aggregates all the trust ratings and usefulness for all comments by each user.

CREATE VIEW UserStats AS SELECT

(

Users.user\_id AS user\_id,

trust\_rating,

usefulness

) FROM

(

(SELECT Users.user\_id, SUM(Trusts.value) AS trust\_rating

FROM Users INNER JOIN Trusts),

(SELECT CommentRating.user\_id, AVG(avg\_rating) AS usefulness

FROM CommentRating GROUP BY CommentRating.user\_id)

)

WHERE Users.user\_id = CommentRating.user\_id

);

Orders of Books – Managers will be able to view a breakdown of each books sales, to be used to get statistics for popular books, publishers, and authors.

CREATE VIEW BookStats AS SELECT

(

isbn,

copies\_sold,

publisher,

quarter

) FROM (

SELECT Books.isbn AS isbn, SUM(quantity) AS copies\_sold, publisher, quarter FROM Books INNER JOIN OrderItems INNER JOIN Orders

GROUP BY Book.isbn

);

Orders to Addresses – Managers need access to the income from each city or possibly region; this aggregation will allow easy querying for additional information.

CREATE VIEW AddressOrderStats AS SELECT

(

city,

region,

country,

quarter,

order\_total

) FROM (

SELECT city, region, country, quarter, SUM(total) AS order\_total FROM

(

SELECT order\_id, address\_id, quantity\*SUM(price) AS total

FROM Orders INNER JOIN OrderItems

GROUP BY order\_id

)

INNER JOIN Addresses GROUP BY quarter, city, region, country

);

# Works Cited

AltexSoft. (2019, October 15). *“Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch and Others*. Retrieved from AltexSoft: www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/

AltexSoft. (2020, July 7). *The Good and the Bad of ReactJS and React Native*. Retrieved from AltexSoft Blog: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/

AngularJS. (2020). *Developer Guide*. Retrieved from AngularJS Docs: https://docs.angularjs.org/guide/introduction

B, A. (2019, September 18). *What is Apache? A Brief Explanation of the Popular Web Server Software*. Retrieved from HostingWiki: https://hostingwiki.org/what-is-apache/

Buna, S. (2017, March 30). *Yes, React is taking over front-end development. The question is why.* Retrieved from freeCodeCamp: https://www.freecodecamp.org/news/yes-react-is-taking-over-front-end-development-the-question-is-why-40837af8ab76/

Django. (2021). *Meet Django*. Retrieved from Django Project: https://www.djangoproject.com/

Flanagan, D. (2011). JavaScript: The Definitive Guide, 6th ed. O'Reilly.

Flask. (2010). *Welcome to Flask*. Retrieved from Pallets Projects: https://flask.palletsprojects.com/en/1.1.x/

Hristozov, K. (2019, July 19). *MySQL vs PostgreSQL -- Choose the Right Database for Your Project*. Retrieved from okta Developer: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres

MariaDB. (2021). *Proprietary Database Comparison*. Retrieved from MariaDB Database Topics: https://mariadb.com/database-topics/competitive-proprietary-comparison/

MDN Constributors. (2021, March 19). *Django introduction*. Retrieved from MDN Web Docs: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

MDN Contributors. (2021, February 16). *Express/Node introduction*. Retrieved from MDN Web Docs: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\_Nodejs/Introduction

MongoDB. (2021). *Comparing the Differences - MongoDB vs MySQL*. Retrieved from MongoDB: www.mongodb.com/compare/mongodb-mysql

Oracle. (2005, July 10). *Database SQL Developer Supplementary Information for MySQL Migrations*. Retrieved from Oracle Help Center: docs.oracle.com/cd/E12151\_01/doc.150/e12155/oracle\_mysql\_compared.htm

PHP. (2020). *What is PHP?* Retrieved from PHP Manual: https://www.php.net/manual/en/intro-whatis.php

solid IT. (2021, March). *DB-Engines*. Retrieved from DB-Engines: db-engines.com/en/ranking

solid IT. (2021, March). *Historical Trend of the Popularity Ranking of Database Management Systems*. Retrieved from DB-Engines: db-engines.com/en/ranking\_trend

SQLite. (2010). *Appropriate Uses For SQLite*. Retrieved from SQLite: https://www.sqlite.org/whentouse.html

SQLite. (2021). *Datatypes In SQLite Version 3*. Retrieved from SQLite: https://www.sqlite.org/datatype3.html

TypeScript. (2021). *Typed JavaScript at Any Scale.* (Microsoft) Retrieved from TypeScript: https://www.typescriptlang.org/