Лабораторная работа 1. Создание базы данных SQLite

I. Проектирование базы данных

Концептуальная (инфологическая) модель данных для библиотеки

Опишем сущности предметной области библиотека.

Сущность «Книга» будет иметь следующие атрибуты: название книги, жанр, год публикации и ISBN.

Сущность «Автор» будет иметь следующие атрибуты: имя автора, дата рождения, национальность.

Сущность «Экземпляр книги» будет иметь следующие атрибуты: внутренний номер тома или экземпляра), статус экземпляра (доступен, в аренде и т. д.), состояние (новый, б/у, поврежденный и т.д.).

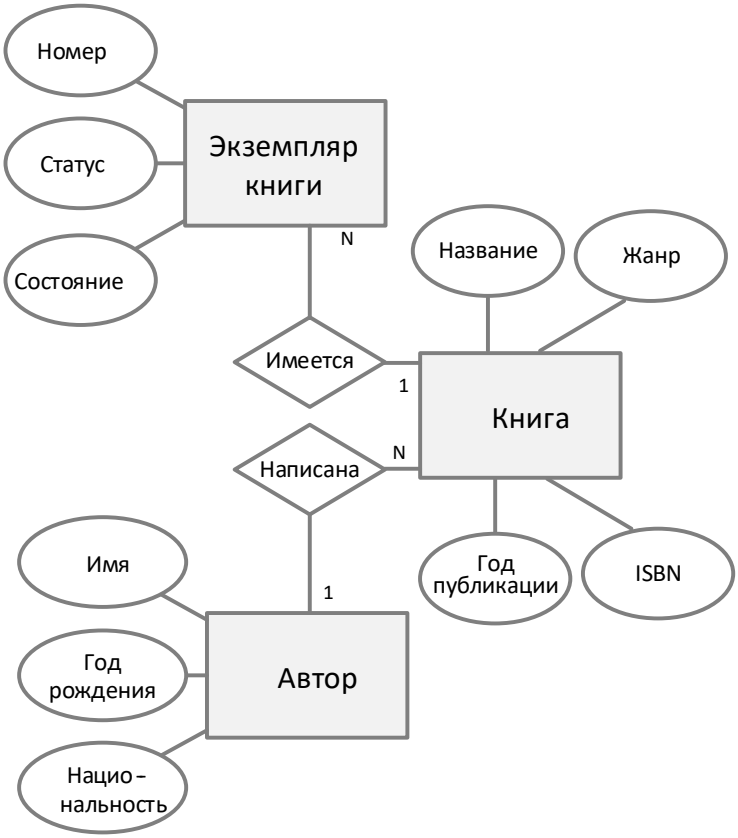

Рисунок 1 – Концептуальная модель (в нотации Питера Чена)

Связь между сущностями «Автор» и «Книга» является связью «один-ко-многим» (one-to-many). Это означает, что каждый автор может иметь много книг, но каждая книга имеет только одного автора (это решение принято для упрощения).

Связь между «Экземпляром» и «Книгой» является связью «одним-ко-многим» (one-to-many). Это означает, что в библиотеке у каждой книги может быть много экземпляров, но каждый экземпляр книги принадлежит только одной книге.

Логическая модель

Эта модель данных представляет собой набор таблиц для хранения информации о сущностях предметной области. Каждая таблица содержит свои собственные *атрибуты* (синонимы – поля или столбцы), а также особые атрибуты – *первичные* (PRIMARY KEY) и *внешние ключи* (FOREIGN KEY).

При переходе от концептуальной модели данных к логической выполняется ряд формальных шагов. Рассмотрим некоторые из них:

1. Каждая сущность приобретает первичный ключ, который уникально идентифицирует записи (синонимы – строки или кортежи) в таблице.
2. Названия столбцов и атрибутов представляются на английском языке и без пробелов.
3. Связь «один-ко многим» между двумя таблицами реализуется через внешний ключ. Внешний ключ появляется в той сущности, которая в данной связи описывается понятием «многим». Другими словами, внешний ключ добавляется к той таблице, которая содержит множество записей, связанных с одной записью в другой таблице.

Так, поле Author, которое появилось в таблице Book, является внешним ключом, связывающим каждую книгу со своим автором из таблицы Author. А поле Book, которое появилось в таблице Volume, является внешним ключом, связывающим экземпляр с книгой из таблицы Book.

Для хранения в БД различных данных, которые часто повторяются в записях одной или даже нескольких таблиц, полезно создать *таблицы-справочники*. Использование таблиц-справочников помогает избежать дублирования данных, обеспечивает единообразие, компактность и целостность информации. Так, атрибут Genre из сущности «Книга» был преобразован в справочную таблицу с названием Genre. А в таблице Book этот атрибут стал внешним ключом. Справочная таблица Genre впоследствии может быть заполнена различными жанрами, например: роман, фантастика, детектив, приключения, фэнтези и т.д. Эти названия один раз будут занесены в БД, а каждая запись книги будет содержать только допустимый номер жанра из справочника. Если нужно изменить или добавить новые значения, можно это сделать только в справочной таблице, и эти изменения автоматически

отразятся во всех местах, где используются эти значения. Такие же справочные таблицы можно было определить и для атрибутов экземпляра книги – статус VolumeStatus, состояние Condition; и для атрибута читателя – ReaderStatus и других. Это не сделано из соображений компактности примера.

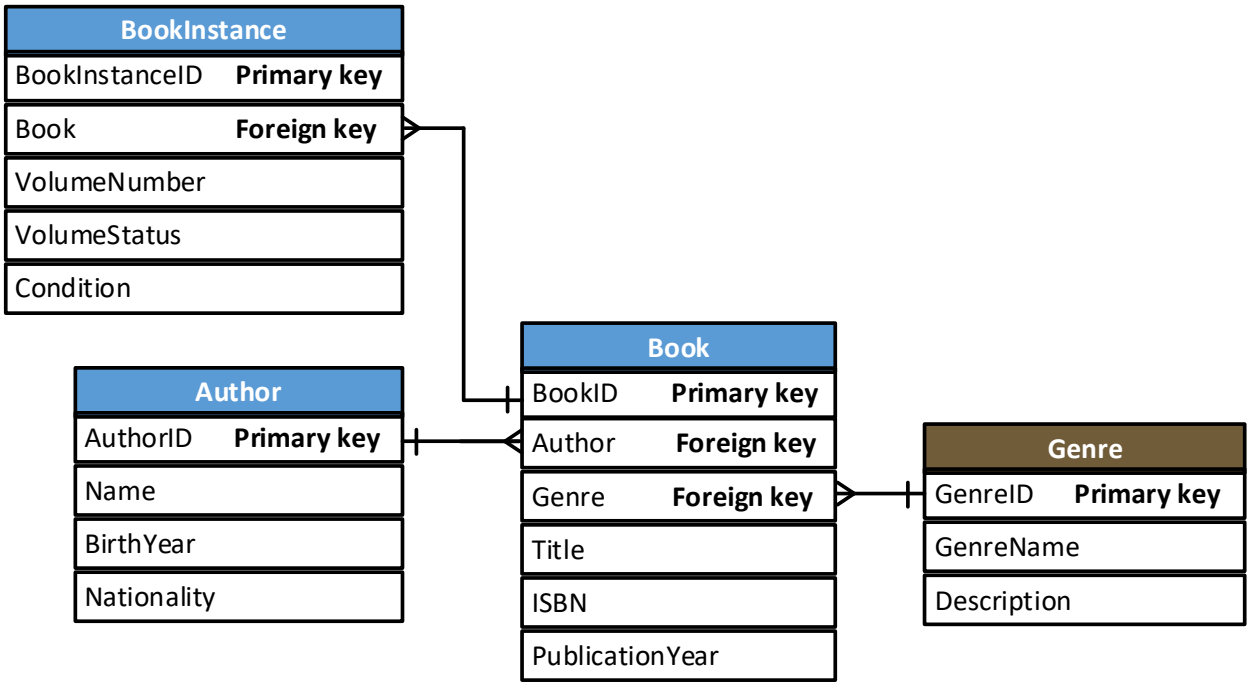

Рисунок 2 – Логическая модель

Физическая модель

Переход от логической модели к физической представляет собой преобразование абстрактной модели данных к конкретной реализации базы данных с учетом выбранной СУБД. В процессе перехода:

1. Определяются типы данных для каждого атрибута.
2. Устанавливаются *ограничения целостности данных*, такие как уникальные ограничения и условия проверки.
3. Устанавливаются *дополнительные свойства* для каждого атрибута, такие как значение по умолчанию, ограничения на длину строки и т. д.
4. При необходимости проводится *нормализация данных* для оптимизации структуры базы данных.

Ниже представлена физическая модель проектируемой БД в текстовом виде с указанием типов и обязательности для СУБД SQLite:

– NOT NULL – указывает, что при занесении записи в базу банных значение атрибута не может быть пустым;

– UNIQUE – гарантирует, что значение атрибута уникально для каждой записи в таблице;

– DEFAULT – устанавливает для атрибута значение по умолчанию, которое будет использоваться, если значение не указано при вставке новой записи;

– CHECK – позволяет определить условие, которое должно выполняться для значения атрибута.

В этой модели добавлен параметр NOT NULL для основных атрибутов таблиц, без которых вся запись в таблице теряет смысл – как можно внести запись о книге, не указав её названия?

Добавлен параметр UNIQUE для атрибута ISBN в таблице Book, для атрибута GenreName в таблице Genre, чтобы гарантировать уникальность ISBN для каждой книги и уникальность имен жанров.

Добавлен параметр CHECK для атрибута BirthYear в таблице Author, чтобы проверить, что дата рождения автора не превышает 2010 год.

Установлены параметры NOT NULL и DEFAULT для атрибута VolumeNumber в таблице Volume, чтобы установить значение по умолчанию равное 1, т.к. у книги должен быть хотя бы один экземпляр.

Эти дополнения обеспечат большую гибкость и защиту целостности данных в базе данных.

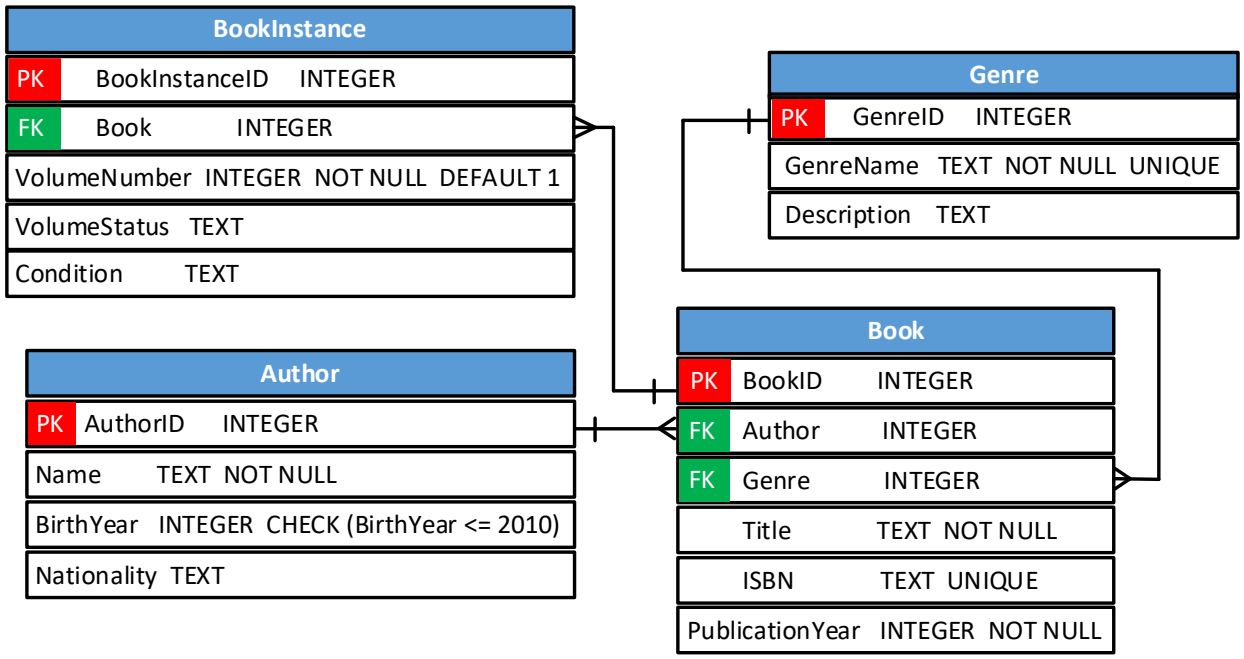

Рисунок 3 – Физическая модель

II. Создание базы данных в Python-программе

Для работы с базой данных под управлением СУБД SQLite из Python-программы не требуется инсталлировать библиотеку sqlite3 в среду программирования, так как этот модуль уже включен в стандартную библиотеку Python. Можно сразу приступать к использованию соответствующих функций и методов из модуля sqlite3 для создания базы данных SQLite, выполнения запросов и работы с данными в Python-программе.

Структура программы

Для создания базы данных SQLite и подключения к ней из Python с использованием модуля sqlite3 потребуется выполнить следующие шаги:

1. Подключение модуля sqlite3 (команда import).
2. Установка соединения с базой данных через объект соединения с БД с помощью функции connect() модуля sqlite3, с указанием имени файла базы данных. Если файл с расширением .db не существует, он будет создан автоматически в том же каталоге, где находится python-приложение.
3. Создание *объекта курсора*, который используется для выполнения SQL-запросов и работы с данными в базе данных. Объект курсор создается путём вызова метода cursor() у объекта соединения.
4. Выполнение SQL-запросов с помощью объекта курсора. Запросы к базе данных, работающей под управлением sqlite3, записываются в виде строк и передаются на выполнение методу execute() объекта курсора.
5. Завершение работы программы путем закрытия курсора и разрыва соединения с базой данных с помощью метода close().

Чтобы избежать ситуации перезаписи ранее созданных таблиц и, возможно, внесённых в БД данных, следует добавить логику, которая проверяет, существует ли файл базы данных, перед тем как осуществлять подключение к базе данных и создавать её таблицы. Если файл уже существует, программа просто подключается к существующей БД.

В SQLite, если создается столбец с типом INTEGER PRIMARY KEY, то он по умолчанию является автоинкрементируемым. Это означает, что при вставке данных в таблицу можно не указывать значение для этого столбца, и SQLite автоматически присвоит ему уникальное значение. Однако, если необходимо, чтобы СУБД SQLite гарантировала неповторяемость значений даже после удаления записей, следует использовать ключевое слово AUTOINCREMENT.

```
import sqlite3
import os

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

# 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,
        Author INTEGER NOT NULL,
        Genre INTEGER NOT NULL,
        Title TEXT NOT NULL,
        PublicationYear INTEGER NOT NULL,
        ISBN TEXT UNIQUE,
        FOREIGN KEY (Author) REFERENCES Author(AuthorID),
        FOREIGN KEY (Genre) REFERENCES Genre(GenreID)
    )''')

    # 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)
    )''')

    # 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()

# Далее можно выполнять операции с базой данных (добавление/чтение данных)
# .  .  .

# Сохранение изменений
conn.commit()

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

III. Добавление данных в таблицы базы данных

Представленный ниже код добавляет данные об авторах, жанрах, читателях, книгах и экземплярах книг в соответствующие таблицы базы данных.

```
# Внесение одиночной записи в таблицу Author
author_data = ("М. Горький", 1868, "русский")
cursor.execute("INSERT INTO Author (Name, BirthYear, Nationality) VALUES (?, ?,
?)", author_data)

# Внесение множества данных в таблицу Author
authors_data = [
    ("А. Чехов", 1860, "русский"),
    ("Т. Шевченко", 1814, "украинец"),
    ("Л. Толстой", 1828, "русский")
]
cursor.executemany("INSERT INTO Author (Name, BirthYear, Nationality) VALUES
(?, ?, ?)", authors_data)

# Добавление жанров
genres_data = [
    ("роман", "Произведение прозаического жанра, длинное по объему, описывающее
развитие сюжета и психологию персонажей"),
    ("драма", "Произведение, предназначенное для театрального исполнения, с
акцентом на драматические события и конфликты"),
    ("повесть", "Произведение прозаического жанра, короткое по объему, часто с
ограниченным числом персонажей и событий"),
    ("поэма", "Произведение, представляющее собой лиро-эпическую или эпическую
форму, чаще всего написанное в стихотворной форме")
]
cursor.executemany("INSERT INTO Genre (GenreName, Description) VALUES (?, ?)",
genres_data)

# Добавление книг
books_data = [
    (1, 1, "Мать", 1996, "978-1234567890"),
    (1, 2, "На дне", 1992, "978-0987654321"),
```

```
    (2, 3, "Каштанка", 2017, "978-5432167890"),
    (2, 3, "Вишневый сад", 2017, "978-6432167890"),
    (3, 4, "Гайдамаки", 1930, "978-2345678901"),
    (4, 1, "Война и мир", 2025, "978-3456789012")
]
cursor.executemany("INSERT INTO Book (Author, Genre, Title, PublicationYear,
ISBN) VALUES (?, ?, ?, ?, ?)", books_data)

# Добавление экземпляров
instances_data = [
    (1, 1, "Доступен", "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, "Доступен", "Отличное")
]
cursor.executemany("INSERT INTO BookInstance (Book, VolumeNumber, VolumeStatus,
Condition) VALUES (?, ?, ?, ?)", instances_data)

# Сохранение изменений
conn.commit()
```

При заполнении базы данных вручную через SQL-запросы есть несколько важных аспектов, на которые следует обратить внимание. Следует убедиться, что данные, которые вставляются в БД, соответствуют ожидаемым типам и ограничениям, установленным для каждого столбца. Например, даты указаны в правильном формате, текстовые значения не превышают допустимой длины и т. д.

При вставке данных в связанные таблицы (например, книги и их авторы), необходимо убедиться, что связи между ними правильно установлены. Например, каждый автор книги должен иметь соответствующую запись в таблице авторов. Следует проверить, что значения, которые вставляются в столбцы с ограничением уникальности (например, ISBN книги, адрес электронной почты читателя), действительно уникальны, а данные с атрибутом CHECK соответствуют условиям, указанным при создании таблицы.

IV. Выборка данных и печать результатов

Для получения нужной информации из базы данных с помощью SQ-запросов используются различные операции фильтрации, сортировки, объединения и другие манипуляции с данными. SQL-запросы не пишутся

напрямую в коде на языке программирования в виде строковых выражений. Вместо этого, они встраиваются в код с помощью специального синтаксиса, предоставляемого конкретной библиотекой или модулем, который обеспечивает доступ к базе данных. Этот синтаксис позволяет программе

Для этих целей в библиотеке sqlite3 используется метод execute() объекта курсора. Ему передается строка с SQL-запросом в качестве аргумента. Затем с помощью средств библиотеки sqlite3 этот запрос обрабатывается и исполняется непосредственно в базе данных SQLite. Результаты исполнения запроса возвращаются обратно в программу на Python для дальнейшей обработки. Существует два варианта синтаксиса для встраивания запроса в код программы.

```
#Вариант 1. Запрос напрямую указывается в методе execute в кавычках:
cursor.execute("SELECT * FROM Book")

#Вариант 2. Запрос записывается в строку, а затем передается в метод execute:
query = "SELECT * FROM Book"
# Выполнение запроса
cursor.execute(query)
```

Оба этих подхода эквивалентны и выполняют одинаковую операцию. Кроме того, для обозначения строк SQL-запроса можно использовать как двойные кавычки ("), так и одинарные кавычки ('). Стандартный стиль написания кода в Python рекомендует использовать один тип кавычек везде в одном коде для лучшей читаемости. Важно выбрать один стиль и придерживаться его во всем проекте для единообразия.

Методы извлечения данных из объекта курсора

Когда выполняется запрос с помощью cursor.execute(), результаты запроса сохраняются в объекте-курсоре. В зависимости от метода извлечения данных (fetchall() или fetchone()), можно получить все строки или только одну. Если результат запроса пуст, оба метода вернут None или пустую коллекцию.

Метод fetchall() используется для извлечения всех строк из курсора в виде списка кортежей, где каждый кортеж представляет одну строку результата. Если в результате запроса нет данных, будет возвращён пустой список ([]).

```
# Получение результатов запроса и вывод на экран
cursor.execute("SELECT * FROM Book")
results = cursor.fetchall()
```

```
# Обработка результатов
if results:  # Проверяем, есть ли данные
    for row in results:
        print(row)
else:
    print("Нет данных для вывода.")
```

Метод fetchone() извлекает только одну строку из результата запроса. В отличие от fetchall(), который возвращает список всех строк, fetchone() вернёт первую строку (или None, если строк нет).

```
# Выполнение запроса для извлечения данных из таблицы Author
cursor.execute("SELECT * FROM Author")

# Получение первой строки результата
row = cursor.fetchone()

# Обработка первой строки
if row:
    print(row)  # Вывод первой строки
else:
    print("Нет данных для вывода.")
```

Использование символа "*" в SQL-запросах

Символ "*" в операторе SELECT означает выбор всех столбцов из указанной таблицы. Это удобно при работе с небольшими таблицами или отладке, но в больших базах данных может замедлять выполнение запросов. Поэтому рекомендуется выбирать только необходимые столбцы, перечисляя их через запятую. Это снижает нагрузку на сервер, уменьшает объем передаваемых данных и ускоряет выполнение запросов.

```
# SQL-запрос на выборку наименований жанров
"SELECT GenreName FROM Genre"

# SQL-запрос на выборку имен авторов и их национальностей
"SELECT Name, Nationality FROM Author"
```

Запросы с условиями WHERE

```
# SQL-запрос на выборку книг, опубликованных после 2000 года
"SELECT Title, PublicationYear, ISBN FROM Book WHERE PublicationYear > 2000"


# SQL-запрос на выборку книг определенного автора (например, с ID 1):
# Если передаём один параметр в execute(), он должен быть
# в кортеже ((author_id,)), иначе Python выдаст ошибку
author_id = 1
cursor.execute('SELECT * FROM Book WHERE Author = ?', (author_id,))
```

Запросы с условиями ORDER BY

DESC и ASC – это сокращение от английских слов "descending" и "ascending", что означает "по убыванию" и "по возрастанию".

```
# SQL-запрос на выборку книг, отсортированных по году публикации по убыванию
"SELECT Title, PublicationYear FROM Book ORDER BY PublicationYear DESC"


# SQL-запрос на выборку авторов, отсортированных по году рождения по возрастанию
"SELECT Name, BirthYear FROM Author ORDER BY BirthYear ASC"
```

Универсальные функции для выполнения запросов

Для выполнения запросов и печати результатов можно создать собственные универсальные функции:

```
#  Функция execute_sql_query принимает SQL-запрос в виде строки,
# выполняет его и возвращает результат запроса
def execute_sql_query(cursor, query, params=None):
    cursor.execute(query, params or ())
    return cursor.fetchall()

#  Функция принимает результат SQL-запроса и печатает его
def print_query_results(results):
    if not results:
        print("Результат запроса пуст")
    else:
        for row in results:
            print(row)
    print("\n")
```

V. Окончательный базовый код 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,
# что бы 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,
        Author INTEGER NOT NULL,
        Genre INTEGER NOT NULL,
        Title TEXT NOT NULL,
        PublicationYear INTEGER NOT NULL,
        ISBN TEXT UNIQUE,
        FOREIGN KEY (Author) REFERENCES Author(AuthorID),
        FOREIGN KEY (Genre) REFERENCES Genre(GenreID)
    )''')

    # 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)
    )''')

    # 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()

# Далее можно выполнять операции с базой данных (добавление/чтение данных)
# # Внесение одиночной записи в таблицу Author
author_data = ("М. Горький", 1868, "русский")
cursor.execute("INSERT INTO Author (Name, BirthYear, Nationality) VALUES (?, ?,
?)", author_data)

# Внесение множества данных в таблицу Author
authors_data = [
    ("А. Чехов", 1860, "русский"),
    ("Т. Шевченко", 1814, "украинец"),
    ("Л. Толстой", 1828, "русский")
]
cursor.executemany("INSERT INTO Author (Name, BirthYear, Nationality) VALUES (?,
?, ?)", authors_data)

# Добавление жанров
genres_data = [
    ("роман", "Произведение прозаического жанра, длинное по объему, описывающее
развитие сюжета и психологию персонажей"),
    ("драма", "Произведение, предназначенное для театрального исполнения, с
акцентом на драматические события и конфликты"),
    ("повесть", "Произведение прозаического жанра, короткое по объему, часто с
ограниченным числом персонажей и событий"),
    ("поэма", "Произведение, представляющее собой лиро-эпическую или эпическую
форму, чаще всего написанное в стихотворной форме")
]
cursor.executemany("INSERT INTO Genre (GenreName, Description) VALUES (?, ?)",
genres_data)

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

# Добавление экземпляров
```

```
instances_data = [
    (1, 1, "Доступен", "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, "Доступен", "Отличное")
]
cursor.executemany("INSERT INTO BookInstance (Book, VolumeNumber, VolumeStatus,
Condition) VALUES (?, ?, ?, ?)", instances_data)


# Сохранение изменений
conn.commit()


# Получение результатов запроса и вывод на экран
cursor.execute("SELECT * FROM Book")
results = cursor.fetchall()

# Обработка результатов
if results:  # Проверяем, есть ли данные
    for row in results:
        print(row)
else:
    print("Нет данных для вывода.")


# Выполнение запроса для извлечения данных из таблицы Author
cursor.execute("SELECT * FROM Author")

# Получение первой строки результата
row = cursor.fetchone()

# Обработка первой строки
if row:
    print(row)  # Вывод первой строки
else:
    print("Нет данных для вывода.")


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

Результат работы программы:

```
(1, 1, 'Мать', '1', 1996, '978-1234567890')
(2, 2, 'На дне', '1', 1992, '978-0987654321')
(3, 3, 'Каштанка', '2', 2017, '978-5432167890')
(4, 3, 'Вишневый сад', '2', 2017, '978-6432167890'
(5, 4, 'Гайдамаки', '3', 1930, '978-2345678901')
(6, 1, 'Война и мир', '4', 2025, '978-3456789012')

(1, 'М. Горький', 1868, 'русский')
```

Задание на самостоятельную работу

1. Запустить базовую программу и получить результаты запросов для извлечения данных из таблиц Book и Author.
2. Добавить данные в таблицы БД: 1 автор, 2 его книги и по 3 экземпляра каждой книги.
3. Написать SQL-запросы на вывод:

   названий жанров;
   названий жанров и их краткие описания;

   фамилий всех авторов и их годы рождения;
   фамилий авторов, родившихся до 1850 года;
   фамилий авторов, отсортированных по алфавиту (в порядке возрастания);

   названий книг, отсортированных в алфавитном порядке (в порядке убывания);
   названий книг, изданных до 2000 года;
   названий книг и их ISBN.

Примечание. Для исполнения запросов и вывода результатов использовать собственные универсальные функции, приведенные в методических указаниях к лабораторной работе.