Московский государственный технический университет
имени Н.Э. Баумана
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ*
КАФЕДРА *ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ*

Курсовая работа

«Приложение хранения библиографических ссылок».

По курсу «Базы данных».

Выполнил: Лапатин В.В.
Проверил: Дубанов А.В.

Москва 2018

Содержание

Введение 2
Разработка интерфейса 3
Программные интерфейсы 3
Графические интерфейсы 4
Интерфейсы командной строки 5
Постановка задачи 6
Разработка программного интерфейса 6
Разработка интерфейса командной строки 7
Разработка графического интерфейса 8
Проектирование базы данных 11
Заключение 17
Список используемой литературы 18

Введение

Разработка интерфейса

Для любой программы интерфейс является одной из наиболее важных составляющих. Ведь именно он определяет, как приложение будет взаимодействовать с другими программами и своими пользователями. Таким образом, можно ввести следующую классификацию интерфейсов:

1. программные,
2. графические,
3. интерфейсы командной строки.

Несмотря на то, что эти интерфейсы имеют между собой мало общего, к ним предъявляется ряд общих требований:

1. функциональность – интерфейс должен отвечать всем требованиям пользователя и соответствовать его задачам,
2. логичность – интерфейс должен быть логичным и запоминающимся, чтобы взаимодействие пользователя с программой было как можно более простым и удобным,
3. защищенность – интерфейс должен быть спроектирован таким образом, чтобы у пользователя не было возможности совершить ошибку.

После определения общих требований к любому интерфейсу, стоит рассмотреть перечисленные выше интерфейсы отдельно.

Программные интерфейсы

К программным интерфейсам можно причислить любой интерфейс, который предназначен для использования разрабатываемой программы внешними приложениями. Обычно, в зависимости от используемых технологий, это набор классов, функций или методов, которые используются внешними программами. В случае данного приложения в качестве интерфейса взаимодействия было решено использовать веб-технологии. Это означает, что приложение будет взаимодействовать с любыми своими пользователями через протокол HTTP. Таким образом, программный интерфейс приложения будет представлять из себя набор HTTP-методов.

В протоколе HTTP есть несколько видов методов, из которых приложением будут использоваться следующие:

1. GET – это методы, которые запрашивают данные и не предназначенные

для их записи,

2. POST – это методы, которые используются и для записи данных, и для их получения.

Говоря об HTTP интерфейсах, стоит отметить, что есть несколько различных подходов к проектированию подобных интерфейсов. Одним из наиболее общепринятых подходов является так называемый REST. Это можно перевести как Representational State Transfer [1]. Данная архитектура предлагает наложить на приложение ряд следующих ограничений:

1. модель клиент-сервер – означает, что вся логика должна выполняться на удаленном сервере, а клиентское приложение должно исключительно предоставлять и получать данные,
2. отсутствие состояния – сервер получает из запроса всю необходимую информацию и не хранит никакую информацию о сессии клиентов,
3. кэширование – сервер сохраняет наиболее частые ответы, что позволяет не выполнять лишние запросы к базе данных и соответствующие вычисления, а сразу вернуть результат,
4. единообразие интерфейса,
5. слои – сокрытие основного сервера за промежуточными. Например, без каких-либо изменений для пользователя можно внедрить между ним и сервером промежуточный сервер, который предназначен для хранения и отдачи хэшированных данных. В случае отсутствия этих данных хэширующий сервер перенаправляет запрос исходному серверу.

Такой подход позволяет добиться лучшей производительности за счет отсутствия состояний между вызовами и кэширования, а архитектура становится более расширяемой из-за требований единообразия и использования слоев.

Графические интерфейсы

Графические интерфейсы – это наиболее востребованные интерфейсы в современном мире, потому что пользователям намного более удобно пользоваться интерфейсами, у которых среди средств донесения информации есть не только текст.

И, пожалуй, самый популярный вид графических приложений – это веб-сайты. Они получили такую популярность из-за своей универсальности:

пользователь может открыть приложение на любом устройстве, на котором есть доступ в интернет и веб-браузер, в то время как любое другое приложение потребует установки на устройство пользователя. С другой стороны, веб-сайты очень удобны для разработчиков. Намного более выгодно разработать одно веб-приложение, чем создавать и поддерживать несколько программ для разных платформ.

Обычно разработку веб-приложения можно разделить на две части: «фронтенд» и «бэкенд».

«Фронтенд» – это та часть приложения, которая исполняется в браузере пользователя. Для написания веб-сайтов используются язык разметки HTML, язык описания стилей CSS и интерпретируемый язык программирования JavaScript.

Опишем роль каждого языка в веб-приложении:

1. HTML – это каркас всего приложения, который определяет, какие элементы будут использоваться и где они будут располагаться,
2. CSS определяет, как будут выглядеть элементы приложения. Сюда входят, например, внешний вид кнопок или вид шрифта отображаемого текста,
3. JavaScript – это интерпретируемый язык программирования, основная область применения которого заключается в придании интерактивности веб-страницам. Например, динамическую загрузку данных в таблицу можно реализовать только через JavaScript.

«Бэкендом» обычно называют ту часть приложения, которая стоит за «фронтендом». В основном это та программа, которая предоставляет данные, выполняет их хранение и агрегацию. Другими словами, «бэкенд» реализует всю «бизнес-логику», в то время как «фронтент» нужен для предоставления пользователям доступа к приложению. Более подробно об этом будет рассказано далее.

Интерфейсы командной строки

Интерфейсы командной строки, в отличии от графических и программных интерфейсов, представляют наименее популярную группу приложений. Но, несмотря на это, они являются незаменимыми. Главное преимущество приложений командной строки заключается в том, что ими могут пользоваться в равной степени эффективно человек, и программа. Таким

образом, интерфейсы командной строки являются сочетанием программных и графических интерфейсов: с одной стороны ими с определенной степенью удобства может пользоваться человек, а с другой стороны без каких-либо трудностей они могут использоваться для взаимодействия между приложениями. Также стоит отметить, что приложениям командной строки не нужен какой-либо графический интерфейс, только окно терминала.

Для того, чтобы заниматься непосредственным проектированием интерфейсов, необходимо понять задачу, которые они будут решать.

Постановка задачи

Необходимо разработать приложение, которое выполняет генерацию списков литературы для учебных курсов в формате BibTeX. Для этого приложение должно иметь следующие функции:

1. Поиск информации о книгах;
2. Хранение и модификация информации о книгах;
3. Создание и модификация данных об учебных курсах;
4. Генерация списка литературы.

Разработка программного интерфейса

При рассмотрении программных интерфейсов была описана архитертура REST. Согласно этой архитектуре каждый метод должен принимать в себя всю необходимую информацию для выполнения запроса, а все методы должны быть унифицированы. Таким образом, для каждого вида данных был реализован следующий набор методов:

- add%Имя таблицы% – POST запрос, содержащий данные для добавления в теле,
- get%Имя таблицы% – GET запрос, получающий данные от приложения,
- prototype%Имя таблицы% – GET запрос, получающий прототип JSON-файла для искомой таблицы.

Также были реализованы две дополнительных команды: report и migrate.

Первая получает данные, необходимые для однозначной идентификации требуемого списка литературы и генерирует для этого списка литературы файл в формате BibTeX.

Команда migrate выполняет копирование списка литературы с одного года на другой. Это необходимо для того, чтобы облегчить работу пользователя, ведь обычно учебные программы слабо меняются от года к году.

Разработка интерфейса командной строки

Как уже было сказано выше, для интерфейсов командной строки крайне важна автоматизируемость. Это делает невозможным использование так называемого интерактивного ввода. Другими словами, любое действие пользователя должно совершаться за один вызов приложения. В результате было решено использовать следующий подход к проектированию интерфейса: для каждого вида данных, с которыми будет работать приложение, будет реализована собственная команда. И для каждой такой команды будет реализован набор подкоманд, выполняющих необходимые действия. В общем случае команду приложения командной строки можно описать в следующем виде:

- %Имя таблицы% prototype – данный метод добавляет новый файл в файловую систему, который содержит в себе прототип для нужного вида данных. После чего пользователь должен заполнить этот прототип информацией, которую хочет добавить,
- %Имя таблицы% add – сохраняет данные в приложение,
- %Имя таблицы% get – позволяет получить данные из приложения, можно конфигурировать флагами.

Также отличительной чертой хорошего приложения командной строки является грамотно оформленная справка. Рисунок 1 показывает результат выполнения команды help.

Также для каждой отдельной команды реализован отдельный флаг –help, который показывает справку для конкретной команды, что можно видеть на рисунке 2.

//Описать дополнительные команды вроде migrate и generate; сделать скриншоты работы программы

```
~/C/C/B/s/w/C/bin ››› ./cli -h
Usage:
  cli [flags]
  cli [command]

Available Commands:
  book             Команда для работы с книгами
  course           Команда для работы с курсами
  department       Команда для работы с кафедрами
  help             Help about any command
  lecturer         Команда для работы с лекторами
  literature       Команда для работы с литературой
  literatureList   Команда для работы со списками литературы
  migrate          Выполнить копирование списка литературы с одного учебного года на другой.
  migratePrototype Получить заготовку JSON для миграции в файл, определяемый флагом.
  report           Выполнить создание списка литературы.
  search           Выполнить поиск книг в онлайн-источниках.

Flags:
  -h, --help   help for cli

Use "cli [command] --help" for more information about a command.
```

Рисунок 1: Пример работы команды help.

```
~/C/C/B/s/w/C/bin ››› ./cli book help
Команда для работы с книгами

Usage:
  cli book [command]

Available Commands:
  add         Отправить книгу на сервер из файла, заданного флагом.
  get         Получить список книг в формате BibTeX, сохраненных в базе данных.
  prototype   Получить заготовку JSON для книги в файл, определяемый флагом.

Flags:
  -h, --help   help for book

Use "cli book [command] --help" for more information about a command.
```

Рисунок 2: Пример работы флага help для команды book

Разработка графического интерфейса

Как уже было сказано выше, графический интерфейс реализован в виде веб-сайта, написанного на языке TypeScript с использованием фреймворка Bootstrap. Выбор фреймворка обусловлен тем, что он предоставляет продвинутый набор стилей и HTML-элементов, что позволяет легко и быстро настроить внешний вид веб-приложения. В связи с тем, что единственный современный и поддерживаемый язык для написания веб-сайтов - это JavaScript, то из альтернатив ему можно использовать только языки, компилируемые в JavaScript. В частности, к таким языкам можно отнести язык TypeScript компании Microsoft и язык Kotlin, разрабатываемый компанией JetBrains,

который имеет отдельный плагин для компиляции JS-код. К сожалению, Kotlin является очень молодым языком, который находится в активной разработке, так что было решено использовать TypeScript. Преимуществом данного языка является то, что он является надмножеством JavaScript. Это означает, что любой JavaScript код является абсолютно корректным. Также важно отметить, что TypeScript поддерживает важные аспекты объектно-ориентированого программирования: классы, инкапсуляцию(уточнить насчет остального). Но главным преимуществом этого языка является его строгая типизированность. Это облегчает разработку и позволяет писать более читаемый код по сравнению с JavaScript.

На рисунках 3 и 4 видно, приложение состоит из четырех логический частей.

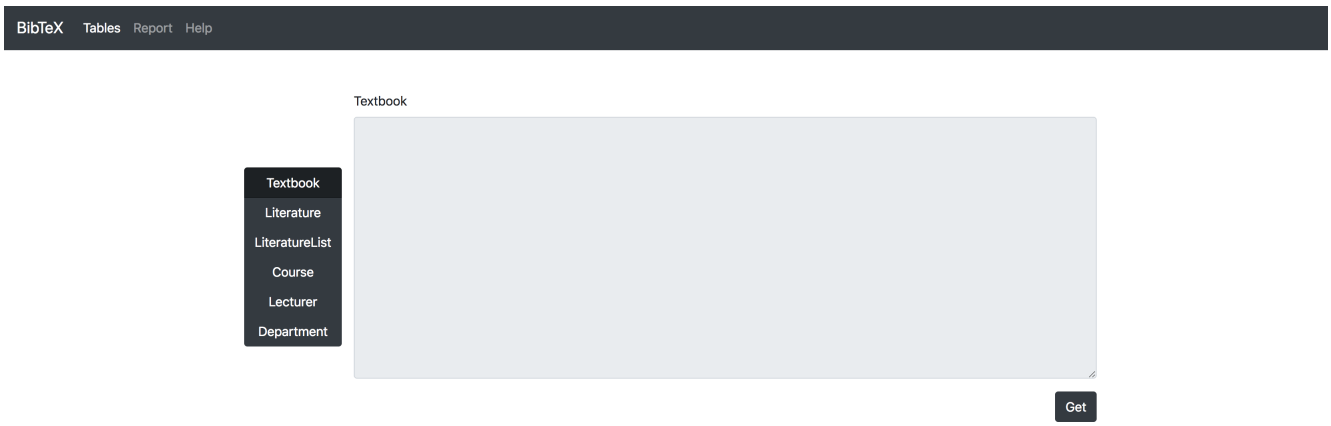

Рисунок 3: Верхняя часть веб-сайта

Первая - шапка сайта. Она содержит название приложения и ссылки на три страницы приложения: основную рабочую область, страницу генерации отчетов и страницу со справкой.

Вторая логическая часть - это большая недоступная для ввода область TextArea вместе с набором кнопок, отвечающих за переключение используемой таблицы. Эта часть нужна для получения данных, которые уже содержатся в базе данных для более удобного добавления новых записей в приложение.

Третья часть – доступная для ввода область TextArea, в которую автоматически загружаются JSON прототипы для текущего вида входных данных.

Заключительная область необходима для поиска необходимых книг в сервисе Google Books.

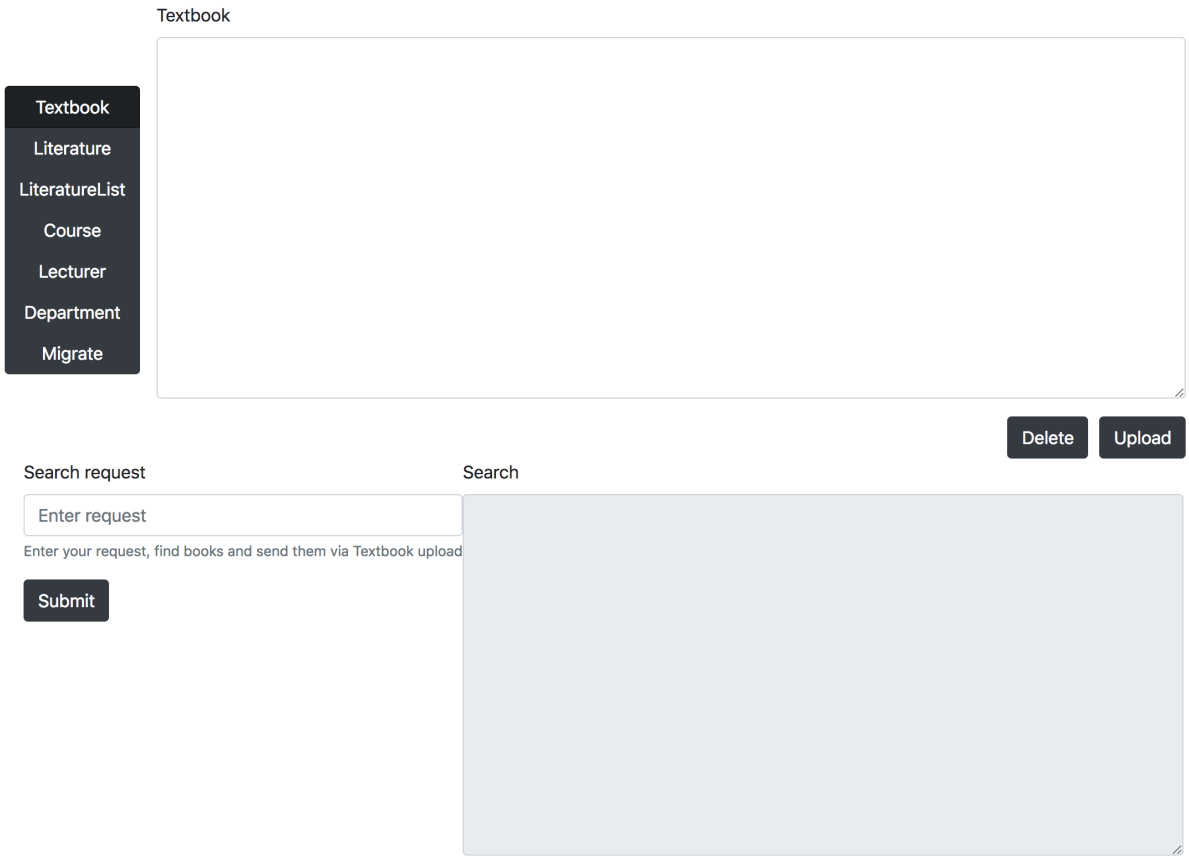

Рисунок 4: Нижняя часть веб-сайта

Проектирование базы данных

База данных является неотъемлемой частью любого приложения, которое выполняет хранение и обеспечивает работу с информацией. Так что выбор правильной технологии хранения данных является чрезвычайно важной задачей.

На текущий момент существует две различные ветви развития систем управления базами данных(СУБД): релационная и нереляционная.

Отличительной чертой реляционных баз данных является понятие отношения или таблицы. Каждая сущность, хранимая в базе данных, должна представлять собой строку таблицы со строго заданным типизированным набором столбцов. Также реляционные СУБД гарантируют выполнение так называемых свойств ACID к транзакционной системе, где под транзакцией понимается последовательность команд, представляющая логическую единицу работы с данными. Опишем свойства ACID:

1. атомарность – транзация либо будет выполнена целиком, либо не выполнена совсем,
2. согласованность – после выполнения транзакции в базе данных находятся корректные значения,
3. изолированность – на транзакцию не могут оказать влияния другие транзакции, выполняемые параллельно,
4. устойчивость – если транзакция была завершена, то даже при сбое системы изменения будут зафиксированы.

Данные свойства накладывают довольно серьезные ограничения на производительность, что послужило поводом для появления нереляционных СУБД. Перед ними стояло требование обеспечить хранение данных для высоконагруженных приложений. В противовес свойствам ACID, нереляционные базы данных гарантируют выполнение свойств BASE (Источник: What NoSQL is and what it is not.):

1. доступность – каждый запрос будет выполнен,
2. гибкость – состояние системы может меняться со временем даже без ввода новых данных,
3. согласованность в конечном счете – данные могут быть несогласованны в некоторые моменты времени, но в итоге приходят в согласованное состояние.

//Проверить точность формулировок в первоисточнике

Стоит отметить, что существует множество видов нереляционных баз данных, перечислим основные:

1. документоориентированные,
2. графовые,
3. ключ-значение.

Графовые базы данных не дадут особого выигрыша из-за относительной простоты хранимых данных.

В произведенном далее сравнении все замечания, относящиеся к документоориентированным базам данных в равной степени относятся и к базам данных вида ключ-значения, так что ниже речь будет идти только о реляционных и документоориентированных СУБД.

Для того, чтобы избежать голословности, сравнение будет проводиться на примере реляционной СУБД PostreSQL и нереляционной MongoDB.

В основе документоориентированной базы данных лежит понятие документа. В общем случае в качестве формата может быть множество различных стандартов: JSON, XML, YAML и так далее. В случае MongoDB для хранения используется BSON – надмножество JSON. В данной СУБД документы группируются в так называемые коллекции. В отличии от реляционной модели, коллекция не имеет какой-либо строгой структуры. То есть в ней могут храниться абсолютно разные документы.

Данных подход имеет ряд своих преимуществ и недостатков. Перечислим положительные стороны:

1. гибкость – так как коллекция не ограничена структурой хранимых данных, информация в ней может несколько отличаться от записи к записи. Это может полезно, если данные имеют в целом общий смысл, но в некоторых документах могут присутствовать какие-то особые поля,
2. производительность – так как не происходит никаких проверок, запись и чтение работают существенно быстрее, чем в реляционном подходе,

Недостатки и ограничения:

1. Подобная организация плохо подходит для создание ссылок между объектами по, например, первичному ключу. Это связано с описанным выше отсутствием проверок. Канонический способ хранения данных – это использование вложенных документов. Но этот способ применим не везде

из-за ограничение на размер документа в 16 МБ.

2. Как было описано в пункте выше, довольно сложно производить нормализацию данных, потому что все проверки необходимо производить не на уровне СУБД, а на уровне приложения.

После анализа плюсов и минусов NoSQL подхода стоит определить, какие данные будут храниться и какие требования должны соблюдаться.

В первую очередь стоит отметить, что все данные имеют абсолютно строгую структуру. Если говорить о хранимых книгах, то для них есть набор полей, определяемый стандартном BibTeX, который обязан быть у каждой записи. Остальные данные, такие как списки литературы и учебные курсы, также не имеют никакой вариативности. Таким образом, гибкость NoSQL подхода только добавит сложностей в связи с необходимостью ручной реализации множества проверок.

А во вторую очередь нужно сказать, что конечная цель приложения – генерировать списки литературы для учебных курсов. И весьма логичным требованием будет то, что в любой момент времени приложение должно генерировать корректные отчеты. Таким образом, для данной задачи больше подходят требования ACID, чем BASE.

Как видно из рассуждений выше, несмотря на все свои преимущества, для поставленной задачи больше подходит реляционная модель.

Следующий этап проектирования – это формализация таблиц базы данных. Перечислим таблицы и их их цели:

- Textbook – таблица с книгами в формате BibTeX. Обладает несколькими UNIQUE столбцами:
  - ident – идентификатор книги, должен быть уникальным, так как именно он используется для идентификации библиографической ссылки в стандарте LaTeX;
  - isbn – уникальный для каждой книги ключ, позволяет исключить дублирование;
- LiteratureList – таблица, хранящая списки литературы. Имеет UNIQUE ограничение на пару из ID курса, которому присвоем список и года этого курса. Это необходимо для идентификации списка литературы;
- Literature – так называемая таблица пересечения, необходимая для создания связи многие-ко-многим между таблицами LiteratureList и Text-

book;

- Course – задает учебный курс. Стоит отметить, что один учебный курс может иметь несколько списков литературы за разные года. Имеет ссылки на кафедру, к которой привязан и лектора, читающего курс. Обладает UNIQUE ограничением на тройку из названия, кафедры и семестра, в котором читается курс;
- Department – задает кафедру, имеет UNIQUE поле title, характеризующее название кафедры;
- Lecturer – хранит всех лекторов. Однозначно идентифицируется именем и датой рождения.

Все таблицы обладают суррогатными первичными ключами.

Стоит также отметить два важных решения, которые были приняты для всей базы данных. Во-первых, каждая запись имеет поле timestamp – временную метку добавления или изменения записи. Это позволит иметь историю изменений. Также нужно уточнить, что данное поле имеет тип integer, что является более общим решением, чем хранение метки во внутреннем формате СУБД. Во-вторых, в базе данных будет отсутствовать возможность удаления записи. Для этого каждая запись имеет флаг isDeleted. При удалении пользователем записи она будет помечаться, как удаленная. У данного решения есть ряд преимуществ. Это позволяет обезопасить базу данных от ошибок пользователя и от потенциального взлома системы. И в том, и в другом случае никому не удастся нанести непоправимый ущерб данным. Причем блокировка будет осуществляться за счет того, что у учетной записи, через которую пользователь будет взаимодействовать с базой данных, не будет прав на удаление из базы данных. При этом все равно будет администратор, у которого данная возможность есть.

Диаграмма модели сущность-связь приведена в рисунке 5.

Стоит отметить отсутствие минимальных кардинальных связей на данной диаграмме. Дело в причинах, описанных выше и относящихся к роли поля isDeleted. Для упрощения конфигурации серверной части предполагается запрет любым пользователям, кроме администратора, какого-либо удаления. Удаление предполагается каскадное, реализованное на серверной части с помощью обновления флага isDeleted. Таким образом, везде неявно предполагается минимальная кардинальная связь единица. В качестве

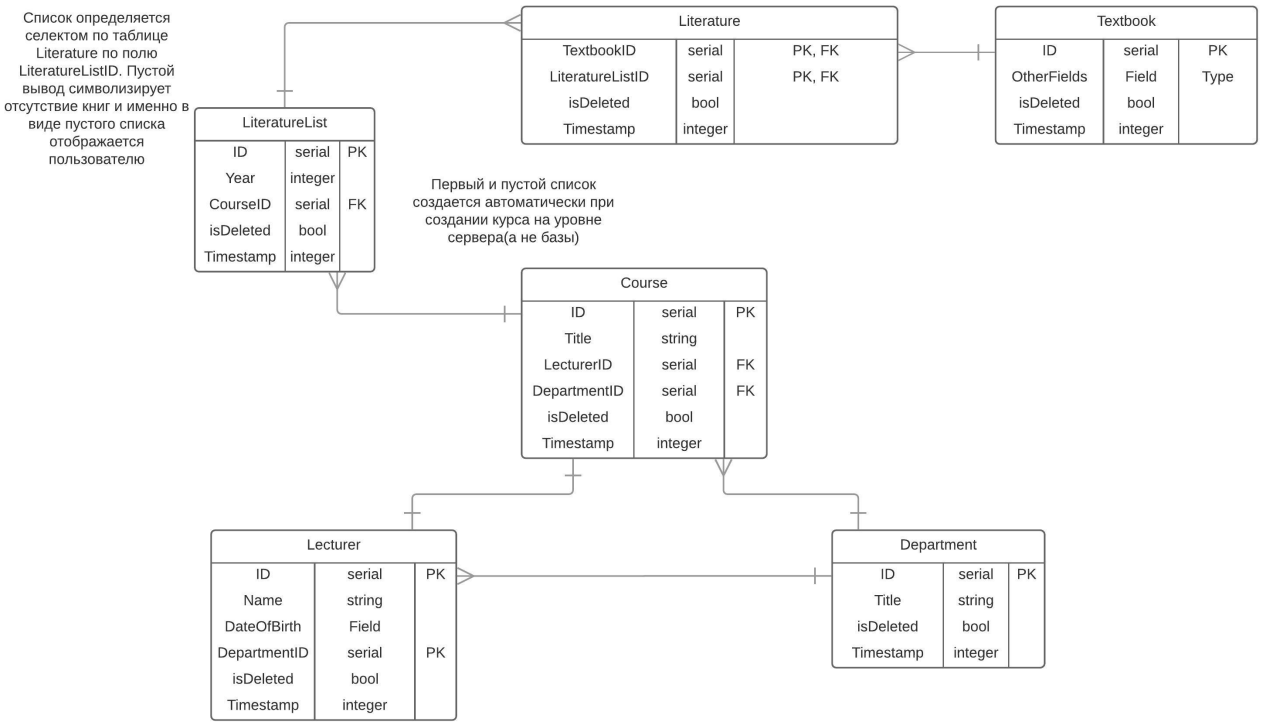

Рисунок 5: Диаграмма модели сущность-связь

альтернативного решения можно было бы использовать триггеры, которые вместо удаления будут производить обнолвение поля. Но в таком случае пострадает переносимость. Дело в том, что не в каждой SQL базе данных присутствуют гибкие настройки прав и будет уже не так легко заменить базу данных. На уровне серверной части работа с базой данной реализована через стандартную библиотеку для работы с SQL, что дает возможность будущей миграции на любую SQL базу данных. Более подробно реализация будет описана в главе 3. В то время как предложенный подход не имеет такой завязки на настройку прав пользователей.

Выбор максимальных кардинальных связей основан исключительно на функциях таблиц, описанных выше и предполагаемой логики. Объясним выбор этих связей:

- Department-Lecturer – предполагается, что преподаватель может числиться только на одной кафедре, а у кафедры может быть много преподавателей,
- Department-Course – аналогично случаю Department-Lecturer,
- Lecturer-Course – обычно в университетских программах при наличии нескольких преподавателей в учебном курсе главным считается лектор, так как именно он определяет программу курса, а значит и его список литературы. Таким образом, для идентификации курса достаточно одного

преподавателя – лектора. С другой стороны, лектор может вести несколько учебных курсов.

- Course-LiteratureList – у каждого курса может быть несколько разных списков литературы за разные года, но у каждого списка только один курс,
- LiteratureList-Literature-Textbook – связь многие-ко-многим между LiteratureList и Textbook, так как каждый список содержит множество книг, а учебник может быть использован в нескольких курсах.

Заключение

Список использованной литературы

[1] Р.Т. Филдинг. *Representational State Transfer*. 2000.