Практическое занятие 4. Развитие модели данных. Связь «многие-ко-многим»

I. Развитие модели реляционной базы данных

В исходной модели базы данных между сущностями «Книга» и «Автор» для упрощения была принята связь «один-ко-многим». Т.е. один автор имеет много книг, но каждая книга написана одним автором. Однако, в связи с: расширением функционала и добавлением возможности работы с книгами, написанными в соавторстве, возникла необходимость перейти к модели, содержащей связь «многие-ко-многим» (рис.1). Это позволит хранить информацию о книгах, имеющих нескольких авторов.

Если бы мы попытались создать прямую связь «многие-ко-многим» между Авторами и Книгами, у нас бы возникла проблема: как хранить информацию о том, что автор написал несколько книг, а книга написана несколькими авторами? Нельзя просто добавить в таблицу Author поле «СписокКниг» или в таблицу Book «СписокАвторов». Не известно каков размер этого списка. А добавление полей из расчета максимально возможных книг у одного автора или максимально возможных авторов у книги приведет к избыточности и усложнит обновление данных.

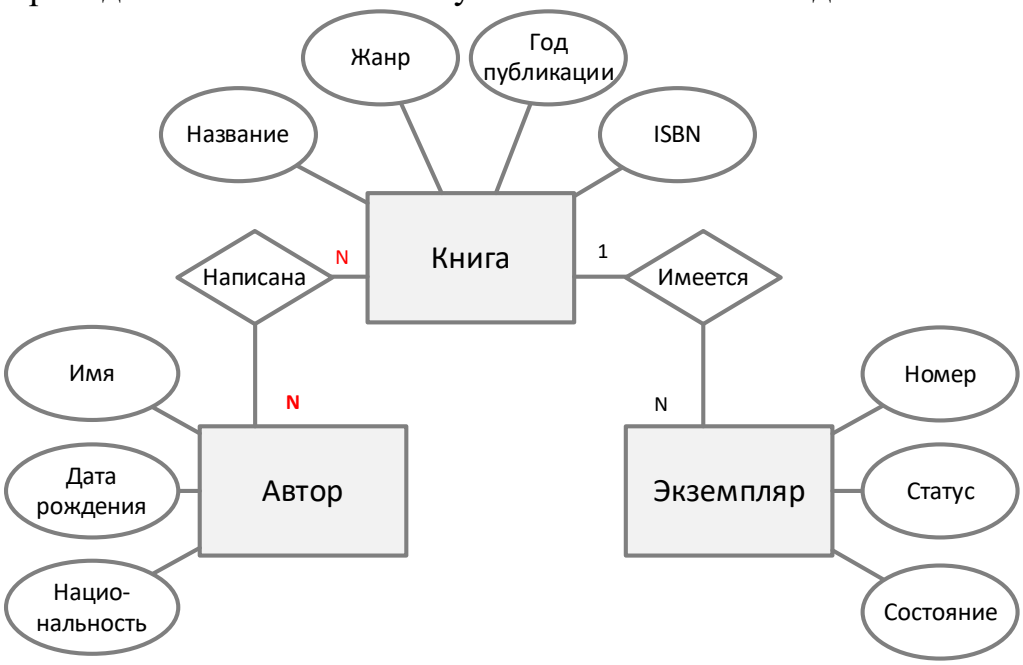

Рисунок 1 – Обновленная концептуальная модель

Для реализации связи «многие-ко-многим» в логической модели между таблицами Book и Author необходимо ввести разрешающую (или связующую) таблицу BookAuthor (рис. 2). Здесь слово «разрешены» означает решение проблемы, связанной с тем, что в плоских таблицах организовать такую связь напрямую нельзя. Связующая таблица будет содержать свой собственный первичный ключ BookAuthorID, а также внешние ключи, ссылающиеся на поле BookID таблицы «Книга» и поле AuthorID таблицы «Автор» (ID_Автора).

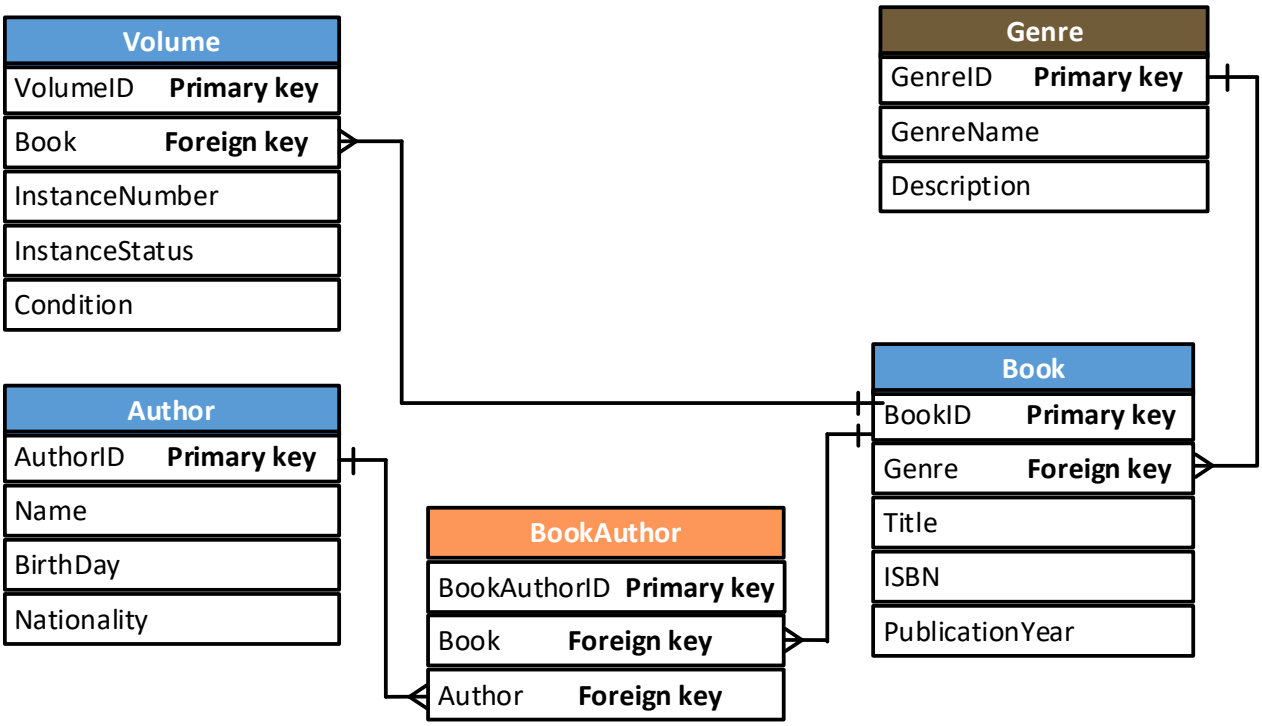

Рисунок 2 – Обновленная логическая модель

Связь «многие ко многим» используется не только для авторов и книг, но и в других сферах:

- фильмы и актеры: один актер может сняться в нескольких фильмах, а один фильм может включать множество актеров.
- студенты и курсы: один студент может пройти несколько курсов, а один курс может быть посещен несколькими студентами.
- товары и заказы: один товар может присутствовать в нескольких заказах, а один заказ может включать несколько товаров.

Если связь «многие ко многим» фактически является связью «один ко многим», то использование промежуточной таблицы может привести к излишним сложностям. Важно заранее понимать, что связь «многие ко многим» действительно имеет место.

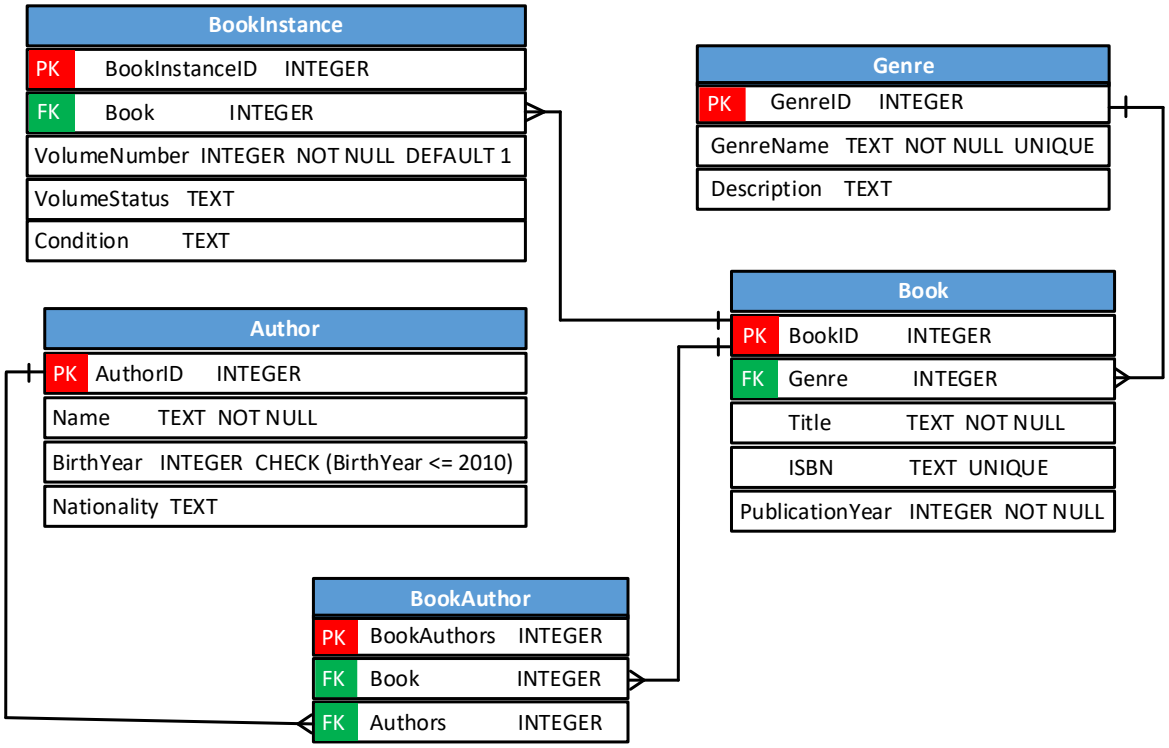

Рисунок 3 – Физическая модель

Как хранятся данные?

Таблица Author

AuthorID	Name
1	Л. Толстой
2	Ф. Достоевский
3	А. Пушкин

Таблица Book

BookID	Title	Year
101	Война и мир	1869
102	Преступление и наказание	1866
103	Евгений Онегин	1833
104	Антология русской литературы	2020

Таблица Author_Book

AuthorID	BookID
1	101
2	102
3	103
1	104
2	104
3	104

Видно, что книга «Антология русской литературы» имеет сразу трёх авторов**,** и это корректно хранится в промежуточной таблице**.**

II. Оператор GROUP_CONCAT

Функция GROUP_CONCAT в SQL используется для объединения значений нескольких строк в одну строку. Это особенно полезно, когда нужно агрегировать данные из нескольких записей в одну строку результата группировки. В отличие от простых агрегатных функций, таких как SUM или AVG, которые возвращают одно значение для группы, GROUP_CONCAT возвращает строку, содержащую все значения из группы, объединенные определенным разделителем.

Пример:

Таблица Book

Book.Title	…
Двенадцать стульев	…
Двенадцать стульев	…

Таблица Author

Author.Name	…
И. Ильф	…
Е. Петров	…

После GROUP_CONCAT(Author.Name, ', ') результат будет:

Book.Title	**Authors (строка со списком авторов)**
Двенадцать стульев	И. Ильф, Е. Петров

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,
# 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,
        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)
    )''')

    # 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", "Стихотворное произведение лиро-эпической или эпической формы ")
]
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, "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"),
    (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 = 4) - "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)

# Сохранение изменений
conn.commit()

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

# Запрос для получения списка книг с их авторами
# SELECT - Выбираем названия книг и имена авторов
# Первый JOIN:
#   - соединяет таблицу Book со связующей таблицей BookAuthor по полю BookID
# Второй JOIN:
#   - подсоединяет к полученному результату имена авторов из таблицы Author по полю AuthorID

cursor.execute('''
SELECT Book.Title, Author.Name
FROM Book
JOIN BookAuthor ON Book.BookID = BookAuthor.Book
JOIN Author ON BookAuthor.Author = Author.AuthorID
''')
# Получение всех результатов запроса
results = cursor.fetchall()
# Печать результатов
print("*** Полный список, 2 столбца: ")
if results:
    for row in results:
        print(f"Книга: {row[0]}, Автор: {row[1]}")
else:
    print("Нет данных для вывода.")
print()
```

```
# Запрос для получения списка книг с их авторами в одной строке
# GROUP_CONCAT(Author.Name, ', ') объединяет имена авторов одной книги в одну строку.
# GROUP BY Book.BookID группирует данные по книгам.
# ORDER BY Book.Title сортирует результаты по названию книги.

cursor.execute('''
SELECT Book.Title, GROUP_CONCAT(Author.Name, ', ') AS Authors
FROM Book
JOIN BookAuthor ON Book.BookID = BookAuthor.Book
JOIN Author ON BookAuthor.Author = Author.AuthorID
GROUP BY Book.BookID
ORDER BY Book.Title
''')
# Получение всех результатов запроса
results = cursor.fetchall()
print("*** Полный список, 2 столбца. Авторы - в одной строке: ")
# Печать результатов
if results:
    for row in results:
        print(f"Книга: {row[0]}, Автор(ы): {row[1]}")
else:
    print("Нет данных для вывода.")
print()

# Запрос для получения книг с заданным числом авторов и списком их именами
# SELECT: Выбираем название книги, имена авторов (через GROUP_CONCAT)
# и  количество авторов (через COUNT)
# JOIN и JOIN: Соединяем Book с BookAuthor (по BookID) и BookAuthor с Author (по AuthorID)
# GROUP BY: Группируем по книге для подсчета авторов
# HAVING: Фильтруем по количеству авторов (num_authors)

num_authors = 2
cursor.execute('''
SELECT Book.Title, GROUP_CONCAT(Author.Name, ', ') AS Authors,
COUNT(BookAuthor.Author) AS AuthorCount
FROM Book
JOIN BookAuthor ON Book.BookID = BookAuthor.Book
JOIN Author ON BookAuthor.Author = Author.AuthorID
GROUP BY Book.BookID
HAVING AuthorCount = ?
''', (num_authors,))

# Получение всех результатов запроса
results = cursor.fetchall()
print(f"*** Книги, у которых {num_authors} автора и список этих авторов:")
# Печать результатов
if results:
    for row in results:
        print(f"Книга: {row[0]}, Автор(ы): {row[1]}")
else:
    print("Нет данных для вывода.")
```

```
print(
)
# Не забудьте закрыть соединение с базой данных, когда закончите работу

conn.close()
```

Результат работы программы:

```
*** Полный список, 2 столбца:
Книга: Мать, Автор: М. Горький
Книга: На дне, Автор: М. Горький
Книга: Каштанка, Автор: А. Чехов
Книга: Вишневый сад, Автор: А. Чехов
Книга: Гайдамаки, Автор: Т. Шевченко
Книга: Война и мир, Автор: Л. Толстой
Книга: Двенадцать стульев, Автор: И. Ильф
Книга: Двенадцать стульев, Автор: Е. Петров
Книга: Золотой теленок, Автор: И. Ильф
Книга: Золотой теленок, Автор: Е. Петров

*** Полный список, 2 столбца. Авторы - в одной строке:
Книга: Вишневый сад, Автор(ы): А. Чехов
Книга: Война и мир, Автор(ы): Л. Толстой
Книга: Гайдамаки, Автор(ы): Т. Шевченко
Книга: Двенадцать стульев, Автор(ы): И. Ильф, Е. Петров
Книга: Золотой теленок, Автор(ы): И. Ильф, Е. Петров
Книга: Каштанка, Автор(ы): А. Чехов
Книга: Мать, Автор(ы): М. Горький
Книга: На дне, Автор(ы): М. Горький

*** Книги, у которых 2 автора и список этих авторов:
Книга: Двенадцать стульев, Автор(ы): И. Ильф, Е. Петров
Книга: Золотой теленок, Автор(ы): И. Ильф, Е. Петров
```

Задание на самостоятельную работу

1. В запросе для получения списка книг с их авторами выдать по 2 поля из таблицы Autor и из таблицы Book (по выбору студента).
2. Написать запрос, который выводит всех авторов и список их книг в одной строке.
3. Написать запрос, который получает число книг и выводит авторов, которые написали заданное количество книг.