Design and Implementation of a Library DBMS in Oracle 11g

Emma Gogic, Ethan Chiu and Hasan Khambaty

Table of Contents

Project Description

The purpose of this project was to design and implement a simple DBMS application, using the knowledge we acquired in the CPS 510 (Database Systems) course. The software we used for this project was Oracle 11g, which we used to execute our queries and display the results. Our topic of choice was a library database system. This report outlines the key features of our DBMS, including the tables/relations, ER diagram, SQL statemtents, functional dependencies, decomposition and normalization of the tables, UI implementation and relational algebra notation.

Initial Tables/Relations

Book:

book_id	title	category_id
######	example_title	#####
#####		

Author:

author_id	name
#####	example_name

book_author:

book_id	author_id
######	#######

Category:

category_id	name
#####	Comedy
######	Drama

Book_copy:

copy_id	year_published	publisher_id	book_id
#####	YYYY/MM/DD	#####	#####

Publisher:

publisher_id	name
########	Example_Name

Checkout:

checkout_id	start_time	end_time	copy_id	patron_id	is_returned
########	YYYY/MM/DD	YYYY/MM/DD	########	########	**Y**
########	YYYY/MM/D	YYYY/MM/D	########	########	**N**

	D	D			

Hold:

hold_id	**start_time**	**end_time**	**patron_id**	**copy_id**
########	YYYY/MM/DD	YYYY/MM/DD	########	########

Waitlist:

book_id	**patron_id**
########	########

Patron:

patron_id	**patron_name**	**patron_surname**	**patron_email**
########	Jhon	Doe	example@domain.ca
########	Jane	Doe	example@domain.ca

Notification:

notification_id	**time_sent**	**notification_type**	**patron_id**
########	YYYY/MM/DD	availability_notfication	########
########	YYYY/MM/DD	end_date_notification	########

Table Descriptions (what it does & what it contains)

Book

This table contains all the books in the database. Each book has a:

- unique identifier (**book_id**)
- title (**title**),
- genre, which is identified by its unique identifier (**category_id**).

Some examples of informal queries that can be applied to a library database using the BOOK table are:

- to find a specific book by title
- to list all the books in a particular genre

Some examples of update operations that can be performed using this table are:

- To add a new book to the library
- Remove a book from the library

Furthermore, each BOOK record is related to the records for:

- CATEGORY
- BOOK_AUTHOR
- WAITLIST
- BOOK_COPY.

Next, one of the constraints for this table is:

- Each **book_id** should be unique for each BOOK record (key constraint).

Finally, users of this database that could use the BOOK records include:

- library patrons
- librarians

Applications for a library patron include:

- Searching for books by id, title or genre

Applications for a librarian include:

- Managing the book inventory (adding/removing books).

Author

This table contains the authors in the database. Each of the authors has their unique identifier (**author_id**) and the name of the author (**name**). An example of informal queries that can be applied to a library database using the AUTHOR table is to find a book by the specific author by the author's name. You can update the books written by this author by adding the book under the author's unique identifier. A constraint for the table is that each **author_id** has to be unique for

each AUTHOR record (key constraint). The users of this database that can use the Author record include library patrons and librarians.

Book_Author

This table contains the books written by the author in the database. Each of the authors has a unique identifier (**author_id**) and each of the books has its unique identifier (**book_id**). An example of informal queries that can be applied to a library database using the Book_AUTHOR table is: To find a book by the specific author by the author's id or book id. A constraint for the table is that each **author_id** has to be unique for each author and each book_id has to be unique for each book. The users of this database that can use the Book_author record include library patrons and librarians.

Category

This table contains the different categories in the database. Each category has a:

- unique category identifier (**category_id**)
- the name of a book (**name**).

An example of informal queries that can be applied to the library database using the category table are:

- To find all books of a certain category
- To find what category a book belongs to

Some examples of update operations that can be performed using this table are:

- Changing the category a specific book is in

The constraints for this table are:

- Each book must have at least one category

Users of this database that could use the category records include:

- library patrons
- librarians

Applications for a library patron include:

- Searching for books by category or finding category by name

Applications for a librarian include:

- Managing the categories of each book.

Book_copy

This table contains the different book copies in the database. Each Book _copy has a:

- Unique identifier for each copy of the book (**copy_id**)
- The year it was published (**year_published**)
- The publisher (**publisher_id**)

- The id of the book (**book_id**)

An example of informal queries that can be applied to the library database using the Book _copy table are:

- Find a specific version of a book (ie book from a different publisher or publishing year)

Some examples of update operations that can be performed using this table are:

- Adding a new copy to the database
- Modifying entries in the table

the constraints for this table are:

- The copy_id must be unique to its book id
- No null fields

users of this database that could use the Book_copy records include:

- library patrons
- librarians

Applications for a library patron include:

- Searching for a specific copy of a book

Applications for a librarian include:

- Adding new entries to the database
- Modifying publisher ID or year published

Publisher

This table contains all the book publishers in the library database. Each publisher has a unique identifier (**publisher_id**) and the name of the publisher (**name**). An example of informal queries that can be applied to a library database using the Publisher table is: finding a publisher of a book using the name of the publisher or the publishers id. The constraints of the table is that the **publisher_id** has to be unique of each publisher. The users of this database that can use the Publisher record include library patrons and librarians.

Checkout

This table contains the book checkouts in the database. Each checkout has a unique identifier (**checkout_id**) and has (**start_time)** and (**end_time**) of each book being checked out. It also has the copy id of the book being checked out (**copy_id**) the id of the patron (**patron_id**) and a boolean condition seeing if the book has been returned (**is_returned**). Some examples of informal queries that can be applied to a library database using the Checkout table are :

- To check when the book was given to the patron
- To check if the book was returned
- If the book was returned when was it returned

Some examples of update operations that can be performed on this table are:

- Update the time the book was given to the patron
- Update the time the book was returned
- Update if the copy is in stock

Some constraints in this table are:

- The checkout id has to be unique
- The patron id has to be unique
- The book_copy id has to be unique

The users of this database that can use the Checkout record include librarians.

Hold

This table contains the different holds in the database. Each hold has a:

- Unique identifier for the hold (**hold_id**)
- The internal id of the patron (**patron_id**)
- The start date of the hold (**start_time**)
- The end date of the hold (**end_time**)
- The specific copy that is being borrowed (**copy_id**)

An example of informal queries that can be applied using the hold table are:

- Viewing which books and copies are currently put on hold

Some examples of update operations that can be performed using this table are:

- Adding a book to patrons' hold list
- Changing the end date of the hold

the constraints for this table are:

- The same copy cannot be put on hold at the same time
- The end time must not surpass the current date
- The patron ID must be valid in the system
- No fields can be null

users of this database that could use the hold records include:

- Library patrons
- Librarians

Applications for a library patron include:

- Requesting a book to be put on hold
- Remove hold request

Applications for a librarian include:

- Extend the end date of the hold
- Removing hold from patron account

Waitlist

This table contains a list of all the books on the waiting list and the patrons who are waiting for those books. Each waitlist entity has the following:

- **book_id** - unique identifier for each book on the waiting list
- **patron_id** - unique identifier for the patron waiting for a specific book

Some examples of informal queries that can be applied to a library database using the WAITLIST table are:

- Check if a patron is waiting for a specific book
- List all the books a specific patron is waiting for
- Count how many patrons are waiting for a specific book

Some examples of update operations that can be performed on this table are:

- Add a patron to the waitlist
- Remove a patron from the waitlist
- Clear a specific

Each WAITLIST record is directly related to the records for:

- BOOK
- PATRON

Next, some constraints for this table are:

- Each value of **book_id** and **patron_id** must also exist in some BOOK and PATRON records respectively (referential integrity constraint)

Finally, users of this database that could use the WAITLIST records include:

- Library patrons
- Librarians/library staff
- Library system administrators
- IT staff

Applications for a library patron include:

- Seeing which books they are currently waiting for
- Seeing how many other patrons are also waiting for the same book

Applications for a librarian include:

- Removing patrons from the waitlist once they have checked out the book
- Managing requests for books and updating the waitlist

Applications for library system administrators include:

- Maintenance of the waitlist table
- Manage rules regarding the waitlist (e.g. how long patrons can stay on the waitlist)

Applications for IT support staff include:

- Improve the functionality of the waitlist system (e.g. more efficient handling of the data)

Patron

This table contains the different patrons in the database. Each hold has a:

- The internal id of the patron(**patron_id**)
- The patron's first name (**patron_name**)
- The patron's surname (**patron_surname**)
- The email of the patron(**patron_email**)

An example of informal queries that can be applied using the hold table are:

- Finding patron information through their ID

Some examples of update operations that can be performed using this table are:

- Changing the first or last name or email address of the patron
- Adding a new patron to the database
- Removing patron from the database

the constraints for this table are:

- No two patrons should have the same patron_id
- No fields can be null

users of this database that could use the Patron records include:

- Library patron
- Librarian

Applications for a library patron include:

- Changing the first name
- Changing the surname
- Changing the email address

Applications for a librarian include:

- Adding a new patron to the database
- Removing a patron from the database

Notification

This table contains the necessary information for notifying patrons about the status of books in the library. Each notification has a:

- unique identifier (**notification_id**)
- a timestamp of when the notification was sent (**time_sent**),
- the type of notification (**notification_type**) which can be **availability_notification** (notifies the patron when a certain book on the waitlist has become available for checkout) and **end_date_notification** (notifies the patron that their book is due and must be returned).
- **patron_id,** which is the account id of the patron who is receiving the notification.

Some examples of informal queries that can be applied to a library database using the NOTIFICATION table are:

- To retrieve all the notifications sent to a specific patron
- To check when a specific notification was sent
- List all the notifications that were sent on a specific date
- List all the notifications that were sent to patrons for overdue books (i.e. list all the **end_date_notifications**)

Some examples of update operations that can be performed on this table are:

- Add a new notification for a patron
- Delete a notification when a patron has returned the book
- Update the time that a notification was sent (to resend a notification or correct an error)

Each NOTIFICATION record is directly related to the records for:

- PATRON

Next, some constraints for this table are:

- each **notification_id** should be unique for each NOTIFICATION record (key constraint)
- A value of **patron_id** in a NOTIFICATION record must also exist in some PATRON record (referential integrity constraint)
- The value of **notification_type** must be one of the values in the set {availability_notification, end_date_notification} (domain constraint)

Finally, users of this database that could use the NOTIFICATION records include:

- Library patrons
- Librarians/library staff
- Library system administrators
- IT support staff

Applications for a library patron include:

- Receiving notifications about book availability and due dates

Applications for a librarian include:

- Sending notifications to patrons

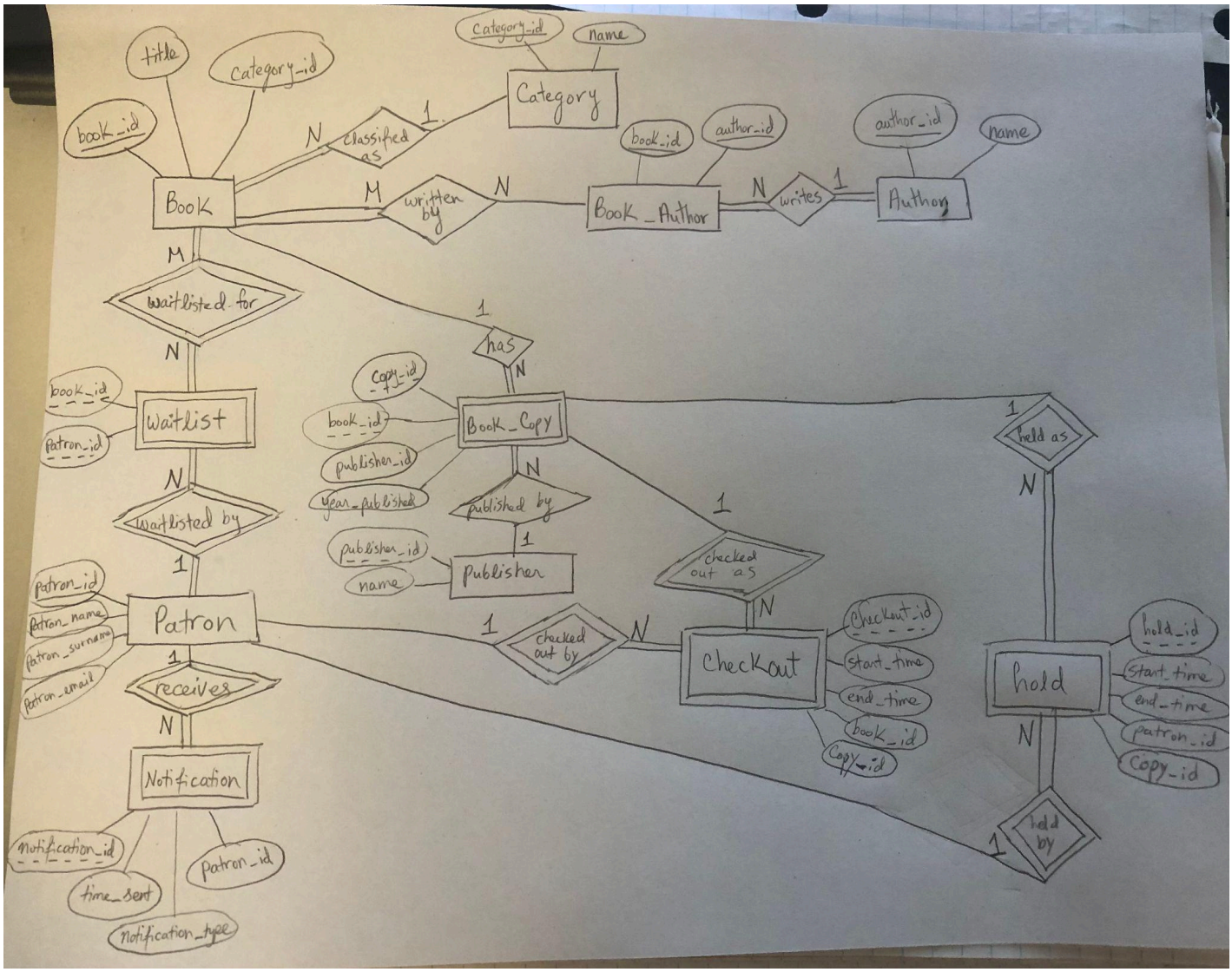
Applications for library system administrators include:

- Maintenance of the library notification system (managing when and how notifications are sent)

Applications for IT support staff include:

- Ensure the proper functioning of the notification system

ER Diagram


Category_id
Name
Category
title
Category_id
book_id
N
Classified as
1
Book
M
written by
N
Book_Author
book_id
author_id
N
writes
1
Author
author_id
name
M
Waitlisted for
1
has
N
N
book_id
patron_id
Waitlist
N
Waitlisted by
1
Copy_id
book_id
publisher_id
year_published
Book_Copy
N
Published by
1
Publisher
publisher_id
name
1
Held as
N
1
Checked out as
N
Checkout_id
start_time
end_time
book_id
Copy_id
Checkout
1
Checked out by
N
Patron_id
Patron_name
Patron_surname
Patron_email
Patron
1
receives
N
Notification
Notification_id
time_sent
Notification_type
Patron_id
hold_id
start_time
end_time
patron_id
copy_id
hold
N
1
Held by

SQL Source Code (creating and populating tables)

```
--CREATE TABLES WITH NO DEPENDENCIES-----------------------------------------
CREATE TABLE Author (
  author_id NUMBER PRIMARY KEY ,
  author_name VARCHAR2(255) NOT NULL
);

CREATE TABLE Category (
  category_id NUMBER PRIMARY KEY ,
  category_name VARCHAR2(255) NOT NULL
);

CREATE TABLE Publisher (
  publisher_id NUMBER PRIMARY KEY ,
  publisher_name VARCHAR2(255) NOT NULL
);

CREATE TABLE Patron (
  patron_id NUMBER PRIMARY KEY ,
  patron_name VARCHAR2(255) NOT NULL ,
  patron_surname VARCHAR2(255) NOT NULL ,
  patron_email VARCHAR2(255) UNIQUE
);
--CREATE TABLES WITH LOWER DEPENDANCIES---------------------------------------
CREATE TABLE Book (
  book_id NUMBER PRIMARY KEY ,
  title VARCHAR2(255) NOT NULL ,
  category_id NUMBER NOT NULL ,
  FOREIGN KEY (category_id) REFERENCES Category(category_id)
);

CREATE TABLE Book_author (
  book_id NUMBER NOT NULL,
  author_id NUMBER NOT NULL ,
  PRIMARY KEY(book_id , author_id) ,
  FOREIGN KEY (book_id) REFERENCES book(book_id),
  FOREIGN KEY (author_id) REFERENCES author(author_id)
);
--CREATE TABLES WITH HIGHER DEPENDANCIES---------------------------------------
```

```
CREATE TABLE Book_Copy (
  copy_id NUMBER PRIMARY KEY ,
  year_published NUMBER(4,0) NOT NULL,
  publisher_id NUMBER NOT NULL ,
  book_id NUMBER NOT NULL ,
  FOREIGN KEY (book_id) REFERENCES book(book_id),
  FOREIGN KEY (publisher_id) REFERENCES publisher(publisher_id)
);

CREATE TABLE Checkout (
  checkout_id NUMBER PRIMARY KEY ,
  start_time DATE DEFAULT SYSDATE ,
  end_time DATE NOT NULL ,
  book_id NUMBER NOT NULL,
  copy_id NUMBER NOT NULL ,
  patron_id NUMBER NOT NULL ,
  is_returned CHAR(1) CHECK (is_returned IN ('Y', 'N')) ,
  FOREIGN KEY (book_id) REFERENCES book(book_id),
  FOREIGN KEY (copy_id) REFERENCES book_copy(copy_id),
  FOREIGN KEY (patron_id) REFERENCES patron(patron_id)
);

CREATE TABLE Hold (
  hold_id NUMBER PRIMARY KEY ,
  start_time DATE DEFAULT SYSDATE ,
  end_time DATE NOT NULL ,
  patron_id NUMBER NOT NULL ,
  copy_id NUMBER NOT NULL ,
  FOREIGN KEY (copy_id) REFERENCES book_copy(copy_id),
  FOREIGN KEY (patron_id) REFERENCES patron(patron_id)
);

CREATE TABLE Waitlist (
  book_id NUMBER NOT NULL,
  patron_id NUMBER NOT NULL ,
  PRIMARY KEY (book_id, patron_id),
  FOREIGN KEY (book_id) REFERENCES book(book_id),
  FOREIGN KEY (patron_id) REFERENCES patron(patron_id)
);
```

```
CREATE TABLE Notification (
  notification_id NUMBER PRIMARY KEY ,
  time_sent DATE DEFAULT SYSDATE ,
  notification_type VARCHAR2(50) NOT NULL ,
  patron_id NUMBER NOT NULL ,
  FOREIGN KEY (patron_id) REFERENCES patron(patron_id)
);

--Copy output and run to drop all tables
--select 'drop table ', table_name, 'cascade constraints;' from user_tables;

--Populating tables----------------------------------------------------

INSERT INTO Author (author_id, author_name) VALUES (1, 'J.D. Salinger');
INSERT INTO Author (author_id, author_name) VALUES (2, 'Isaac Asimov');
INSERT INTO Author (author_id, author_name) VALUES (3, 'Agatha Christie');
INSERT INTO Author (author_id, author_name) VALUES (4, 'David McCullough');

INSERT INTO Category (category_id, category_name) VALUES (1, 'Fiction');
INSERT INTO category (category_id, category_name) VALUES (2, 'Science Fiction');
INSERT INTO category (category_id, category_name) VALUES (3, 'Mystery');
INSERT INTO category (category_id, category_name) VALUES (4, 'History');

INSERT INTO Publisher (publisher_id, publisher_name) VALUES (1, 'Little, Brown and
Company');
INSERT INTO publisher (publisher_id, publisher_name) VALUES (2, 'Random House');
INSERT INTO publisher (publisher_id, publisher_name) VALUES (3, 'Penguin Books');
SET DEFINE OFF;
-- allows for use of '&' character in the string, otherwise it is a substitution variable
INSERT INTO publisher (publisher_id, publisher_name) VALUES (4, 'Simon & Schuster');
SET DEFINE ON;

INSERT INTO Patron (patron_id, patron_name, patron_surname, patron_email) VALUES (1,
'John', 'Doe', 'john.doe@example.com');
INSERT INTO patron (patron_id, patron_name, patron_surname, patron_email) VALUES (2,
'Jane', 'Smith', 'janesmith@example.com');
INSERT INTO patron (patron_id, patron_name, patron_surname, patron_email) VALUES (3,
'Alice', 'Johnson', 'alicejohnson@example.com');

INSERT INTO Book (book_id, title, category_id) VALUES (1, 'The Catcher in the Rye', 1);
```

```
INSERT INTO book (book_id, title, category_id) VALUES (2, 'Foundation', 2);
INSERT INTO book (book_id, title, category_id) VALUES (3, 'Murder on the Orient Express',
3);
INSERT INTO book (book_id, title, category_id) VALUES (4, '1776', 4);

INSERT INTO Book_author (book_id, author_id) VALUES (1, 1);
INSERT INTO book_author (book_id, author_id) VALUES (2, 2); -- Foundation by Isaac
Asimov
INSERT INTO book_author (book_id, author_id) VALUES (3, 3); -- Murder on the Orient
Express by Agatha Christie
INSERT INTO book_author (book_id, author_id) VALUES (4, 4); -- 1776 by David
McCullough

INSERT INTO book_copy (copy_id, book_id, publisher_id, year_published) VALUES (1, 1, 1,
1951);
INSERT INTO book_copy (copy_id, book_id, publisher_id, year_published) VALUES (4, 1, 4,
2024);
INSERT INTO book_copy (copy_id, book_id, publisher_id, year_published) VALUES (2, 2, 2,
1934);
INSERT INTO book_copy (copy_id, book_id, publisher_id, year_published) VALUES (3, 3, 3,
2005);

INSERT INTO Checkout (checkout_id, book_id, copy_id, patron_id, start_time, end_time
,is_returned) VALUES (1, 1, 1, 1, SYSDATE, SYSDATE + 14 , 'Y'); -- John Doe checks out
"Foundation"
INSERT INTO Checkout (checkout_id, book_id, copy_id, patron_id, start_time, end_time ,
is_returned) VALUES (2, 2, 2, 2, SYSDATE, SYSDATE + 20 , 'N'); -- Jane Smith checks out
"Murder on the Orient Express"
INSERT INTO Checkout (checkout_id, book_id, copy_id, patron_id, start_time, end_time ,
is_returned) VALUES (3, 3, 3, 3, SYSDATE, SYSDATE + 60, 'Y'); -- Alice Johnson checks out
"1776"

INSERT INTO hold (hold_id, copy_id, patron_id, start_time, end_time) VALUES (1, 1, 2,
SYSDATE, SYSDATE + 7); -- Jane Smith holds a copy of "Foundation"
INSERT INTO hold (hold_id, copy_id, patron_id, start_time, end_time) VALUES (2, 2, 3,
SYSDATE, SYSDATE + 7); -- Alice Johnson holds a copy of "Murder on the Orient Express"
INSERT INTO hold (hold_id, copy_id, patron_id, start_time, end_time) VALUES (3, 3, 1,
SYSDATE, SYSDATE + 7); -- John Doe holds a copy of "1776"
```

```
INSERT INTO waitlist (book_id, patron_id) VALUES (1, 1); -- John Doe is waitlisted for "Foundation"
INSERT INTO waitlist (book_id, patron_id) VALUES (2, 2); -- Jane Smith is waitlisted for "Murder on the Orient Express"
INSERT INTO waitlist (book_id, patron_id) VALUES (3, 3); -- Alice Johnson is waitlisted for "1776"

INSERT INTO notification (notification_id, patron_id, time_sent, notification_type) VALUES (1, 1, SYSDATE, 'availability_notification');
INSERT INTO notification (notification_id, patron_id, time_sent, notification_type) VALUES (2, 2, SYSDATE, 'end_date_notification');
```

SQL source code (creating simple queries)

```
--CREATING QUIERIES----------------------------------------------------------------
--Author - count the number of Books written by a specific Author ('Agatha Christie')
SELECT Author_name AS "Author", COUNT(Book.book_id) AS "Number of Books"
FROM Author
JOIN Book_Author ON Author.author_id = Book_Author.author_id
JOIN Book ON Book_Author.book_id = Book.book_id
WHERE Author_name = 'Agatha Christie'
GROUP BY Author_name;

--Category - get categories ordered by name:
SELECT *
FROM Category
ORDER BY category_name;

--Publisher - get publisher details by publisher_id
SELECT *
FROM Publisher
WHERE publisher_id = 1;

--Patron - get patrons with a specific email:
SELECT *
FROM Patron
WHERE patron_email = 'example@example.com';

--Book - list all books in the Science Fiction category
SELECT Book.title AS "Book Title"
FROM Book
JOIN Category ON Book.category_id = Category.category_id
WHERE category_name = 'Science Fiction';

--Book_author - get books written by a specific author:
SELECT *
FROM Book_author
WHERE author_id = 3;

--Book_Copy - get all book copies published by a specific publisher:
SELECT *
FROM Book_Copy
WHERE publisher_id = 4;

--Checkout - list all patron id checkouts that are returned
SELECT patron_id
FROM CHECKOUT
WHERE is_returned = 'Y';
```

```
--Hold - get all holds that end before a certain date:
SELECT *
FROM Hold
WHERE end_time < TO_DATE('2024-12-31', 'YYYY-MM-DD');

--Waitlist - get all waitlisted patrons for a specific book:
SELECT *
FROM Waitlist
WHERE book_id = 2;

--list all notifications sent to a specific patron ('Jane Smith')
SELECT Notification.notification_type AS "Notification Type", Notification.time_sent AS
"Time Sent"
FROM Notification
JOIN Patron ON Notification.patron_id = Patron.patron_id
WHERE Patron.patron_name = 'Jane' AND Patron.patron_surname = 'Smith';
```

SQL source code (creating view tables)

```
--VIEW TABLES----------------------------------------
--CHECKEDOUT - Creates view table for checked out items that are not returned
CREATE VIEW Checkedout(checkout_id, book_id, copy_id, patron_id)
AS(SELECT checkout_id, book_id, copy_id, patron_id
FROM CHECKOUT
WHERE IS_RETURNED = 'N');


drop view List_Science_Fiction;
----List_Science_Fiction - List all titles where the books are science fiction
CREATE VIEW List_Science_Fiction(book_title)
AS(SELECT title
FROM BOOK
JOIN CATEGORY ON BOOK.category_id = CATEGORY.category_id
WHERE CATEGORY.category_name = 'science fiction');

--count the number of copies of a specific book
CREATE VIEW COUNT_COPIES(title, copy_id, book_id)
AS(SELECT title, BOOK_COPY.copy_id, BOOK_COPY.book_id
FROM BOOK
JOIN BOOK_COPY ON BOOK.book_id = BOOK_COPY.book_id
WHERE BOOK.title = 'The Catcher in the Rye');
SELECT COUNT(copy_id) AS Number_of_copies FROM COUNT_COPIES;
```

SQL source code (creating advanced queries)

```
--5 NEW QUERIES (using set-based keywords and aggregate functions)----------
--List all books that have at least 2 copies in the library
SELECT Book.title AS "Book Title"
FROM Book
JOIN Book_Copy ON Book.book_id = Book_Copy.book_id
GROUP BY Book.title
HAVING COUNT(Book_Copy.copy_id) >= 2;

--List all patrons who have checked out a book and received a notification for that book
SELECT DISTINCT p.patron_name || ' ' || p.patron_surname AS "Patron Name"
FROM Patron p
JOIN Checkout c ON p.patron_id = c.patron_id
WHERE EXISTS (
   SELECT *
   FROM Notification n
   WHERE n.patron_id = p.patron_id
   AND n.notification_type = 'end_date_notification'
   AND n.patron_id = c.patron_id
   );

--List all books that are in the "Science Fiction" or "History" category
SELECT b.title AS "Book Title"
FROM Book b
JOIN Category c ON b.category_id = c.category_id
WHERE c.category_name = 'Science Fiction'

UNION

SELECT b.title AS "Book Title"
FROM Book b
JOIN Category c ON b.category_id = c.category_id
WHERE c.category_name = 'History';

--List all books that are in the "Science Fiction" category and are currently checked out
SELECT book.title AS "Science Fiction Books"
FROM Book
JOIN Category ON book.category_id = category.category_id
where category.category_name = 'Science Fiction'

MINUS

SELECT book.title AS "Returned Science Fiction Books"
FROM Book
JOIN Category ON book.category_id = category.category_id
```

```
JOIN Book_Copy ON book.book_id = book_copy.book_id
JOIN Checkout ON book_copy.copy_id = checkout.copy_id
WHERE category.category_name = 'Science Fiction' AND checkout.is_returned = 'Y'

--calculate the average number of books checked out per patron
SELECT AVG(Checkout_Count) AS "Average Checkouts Per Patron"
FROM (
   SELECT patron.patron_id, COUNT(checkout.checkout_id) AS Checkout_Count
   FROM Patron
   LEFT JOIN Checkout ON patron.patron_id = checkout.patron_id  --Ensures that patrons with
no checkouts are included in the calculation (with a count of 0).
   GROUP BY patron.patron_id
  );
```

Functional Dependencies

1. Author :
author_id → author_name

2. Category
category_id → category_name

3. Publisher
publisher_id → publisher_name

4. Patron
patron_id → patron_name, patron_surname, patron_email
patron_email → patron_id

5. Book
book_id → title, category_id
category_id → category_name

6. Book_Author
{book_id, author_id} → (book_id, author_id)

7. Book_Copy
copy_id → year_published, publisher_id, book_id
book_id → category_id (inferred from Book table)

8. Checkout
checkout_id → start_time, end_time, book_id, copy_id, patron_id, is_returned
book_id → category_id
patron_id → patron_name, patron_surname

9. Hold
hold_id → start_time, end_time, patron_id, copy_id
patron_id → patron_name, patron_surname

10. Waitlist
{book_id, patron_id} → (book_id, patron_id)

11. Notification
notification_id → time_sent, notification_type, patron_id
patron_id → patron_name, patron_surname (from Patron table)

--TRANSITIVE & PARTIAL DEPENDANCIES----------

TRANSITIVE DEPENDENCY EXAMPLE:

in the Checkout table:
checkout_id → book_id and book_id → category_id imply:
checkout_id → category_id

COMPOUND PRIMARY KEY & PARTIAL DEPENDENCY EXAMPLE:
in the Book_Copy table:
{copy_id, book_id} → category_id
partial dependency:
{book_id} → category_id
category_id only depends on part of the compound primary key (book_id), rather than the whole primary key.

Table Decomposition to 2NF and 3NF

--TABLE DECOMPOSITION----------

DECOMPOSITION TO 2NF for Checkout table:
Checkout(checkout_id, start_time, end_time, book_id, copy_id, patron_id, is_returned)
Book_Category(book_id, category_id)

DECOMPOSITION TO 3NF for Checkout table:
Checkout(checkout_id, start_time, end_time, book_id, copy_id, patron_id, is_returned)
Book_Category(book_id, category_id)
Patron_Details(patron_id, patron_name, patron_surname)

DECOMPOSITION TO 3NF for Hold table:
Hold(hold_id, start_time, end_time, patron_id, copy_id)
Patron_Details(patron_id, patron_name, patron_surname)

Normalization/BCNF

---All tables converted to BCNF----------
--The tables already in BCNF: Book, Author, Category, Publisher, Book_Author, Waitlist, Notification

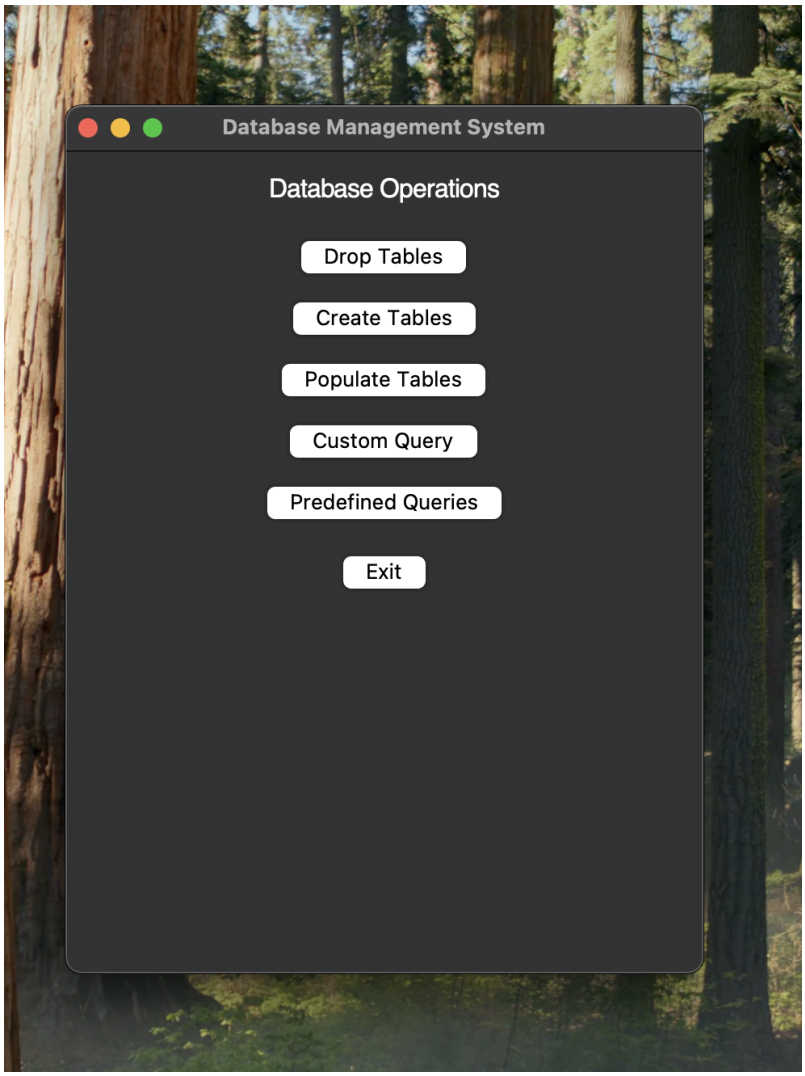
Patron(patron_id, patron_name, patron_surname)
Patron_Email(patron_email, patron_id)

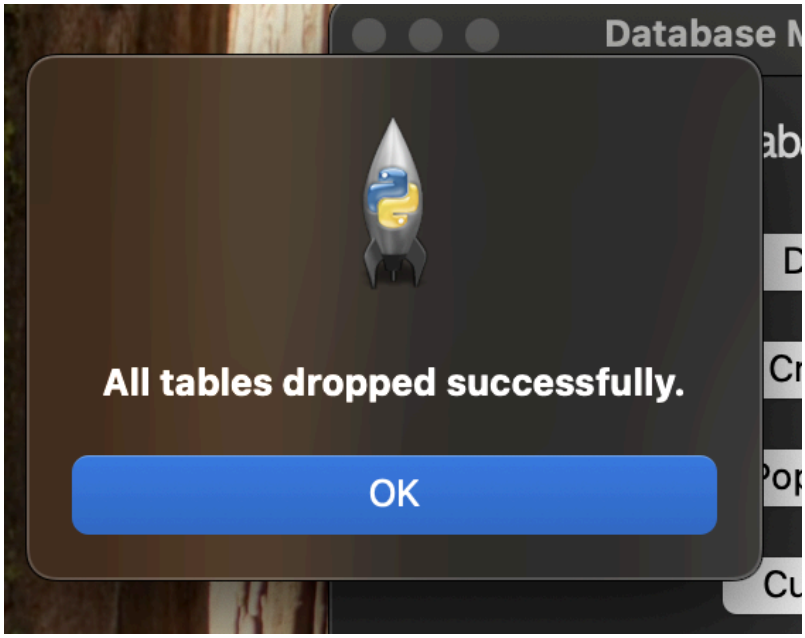
Book_Copy(copy_id, year_published, publisher_id, book_id)
Book_Category(book_id, category_id)

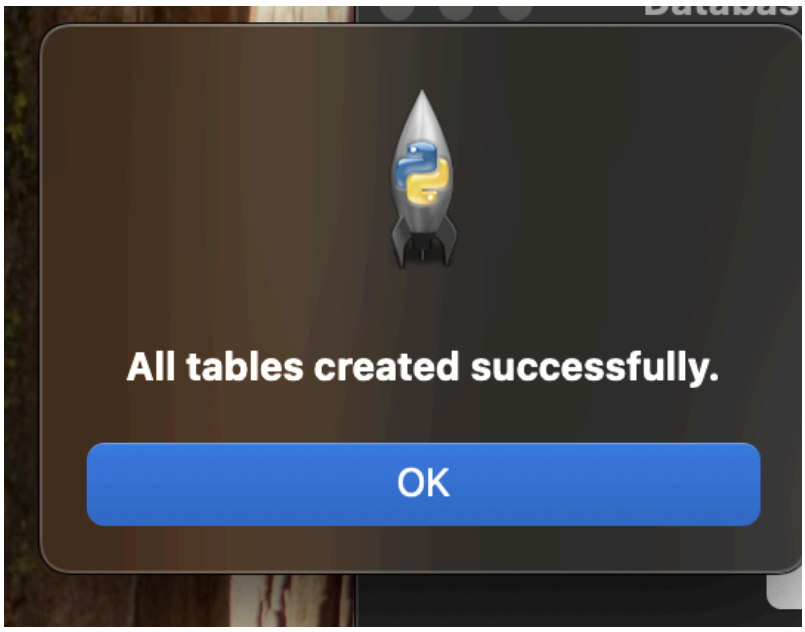
Checkout(checkout_id, start_time, end_time, book_id, copy_id, patron_id, is_returned)
Patron_Details(patron_id, patron_name, patron_surname)

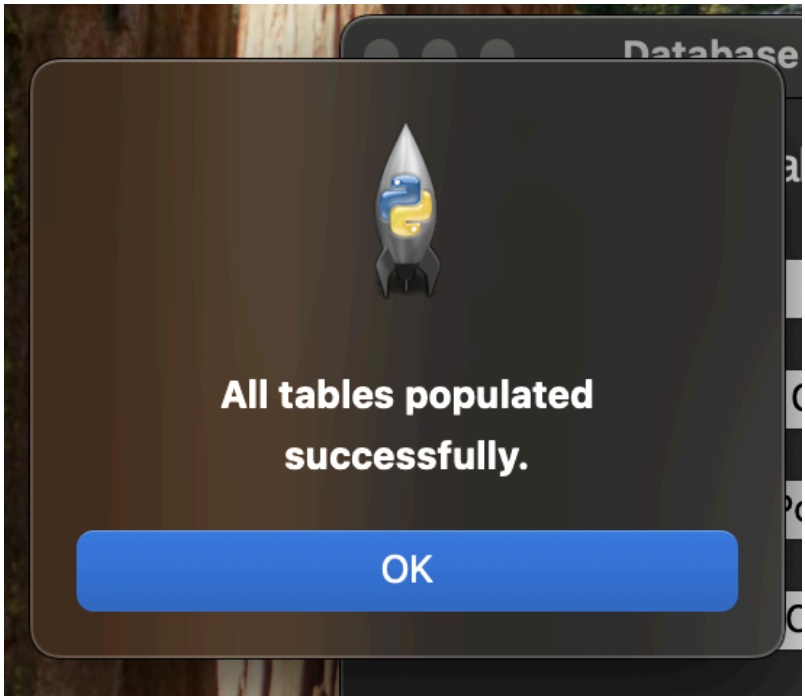
Hold(hold_id, start_time, end_time, patron_id, copy_id)
Patron_Details(patron_id, patron_name, patron_surname)

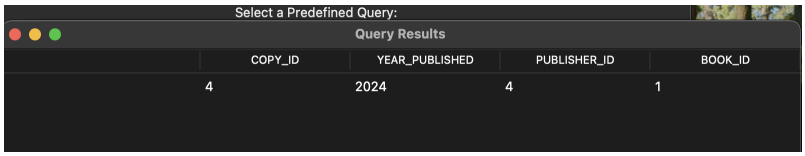
UI Implementation




Database M
All tables dropped successfully.
OK


All tables created successfully.
OK


All tables populated
successfully.
OK


Select a Predefined Query:
Query Results
COPY_ID
YEAR_PUBLISHED
PUBLISHER_ID
BOOK_ID
4
2024
4
1

Relational Algebra (RA) Notation

1. count the number of Books written by a specific Author ('Agatha Christie')

$$\text{Author\_name}\mathcal{F}\text{COUNT}_{\text{book\_id}}(\sigma_{\text{Author\_name}=\text{'AgathaChristie}}(\text{Author} \bowtie \text{Book\_Author} \bowtie \text{Book}))$$

2. get publisher details by publisher_id

$$\sigma_{\text{publisher\_id}=1}(\text{Publisher})$$

3. list all books in the Science Fiction category

$$\pi_{\text{title}}(\sigma_{\text{category\_name}=\text{'ScienceFiction'}}(\text{Book} \bowtie \text{Category}))$$

4. get all books written by a specific author (author_id = 3)

$$\sigma_{\text{author\_id}=3}(\text{Book\_Author})$$

5. list all patron id checkouts that are returned

$$\pi_{\text{patron\_id}}(\sigma_{\text{is\_returned}=\text{'Y'}}(\text{Checkout}))$$

6. get all holds that end before a certain date (‘2024-12-31’)

$$\sigma_{\text{end\_time}<\text{'2024-12-31'}}(\text{Hold})$$

7. list all notifications (by notification_type and time_sent) sent to a specific patron ('Jane Smith')

$$\pi_{\text{notification\_type,time\_sent}}(\sigma_{\text{patron\_name}=\text{'Jane'}} \cap \sigma_{\text{patron\_surname}=\text{'Smith'}}(\text{Notification} \bowtie \text{Patron}))$$

8. List all books that have at least 2 copies in the library

$$\text{title}\mathcal{F}\text{COUNT}_{(\text{copy\_id})\geq 2}(\text{Book} \bowtie \text{Book\_Copy})$$

9. List all patrons (name and surname) who have checked out a book **and** received a notification for that book

$$\pi_{\text{patron\_name,patron\_surname}}(\sigma_{\text{is\_returned}=\text{'Y'}} \cap \sigma_{\text{notification\_type}=\text{'end\_date\_notification'}}(\text{Patron} \bowtie \text{Checkout} \bowtie \text{Notification}))$$

10. List all books that are in the "Science Fiction" **or** "History" category

$$\pi_{\text{title}}(\sigma_{\text{category\_name}=\text{'ScienceFiction'}}(\text{Book} \bowtie \text{Category})) \cup \pi_{\text{title}}(\sigma_{\text{category\_name}=\text{'History'}}(\text{Book} \bowtie \text{Category}))$$

11. calculate the average number of books checked out per patron

$$F_{\text{AVG(COUNT(copy\_id))}}(\text{patron\_id} F_{\text{COUNT(checkout\_id)}}(\text{Patron} \bowtie \text{Checkout}))$$