Advanced SQLAlchemy Exercise for Students

In this exercise, students will apply their knowledge of **SQLAlchemy** to model a more complex database with advanced relationships, queries, and features. The goal is to build a **Library Management System** that includes books, authors, members, and loans. Students are expected to implement **CRUD operations**, **relationships**, **transactions**, and **custom queries**.

Scenario: Library Management System

You are tasked with building a Library Management System using SQLAlchemy. In this system:

- **Books** can have multiple authors (many-to-many).
- **Members** can borrow multiple books, and each book can be borrowed by multiple members (many-to-many).
- You need to track **loans**, which record when a book was borrowed and when it is due.
- Implement a **one-to-one** relationship between **members** and **membership cards**.

Step 1: Model the Database

1. Books Table:

- `id`: Integer, primary key
- `title`: String
- `isbn`: String (unique identifier)

2. Authors Table:

- `id`: Integer, primary key
- `name`: String

3. Members Table:

- `id`: Integer, primary key
- `name`: String
- `membership_card_id`: Integer, Foreign key (one-to-one relationship with membership cards)

4. MembershipCards Table:

- `id`: Integer, primary key
- `card_number`: String (unique identifier)

5. Loans Table:

- `id`: Integer, primary key
- `book_id`: Integer, Foreign key (many-to-many relationship with books)
- `member_id`: Integer, Foreign key (many-to-many relationship with members)
- `borrowed_date`: DateTime
- `due_date`: DateTime

Step 2: Relationships

1. **Many-to-Many** between **Books** and **Authors**.
2. **Many-to-Many** between **Books** and **Members** (via Loans).
3. **One-to-One** between **Members** and **Membership Cards**.

Step 3: Requirements for the Exercise

1. Define Models and Relationships:

- Create SQLAlchemy models for **Books**, **Authors**, **Members**, **Loans**, and **MembershipCards**.
- Use SQLAlchemy's relationship and association table for many-to-many relationships.

2. CRUD Operations:

- Implement the following operations:
  - Add a new **book** and associate it with one or more authors.
  - Add a new **member** with a membership card.
  - Record a **loan** when a member borrows a book, including the borrowed and due dates.
  - Implement a query to list all books borrowed by a specific member.

3. Custom Queries:

- Query all members who have borrowed more than 3 books.
- Find all books that have **not been borrowed** in the last 30 days.

4. Transaction Management:

- Ensure that when a **loan** is created, all changes are committed atomically.
- If the book is already on loan and a duplicate loan is attempted, roll back the transaction.

Step 4: Bonus Challenges

- Add validation to ensure that the same member cannot borrow the same book twice before returning it.
- Implement a query that finds all **overdue loans** (loans where the due date has passed).
- Add a trigger to automatically set the **due date** to 14 days after the borrowed date when a loan is created.

Deliverables

1. **SQLAlchemy Models**: Create models for the required entities and define their relationships.
2. **CRUD Operations**: Implement functions to add, retrieve, update, and delete records from the database.
3. **Custom Queries**: Write SQLAlchemy queries to satisfy the custom query requirements.
4. **Transaction Handling**: Ensure all database operations are wrapped in proper transactions.

Example Starter Code

Here's some starter code to help you define the models:

```
python
Copy code
from sqlalchemy import create_engine, Column, Integer, String, DateTime,
ForeignKey, Table
from sqlalchemy.orm import relationship, sessionmaker
from sqlalchemy.ext.declarative import declarative_base
from datetime import datetime, timedelta

Base = declarative_base()

# Many-to-Many relationship between Books and Authors
book_author_association = Table(
    'book_author', Base.metadata,
    Column('book_id', Integer, ForeignKey('books.id')),
    Column('author_id', Integer, ForeignKey('authors.id'))
)

# Many-to-Many relationship between Books and Members (via Loans)
class Loan(Base):
    __tablename__ = 'loans'

    id = Column(Integer, primary_key=True)
    book_id = Column(Integer, ForeignKey('books.id'))
    member_id = Column(Integer, ForeignKey('members.id'))
    borrowed_date = Column(DateTime, default=datetime.now)
    due_date = Column(DateTime, default=datetime.now() +
timedelta(days=14))

class Book(Base):
    __tablename__ = 'books'

    id = Column(Integer, primary_key=True)
    title = Column(String)
    isbn = Column(String, unique=True)

    authors = relationship('Author', secondary=book_author_association,
back_populates='books')
    loans = relationship('Loan', back_populates='book')

class Author(Base):
    __tablename__ = 'authors'

    id = Column(Integer, primary_key=True)
    name = Column(String)

    books = relationship('Book', secondary=book_author_association,
back_populates='authors')
```

```
class Member(Base):
    __tablename__ = 'members'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    membership_card_id = Column(Integer, ForeignKey('membership_cards.id'))

    membership_card = relationship('MembershipCard',
back_populates='member')
    loans = relationship('Loan', back_populates='member')

class MembershipCard(Base):
    __tablename__ = 'membership_cards'

    id = Column(Integer, primary_key=True)
    card_number = Column(String, unique=True)

    member = relationship('Member', back_populates='membership_card')
```

Step 5: Testing

- **Run tests**: Create unit tests to verify that each of the CRUD operations and relationships is functioning correctly.
- **Write queries**: Test the custom queries, ensuring that they return the expected results.

What Students Will Learn

1. How to model advanced relationships (many-to-many, one-to-one) in SQLAlchemy.
2. How to perform CRUD operations in a relational database.
3. How to handle transactions and rollback scenarios.
4. Writing custom queries using SQLAlchemy’s ORM features.