# CMPT 354: Mini-project

Library Database

Group 44
Members:
Armin Hatami
Jesse Huang

Date: 2023-08-02

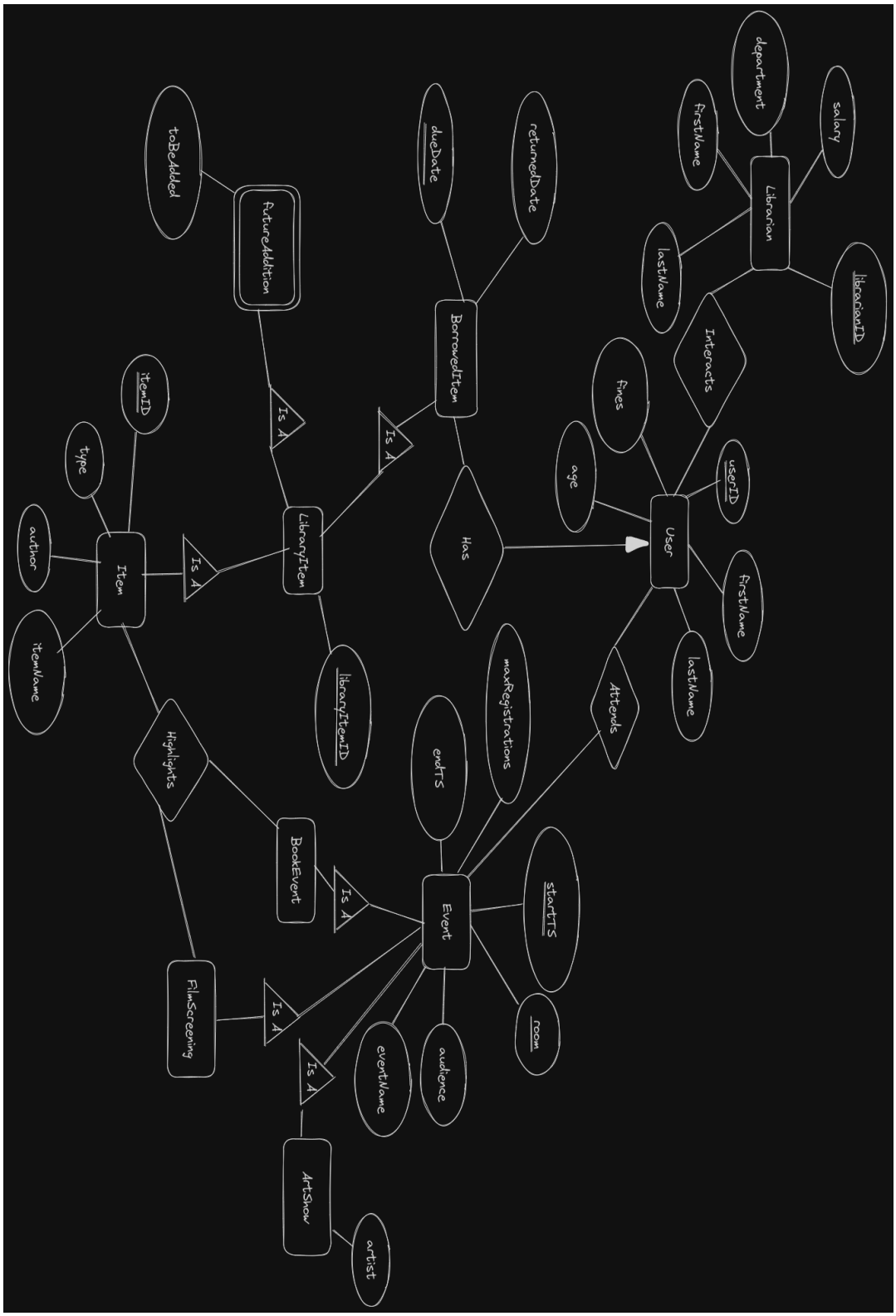
Github: /dreaminred/cmpt354-miniproject/tree/main

# 2 - Project Specifications

- An Item represents an entity a User can borrow
    - The library can have multiple copies of the same Item, however each copy has a unique itemID
- Authors first name + last name is represented by the Author attribute and assumes that each author has a unique name.
- Library has different types of Items such as print books, online books, magazines, scientific journals, CDs, records and films
- Library keeps track of all Items, including ones they want to add in a future date
- Library keeps track of Users ID and their name
- Library keeps track of the librarians that work there
- Users must be at least seven years old to register for library membership
- Users can borrow Items from the library and return them
- Users will be fined only once when they are not returned by due date
- Users with outstanding fees will not be able to borrow Items
- Users have a maximum of two weeks to return items without getting fined
- Users can borrow as many LibraryItems as they wish
- If a User has multiple overdue items, the User will only get fined for one.
- The maximum fine a User can accrue is $3.00
- Library events are recommended for specific audiences
- Library events can highlight an Item (movie, book, etc.)
- Library can host events in library rooms that are free to any user, such as:
    - Book events: such as book clubs, book signings, etc. which highlights an Item of type Book
    - Film Screenings: highlights an Item of type Film
    - Art Shows: list of artists that will have their work shown
- A librarian might register to be a user as well.
- A librarian can work in the following departments
    - ( 'admin', 'catalog', 'circulation', 'infotech', 'volunteer')

# 3 - E/R Diagrams

Click HERE for larger image



# 4 - FDs, Schema and BCNF

Database is in BCNF because all relations are in BCNF

- User(<u>userID</u>, firstName, lastName, age, fines)
    - FDs: UserID -> FirstName, LastName, Age, Fines
    - BCNF because for X → Y in FDs, X is a key
    - Constraints
        - age, firstName, lastName NOT NULL
        - fines >= 0
        - age>= 7
    - Types
        - userID, age INTEGER
        - firstName, lastName VARCHAR(30)
        - fines NUMERIC(5,2) DEFAULT = 0
- Librarian(<u>librarianID</u>, firstName, lastName, salary, department, userID$^{FK\text{-}User}$)
    - FDs: librarianID-> firstName, lastName, salary, department, userID
    - BCNF because for X → Y in FDs, X is a key
    - Constraints
        - firstName, lastName, department NOT NULL
        - salary >= 0
        - department IS NOT ‘volunteer’ OR salary=0
        - department IN ( 'admin', 'catalog', 'circulation', 'infotech', 'volunteer')
        - userID UNIQUE
    - Types
        - librarianID INTEGER
        - firstName, lastName VARCHAR(30)
        - department VARCHAR(4)
        - salary NUMERIC(6,2) DEFAULT 0
        - userID DEFAULT NULL
- BorrowedItem(<u>userID</u>$^{FK\text{-}User}$, <u>libraryItemID</u>$^{FK\text{-}LibraryItem}$, <u>dueDate</u>, returnedDate)
    - FDs: userID, libraryItemID, dueDate-> returnedDate
    - BCNF because for X → Y in FDs, X is a key
    - Constraints
        - userID, libraryItemID, dueDate NOT NULL
    - Types
        - returnedDate DATETIME
        - dueDate DATETIME DEFAULT (datetime('now', '+14 day'))
- LibraryItem(<u>libraryItemID</u>, itemID$^{FK\text{-}Item}$, toBeAdded)
    - FDs: libraryItemID-> itemID, toBeAdded
    - BCNF because for X → Y in FDs, X is a key
    - Constraints
        - itemID NOT NULL

    - Types
        - libraryItemID INTEGER
        - itemID INTEGER
        - toBeAdded BOOLEAN
- Item(<u>itemID</u>, author, itemName, type)
    - FDs: itemID -> author, itemName, type
    - BCNF because for X → Y in FDs, X is a key
    - Constraints
        - author, itemName, type NOT NULL
        - type IN ('movie', 'book', 'song', 'paper')
    - Types
        - itemID INTEGER
        - Author, ItemName, Type VARCHAR(30)
- Event(<u>startTS</u>, <u>room</u>, eventName, audience, maxRegistrations, endTS, artist, itemID$^{FK\text{-}Item}$)
    - FDs:
        - startTS, Room -> eventName, maxRegistrations, endTS, artist, audience, itemID
    - We assume that EventName, itemID, Artist, or Film can have multiple audiences, which the event organizer determines.
    - BCNF because for X → Y in FDs, X is a key
    - Constraints
        - startTS, room, eventName, endTS, maxRegistrations NOT NULL
        - maxRegistrations >= 0
    - Types
        - startTS, endTS DATETIME
        - maxRegistrations INTEGER  DEFAULT 0
        - room, eventName, audience, maxRegistrations , artist VARCHAR(30)
- EventRegistration(<u>startTS</u>$^{FK\text{-}Event}$, <u>room</u>$^{FK\text{-}Event}$, <u>userID</u>$^{FK\text{-}User}$)
    - FDs: startTS, room-> userID
    - BCNF because for X → Y in FDs, X is a key
    - Constraints
        - StartTimestamp, Room, UserID NOT NULL

# 5 - SQL Schema

## 5.1 - Tables

### 5.1.1 User

```
CREATE TABLE User(
    userID INTEGER PRIMARY KEY,
    firstName VARCHAR(30) NOT NULL,
    lastName VARCHAR(30) NOT NULL,
    age INTEGER NOT NULL,
    fines NUMERIC(5,2) DEFAULT 0,
    CHECK (age >= 7),
    CHECK (fines >= 0)
);
```

### 5.1.2 Librarian

```
CREATE TABLE Librarian(
    librarianID INTEGER PRIMARY KEY,
    firstName VARCHAR(30) NOT NULL,
    lastName VARCHAR(30) NOT NULL,
    salary NUMERIC(6,2) DEFAULT 0,
    department VARCHAR(20) NOT NULL,
    userID REFERENCES User(userID) UNIQUE DEFAULT NULL,
    CHECK (salary >= 0),
    CHECK ((department <> 'volunteer') OR (salary = 0)),
    CHECK (department IN ('admin', 'catalog', 'circulation', 'infotech',
'volunteer'))
);
```

### 5.1.3 BorrowedItem

```
CREATE TABLE BorrowedItem(
    userID REFERENCES User(userID) NOT NULL,
    libraryItemID REFERENCES LibraryItem(libraryItemID) NOT NULL,
    dueDate DATETIME DEFAULT (datetime('now', '+14 day')) NOT NULL,
    returnedDate DATETIME DEFAULT NULL,
    PRIMARY KEY (userID,libraryItemID,dueDate)
```

```
);
```

### 5.1.4 LibraryItem

```
CREATE TABLE LibraryItem(
    libraryItemID INTEGER PRIMARY KEY,
    itemID REFERENCES Item(itemID) NOT NULL,
    toBeAdded BOOLEAN
);
```

### 5.1.5 Item

```
CREATE TABLE Item(
    itemID INTEGER PRIMARY KEY,
    author VARCHAR(30) NOT NULL,
    itemName VARCHAR(30) NOT NULL,
    type VARCHAR(15) NOT NULL,
    CHECK (type IN ('movie', 'book', 'song', 'paper'))
);
```

### 5.1.6 Event

```
CREATE TABLE Event(
    startTS DATETIME NOT NULL,
    endTS DATETIME NOT NULL,
    room VARCHAR(10) NOT NULL,
    eventName VARCHAR(30) NOT NULL,
    audience VARCHAR(20),
    maxRegistrations INTEGER DEFAULT 0,
    artist VARCHAR(30),
    itemID REFERENCES Item(itemID),
    PRIMARY KEY (startTS, room),
    CHECK (maxRegistrations >= 0)
);
```

### 5.1.7 EventRegistration

```
CREATE TABLE EventRegistration(
    startTS REFERENCES Event(startTS) NOT NULL,
    room REFERENCES Event(room) NOT NULL,
```

```
    userID REFERENCES User(userID) NOT NULL,
    PRIMARY KEY (startTS, room, userID)
);
```

# 5.2 - Triggers

## 5.2.1 Preventing returning an item that has already been returned

```
CREATE TRIGGER prevent_multiple_returns
BEFORE UPDATE OF returnedDate ON BorrowedItem
FOR EACH ROW
WHEN NEW.returnedDate IS NOT NULL AND OLD.returnedDate IS NOT NULL
BEGIN
    SELECT RAISE(ABORT, 'The item has already been returned.');
END;
```

## 5.2.2 Preventing users from registering for events at full capacity

```
CREATE TRIGGER prevent_max_registrations
BEFORE INSERT ON EventRegistration
FOR EACH ROW
WHEN (
    SELECT COUNT(*) FROM EventRegistration WHERE startTS = NEW.startTS AND
room = NEW.room
) >= (
    SELECT maxRegistrations FROM Event WHERE startTS = NEW.startTS AND room
= NEW.room
)
BEGIN
    SELECT RAISE(ABORT, 'Event is at max capacity.');
END;
```

## 5.2.3 Preventing users from registering for events they have already registered for

```
CREATE TRIGGER prevent_duplicate_event_registration
BEFORE INSERT ON EventRegistration
FOR EACH ROW
```

```
WHEN (
    SELECT COUNT(*) FROM EventRegistration
    WHERE startTS = NEW.startTS AND room = NEW.room AND userID = NEW.userID
) > 0
BEGIN
    SELECT RAISE(ABORT, 'The user has already registered for this event.');
END;
```

### 5.2.4 Preventing users from registering for past events

```
CREATE TRIGGER prevent_past_event_registration
BEFORE INSERT ON EventRegistration
FOR EACH ROW
WHEN (
    SELECT datetime(startTS) FROM Event WHERE startTS = NEW.startTS AND
room = NEW.room
) < datetime('now')
BEGIN
    SELECT RAISE(ABORT, 'Cannot register for past events.');
END;
```

### 5.2.5 Preventing users from borrowing items that are checked out

```
CREATE TRIGGER prevent_borrow_unavailable_item
BEFORE INSERT ON BorrowedItem
FOR EACH ROW
WHEN (
    SELECT COUNT(*) FROM BorrowedItem
    WHERE libraryItemID = NEW.libraryItemID
    AND returnedDate IS NULL
) > 0
BEGIN
    SELECT RAISE(ABORT, 'The item is not available for borrowing.');
END;
```

### 5.2.6 Preventing users from borrowing items that have not been added yet

```
CREATE TRIGGER prevent_borrow_unavailable_item_tba
BEFORE INSERT ON BorrowedItem
```

```
FOR EACH ROW
WHEN (
    SELECT toBeAdded FROM LibraryItem WHERE libraryItemID =
NEW.libraryItemID
) = 1
BEGIN
    SELECT RAISE(ABORT, 'The item is not available for borrowing.');
END;
```

### 5.2.6 Update user fines once an item is borrowed

```
CREATE TRIGGER add_fine_past_due
AFTER UPDATE ON BorrowedItem
FOR EACH ROW
WHEN (
    datetime('now') > NEW.dueDate
    AND NEW.returnedDate IS NULL
    AND NEW.dueDate IS NOT NULL
)
BEGIN
    UPDATE User
    SET fines = fines + 3.00
    WHERE userID = NEW.userID AND fines = 0.00;
END;
```

### 5.2.7 Prevent users with non-zero fines from borrowing items

```
CREATE TRIGGER prevent_borrow_nonzero_fines
BEFORE INSERT ON BorrowedItem
FOR EACH ROW
WHEN (
    SELECT fines FROM User WHERE userID = NEW.userID
) > 0
BEGIN
    SELECT RAISE(ABORT, 'Users with non-zero fines cannot borrow items.');
END;
```

## 5.3 Indexes

```
CREATE INDEX ck_libraryitem_id ON LibraryItem(itemID);
CREATE INDEX ck_event_room ON Event(room);
CREATE INDEX ck_eventregistration_userid ON EventRegistration(userID);
```

# 6 - Populate Tables

Individual table data can be found in the data directory in this github page: HERE

# 7 - DB Application

To run the application, open a cmd/powershell window in the main.py directory and enter the following line:l

```
python main.py
```