

Mini Project Documentation

by:

Auston Ng, 301565494, acn8@sfu.ca

Minh Hoang Le (Ted), 301477912, hml15@sfu.ca

▼ Step (2): Project Specifications

Our database system is designed to manage the operations of a modern public library that offers a diverse collection of resources and services to its community.

Resources Management

- The library maintains a collection of various items including print books, online books, magazines, scientific journals, CDs, and records.
- Each resource is categorized and tracked with relevant information such as title, publication date, and availability status.
- Authors are tracked and associated with their respective works.
- Resources can be classified into multiple categories to facilitate browsing and searching.

Lending System

- Library members can borrow items from the collection for a specified time period (default 14 days).
- The system tracks all current and past loans, including loan date, due date, and return date.
- Items have a status that indicates whether they are available or unavailable.
- Members are limited to 2 active loans at any given time.

Fine Management

- Overdue items incur a daily fine of \$0.50 until they are returned.
- The system automatically calculates and records fines when items are returned late.
- Members with unpaid fines cannot borrow additional items until their fines are paid.

Event Management

- The library hosts various events such as book clubs, art shows, and film screenings.
- Events are categorized by type and have designated target audiences.
- Events take place in specific rooms within the library.
- Members can register to attend events free of charge.

Personnel Management

- The library maintains records of its personnel, including librarians, volunteers, and administrators.
- Personnel information includes hire date, job title, salary, and contact information.
- Personnel members can host and manage library events.

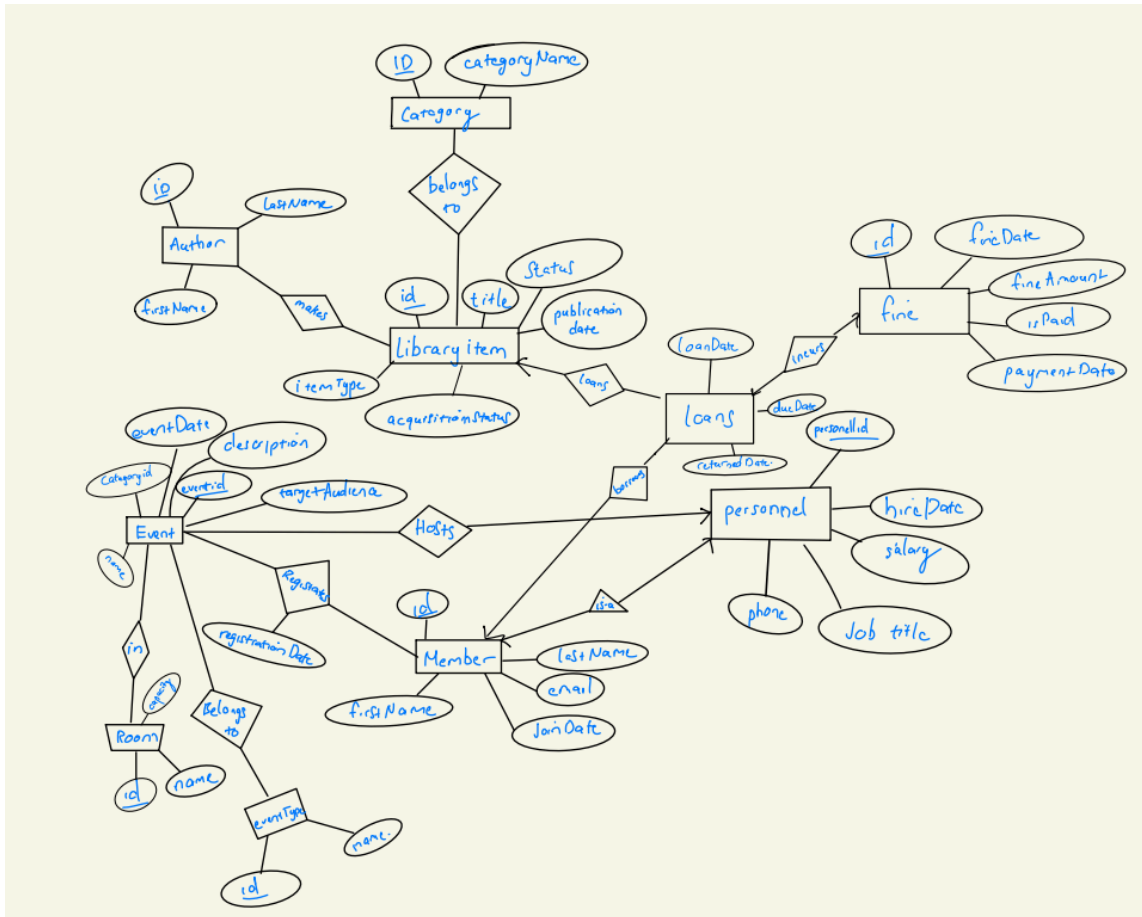
Acquisition Planning

- The system tracks items that might be added to the library collection in the future.
- Items can be marked with an acquisition status of "planned" or "owned".

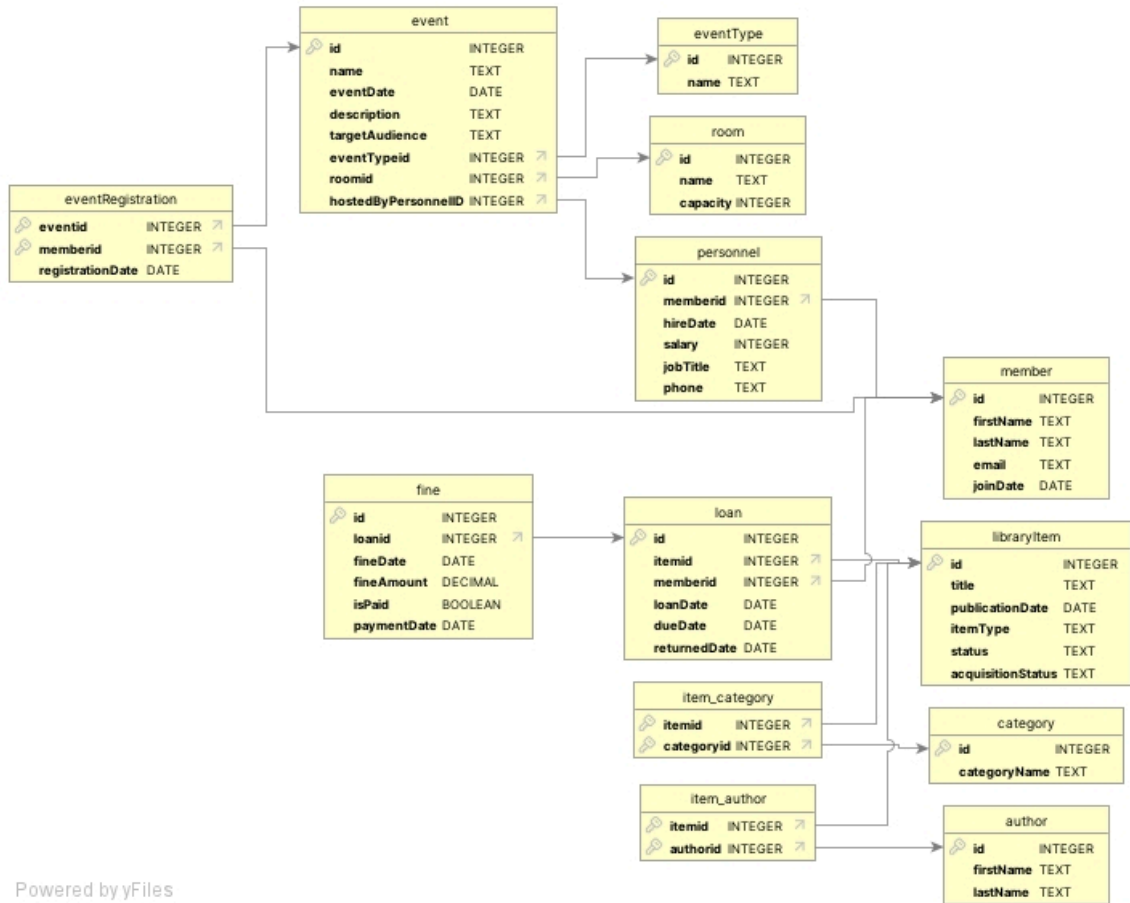
This specification provides a foundation for developing a comprehensive database system that will effectively support the library's daily operations and future growth.

▼ Step (3): E/R Diagrams

- Here is an E/R diagram that describes the library schema:



- Here is a hierarchical relational diagram showing the structure of database tables, with their attributes and relationships on a high level



Powered by yFiles

▼ Step (4): Does your design allow anomalies?

- For each table in the schema we will check if potential FDs violate BCNF:

▼ libraryItem

Columns: (id, title, publicationDate, itemType, status, acquisitionStatus)

- Primary Key:** id (unique identifier)
- FDs:** Say, $id \rightarrow (title, publicationDate, itemType, status, acquisitionStatus)$
- Is it BCNF?**
 - Any FD must have id on the LHS, because id is the ONLY unique identifier
 - And since id is a key, all FDs satisfy BCNF
 - So, there are no anomalies, meaning the table is in BCNF



author

Columns: (id, firstName, lastName)

- **Primary Key:** id
- **FDs:** $\text{id} \rightarrow (\text{firstName}, \text{lastName})$
- **Is it BCNF?**
 - Yes, the only FD has the primary key on the LHS



category

Columns: (id, categoryName)

- **Primary Key:** id
- **FD:** $\text{id} \rightarrow \text{categoryName}$
- **BCNF?**
 - Yes, id is the key, so no violation



item_author

Columns: (itemid, authorid)

- **Primary Key:** (itemid, authorid)
- **FD:** $(\text{itemid}, \text{authorid}) \rightarrow (\text{itemid}, \text{authorid})$ (note: this is trivial)
- **BCNF?**
 - Yes, there are no partial dependencies (the composite key is the entire table)



item_category

Columns: (itemid, categoryid)

- **Primary Key:** (itemid, categoryid)
- **FD:** Similar to item_author, the only FD is the full key
- **BCNF?**
 - Yes, for the same reason as item_author

▼ member

Columns: (id, firstName, lastName, email, joinDate)

- **Primary Key:** id
- **FDs:**
 - id → (firstName, lastName, email, joinDate)
 - email → (id, firstName, lastName, joinDate)
 - email is unique, so it can act as an alternate key
- **BCNF?**
 - Yes. Both id and email are keys, and any FD is key-based.

▼ loan

Columns: (id, itemid, memberid, loanDate, dueDate, returnedDate)

- **Primary Key:** id
- **FD:** id → (itemid, memberid, loanDate, dueDate, returnedDate)
 - Since members can keep borrowing the same libraryItem (as long as it's available) (itemid, memberid) alone aren't unique
- **BCNF?**
 - Yes. The only FD uses the primary key on the LHS

▼ fine

Columns: (id, loanid, fineDate, fineAmount, isPaid, paymentDate)

- **Primary Key:** id
- **FD:** id → (loanid, fineDate, fineAmount, isPaid, paymentDate)
- **BCNF?**
 - Yes. No FD violates the key rule

▼ personnel

Columns: (id, memberid, hireDate, salary, jobTitle, phone)

- **Primary Key:** `id`
- `memberid` also has a `UNIQUE` constraint, so there are effectively two keys:
- **FDs:**
 - `id → (memberid, hireDate, salary, jobTitle, phone)`
 - `memberid → (id, hireDate, salary, jobTitle, phone)`
- **BCNF?**
 - Yes. LHS of every FD is a key

▼ `room`

Columns: `(id, name, capacity)`

- **Primary Key:** `id`
- **FD:** `id → (name, capacity)`
 - `name` is not unique in code
- **BCNF?**
 - Yes, no FD issues

▼ `eventType`

Columns: `(id, name)`

- **Primary Key:** `id`
- There's also a `UNIQUE` constraint on `name`, so `name` is an alternate key:
- **FDs:**
 - `id → name`
 - `name → id`
- **BCNF?**
 - Yes, both sides are candidate keys

▼ `event`

Columns: (id, name, eventDate, description, targetAudience, eventTypeid, roomid, hostedByPersonnelID)

- **Primary Key:** id
- No other FD is implied because two different events can share the same name on different days, etc.
- **BCNF?**
 - Well, it can't violate BCNF so yes

▼ eventRegistration

Columns: (eventid, memberid, registrationDate)

- **Primary Key:** (eventid, memberid)
- **FD:** (eventid, memberid) → registrationDate
- **BCNF?**
 - Yes. The composite PK is the only LHS
- We have shown each table is in BCNF—or at least trivially satisfies 3NF with NO harmful dependencies—so, we don't have bad FDs that would cause anomalies
- Analysis of anomalies:
 - Update Anomalies are avoided, since each table's attributes are fully dependent on a primary or candidate key
 - Insertion Anomalies are avoided by splitting many-to-many relations into separate bridge tables (item_author , item_category , etc.)
 - Deletion Anomalies are minimized because no table is storing unrelated attributes in the same row

▼ Step (5)+: SQL Schema Explained

▼ Library Items

- ~~Library has print books, online books, magazines, scientific journals, CDs, records, etc.~~

- Library also keeps records of items (books, etc.) that might be added to library in the future.

```
CREATE TABLE libraryItem (
  id INTEGER,
  title TEXT NOT NULL,
  publicationDate DATE NOT NULL CHECK(date(publicationDate) is not null),
  itemType TEXT NOT NULL, -- e.g. 'book', 'magazine', 'cd', etc.
  status TEXT NOT NULL DEFAULT 'available' CHECK(status IN ('available', 'unavailable')),
  acquisitionStatus TEXT NOT NULL DEFAULT 'owned' CHECK(acquisitionStatus IN ('owned', 'on_order')),
  PRIMARY KEY(id)
);
```

- Allows the library to have unique items of choice to be stored with the `itemType` attribute
- Tracks if an item is available or unavailable with `status` attribute
- Library items that might be added in the future can be tracked with the `acquisitionStatus` field

▼ Authors and categories of library items

- A library item (i.e. book) can have multiple authors, and an author can write multiple books—the same applies with library items and categories—
- These many-to-many relationships are captured by tables: `author`, `category`, `item_author`, `item_category`, they allow for the association of multiple authors and or categories with any item:

```
CREATE TABLE author(
  id INTEGER,
  firstName TEXT NOT NULL,
  lastName TEXT NOT NULL,
  PRIMARY KEY(id)
);
```

```
CREATE TABLE category(
```

```

    id INTEGER,
    categoryName TEXT NOT NULL, -- fiction, music, action, romance, i
    PRIMARY KEY(id)
);

CREATE TABLE item_author(
    itemid INTEGER NOT NULL,
    authorid INTEGER NOT NULL,
    PRIMARY KEY(itemid, authorid),
    FOREIGN KEY(itemid) REFERENCES libraryItem(id),
    FOREIGN KEY(authorid) REFERENCES author(id)
);

CREATE TABLE item_category(
    itemid INTEGER NOT NULL,
    categoryid INTEGER NOT NULL,
    PRIMARY KEY(itemid, categoryid),
    FOREIGN KEY(itemid) REFERENCES libraryItem(id),
    FOREIGN KEY(categoryid) REFERENCES category(id)
);

```

▼ Borrowing and returning library items

- ~~People can borrow the items from library and return by the due date.~~

```

CREATE TABLE member (
    id INTEGER,
    firstName TEXT NOT NULL,
    lastName TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    joinDate DATE DEFAULT CURRENT_DATE CHECK(joinDate = date(joinDa
    PRIMARY KEY(id)
);

CREATE TABLE loan(

```

```

id INTEGER PRIMARY KEY AUTOINCREMENT,
itemid INTEGER NOT NULL,
memberid INTEGER NOT NULL,
loanDate DATE NOT NULL DEFAULT CURRENT_DATE CHECK(loanDate =
dueDate DATE NOT NULL DEFAULT (date('now', '+14 days')),
returnedDate DATE,
FOREIGN KEY(itemid) REFERENCES libraryItem(id),
FOREIGN KEY(memberid) REFERENCES member(id)
CHECK(dueDate >= loanDate),
CHECK(returnedDate IS NULL OR returnedDate >= loanDate)
);

```

- The `member` table stores the personal information of library members
- The `loan` table tracks borrowed library items, due dates, and returns
 - the due date is defaulted to 14 days from the loan date

```

CREATE TRIGGER set_item_unavailable
AFTER INSERT ON loan
FOR EACH ROW
WHEN NEW.returnedDate is NULL
BEGIN
    UPDATE libraryItem SET status = 'unavailable' where id=NEW.itemid;
END;

CREATE TRIGGER set_item_available
AFTER UPDATE ON loan
FOR EACH ROW
WHEN NEW.returnedDate is not NULL and old.returnedDate is NULL
BEGIN
    UPDATE libraryItem SET status = 'available' where id=new.itemid;

END;

```

- Trigger `set_item_unavailable` sets items as unavailable when loaned
- Trigger `set_item_available` sets items as available when returned

```
-- Trigger that checks for a maximum loan constraint a person can have, w
CREATE TRIGGER max_active_loans_check
BEFORE INSERT ON loan
FOR EACH ROW
BEGIN
    SELECT CASE
        WHEN (SELECT COUNT(*) FROM loan
              WHERE memberid = NEW.memberid AND returnedDate IS NULL) :
        THEN RAISE(ABORT, 'Member has reached maximum number of activ
    END;
END;
```

```
CREATE TRIGGER prevent_loan_with_unpaid_fines
BEFORE INSERT ON loan
FOR EACH ROW
BEGIN
    SELECT CASE
        WHEN EXISTS (
            SELECT 1 FROM fine f
            JOIN loan l ON f.loanid = l.id
            WHERE l.memberid = NEW.memberid AND f.isPaid = FALSE
        )
        THEN RAISE(ABORT, 'Cannot issue loan to member with unpaid fines
    END;
END;
```

```
CREATE TRIGGER prevent_loan_unavailable_item
BEFORE INSERT ON loan
FOR EACH ROW
```

```

BEGIN
  SELECT CASE
    WHEN EXISTS (
      SELECT 1 FROM libraryItem WHERE id = NEW.itemid AND status = '
    )
    THEN RAISE(ABORT, 'This item is currently unavailable')
  END;
END;

```

- trigger `max_active_loans_check` prevents members from borrowing more than 2 items
- trigger `prevent_loan_with_unpaid_fines` blocks new loans if the member has unpaid fines
- trigger `prevent_loan_unavailable_item` prevents borrowing of already loaned items

▼ Fines on overdue library items

- ~~People may be subject to fines if they do not return items by the due date.~~

```

CREATE TABLE fine(
  id INTEGER,
  loanid INTEGER NOT NULL,
  fineDate DATE,
  fineAmount DECIMAL(18,2),
  isPaid BOOLEAN DEFAULT FALSE,
  paymentDate DATE,
  PRIMARY KEY(id),
  FOREIGN KEY(loanid) REFERENCES loan(id)
  CHECK(paymentDate IS NULL OR (isPaid = TRUE AND paymentDate >=
);

CREATE TRIGGER generate_fine_on_late_return

```

```

AFTER UPDATE ON loan
FOR EACH ROW
WHEN NEW.returnedDate IS NOT NULL AND NEW.returnedDate > NEW.dueDate
BEGIN
    INSERT INTO fine (loanid, fineDate, fineAmount, isPaid)
    VALUES (NEW.id, NEW.returnedDate,
            (julianday(NEW.returnedDate) - julianday(NEW.dueDate)) * 0.50, --
            FALSE);
END;

```

- `fine` table records the monetary penalties for items returned after their due date
- trigger `generate_fine_on_late_return` applies a fine of \$0.50 per late day when an overdue item is returned

▼ Library personnel

- ~~Library also has personnel and record keeping for personnel.~~

```

CREATE TABLE personnel (
    id INTEGER,
    memberid INTEGER UNIQUE NOT NULL,
    hireDate DATE DEFAULT CURRENT_DATE,
    salary INTEGER DEFAULT 0,
    jobTitle TEXT NOT NULL CHECK(jobTitle IN ('librarian', 'volunteer', 'admin'),
    phone TEXT,
    PRIMARY KEY(id),
    FOREIGN KEY(memberid) REFERENCES member(id)
);

```

- `personnel` table stores information on library employees and volunteers and links members to personnel (all employees and volunteers are library members too)

▼ Library event system

- ~~Library also holds book clubs, book related events, art shows, film screenings, etc.~~
- ~~Library events are recommended for specific audiences.~~
- ~~Library events are held on library social rooms.~~
- ~~People can attend library events for free.~~

```
CREATE TABLE room (
  id INTEGER PRIMARY KEY,
  name TEXT NOT NULL,      -- e.g. "Main Hall", "Room A", "Community R
  capacity INTEGER NOT NULL
);
```

```
CREATE TABLE eventType (
  id INTEGER PRIMARY KEY,
  name TEXT NOT NULL UNIQUE -- e.g. 'Book Club', 'Art Show', etc.
);
```

```
CREATE TABLE event (
  id INTEGER PRIMARY KEY,
  name TEXT NOT NULL,
  eventDate DATE NOT NULL CHECK(eventDate = date(eventDate)),
  description TEXT,
  targetAudience TEXT, -- or FK to category
  eventTypeid INTEGER NOT NULL,
  roomid INTEGER NOT NULL,
  hostedByPersonnelID INTEGER, -- the personnel responsible for hosting
  FOREIGN KEY(eventTypeid) REFERENCES eventType(id),
  FOREIGN KEY(roomid) REFERENCES room(id),
  FOREIGN KEY(hostedByPersonnelid) REFERENCES personnel(id)
  CHECK(eventDate >= CURRENT_DATE)
);
```

```
CREATE TABLE eventRegistration (
  eventid INTEGER NOT NULL,
```

```
memberid INTEGER NOT NULL,  
registrationDate DATE NOT NULL DEFAULT CURRENT_DATE,  
PRIMARY KEY(eventid, memberid),  
FOREIGN KEY(eventid) REFERENCES event(id),  
FOREIGN KEY(memberid) REFERENCES member(id)  
);
```

- Tables:
 - `eventType` classifies the type of event, and contains a list of valid values, it's used as a foreign key to label events in the `event` table. Instead of leaving `eventTypeid` as a text attribute, which could introduce inconsistencies
 - `room` table represents library rooms where events are hosted, and `roomid` in `event` table uses it to assign events to rooms
 - `event` stores all event details. `targetAudience` is meant to be used in conjunction with `category`. There is also a constraint ensure event dates are not in the past
 - `eventRegistration` records which members have registered for which events (many-to-many relationship)