A colorful text on a black background

Description automatically generatedA stack of books with red blue and white covers

Description automatically generated

Tamara Plante, Iana Setrakova, Médérick Blouin

Team 6

**Library Management System**

Table of Contents

[Scope 2](#_Toc165849909)

[Data Model 3](#_Toc165849910)

[Queries 4](#_Toc165849911)

[Catalogue Management 4](#_Toc165849912)

[Get the complete book catalogue data ***Iana*** 4](#_Toc165849913)

[Get the book formats with number of books available for each format ***Iana*** 5](#_Toc165849914)

[Get books by a category ***Iana*** 6](#_Toc165849915)

[Get the books of a series, in reading order ***Tamara*** 7](#_Toc165849916)

[Get available books by Id ***Tamara*** 8](#_Toc165849917)

[Get the formatted book location ***Tamara*** 9](#_Toc165849918)

[Get a book’s status ***Tamara*** 10](#_Toc165849919)

[Get books by an author’s name ***Médérick*** 11](#_Toc165849920)

[Membership Management 12](#_Toc165849921)

[Get the total number of new members, by year ***Médérick*** 12](#_Toc165849922)

[Get the number of students vs non-students ***Médérick, Tamara*** 12](#_Toc165849923)

[Update a member’s info ***Tamara, Iana*** 13](#_Toc165849924)

[Change a member’s address ***Iana*** 14](#_Toc165849925)

[Get the total newsletter subscribers ***Iana*** 14](#_Toc165849926)

[Checkout Management 15](#_Toc165849927)

[Checkout book ***Tamara*** 15](#_Toc165849928)

[Most borrowed books, for the last 6 months ***Iana*** 17](#_Toc165849929)

[Get a checkout return date ***Iana*** 18](#_Toc165849930)

[Late Fee Management 19](#_Toc165849931)

[Return book ***Tamara*** 19](#_Toc165849932)

[Get the total late fees in the current year ***Médérick*** 20](#_Toc165849933)

[Get the members with unpaid late fees ***Médérick*** 21](#_Toc165849934)

# Scope

The library system helps keep track of a large catalogue of books and supports managing the borrowing and returning them.

Key features include:

* **Catalogue Management**:   
  Keeping detailed information about each book, like the author, category, title, publisher, publication year, location and how many copies are available.
* **Membership Management**:   
  Storing members' personal details, such as names, birthdates, contact info (phone and email), addresses, and whether they're students or subscribe to newsletters.
* **Checkout Management**
  + Tracking when books are borrowed and returned, including due dates.
  + Members accounts have a limit of 10 physical books borrowed at a time and must be in good standing to borrow books.
  + eBook are automatically returned upon their checkout’s end date.
  + eBook checkout online is done using the “system” employee.
* **Late Fee Management:**Keeping track of late fees for overdue books, including how much is owed and when it needs to be paid.
* **Employee Database:**Managing information about library staff, their names, and roles. Additionally, a special 'system' row is used when managing checkout for viewing eBooks online.

# Data Model

A computer screen shot of a computer

Description automatically generated

# Queries

## Catalogue Management

### Get the complete book catalogue data ***Iana***

Retrieve all the books with authors, categories, and location details.

CREATE VIEW BookDisplayData AS

SELECT

b.id,

b.ISBN,

b.title AS 'Title',

GROUP\_CONCAT(DISTINCT(a.`name`) ORDER BY ba.author\_order ASC SEPARATOR ', ') as 'Author(s)',

GROUP\_CONCAT(DISTINCT(c.`name`) ORDER BY c.`name` ASC SEPARATOR ', ') as 'Category',

s.`name` AS 'Series',

b.`description` AS 'Description',

p.`name` AS 'Publisher''s name',

b.cover\_image AS 'Cover image',

fnBookFormat(b.id) AS 'Book Format',

b.pages AS 'Number of pages',

b.`language` AS 'Language',

b.`year` AS 'Year',

fn\_Location\_Format(b.location\_id) AS 'Location'

FROM book AS b

LEFT JOIN book\_author AS ba ON b.id = ba.book\_id

LEFT JOIN author AS a ON ba.author\_id = a.id

LEFT JOIN book\_category AS bc ON b.id = bc.book\_id

LEFT JOIN category AS c ON bc.category\_id = c.id

LEFT JOIN publisher AS p ON b.publisher\_id = p.id

LEFT JOIN book\_series AS bs ON b.id = bs.book\_id

LEFT JOIN series AS s ON bs.series\_id = s.id

GROUP BY b.id;

SELECT \* FROM BookDisplayData;



### Get the book formats with number of books available for each format ***Iana***

Get the book format total copies, ***book\_copies*** and ***eBook\_copies*** (format: “eBooks: x, Printed books: x”).

DELIMITER //

CREATE FUNCTION fnBookFormat(argBookID INT)

RETURNS VARCHAR(50) NOT DETERMINISTIC

BEGIN

DECLARE bookFormatString VARCHAR(50);

DECLARE ebook\_copies\_var INT;

DECLARE book\_copies\_var INT;

SELECT ebook\_copies, book\_copies

INTO ebook\_copies\_var, book\_copies\_var

FROM book WHERE id = argBookID;

IF ebook\_copies\_var > 0 AND book\_copies\_var > 0 THEN

SET bookFormatString = CONCAT('eBooks: ', ebook\_copies\_var, ', Printed books: ', book\_copies\_var);

ELSEIF ebook\_copies\_var > 0 THEN

SET bookFormatString = CONCAT('eBooks: ', ebook\_copies\_var);

ELSEIF book\_copies\_var > 0 THEN

SET bookFormatString = CONCAT('Printed books: ', book\_copies\_var);

ELSE

SET bookFormatString = 'Unknown format';

END IF;

RETURN bookFormatString;

END //

DELIMITER ;

SELECT fnBookFormat(20) AS 'Book Format';



### Get books by a category ***Iana***

Get books by a ***category name***.

SELECT

c.`name` as 'Category',

b.id,

b.ISBN,

b.title AS 'Title',

GROUP\_CONCAT(DISTINCT(a.`name`) ORDER BY ba.author\_order ASC SEPARATOR ', ') as 'Author(s)',

s.`name` AS 'Series',

fnBookFormat(b.id) AS 'Book Format'

FROM book AS b

JOIN book\_author AS ba ON b.id = ba.book\_id

JOIN author AS a ON ba.author\_id = a.id

JOIN book\_category AS bc ON b.id = bc.book\_id

JOIN category AS c ON bc.category\_id = c.id

JOIN book\_series AS bs ON b.id = bs.book\_id

JOIN series AS s ON bs.series\_id = s.id

WHERE c.name LIKE '%fiction%'

GROUP BY b.id;



### Get the books of a series, in reading order ***Tamara***

Retrieve the books of a specific ***series id***, making use of the ***book\_order***.

SELECT

id AS book\_id,

book\_order AS '#',

title,

`year`

FROM book

JOIN book\_series ON id = book\_id

WHERE series\_id = 6

ORDER BY 2;



### Get available books by Id ***Tamara***

Get the availability (book or eBook) for a specific ***book id***.

DELIMITER //

CREATE FUNCTION fn\_Book\_GetAvailable(arg\_book\_id INT, arg\_is\_book BOOLEAN)

RETURNS INT NOT DETERMINISTIC

BEGIN

DECLARE val\_total INT;

DECLARE val\_borrowed INT;

-- Total borrowed book

IF arg\_is\_book = TRUE THEN

SELECT COUNT(\*) INTO val\_borrowed

FROM checkout

WHERE

employee\_id <> 1

AND return\_date IS NULL

AND book\_id = arg\_book\_id;

SELECT book\_copies INTO val\_total FROM book

WHERE id = arg\_book\_id;

-- Total borrowed ebook

ELSE

SELECT COUNT(\*) INTO val\_borrowed

FROM checkout

WHERE

employee\_id = 1

AND return\_date IS NULL

AND book\_id = arg\_book\_id;

SELECT ebook\_copies INTO val\_total FROM book

WHERE id = arg\_book\_id;

END IF;

RETURN val\_total - val\_borrowed;

END //

DELIMITER ;

SELECT

title,

`year`,

fn\_Book\_GetAvailable(id, TRUE) AS book\_avail,

fn\_Book\_GetAvailable(id, FALSE) AS ebook\_avail

FROM book WHERE id = 29;



### Get the formatted book location ***Tamara***

Return the formatted book location by the ***location id*** (format: "Floor: -, Aisle: -, Shelf: -").

DELIMITER //

CREATE FUNCTION fn\_Location\_Format(arg\_location\_id INT)

RETURNS VARCHAR(50) NOT DETERMINISTIC

BEGIN

DECLARE val\_location VARCHAR(50);

SELECT CONCAT\_WS(', ',

CONCAT('Floor: ', COALESCE(floor, '-')),

CONCAT('Aisle: ', COALESCE(aisle, '-')),

CONCAT('Shelf: ', COALESCE(shelf, '-')))

INTO val\_location

FROM location

WHERE id = arg\_location\_id;

RETURN val\_location;

END //

DELIMITER ;

SELECT fn\_Location\_Format(1) AS location;



### Get a book’s status ***Tamara***

Return the book's location, quantity available and the earliest return date with the ***book id***.

DELIMITER //

CREATE PROCEDURE usp\_Book\_GetStatus(arg\_book\_id INT)

BEGIN

SELECT

fn\_Book\_GetAvailable(id, TRUE) AS available,

fn\_Location\_Format(location\_id) AS location,

(SELECT end\_date FROM checkout

WHERE

book\_id = book.id

AND employee\_id <> 1

AND return\_date IS NULL

ORDER BY end\_date

LIMIT 1) AS earliest\_return

FROM book

WHERE id = arg\_book\_id;

END //

DELIMITER ;

CALL usp\_Book\_GetStatus(20);



### Get books by an author’s name ***Médérick***

Get a list of books written by a particular author, using the ***author name***.

SELECT

author.`name`,

book.title,

book.`description`,

book.pages,

book.`language`,

book.id

FROM book

JOIN book\_author ON book.id = book\_author.book\_id

JOIN author ON book\_author.author\_id = author.id

WHERE author.`name` LIKE '%Tolkien%';



## Membership Management

### Get the total number of new members, by year ***Médérick***

Show how many new members joined this year.

SELECT

YEAR(registration\_date) AS years,

COUNT(\*) AS year\_count

FROM `member`

GROUP BY YEAR(registration\_date)

ORDER BY years ASC;



### Get the number of students vs non-students ***Médérick, Tamara***

Show the number of student and non-student members.

SELECT

IF(is\_student = TRUE, 'Student', 'Non-Student') AS type,

COUNT(\*) AS count

FROM `member`

GROUP BY is\_student;



### Update a member’s info ***Tamara, Iana***

Update the member's phone, email, student status and newsletter subscription, using the ***member id***.

DELIMITER //

CREATE PROCEDURE usp\_Member\_UpdateInfo(arg\_member\_id INT, arg\_phone VARCHAR(50), arg\_email VARCHAR(100), arg\_is\_newsletter BOOLEAN, arg\_is\_student BOOLEAN)

BEGIN

UPDATE `member`

SET

phone = COALESCE(arg\_phone, phone),

email = COALESCE(arg\_email, email),

is\_student = COALESCE(arg\_is\_student, is\_student),

is\_newsletter = COALESCE(arg\_is\_newsletter, is\_newsletter)

WHERE id = arg\_member\_id;

SELECT id, first\_name, last\_name, phone, email, is\_student, is\_newsletter

FROM `member` WHERE id = arg\_member\_id;

END //

DELIMITER ;

CALL usp\_Member\_UpdateInfo(10, NULL, 'juliagarcia@example.com', NULL, NULL);



### Change a member’s address ***Iana***

Update the member’s address, city, postal code, and province, using the ***member id***.

DELIMITER //

CREATE PROCEDURE usp\_Member\_ChangeAddress(arg\_member\_id INT, arg\_adress VARCHAR(150), arg\_city VARCHAR(50), arg\_postal\_code VARCHAR(20), arg\_province VARCHAR(50))

BEGIN

UPDATE `member`

SET

address = COALESCE(arg\_adress, address),

city = COALESCE(arg\_city, city),

postal\_code = COALESCE(arg\_postal\_code, postal\_code),

province = COALESCE(arg\_province, province)

WHERE id = arg\_member\_id;

SELECT id, first\_name, last\_name, address, city, postal\_code, province

FROM `member` WHERE id = arg\_member\_id;

END //

DELIMITER ;

CALL usp\_Member\_ChangeAddress(3, '114 Boul des Prairies', NULL, 'H7N 2T5', NULL);



### Get the total newsletter subscribers ***Iana***

Get the total number of people subscribed to newsletter.

SELECT COUNT(\*) AS 'Number of members'

FROM `member`

WHERE is\_newsletter = TRUE;



## Checkout Management

### Checkout book ***Tamara***

Checkout a physical book using the ***book id***, ***member id*** and ***employee id***. The member needs to have no unpaid late fees or reached the maximum of borrowed books. An additional validation is made to guarantee the member doesn’t borrow the same book twice, at the same time.

DELIMITER //

CREATE PROCEDURE usp\_Checkout\_AddBook(arg\_book\_id INT, arg\_member\_id INT, arg\_employee\_id INT)

BEGIN

DECLARE val\_book\_limit TINYINT DEFAULT 10;

DECLARE val\_book\_count TINYINT;

DECLARE val\_has\_book TINYINT;

DECLARE val\_fees DECIMAL(5, 2);

DECLARE val\_late\_books TINYINT;

DECLARE error\_msg VARCHAR(50);

-- Get the member's currently borrowed books (ignoring ebooks)

SELECT COUNT(\*), SUM(IF(book\_id = arg\_book\_id, 1, 0))

INTO val\_book\_count, val\_has\_book

FROM checkout

WHERE

member\_id = arg\_member\_id

AND employee\_id <> 1

AND return\_date IS NULL;

IF val\_has\_book > 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Checkout denied. Book already checked out by this member.';

ELSEIF val\_book\_count = val\_book\_limit THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Checkout denied. Book limit reached.';

END IF;

-- Get the member's unpaid late fees

SELECT SUM(amount\_due) INTO val\_fees

FROM checkout

JOIN late\_fees ON late\_fees\_id = late\_fees.id

WHERE

member\_id = arg\_member\_id

AND payment\_date IS NULL;

IF val\_fees IS NOT NULL THEN

SET error\_msg = CONCAT('Checkout denied. Late fees due: ', val\_fees, '$');

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = error\_msg;

END IF;

-- If there's any checkout that are late and not returned

SELECT SUM(IF(DATEDIFF(CURDATE(), end\_date) > 0, 1, 0))

INTO val\_late\_books

FROM checkout

WHERE

member\_id = arg\_member\_id

AND return\_date IS NULL;

IF val\_late\_books IS NOT NULL THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Checkout denied. Late book has not been returned.';

END IF;

-- Checkout

INSERT INTO checkout (book\_id, member\_id, start\_date, end\_date, employee\_id)

VALUES (arg\_book\_id, arg\_member\_id, CURDATE(), DATE\_ADD(CURDATE(), INTERVAL 14 DAY), arg\_employee\_id);

SELECT \* FROM checkout WHERE id = LAST\_INSERT\_ID();  
END //

DELIMITER ;

CALL usp\_Checkout\_AddBook(8, 2, 6);



### Most borrowed books, for the last 6 months ***Iana***

Get the most borrowed books for the last 6 months.

SELECT

b.id,

b.title AS 'Title',

GROUP\_CONCAT(DISTINCT(a.`name`) ORDER BY ba.author\_order ASC SEPARATOR ', ') as 'Author(s)',

COUNT(\*) AS 'No of times borrowed'

FROM checkout AS ch

JOIN book AS b ON ch.book\_id=b.id

JOIN book\_author AS ba ON b.id = ba.book\_id

JOIN author AS a ON ba.author\_id = a.id

WHERE ch.start\_date >= DATE\_SUB(NOW(), INTERVAL 6 MONTH)

GROUP BY b.id;



### Get a checkout return date ***Iana***

Get the expected end date/return date for a specific ***book id*** borrowed by a ***member id***.

SELECT

id AS checkout\_id,

book\_id,

member\_id,

start\_date,

end\_date,

CASE

WHEN return\_date IS NULL AND end\_date IS NOT NULL THEN end\_date

ELSE DATE\_ADD(start\_date, INTERVAL 14 DAY)

END AS expected\_return\_date

FROM checkout

WHERE

book\_id = 20

AND member\_id = 2;



## Late Fee Management

### Return book ***Tamara***

Return a physical book using the ***book\_id*** and ***member\_id*** and generate late fees (0.50$/late day) if there are any.

DELIMITER //

CREATE PROCEDURE usp\_Checkout\_ReturnBook(arg\_book\_id INT, arg\_member\_id INT)

BEGIN

DECLARE val\_today DATE DEFAULT CURDATE();

DECLARE val\_late\_days INT DEFAULT 0;

DECLARE val\_fees DECIMAL(5, 2) DEFAULT 0;

DECLARE val\_amount\_per DECIMAL(5, 2) DEFAULT 0.50;

DECLARE val\_id INT;

DECLARE val\_end\_date DATE;

DECLARE insert\_lfi INT;

-- Get checkout infos

SELECT id, end\_date

INTO val\_id, val\_end\_date

FROM checkout

WHERE

member\_id = arg\_member\_id

AND book\_id = arg\_book\_id

AND return\_date IS NULL;

-- Check if the return was late

SET val\_late\_days = DATEDIFF(val\_today, val\_end\_date);

IF val\_late\_days > 0 THEN

SET val\_fees = val\_amount\_per \* val\_late\_days;

INSERT INTO late\_fees (amount\_due) VALUES (val\_fees);

SET insert\_lfi = LAST\_INSERT\_ID();

END IF;

-- Update the checkout entry with return date and late\_fees if any

UPDATE checkout

SET return\_date = val\_today, late\_fees\_id = insert\_lfi

WHERE id = val\_id;

SELECT

member\_id,

book\_id,

late\_fees\_id,

val\_late\_days AS days\_late,

val\_fees AS amount\_due

FROM checkout

WHERE id = val\_id;

END //

DELIMITER ;

CALL usp\_Checkout\_ReturnBook(29, 1);



### Get the total late fees in the current year ***Médérick***

Show the total of fees charged for the current year, regardless of the payment status.

SELECT SUM(late\_fees.amount\_due) AS yearly\_fee\_total

FROM late\_fees

JOIN checkout ON late\_fees.id = checkout.late\_fees\_id

WHERE YEAR(checkout.end\_date) = YEAR(NOW());



### Get the members with unpaid late fees ***Médérick***

Show a list of members and their unpaid late fees.

SELECT

`member`.id,

`member`.first\_name,

`member`.last\_name,

late\_fees.amount\_due

FROM late\_fees

JOIN checkout ON late\_fees.id = checkout.late\_fees\_id

JOIN `member` ON checkout.member\_id = `member`.id

WHERE late\_fees.payment\_date IS NULL;

