**Updated Documentation for Library Management System Database Schema**

**1. Introduction**

This document details the Library Management System (LMS) database schema, which is normalized to Third Normal Form (3NF). The design prioritizes data integrity and consistency by incorporating strong referential integrity constraints. Explicit update and delete actions on foreign keys ensure that changes in primary key values propagate correctly, thereby maintaining the relationships between entities.

**2. Schema Overview**

The LMS schema consists of the following primary entities:

* **Authors**: Stores author details.
* **Publishers**: Contains publisher information.
* **Categories**: Defines the different book categories or genres.
* **Books**: Records book details along with their publisher relationships.
* **BookAuthors**: Manages the many-to-many relationship between Books and Authors.
* **BookCategories**: Manages the many-to-many relationship between Books and Categories.
* **Members**: Contains details about library members.
* **Users**: Contains librarian and staff information.
* **Loans**: Tracks loan transactions for books.
* **Fines**: (Optional) Tracks fines related to overdue books.

**3. Referential Integrity and Update Actions**

**3.1 Books and Publishers**

* **Constraint**: fk\_books\_publisher
* **ON DELETE SET NULL**: If a publisher is deleted, the PublisherID field in the Books table is set to NULL—this preserves the book record even though its publisher is no longer in the system.
* **ON UPDATE CASCADE**: Should the PublisherID in the Publishers table change, the new key is automatically updated in the Books table, ensuring consistent references.

**3.2 BookAuthors Table**

* **Constraints**: fk\_bookauthors\_book and fk\_bookauthors\_author
* **ON DELETE CASCADE**: If a Book or an Author is deleted, the corresponding association in BookAuthors is automatically removed.
* **ON UPDATE CASCADE**: Changes to primary keys in Books or Authors automatically propagate to BookAuthors to maintain correct relationships.

**3.3 BookCategories Table**

* **Constraints**: fk\_bookcategories\_book and fk\_bookcategories\_category
* **ON DELETE CASCADE**: Association records are removed if a Book or Category is deleted.
* **ON UPDATE CASCADE**: Any update to the primary keys in Books or Categories is cascaded to keep the link table in sync.

**3.4 Loans Table**

* **Constraints**: fk\_loans\_book, fk\_loans\_member, fk\_loans\_issuedby, and fk\_loans\_receivedby
* **ON DELETE RESTRICT**: Protects the integrity of loan records by preventing deletion of Books or Members that are referenced in an active or historical loan record.
* **ON UPDATE CASCADE**: Any change to the primary keys (e.g., BookID or MemberID) in the referenced tables is automatically cascaded to the Loans table.
* **ON DELETE SET NULL** (for IssuedByUserID and ReceivedByUserID): If a User (librarian) is removed, the respective field in a loan record is set to NULL to preserve the transaction without pointing to a nonexistent user.
* **Checks**: A check constraint ensures that if a ReturnDate exists, it is not before the LoanDate.

**3.5 Fines Table**

* **Constraints**: fk\_fines\_loan and fk\_fines\_member
* **ON DELETE CASCADE**: The fine record is automatically removed if its associated loan is deleted.
* **ON UPDATE CASCADE**: Any changes to the LoanID or MemberID in the referenced tables are automatically reflected in the Fines table.
* **ON DELETE RESTRICT** (for MemberID): Prevents the deletion of a Member record if there is an associated fine, thus preserving important financial history.

**4. Update Actions – A Closer Look**

**Cascading Updates (ON UPDATE CASCADE)**

* **Purpose**: This setting ensures that if a primary key value in any parent table (e.g., Publisher, Book, Author) changes, the corresponding foreign keys in all dependent tables are automatically updated. This removes the risk of orphaned records and maintains data consistency across the database.
* **Implementation**: Almost every foreign key in the schema includes the ON UPDATE CASCADE clause, meaning that changes are automatically propagated to associated records.

**Setting Fields to NULL (ON UPDATE/DELETE SET NULL)**

* **Purpose**: In situations where an associated record (such as a publisher or user) is removed or updated, and it is not feasible or logical to cascade changes, the foreign key fields are set to NULL. This retains the record (for historical purposes) without forcing an incorrect association.
* **Implementation**: For example, the Books table’s PublisherID and the Loans table’s user fields (IssuedByUserID and ReceivedByUserID) use ON DELETE SET NULL combined with ON UPDATE CASCADE to ensure that updates are properly handled while preventing deletions from causing data integrity issues.

**5. Data Type Considerations**

* The PublicationYear in the Books table has been changed from the MySQL YEAR datatype to INT to bypass the range limitation (1901 to 2155) inherent in the YEAR datatype.

**6. Security Considerations**

* Passwords for both Members and Users are stored using hashed values along with unique salts. This approach helps protect sensitive data in the event of a breach.

**7. Conclusion**

By incorporating explicit update actions (ON UPDATE CASCADE) and delete actions (ON DELETE SET NULL or RESTRICT) within the foreign key constraints, this schema robustly enforces referential integrity. Any update to key fields in parent tables automatically cascades to related records, preventing orphaned data and ensuring overall consistency. The design is ready for use in a production system where maintaining data integrity and facilitating seamless updates are critical.

**Comprehensive Guide to Setting Up Python Environment and Generating Synthetic Data Using Faker**

**1. Setting Up the Project Directory:**

* **Navigate to the Documents Folder:**

cd ~/Documents

* **Create a New Directory for the Project:**

mkdir fekercsv

* **Move into the Project Directory:**

cd fekercsv

**2. Creating and Activating a Virtual Environment:**

* **Create a Virtual Environment Named faker\_env:**

python3 -m venv faker\_env

* **Activate the Virtual Environment:**

source faker\_env/bin/activate

*Note: After activation, the terminal prompt will display (faker\_env), indicating that the virtual environment is active.*

**3. Installing Necessary Libraries:**

* **Upgrade pip to the Latest Version:**

pip install --upgrade pip

* **Install the Faker Library:**

pip install faker

* **Install Pandas for Data Manipulation:**

pip install pandas

* **Install tqdm for Progress Bars (Optional):**

pip install tqdm

*Note: tqdm provides a visual indication of the progress of loops and is useful for tracking the data generation process.*

**4. Generating Synthetic Data Using Faker:**

* **Create a Python Script Named generate\_csv.py:**

touch generate\_csv.py

## Open generate\_csv.py with a Text Editor and Add the Following Code:

from faker import Faker

import pandas as pd

from datetime import datetime, timedelta

import random

fake = Faker()

Faker.seed(0)

random.seed(0)

# Set up a 5-year future range for expiry dates

start\_date = datetime.now()

end\_date = start\_date + timedelta(days=365 \* 5)

# 1. Authors – no change needed

def generate\_authors(num\_authors):

authors = []

for \_ in range(num\_authors):

authors.append({

'FirstName': fake.first\_name(),

'LastName': fake.last\_name(),

'Biography': fake.text(max\_nb\_chars=200).replace('\n', ' ')

})

return pd.DataFrame(authors)

# 2. Publishers – ensure uniqueness by appending a counter

def generate\_publishers(num\_publishers):

publishers = []

for i in range(num\_publishers):

publishers.append({

'Name': f"{fake.company()} {i}", # append counter to guarantee uniqueness

'Address': fake.address().replace('\n', ', '),

'ContactInfo': fake.phone\_number()

})

return pd.DataFrame(publishers)

# 3. Categories – likewise, append a counter to guarantee uniqueness

def generate\_categories(num\_categories):

categories = []

for i in range(num\_categories):

categories.append({

'Name': f"{fake.word()}\_{i}",

'Description': fake.text(max\_nb\_chars=100)

})

return pd.DataFrame(categories)

# 4. Members – use uuid4 for card numbers; note that booleans sometimes should be converted to 1/0

def generate\_members(num\_members):

members = []

for i in range(num\_members):

# Get a phone number and truncate to 20 characters

phone = fake.phone\_number()[:20]

members.append({

'LibraryCardNumber': fake.unique.uuid4(),

'FirstName': fake.first\_name(),

'LastName': fake.last\_name(),

'Address': fake.address().replace('\n', ', '),

'Email': fake.unique.email(),

'PhoneNumber': phone,

'RegistrationDate': fake.date\_time\_this\_decade(),

'ExpiryDate': fake.date\_time\_between(start\_date=datetime.now(), end\_date=datetime.now() + timedelta(days=365\*5)),

'IsActive': int(fake.boolean()),

'PasswordHash': fake.sha256(),

'PasswordSalt': fake.sha1()

})

return pd.DataFrame(members)

# 5. Users – same note on uniqueness and booleans

def generate\_users(num\_users):

users = []

roles = ['Admin', 'Librarian', 'Assistant']

for \_ in range(num\_users):

users.append({

'Username': fake.unique.user\_name(),

'PasswordHash': fake.sha256(),

'PasswordSalt': fake.sha1(),

'FirstName': fake.first\_name(),

'LastName': fake.last\_name(),

'Email': fake.unique.email(),

'Role': random.choice(roles),

'IsActive': int(fake.boolean()), # ensure boolean as integer if needed

'LastLogin': fake.date\_time\_this\_year()

})

return pd.DataFrame(users)

# 6. Books – ensure unique ISBNs, cast PublicationYear to int, and ensure logical consistency for copies

def generate\_books(num\_books, publishers\_df):

books = []

publisher\_ids = publishers\_df.index.tolist()

for \_ in range(num\_books):

publisher\_id = random.choice(publisher\_ids) if publisher\_ids else None

total\_copies = random.randint(1, 10)

available\_copies = random.randint(0, total\_copies)

books.append({

'ISBN': fake.unique.isbn13(), # ensuring uniqueness of ISBN

'Title': fake.sentence(nb\_words=4),

'PublisherID': publisher\_id,

'PublicationYear': int(fake.year()), # cast to int for MySQL compatibility

'TotalCopies': total\_copies,

'AvailableCopies': available\_copies,

'ShelfLocation': fake.bothify(text='??-###')

})

return pd.DataFrame(books)

# -------------------------------------------------------------------------

# NEW: Functions to generate many-to-many relationship tables

# -------------------------------------------------------------------------

def generate\_bookauthors(books\_df, authors\_df, min\_authors\_per\_book=1, max\_authors\_per\_book=3):

"""

For each book, randomly assign between min\_authors\_per\_book and max\_authors\_per\_book authors.

The function assumes that auto-incremented IDs will start at 1 and that the row index corresponds

to the final ID in the database.

"""

relationships = []

for book\_index in range(len(books\_df)):

num\_authors\_for\_book = random.randint(min\_authors\_per\_book, max\_authors\_per\_book)

sampled\_authors = authors\_df.sample(n=num\_authors\_for\_book, replace=False)

for author\_index in sampled\_authors.index:

relationships.append({

'BookID': book\_index + 1, # Adjusting for 1-based indexing in DB

'AuthorID': author\_index + 1

})

return pd.DataFrame(relationships)

def generate\_bookcategories(books\_df, categories\_df, min\_categories\_per\_book=1, max\_categories\_per\_book=2):

"""

For each book, randomly assign between min\_categories\_per\_book and max\_categories\_per\_book categories.

"""

relationships = []

for book\_index in range(len(books\_df)):

num\_categories\_for\_book = random.randint(min\_categories\_per\_book, max\_categories\_per\_book)

sampled\_categories = categories\_df.sample(n=num\_categories\_for\_book, replace=False)

for cat\_index in sampled\_categories.index:

relationships.append({

'BookID': book\_index + 1,

'CategoryID': cat\_index + 1

})

return pd.DataFrame(relationships)

# -------------------------------------------------------------------------

# Specify the number of records for each table

num\_authors = 110000

num\_publishers = 110000

num\_categories = 110000

num\_members = 110000

num\_users = 110000

num\_books = 110000

# Generate base table data

authors\_df = generate\_authors(num\_authors)

publishers\_df = generate\_publishers(num\_publishers)

categories\_df = generate\_categories(num\_categories)

members\_df = generate\_members(num\_members)

users\_df = generate\_users(num\_users)

books\_df = generate\_books(num\_books, publishers\_df)

# Save base table data to CSV with quoting (this helps manage any commas or special characters)

csv\_options = {'index': False, 'quoting': 1} # quoting=1 is equivalent to csv.QUOTE\_ALL

authors\_df.to\_csv('authors.csv', \*\*csv\_options)

publishers\_df.to\_csv('publishers.csv', \*\*csv\_options)

categories\_df.to\_csv('categories.csv', \*\*csv\_options)

members\_df.to\_csv('members.csv', \*\*csv\_options)

users\_df.to\_csv('users.csv', \*\*csv\_options)

books\_df.to\_csv('books.csv', \*\*csv\_options)

# -------------------------------------------------------------------------

# Generate and save relationship data for BookAuthors and BookCategories

bookauthors\_df = generate\_bookauthors(books\_df, authors\_df)

bookauthors\_df.to\_csv('bookauthors.csv', index=False, quoting=1)

bookcategories\_df = generate\_bookcategories(books\_df, categories\_df)

bookcategories\_df.to\_csv('bookcategories.csv', index=False, quoting=1)

For faster import of csv I used the LOAD DATA LOCAL INFILE

LOAD DATA LOCAL INFILE 'C:/Users/YourUsername/Downloads/yourfile.csv'

INTO TABLE your\_table\_name

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

To check the

SHOW VARIABLES LIKE 'local\_infile'; -- this should be on

To turn it on

SET GLOBAL local\_infile = 1;

Next edit the

A screenshot of a computer

AI-generated content may be incorrect.



Click advanced and add this to others OPT\_LOCAL\_INFILE=1

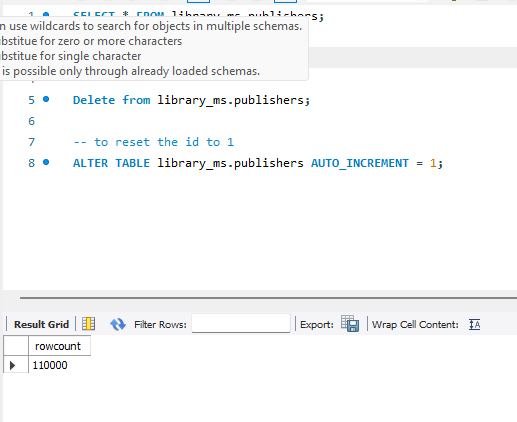
A screenshot of a computer

AI-generated content may be incorrect.



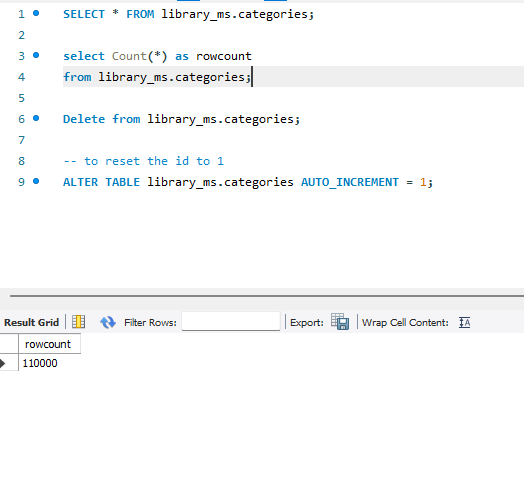
A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.