COMP 3005 Winter 2020 Project

Sharjeel Ali

March 28th, 2020

# Conceptual Design

The idea of the book store is

# 2. Reduction to Relation Schemas

Relations:

1. *author(id, name)*
2. *genre(id, name, type)*
3. *publisher(id, phone, email, address\_id, transit\_number, account\_number, institution\_number)*
4. *book(isbn, title, description, price, page\_count, published\_date, add\_date, rating, rating\_count, sale\_percent, author\_id, genre\_id, publisher\_id)*
5. *orders(order\_number, username, order\_date, status\_id, card\_number, address\_id)*
6. *status(status\_id, name, description)*
7. *order\_book(order\_number, isbn, warehouse\_id, quantity)*
8. *cart(username, isbn, quantity)*
9. *client(username, email, first\_name, last\_name, password)*
10. *admin(email, first\_name, last\_name, password)*
11. *sales\_report(report\_number, admin\_email, start\_date, end\_date, file)*
12. *report\_transaction(report\_number, transaction\_id)*
13. *transaction(transaction\_id, transaction\_name, transaction\_type, amount, date)*
14. *warehouse(id, address\_id)*
15. *warehouse\_books(warehouse\_id, isbn, stock)*
16. *address(id, country, state, city, code, street, apt\_number)*
17. *client\_address(username, address\_id)*
18. *payment\_info(card\_number, name, expiry\_date, security\_code)*
19. *client\_billing(username, card\_number)*
20. *restock(restock\_number, isbn, warehouse\_id, quantity, restock\_date)*
21. *request\_book(request\_number, username, request\_name, request\_isbn, date)*
22. *admin\_decides(email, request\_number, decision, date )*
23. *history(username, isbn, rank)*

test

1. *order\_book(order\_number, isbn, warehouse\_id, quantity)*
2. *warehouse(id, address\_id)*
3. *warehouse\_books(warehouse\_id, isbn, stock)*

# 3. Normalization of Relation Schemas

For the following section, all relations will be reduced to BCNF form. For this section, before I reduce, and normalize my relations, the following relations from section 2 above cannot be normalized any further due to them containing only two attributes, where the primary key can determine the other attribute.

Each relation that can be normalized will follow under the following three steps:

1. Check if R is in BCNF Using the simplified test (If *R* has not been decomposed yet)
   1. Find a non-trivial dependency *a 🡪 b* such that it causes a violation of BCNF
   2. Computer *a+*
   3. Verify that *a+* includes all attributes of *R*, proving it is a superkey of *R*.
2. Decompose *R* into BCNF form
3. *author(id, name)*

Functional dependencies:

*F = {*

*­id 🡪 name*

*}*

This relation is already normalized due to the following rule:

*For every set of attributes a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Because there is only two attributes, it cannot be normalized any further.

1. *genre(id, name, type)*

Functional dependencies:

*F = {*

*Id 🡪 name, type*

*genre\_name 🡪 id, type*

*}*

This relation is already normalized due to the following rule:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Because there are only two attributes, it cannot be normalized any further.

1. *publisher(id, name, phone, email, address\_id, transit\_number, account\_number, institution\_number)*

Functional dependencies:

F = {

*Id🡪 name, phone, email, address\_id, dd\_id*

*phone🡪 name, Id, email, address\_id, dd\_id*

*email 🡪 name, Id, phone, address\_id, dd\_id*

*address\_id 🡪 name, Id, phone, email, dd\_id*

*dd\_id 🡪 name, Id, phone, email, address\_id*

*name 🡪 id, phone, email, address\_id, dd\_id*

}

We know that by looking at the attributes, that each of them can be used to identify the other in this case (As there only exists a single phone, email, address and direct deposit bank information for a given publisher). Therefore, each functional dependency contains a superkey. Using the following rule, we can prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* includes all attributes of R. Therefore, we can say publisher is normalized.

1. *book(isbn, title, description, price, page\_count, published\_date, add\_date, rating, rating\_count, sale\_percent, author\_id, genre\_id, publisher\_id)*

Functional dependencies:

*F = {*

*isbn🡪 title, description, price, page\_count, published\_date, add\_date, rating, rating\_count, sale\_percent, author\_id, genre\_name, publisher\_id*

*}*

Note: The rating and ratings\_count. These two although one would assume are dependent upon each other, and therefore not a superkey, these are not functional dependencies of book, due to multiple books having the same count but different ratings and vise versa.

Another important note is author\_id and publisher\_id. Author\_id does not determine publisher\_id. This is made under the assumption that although rare, some authors switch publishers during their writing careers, so this functional dependency cannot exist.

We know that by looking at the attributes, that the only attribute that can determine another is isbn, which determines the rest of the attributes. Using the following rule, we can prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* (In this case just the isbn🡪) includes all attributes of R. Therefore, we can say book is normalized.

1. *orders(order\_number, username, order\_date, status\_id, card\_number, address\_id)*

Functional dependencies:

*F = {*

*Order\_number-> username, order\_date, status\_id, card\_number, address\_id*

*}*

The important realization about this is the card\_number, address\_id and username. These three attributes do not share a functional dependency (i.e. *card\_number 🡪 username, address\_id 🡪 username*). This is under the assumption that there could be the possibility of multiple orders sharing the same address but being ordered by different users (i.e. family members or roommates). Therefore, requiring username in this relation is important and is then as a functional dependency, but *card\_number 🡪 username, address\_id 🡪 username* are not however.

We know that by looking at the attributes, that the only attribute that can determine another is order\_number, which determines the rest of the attributes. Using the following rule, we can prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* (In this case just the order\_number🡪) includes all attributes of R. Therefore, we can say orders is normalized.

1. *status(status\_id, name, description)*

*F = {*

*Status\_id 🡪 name, description*

*Name🡪 status\_id, description*

*Description🡪name, status\_id*

*}*

This one is simple and similar to genre normalized above. We know that by looking at the attributes, that every attribute is defined as the *a* in *a🡪b*, and that every *b* contains the rest of the attributes (*R-a*), meaning each attribute determines every other attributes. We can use the following proof to check:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* includes all attributes of R. Therefore, we can say status is normalized.

1. *order\_book(order\_number, isbn, warehouse\_id, quantity)*

*F = {*

*Order\_number, isbn 🡪 warehouse\_id, quantity*

*}*

1. *cart(username, isbn, quantity)*

*F = {*

*username, isbn 🡪 quantity*

*}*

1. *client(username, email, first\_name, last\_name, password)*

*F = {*

*Username🡪 email, first\_name, last\_name, password*

*Email 🡪 username, first\_name, last\_name, password*

*}*

We know that by looking at the attributes, that there are only two functional dependencies which are username and email that determine everything else. Using the following rule, we can prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* includes all attributes of R. Therefore, we can say clients is normalized.

1. *admin(email, first\_name, last\_name, password)*

*F = {*

*Email 🡪 first\_name, last\_name, password*

*}*

We know that by looking at the attributes, that there are only two functional dependencies which is just email. Which determines everything else, making email a superkey. Using the following rule, we can prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* (In this case just the email🡪) includes all attributes of R. Therefore, we can say admin is normalized.

1. *sales\_report(report\_number, admin\_email, start\_date, end\_date, file)*

*F = {*

*Report\_number 🡪 admin\_email, start\_date, end\_date*

*}*

We know that by looking at the attributes, that there are only two functional dependencies which is just report\_number. Which determines everything else, making report\_number a superkey. Using the following rule, we can prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* (In this case just the report\_number 🡪) includes all attributes of R. Therefore, we can say sales\_report is normalized.

1. *report\_transaction(report\_number, transaction\_id)*

This table does not have any functional dependencies. This is due to how report\_transaction is set up. The report\_number and transaction\_id are both needed to get a unique row, meaning simply using report\_number or transaction\_id will not determine the other (As they are both many-many).

Because the relation only contains two attributes, we can use the following rule to prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

This rule holds, therefore it is normalized.

1. *transaction(transaction\_id, transaction\_name, amount, date)*

*F = {*

*Transaction\_id 🡪 transaction\_name, transaction\_type, amount, date*

*}*

We know that by looking at the attributes, that there is only one functional dependency which is just transaction\_id. Which determines everything else, making transaction\_id a superkey. Using the following rule, we can prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* (In this case just the transaction\_id 🡪) includes all attributes of R. Therefore, we can say transaction is normalized.

1. *warehouse(id, address\_id)*

*F = {*

*id 🡪 address\_id,  
address\_id 🡪 id*

*}*

This relation is similar to publisher from above in that all attributes can determine the author (As a warehouse can have at most one address, and that address can contain just the warehouse). We can also use the following rule to prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* includes all attributes of R. Therefore, we can say warehouse is normalized.

1. *warehouse\_books(warehouse\_id, isbn, stock)*

*F = {*

*Warehouse\_id, isbn 🡪 stock*

*}*

1. *address(id, country, state, city, code, street, apt\_number)*

*F = {*

*Id 🡪 state, city, code, street, apt\_number,*

*}*

This relation contains a single functional dependency, which is determined by ID. However, we can further add more dependencies:

We know that code🡪 city, state.

We know

1. *client\_address(username, address\_id)*

This relation has no functional dependencies. This is because a username cannot determine an address (As a user can have multiple addresses), and an address cannot determine user (As an address can have multiple users listed).

1. *payment\_info(card\_number, name, expiry\_date, security\_code)*

*F = {*

*Card\_number 🡪 name, expiry\_date, security\_code,*

*}*

1. *client\_billing(username, card\_number)*

This relation has no functional dependencies. This is because a username cannot determine a card number (As a user can have multiple cards), and a card cannot determine user (As a card can have multiple users listed).

1. *restock(restock\_number, isbn, warehouse\_id, quantity, restock\_date)*

*F = {*

*Restock\_number, isbn, warehouse\_id 🡪 quantity, restock\_date*

*}*

1. *request\_book(request\_number, username, request\_name, request\_isbn, date)*

*F = {*

*Request\_number 🡪 username, status, request\_name, request\_isbn, date*

*}*

We know that by looking at the attributes, that there is only one functional dependency which is just *Request\_number*. Which determines everything else, making *Request\_number* a superkey. Using the following rule, we can prove it is normalized:

*For every set of attributes, a ⊆ Ri , check that a+ (the attribute closure of a) either includes no attribute of Ri - a, or includes all attributes of R­i*

Therefore, each *a+* of every *a🡪b* (In this case just the *Request\_number* 🡪) includes all attributes of R. Therefore, we can say request\_book is normalized.

1. *admin\_decides(email, request\_number, decision, date)*

*F = {*

*Email, request\_number 🡪 decision, date*

*}*

1. *history(username, isbn, rank)*

*F = {*

*username, isbn 🡪 rank*

*}*

Mention discussion with email and username and how it is normalized for admin and client. Assumption that admin emails are constant and cannot be changed, but client ones can.

# 4. Database Schema Diagram

# 5. Implementation

# 6. Bonus Features

1. **Fuzzy search**
2. **Bestsellers**
3. **Recently viewed**
4. **Requesting books**
5. **GUI Webpage**
6. **Recently released**
7. **Download PDF sales reports**
8. **Transactions and viewing publisher information**

# 7. Github Repository

<https://github.com/SharjeelAliBCS/comp3005W20-project>

# 8. Availability