Kew & Willow Books

Database Design Proposal

Gabriella Nina

Table of Contents

Executive Summary

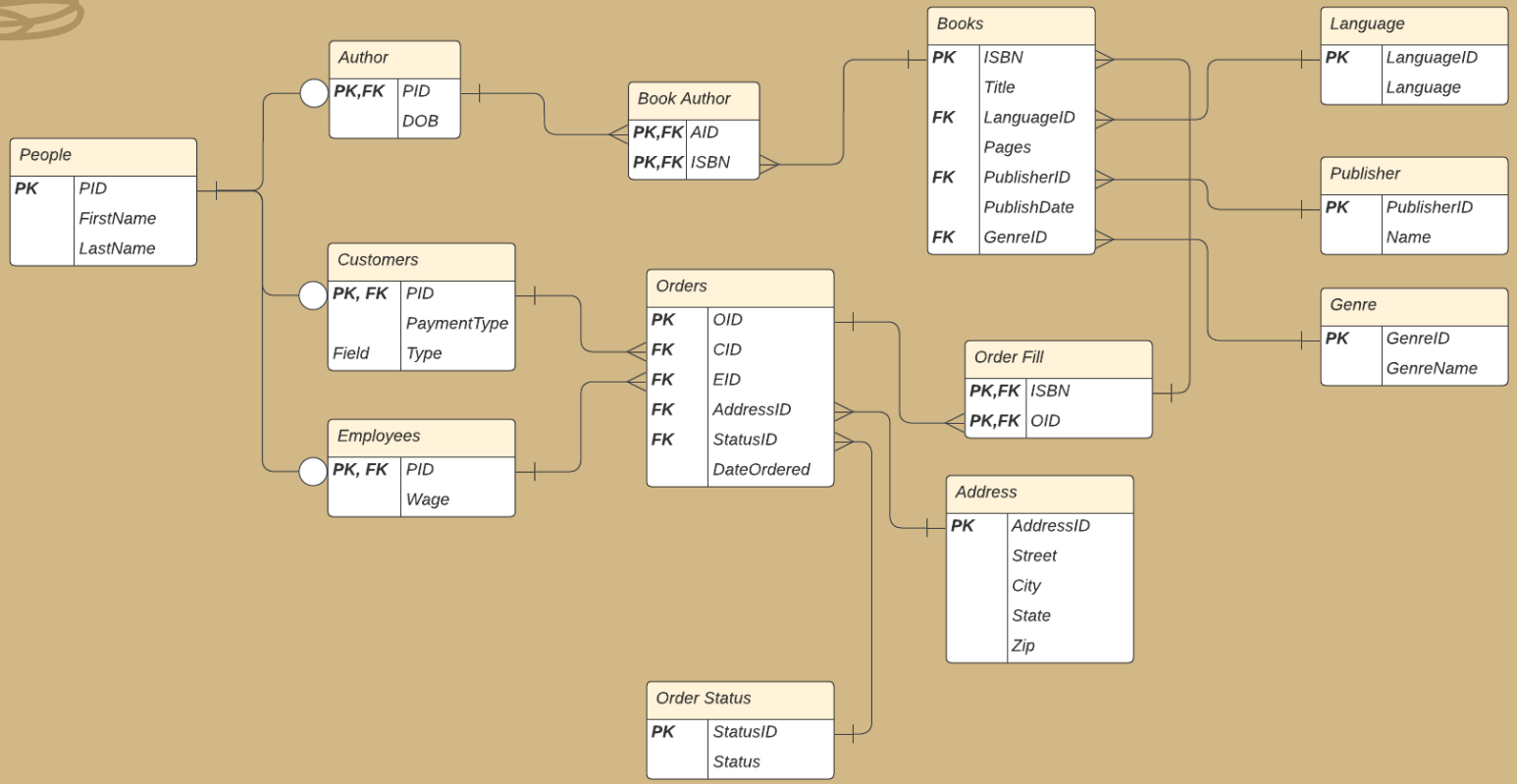
This document will outline a database system to manage Kew & Willow Books, a small local bookstore in Queens, NY. This bookstore allows for book purchasing to be done online as well as in person. This store really focuses on selling books of all kinds and making a big effort to find and expose books that may not be as popular as others.

The purpose of my database design will be to provide information on books, customers, employees, and orders. Throughout this documentation I will go through each of the tables needed for this database and their functional dependencies, stored procedures, as well as some sample data with its outputs.

I believe this is a great start to manage this wonderful bookstore

Kew & Willow Books

ER Diagram


People
PK PID
FirstName
LastName
Author
PK,FK PID
DOB
Customers
PK, FK PID
PaymentType
Field Type
Employees
PK, FK PID
Wage
Book Author
PK,FK AID
PK,FK ISBN
Orders
PK OID
FK CID
FK EID
FK AddressID
FK StatusID
DateOrdered
Order Status
PK StatusID
Status
Books
PK ISBN
Title
FK LanguageID
Pages
FK PublisherID
PublishDate
FK GenreID
Order Fill
PK,FK ISBN
PK,FK OID
Address
PK AddressID
Street
City
State
Zip
Language
PK LanguageID
Language
Publisher
PK PublisherID
Name
Genre
PK GenreID
GenreName

People Table

The People Table holds the information in common with all the people this database. It will have a PID, first name, last name, and email

```
-- People --
CREATE TABLE People (
   pid          int not null,
   firstName    text,
   lastName     text,
 primary key(pid)
);
```

Functional Dependency: pid → firstname, lastname

	pid [PK] integer	firstname text	lastname text
1	1	Taylor	Jenkins-Reid
2	2	Hanya	Yanagihara
3	3	Colleen	Hoover
4	4	Gabriel	Garcia-Marquez
5	5	Dalai	Lama
6	6	Delia	Owens
7	7	Paulo	Coelho
8	8	Zakiya	Dalila-Harris
9	9	Gabriella	Nina
10	10	Mya	Fernandez
11	11	Bria	Royer
12	14	Alan	Labouseur
13	13	Booke	David
14	12	Wilfredo	Nina
15	15	Angelina	Chirichella
16	16	Nathan	Scott
17	17	Karen	Roe
18	18	Karin	Yannes

Author Table

The Author Table contains the author's PID and DOB

```
--Authors--
CREATE TABLE Authors (
   pid         int not null references people(pid),
   DOB         date,
 primary key(pid)
);
```

Functional Dependency: pid → dob

	pid [PK] integer	dob date
1	1	1983-12-20
2	2	1974-09-20
3	3	1979-12-11
4	4	1927-03-06
5	5	1935-07-06
6	6	1949-04-04
7	7	1947-08-24
8	8	1979-07-03

Employees Table

The Employees Table contains PID, hourly wage, and Payment Terms

```
--Employees--
CREATE TABLE Employees (
   pid         int not null references people(pid),
   wageUSD     numeric(10,2),
 primary key(pid)
);
```

	pid [PK] integer	wageusd numeric (10,2)
1	15	15.00
2	16	15.00
3	17	17.00
4	18	15.00

Functional Dependency: pid → wageUSD

Customers Table

The Customer Table contains PID and payment type

```
--Customers
 drop table customers;
 CREATE TABLE Customers (
   PID           int references people(pid),
   paymenttype.  text,
   email         text,
 primary key(PID)
);
```

	pid [PK] integer	paymenttype text	email text
1	9	ApplePay	gabriellanina@aol.com
2	10	Cash	myafernandez@yahoo.com
3	11	Credit	briaroyer45@gmail.com
4	12	Cash	alanlabouseur34@gmail.com
5	13	ApplePay	Brookedavis87@aol.com
6	14	Credit	Wilfredonina@yahoo.com

Functional Dependency: pid → paymenttype, email

Book Author Table

The Book Author Table is a mapping table that will allow us to connect Authors to books. This table contains AID(author ID) and BID (book id)

```
--BookAuthor--
 CREATE TABLE BookAuthor (
   aid          int not null,
   ISBN         varchar(13),
 primary key(aid, ISBN)
 );
```

Functional Dependency: aid, isbn →

	aid [PK] integer	isbn [PK] character varying (13)
1	1	9781501161933
2	2	9780804172707
3	3	9781501110368
4	4	9780307474728
5	5	9781573228831
6	6	9780735219106
7	7	9780062315007
8	8	9781982160135
9	3	9781538724736

Books Table

The Book Table contains International Standard Book Number (ISBN), price of book (PriceUSD), LanguageID, Title, number of pages, PublisherID, PublishDate, GenreID

```
--Books--
CREATE TABLE Books (
   ISBN          varchar(13) not null,
   title         varchar(400),
   priceUSD      numeric(10,2),
   languageID    varchar(5),
   pages         int,
   publisherID   char(4),
   PublishDate   date,
   genreID       varchar(4),
 primary key(ISBN)
);
```

	isbn [PK] character varying (13)	title character varying (400)	priceusd numeric (10,2)	languageid character varying (5)	pages integer	publisherid character (4)	publishdate date	genreid character varying (4)
1	9781501161933	The Seven Husbands of Evelyn Hugo	17.00	L0001	400	pub1	2018-05-29	gfic
2	9780804172707	A Little Life	18.00	L0001	832	pub2	2016-01-28	gfic
3	9781501110368	It Ends with Us	16.99	L0001	384	pub3	2016-08-02	grom
4	9780307474728	Cien Años de Soledad	17.00	L0002	496	pub4	2009-09-22	gmag
5	9781573228831	Ethics for the New Millennium	16.00	L0001	256	pub5	2001-05-01	grel
6	9780735219106	Where the Crawdads Sing	18.00	L0001	400	pub6	2021-03-30	gifc
7	9780062315007	The Alchemist	16.99	L0001	108	pub7	2014-04-15	gfan
8	9781982160135	The Other Black Girl	27.00	L0001	368	pub3	2021-06-01	gfic
9	9781538724736	Verity	16.99	L0001	336	pub8	2021-09-26	gthr

Functional Dependency: ISBN → Title, PriceUSD, LanguageID, pages, publisherID, PublishDate, genreID

Publisher Table

The Publisher Table contains PublisherID and the name of the publisher

```
----Publisher--
CREATE TABLE Publisher (
   PublisherID      varchar(4),
   PublisherName    text,
 primary key(PublisherID)
);
```

	publisherid [PK] character varying (4)	publishername text
1	pub1	Washington Square Press
2	pub2	Anchor
3	pub3	Atria Books
4	pub4	Vintage Espanol
5	pub5	Riverhead Books
6	pub6	G.P. Putnams Sons
7	pub7	HarperOne
8	pub8	Grand Central Publishing

Functional Dependency: publisherid → publishername

Orders Table

The Orders Table contains the order ID, Customer ID(CID), Employee ID(EID) who packaged the order, Address ID to connect to the address of the customer, Status ID which will connect to the actual status of the order

```
--Orders
 CREATE TABLE Orders (
  OID                 varchar(4) ,
  CID                 int references customers(pid),
  EID                 int references employees(pid),
  AddressID           varchar(3),
  StatusID            text,
  DateOrdered         date,
 primary key(OID)
);
```

Functional Dependency: oid → cid, eid, addressed, statusid, dateordered

	oid [PK] character varying (4)	cid integer	eid integer	addressid character varying (3)	statusid text	dateordered date
1	o001	9	15	a09	S	2022-05-01
2	o002	14	17	a14	R	2022-02-05
3	o004	12	16	a13	D	2022-03-09

Order Fill Table

The Order Fill Table contains the BID for all the books in the order, and the Order ID number(OID).

```
 --OrderFill--
CREATE TABLE OrderFill (
   OID         varchar(4) ,
   ISBN        varchar(13) not null,
 primary key(OID,ISBN)
);
```

	oid [PK] character varying (4)	isbn [PK] character varying (13)
1	o001	9780062452993
2	o002	9780307474728
3	o003	9780062315007
4	o004	9781501110368
5	o004	9780804172707

Functional Dependency: oid isbn →

Address Table

The Address Table contains the Address ID, Street, City, state, and ZIP.

```
--Address--
drop table address
 CREATE TABLE Address (
   addressID          varchar(3) not null,
   street             text,
   city               text,
   state              text,
   ZIP                int,
 primary key(addressID)
 );
```

	addressid [PK] character varying (3)	street text	city text	state text	zip integer
1	a09	8410 101st	Richmond Hill	NJ	11418
2	a13	17 Sunset AVE	Lynbrook	NY	11563
3	a14	209E 81st	New York	NY	10028

Functional Dependency: addressid → street, city, state, zip

Order Status Table

The Order Status Table contains contains the status ID which connects the orders to the Order Status table which also includes the status.

```
 --OrderStatus
CREATE TABLE OrderStatus (
   StatusID            varchar(4) ,
   Status              text,
 primary key(StatusID)
);
```

Functional Dependency: statusid → status

	statusid [PK] character varying (4)	status text
1	D	Delivered
2	P	Processing
3	S	Shipped

Genre Table

The Genre Table connects each book to its matching genre with a GenreID and the name of the Genre

```
----Genre--
CREATE TABLE Genre (
   GenreID    varchar(4),
   GenreName  text,
   primary key(GenreID)
);
```

	genreid [PK] character varying (4)	genrename text
1	gfic	Fiction
2	grom	Romance
3	grel	Religion
4	gmag	Magical Realism
5	gfan	Fantasy
6	gthr	Thriller

Functional Dependency: genreid → genrename

Language Table

The Language Table contains the Address LanguageID and LanguageName.

```
----Language--
CREATE TABLE Language (
   languageID     varchar(5),
   languageName   text,
 primary key(languageID)
);
```

	languageid [PK] character varying (5)	languagename text
1	L0001	English
2	L0002	Spanish

Functional Dependency: languageid → languagename

Interesting Queries

GET THE PUBLISHER WITH THE MOST BOOKS

```
SELECT p.publishername, count(distinct b.isbn) as publishersum
FROM books b inner join publisher p on b.publisherid=p.publisherid
GROUP BY p.publishername
ORDER BY "publishersum" DESC
LIMIT 1;
```

	publishername text	publishersum bigint
1	Atria Books	2

WHICH EMPLOYEE PACKED ALAN LABOUSER'S ORDER?

```
SELECT o.eid
FROM orders o inner join employees e on o.eid=e.pid
              inner join people p on o.cid = p.pid
WHERE firstname = 'Alan'
```

	eid integer
1	17

WHAT IS IN WILFREDO NINA'S ORDER?

```
SELECT b.title
FROM orders o inner join orderfill of on o.oid = of.oid
              inner join people p on p.pid = o.cid
              inner join books b on b.isbn = of.isbn
WHERE firstname = 'Wilfredo' and lastname ='Nina'
GROUP BY b.title;
```

	title character varying (400)
1	A Little Life
2	It Ends with Us

Orders in NY

View that will inner join orders with addresses and print out orders in New York

```
-- customers in New York
CREATE VIEW customersNY
AS
SELECT o.cid
FROM orders o INNER JOIN address a on o.addressid=a.addressid
WHERE state = 'NY';
```

v All the orders attached to their address v

	oid character varying (4)	cid integer	eid integer	addressid character varying (3)	statusid text	dateordered date	addressid character varying (3)	street text	city text	state text	zip integer
1	o001	9	15	a09	S	2022-05-01	a09	8410 101st	Richmond Hill	NJ	11418
2	o002	14	17	a14	R	2022-02-05	a14	209E 81st	New York	NY	10028
3	o004	12	16	a13	D	2022-03-09	a13	17 Sunset AVE	Lynbrook	NY	11563

v All the orders in New York v

	cid integer	addressid character varying (3)	street text	city text	state text	zip integer
1	14	a14	209E 81st	New York	NY	10028
2	12	a13	17 Sunset AVE	Lynbrook	NY	11563

Authors and Books

View that will inner join Authors and their Books

```
CREATE VIEW bookswithauthors
AS
SELECT p.firstname, p.lastname,
FROM bookauthor ba INNER JOIN authors a on ba.aid=a.pid
                   INNER JOIN books b on ba.isbn = b.isbn
                   INNER JOIN people p on p.pid=ba.aid;
```

	firstname text	lastname text	title character varying (400)
1	Taylor	Jenkins-Reid	The Seven Husbands of Evelyn Hugo
2	Hanya	Yanagihara	A Little Life
3	Colleen	Hoover	It Ends with Us
4	Gabriel	Garcia-Marquez	Cien Años de Soledad
5	Dalai	Lama	Ethics for the New Millennium
6	Delia	Owens	Where the Crawdads Sing
7	Paulo	Coelho	The Alchemist
8	Zakiya	Dalila-Harris	The Other Black Girl
9	Colleen	Hoover	Verity

Employee Names

This view will allow us to easily see all employees with their name and PID

```
CREATE VIEW employeenames
AS
SELECT p.firstname, p.lastname, p.pid
FROM employees e INNER JOIN people p on e.pid = p.pid;
```

	firstname text	lastname text	pid integer
1	Angelina	Chirichella	15
2	Nathan	Scott	16
3	Karen	Roe	17
4	Karin	Yannes	18

Stored Procedure: Look up a Book

This stored procedure that will allow users to look up a book within this database. The procedure takes in a title of a book and will print out all corresponding data of that book. This procedure can easily be modified to look up other elements of the database.

```
CREATE OR REPLACE FUNCTION
searchBook(TEXT, REFCURSOR) RETURNS
refcursor AS
$$
DECLARE
searchFirst TEXT := $1;
resultSet REFCURSOR := $2;
BEGIN
OPEN resultset FOR
 SELECT *
 FROM books
 WHERE searchFirst = title;
return resultSet;
end;
$$
LANGUAGE plpgsql;
```

SAMPLE RESULTS

FIND "A Little Life'

```
SELECT searchBook('A Little Life', 'results');
FETCH ALL FROM results;
```

	ibsn [PK] character varying (13)	title character varying (400)	priceusd numeric (10,2)	languageid character varying (5)	pages integer	publisherid character (4)	publishdate date	genreid character varying (4)
1	9780804172707	A Little Life	18.00	L0001	832	pub2	2016-01-28	gfic

FIND "The Seven Husbands of Evelyn Hugo'

```
SELECT searchBook('The Seven Husbands of Evelyn Hugo', 'results');
FETCH ALL FROM results;
```

	ibsn [PK] character varying (13)	title character varying (400)	priceusd numeric (10,2)	languageid character varying (5)	pages integer	publisherid character (4)	publishdate date	genreid character varying (4)
1	9781501161933	The Seven Husbands of Evelyn Hugo	17.00	L0001	400	pub1	2018-05-29	gfic

Stored Procedure: How many pages?

This stored procedure that will allow users to look up how many pages are in an imputed book.

```
CREATE OR REPLACE FUNCTION
howmanypages(TEXT,  REFCURSOR) RETURNS
refcursor AS
$$
DECLARE
searchFirst TEXT := $1;
resultSet REFCURSOR := $2;
BEGIN
OPEN resultset FOR
 SELECT pages
 FROM books
 WHERE searchFirst = title;
return resultSet;
end;
$$
LANGUAGE plpgsql;
```

SAMPLE RESULTS

FIND "Cien Años de Soledad"

```
SELECT howmanypages('Cien Años de Soledad', 'results');
FETCH ALL FROM results;
```

	pages integer
1	496

FIND "The Alchemist'

```
SELECT howmanypages('The Alchemist', 'results');
FETCH ALL FROM results;
```

	pages integer
1	108

Stored Procedure: Books Written by

This stored procedure that will allow users to look up how many pages are in an imputed book.

```
CREATE OR REPLACE FUNCTION
bookswrittenby(TEXT, TEXT, REFCURSOR) RETURNS
refcursor AS
$$
DECLARE
searchFirst TEXT := $1;
searchLast TEXT := $2;
resultSet REFCURSOR := $3;
BEGIN
OPEN resultset FOR
 SELECT *
 FROM bookswithauthors
 WHERE firstname = searchFirst
 AND lastname = searchLast;
return resultSet;
end;
$$
LANGUAGE plpgsql;
```

SAMPLE RESULTS

FIND BOOKS WRITTEN BY "Colleen Hoover"

```
SELECT bookswrittenby('Colleen','Hoover', 'results');
FETCH ALL FROM results;
```

	firstname text	lastname text	title character varying (400)
1	Colleen	Hoover	It Ends with Us
2	Colleen	Hoover	Verity

FIND BOOKS WRITTEN BY "Paulo Coelho"

```
SELECT bookswrittenby('Paulo','Coelho', 'results');
FETCH ALL FROM results;
```

	firstname text	lastname text	title character varying (400)
1	Paulo	Coelho	The Alchemist

Triggers

Validate People Trigger: this will make sure we don't input any people without first or last names

```
CREATE OR REPLACE FUNCTION ValidatePeople()
RETURNS TRIGGER AS
$$
BEGIN
IF NEW.firstName IS NULL THEN
RAISE EXCEPTION 'firstName may not be NULL';
END IF;
IF NEW.lastName IS NULL THEN
RAISE EXCEPTION 'lastName may not be NULL';
END IF;
RETURN NEW;
END
$$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER validPeople
BEFORE INSERT OR UPDATE ON People
FOR EACH ROW
EXECUTE PROCEDURE ValidatePeople();
```

Sample output

```
INSERT INTO People(pid, firstName, lastName)
VALUES('19', NULL, NULL);
```

```
ERROR:  firstName may not be NULL
CONTEXT:  PL/pgSQL function validatepeople() line 4 at RAISE
SQL state: P0001
```

```
INSERT INTO People(pid, firstName, lastName)
VALUES('19', 'Lizbeth', NULL);
```

```
ERROR:  lastName may not be NULL
CONTEXT:  PL/pgSQL function validatepeople() line 7 at RAISE
SQL state: P0001
```

```
INSERT INTO People(pid, firstName, lastName)
VALUES('19', NULL,'Lizbeth');
```

```
ERROR:  firstName may not be NULL
CONTEXT:  PL/pgSQL function validatepeople() line 4 at RAISE
SQL state: P0001
```

Triggers

Validate Book Trigger: this will make sure we don't input any books without a title

```
CREATE OR REPLACE FUNCTION ValidateBook()
RETURNS TRIGGER AS
$$
BEGIN
IF NEW.title IS NULL THEN
RAISE EXCEPTION 'You must enter the title
of the book';
END IF;
RETURN NEW;
END
$$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER validbook
BEFORE INSERT OR UPDATE ON Books
FOR EACH ROW
EXECUTE PROCEDURE ValidateBook();
```

Sample output:

```
INSERT INTO Books(ISBN, title, priceUSD, languageid, Pages, PublisherID, PublishDate, genreID)
VALUES(9780307389732, NULL , 16.95, 'L0001', 368, 'pub4', '2007-10-05','gfic');
```

```
ERROR:  You must enter the title of the book
CONTEXT:  PL/pgSQL function validatebook() line 4 at RAISE
SQL state: P0001
```

Security

There are two user roles:
Owner and Manager

```
CREATE ROLE owner;
CREATE ROLE manager;
```

Owner: Has the power over the entire database

```
CREATE ROLE owner;
GRANT ALL ON ALL TABLES IN SCHEMA PUBLIC TO owner;
```

Manager: Has the power to select, insert, or update almost all the tables. However, the Manager cannot manage the employee table. This is to prevent change in wage.

```
CREATE ROLE associate;
grant SELECT, INSERT, UPDATE, DELETE on address to manager;
grant SELECT, INSERT, UPDATE, DELETE on bookauthor to manager;
grant SELECT, INSERT, UPDATE, DELETE on authors to manager;
grant SELECT, INSERT, UPDATE, DELETE on books to manager;
grant SELECT, INSERT, UPDATE, DELETE on orders to manager;
grant SELECT, INSERT, UPDATE, DELETE on customers to manager;
grant SELECT, INSERT, UPDATE, DELETE on genre to manager;
grant SELECT, INSERT, UPDATE, DELETE on language to manager;
grant SELECT, INSERT, UPDATE, DELETE on orderfill to manager;
grant SELECT, INSERT, UPDATE, DELETE on orderstatus to manager;
grant SELECT, INSERT, UPDATE, DELETE on people to manager;
grant SELECT, INSERT, UPDATE, DELETE on publisher to manager;
grant SELECT, on employees to manager;
```

Implementation Notes

1. All the books and authors mentioned in this database design are actual books and authors that are available for purchase at Kew and Willow ‘s online store
2. All information on books and authors are real records
3. Some of the IDs used, is formatted for the sake of this database design. The number of characters may need to be modified depending on the history of the stores orders
4. There are 7,111 recorded languages in the world, hence I created the languageID large enough to account for each of the languages
5. The orders to order fill table are designed to allow for multiple books to be purchased in one order

Known Problems/Future Enhancements

I believe this design is very effective for the size of the store. One aspect I would like to look at is possibly finding a way to query the best sellers within a genre. Eventually if the owners decide to open a second location, then this will affect the design tremendously. This will most likely lead to creating more mapping tables to connect the stock of books to the correct store. Another thing to keep in mind for the future is expanding merchandise to things other than books such as bags and book accessories. This would make major updates necessary to our design.