СВЯЗИ МЕЖДУ СУЩНОСТЯМИ В БД

Что такое сущности и связи между ними

Сущность – это объект или концепт, который имеет значение в контексте базы данных. В реляционной модели данных сущности представляют собой таблицы. Примеры сущностей:
Клиенты, Заказы, Продукты и т.д.
Книги (Books), Авторы (Authors), Жанры (Genres)

Атрибуты сущности – это характеристики, которые описывают сущность.
Например, сущность "Клиент" может иметь атрибуты: Name, Address.
Сущность "Книга" может иметь следующие атрибуты: Title, PublishDate.

Связь между сущностями – это отношение, которое существует между двумя или более сущностями. Эти связи отображают, как данные в одной таблице связаны с данными в другой. В реляционных базах данных эти связи реализуются с помощью внешних ключей.

Типы связей между сущностями

1. Один к одному (1:1)

Связь **1:1** возникает, когда каждой записи в одной таблице соответствует только одна запись в другой таблице, и наоборот.
Пример: у каждого клиента есть только один адрес (предположим, что в системе учитывается только один постоянный адрес).
В реляционной БД это может быть реализовано с помощью внешнего ключа. Например, таблица `Client` может иметь внешний ключ, указывающий на таблицу `Address`, или наоборот.

2. Один ко многим (1:M)

Связь **1:M** возникает, когда одной записи в одной таблице соответствует несколько записей в другой таблице, но каждая запись во второй таблице относится только к одной записи в первой таблице.
Пример: один клиент может иметь несколько заказов, но каждый заказ связан только с одним клиентом.
В этом случае внешняя таблица (например, `Orders`) будет содержать внешний ключ, который ссылается на первую таблицу (`Clients`).

3. Многие ко многим (M:N)

Связь **M:N** возникает, когда несколько записей в одной таблице могут быть связаны с несколькими записями в другой таблице.
Пример: один заказ может включать несколько продуктов, и один продукт может быть в нескольких заказах.
Для реализации такой связи используется промежуточная таблица (например, `OrderProducts`), которая хранит внешний ключ на каждую из таблиц (например, `Orders` и `Products`).

Реализация связей в реляционных БД

Первичный ключ (Primary key) – это уникальный идентификатор записи в таблице. Он используется для гарантии, что каждая запись в таблице будет уникальной. В реляционной базе данных для каждой таблицы обычно есть только один первичный ключ, и его значения должны быть уникальными и не равными NULL.
Пример:
В таблице Authors может быть поле AuthorID, которое будет первичным ключом.
В таблице Books может быть поле BookID, которое также будет первичным ключом.

Внешний ключ (Foreign Key) – это идентификатор, который служит для связывания таблиц (а вернее – записей в разных таблицах). Внешний ключ в одной таблице указывает на первичный ключ другой таблицы, тем самым создавая связь между сущностями. Внешний ключ позволяет **поддерживать согласованность данных** между таблицами. Внешний ключ в одной таблице указывает на первичный ключ другой таблицы. Внешний ключ позволяет поддерживать связи между сущностями.

Целостность данных:

Ссылочная целостность (Referential Integrity): Гарантирует, что данные в таблицах всегда будут согласованы. Например, если автор удаляется из таблицы Authors, то все связанные с ним книги должны быть либо удалены, либо обновлены (каскадное удаление или обновление).

Связь многие ко многим (M:N)

Связь многие ко многим возникает, когда несколько записей в одной таблице могут быть связаны с несколькими записями в другой таблице. Это тип связи, при котором один объект может быть связан с множеством других объектов, и наоборот.

Пример:

Представим базу данных библиотеки, где одна книга может быть написана несколькими авторами, а один автор может быть автором нескольких книг. В этом случае мы имеем связь многие ко многим между таблицами Books и Authors.

Чтобы реализовать такую связь в реляционной базе данных, создается **промежуточная таблица** (иногда ее называют **таблицей-связки**), которая будет содержать внешние ключи на обе связанные таблицы. Таким образом, эта таблица будет хранить все возможные комбинации записей, которые связывают книги и 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.

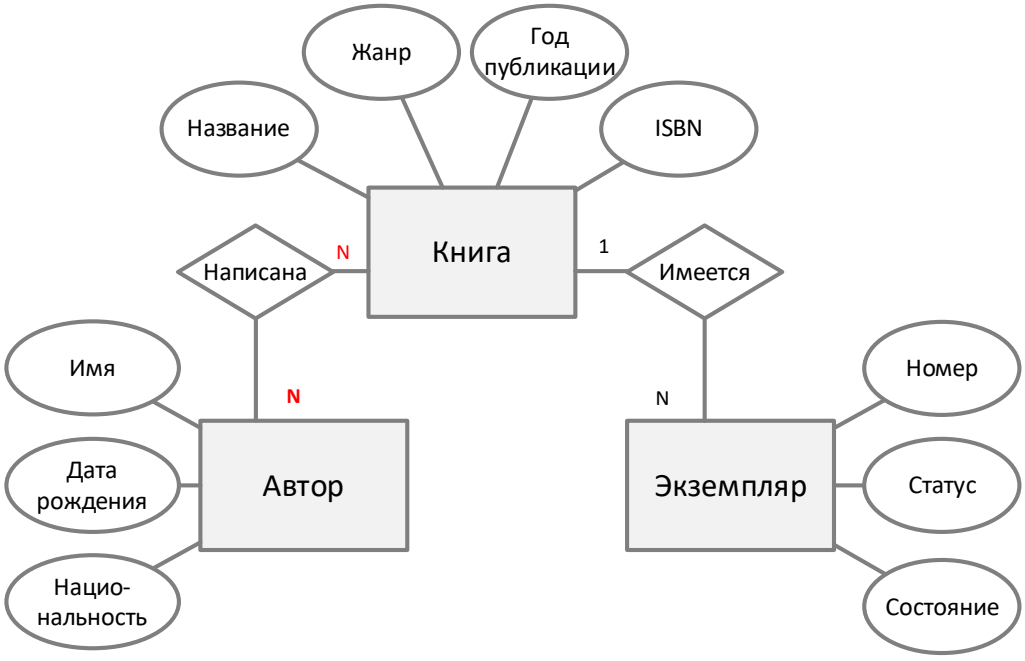

Рисунок 1 – Обновленная концептуальная модель

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).

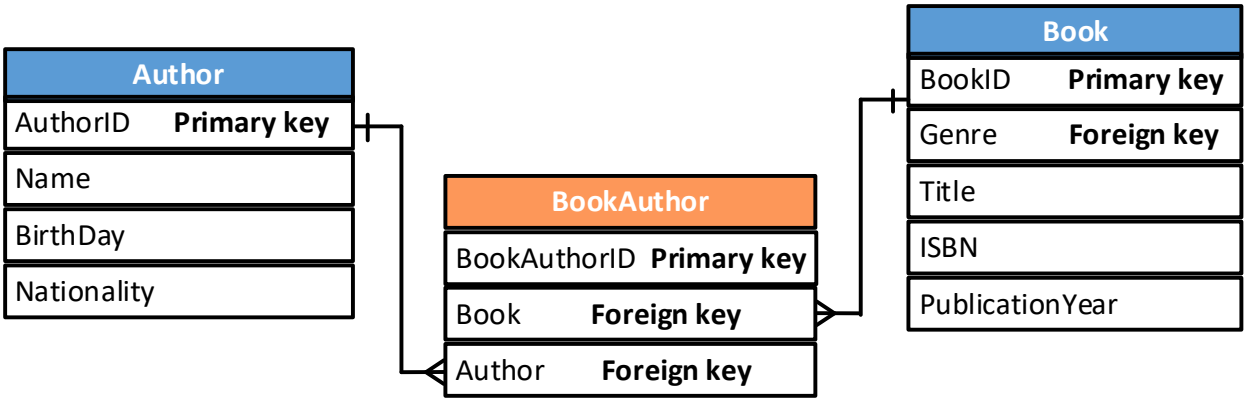

Рисунок 2 – Обновленная логическая модель

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.

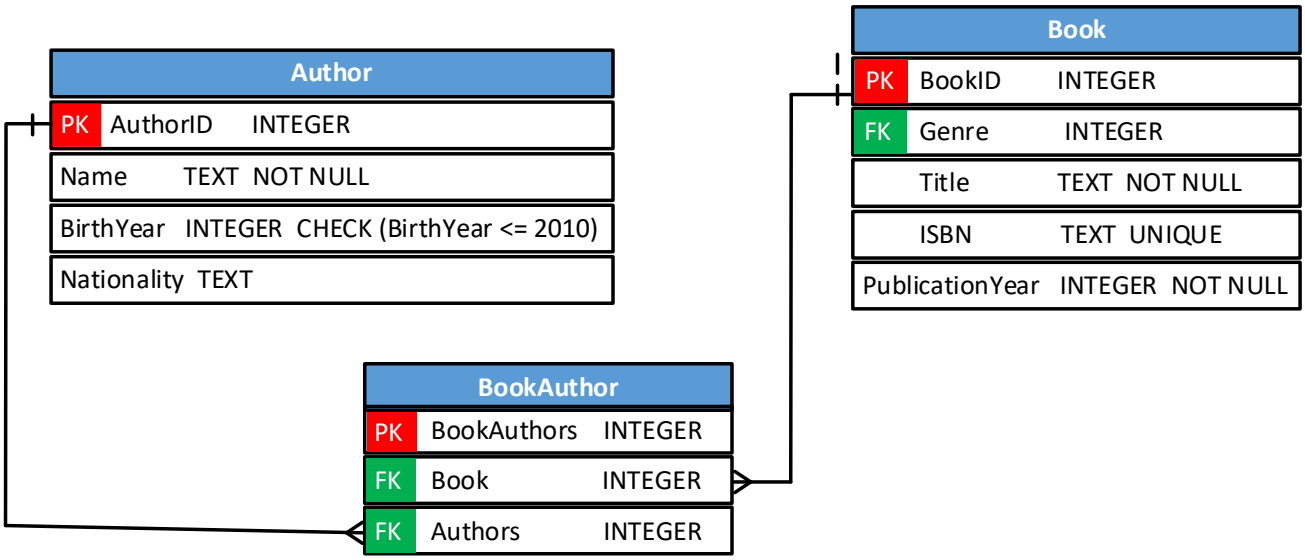

Рисунок 3 – Физическая модель

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

Рисунок 4 – Внутреннее хранение данных

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

Когда создается связующая таблица для реализации связи многие-ко-многим, принято давать ей названия, которые четко отражают сущности, между которыми устанавливается связь. Обычно название такой таблицы состоит из имен двух связанных сущностей. Например, BookAuthors, StudentCourses и т.д.). Примеры:

BookAuthors: Эта таблица связывает книги и авторов. Название отражает сущности – книги (Books) и авторы (Authors).

StudentCourses: В этом случае связь устанавливается между студентами (Students) и курсами (Courses), на которых они учат.

EmployeeDepartments: Связующая таблица для связи сотрудников (Employees) с департаментами (Departments), в которых они работают.

OrderProducts: Связующая таблица, которая используется для связи заказов (Orders) и продуктов (Products), которые включены в заказ.

CustomerOrders: В таблице хранятся данные о заказах (Orders), сделанных клиентами (Customers).

Часто используются такие варианты:

Со словом **To:** Customer**To**Orders, Product**To**Categories).
Со словом **Relations**: CustomerOrder**Relations**
Со словом **Link:** EmployeeDepartment**Link**

Связь с атрибутами

Когда между двумя сущностями существует связь с дополнительными атрибутами, то такая связь требует использования промежуточной таблицы, которая хранит не только ссылки на внешние ключи этих сущностей, но и дополнительные атрибуты, которые описывают саму связь.

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.

Вот как это может быть реализовано:

Новая таблица Reader для хранения сведений о читателях, которые берут книги в библиотеке. Атрибуты: ReaderID, Name, Email и т. д.

Связующая таблица: Rental **(**Аренда**)**: Это таблица для связи между Читателем и Экземпляром книги, которая также хранит атрибуты, описывающие саму аренду (например, дата взятия и дата возврата).

- Атрибуты: `ReaderID` (внешний ключ на таблицу Reader), `BookCopyID` (внешний ключ на таблицу BookCopy), `DateRented` (дата взятия книги), `DateReturned` (дата возврата книги).

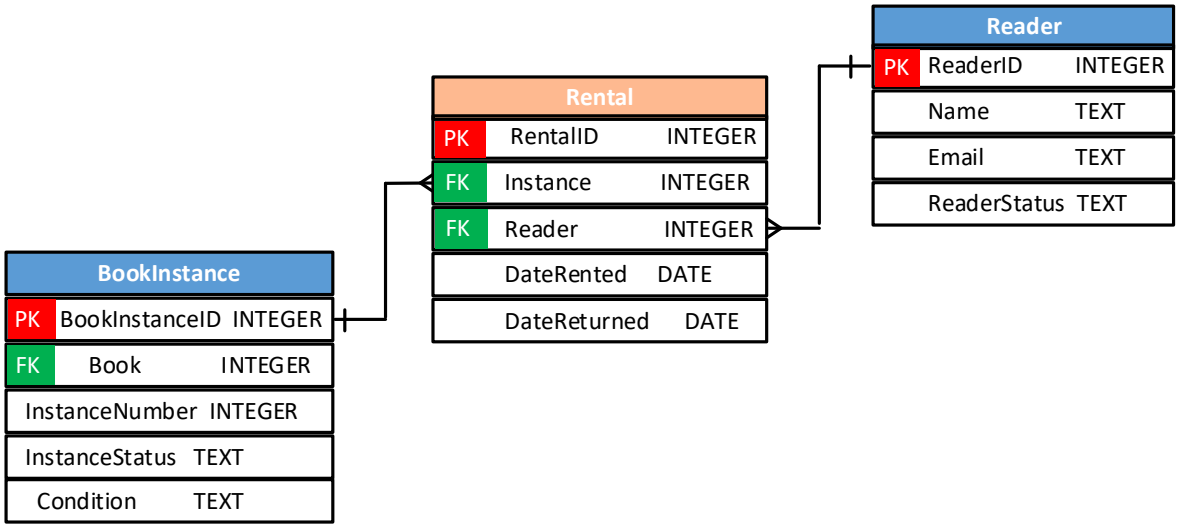

Рисунок 5 – Добавление связующей таблицы с дополнительными атрибутами

Пример структуры таблиц:

1. **Reader**

ReaderID	Name	Email
1	Иванов Иван	ivanov@mail.ru
2	Петров Петр	petrov@mail.ru

2. **BookInstance**

BookInstanceID	BookID	Status
1	101	Доступна
2	102	Арендована

3. **Rental (Аренда)**

ReaderID	BookInstanceID	DateRented	DateReturned
1	1	2025-03-01	2025-03-15
2	2	2025-03-05	NULL

- ReaderID и BookInstanceID – это внешние ключи, которые ссылаются на таблицы Reader и BookCopy соответственно.
- DateRented – атрибут, который хранит дату, когда книга была взята.
- DateReturned – атрибут, который хранит дату, когда книга была возвращена. Если книга еще не возвращена, это поле может быть пустым (NULL).

В данном случае, связь между Читателем и Экземпляром книги не является простой – она имеет дополнительные атрибуты, описывающие саму аренду (дата взятия и дата возврата). Поэтому для правильного моделирования связи, нам нужно использовать промежуточную таблицу (Rental), которая будет содержать не только внешние ключи, но и дополнительные данные, относящиеся к самой связи.

Преимущества такого подхода:

– Можно хранить подробную информацию о каждой аренде (дата взятия, дата возврата, возможно, статус книги и т. д.).
– Если связь между сущностями изменяется (например, возвращение книги позже или продление аренды), можно легко обновить записи в связующей таблице, не затрагивая другие сущности.

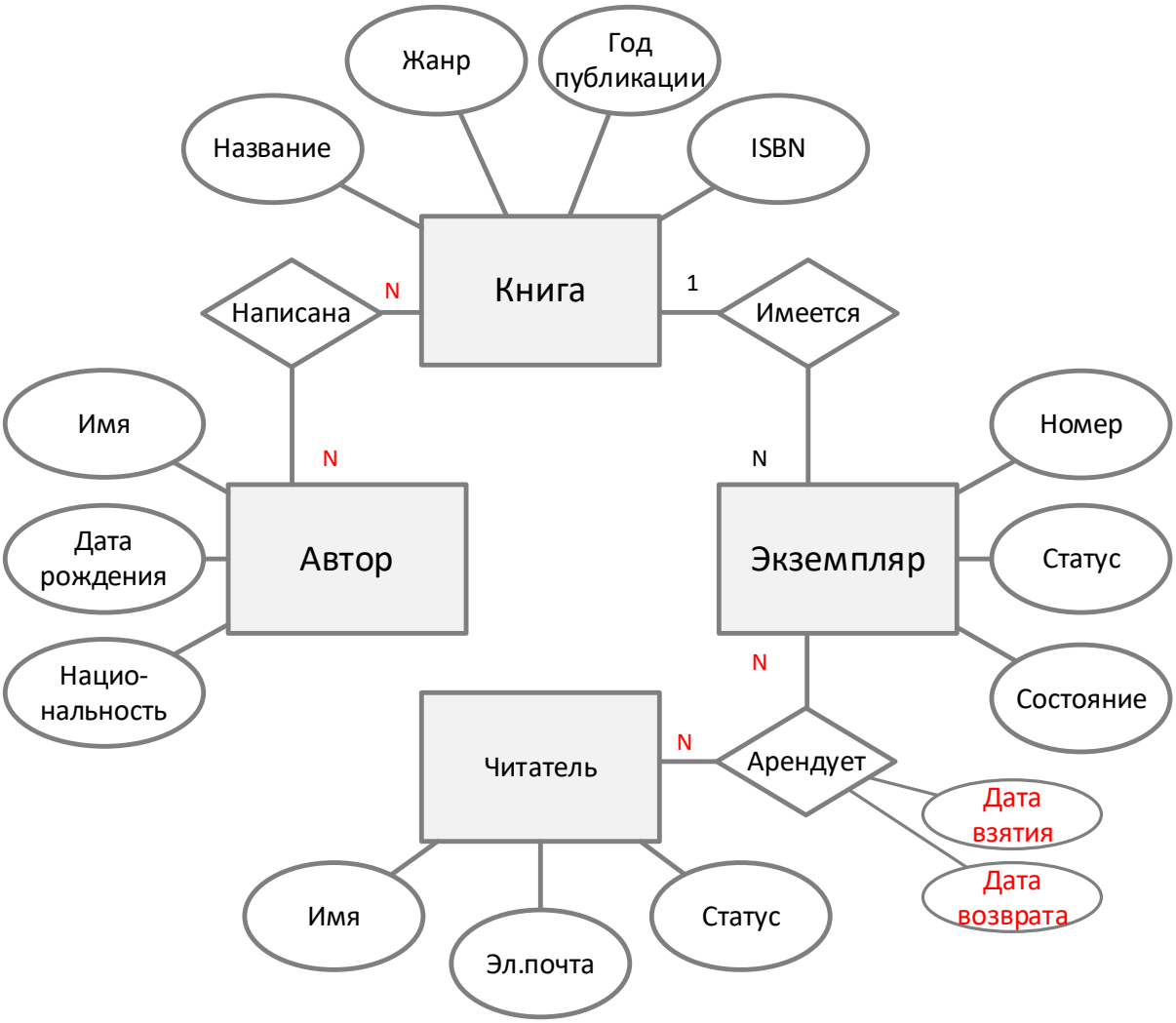

Рисунок 6 – Итоговая концептуальная модель

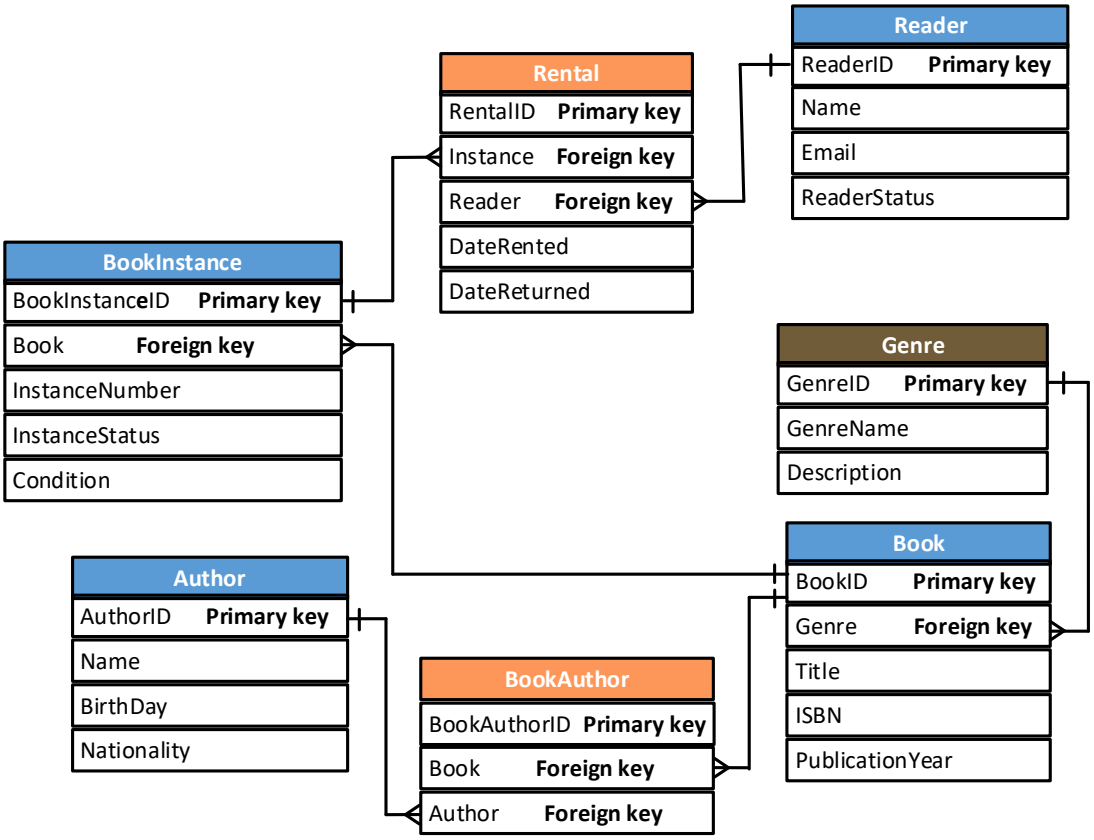

Рисунок 7 – Итоговая логическая модель

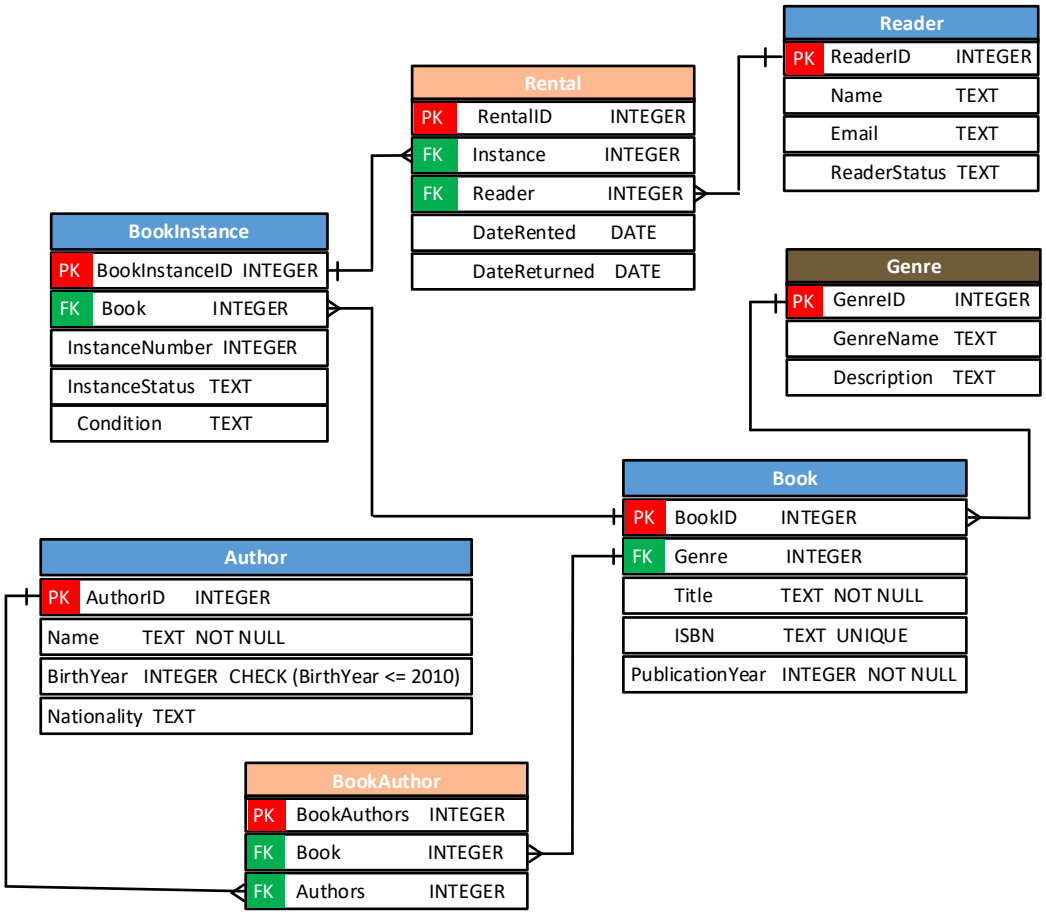

Рисунок 8 – Итоговая физическая модель

Нам нужно вывести информацию о книгах, которые были выданы читателю, с датами выдачи и возврата. Если книга не была возвращена, то значение `DateReturned` будет равно `NULL`.

Вот пример запроса:

```
reader_name = "Иванов"
SELECT b.Title, bi.InstanceNumber, r.DateRented, r.DateReturned
FROM Rental r
JOIN BookInstance bi ON r.BookInstanceID = bi.BookInstanceID
JOIN Book b ON bi.Book = b.BookID
JOIN Reader reader ON r.ReaderID = reader.ReaderID
WHERE reader.Name = reader_name;
```

JOIN-ы связыва.т таблицы:

`Rental` связывается с `BookInstance`, чтобы найти экземпляр книги.

`BookInstance` связывается с `Book`, чтобы получить информацию о книге.

`Reader` связывается с `Rental`, чтобы найти книги, выданные конкретному читателю.

WHERE фильтрует по имени читателя, используя переменную `reader_name`.

В выводе будет показана информация о названии книги (`Title`), номер экземпляра книги (`InstanceNumber`), а также дата выдачи (`DateRented`) и дата возврата (`DateReturned`).

Пример результата:

Title	InstanceNumber	DateRented	DateReturned
"Война и мир"	15	2025-02-01	NULL
"Преступление и наказание"	2	2025-02-10	2025-02-20

- Читатель взял 15-й экземпляр книги «Война и мир» 1 февраля 2025 года, но не вернул её (поэтому `DateReturned` равно `NULL`).
- Читатель вернул 2-й экземпляр книги «Преступление и наказание» 20 февраля 2025 года.

Этот запрос будет отображать все книги, которые были выданы читателю, с датами их выдачи и возврата.

Связь трех и более сущностей

Связующие таблицы могут использоваться не только для связи двух сущностей, но и для более сложных случаев, когда требуется связать три и более сущностей. Такие таблицы помогают моделировать более сложные отношения в базе данных. Вот несколько примеров, когда связующая таблица может включать три сущности:

Предположим, у нас есть три сущности: Читатели**,** Книги и Библиотеки. Читатель может брать книги из разных библиотек, и каждая библиотека может содержать несколько книг. В этом случае связывающая таблица может включать атрибуты, такие как дата выдачи и дата возврата.

```
CREATE TABLE IF NOT EXISTS Rental (
    RentalID INTEGER PRIMARY KEY,
    ReaderID INTEGER NOT NULL,
    BookID INTEGER NOT NULL,
    LibraryID INTEGER NOT NULL,
    DateRented DATE,
    DateReturned DATE,
    FOREIGN KEY (ReaderID) REFERENCES Reader(ReaderID),
    FOREIGN KEY (BookID) REFERENCES Book(BookID),
    FOREIGN KEY (LibraryID) REFERENCES Library(LibraryID)
);
```

В этой таблице три сущности: Читатель (Reader)**,** Книга (Book) и Библиотека (Library)**.** Каждый экземпляр в этой таблице показывает, какой читатель взял какую книгу из какой библиотеки, а также включает дополнительные атрибуты, такие как дата выдачи и дата возврата.

Предположим, что у нас есть три сущности: Студенты**,** Курсы **и** Преподаватели**.** Студенты могут посещать несколько курсов, а преподаватели могут вести несколько курсов. Можно использовать связующую таблицу для отображения, кто преподает какой курс студентам, а также добавить дополнительные атрибуты, такие как дата начала курса или оценка.

Пример таблицы:

```
CREATE TABLE IF NOT EXISTS StudentCourseInstructor (
    StudentID INTEGER NOT NULL,
    CourseID INTEGER NOT NULL,
    InstructorID INTEGER NOT NULL,
    EnrollmentDate DATE,
    Grade TEXT,
    PRIMARY KEY (StudentID, CourseID, InstructorID),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
    FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)
);
```

В этой таблице: StudentID указывает на студента, CourseID указывает на курс, InstructorID указывает на преподавателя, EnrollmentDate и Grade являются дополнительными атрибутами, которые показывают дату зачисления студента на курс и его оценку.

Для моделирования «Сделки» можно использовать три сущности: Клиенты, Продавцы и Продукты. Клиенты могут делать покупки через продавцов, а каждый продукт может быть продан множеству клиентов. Для моделирования такой связи можно создать связующую таблицу с дополнительными атрибутами, например, количеством купленных продуктов и ценой.

Пример таблицы:

```
CREATE TABLE IF NOT EXISTS Transaction (
    TransactionID INTEGER PRIMARY KEY,
    ClientID INTEGER NOT NULL,
    SellerID INTEGER NOT NULL,
    ProductID INTEGER NOT NULL,
    Quantity INTEGER NOT NULL,
    Price REAL NOT NULL,
    Date DATE,
    FOREIGN KEY (ClientID) REFERENCES Client(ClientID),
    FOREIGN KEY (SellerID) REFERENCES Seller(SellerID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

В этой таблице: ClientID указывает на клиента, SellerID указывает на продавца, ProductID указывает на продукт, Quantity и Price являются дополнительными атрибутами, которые показывают количество продукта в сделке и его цену.

Запросы с использованием связующих таблиц, особенно с участием нескольких сущностей, широко распространены в различных приложениях, от баз данных электронной коммерции и управления проектами до образовательных систем и социальных сетей. Их использование зависит от сложности модели данных и количества взаимосвязанных сущностей. В простых приложениях такие запросы могут встречаться реже, но в более сложных системах они становятся неотъемлемой частью архитектуры.

Выводы

Основные типы связей и их особенности:

- **Один к одному (1:1)**: Связь, где каждой записи в одной таблице соответствует только одна запись в другой.
- **Один ко многим (1:M)**: Связь, где одной записи в одной таблице может соответствовать несколько записей в другой таблице.
- **Многие ко многим (M:N)**: Связь, где несколько записей в одной таблице могут быть связаны с несколькими записями в другой, и для этого используется промежуточная таблица.

2. Практическое применение

- **Один к одному** используется в ситуациях, когда два объекта тесно связаны и имеют только одну пару данных, например, клиент и его постоянный адрес.
- **Один ко многим** применим в случаях, когда один объект может иметь несколько зависимых объектов, например, один заказ и несколько продуктов.
- **Многие ко многим** полезен, когда объекты обеих сущностей могут иметь взаимные связи, например, студенты и курсы, где один студент может записаться на несколько курсов, а курс может включать нескольких студентов.

3. Связующие таблицы и их роль

Связующие таблицы необходимы в случае связей многие ко многим, а также когда может понадобиться дополнительная информация о связи (например, дата аренды, количество и т. д.).

4. Ссылочная целостность

Внешние ключи обеспечивают целостность данных и поддерживают связи между сущностями. Они помогают избежать ошибок, например, удаление данных без учета связанных записей.

Не забывайте, что понимание связей важно не только для баз данных, но и для того, чтобы эффективно управлять информацией в различных системах и приложениях.