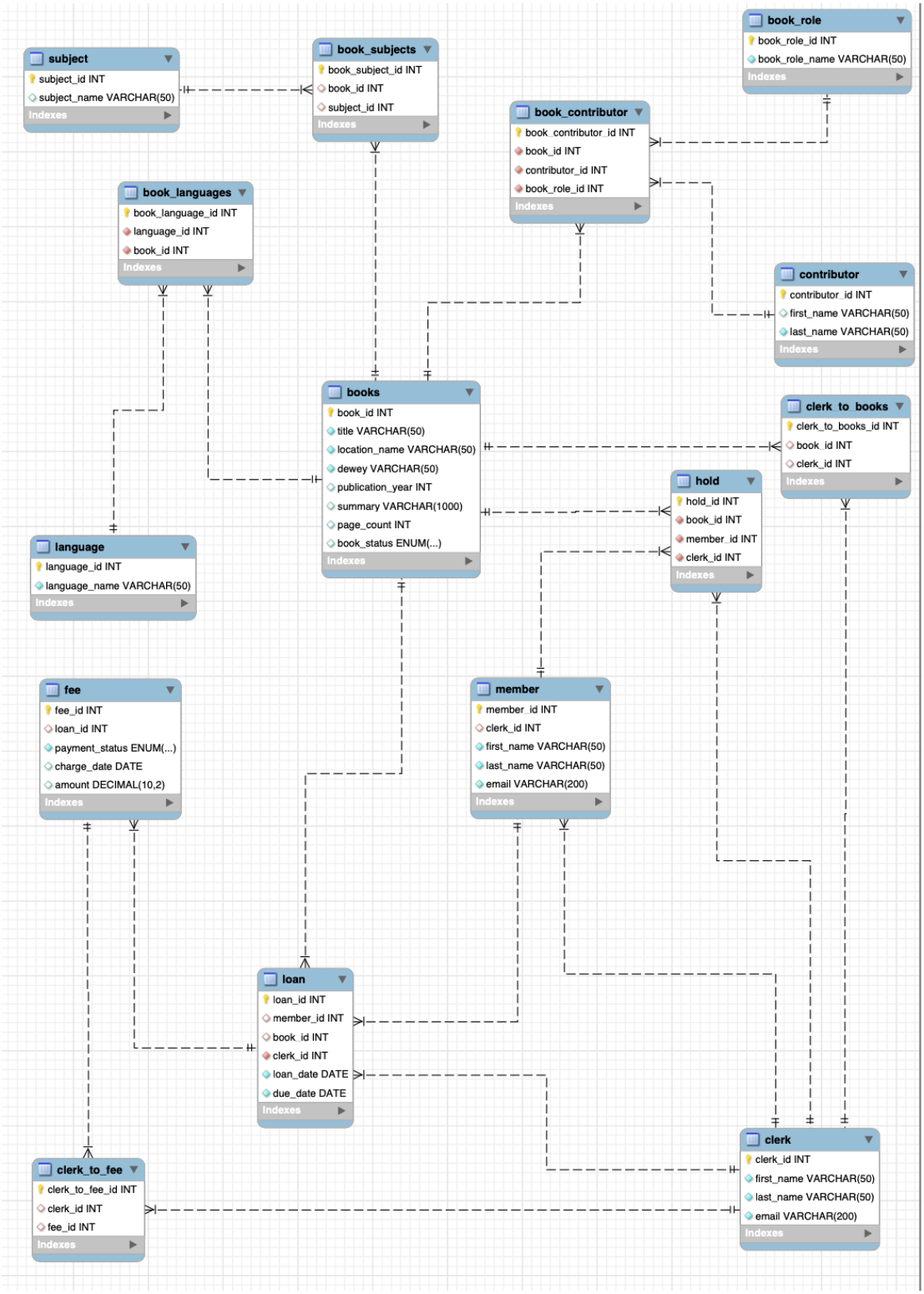
ERD


book_role
book_role_id INT
book_role_name VARCHAR(50)
Indexes
subject
subject_id INT
subject_name VARCHAR(50)
Indexes
book_subjects
book_subject_id INT
book_id INT
subject_id INT
Indexes
book_contributor
book_contributor_id INT
book_id INT
contributor_id INT
book_role_id INT
Indexes
book_languages
book_language_id INT
language_id INT
book_id INT
Indexes
contributor
contributor_id INT
first_name VARCHAR(50)
last_name VARCHAR(50)
Indexes
books
book_id INT
title VARCHAR(50)
location_name VARCHAR(50)
dewey VARCHAR(50)
publication_year INT
summary VARCHAR(1000)
page_count INT
book_status ENUM(...)
Indexes
clerk_to_books
clerk_to_books_id INT
book_id INT
clerk_id INT
Indexes
hold
hold_id INT
book_id INT
member_id INT
clerk_id INT
Indexes
language
language_id INT
language_name VARCHAR(50)
Indexes
fee
fee_id INT
loan_id INT
payment_status ENUM(...)
charge_date DATE
amount DECIMAL(10,2)
Indexes
member
member_id INT
clerk_id INT
first_name VARCHAR(50)
last_name VARCHAR(50)
email VARCHAR(200)
Indexes
loan
loan_id INT
member_id INT
book_id INT
clerk_id INT
loan_date DATE
due_date DATE
Indexes
clerk
clerk_id INT
first_name VARCHAR(50)
last_name VARCHAR(50)
email VARCHAR(200)
Indexes
clerk_to_fee
clerk_to_fee_id INT
clerk_id INT
fee_id INT
Indexes

QUERIES TO SUPPORT USER'S TASKS

#Deleting data to absolve a fee

```
DELETE FROM fee
WHERE fee_id = 1;
SELECT * FROM fee;
```

#Updating data to say that book 1 was damaged

```
UPDATE catalog
SET book_status = 'damaged'
WHERE book_id = 1;
SELECT * FROM catalog;
```

#Joining tables to see who damaged book 1

```
SELECT m.first_name, m.last_name
FROM loan l
JOIN `member` m ON l.member_id = m.member_id
JOIN catalog c ON l.book_id = c.book_id
WHERE c.book_id = 1 AND c.book_status = 'damaged';
```

#Creating view of the catalog for users to browse books by subject tags

```
CREATE VIEW subjects_view AS
SELECT
   s.subject_name AS subject,
   c.title AS book_title,
   c.location_name,
   c.dewey,
   c.book_status
FROM `subject` s
JOIN book_subject bs ON s.subject_id = bs.subject_id
JOIN catalog c ON bs.book_id = c.book_id
ORDER BY s.subject_name, c.title;

SELECT * FROM subjects_view;
```

#Creating another view to browse books by languages

```
CREATE VIEW languages_view AS
SELECT
   l.language_name AS language,
   c.title AS book_title,
   c.location_name,
```

```
    c.dewey,
    c.book_status
FROM `language` l
JOIN book_language bl ON l.language_id = bl.language_id
JOIN catalog c ON bl.book_id = c.book_id
ORDER BY l.language_name, c.title;

SELECT * FROM languages_view;

#Creating a view for members to see what fees are attributed to them

CREATE VIEW member_fees AS
SELECT
    m.member_id,
    CONCAT(m.first_name, ' ', m.last_name) AS member_name,
    l.loan_id,
    f.fee_id,
    f.payment_status,
    f.charge_date,
    f.amount
FROM
    member m
JOIN
    loan l ON m.member_id = l.member_id
JOIN
    fee f ON l.loan_id = f.loan_id;

SELECT *
FROM member_fees
WHERE member_id = 5;

#Transaction to pay off the fee for loan 1

START TRANSACTION;

UPDATE fee
SET payment_status = 'paid',
    charge_date = CURDATE()
WHERE fee_id = 2;

SELECT m.member_id, m.first_name, m.last_name, f.amount
FROM member m
JOIN loan l ON m.member_id = l.member_id
JOIN fee f ON l.loan_id = f.loan_id
WHERE f.fee_id = 2;
```

```
COMMIT;
SELECT * FROM loan;

#Trigger to update clerk_to_catalog whenever catalog is updated

SET @current_clerk_id = 1;
UPDATE catalog
SET book_status = 'on_loan'
WHERE book_id = 1;

DELIMITER $$
CREATE TRIGGER after_catalog_update
AFTER UPDATE ON catalog
FOR EACH ROW
BEGIN
   IF OLD.book_status != NEW.book_status THEN
      INSERT INTO clerk_to_catalog (clerk_id, book_id, update_date)
      VALUES (@current_clerk_id, NEW.book_id, CURDATE());
   END IF;
END;
$$
DELIMITER ;

#Inner query to see what languages a book is written in

SELECT language_name
FROM language
WHERE language_id IN (
   SELECT language_id
   FROM book_language
   WHERE book_id = 2
);

SELECT * FROM catalog;

#Inserting data for a member to loan out a book

INSERT INTO loan (member_id, book_id, clerk_id, loan_date, due_date)
VALUES (
   5,
   5,
   2,
   CURDATE(),
   DATE_ADD(CURDATE(), INTERVAL 30 DAY)
);
```

```
UPDATE catalog
SET book_status = 'on_loan'
WHERE book_id = 5;
SELECT * FROM catalog;

#Inserting data for a member to place a hold on a book

INSERT INTO hold (hold_id, book_id, member_id, clerk_id, hold_placed_date)
VALUES (
   6,
   4,
   5,
   3,
   CURDATE()
);

UPDATE catalog
SET book_status = 'on_hold'
WHERE book_id = 4;
SELECT * FROM catalog;

#Inserting data to register a new member

INSERT INTO member (member_id, clerk_id, first_name, last_name, email, date_registered)
VALUES (
   6,
   4,
   'Waylon',
   'Jennings',
   'wjennings@pratt.edu',
   CURDATE()
);
```

SUMMARY OF CHANGES

When redesigning my ERD, I added unique PKs to each associative entity so that the existing FKs can remain as is, such as language_name and book_role_name. I removed the catalog associate entity and renamed the book entity to catalog. My reasoning for this was so that members can interact with the catalog as a one-to-many relationship (one catalog for many users) and have their loans and holds take directly from the catalog similarly. In my actual dataset, multiple copies of unique titles have their own rows and would constitute unique book_ids. This approach also reduces redundancies. I kept an associative entity between clerk and catalog, clerk_to_catalog, so that any update to the catalog can be marked with a timestamp. I also added more columns to the catalog table that my dataset details: format and publisher. I removed the location entity since there are only two locations possible in the dataset, which can be listed explicitly in the catalog. I also removed the associative entity from clerk to member so that a member is always registered by one clerk directly. The relationship between member and fee was removed since a member may be charged between the two entities through a join with loan. Because the clerk_to_fee associative entity exists so that many clerks and deal with many fees, I added a clerk_to_fee_action column so that their varying actions can be specified.