# ECE 356

# Final Project : 20-Books

Group 57
Si Qi Guo
sqguo@uwaterloo.ca

https://git.uwaterloo.ca/ece356-f21-g57/final-project

## Contents

# Project Summary

This project combines 2 large Kaggle datasets from Goodreads and the Seattle Public Library. While each of the datasets only contain a small number of fields, limiting their respective effectiveness, when combined the datasets become a high-quality source of information for both our client application and our data mining operations. This project's source code is located at https://git.uwaterloo.ca/ece356-f21-g57/final-project. The source code contains three parts, the data loading scripts, the client application and the data mining scripts. Please read the README.md and MANUAL.md found within the project directory to understand how to install and run the project.

# The Entity-Relationship Design

One of the main challenges of the project comes from merging the Goodreads dataset and the Seattle Library dataset. Both datasets contain the (sometime conflicting) definition of a book. Merging them directly is not ideal. Therefore, we must carefully understand how these two datasets relate to each other and make decisions to avoid database errors.

## A brief Primer on ISBN and Decisions

The only field that relates the two datasets is the ISBN field. International Standard Book Number (ISBN) is an internationally recognized book identifier issued to each book. Each edition of the book will have different ISBNs issued for it even if the difference is minor. Sometimes 2 books may be virtually indistinguishable from each other using only human readable fields

Before 2007 ISBN10 was issued. After 2007, ISBN13 replaced ISBN10. All ISBN10 can be mapped to a unique ISBN13. However, the reverse is not true. Both the Goodreads and the Library dataset contains a mixture of ISBN10 and ISBN13. To ensure we do not have duplicates of the same data; we convert all ISBNs to ISBN13 and use as our canon identifier.

Some books may not have ISBN. As a result, in the Goodreads dataset their ISBN number is empty. These data are not useful to us because we have no way to relate them to the inventory of the library. Therefore, we made the decision to discard them.

Since an ISBN identifier contain multiple digits, it is prone to user input error. Luckily, each ISBN identifier implements check digits for the purpose of validation. Ideally, the database would implement check functions internally to ensure each ISBN number stored is correct. This would significantly reduce the chance of error. However, this is time consuming, and we consider this to be out of the scope of this project. Instead, to reduce error, all our ISBN validation is done client side, and before the data enters the database. The client application will perform internal checking to ensure a valid ISBN is entered. The data loading scripts also preforms ISBN validation on each row of the exported CSV file and filter out rows that do not have a valid ISBN10 or ISBN13.

The ISBN13 contains 13 digits, as its name implied. Unlike the ISBN10, all digits of ISBN13 are decimals. Therefore, for all ISBN fields in our database, the attribute type will be decimal(13). We zero-fill it to 13 digits if necessary. Finally, we implement a minor check function to ensure no negative numbers is contained in the field.

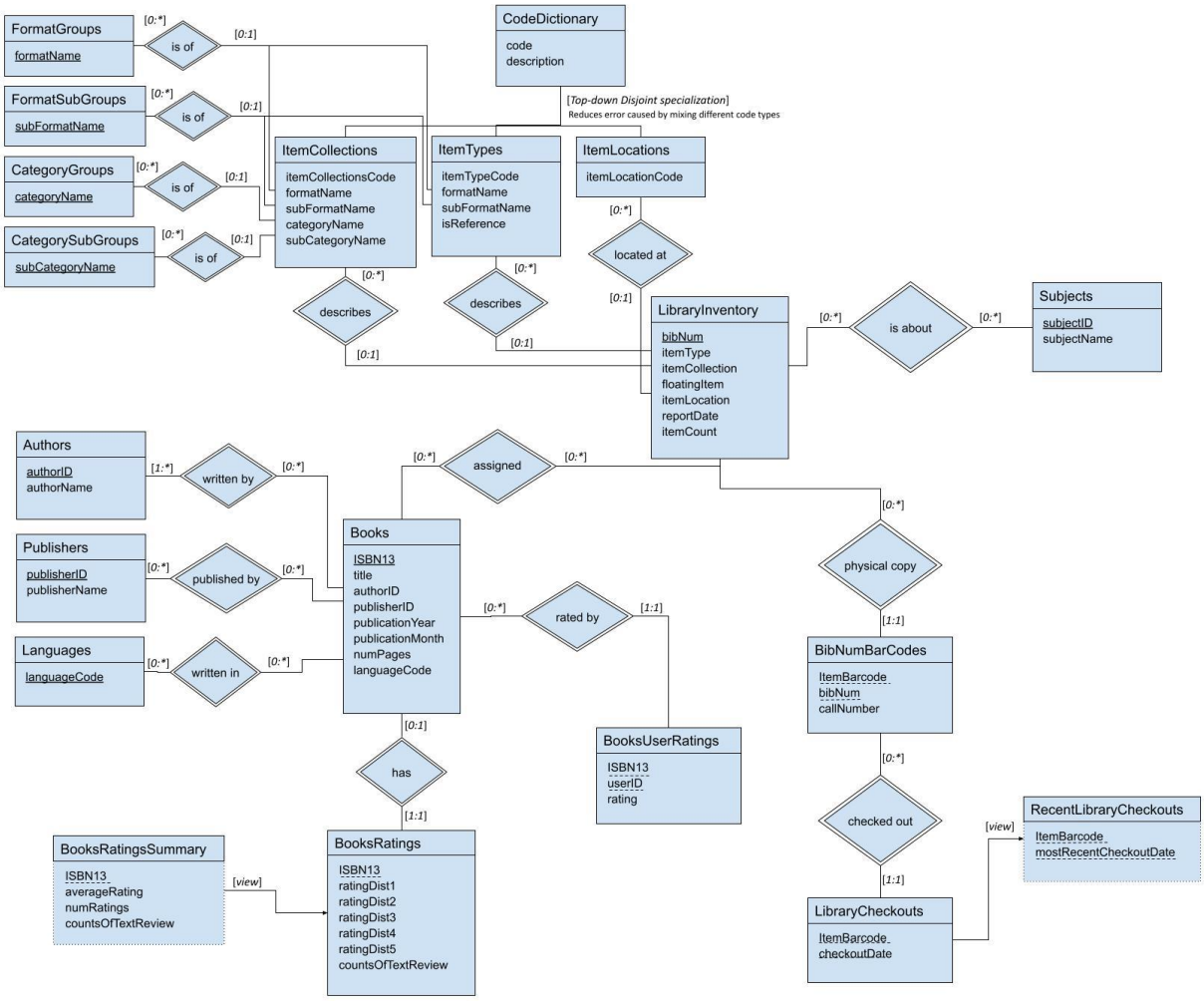


*Figure 0.1 A diagram illustrating the ER model of our database*

## The Books Tables and other Entities

Based on our understanding of the datasets, we have determined that each row of the Goodreads books dataset will refer to an edition of a book. On the other hand, each row of the Library dataset refers to entry that groups together similar editions of books, because a reader who wants to read a book would be unlikely to care about the specific edition of a book, it will be used for user-facing applications. Therefore, we have two main tables, the Books table (referring to the book editions) and the LibraryInventory table. The books table will have the primary key ISBN13 while the inventory table will have the primary key BibNumber. The two tables will be connected using the relationship set BibNumISBNs. Not all entries in the Library inventory are books – some are physical equipment, and some are digital items. Not all book editions will be owned by the library. Libraries at multiple locations may each contain an entry in the inventory that refers to the same book edition. Given this information, both book editions and inventory item may appear in the relationship set zero to infinite times. In addition, the relationship set will have the primary key composing of both the ISBN13 and the BibNumber.

Another entity worth noting is the BibNumBarCodes entity. In this project, we consider every physical copy of an item (that can be borrowed) to be identified by a Barcode. Every barcode will be unique (hence the primary key on it), and multiple barcodes can map to the same Library entry, but no barcode can be mapped to two different library entry simultaneously. Since a BibNumBarCodes entity does not exists without a library entry, we consider it to be a weak entity and we designed it with a non-nullable foreign key to its parent entity. It also has an additional field called call number. A call number is used in the library to order and group items together. The call number is not guaranteed to be unique, does is it guaranteed to be valid (at least according to our dataset), hence we do not make it part of the key.

The LibraryCheckouts entity is a weak entity that depends on the existence of BibNumBarCodes (because we assume that a checkout requires a physical barcode), therefore the entity contains a non-nullable foreign key to BibNumBarCodes. Since a physical copy of the book can be checked out multiple times (often in a year), however a physical copy can not be checked out simultaneously at the same time, therefore we decided to make the primary key of the entity to be a combination of barcode and checkout date, (using dateTime with precision down to minutes, to allow us to understand when in the day a user checkout a book).

One common use case of the checkout's entity is to look for the most recent date the item is checked out (so we can see whether an item is overdue or not). Therefore, we created a view based on LibraryCheckouts that computes the maximum checkout date by barcode. This design simplifies our client application's checkout functionality. Ideal, we would know the return period as well to create new views to indicate which barcode entity is available. However, since we do not have the required dataset, this is not implemented.

## Authors and Publishers

In addition to the books' entities, we found the need to create entities sets for publishers and authors. We consider them to be independent of the books' entities (because an author may have never published a book). Each author is uniquely identified by an ID (primary key), this ensures that even if two authors exist at the same time, they can be distinguished. However, since the dataset only provides author names, not unique identifiers, we have to make the assumption when population our database that authors with same names will refer to the same person. This is of course not ideal. Similar thing can be said of the publisher's table. Another advantage of making separate entities for authors and publishers is that it allows us to specify additional attribute on them, such as the author's date of birth (which is useful for understanding books from the last century). The books' entities will relate to these tables using a nullable-foreign key (nullable because a book may be self-published or may be written by a machine). We also assumed that a book edition can have at most one author and one publisher. This is also not ideal. In the future, we may consider a relationship set BooksAuthors to record one to many relationships for these entities.

## Subjects, Languages, and Enumeration

We heavily used enumeration in the database design, for subjects, languages, and categories. We created entity sets for each that places a unique key on the name of subjects, languages, or categories. Enumeration allows us to reduce the chance of input error, by requiring for example, a new language to be added to the enumeration before adding a library entry that refers a language. This ensures that we don't have different spellings of ENG and English at the same time. However, during data loading process, we have no way to check for the semantic distinctiveness of the enumerations. This is the ideal of course.

Ideally, we would import official databases for languages and subjects to further avoid duplication error. The library entities will refer to languages and categories respectively using foreign key. These keys are nullable because sometime this information is not present in the dataset. We also have a relationship set called BibNumSubjects to allow multiple subjects to be related to the same library entry. Therefore, it is a one-to-many relationship, with foreign keys to the library entries and subjects, and a unique key on the combination of both (two books can have the same subject, a book can have multiple subjects).

## The Code Dictionary and Specialization

In this project we also implement a code dictionary for library entries. There are three types of codes that we are concerned about: type code, category code and location code. Since every code must have a description, we found it useful to have a parent Code Dictionary entity that contains both the codes and their description. Each type of code is a disjoint specialization of the Code Dictionary entity, and by creating specializations we allow each type to have additional attributes attached to them. For example, categoryName is available only to ItemCollections entity and not others. It also allows us to make sure that a foreign key referring to the code is correct when used by a library entity (To avoid using a location code when we should be using item type). The original dataset does not create specialization; therefore, we have to manually parse the specializations during data loading.

## Books Ratings Tables and Views

In the Goodreads dataset we have counts of ratings for each edition of a book. Some books may not have ratings yet. Therefore, we created a weak entity called BooksRatings with foreign key to the books' entities. The primary key is also the foreign key because no book can have two sets of ratings. The rating counts are separate by the number of 1 to 5 stars. However, this is not useful to us because the most likely use case of the ratings will be average rating. Therefore, we created a view called BooksRatingsSummary that calculates the average rating (as well as the total number of ratings) from the BooksRatings entity.

## Indexes of the Database and the Issues of String Comparison

In the database we employ multiple additional indexes to speed up performance. One use case we consider is searching a physical copy of the book by barcode. As mentioned previously a call number is typically used in this scenario, therefore we decided to place an index on the column. A frequent thing people does it looking up books by their publication date. Therefore, we have an index on the combination of year, month, and day, strictly in that order. We place year first instead of the other columns because no sane users would try to find book using only day of the month. Another frequent thing people does is looking up books by their title. Since our database contain foreign language books, with titles not represented by the standard latin1 encoding, we must use utf8mb4_bin. In addition, the length of titles ranges from below 10 characters to a maximum length of 1035 (a valid research paper title), we had to make titles a varchar of maximum length 1050. This reduces the available size of the column. When we tried to add an index to the title column, MySQL informed us that the index cannot be added due to its excess size. Ideally, we should resolve this problem by updating server configurations. However, since we do not have control of the server, we ended up skipping this step.

## Data Inconsistencies: Discarded and Unused Data

When we are loading the data, we found tones of invalid entries in all tables. We have previously discussed ISBNs inconsistencies and how we resolved them. However, there are other inconsistencies such as barcode referring to multiple Bib entries, or duplicate Bib entries. We solve the inconsistencies by skipping the loading of these rows. Since we pull all data in memory to parse and valid them first before sending to the database, the database constraints (discussed previously) acts as our test case for data integrity. For example, duplicate unique BIB entry keys would be rejected. Since we are able to load all data into both marmoset3 and marmoset4 server, we passed all data integrity test cases. However, there are still errors we are unable to fix or discover. For example, the author J.K Rowling has multiple spellings, and the publishers' names come in varieties with out without the .ltd or .inc suffix. Ideally, we would be able to check for these semantically, however, it is not possible without a additional datasets.

## Verifying Database with Additional Information

In additional to the previously discussed test scenarios, we also tested our data, the all-important review score data in particular, with additional data in the form of individual user rating. Goodreads dataset provides a snippet of reviews uniquely identified by both title information and Goodreads user ID (which is very useful for our recommendation algorithm, discussed later). We create the relationship table BooksUserRatings with unique keys on these attributes to store individual ratings, and a view called BooksUserRatingsSummary to aggregate the ratings by title (since multiple ISBNs refer to the same title, we assume that the ratings apply to every ISBN with the same title). By comparing average rating from the view and the average rating from the summary view directly from review totals, we can get an idea of whether our data is roughly correct. (not 100 percent because the user dataset is incomplete).

# The Client Application

The purpose of the client application is to helps users find the book they need without going to the library. It is important to note that the user will be ordinary readers, and not institutions (libraries), therefore we do not implement certain functionalities. For the full list of functionalities, please see the MANUAL.md. For this project, we implemented a range of pages (functionalities), searching, publishing (aka. creating a book), checkouts, availability checking, user recommendation and user review.

## Searching and Availability Checking

The ability to search books using multiple parameters can be very powerful. We implemented this feature using our very own custom query engine (with optional optimized joins and fuzzy matching) to allow users to quickly refine their search results within the multiple entities: books, authors, publishers, reviews. The same search functionality is also available for library inventories. Ideally inventories should be linked to the books database directly, so to allow a user to know the availability of an ISBN immediately on a search result. However, due to complications relating to poor performance and visual cluttering, we abandoned combining the two features.

Still, our search functionality is very flexible. For example, a user may look up a title by using both the author's title and review score, and publication date. By simply entering the title "database", specify a minimal review score of 4.1, and a publication date of 2015, the application will immediately return all results with the minimal score and relating to the concept "database". We could also allow user to specify maximal review scores, However, due to UI cluttering and time constraint, we decided to cut the feature.

## The Recommendation Engine

Another complex and powerful feature of the application is a recommendation engine, built entirely in SQL, that recommends books to a user based on what he or she has rated highly in the past (naturally, a user without any history cannot be recommended any books). . One of the challenges we faced is that popular books (rated highly by tons of irrelevant users) often overshadow books rated highly by users sharing similar taste. As a result, a user who has liked a certain book (Harry Potter) may only get recommendations based on that single book, and not say Macbeth.

To resolve this problem, we introduced a scoring system that weights the user similarity score against popularity score. The rating engine works by first finding other users who have a significant amount of overlap in book ratings with the current user. It assigns a similarity score (user taste score) to each of the other users based on the degree of overlaps of interests. Adding it to the popularity score of a book (the more similar users who liked the book, the higher it is). In essence, it is saying that a potential recommendation liked by one user who has a high degree of similarity to the current user (who seeks recommendation), is as important as, a recommendation that is liked a lot of users with very little similarity. Ideally, we would fine tune the weight between them (currently one + one), however we ran out of time.

Finally, the current SQL is very slow, measured in seconds. Ideally, we would create a materialized view to cache results for each user so they don't have to wait for a recommendation.

## Book Creation and Publication

We originally envisioned a user to be able to both create an ISBN book and add it to the library inventory; however, the latter part is cut due to time constraint. Currently, to publish a book, we allow the user to enter the book's ISBN, the author's name, the publisher's name, as well as other relevant information. We validate every user input and ensure the book the user is trying to publish does not yet exists in our database (by doing an ISBN lookup), and that the parameters are reasonable (ideally, we would have more checking). We also allow books with the same name to be published multiple times as long as they are of different ISBNs. Every time a book is created, the publishers, languages, authors, and books entities are optionally updated depending on the input.

## Other Functionalities

In the current state of our application, we allow a user to checkout a book by entering its BibNumber or ISBN. This will allow us to check whether we have any physical books on record, as well as each of their last checkout time. Ideally, we would know the return time of each copy, however, since we don't have the dataset, a user will have to make his or her own judgement on whether a book Is available (using the last checkout date provided by the application). Every time a checkout is confirmed, we add a new entry to the database.

We also allow users to add reviews. We check that the review score is within the valid range. Ideally once a user score is updated or created, the average score of the book will be updated using a View. However, since user score and rating counts are not linked in our system, this function is not yet implemented.

## Testing of Client Application

We primarily tested our client application using manual inputs. For example, we would test entering invalid ISBNs to see if they get rejected by the client, or would it make it to the database (never). We also tested that the appropriate success and error message is shown to the user on each user input. We tested to make sure pages redirection are correct. We tested to ensure the UI shows empty message when there are no results and reduce large results if there are too many returned.

Since large portions of SQL is generated by our own ORM, we want to validate the syntax and efficiency of these queries. We tested out SQL queries by logging the content to the console. We visually inspected these generated queries to ensure they match our intended use cases. We also make sure that the tables are joined in such a way that is optimal. Finally, on each database interaction - inserting new records and modifying existing ones - we validate that these records are inserted or modified by making a database query to find the changes.

# Data Mining

## Hypothesis

Do higher Goodreads ratings make a book more popular at the Seattle Library or vice versa? This question is important because it allows the library to predict the popularity of its books, and therefore ensure it has enough stocks for popular books and does not overstock books that people are no going to read.

## Method

We define popularity using the frequency of checkouts of a particular book (grouped by BibNumber). To ensure efficiency, we used a view called BibNumCheckoutSummary that maps each BibNumber to a frequency. The view is reused for each data point we computes. No additional indexes are needed for the checkout records, since they are keys already.

We first use the dataset of all checkout records from 2015 to 2017 (with T = 94124 after grouping) to compute the frequencies of checkouts of each library inventory item. We found the average number of checkouts to be 35, and the maximal number of checkouts to be 4015 (per Bib). In addition, we compute the define the following frequency categories to transform our continuous data into discrete sets.

| Popularity | Minimal Checkouts | Maximal Checkouts |
|---|---|---|
| Extremely Unpopular | 0 | 1 |
| Very Unpopular | 2 | 3 |
| Unpopular | 4 | 5 |
| Normal | 6 | 10 |
| Popular | 11 | 20 |
| Very Popular | 21 | 50 |
| Extremely Popular | 51 | INF |

Next, we evenly categorize Goodreads ratings into 4 categories, based on intervals of 1. Finally, we relate the Goodreads ratings to our checkout records using the combined key of ISBN13 and BibNumber, and we count the number of occurrences in each interval set.

## Results of Data-Mining

We found the 2-dimensional associativity Bayer's Index to be approximately 0.57, which indicates a weak to moderate correlation between the frequency of checkouts and Goodreads ratings. This is different from our expectations and indicates that our hypothesis may be incorrect, or that there are other factors at play here, for example, the subject of the book, and whether the book is mandated by the Instruction team.

| Library Checkouts (2015 - 2017) | Goodreads Ratings | | | |
|---|---|---|---|---|
| Popularity | 1 | 2 | 3 | 4 |
| Extremely Unpopular | 135 | 898 | 8221 | 4001 |
| Very Unpopular | 111 | 892 | 11776 | 5883 |
| Unpopular | 42 | 390 | 7205 | 3733 |
| Normal | 50 | 362 | 9046 | 5270 |
| Popular | 21 | 247 | 7071 | 4673 |
| Very Popular | 10 | 115 | 5971 | 4586 |
| Extremely Popular | 3 | 17 | 3898 | 4337 |

## Validation of Results

We perform validation using a different set of checkout records, this time ranging from the year 2013 to 2014. The smaller dataset produces a similar result with minimal deviation from our previous results. Which again confirms that our hypothesis may need to be re-thinked.

| Library Checkouts (2013 - 2014) | Goodreads Ratings | | | |
|---|---|---|---|---|
| Popularity | 1 | 2 | 3 | 4 |
| Extremely Unpopular | 123 | 886 | 8131 | 4177 |
| Very Unpopular | 117 | 872 | 11193 | 6026 |
| Unpopular | 58 | 401 | 6439 | 3476 |
| Normal | 55 | 385 | 8186 | 4621 |
| Popular | 22 | 255 | 6900 | 4133 |
| Very Popular | 17 | 179 | 6490 | 4415 |
| Extremely Popular | 4 | 50 | 4503 | 4138 |

# Project Installation Guide

## Prerequisites

Navigate to the project directory:

```
cd ./src
```

Install all dependencies:

```
pip install -r ./requirements.txt
```

## Loading Data

Download all CSV files to the following path:

```
./data/my_data_files.csv
```

Install all dependencies:

```
pip install -r ./requirements.txt
```

Run the loading scripts using python3:

```
python ./load_ILS.py
python ./load_goodreads.py
python ./load_reviews.py
python ./load_inventory.py
python ./load_checkouts.py
```

*Important:* the script must be run in the same order as specified above.

## Loading Data (Alternative)

Upload parsed CSV files to marmoset servers:

```
/var/lib/mysql-files/Group57/
```

In the mysql console run:

```
source scripts/load_data.sql
```

*This takes about 20 minutes per table to run on an unloaded server.*
*Do not wait till the marking deadline to create your version of the dataset.*

## Running the Client APP

Open PowerShell and install all dependencies:

```
pip install -r ./requirements.txt
```

Start the application using python3:

```
python ./application.py
```

Read the instructions located at:

```
./MANUAL.md
```

## Data Mining

To run the data mining SQL query run:

```
source scripts/data_mining.sql
```

The data is located in the folder

```
DM_results
```

# Client Instruction Manual

This user manual provides a basic understanding of the client application's capabilities.
See README.md for instructions on setting up the client application.

## Homepage

After starting the application, user will arrive at the homepage. A user will be presented with the following options:

```
lookup a book                   f
borrow a book                   b
publish a book                  p
review a book                   r
checkout a book                 c
recommand a book                e
exit                            q
```

The user may select any option by typing in their respective character key, then press ENTER. If the user failed to select a valid option, the application will prompt the user to try again. At any point in time, the user may quit the application by pressing CTRL C

## Looking Up A Book

One of the most common use cases of the client application is looking up a book from our database. This application allows users to mix and match multiple search criteria. Upon enter the lookup page, the user will be presented with the following options:

```
ISBN number                     i
book title                      t
author full name                n
publisher name                  p
publication year                y
publication month               m
publication day                 d
language code                   u
MIN number of pages             c
MIN average score               r
complete criteria selection     x
```

To specify a criterion, simply type the option character and press enter
For example, if the user want to search books by review score, he/she may enter R

```
Select criteria: r
```

Once an option is selected, the user will be asked to specify a valid condition. In the case of a review score, for example, only a floating point values will be accepted:

```
Enter MIN average score: 4.5
```

Once a search criteria is specified, the user may add additional search crinieres, or simply redefine his/her previous criteria. For example, if the user wants to also filter by title, the user may enter T, and then enter the title of the book:

```
Select criteria: t
Enter book title: database
```

For titles, the client will automatically perform fuzzy matching to provide better results. For certain other search criteria, the client will also perform automatic input conversions. For example, the client application will automatically convert ISBN10 to ISBN13.

Once the user has specified all criteria he/she needs, the user may press X to complete the criteria selection.
*note: at least one criteria must be entered to preform the search*

```
-------------------------------------------------------------------------------------------------------------------------
ISBN13         Title    Author              Publisher           Score   Language  Pages     Publication Dat
9780071613705  Applied Oracle Security: ...  David C. Knox       McGraw-Hill Educat  4.0     ENG       610       01 Oct 2009
9780072231304  Effective Oracle Database...  David C. Knox       McGraw-Hill Educat  3.69    ENG       512       08 Jul 2004
9780120887996  Moving Objects Databases      Ralf Hartmut Gütin  Morgan Kaufmann Pu  4.5     ENG       389       23 Aug 2005
9780130353009  A First Course in Databas...  Jeffrey D. Ullman   Prentice Hall       3.55    ENG       528       12 Oct 2001
9780201107159  Concurrency Control and R...  Philip A. Bernstei  Addison Wesley Pub  4.17    ENG       370       01 Jan 1987
9780201537710  Foundations of Databases:...  Richard G. Hull     Pearson             4.43    ENG       704       02 Dec 1994
9780321204486  Fundamentals of Database ...  Ramez Elmasri       Addison Wesley      3.81    ENG       1009      21 Aug 2003
9780470101865  Wiley Pathways Introducti...  Mark L. Gillenson   Wiley               3.56    ENG       478       01 Feb 2007
9780596002732  Access Database Design & ...  Steven Roman        O'Reilly Media      3.6     ENG       448       17 Jan 2002
9780976830221  Tera-Tom on Teradata Data...  Tom Coffing         Coffing Publishing  3.52    ENG       312       01 Nov 2004
-------------------------------------------------------------------------------------------------------------------------
<!> too many results, showing the first 10 matches...
```

*note: if the number of results returned exceeds 10, the application will only show the first 10 records*

## Reviewing A Book

A user may wish to add a review to a title. To do so, the user must first enter his/her interger userID. For example, 5400. After entering a userID, the user will be redirected to the lookup page to select one ISBN book. Once the book is selected, the user will be prompted to add a rating for the ISBN book:

```
did not like it                1
it was ok                      2
liked it                       3
really liked it                4
it was amazing                 5
```

Once a valid rating is added, the review will be published to our database.

## Borrowing A Book

A user may wish to borrow a book. To do so, the user must input either the library BibNumber or a valid ISBN number. Upon enter the book's identifier, the application will list the item's availability and location as well as the last checkout date of any physical copies of the book. For example:

```
-------------------------------------------------------------------------------------------------------------------
BibNumber           WHERE TO FIND IT              Type        Collection  Location    Report Date         Count
29747               Central Library, 1000 4TH AV  acbk        canf        cen         2017-09-01          1
-------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------------
Barcode  Last Checkout Time
1004459232017-08-08 12:16:00
-------------------------------------------------------------------------------------------------------------------
```

## Checking Out A Book

This is the final step to borrow a book. A user can check out any book with a physical barcode. The application will prompt the user for a valid barcode and upon receiving a valid input, the application will add a new checkout record to the database with checkout time set to the current time.

## Publishing Out A Book

This is designed for users who wish to add a new ISBN book to our database. After loading the page, the user will be prompted to enter various attributes of the ISBN book. The application will ensure the correctness of user inputs, in addition to the uniqueness of the ISBN edition. (In other words, if there already exists a record in our database, the user may not add a duplicate). For example:

```
lets start by entering some required information for your book...
Enter ISBN13 (or ISBN10): 9780552993692
Enter book title: paul ward's dumpster fire database
Enter author name(or enter nothing to skip): Jeff Zarnett
Enter publisher name(or enter nothing to skip): University of Waterloo
Enter number of pages: 2434
Enter language code (eg. ENG): ENG
Enter publication year: 2022
Enter publication month(or enter nothing to skip): 12
Enter publication day(or enter nothing to skip):
are you sure you want to publish the book paul ward's dumpster fire database ? (y/n):
```

## Recommending A Book

The application has the capability to recommend a user new books based on his/her existing ratings of books. The recommendation engine works by finding books that the user has not read before, and that is highly liked by users who share similar taste to the current user. Recommendation weighting depends both on the degree of overlap of the taste of similar users as well as the number of similar users who enjoyed a certain book that may be recommended. The application will only show the top 20 recommendations. Here is an example of the recommendation engine in action:

```
Enter your userId: 10986
here are some of the books you liked in the past, we will find recommendations based on these
-----------------------------------------------------------------------------------------------------------------------
ISBN13              Your Rating    Title
9780030957673       4              The Return of the Native
9780060172220       4              Cosette: The Sequel to Les Miserables
9780060467210       4              Adventures of Huckleberry Finn
9780099428640       5              The Trial
9780099511540       4              Heart of Darkness
```

```
9780140005295       5              Lord Jim
9780140009071       5              The Trial
9780140038835       4              Daisy Miller
9780140186222       5              The Trial
9780140281637       4              Heart of Darkness
-------------------------------------------------------------------------------------------------------------------
looking for recommendations, please wait...


:) based on books you really liked, here is our top recommendations:
-------------------------------------------------------------------------------------------------------------------
ISBN13         Title                       Author             Publisher           Score   Language  Pages     Publication
9780060740450  One Hundred Years of Sol...  Gabriel García Már  Harper Perennial    4.07    ENG       458       20 Jan 2004
9780141321097  The Adventures of Huckle...  Mark Twain         Puffin              3.82    ENG       466       06 Mar 2008
9780140012484  The Catcher in the Rye      J.D. Salinger      Penguin Books       3.8     ENG       220       01 Nov 1986
9780060173227  To Kill a Mockingbird       Harper Lee         HarperCollins Publ  4.28    ENG       323       01 Sep 1995
9780060735555  Slaughterhouse-Five         Kurt Vonnegut Jr.  Harper Audio        4.08    ENG       6         11 Nov 2003
9780075535751  The Brothers Karamazov      Fyodor Dostoyevsky Random House, Inc.  4.32    ENG       940       01 Sep 1950
9780140185546  Dubliners                   James Joyce        Penguin Books       3.85    ???       317       05 Nov 1992
9780176048136  Hamlet                      William Shakespear Thomson South-West  4.02    ???       208       02 Dec 2005
9780020198826  The Great Gatsby            F. Scott Fitzgeral Scribner            3.92    ???       193       01 Jun 1992
9780345294661  Fahrenheit 451              Ray Bradbury       Del Rey             3.99    ???       167       12 Feb 1981
9780007258055  Macbeth                     William Shakespear HarperCollins Publ  3.9     ENG       264       23 May 2007
9780140620658  King Lear                   William Shakespear Penguin Ltd.        3.91    EN-GB     160       28 Apr 1994
9780198319955  Othello                     William Shakespear Oxford University   3.9     ???       162       01 Jan 1996
9780571229116  Waiting for Godot           Samuel Beckett     Faber and Faber     3.83    EN-GB     87        05 Jan 2006
9780140447231  Anna Karenina               Leo Tolstoy        Penguin Books Ltd   4.05    ENG       864       06 Dec 2001
9780142437209  Jane Eyre                   Charlotte Brontë   Penguin             4.12    ENG       532       04 Feb 2003
9780020518709  The Sun Also Rises          Ernest Hemingway   Collier Books; Mac  3.82    ENG       247       01 Mar 1987
9780399502675  Lolita                      Vladimir Nabokov   Perigee Trade       3.89    ???       320       17 Aug 1972
9780140455366  The Odyssey                 Homer              Penguin Classics    3.77    ENG       416       29 Aug 2006
9780194228787  The Adventures of Tom Sawyer Nick Bullard      Oxford University   4.1     ENG       44        01 Jan 1999
-------------------------------------------------------------------------------------------------------------------
-----------------------------------------
```