Практическое занятие 5. Развитие модели данных. Связь «многие-ко-многим» с атрибутами

I. Связь с атрибутами

Связующая таблица в случае связи многие-ко-многим не просто соединяет две сущности, но и **может хранить атрибуты, относящиеся к самой связи**.

- **Если связь не имеет атрибутов**, то таблица просто содержит **два внешних ключа**.
- **Если связь имеет атрибуты**, то эти атрибуты добавляются в связующую таблицу.

Между сущностями **Читатель** и **Экземпляр книги** возникает связь, которая описывает аренду книги. Эта связь может содержать дополнительные атрибуты, такие как **дата взятия** и **дата возврата**.

Новая таблица **Reader** для хранения сведений о читателях, которые берут книги в библиотеке. Атрибуты: ReaderID, Name, Email и Status. Статус определяет активность читателя: 0 – неактивный, например, давно не посещал библиотеку; `1` – активный, может брать книги; `2` – заблокирован, есть задолженность, нарушение правил.

Связующая таблица: **Rental (Аренда)** предназначена для связи между Читателем и Экземпляром книги, которая также хранит атрибуты, описывающие саму аренду. Атрибут ReaderID – внешний ключ на таблицу Reader, атрибут Instance – внешний ключ на таблицу BookInstance, DateRented – дата взятия книги, DateReturned – дата возврата книги. Атрибут RedtalID – первичный ключ записей в таблицеRental.

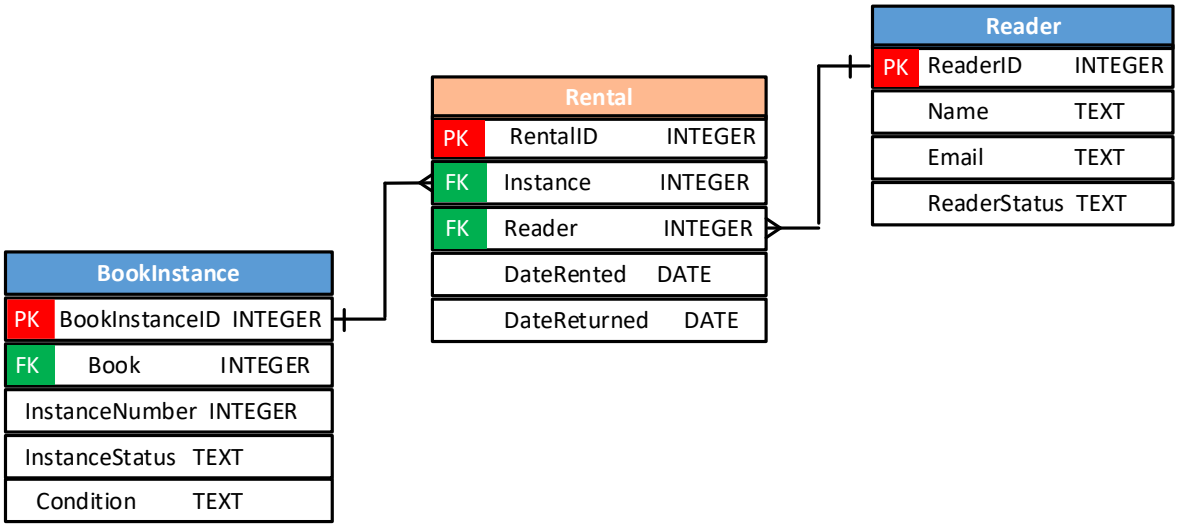

Рисунок 5 – Добавление связующей таблицы с дополнительными атрибутами

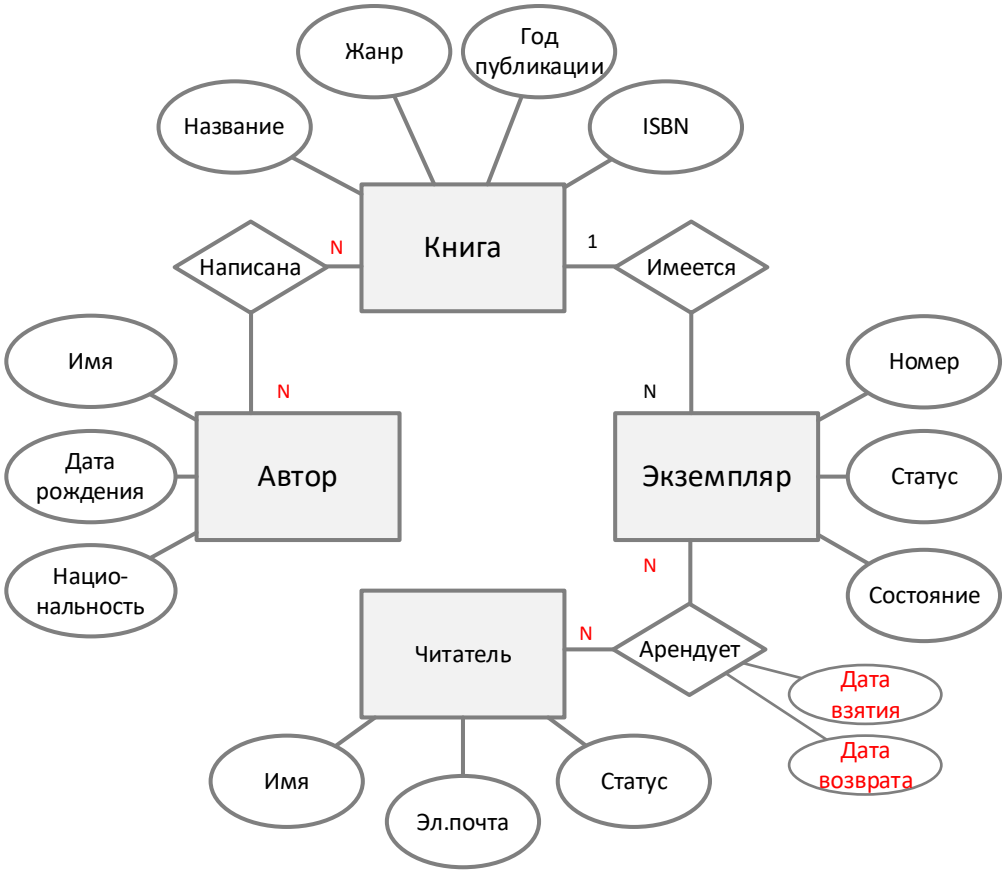

Рисунок 6 – Итоговая концептуальная модель

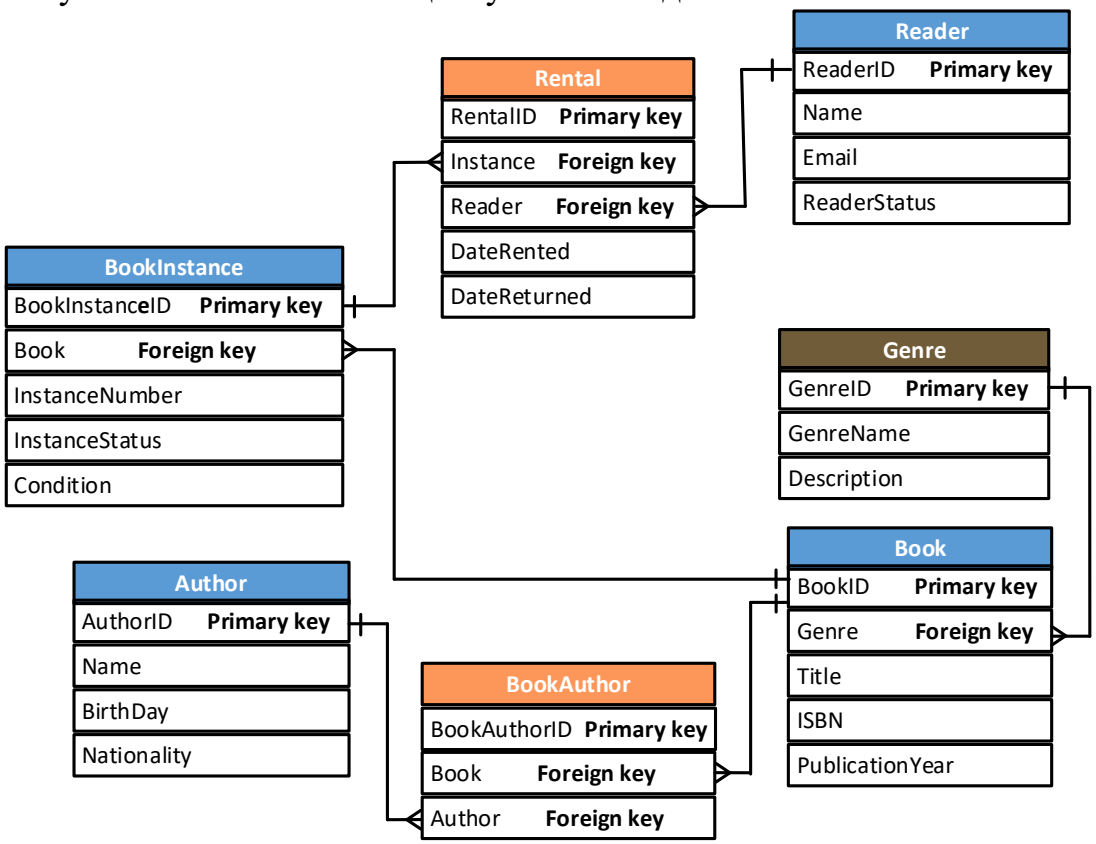

Рисунок 7 – Итоговая логическая модель

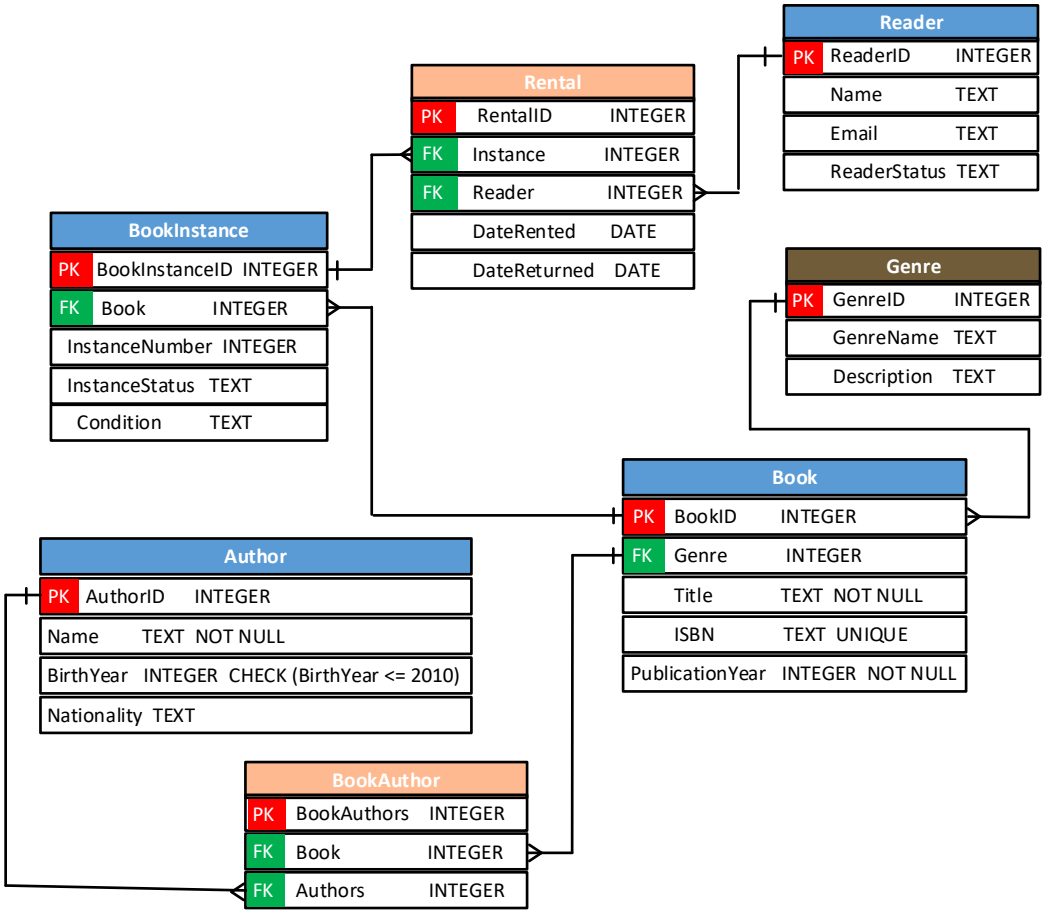

Рисунок 8 – Итоговая физическая модель

В СУБД **SQLite** нет отдельного типа данных `DATE` или `DATETIME`, потому что он хранит все данные в формате TEXT, INTEGER или REAL**.** Но это не проблема. Есть несколько способов хранить даты, в зависимости от удобства работы. Будем использовать хранение даты в виде текста

```
'YYYY-MM-DD'          → '2025-03-08'
'YYYY-MM-DD HH:MM'    → '2025-03-08 14:30'
'YYYY-MM-DD HH:MM:SS' → '2025-03-08 14:30:15'
```

Такое представление читаемо для людей; поддерживает стандартный SQL-фильтр: WHERE DateTaken BETWEEN '2025-03-01' AND '2025-03-10' и позволяет использовать встроенные функции SQLite: strftime(), date(), datetime(). Это уже готовые функции для работы с датами, встроенные в сам движок базы данных. Они позволяют легко преобразовывать даты, фильтровать записи и вычислять интервалы.

II. *Транзакции*

Транзакция – это единица работы с базой данных, которая выполняет несколько операций (например, вставку, обновление или удаление данных). Все эти операции должны быть выполнены как одно целое, что означает, что

либо все операции завершаются успешно, либо, в случае ошибки, все изменения отменяются.

В реальных приложениях транзакции часто используются для обеспечения целостности и консистентности данных, особенно в таких сценариях, как аренда и возврат книг.

Рассмотрим, как организовать транзакции на практике, чтобы гарантировать корректность работы с базой данных:

1. **Начало транзакции**: Транзакция начинается с SQL-команды `BEGIN TRANSACTION`. Это сообщает системе, что все операции, которые будут выполняться после этой команды, должны быть рассмотрены как часть одной транзакции. Например, при аренде книги мы сначала начинаем транзакцию, чтобы все действия (например, обновление статуса читателя и запись аренды) выполнялись атомарно.
2. **Проверка условий**: Чтобы избежать ошибок и неконсистентности данных, важно проверить, что условия для выполнения транзакции выполнены. Например, перед тем как арендовать книгу, мы должны убедиться, что:
    - Книга существует в базе данных.
    - Есть доступные экземпляры книги.
    - Читатель активен и может арендовать книгу.

    Если хотя бы одно из этих условий не выполняется, транзакция не должна продолжаться, и все изменения должны быть откатаны.

    Пример:

```
# Проверка наличия книги
cursor.execute("SELECT BookID FROM Book WHERE Title = ?", (book_title,))
book = cursor.fetchone()
if not book:
    conn.rollback()  # Откатываем транзакцию, если книга не найдена
    print("Книга не найдена.")
```

3. **Запись в базу данных**: Если все условия проверки выполнены, можно выполнять изменения, например, добавлять запись о том, что книга была арендована. Эти изменения происходят внутри транзакции и не будут видны другим пользователям базы данных до завершения транзакции.
4. **Завершение транзакции**: Когда все операции внутри транзакции успешно выполнены, нужно завершить транзакцию с помощью команды `COMMIT`. Это сохранит все изменения в базе данных. В противном случае, если произошла ошибка на любом этапе, используется команда `ROLLBACK`, чтобы откатить все изменения и вернуть базу данных в исходное состояние.

    Пример:

```
try:
    # Выполнение операций
    cursor.execute("UPDATE Rental SET DateReturned = ? WHERE RentalID = ?",
(current_date, rental_id))
```

```
    conn.commit()  # Завершаем транзакцию, сохраняем изменения
except Exception as e:
    print(f"Ошибка: {e}")
    conn.rollback()  # Откатываем транзакцию в случае ошибки
```

5. **Обработка ошибок**: Если в процессе транзакции возникает ошибка (например, проблемы с сетью или некорректные данные), важно откатить все изменения, чтобы база данных не оказалась в неконсистентном состоянии. Ошибка может быть поймана в блоке `except`, где будет вызван `ROLLBACK`, чтобы отменить все изменения, сделанные в рамках текущей транзакции.
6. **Использование транзакций для логических операций**: Важно помнить, что транзакции используются не только для изменения данных, но и для выполнения логических операций, где несколько шагов должны быть выполнены в строгом порядке. Например, при аренде книги сначала обновляется статус читателя, затем создается запись об аренде, а в случае возврата — фиксируется дата возврата и статус книги. Все эти операции могут зависеть друг от друга, и их нужно выполнять внутри одной транзакции.

Практическое оформление

- Транзакции начинаются с команды `BEGIN TRANSACTION`.
- Все условия, необходимые для успешного выполнения транзакции, проверяются на начальном этапе.
- Если всё в порядке, изменения выполняются и транзакция завершается командой `COMMIT`.
- В случае возникновения ошибок или несоответствий в данных выполняется откат изменений с помощью `ROLLBACK`.

Таким образом, транзакции обеспечивают выполнение операций в целостности и защищают базу данных от неконсистентных изменений.

III. *Использование ID вместо фамилии*

Использование идентификаторов (ID) вместо фамилий или других строковых значений – это стандартная практика в реальных базах данных.

Уникальность**:** Фамилия может быть неуникальной, и два человека могут иметь одинаковую фамилию, что создаст путаницу. ID же всегда уникален, что исключает ошибки при поиске читателя.

Производительность**:** Числовые идентификаторы работают быстрее, чем строковые значения. Это упрощает обработку данных и ускоряет работу с базой.

Гибкость: В случае изменения фамилии читателя, достаточно обновить одно поле, а не все записи, связанные с его фамилией. ID остаётся неизменным, что делает систему более стабильной.

Практика: В реальных приложениях для связи между таблицами обычно используют ID, а фамилия может использоваться только для поиска. Это упрощает поддержку системы и минимизирует вероятность ошибок.

Таким образом несмотря на то, что при вводе пользователем фамилии приложение может найти нужного читателя, в коде и запросах используется ID для точной идентификации записей в базе данных.

III. Код базовой программы:

```
import sqlite3
import os

# Путь к файлу базы данных SQLite
db_file = 'library.db'

# ***************************
# эта команда нужна для отладки (для повторных запусков)
# Если файл БД уже существует, то удалить существующий файл БД,
# чтобы предотвратить повторную запись тех же самых данных
if  os.path.isfile(db_file):
  os.remove(db_file)
# ***************************

# Проверка существования файла базы данных
if not os.path.isfile(db_file):
    # Если файл базы данных не существует, создаем его и подключаемся
    conn = sqlite3.connect(db_file)

    # Создание объекта курсора для выполнения SQL-запросов
    cursor = conn.cursor()

    # Создание таблиц и внесение данных в базу данных

    # Создание таблицы Genre
    cursor.execute('''CREATE TABLE IF NOT EXISTS Genre (
        GenreID INTEGER PRIMARY KEY,
        GenreName TEXT NOT NULL UNIQUE,
        Description TEXT
    )''')

    # Создание таблицы Author
    cursor.execute('''CREATE TABLE IF NOT EXISTS Author (
        AuthorID INTEGER PRIMARY KEY,
        Name TEXT NOT NULL,
        BirthYear INTEGER CHECK (BirthYear <= 2010),
        Nationality TEXT
    )''')
```

```
    # Создание таблицы Book
    cursor.execute('''CREATE TABLE IF NOT EXISTS Book (
        BookID INTEGER PRIMARY KEY,
        Genre INTEGER NOT NULL,
        Title TEXT NOT NULL,
        PublicationYear INTEGER NOT NULL,
        ISBN TEXT UNIQUE,
        FOREIGN KEY (Genre) REFERENCES Genre(GenreID)
    )''')

    # Создание связующей таблицы BookAuthor
    cursor.execute('''CREATE TABLE IF NOT EXISTS BookAuthor (
        BookAuthorID INTEGER PRIMARY KEY AUTOINCREMENT,  -- автоинкрементный ID
        Author INTEGER NOT NULL,                         -- внешний ключ на
таблицу Author
        Book INTEGER NOT NULL,                           -- внешний ключ на
таблицу Book
        FOREIGN KEY (Author) REFERENCES Author(AuthorID),  -- ссылка на таблицу
Author
        FOREIGN KEY (Book) REFERENCES Book(BookID)        -- ссылка на таблицу
Book
    )''')
    # Создание таблицы BookInstance
    cursor.execute('''CREATE TABLE IF NOT EXISTS BookInstance (
        BookInstanceID INTEGER PRIMARY KEY,
        Book INTEGER NOT NULL,
        VolumeNumber INTEGER NOT NULL DEFAULT 1,
        VolumeStatus TEXT,
        Condition TEXT,
        FOREIGN KEY (Book) REFERENCES Book(BookID)
    )''')
 # Создание таблицы Reader
    cursor.execute('''CREATE TABLE IF NOT EXISTS Reader (
        ReaderID INTEGER PRIMARY KEY,
        Name TEXT NOT NULL,
        Email TEXT NOT NULL,
        ReaderStatus INTEGER
    )''')
# Создание таблицы Rental
    cursor.execute('''CREATE TABLE IF NOT EXISTS Rental (
        RentalID INTEGER PRIMARY KEY,
        Reader INTEGER NOT NULL,
        Instance INTEGER NOT NULL,
        DateRented TEXT NOT NULL,    -- Дата взятия (ISO-формат)
        DateReturned TEXT,          -- Дата возврата (может быть NULL)
        FOREIGN KEY (Reader) REFERENCES Reader(ReaderID),
        FOREIGN KEY (Instance) REFERENCES BookInstance(BookInstanceID)
    )''')
```

```
    # Сохранение изменений
    conn.commit()


else:
    # Если файл базы данных существует, просто подключаемся к нему
    conn = sqlite3.connect(db_file)
    # и создаем объект курсор для выполнения SQL-запросов
    cursor = conn.cursor()


# Внесение множества данных в таблицу Author
authors_data = [
    ("М. Горький", 1868, "русский"),
    ("А. Чехов", 1860, "русский"),
    ("Т. Шевченко", 1814, "украинец"),
    ("Л. Толстой", 1828, "русский"),
    ("И. Ильф", 1897, "еврей"),
    ("Е. Петров",1903, "русский")
]
cursor.executemany("INSERT INTO Author (Name, BirthYear, Nationality) VALUES (?,
?, ?)", authors_data)

# Добавление жанров
genres_data = [
    ("роман", "Произведение прозаического жанра, длинное по объему"),
    ("драма", "Произведение, предназначенное для театрального исполнения"),
    ("повесть", "Произведение прозаического жанра, короткое по объему"),
    ("поэма", "Произведение лиро-эпической или эпической формы, чаще всего в
стихотворной форме")
]
cursor.executemany("INSERT INTO Genre (GenreName, Description) VALUES (?, ?)",
genres_data)

# Добавление книг
books_data = [
    (1, "Мать", 1996, "978-1234567890"),
    (2, "На дне", 1992, "978-0987654321"),
    (3, "Каштанка", 2017, "978-5432167890"),
    (3, "Вишневый сад", 2017, "978-6432167890"),
    (4, "Гайдамаки", 1930, "978-2345678901"),
    (1, "Война и мир", 2025, "978-3456789012"),
    (1,"Двенадцать стульев", 1828,"978-22334455"),
    (1,"Золотой теленок", 1833,"978-33445522")
]
cursor.executemany("INSERT INTO Book ( Genre, Title, PublicationYear, ISBN)
VALUES (?, ?, ?, ?)", books_data)

# Добавление экземпляров
instances_data = [
    (1, 1, "Выдан", "Хорошее"),
```

```
    (1, 2, "Доступен", "Хорошее"),
    (2, 1, "Доступен", "Хорошее"),
    (2, 2, "Доступен", "Хорошее"),
    (3, 1, "Доступен", "Отличное"),
    (3, 2, "Доступен", "Отличное"),
    (4, 1, "Доступен", "Хорошее"),
    (5, 1, "Доступен", "Удовлетворительное"),
    (6, 1, "Выдан", "Отличное"),
    (7, 1, "Доступен", "Отличное"),
    (8, 1, "Доступен", "Отличное")
]
cursor.executemany("INSERT INTO BookInstance (Book, VolumeNumber, VolumeStatus,
Condition) VALUES (?, ?, ?, ?)", instances_data)


# Заполнение связующей таблицы BookAuthor
book_authors_data=[
    (1, 1),  # "М. Горький"(AuthorID = 1) - "Мать"(AuthorID = 1)
    (1, 2),  # "М. Горький"(AuthorID = 1) - "На дне"(AuthorID = 2)
    (2, 3),  # "А. Чехов"(AuthorID = 2) - "Каштанка"(AuthorID = 3)
    (2, 4),  # "А. Чехов"(AuthorID = 2) - "Вишневый сад"(AuthorID = 4)
    (3, 5),  # "Т. Шевченко"(AuthorID = 3) - "Гайдамаки"(AuthorID = 5)
    (4, 6),  # "Л. Толстой"(AuthorID = 4) - "Война и мир"(AuthorID = 6)
    (5, 7),  # "И. Ильф"(AuthorID = 5) - "Двенадцать стульев"(AuthorID = 7)
    (6, 7),  # "Е. Петров"(AuthorID = 6) - "Двенадцать стульев"(AuthorID = 7)
    (5, 8),  # "И. Ильф"(AuthorID = 5) - "Золотой теленок"(AuthorID = 8)
    (6, 8)   # "Е. Петров"(AuthorID = 6) - "Золотой теленок"(AuthorID = 8)
]


# Вставляем данные в таблицу BookAuthor
cursor.executemany("INSERT INTO BookAuthor (Author, Book) VALUES (?, ?)",
book_authors_data)

# Добавление двух читателей в таблицу Reader
readers_data = [
    ("Иванов", "ivanov@example.com", 1),  # Читатель 1 (статус 1 - активный)
    ("Петров", "petrov@example.com", 1)   # Читатель 2 (статус 1 - активный)
]
cursor.executemany("INSERT INTO Reader (Name, Email, ReaderStatus) VALUES (?, ?,
?)", readers_data)


# Добавление книг в аренду
# Мы предполагаем, что BookID для "Мать" = 1, а для "Война и мир" = 6
# И BookInstanceID для первого экземпляра книги 1 = 1, а для второго экземпляра
книги "Война и мир" = 7
rental_data = [
    (2, 1, '2023-03-15', '2023-05-15'),  # Читатель 2 Петров арендовал первый
экземпляр книги "Мать" и вернул её
```

```
    (1, 1, '2024-02-01',None),  # Читатель_1 Иванов арендует первый экземпляр 1 книги "Мать"
    (2, 9, '2024-02-01', None)   # Читатель 2 Петров арендует единственный экземпляр  книги "Война и мир"
]
cursor.executemany('''
    INSERT INTO Rental (Reader, Instance, DateRented, DateReturned)  VALUES (?, ?, ?, ?)
''', rental_data)


# Сохранение изменений
conn.commit()

# ****************************
# Запросы на извлечение данных
# ****************************

# Запрос для получения всех экземпляров книг и информации о читателях
cursor.execute('''
    SELECT b.Title, bi.VolumeNumber, r.Name, ra.DateRented,ra.DateReturned
    FROM Rental ra
    JOIN BookInstance bi ON ra.Instance = bi.BookInstanceID
    JOIN Book b ON bi.Book = b.BookID
    JOIN Reader r ON ra.Reader = r.ReaderID
''')

# Получаем все результаты
all_rented_books = cursor.fetchall()

# Выводим результат
if all_rented_books:
    for book in all_rented_books:
        title, volume, reader_name, date_rented, date_returned = book
        print(f"Книга: {title}, Экземпляр: {volume}, Читатель: {reader_name}, Дата аренды: {date_rented}, Дата возврата: {date_returned}")
else:
    print("Нет арендованных экземпляров книг.")

print()


# !!!!!!!!! ЗАДАНИЕ 1 СТУДЕНТАМ
# Написать запрос, который выводит только те книги, которые выданы


# !!!!!!!!! ЗАДАНИЕ 2 СТУДЕНТАМ
# Написать запрос, который выводит только те книги, которые взяли и вернули
```

```
# Написать запрос - какие книги на руках у читателя с заданным ID
# Задаем ID читателя
reader_id = 1  # ID читателя для поиска

cursor.execute('''
    SELECT b.Title, bi.VolumeNumber, r.DateRented
    FROM Rental r
    JOIN BookInstance bi ON r.Instance = bi.BookInstanceID
    JOIN Book b ON bi.Book = b.BookID
    WHERE r.Reader = ? AND r.DateReturned IS NULL  -- Фильтруем по ID читателя
''', (reader_id,))

rented_books = cursor.fetchall()

if rented_books:
    print(f"Список книг на руках у читателя с ID {reader_id}:")
    for book in rented_books:
        title, volume, date_rented = book
        print(f"Книга: {title}, Экземпляр: {volume}, Дата аренды: {date_rented}")
else:
    print(f"У читателя с ID {reader_id} нет арендованных книг.")
print()


# ****************************
# РЕАЛИЗАЦИЯ ТРАНЗАКЦИИ
# ****************************

# Читатель с идентификатором ID
# арендует книгу в указанную дату

# Параметры
reader_id = 1
reader_name = 'Иванов'  # Фамилия читателя
book_title = 'Каштанка'  # Название книги
current_date = '2024-02-01'

try:
    # Начинаем транзакцию
    conn.execute('BEGIN TRANSACTION')

    # Проверка, существует ли книга с таким названием
    cursor.execute('''
        SELECT b.BookID
        FROM Book b
        WHERE b.Title = ?
    ''', (book_title,))
    book_result = cursor.fetchone()
```

```
    if not book_result:
        print(f"Книга с названием '{book_title}' не найдена.")
        conn.rollback()  # Откатываем изменения, если книга не найдена
    else:
        book_id = book_result[0]

        # Проверка, есть ли доступный экземпляр этой книги
        cursor.execute('''
            SELECT bi.BookInstanceID
            FROM BookInstance bi
            WHERE bi.Book = ? AND bi.BookInstanceID NOT IN (
                SELECT r.Instance
                FROM Rental r
                WHERE r.DateReturned IS NULL
            )
            LIMIT 1
        ''', (book_id,))
        available_instance = cursor.fetchone()

        if not available_instance:
            print(f"Нет доступных экземпляров книги '{book_title}' для аренды.")
            conn.rollback()  # Откатываем изменения, если нет доступных экземпляров
        else:
            book_instance_id = available_instance[0]

            # Обновление статуса читателя на активный (статус 1)
            cursor.execute('''
                UPDATE Reader SET ReaderStatus = 1 WHERE ReaderID = ?
            ''', (reader_id,))

            # Внесение записи о аренде
            cursor.execute('''
                INSERT INTO Rental (Reader, Instance, DateRented) VALUES (?, ?,
?)
            ''', (reader_id, book_instance_id, current_date))

            # Сохраняем изменения
            conn.commit()

            print(f"Читатель '{reader_name}' (ID: {reader_id}) успешно арендовал
книгу '{book_title}' (экземпляр ID: {book_instance_id}).")

except Exception as e:
    print(f"Произошла ошибка: {e}")
    conn.rollback()  # Откатываем изменения при ошибке
```

```
# написать код, который использует транзакцию для обеспечения атомарности
изменений:
# Определите в переменных ID читателя, номер экземпляра книги и дату возврата.
# Найдете в базе данных аренду этого читателя.
# Если аренда не найдена, откатите транзакцию
# Если аренда найдена (результат предыдущего запроса не пустой), обновите дату
возврата книги.
# Проверьте, арендовал ли читатель другие книги
# Если у читателя больше нет книг, обновит его статус на "неактивный".

# Параметры
reader_id = 1  # ID читателя
book_instance_id = 1  # ID экземпляра книги
return_date = '2024-02-10'  # Дата возврата

try:
    # Начинаем транзакцию
    conn.execute('BEGIN TRANSACTION')

    # Проверка, существует ли аренда для данного читателя и экземпляра книги
    cursor.execute('''
        SELECT r.RentalID
        FROM Rental r
        WHERE r.Reader = ? AND r.Instance = ? AND r.DateReturned IS NULL
    ''', (reader_id, book_instance_id))
    rental_record = cursor.fetchone()

    # Если аренда не найдена, откатываем транзакцию
    if not rental_record:
        print(f"Читатель с ID {reader_id} не арендовал этот экземпляр книги.")
        conn.rollback()
    else:
        # 1. Обновление записи о возврате книги
        cursor.execute('''
            UPDATE Rental
            SET DateReturned = ?
            WHERE RentalID = ?
        ''', (return_date, rental_record[0]))

        # 2. Проверка, арендовал ли читатель другие книги
        cursor.execute('''
            SELECT COUNT(*)
            FROM Rental
            WHERE Reader = ? AND DateReturned IS NULL
        ''', (reader_id,))
        active_rentals = cursor.fetchone()[0]

        # 3. Если у читателя нет других арендованных книг, обновляем его статус
на "неактивный"
        if active_rentals == 0:
            cursor.execute('''
```

```
                UPDATE Reader
                SET ReaderStatus = 2
                WHERE ReaderID = ?
            ''', (reader_id,))

        # Завершаем транзакцию
        conn.commit()
        print(f"Читатель {reader_id} успешно вернул книгу с экземпляром
{book_instance_id}.")

except Exception as e:
    # Если произошла ошибка, откатываем изменения
    print(f"Произошла ошибка: {e}")
    conn.rollback()  # Откатываем изменения при ошибке

finally:

# Не забудьте закрыть соединение с базой данных, когда закончите работу
  conn.close()
```

Результат работы программы:

```
Книга: Мать, Экземпляр: 1, Читатель: Петров, Дата аренды: 2023-03-15, Дата возврата: 2023-05-15
Книга: Мать, Экземпляр: 1, Читатель: Иванов, Дата аренды: 2024-02-01, Дата возврата: None
Книга: Война и мир, Экземпляр: 1, Читатель: Петров, Дата аренды: 2024-02-01, Дата возврата: None

Список книг на руках у читателя с ID 1:
Книга: Мать, Экземпляр: 1, Дата аренды: 2024-02-01

Читатели, которые арендовали книгу 'Мать' (Экземпляр 1):
Читатель: Петров, Дата аренды: 2023-03-15, Дата возврата: 2023-05-15
Читатель: Иванов, Дата аренды: 2024-02-01, Дата возврата: None
Читатель 'Иванов' (ID: 1) успешно арендовал книгу 'Каштанка' (экземпляр ID: 5).
Читатель 1 успешно вернул книгу с экземпляром 1.
```

Задание на самостоятельную работу

Написать код и **оформить его в виде транзакции** для возврата книги в библиотеку. Для этого нужно выполнить следующие шаги:

1. Проверить, арендована ли книга. Для этого нужно проверить в таблице Rental, что экземпляр книги (например, по номеру экземпляра) был арендован (не имеет значения, была ли дата возврата уже указана или нет).
2. Проверить, что книга действительно арендована этим читателем. Нужно удостовериться, что указанная запись в таблице Rental относится к данному читателю.

3. Обновить запись о возврате книги. После проверки, что книга была арендована, необходимо обновить таблицу Rental, указав в поле DateReturned дату возврата книги.
4. Обновить статус читателя. Если у читателя больше нет арендованных книг, его статус должен быть обновлен в таблице Reader на "неактивный" (статус = 2).

Все изменения должны быть выполнены внутри одной транзакции, чтобы гарантировать, что либо все изменения будут выполнены, либо, в случае ошибки, не будет никаких частичных изменений.