Step (2): Project Specifications (10 points)

In this step, you will define the details of your project. We want you to describe and build **a database for a library**. We only provide you with the following information

- Library has print books, online books, magazines, scientific journals, CDs, records, etc.
- People can borrow the items from the library and return by the due date.
- People may be subject to fines if they do not return items by the due date.
- Library also holds book clubs, book related events, art shows, film screenings, etc.
- Library events are recommended for specific audiences.
- Library events are held in library social rooms.
- People can attend library events for free.
- Library also has personnel and record keeping for personnel.
- Library also keeps records of items (books, etc.) that might be added to the library in the future.

The rest of the definitions of the domain is your work. Approach it as a real-world phase of finding out about a database as we described in the Entity Relationship Modelling discussions. Find out about the needs and requirements of a library database and specify it with simple words.

1. **What data is going to be stored?**
    a. Library Catalogue
        i. Physical collections
        ii. Course reserves
        iii. Ebooks
        iv. Journal articles
    b. Library Account
        i. Account number
        ii. Fines
        iii. Personal Information
            1. Address
            2. Phone number
        iv. Borrowing history
        v. Holds
        vi. Checked-out items

            1. Check out due dates
            2. Pick-up locations
    c. Library Services
        i. Hours of operation
        ii. Events
            1. Target audiences
            2. Locations
            3. Price
        iii. Room bookings
        iv. Printing
        v. Loans policies
        vi. Other resources
        vii. Digitized collections
    d. Library Personnel
        i. Employee info: id, address, salary, position,...
    e. Acquisitions list
2. **What are we going to do with the data?**
    a. Store
    b. Analyze
    c. Display
3. **Who should access the data?**
    a. Library personnel: almost everything
    b. Customers
        i. With accounts: personal account information, library catalogue, services.
        ii. Without accounts: library catalogue, services.

A database for a library.

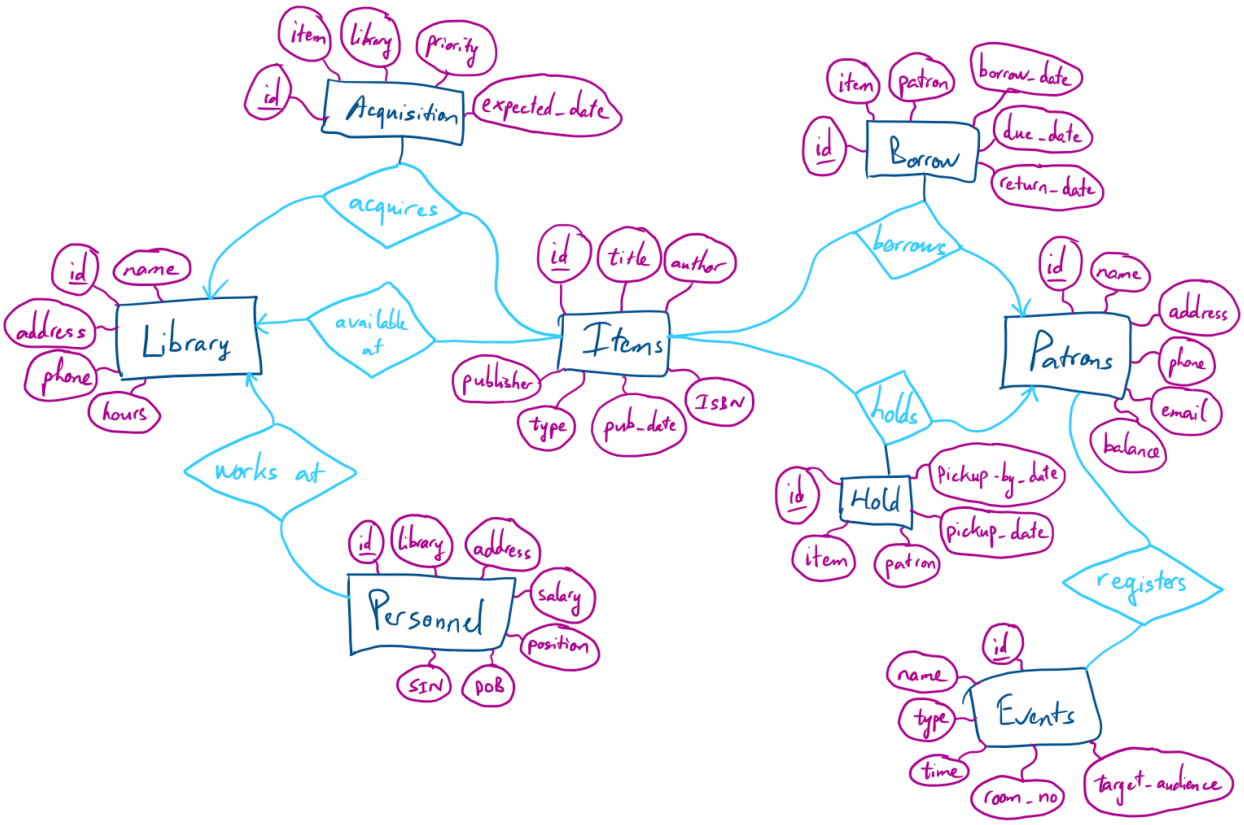
- A catalogue is a list of all **items** in a library.
    - For each item, we may want to keep: title, author, publisher, type, publication date, ISBN.
    - Each item may be <u>available</u> at one or more libraries. For each availability, we may keep: *library*, *item*, quantity.
- A **library** may have: name, address, hours, phone number.
- Library **patrons** may have: names, addresses, phone numbers, emails, account number, and balances *(for fines)*.

- A patron can borrow items from a library. We store each borrowing transaction that's made:
    - *Item*
    - *Patron*
    - Borrowing date
    - Due date
    - Return date
- A patron can hold multiple items. Each hold may have: *item*, *patron*, pickup-by date, pickup date.
- A patron may register for an event. An **event** may have: name, type, time, location *(room #)*, target audience.
- A **personnel** works at a library and may have: id, *library*, address, salary, position, SIN, DOB.
- A library may acquire items. Each acquisition may contain: *item*, *library*, priority, expected date.

Step (3): E/R Diagrams (15 points)

In this step, you should define your entity-relationship model and draw your E/R diagrams. You can use any drawing software of choice for this step, or draw by hand and scan if it would be clean and readable.

- Please use the ER diagram notations described in the class.
- Try to make your application interesting and examine your knowledge by using a variety of different entities, attributes, and relationships with different key and participation constraints.



Step (4): Does your design allow anomalies? (15 points)

In this step you will review your design as it translates to schemas. You should decide whether your design meets the requirements we discussed to avoid anomalies. To perform this task, you have to analyze all non-trivial functional dependencies in your database based on characteristics of your data, and ensure your tables are free of bad functional dependencies.

Either your design does not allow anomalies, or you need to re-design. If your design does not allow anomalies, that is the output. Otherwise, repeat decomposition, until you have your database in BCNF. Provide your written work explaining why your final design does not have anomalies (FDs, BCNF, prove no bad FDs)

Schema:

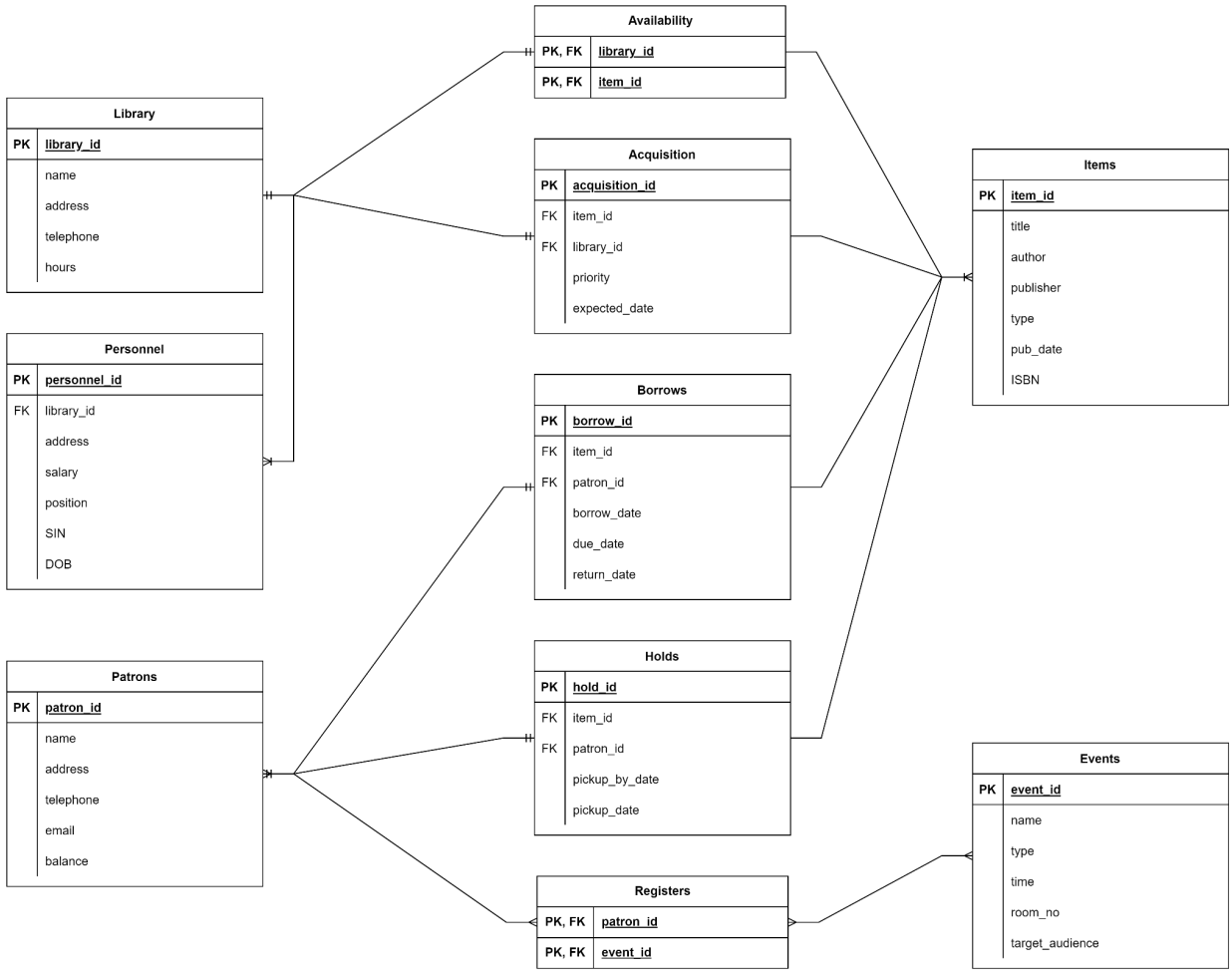
- **Items** { item_id, title, author, publisher, type, pub_date, ISBN }
- **Library** { library_id, name, address, hours, telephone}
- **Patrons** { patron_id, name, address, telephone, email, balance }
- **Events** { event_id, name, type, time, room_no, target_audience }
- **Personnel** { personnel_id, library_id$^{FK\text{-}Library}$, address, salary, position, SIN, DOB }
- **Organizes** { library_id$^{FK\text{-}Library}$, event_id$^{FK\text{-}Events}$ }
- **Availability** { item_id$^{FK\text{-}Items}$, library_id$^{FK\text{-}Library}$ }
- **Acquisition** { acquisition_id, item_id$^{FK\text{-}Items}$, library_id$^{FK\text{-}Library}$, priority, expected_date }
- **Borrows** { borrow_id, item_id$^{FK\text{-}Items}$, patron_id$^{FK\text{-}Patrons}$, borrow_date, due_date, return_date }
- **Holds** { hold_id, item_id$^{FK\text{-}Items}$, pickup_by_date, pickup_date, patron_id$^{FK\text{-}Patrons}$ }

Functional dependencies:

- Items
  - item_id -> title, author, publisher, type, pub_date, ISBN
  - title, author, publisher, type, pub_date, ISBN -> item_id
- Library
  - library_id -> address
  - address -> library_id
  - address -> name, hours, telephone
- Patrons
  - patron_id -> email
  - email -> patron_id
  - email -> name, address, telephone, balance
- Events
  - event_id -> name, time, room_no
  - name, time, room_no -> event_id
  - name, time, room_no -> type, target_audience
- Personnel
  - personnel_id -> library_id
  - personnel_id -> SIN
  - SIN -> personnel_id
  - SIN -> address, salary, position, DOB
- Availability

    - item_id -> library_id
- Acquisition
    - acquisition_id -> item_id, library_id
    - item_id, library_id -> acquisition_id
    - item_id, library_id -> priority, expected_date
- Borrows
    - borrow_id -> item_id, patron_id, borrow_date, due_date, return_date
- Holds
    - hold_id -> item_id, patron_id, pickup_by_date, pickup_date

The schema is in BCNF because all the functional dependencies listed above have their l.h.s. as superkeys for their respective relations.



Step (5): SQL Schema (15 points)

Create your database and convert your ERDs to table schemas in this database using sqlite. Make sure you have the required constraints and triggers in place to ensure the integrity of your data.

```
PRAGMA foreign_keys = ON;

CREATE TABLE "Personnel" (
      "personnel_id"    INTEGER NOT NULL UNIQUE,
      "library_id"      INTEGER NOT NULL,
      "name"      TEXT NOT NULL,
      "address"   TEXT,
      "salary"    INTEGER NOT NULL DEFAULT 0,
      "position"  TEXT NOT NULL,
      "sin" INTEGER UNIQUE,
      "dob" DATE,
      PRIMARY KEY("personnel_id" AUTOINCREMENT),
      FOREIGN KEY("library_id") REFERENCES "Library"("library_id"),
      UNIQUE (sin),
      CHECK (salary >= 0),
      CHECK (sin BETWEEN 100000000 and 999999999) -- 9-digit SIN.
);
CREATE TRIGGER trigger_check_dob
BEFORE INSERT ON "Personnel"
FOR EACH ROW
WHEN NEW.dob > DATE("now")
BEGIN
  SELECT RAISE(ABORT, "Can't have a personnel born in the future");
END;

CREATE TABLE "Patrons" (
      "patron_id" INTEGER NOT NULL UNIQUE,
      "name"      TEXT NOT NULL,
      "address"   TEXT,
      "telephone" TEXT,
      "email"     TEXT,
      "balance"   REAL,
      PRIMARY KEY("patron_id" AUTOINCREMENT),
      UNIQUE (email)
```

```
);
CREATE TABLE "Events" (
	"event_id"  INTEGER NOT NULL UNIQUE,
	"name"	    TEXT NOT NULL,
	"type"	    TEXT NOT NULL,
	"time"	    DATETIME,
	"room_no"   INTEGER NOT NULL,
	"target_audience" TEXT,
	PRIMARY KEY("event_id" AUTOINCREMENT),
	UNIQUE (name, time, room_no),
	CHECK (room_no >= 0)
);
CREATE TABLE "Items" (
	"item_id"   INTEGER NOT NULL UNIQUE,
	"title"	    TEXT NOT NULL,
	"author"    TEXT,
	"publisher" TEXT,
	"type"	    TEXT NOT NULL,
	"pub_date"  DATE,
	"isbn"	    INTEGER,
	PRIMARY KEY("item_id" AUTOINCREMENT),
	UNIQUE (title, author, publisher, type, pub_date, isbn),
	CHECK (isbn BETWEEN 1000000000000 and 9999999999999) -- 13-digit
ISBN.
);
CREATE TRIGGER trigger_check_pub_date
BEFORE INSERT ON "Items"
FOR EACH ROW
WHEN NEW.pub_date > DATE("now")
BEGIN
  SELECT RAISE(ABORT, "Publication date must be today or earlier");
END;

CREATE TABLE "Availability" (
	"library_id"	  INTEGER NOT NULL,
	"item_id"   INTEGER NOT NULL,
	PRIMARY KEY("library_id","item_id"),
	FOREIGN KEY("library_id") REFERENCES "Library"("library_id") ON
DELETE CASCADE,
	FOREIGN KEY("item_id") REFERENCES "Items"("item_id") ON DELETE
CASCADE
);
CREATE TRIGGER trigger_return_an_item
```

```
AFTER INSERT ON "Availability"
BEGIN
      UPDATE "Borrows"
      SET return_date = DATE('now')
      WHERE item_id = NEW.item_id;
END;

CREATE TABLE "Registers" (
      "patron_id" INTEGER NOT NULL,
      "event_id"  INTEGER NOT NULL,
      PRIMARY KEY("patron_id","event_id"),
      FOREIGN KEY("event_id") REFERENCES "Events"("event_id") ON DELETE
CASCADE,
      FOREIGN KEY("patron_id") REFERENCES "Patrons"("patron_id") ON DELETE
CASCADE
);
CREATE TABLE "Acquisition" (
      "acquisition_id"  INTEGER NOT NULL UNIQUE,
      "item_id"   INTEGER NOT NULL,
      "library_id"      INTEGER NOT NULL,
      "priority"  INTEGER NOT NULL,
      "expected_date"   DATE,
      PRIMARY KEY("acquisition_id" AUTOINCREMENT),
      FOREIGN KEY("item_id") REFERENCES "Items"("item_id"),
      FOREIGN KEY("library_id") REFERENCES "Library"("library_id"),
      UNIQUE(item_id, library_id),
      CHECK (priority >= 0)
);
CREATE TRIGGER trigger_check_expected_acquisition_date
BEFORE INSERT ON "Acquisition"
FOR EACH ROW
WHEN NEW.expected_date < DATE("now")
BEGIN
  SELECT RAISE(ABORT, "Expected acquisition date must be today or later");
END;

CREATE TABLE "Borrows" (
      "borrow_id" INTEGER NOT NULL UNIQUE,
      "item_id"   INTEGER NOT NULL,
      "patron_id" INTEGER NOT NULL,
      "borrow_date"     DATE NOT NULL,
      "due_date"  DATE,
      "return_date"     DATE,
```

```
      PRIMARY KEY("borrow_id" AUTOINCREMENT),
      FOREIGN KEY("patron_id") REFERENCES "Patrons"("patron_id"),
      FOREIGN KEY("item_id") REFERENCES "Items"("item_id"),
      CHECK (borrow_date <= due_date),
      CHECK (borrow_date <= return_date)
);
CREATE TRIGGER trigger_borrow_an_item
AFTER INSERT ON "Borrows"
BEGIN
      DELETE FROM "Availability"
      WHERE item_id = NEW.item_id;
END;

CREATE TRIGGER trigger_fine_for_late_return
AFTER UPDATE ON "Borrows"
WHEN NEW.return_date > NEW.due_date
BEGIN
      UPDATE "Patrons"
      SET balance = balance + 10
      WHERE patron_id = NEW.patron_id;
END;

CREATE TABLE "Holds" (
      "hold_id"   INTEGER NOT NULL UNIQUE,
      "item_id"   INTEGER NOT NULL,
      "patron_id" INTEGER NOT NULL,
      "pickup_by_date"  DATE,
      "pickup_date"     DATE,
      PRIMARY KEY("hold_id" AUTOINCREMENT),
      FOREIGN KEY("item_id") REFERENCES "Items"("item_id"),
      FOREIGN KEY("patron_id") REFERENCES "Patrons"("patron_id"),
      CHECK (pickup_date <= pickup_by_date)
);
CREATE TABLE "Library" (
      "library_id"      INTEGER NOT NULL UNIQUE,
      "name"      TEXT NOT NULL,
      "address"   TEXT NOT NULL,
      "hours"     TEXT NOT NULL,
      "telephone" TEXT,
      PRIMARY KEY("library_id" AUTOINCREMENT),
      UNIQUE (address)
);
```

Step (6): Populate Tables (10 points)

Generate and insert at least 10 realistic tuples, based on your project description, into each of the tables in your database. You can use random custom data generators or use available real data. The choice is up to you.

```
INSERT INTO Library (name, address, telephone, hours)
VALUES
 ("Verne Library - Atlantis Branch", "123 Ocean Avenue, Atlantis",
"123-456-7890", "Mon-Fri: 9 AM - 6 PM, Sat: 10 AM - 4 PM"),
 ("Verne Library - Stapi Branch", "56 Volcano Street, Stapi",
"987-654-3210", "Tue-Sat: 10 AM - 5 PM"),
 ("Verne Library - Yokohama Branch", "7 Cherry Blossom Lane,
Yokohama", "+81-9-8765-4321", "Mon, Wed, Fri: 11 AM - 7 PM, Sat-Sun:
9 AM - 3 PM"),
 ("Verne Library - Ilium Branch", "789 Comet Avenue, Ilium",
"246-813-5790", "Mon-Thu: 10 AM - 8 PM, Fri: 11 AM - 5 PM"),
 ("Verne Library - Villeurbanne Branch", "10 Progress Lane,
Villeurbanne", "+33-4-5678-9012", "Mon-Sat: 9 AM - 6 PM"),
 ("Verne Library - Esquimaux Branch", "100 Iceberg Road, Esquimaux
Village", "123-555-1212", "Tue-Sat: 10 AM - 4 PM"),
 ("Verne Library - Huntstown Branch", "987 Rocket Street, Huntstown",
"+1-800-1234-5678", "Mon-Fri: 8 AM - 5 PM"),
 ("Verne Library - Kôr Branch", "369 Lost City Street, Kôr",
"+212-987-6543-210", "Wed-Sat: 12 PM - 8 PM"),
 ("Verne Library - Kholby Branch", "789 Indian Bazaar Road, Kholby",
"+91-80-1357-2468", "Mon-Fri: 9 AM - 5 PM, Sat: 10 AM - 2 PM"),
 ("Verne Library - Ahtotal Branch", "(Virtual Library)", NULL, "Open
24/7");
INSERT INTO Personnel (library_id, name, address, salary, position,
sin, dob)
VALUES
 (1, "Bob", "15 Nautilus Lane, Nemo City", 50000, "Librarian",
"123456789", "1985-05-15"),
 (2, "Billy", "9 Phileas Fogg Street, London", 48000, "Assistant
Librarian", "234567890", "1990-09-21"),
```

```
 (3, "Santa", "55 Captain Nemo Road, Atlantis", 55000, "Head
Librarian", "345678901", "1982-03-10"),
 (4, "Greg", NULL, 52000, "Library Technician", NULL, NULL),
 (5, "Rob", "7 Round World Avenue, Paris", 49000, "Archivist",
"567890123", "1995-07-30"),
 (6, "John", "30 Phileas Fogg Street, A Foggy City", 47000,
"Circulation Clerk", "678901234", "1992-11-18"),
 (7, "Lilly", "3 Mysterious Island Way, Uncharted Land", 53000,
"Reference Librarian", "789012345", "1987-06-25"),
 (8, "Jane", "11 Jules Verne Street, Vernesville", 51000,
"Cataloger", "890123456", "1991-04-08"),
 (9, "Hoho", "20 Sea Leagues Avenue, Oceanopolis", 54000, "Youth
Services Librarian", "901234567", "1984-08-12"),
 (10, "Cameron", "18 Moon Crescent, Spaceport", 48000, "Media
Specialist", "912345678", "1993-10-02");
INSERT INTO Patrons (name, address, telephone, email, balance)
VALUES
 ("Alice Johnson", "25 Steampunk Lane, Clocksville", "123-456-7890",
"alice@example.com", 0),
 ("Bob Smith", "10 Airship Avenue, Zeppelintown", "987-654-3210",
"bob@example.com", 25.50),
 ("Eve Brown", "3 Gears Drive, Cogsworthville", "444-555-6666",
"eve@example.com", 10),
 ("Charles Williams", "8 Brass Street, Gearsville", "777-888-9999",
"charles@example.com", 5),
 ("Olivia Turner", "12 Victorian Terrace, Retroville",
"222-333-4444", "olivia@example.com", 100),
 ("James Miller", "6 Clockwork Crescent, Ticktock City",
"666-777-8888", "james@example.com", 0),
 ("Sophia Davis", "15 Steam Engine Road, Brassburg", "111-222-3333",
"sophia@example.com", 50.75),
 ("William Wilson", "1 Gadget Lane, Gizmotown", "999-888-7777",
"william@example.com", 15),
 ("Ava Martinez", "7 Inventor Street, Mechanism City",
"555-444-3333", "ava@example.com", 30.25),
 ("Liam Anderson", "4 Contraption Close, Widgetville",
"333-222-1111", "liam@example.com", -5);
INSERT INTO Events (name, type, time, room_no, target_audience)
VALUES
```

```
 ("Time Travel Symposium", "Conference", "2023-08-15", "A101",
"Researchers"),
 ("Journey to the Center of the Earth", "Lecture", "2023-09-10",
"B205", "General Public"),
 ("Exploring the Abyss", "Workshop", "2023-08-28", "C306",
"Students"),
 ("20,000 Leagues Under the Sea", "Movie Screening", "2023-09-05",
"A101", "All Ages"),
 ("Steampunk Extravaganza", "Festival", "2023-08-22", "Courtyard",
"All Ages"),
 ("Inventors Meetup", "Networking", "2023-09-12", "A101", "Inventors
and Innovators"),
 ("Victorian Literature Seminar", "Seminar", "2023-08-31", "B205",
"Academics"),
 ("Clockwork Creations Showcase", "Exhibition", "2023-09-15",
"Atrium", "Art Enthusiasts"),
 ("Automaton Robotics Demo", "Demonstration", "2023-08-25", "C306",
"Tech Enthusiasts"),
 ("Intergalactic Fashion Show", "Fashion Show", "2023-09-18",
"Courtyard", "Fashion Enthusiasts");
INSERT INTO Items (title, author, publisher, type, pub_date, isbn)
VALUES
 ("Twenty Thousand Leagues Under the Sea", "Jules Verne",
"Pierre-Jules Hetzel", "Book", "1870-01-27", 9781503292382),
 ("The Time Machine", "H.G. Wells", "William Heinemann", "Book",
"1895-05-07", 9781499749686),
 ("Metropolis", "Thea von Harbou", "August Scherl", "Book",
"1926-01-10", 9781773233626),
 ("The Cat Returns", "Hiroyuki Morita", "Studio Ghibli", "Movie",
"2002-07-20", 9781598162495),
 ("Avatar: The Last Airbender - The Promise", "Gene Luen Yang", "Dark
Horse Comics", "Comic", "2012-01-25", 9781616550745),
 ("Around the World in Eighty Days", "Jules Verne", "Pierre-Jules
Hetzel", "Book", "1873-01-30", 9780486411115),
 ("Castle in the Sky", "Hayao Miyazaki", "Toei Company", "Movie",
"1986-08-02", 9781421565993),
 ("The Difference Engine", "William Gibson and Bruce Sterling",
"Gollancz", "Book", "1990-06-01", 9780140179419),
 ("The City & the City", "China Miéville", "Macmillan", "Book",
```

```
"2009-05-26", 9780330534192),
 ("Wild Wild West", "Jim Thomas and John Thomas", "Warner Bros.",
"Movie", "1999-06-30", 9780790730202);
INSERT INTO Availability (library_id, item_id)
VALUES
 (1, 1),
 (2, 2),
 (3, 3),
 (1, 4),
 (2, 5),
 (3, 6),
 (1, 7),
 (2, 8),
 (3, 9),
 (1, 10);
INSERT INTO Registers (patron_id, event_id)
VALUES
 (1, 1),
 (2, 2),
 (3, 3),
 (1, 4),
 (2, 5),
 (3, 6),
 (1, 7),
 (2, 8),
 (3, 9),
 (1, 10);
INSERT INTO Acquisition (item_id, library_id, priority,
expected_date)
VALUES
 (1, 1, 2, "2023-08-20"),
 (2, 2, 1, "2023-08-25"),
 (3, 3, 3, "2023-09-01"),
 (4, 1, 1, "2023-08-22"),
 (5, 2, 2, "2023-08-28"),
 (6, 3, 3, "2023-09-05"),
 (7, 1, 2, "2023-08-24"),
 (8, 2, 1, "2023-08-30"),
 (9, 3, 3, "2023-09-08"),
```

```
 (10, 1, 1, "2023-08-26");
INSERT INTO Borrows (item_id, patron_id, borrow_date, due_date,
return_date)
VALUES
 (1, 1, "2013-08-10", "2013-08-24", "2023-08-24"),
 (2, 2, "2020-08-15", "2020-08-29", "2023-08-29"),
 (3, 3, "2022-08-20", "2022-09-03", "2023-09-03"),
 (4, 4, "2021-08-12", "2021-08-26", "2023-08-26"),
 (5, 5, "2020-08-18", "2023-06-01", NULL),
 (6, 6, "2021-08-25", "2023-05-08", NULL),
 (7, 7, "2023-02-22", "2023-03-05", NULL),
 (8, 8, "2022-08-28", "2023-04-11", NULL),
 (9, 9, "2023-01-30", "2023-02-13", NULL),
 (10, 10, "2023-05-27", "2023-09-10", NULL);
INSERT INTO Holds (item_id, patron_id, pickup_by_date, pickup_date)
VALUES
 (1, 1, "2023-08-15", "2023-08-12"),
 (2, 2, "2023-08-20", NULL),
 (3, 3, "2023-08-25", NULL),
 (4, 4, "2023-08-18", "2023-08-16"),
 (5, 5, "2023-08-24", NULL),
 (6, 6, "2023-08-29", NULL),
 (7, 7, "2023-08-23", "2023-08-21"),
 (8, 8, "2023-08-30", NULL),
 (9, 9, "2023-09-01", NULL),
 (10, 10, "2023-08-28", "2023-08-25");
```

Step (7): Build Your Database Application (20 points)

Use python and sqlite to build your database application to allow a library user to:

- Find an item in the library
- Borrow an item from the library
- Return a borrowed item
- Donate an item to the library
- Find an event in the library
- Register for an event in the library

- Volunteer for the library
- Ask for help from a librarian

```
import datetime
import random
import sqlite3

conn = sqlite3.connect('library.db')
cursor = conn.cursor()
cursor.execute("PRAGMA foreign_keys = ON;")  # Enforce foreign key
constraints.
print("Opened database successfully")

def borrow_item(cursor, item_id: int, patron_id: int) -> None:
    borrow_date = datetime.date.today()
    due_date = borrow_date + datetime.timedelta(weeks=2)
    cursor.execute("INSERT INTO Borrows (item_id, patron_id, borrow_date,
due_date) VALUES (?, ?, ?, ?);",
                   (item_id, patron_id, borrow_date, due_date))
    conn.commit()
    print("Your item is checked out.")
    print("Keep your item id when returning:", item_id)
    print("Due date:", due_date)

def return_item(cursor, item_id: int) -> None:
    library_id = random.randint(1, 10)
    cursor.execute("INSERT INTO Availability (library_id, item_id) VALUES
(?, ?);",
                   (library_id, item_id))
    conn.commit()
    print("Item returned successfully.")
    print("Please check your current balance for any late fines.")

def donate_item(cursor, title: str, author: str, pub_date: str, item_type:
str, isbn: int) -> None:
    cursor.execute("INSERT INTO Items (title, author, pub_date, type, isbn)
VALUES (?, ?, ?, ?, ?);",
                   (title, author, pub_date, item_type, isbn))
    cursor.execute("INSERT INTO Availability (library_id, item_id) VALUES
(?, (SELECT MAX(item_id) FROM Items));",
                   (random.randint(1, 10),))
    conn.commit()
```

```
    print("Item added to collections.")
    print("Thank you for your contributions!")

def register_event(cursor, event_id: int, patron_id: int) -> None:
    cursor.execute("INSERT INTO Registers (event_id, patron_id) VALUES (?,
?);", (event_id, patron_id))
    conn.commit()
    print("You are registered for the event.")
    print("See you soon.")

def volunteer(cursor, name: str, address: str, dob: str) -> None:
    cursor.execute("INSERT INTO Personnel (library_id, name, address,
position, dob) VALUES (?, ?, ?, ?, ?);",
                   (random.randint(1, 10), name, address, "Volunteer",
dob))
    conn.commit()
    print("Welcome to the team.")
    print("Thank you for volunteering!")

def ask_librarian(cursor) -> None:
    print()
    print("Available librarians:")
    all_librarians = cursor.execute("SELECT personnel_id, name, position
FROM Personnel WHERE position LIKE ?;",
                                    ("%Librarian%",)).fetchall()
    prettyprint(all_librarians)
    if all_librarians:
        librarian_id = input("Please enter the id (first number) of the
librarian you would like to ask for help: ")
        print(f"Your request has been assigned to librarian {librarian_id}.
Please monitor your inbox for replies.")

def prettyprint(tuples: list[tuple]) -> None:
    if tuples:
        for tuple in tuples:
            print(tuple)
    else:
        print("No records found.")

# Constants for menu options
OPTION_EXIT = '0'
OPTION_FIND_ITEM = '1'
OPTION_BORROW_ITEM = '2'
```

```
OPTION_RETURN_ITEM = '3'
OPTION_DONATE_ITEM = '4'
OPTION_FIND_EVENT = '5'
OPTION_REGISTER_EVENT = '6'
OPTION_VOLUNTEER = '7'
OPTION_ASK_LIBRARIAN = '8'

def get_main_menu() -> str:
    print()
    print('Select one of the following options.')
    print()
    print('1: Find an item in the library')
    print('2: Borrow an item from the library')
    print('3: Return a borrowed item')
    print('4: Donate an item to the library ')
    print('5: Find an event in the library')
    print('6: Register for an event in the library')
    print('7: Volunteer for the library')
    print('8: Ask for help from a librarian')
    print('0: Exit')
    return input('-> ')

def find_records_by_field(cursor, table: str, field: str, value: str,
is_string: bool) -> list[tuple]:
    if is_string:
        # e.g. WHERE title LIKE "%ar%"
        where_clause = f"WHERE {field} LIKE \"%{value}%\""
    else:
        # e.g. WHERE ISBN = 90238098240
        where_clause = f"WHERE {field} = {value}"
    query = f"SELECT * FROM {table} {where_clause};"
    return cursor.execute(query).fetchall()

def get_item_search_menu() -> str:
    print()
    print('Search items with one of the following options.')
    print()
    print('1: title')
    print('2: author')
    print('3: publisher')
    print('4: type ')
    print('5: publication date')
    print('6: ISBN')
```

```
    print('0: Exit')
    return input('-> ')

def find_items_by_field(cursor, field: str, value: str, is_string: bool) ->
list[tuple]:
    return find_records_by_field(cursor, 'Items', field, value, is_string)

def input_items_to_find(find_item_option: str) -> list[tuple]:
    match find_item_option:
        case '1':
            # Find by title
            search_value = input("Enter item's title to search: ")
            return find_items_by_field(cursor, "title", search_value, True)
        case '2':
            # Find by author
            search_value = input("Enter item's author to search: ")
            return find_items_by_field(cursor, "author", search_value,
True)
        case '3':
            # Find by publisher
            search_value = input("Enter item's publisher to search: ")
            return find_items_by_field(cursor, "publisher", search_value,
True)
        case '4':
            # Find by type
            search_value = input("Enter item's type to search: ")
            return find_items_by_field(cursor, "type", search_value, True)
        case '5':
            # Find by pub_date
            search_value = input("Enter item's publishing date to search:
")
            return find_items_by_field(cursor, "pub_date", search_value,
True)
        case '6':
            # Find by ISBN
            search_value = input("Enter item's ISBN to search: ")
            return find_items_by_field(cursor, "ISBN", search_value, False)

def quote(str: str) -> str:
    return "\"" + str + "\""

def get_event_search_menu() -> str:
    print()
```

```
    print('Search events with one of the following options.')
    print()
    print('1: event id')
    print('2: name')
    print('3: event type')
    print('4: date')
    print('5: room number')
    print('6: target audience')
    print('0: Exit')
    return input('-> ')

def find_events_by_field(cursor, field: str, value: str) -> list[tuple]:
    return find_records_by_field(cursor, 'Events', field, value, True)

def input_events_to_find(find_event_option: str) -> list[tuple]:
    match find_event_option:
        case '1':
            # Find by event id
            search_value = input("Enter event id to search: ")
            return find_events_by_field(cursor, "event_id", search_value)
        case '2':
            # Find by name
            search_value = input("Enter event name to search: ")
            return find_events_by_field(cursor, "name", search_value)
        case '3':
            # Find by event type
            search_value = input("Enter event type to search: ")
            return find_events_by_field(cursor, "type", search_value)
        case '4':
            # Find by date
            search_value = input("Enter event's date to search: ")
            return find_events_by_field(cursor, "time", search_value)
        case '5':
            # Find by room number
            search_value = input("Enter room number to search: ")
            return find_events_by_field(cursor, "room_no", search_value)
        case '6':
            # Find by target audience
            search_value = input("Enter target audience to search: ")
            return find_events_by_field(cursor, "target_audience",
search_value)

def main() -> None:
```

```
    '''Main loop'''
    while True:
        main_menu_option = get_main_menu()

        if main_menu_option == OPTION_EXIT:
            break

        elif main_menu_option == OPTION_FIND_ITEM:
            # Find items.
            items = input_items_to_find(get_item_search_menu())
            prettyprint(items)

        elif main_menu_option == OPTION_BORROW_ITEM:
            # Borrow an item.
            items = input_items_to_find(get_item_search_menu())
            prettyprint(items)
            if items:
                # Borrow this item.
                item_id = input("Enter the id (first number) of the item
you want to borrow: ")
                patron_id = input("Enter your account number / patron id:
")
                borrow_item(cursor, item_id, patron_id)

        elif main_menu_option == OPTION_RETURN_ITEM:
            # Return a borrowed item
            item_id = input("Please enter the id of the item you are
returning: ")
            return_item(cursor, item_id)

        elif main_menu_option == OPTION_DONATE_ITEM:
            # Donate an item to the library
            title = input('Please enter the title: ')
            author = input('Please enter the author: ')
            pub_date = input('Please enter the publication date: ')
            item_type = input('Please enter the item type: ')
            isbn = input('Please enter a 13-digit ISBN number: ')
            donate_item(cursor, title, author, pub_date, item_type, isbn)

        elif main_menu_option == OPTION_FIND_EVENT:
            # Find an event in the library
            events = input_events_to_find(get_event_search_menu())
            prettyprint(events)
```

```
        elif main_menu_option == OPTION_REGISTER_EVENT:
            # Register for an event in the library
            events = input_events_to_find(get_event_search_menu())
            prettyprint(events)
            if events:
                # Register for this event.
                event_id = input("Enter the id (first number) of the event
you want to register: ")
                patron_id = input("Enter your account number / patron id:
")
                register_event(cursor, event_id, patron_id)

        elif main_menu_option == OPTION_VOLUNTEER:
            # Volunteer for the library
            name = input('Please enter your name: ')
            address = input('Please enter your address: ')
            dob = input('Please enter your date of birth: ')
            volunteer(cursor, name, address, dob)

        elif main_menu_option == OPTION_ASK_LIBRARIAN:
            # Ask for help from a librarian
            ask_librarian(cursor)

if __name__ == "__main__":
    main()

conn.close()
```