CMPSC 431w

Phase 1

Arpit Singla

abs6339

Table of Contents

Table of Figure

TASK 1:

Analysis of system functionalities:

Customer Registration:

- Data needed: login name, password, first name, last name, address and phone number.

When a customer who wants to buy/ order a book, first needs to register an account in this database. So, this function collects all the appropriate information about the user and allows them to pick a login name and a password. Note that, it is after registering for an account that the customer can order books through our online bookstore.

- Constraint: it is required that each user has his own unique login name therefore we should use login name as primary key in our customer data table. So if a user tries to enter a login name that is already taken by another user then we will ask them to pick a different login name.

Multiple Managers:

- Data needed: login name, password

This function introduces a set of managers. So, as the system starts for the very first time, whoever wants to be the super user can register his account using customer registration functionality and he will automatically be given a privileged access and that first customer registration will be considered as the superuser who is the first manager. And after that this superuser can add login name from customer table to manager table so to give that user a privileged access. Note, if a user's login_name is present in the manager table then that is considered a privilege access.

- Constraint: In this database whoever is a manager is also a customer.

Ordering:

- Data needed: login name, orderID, ISBN, copies, totalCopies, totalAmount

After a new user/ new customer is done registering an account, now he can order books. Based on the price of the book, users can order books. We also keep a record of the

total number of books he ordered, total copies for each book and total amount of the books.

Note, other tables will get affected as well so, we will need to update the stock of each unique book after the order has been placed.

- Constraint: Users can only order books that are currently available in the inventory, so if some books have zero stock in the warehouse then the users will not be able to order. Now, a user can order one or more books and even multiple books one or more times.

New book:

- Data needed: ISBN, title, author(s), publisher, language, publication date, number of pages, price, keywords, subject, copies arrived

When a new book arrives in the inventory, the bookstore manager can record the details of the new book and even note the number of copies of the books that arrived. If the book name is already present in the database, then we will only update the stock level but if not then we can fill in all the details mentioned above.

- Constraint: the database will check the user power, only if 'power == 1', we will allow him to add the book or update the stock otherwise he will not be allowed to do this.

Arrival of more copies:

- Data needed: ISBN, stock updated

On the arrival of more copies of a book, we only need to update the stock level of those books.

- Constraint: if the book is present in the database then it will update the stock level, if not then it will make use of new book functionality and add the book details in the database.

Comments:

- Data needed: login name, date, score, short text, ISBN

A user can write a comment on a book and give it a score which ranges from (0 to 10). Note that the writing the short text in a comment is optional but if some user do chooses to write a comment then he will have to give it a numerical score.

- Constraint: a user is allowed to write only one comment for each book.

Usefulness ratings:

- Data needed: login nameA, login nameB, usefulness score

Here the users can access other user's comments for each book and give it a usefulness score. The usefulness score has three levels namely 'useless', 'useful' and 'very useful'.

- Constraint: User cannot rate his own comment this will be achieved while implementing the database where we will not allow 'login nameA' equal to 'login nameB'. This way a user can't provide a usefulness rating for his own comments.

Trust recordings:

- Data needed: login nameA, login nameB, trusted or not

This is a simple function where a user can declare other users as trusted or not-trusted. If a user trusts another user then he gets '1' in the ….. database, if not trusted then he will get 0

- Constraint: Note it does not make sense to declare yourself as trusted or not-trusted therefore when implementing the database, we will not allow 'login nameA' equal to 'login nameB'. This way a user can only declare other users as trusted or not.

Book browsing:

- Data needed: author(s), publisher, title, language, publish date, average score of comments and average score of trusted user

Here we will make use of the new book table and comments table where in a search box, the users can search the books by choosing the author, publisher, title or language and results will be books and relevant information stored in the book database. Users can then sort those results on the basis of publish date, average score of comments and average score of trusted user.

Useful comments:

- Data needed: ISBN, score from comments

When a user wants to read the top n most useful comments for a specific book, he will make use of this function and specify the book ISBN and number of comments 'n' he wants to see. Note, we are making use of the existing tables i.e. comments table and using aggregation we will be able to fetch the top n most useful comments from the database.

Buying suggestions:

- Data needed: login name, books ordered

When a user orders a book then the bookstore database makes use of other orders where users bought the same book and note which other books did they also bought. This way the database can recommend books to the users to buy. Now, on the basis of what the user bought, we will look for orders containing the same book and even note which other books were bought and then we can sort them according to the decreasing sales amount.

Degrees of separation:

- Data needed: new book table, specified author name

Using the book data, where we have stored the information about all the books available in the bookstore, to find 1-degree separated authors we just need to match the names of each books authors with other book's authors. Now, to find 2-degree separated authors lets filter out the authors who have written more than or equal to 2 books. Now let's say, we have two books A and B. A is written by 'C, D' while B is written by 'C, E'. Now first we will take the common author in the books i.e. 'C', then we will take all other authors except C, i.e. 'D, E' which is what I will return as a result of 2-degree separated authors.

Book Statistics:

- Data needed: Ordering table, new book table

At the end of each semester, the store manager can use the database to list out the most popular books, most popular authors and most popular publishers all in terms of copies sold. We will make use of ordering table to know the most popular book as we can count the copies sold for each book. To find the most popular author, we can again make use of the ordering table and new book table to see which authors have sold the most number of books and likewise we can find the 'm' most popular publishers.

User awards:

- Data needed: trust recording table and usefulness ratings

Manager can make use of this function by just inputting the number of most trusted users and most useful users he wants to give best users award to. To find the top m most trusted users, we will make use of the trust recording table and use aggregations to find the

top m records likewise using usefulness ratings table we can find the top m most useful users.

Extra Functionalities:

Get order history:

- Data needed: login_name

This is an additional system function which the users can use to see their order history. They can press on a button 'get order history' and using the order data and book order data we can show their order history. If no order has been placed, then we will simply return 'you haven't placed any orders yet.'

Order delivery:

- Data needed: orderID, login name, order status

This is an additional system function where we are keeping track of delivery of orders. When a user places an order, automatically it is set up for the delivery to their address provided. Users can go to their orders and look for order status, where it will show if its delivered or not delivered.

- Constraint: Note, we are assuming here that the whole order will be delivered in one go i.e. because the user can only order books that are available in the inventory so all the books will be delivered in one package instead of different packages.

Book availability reminder:

- Data needed: login name, ISBN, remind me checked or not, notified successfully as book status is available

Because users might find it difficult to check again and again the availability of a specific book(s), they can make use of this functionality, here users can sign up to 'remind me' by saying 'yes' and our system can make use of their phone number to send a message of book availability. Now, as the stock of that specific book comes in, all the users who signed up for the service will be notified about the book status.

Delete book:

- Data needed: login name, ISBN

Upon the end of a semester, if the manager thinks that some books are not selling and therefore they don't want that book to available anymore. Then he can use this function to delete the book from the new book table. After doing this, that book's stock in the inventory will not be their anymore. So we are assuming that some imaginary vendor will buy all those books back.

Note, only if zero copies of a certain book are sold, then only the manager will delete the book or else not.

Request a book:

- Data needed: login name, ISBN

If users are not able to find a specific book that they are looking for on my online bookstore, then they can put a request in the system notifying the managers about the book they want which they think should be available on our bookstore. We are assuming that if a book gets requested then the managers will definitely add it to the book store and they can ask some imaginary vendor for the stock.

Note that we are only asking the user to input the ISBN number of the book so we are assuming that every book has an ISBN number. And because this number is unique for each book, it is sufficient information to identify this book uniquely.

Reselling a book back to the bookstore:

- Data needed: resellID, login name, ISBN, quantity

This function allows users to sell back their books to the bookstore in return of max 50% of the original price. Note the user can't sell books to other users through this platform but only to bookstore itself.

Here, the user will only input the ISBN of the book and 50% of the original price will be given to the user. Note we could have asked for book's condition but that will make the design a little more complicated. So to keep it simple each resold book by a user will be considered new in condition same as newly arrived stock.

Also note that only books that are already been registered in the system i.e. only the books that are currently being sold by the bookstore will be accepted so if some book is not sold by the bookstore then that book will not be accepted for this resell proposition.

We should also take note of the primary key, as users can resell multiple books and each book multiple times (like if they have same ISBN book but multiple copies of it) then resellID is a good way to differentiate each record.

Now, because it is same as a freshly arrived stock then we will need to update the stock of this book.

Tables:

Book data: In this database table, we store information about books i.e. ISBN, book_title, authors, publisher, language, publication_date, number_of_pages, price, keywords, subject, current_stock. This information can be accessed by both customers and managers. But, only managers can change the information in the tables and customers can only view the information. This is done by checking if this user's login_name is present in the manager data table or not. If it is present, then they are considered manager and can change the information of the book otherwise not. The ISBN is the primary key for this table.

There can be multiple authors for each book and each author can write many books so there is a many to many relationship between books and authors. since a book can have many authors, we can remove the authors column from this book data table and create another table called authors data which contains ISBN and authors, this can be done using a join operation.

Customer data: In this database table, we store information about customers i.e. login_name, password, first_name, last_name, address and phone_number. The primary key is login_name for this table i.e. every customer will have to select unique login_name. As the customers will order as well, so for that we will create a new table called order data which will contain information about customer's orders.

Manager data: In this database table, we store information about the customers that are given the access of a manager. Here, we store login_name, password. It's the super user that will give add the users that should be given manager access to this database table. Here, we are using login_name as the primary key. So, every time a user (customer, manager) tries to access functions that can only be accessed by a manager, we will check if his login_name is present in the manage data table or not. So if the login_name is present in the manager data then they can access all the functions otherwise they have restricted access.

Comments data: In this database table, we are storing information about the customer's comments on each book. So we are storing login_name, ISBN, date, short_text, score, useless, useful, very useful. We are using login_name and ISBN as the primary key. Note, here we have many to many relationship between login_name and ISBN as there can be multiple comments for each ISBN and each login_name i.e. each customer can comment on multiple books. Note that a customer can only comment ones on each book. If a customer wants to write a comment then writing a short_text is optional but giving it a score is compulsory. Moreover, we are only storing the total usefulness ratings given in each category i.e. useless, useful, very useful.

Order data: In this database table, we store information about the orders placed by customers. So we store data like, login_name, orderID, totalCopies, totalAmount, delivery status. Here, orderID is the primary key. Since, a customer can order multiple times, therefore there is a one to many relationship between login_name and orderID. TotalCopies is the total number of book's copies that the customer ordered. Customers can use delivery status attribute to check if their order is delivered or not.

Authors data: In this database table, we store information about each book's authors. Because there can be multiple authors of a book, there will be multiple records with each ISBN so to store each authors name in different row. This is done to have atomicity in that table and achieve flat table. So, this table will store, ISBN and authors. Therefore, we are using ISBN and authors as the primary key as well. Note, we have a one to many relationship between ISBN and authors.

Task 2

ER Diagram

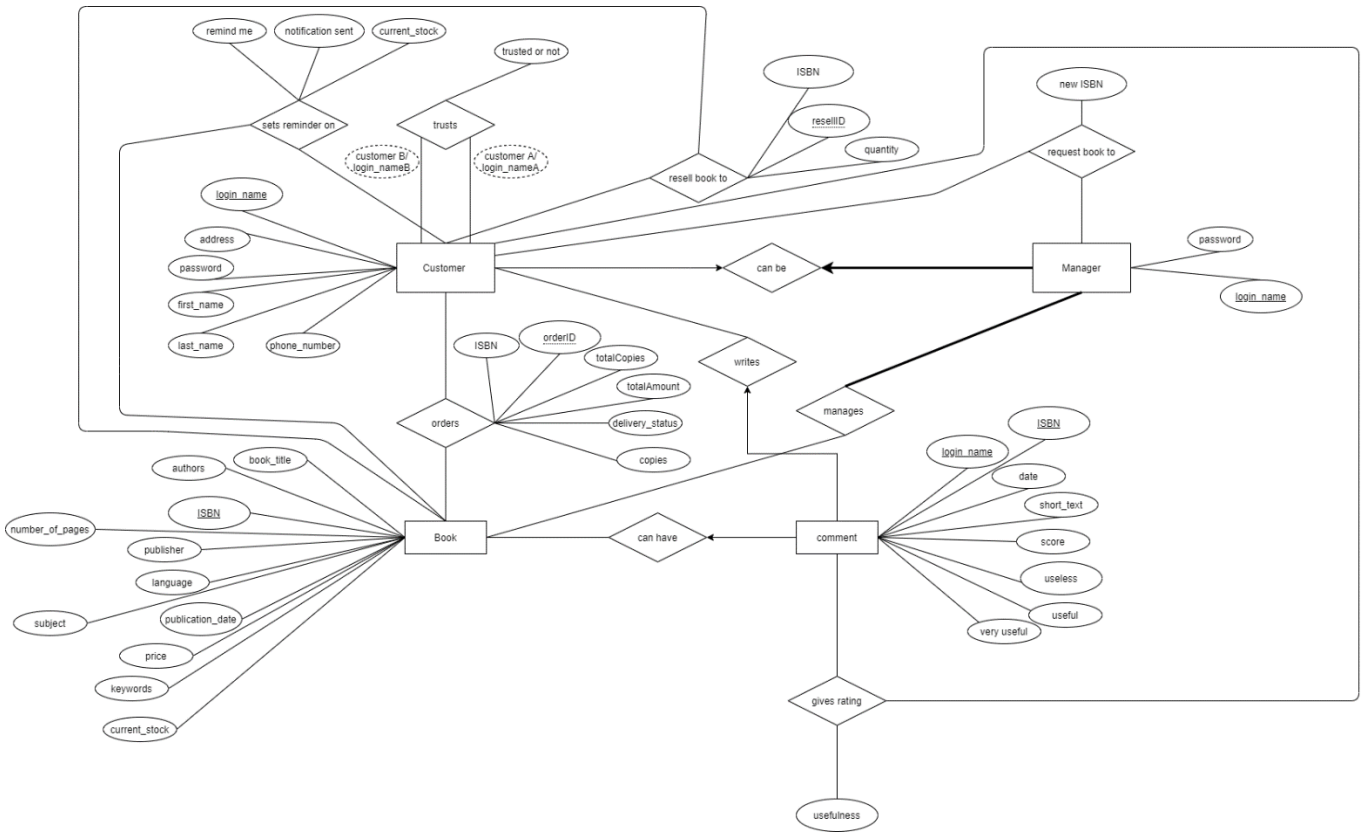

Figure 1, The ER-diagram of the project

Entities and relationships:

Customer: It is the important entity set in our database design. It has total of 6 attributes i.e. login_name, address, password, first_name, last_name and phone_number. The primary key for this entity set is 'login_name'. In the ER diagram above, we notice it forms 8 relationships in total. Out of 8, three binary relationships are formed between customer and book. First being the relation 'orders', which says customers can place order of books wherein it is a many to many relationship. The relation has its own set of descriptive attributes. So, orders relation contain all the information about the orders placed by the customers.

The other is the relation ‘sets reminder on’, which allows the user to set a book availability reminder. So if a book is available then we will send a notification to the customer’s phone number. ‘resell book to’ is a relation as well which allows the user to sell their books back to the bookstore. Note, all these three relationships are many to many.

One of the major relationship is of the customer and manager which is named ‘can be’. Here it says that a manager is always a customer and there has to be at least one manager who has to be a customer at any moment of time in the database. Note that it’s a 1-1 relationship as a manager’s account can only be connected to one customer account and vice versa. Theirs one other relationship between customer and manager i.e. ‘request book to’ where a customer can request a book to available in the bookstore to the manager. As it’s the manager that manages the bookstore therefore we are requesting him through this relationship.

Customer also forms a relationship with itself called the ‘trusts’, where it’s also a many to many relationship and says that customer_A can trust customer_B where customer_A gives a ‘trusted or not’ recording to the customer_B. Customer has also the liberty to write a comment which is achieved using ‘writes’ relationship. Note that this is a one to many relationship, as a customer can write many comments (each comment on different book) but each comment belongs to only one customer.

The last relation that customer forms is ‘gives rating’ with comment is explained below in **comment entity set.**

Note: because there is no representation of foreign keys in ER diagram, therefore it is noted down in task 4 where database schema is being formed.

Manager: in this entity set, there are a total of 2 attributes i.e. login_name and password. Note, login_name is the primary key. And it forms a major relationship with book entity set i.e. ‘manages’, where this relationship allows manager to add book, remove book, update stock and basically manage the bookstore inventory information (book table) which has all the information on books in the inventory. Note there is a bold line from manager to manages relation, because at any given time in the database there has to be at least one manager who is managing the books.

Book: this entity set is the core of the online bookstore as it contains information about all the books stored in the inventory. It has a total of 11 attributes i.e. ISBN, book_title, authors, publisher, language, publication_date, number_of_pages, price, keywords, subject, current_stock. Here ISBN is the primary key. Book forms a relation ‘can have’ with comment

which is a one to many relationship where book can have multiple comments but each comment can only belong to only one book.

Comment: this entity set has a total of 8 attributes which are login_name, ISBN, date, short_text, score, useless, useful, very useful. It has login_name and ISBN as its primary key. Note it forms 3 relationships in total and two of them are explained above one with book and the other with customer entity set. The third relationship is formed with customer as well i.e. 'gives rating'. Now each customer gives usefulness rating to comments. Note 'useless', 'useful' and 'very useful' are attributes of comment which only contains the total number of customers that gave this comment these specific types of ratings. But with this relationship, we are storing individual usefulness rating by a specific customer given to a specific comment. It's a many to many relationship as well because each customer can give many ratings to many comments and each comment can receive multiple ratings from multiple customers.

Task 3

In this task, we need to finalize the front end framework to be used for phase 2. We also need to check which framework would be best for back end work. And finally we need to decide upon a RDBMS system.

So, starting with front end, there are many frameworks and all have their pros and cons. Here, I am interested in comparing react, angular and bootstrap.

React.js is a javascript library for building user interfaces. React is used to build single page applications and it also allows us to create reusable UI components. Because it is declarative, it is easier to debug and the code is predictable as well. React, interestingly, combines the UI and behavior of components. In React, the same part of the code is responsible for creating a UI element and dictating its behavior.

Pros:

1. It's easier to learn
2. It helps to build rich user interfaces
3. Offers wide range of JS libraries
4. It is known for its speed and component based coding

Cons:

1. Because it uses JSX i.e. JS with an extension of HTML, it can be a little complex to learn.
2. Poor documentation
3. It offers limited support for third party applications to connect with it.

Angular on the other hand is a development platform that uses TypeScript instead of JS. Note, JS comes under TS. It's a leading frond end framework so its updated quite frequently which may add to the learning curve. But it is also used to build single page applications and it has a lot of packages inbuilt in its framework which makes everything easier to implement. So need to use third party libraries.

Pros:

1. Uses TS instead of JS,
2. Built in packages
3. Well documented information
4. Follows MVC architecture to build code (easier to understand the development process

Cons:

1. But steep learning curve
2. Complex to understand concepts and libraries
3. Complex debugging as well
4. Performance is not better then react

It feels like, react is much better then angular in some ways. Because its learning curve is not steep and is therefore easier to understand overall. It also has a lot of JS libraries to implement components and different functions. One thing to note here is that react allows developers to reuse the components in different fashion and because it supports isolated components. It really speeds up the development process.

So, I will be going forward with react js.

As far as back end is concerned, we have a lot of options, such as Django and flask.

Django is a python based framework that enables fast development of websites and is built so that developers have easy way to connect with online databases, which is exactly what this assignment requires. A very well documented framework, widely used and easy interface as well. It can work with any client-side framework and can deliver content in any format like HTML, XML, JSON

Pros:

1. Build for easy development with databases
2. Well documented documentation
3. Easy to work with interface
4. Since it's a python framework so easy for me as I have already learned python

Cons:

1. Difficult to debug as it makes use of regex
2. Performance is not so great when compared to its competitors
3. Not good for small projects

Flask on the other hand is a micro web framework written in Python. It is classified as a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself.

Pros:

1. Light weight python framework
2. Way faster then Django
3. Offers highly sophisticated third party libraries
4. Debugging is easier

Cons:

1. Security is an issue with it
2. Not so well documented information
3. It's a little hard sometimes to connect to other frameworks

It seems Django for me is a wiser option as I have used it once before and it has built in libraries instead of third party and it is specially designed to do development with online databases and frameworks. And Django does a lot of the things automatically like provides the developer with UI for managing the data.

Like frontend and backend frameworks, there are lot of options to maintain DMBS as well. Such as MySQL, SQLite.

MySQL as we all know is an open source SQL based RDBMS system i.e. to manage relational database.

Pros:

1. High performance
2. Easy installation process
3. Easy to use as well

Cons:

1. Its highly depends on third party add ons for additional features
2. Doesn't support functions such as INTERSECT.
3. Limited support for fault tolerance

SQLite is a relational database management system contained in a C library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

Pros:

1. Best for small projects
2. High performance based

3. Server less deployment

Cons:

1. Performance hits with large database
2. Complex to connect with the data over network

I am going for MySQL because it is really easy to use and easy to deploy as well.

Conclusion, I will be developing this project using react.js, Django and MySQL.

Task 4

In this section, we are converting the ER diagram in task 2 to a schema and then we will normalize this schema.

Note: final schemas are represented by DDL queries in bold i.e. after considering schema refinement.

Schemas for entity sets:

Customer:

First we can make schema for all the entity sets. So 'customer' has schema:

Create Table customer(

login_name: varchar(20) NOT NULL PRIMARY KEY,

password: varchar(20) NOT NULL,

first_name: varchar(20) NOT NULL,

last_name: varchar(20) NOT NULL,

address: varchar(100) NOT NULL,

phone_number: char(10) NOT NULL);

In customer table, login_name is the primary key. It doesn't make sense to decompose this table as it is important that we store all the information about each customer in one table. decomposing it because of BCNF will only make its performance low. So above is the final schema for customer table.

Manager:

Next we have entity set, 'manager' which has schema:

Create table manager(

login_name: varchar(20) NOT NULL PRIMARY KEY,

password: varchar(20) NOT NULL

FOREIGN KEY (login_name) REFERENCES customer(login_name));

Note login_name is the primary key for this table but it is a foreign key referring to customer table. FK is needed because a manager has to be a customer first. Again, there is no need of schema refinement here.

Book:

For the book table, we have schema: book(ISBN: char(13), book_title: varchar(50), author: varchar(50), publisher: varchar(50), language: varchar(30), publication_date: int, number_of_pages: int, price: int, keywords: varchar(50), subject: varchar(50), current_stock: int). 'ISBN' is the primary key and char(13) is the domain because exactly 13 characters define each book sold in the bookstore uniquely. We know that there can be multiple authors and keywords for a book therefore using 1NF, we have decomposed the table into three.

Create table book(

ISBN: char(13) NOT NULL PRIMARY KEY,

book_title: varchar(50) NOT NULL,

publisher: varchar(50) NOT NULL,

language: varchar(30) NOT NULL,

publication_date: int NOT NULL,

number_of_pages: int NOT NULL,

price: int NOT NULL,

subject: varchar(50) NOT NULL,

current_stock: int NOT NULL);

ISBN is the primary key.

Create table book_Author(

ISBN: char(13) NOT NULL,

author: varchar(50) NOT NULL,

PRIMARY KEY(ISBN, author)

FOREIGN KEY (ISBN) REFERENCES book(ISBN)

);

ISBN and author is the primary key and ISBN is also the foreign key referring to ISBN in book table.

book_Keyword(

ISBN: char(13) NOT NULL,

keyword: varchar(50) NOT NULL,

PRIMARY KEY(ISBN, keyword),

FOREIGN KEY (ISBN) REFERENCES book(ISBN));

ISBN and keyword is the primary key and ISBN is also the foreign key referring to ISBN in book table.

Now, we have final schemas. We don't need to decompose book any further due to performance issues. It's important that we keep the information about a book in the same table because it won't make sense to store them in different tables.

Comment:

For the comment table, we have schema:

Create table comment(

login_name: varchar(20) NOT NULL,

ISBN: char(13) NOT NULL,

date: int NOT NULL,

short_text: varchar(200),

score: int NOT NULL,

useless: int,

useful: int,

very useful: int,

PRIMARY KEY(login_name, ISBN),

FOREIGN KEY (login_name) REFERENCES customer(login_name)

FOREIGN KEY (ISBN) REFERENCES book(ISBN));

Primary key for it is login_name and ISBN. Note, login_name is the FK referring to table customer and ISBN is also the FK referring to table book. We are storing the total customers who find the book useless, useful and very useful, because at every moment of time we need to show the comments information and if we don't calculate it now then we will need to query the database to calculate individuals who gave the rating to a comment. No need of schema refinement as it will only complicate the things.

Schemas for relationships between entity sets:

Now, we start with the converting the ER diagram of relationships between entity sets.

Orders:

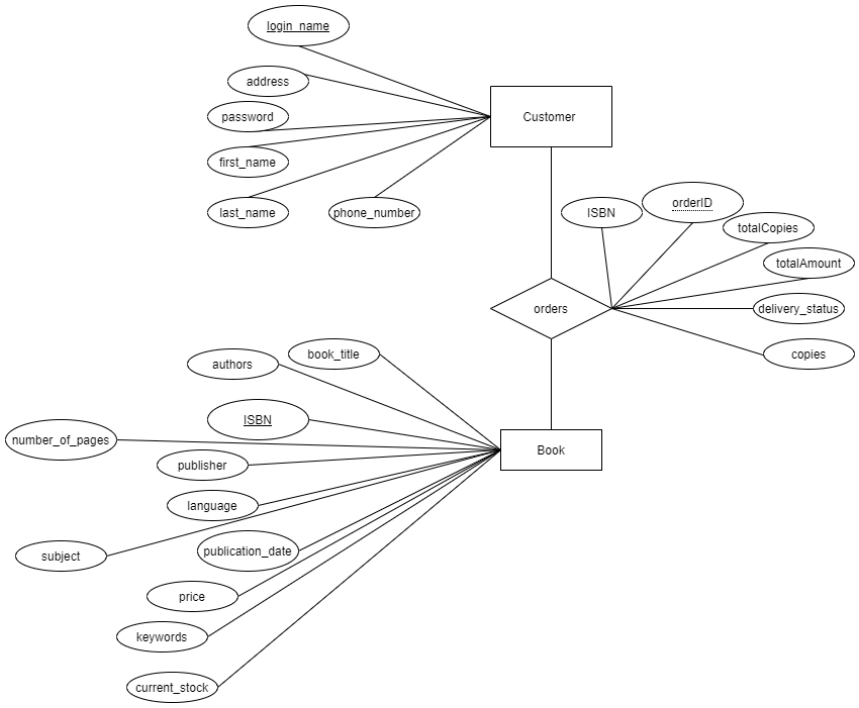

Figure 2, The ER-diagram of the relation orders

For the above figure we have a relationship 'orders' between customer and book. So, the schema for the relation is:

orders(login_name: varchar(20) , ISBN: char(13), orderID: int, copies: int, totalCopies: int, totalAmount: int, delivery_status: int)

Primary key is login_name, ISBN and orderID. FK is login_name referring to table customer and ISBN referring to table book.

Now, after doing closure of all the attributes in orders table, we find a FD:

{orderID} → {login_name, totalCopies, totalAmount, delivery_status}

This is a bad FD because orderID is not the key that is $\{orderID\}^+$ closure is not equal to all the attributes of the table order. Therefore, we use this FD to decompose the table order into two different tables:

Create table OrderInfo(

login_name: varchar(20) NOT NULL,

orderID: int NOT NULL AUTO_INCREMENT,

totalCopies: int NOT NULL,

totalAmount: int NOT NULL,

delivery_status: int NOT NULL,

PRIMARY KEY(login_name, orderID),

FOREIGN KEY (login_name) REFERENCES customer(login_name));

login_name and orderID is the primary key. login_name is also the FK referring to customer table.

Create table bookOrderInfo(

orderID: int NOT NULL AUTO_INCREMENT,

ISBN: char(13) NOT NULL,

copies: int NOT NULL,

PRIMARY KEY(orderID, ISBN),

FOREIGN KEY (orderID) REFERENCES orderInfo(orderID),

FOREIGN KEY (ISBN) REFERENCES book(ISBN));

OrderID and ISBN is the primary key and ISBN is the FK referring to book table, orderID is the FK referring to orderInfo table.

‘copies’ attribute stores number of copies ordered for a specific book by a specific customer and ‘totalCopies’ attribute stores the total number of book’s copies ordered given the specific orderID. And if you observe storage space is being saved as redundancy has been reduced.

Sets reminder on:

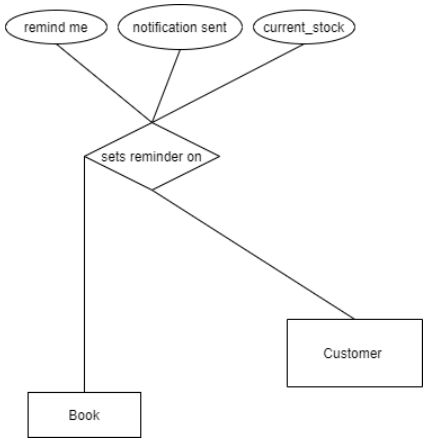

Figure 3, The ER-diagram of the relation ‘sets reminder on’

For this relation, we have schema:

sets_reminder_on(login_name: varchar(20), ISBN: char(13), current_stock: int, remindme_ON: int, notification_sent: int);

‘Login_name, ISBN’ is the primary key where login_name is FK referring to table customer and ISBN is also FK referring to table book, current_stock is FK as well referring to book table.

Again, we have found a FD:

{ISBN} → {current_stock, notification_sent}

This is a bad FD and if we use this FD to decompose the table in two then we will save storage. Reason being, we are storing the current_stock attribute again and again while many customers have set reminder for the same book. Even the notification_sent attribute, using the decomposition table we will see that we only need store it once per book and not for all the records as before. So decomposing the schema:

Create table Sets_reminder_on(

login_name: varchar(20) NOT NULL,

ISBN: char(13) NOT NULL,

remindme_ON: int NOT NULL,

PRIMARY KEY(login_name, ISBN),

FOREIGN KEY (login_name) REFERENCES customer(login_name)

FOREIGN KEY (ISBN) REFERENCES book(ISBN));

'login_name, ISBN' is the primary key. login_name is the FK referring to table customer and ISBN is the FK referring to table book.

Create table Reminder_notificationInfo(

ISBN: char(13) NOT NULL,

current_stock: int NOT NULL,

notification_sent: int NOT NULL,

PRIMARY KEY(ISBN),

FOREIGN KEY (ISBN) REFERENCES book(ISBN),

FOREIGN KEY (current_stock) REFERENCES book(current_stock));

'ISBN' is the primary key and it is also the FK referring to table book and current_stock is also the FK referring to table book.

Resell book to:

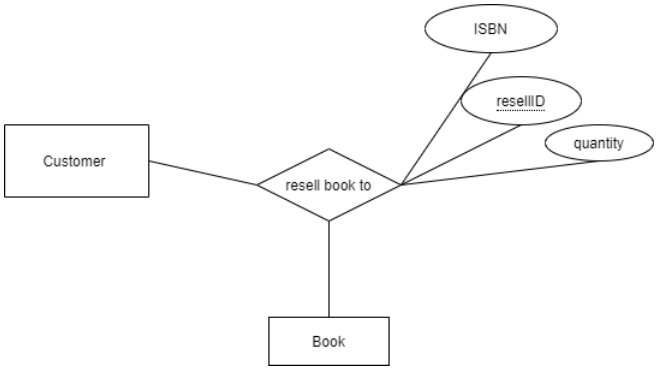

Figure 4, The ER-diagram of the relation 'resell book to'

For this relation, we have schema:

Create table resell_book_to(

resellID: int NOT NULL AUTO_INCREMENT,

login_name: varchar(20) NOT NULL,

ISBN: char(13) NOT NULL,

quantity: int NOT NULL,

PRIMARY KEY(login_name, ISBN, resellID),

FOREIGN KEY (login_name) REFERENCES customer(login_name)

FOREIGN KEY (ISBN) REFERENCES book(ISBN));

'Login_name, ISBN, resellID' is the primary key where login_name is FK referring to table customer and ISBN is also FK referring to table book.

The schema looks fine as it is, so there is no need to apply schema refinement on it.

Request book to:

For this relation, we have schema:

Create table request_book_to(

customer.login_name: varchar(20) NOT NULL,

manager.login_name: varchar(20) NOT NULL,

new_ISBN: char(13) NOT NULL,

PRIMARY KEY(customer.login_name,new_ISBN),

FOREIGN KEY (customer.login_name) REFERENCES customer(customer.login_name)

FOREIGN KEY (manager.login_name) REFERENCES manager(manager.login_name));

'customer.Login_name, new_ISBN' is the primary key where login_name is FK referring to table customer. 'manager.login_name' is also a FK referring to table manager.

Note when a customer requests for a book by providing ISBN, request will be sent randomly to any manager in the database. Moreover, just to clarify again we are assuming that given an ISBN for a book, the managers can find the book information and make the book available in the bookstore.

No need of the schema refinement here.

writes:

For this relation, we have schema:

Create table writes(

customer.login_name: varchar(20) NOT NULL,

comment.login_name: varchar(20) NOT NULL,

ISBN: char(13) NOT NULL,

PRIMARY KEY(customer.login_name,ISBN),

FOREIGN KEY (customer.login_name) REFERENCES customer(customer.login_name)

FOREIGN KEY (comment.login_name) REFERENCES comment(comment.login_name)

FOREIGN KEY (ISBN) REFERENCES book(ISBN));

‘customer.Login_name, ISBN’ is the primary key where login_name is FK referring to table customer, ISBN is also a FK referring to book and comment.login_name is also a FK referring to table comment.

Even though it is a one to many relationship, so we might be thinking about combining the table ‘writes’ with ‘comment’, but because comment table forms a many to many relationship with customer table it is not a good idea to combine them as it destroys the relationship with the customer in a way that customer is not able to refer to the data it could before combining.

Therefore, no need of the schema refinement here.

Can have:

For this relation, we have schema:

Create table Can have(

book.ISBN: char(13) NOT NULL,

comment.login_name: varchar(20) NOT NULL,

comment.ISBN: char(13) NOT NULL,

PRIMARY KEY(customer.login_name,ISBN),

FOREIGN KEY (customer.login_name) REFERENCES customer(customer.login_name),

FOREIGN KEY (comment.login_name) REFERENCES comment(comment.login_name),

FOREIGN KEY (ISBN) REFERENCES book(ISBN));

‘customer.Login_name, ISBN’ is the primary key where login_name is FK referring to table customer, ISBN is also a FK referring to book and comment.login_name is also a FK referring to table comment.

Again we see a one to many relationship but because of reasons discussed above in ‘writes’, we won’t be combining the tables ‘can have’ and ‘comment’.

So, no need of schema refinement here.

Gives rating:

For this relation, we have schema:

Create table Gives rating(

customer.login_name: varchar(20) NOT NULL,

comment.login_name: varchar(20) NOT NULL,

ISBN: char(13) NOT NULL,

Usefulness: int NOT NULL

PRIMARY KEY(customer.login_name, comment.login_name, ISBN),

FOREIGN KEY (customer.login_name) REFERENCES customer(customer.login_name),

FOREIGN KEY (comment.login_name) REFERENCES comment(comment.login_name)

FOREIGN KEY (ISBN) REFERENCES comment(ISBN));

'customer.Login_name, comment.login_name, ISBN' is the primary key where customer.login_name is FK referring to table customer, comment.login_name and ISBN is also a FK referring to comment table.

Here, customer.login_name is the customer who gives usefulness rating to comment.login_name's comment.

No need of schema refinement here.

trusts:

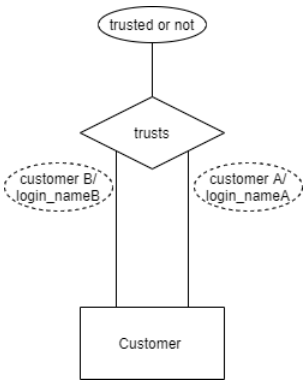

Figure 5, The ER-diagram of the relation trusts

For this relation, we have schema:

Create table trusts(

customerA.login_name: varchar(20) NOT NULL,

customerB.login_name: varchar(20) NOT NULL,

trusted_or_not: int NOT NULL,

PRIMARY KEY(customerA.login_name, customerB.login_name),

FOREIGN KEY (customerA.login_name) REFERENCES customer(customerA.login_name),

FOREIGN KEY (customerB.login_name) REFERENCES customer(customerB.login_name));

'customerA.login_name, customerB.login_name' is the primary key where customerA.login_name and customerB.login_name is FK referring to table customer.

Here, customerA.login_name is the customer who trusts the customerB.login_name.

No need of schema refinement here.

References:

“Flask (Web Framework).” *Wikipedia*, Wikimedia Foundation, 10 Mar. 2021, en.wikipedia.org/wiki/Flask_(web_framework).

KnowledgeHut. “Advantages and Disadvantages of Angular.” *Knowledgehunt*, 18 July 2020, www.knowledgehut.com/blog/web-development/advantages-and-disadvantages-of-angular.

Team, DataFlair. “Django Advantages and Disadvantages – Why You Should Choose Django?” *DataFlair*, 24 Mar. 2021, data-flair.training/blogs/django-advantages-and-disadvantages.

“The Web Framework for Perfectionists with Deadlines | Django.” *Djangoproject*, 2019, www.djangoproject.com.

“Angular Pros, Cons, Features, and More | Pluralsight.” *Pluralsight*, 12 June 2019, www.pluralsight.com/blog/software-development/angular-101.

Editor. “The Good and the Bad of Angular Development.” *AltexSoft*, 27 Feb. 2020, www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development.

“Angular 7 Tutorial - Javatpoint.” *Www.Javatpoint.Com*, 2019, www.javatpoint.com/angular-7-tutorial.

“Access Denied | Www.Codeinwp.Com Used Cloudflare to Restrict Access.” *Codeinwp*, 2020, www.codeinwp.com/blog/angular-vs-vue-vs-react.

“React – A JavaScript Library for Building User Interfaces.” *React*, 2018, reactjs.org.