

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ
по дисциплине «Разработка ПО информационных
систем»

Тема: Система управления библиотечными карточками

Студенты гр. 4303

Азаров А.И.

Туров В.В.

Шадринцев И.Д.

Преподаватель

Заславский М.М.

Санкт-Петербург

2017

ЗАДАНИЕ

Студенты

Азаров А.И.

Туров В.В.

Шадринцев И.Д.

Группа 4303

Тема проекта: Разработка приложения для управления библиотечными карточками.

Исходные данные:

Необходимо реализовать приложение для управления библиотечными карточками для трёх СУБД(MongoDB, Neo4j, Memcached) с возможностью переключения между ними.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Качественные требования к решению»

«Сценарий использования»

«Модель данных»

«Разработка приложения»

«Вывод»

«Приложение»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студенты гр. 4303

Азаров А.И.

Туров В.В.

Шадринцев И.Д.

Преподаватель

Заславский М.М.

АННОТАЦИЯ

В рамках данного курса предполагалось разработать какое-либо приложение в команде на одну из поставленных тем. Была выбрана тема создания приложения для управления библиотечными карточками для трёх СУБД(MongoDB^[1], Memcached^[2], Neo4j^[3]), так как это отличная возможность понять, какая СУБД лучше подходит для разработки такой системы. Во внимание будут приниматься такие аспекты как производительность и удобство разработки. Найти исходный код и всю дополнительную информацию можно по ссылке: https://github.com/moevm/nosql-2017-lib_card

ANNOTATION

In the course was planned to develop an application in the team for one of the themes. Topic related to management of library cards for three different DMS's (MongoDB, Memcached, Neo4j), because it's a excellent opportunity to understand with DMS is better for such system development. Attention will be paid to performance and ease of development. Source code and all the additional information can be found at: https://github.com/moevm/nosql-2017-lib_card

Оглавление

1.	Введение	7
2.	Качественные требования к решению	7
3.	Сценарии использования.....	7
4.	Модель данных.....	12
5.	Разработанное приложение.....	23
6.	Вывод	24
7.	Приложения.....	24
8.	Используемая литература.....	24

1. Введение

Цель работы – создать высокопроизводительное и удобное решение для хранения библиотечных карточек.

Было решено разработать веб-приложение, которое позволит хранить в электронном виде библиотечные карточки, при этом позволяющее удобно с ними взаимодействовать.

2. Качественные требования к решению

Требуется разработать приложение с использованием трёх СУБД – MongoDB, Memcached и Neo4j, с возможностью переключения между ними в режиме реального времени.

3. Сценарии использования

Макеты UI

1. Экран выдачи книги (Рис. 1).

Система управления библиотечными карточками

Идентификатор - 123

Книга в наличии

Список выдачи

Название:

Автор:

Год выпуска:

ФИО читателя

Дата выдачи

Выдать книгу

Сохранить

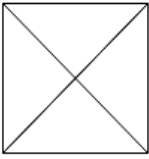
Удалить

Подтвердить

Рисунок 1. Экран выдачи книги.

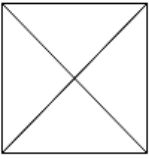
2. Экран со списком книг и открытой панелью добавления книги (Рис. 2).

Система управления библиотечными карточками



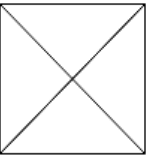
Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id



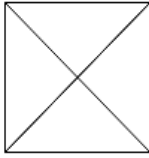
Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id



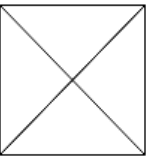
Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id



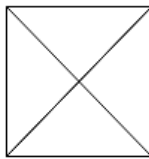
Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id



Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id



Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id

+

🔍

⚙️

Название книги

Автор книги

Год выпуска

Загрузить картинку

[путь к картинке]

Рисунок 2. Экран со списком книг и открытой панелью добавления книги.

3. Экран со списком книг и открытой панелью поиска книг (Рис. 3).

Система управления библиотечными карточками

Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id

Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id

Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id

Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id

Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id

Название:
Евгений Онегин
Год выпуска
2011
Автор
Пушкин А.С.

id

+

🔍

⚙️

По автору

По названию

Год выпуска

Идентификатор

☒ Любые

☐ Только свободные

☐ Только занятые

Найти

Рисунок 3. Экран со списком книг и открытой панелью поиска книг.

4. Экран со списком книг и открытой панелью переключения БД (Рис. 4).

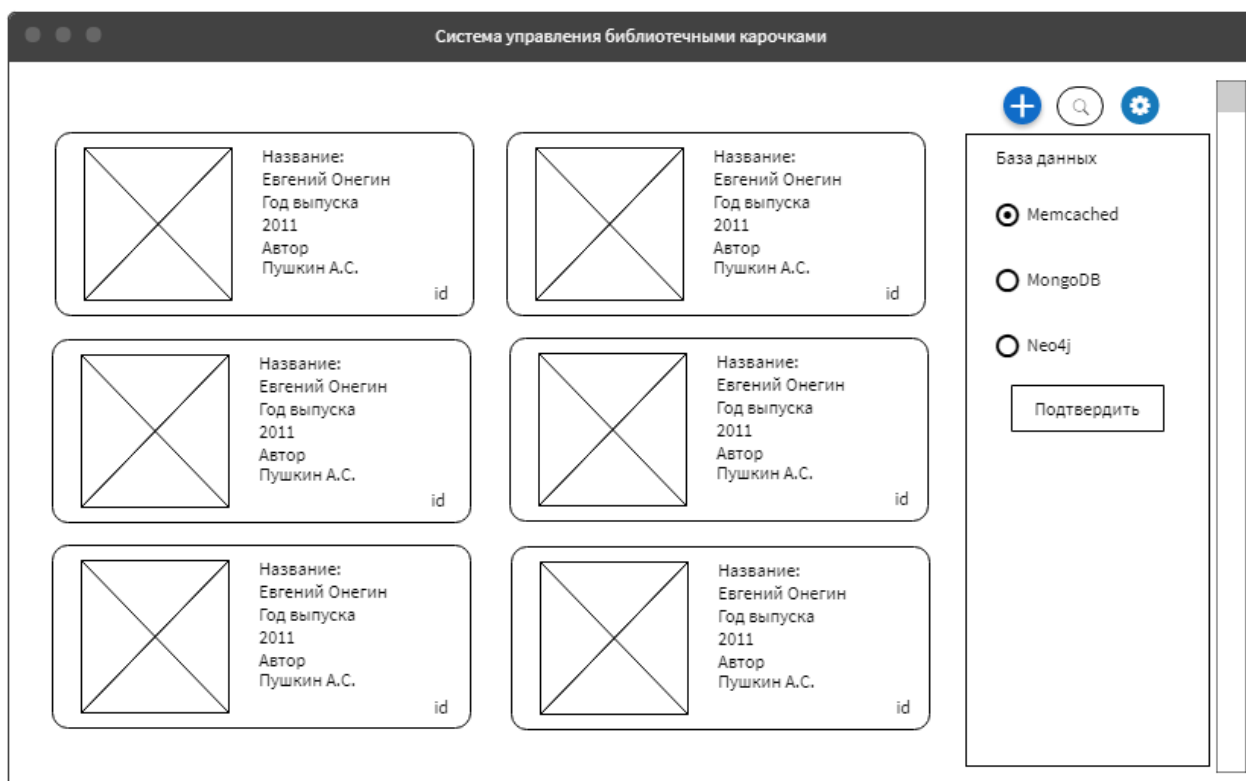


Рисунок 4. Экран со списком книг и открытой панелью переключения БД.

Описание сценариев использования

Единственная роль в системе – библиотекарь. Пользователь входит на страницу. Он видит набор карточек. Справа от него панель поиска.

Для того, чтобы найти нужную книгу, пользователь должен:

1. Нажать на иконку "Лупа".
2. В поле "Название" ввести название интересующей книги.
3. В поле "Автор" ввести автора интересующей книги.
4. В поле "Год издания" ввести год издания интересующей книги.
5. В поле "Идентификатор" ввести идентификатор интересующей книги.
6. В поле "Доступность книги" выбрать: искать только по свободным книгам, только по занятым книгам, либо по всем книгам.

Для того, чтобы добавить книгу, пользователь должен:

1. Нажать на кнопку "Плюс".
2. В поле "Название книги" ввести название добавляемой книги.
3. В поле "Автор книги" ввести автора добавляемой книги.
4. В поле "Год издания книги" ввести год издания добавляемой книги.
5. Нажать на кнопку "Выбрать картинку" и вставить ссылку на изображение.
6. Нажать на кнопку "Добавить книгу"
7. Подождать, пока "Ожидание ответа" не сменится на "Карточка была добавлена".
8. Если "Ожидание ответа" сменилось на "Ошибка соединения", то попробовать в другое время.

Для того, чтобы сменить СУБД:

1. Нажать на кнопку "Шестерёнка".
2. Выбрать СУБД.
3. Нажать на кнопку "Подтвердить".
4. Подождать, пока "Ожидание ответа" не сменится на "СУБД была изменена".
5. Если "Ожидание ответа" сменилось на "Ошибка соединения", то попробовать в другое время.

Для того, чтобы редактировать карточку, пользователь должен:

1. Нажать в любое место интересующей карточки.
2. В появившемся окне изменить нужные поля ("Название" / "Автор" / "Год выпуска").
3. Нажать на кнопку "Сохранить".
4. Подождать, пока окно не закроется.
5. Если окно не закрылось, а появилось сообщение "Ошибка соединения", то попробовать в другое время.

Для того, чтобы зарегистрировать возвращение книги, пользователь должен:

1. Нажать в любое место карточки возвращаемой книги.
2. В появившемся окне в поле "Дата" ввести дату возврата книги или оставить текущую дату.

3. Нажать на кнопку "Вернуть книгу".
4. Нажать на кнопку "Сохранить".
5. Подождать, пока окно не закроется.
6. Если окно не закрылось, а появилось сообщение "Ошибка соединения", то попробовать в другое время.

Для того, чтобы зарегистрировать выдачу книги, пользователь должен:

1. Нажать в любое место карточки регистрируемой книги.
2. В открывшемся окне в поле "ФИО читателя" ввести ФИО читателя книги.
3. В поле "Дата" ввести дату выдачи книги или оставить текущую дату.
4. Нажать на кнопку "Выдать книгу".
5. Нажать на кнопку "Сохранить".
6. Подождать, пока окно не закроется.
7. Если окно не закрылось, а появилось сообщение "Ошибка соединения", то попробовать в другое время.

Возможность пользователя добавлять, редактировать, удалять, просматривать данные реализованные с помощью веб-интерфейса.

Массовое добавление данных в базу реализовано с помощью unit тестов.

Для данного решения в равной степени будут использоваться операции чтения и записи, так как нужно будет искать книги для выдачи и редактировать список выдачи книг.

4. Модель данных

Нереляционные модели данных

1. MONGODB

Каждая карточка представляет собой отдельный документ. Идентификатором документа является идентификатор карточки. В документе есть поля "Название", "Автор", "Год выпуска", "История выдачи", "Изображение". Поле "История выдачи" является списком, у элементов которого есть поля "ФИО читателя" и "Дата выдачи", "Дата возврата". У последнего элемента поле "Дата возврата" может быть null - это означает, что читатель ещё не вернул книгу.

Модель документа:

```
{
  "_id": ObjectId,
  "title": string // название книги
  "author": string // автор книги
  "year": string //год выдачи
  "history": [
    ...
    {
      "reader": string // Ф.И.О читателя
      "date_from": string // дата выдачи
      "date_to": string // дата сдачи
    }
    ...
  ] //история взятия книги
  "image": string// ссылка на изображение книги
}
```

2. Memcached

Информация о каждой карточке хранится по отдельному ключу. Ключ - идентификатор карточки. Значение - переведённый в строку объект python - словарь с ключами "Название", "Автор", "Год издания", "История выдачи", по ключу "История выдачи" будет храниться список словарей с ключами "ФИО читателя", "Дата выдачи", "Дата возврата". У последнего элемента поле "Дата возврата" может быть None - это означает, что читатель ещё не вернул книгу.

Модель документа:

```
{
  "title": string // название книги
  "author": string // автор книги
  "year": string //год выдачи
  "history": [
    ...
    {
      "reader": string // Ф.И.О читателя
      "date_from": string // дата выдачи
      "date_to": string // дата сдачи
    }
    ...
  ] //история взятия книги
  "image": string// ссылка на изображение книги
}
```

3. Neo4j

Neo4j: Для каждого набора книг, имеющих одинаковые названия, авторов и года выпуска, существует отдельная вершина с атрибутами "Название", "Автор", "Год выпуска", "Изображение". Каждая карточка представляется отдельной вершиной, имеющей атрибуты "Идентификатор" и "История выдачи". Поле "История выдачи" является JSON - списком, у элементов которого есть поля "ФИО читателя" и "Дата выдачи", "Дата возврата". У последнего элемента поле "Дата возврата" может быть null - это означает, что читатель ещё не вернул книгу. Каждая карточка имеет связь с соответствующей ей книгой.

Модель вершины каждого набора книг:

```
{
  "title": string // название книги
  "author": string // автор книги
  "year": string //год выдачи
  "history": [
    ...
    {
      "reader": string // Ф.И.О читателя
      "date_from": string // дата выдачи
      "date_to": string // дата сдачи
    }
    ...
  ] //история взятия книги
  "image": string// ссылка на изображение книги
}
```

Модель карточки:

```
{
  "_id": ObjectId,
  "history": [
    ...
    {
      "reader": string // Ф.И.О читателя
      "date_from": string // дата выдачи
      "date_to": string // дата сдачи
    }
    ...
  ] //история взятия книги
}
```

Визуальное представление (Рис. 5):

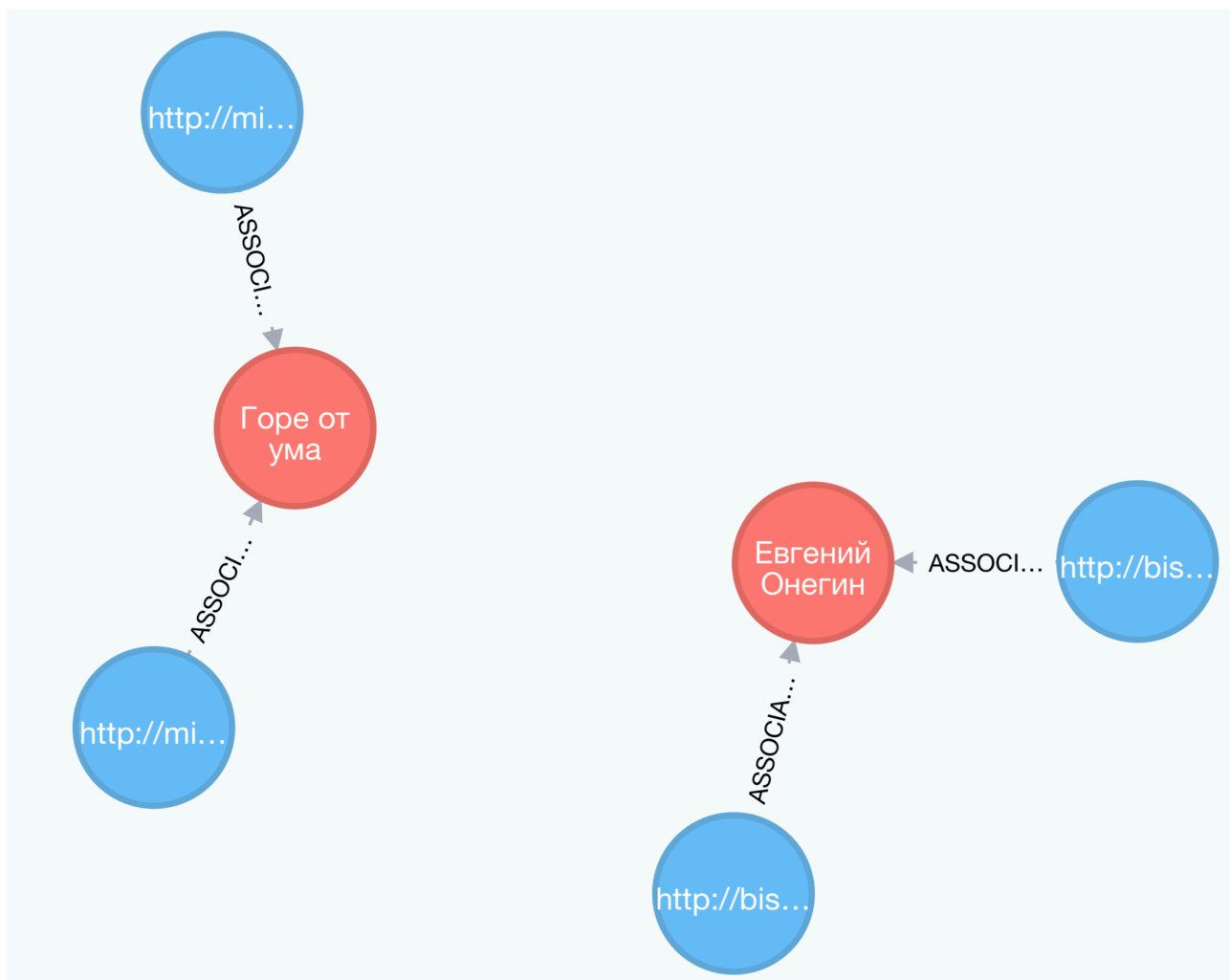


Рисунок 5. Визуальное представление модели данных Neo4j.

Оценка удельного объема информации, хранимой в моделях

Пусть совокупность полей ("Название", "Год выпуска", "Автор", "Изображение") занимает N памяти, M - число доступных книг, K - количество памяти, занимаемое совокупностью полей ("Дата выдачи", "Дата возврата"), H - среднее количество выдач и возвратов всех книг, P - среднее количество экземпляров каждой книги.

Тогда хранение данных

1. В Memcached будет занимать

$$M \cdot P \cdot N + M \cdot P \cdot H \cdot K$$

2. В Neo4j будет занимать

$$M \cdot N + M \cdot P \cdot H \cdot K$$

3. В MongoDB будет занимать

$$M \cdot P \cdot N + M \cdot P \cdot H \cdot K$$

Тогда при условии того, что $N = (50 \text{ байт строка} + 2 \text{ байта число} + 18 \text{ байт автор})$, число доступных книг $M = 300$, $K = (10 \text{ байт дата выдачи} + 10 \text{ байт дата возврата})$, среднее количество выдач и возвратов всех книг $H = 20$, среднее количество экземпляров каждой книги $P = 3$

Тогда хранение данных будет занимать

1. В memcached: $300 \cdot 3 \cdot 70 + 300 \cdot 3 \cdot 20 \cdot 20 = 423000$ байт.
2. В Neo4j: $300 \cdot 70 + 3 \cdot 20 \cdot 20 = 381000$ байт.
3. В MongoDB: $300 \cdot 3 \cdot 70 + 300 \cdot 3 \cdot 20 \cdot 20 = 423000$ байт.

Примеры запросов

Примеры на добавление карточки с идентификатором 1, автором А, годом 2017, названием В и ссылкой на изображение С

1. memcached - "add 1 0 0 54 \r\n{year:2017,author:"A",name:"B",history:[],"image":"C"}\r"
2. Neo4j - "CREATE (card:Card {id: "1", histoty: "null","image" : "C"})
MERGE (book:Book {title: "B", releaseDate: "2017", author: "A"})
MERGE (card)--(book)"
3. MongoDB -
"db.library.insertOne({_id:1,title:"B",releaseDate:"2017",author:"A",histoty:"null","image" :
"C"})"

Примеры на удаление карточки с идентификатором 1:

1. memcached - "delete 1\r"
2. Neo4j - "MATCH (book:Book)-[rel]-(card:Card {id: 1}) DELETE book, rel"
3. MongoDB - "db.library.remove({_id : 1})"

Примеры на поиск истории по идентификатору карточки:

1. memcached - "get 1\r"
2. Neo4j - "MATCH (card:Card {id: 1}) RETURN card.history AS history"
3. MongoDB - "db.library.find({_id: 1})"

Аналог модели данных для SQL СУБД

Вышеописанную модель данных для библиотечных карточек также можно представить в виде реляционной модели с помощью таблиц (Рис. 6):

1. Множество всех книг с идентификатором, названием, годом издания, автором, ссылкой на изображение.
2. Все книги с идентификатором конкретной книги, идентификатором из множества книг.
3. История выдачи всех книг с идентификатором конкретной книги, датой выдачи, датой возврата.

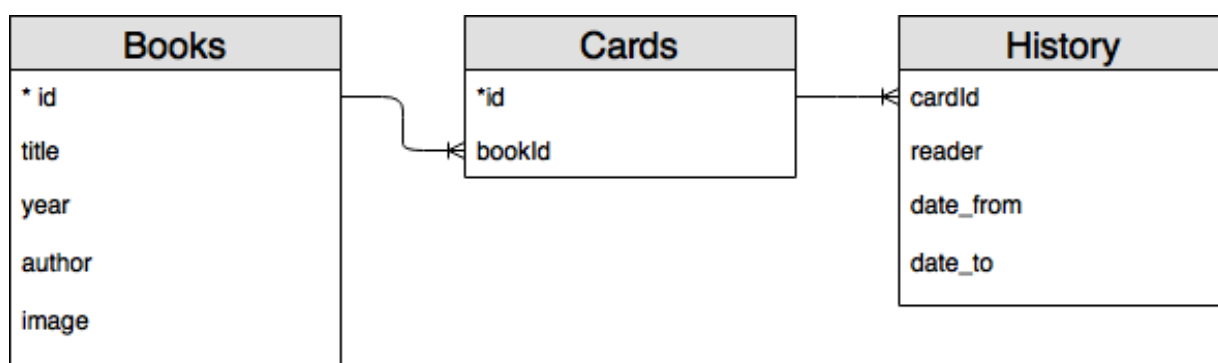


Рисунок 6. Визуальное представление модели данных реляционной базы данных.

Оценка удельного объема информации, хранимой в модели

Пусть совокупность полей ("Название", "Год выпуска", "Автор", "Изображение") занимает N памяти, M - число доступных книг, K - количество памяти, занимаемое совокупностью полей ("Дата выдачи", "Дата возврата"), H - среднее количество выдач и возвратов всех книг, P - среднее количество экземпляров каждой книги. Тогда хранение данных будет занимать $M \cdot N + M \cdot P \cdot H \cdot K$. При условии того, что $N = (50 \text{ байт строка} + 2 \text{ байта число} + 18 \text{ байт автор})$, число доступных книг $M = 300$, $K = (10 \text{ байт дата выдачи} + 10 \text{ байт дата возврата})$, среднее количество выдач и возвратов всех книг $H = 20$, среднее количество экземпляров каждой книги $P = 3$, хранение данных будет занимать 381000 байт.

Примеры запросов

1. Запрос на добавление карточки:

```
INSERT INTO Books VALUES (title, year, author, image) ;
```

```
INSERT INTO Cards VALUES (id, bookId) WHERE bookId = Books.id in
```

```
SELECT id FROM Books WHERE Books.title = title AND Books.year = year AND  
Books.author = author
```

2. Запрос на удаление карточки:

```
DELETE * FROM Cards WHERE Cards.id = id;
```

3. Поиск истории по идентификатору:

```
SELECT * FROM History WHERE History.id = id
```

Сравнение моделей

Пусть совокупность полей ("Название", "Год выпуска", "Автор", "Изображение") занимает N памяти, M - число доступных книг, K - количество памяти, занимаемое совокупностью полей ("Дата выдачи", "Дата возврата"), H - среднее количество выдач и возвратов всех книг, P - среднее количество экземпляров каждой книги

При условии того, что $N = (50 \text{ байт строка} + 2 \text{ байта число} + 18 \text{ байт автор})$, число доступных книг $M = 300$, $K = (10 \text{ байт дата выдачи} + 10 \text{ байт дата возврата})$, среднее количество выдач и возвратов всех книг $H = 20$, среднее количество экземпляров каждой книги $P = 3$

1. В memcached: 423000 байт.
2. В Neo4j: 381000 байт.
3. В MongoDB: 423000 байт
4. В реляционных базах данных: 381000 байт.

Количество запросов на поиск и удаление карточки не отличается. Для добавления карточки в реляционной модели нужно сделать на 1 запрос больше.

Запрос на поиск полей книги по идентификатору конкретной книги

1. В memcached будет занимать $O(1)$.
2. В Neo4j будет занимать $O(M \cdot P)$.
3. В MongoDB будет занимать $O(M \cdot P)$.
4. В реляционных базах необходимо сделать 1 join, по таблице с идентификатором и таблице с книгами, сложность поиска $O((M \cdot P) \cdot M)$.

Запрос на поиск всей истории выдачи по идентификатору конкретной книги

1. В memcached будет занимать $O(1)$.
2. В Neo4j будет занимать $O(M \cdot P)$.
3. В MongoDB будет занимать $O(M \cdot P)$.
4. В реляционных базах необходимо сделать 1 join, по истории выдачи и таблице с идентификаторами, сложность поиска $O((M \cdot P) \cdot (M \cdot P \cdot H))$.

Вывод

Исходя из всех сравнений, проведённых выше, можно сделать вывод, что нереляционные СУБД в данной задаче имеют преимущество в количестве занимаемой памяти, количестве запросов и сложности запросов.

Так как в данной задаче важны скорость чтения и записи, то было решено сравнить нереляционные СУБД между собой по критерию затрачиваемого времени на добавление и чтение данных. Для этого в каждую БД было сначала добавлено 1000 записей, а затем 1000 раз получены записи по случайному индексу. Для точности данная процедура была проделана 10 раз, а в качестве конечного результата было взято среднее значение. Результаты приведены в таблицах 1 и 2.

СУБД	Затраченное время, мин
MongoDB	0:45
Neo4j	2:35
Memcached	1:43

Таблица 1. Время, затраченное на добавление 1000 записей.

СУБД	Затраченное время, мин
MongoDB	0:45
Neo4j	1:00
Memcached	1:21

Таблица 2. Время, затраченное на получение 1000 записей.

Исходя из результатов сравнения можно сделать вывод, что в данном решении MongoDB имеет преимущество перед Memcached и Neo4j в скорости добавления и получения записей.

По удельному объёму информации, хранимой в модели, Neo4j имеет преимущество, так как занимает на 10% меньше памяти.

Что касается удобства в разработке, то здесь у Memcached есть один недостаток: нужно хранить по отдельному ключу список всех идентификаторов записей. Neo4j также не является лучшим решением для данной задачи, так как все возможности работы с графовыми структурами данных задействовать не удалось, потому что графовая структура не очень подходит для хранения книг и читателей. Лучшим решением для данной задачи является MongoDB, так как у него отсутствуют вышеперечисленные недостатки.

5. Разработанное приложение

Краткое описание

Back-end представляет из себя python-приложение. Возможность переключения между СУБД реализована следующим образом:

1. Данные из одной СУБД экспортируются в промежуточный формат (список кортежей, содержащих идентификатор карточки и всю информацию о карточке).
2. Выполняется импорт данных из промежуточного формата в выбранную СУБД.

Front-end – это web-приложение, которое использует API back-end приложения и отображает данные удобным образом для пользователя.

Схема экранов приложения

Экраны приложения и переходы между ними отображены на рисунке 7.



Рисунок 7. Схема экранов приложения.

Использованные технологии

БД: MongoDB, Memcached, Neo4j

Back-end: Python 3.6

Front-end: HTML, CSS, JavaScript.

Ссылки на Приложение

1. Ссылка на github: https://github.com/moevm/nosql-2017-lib_card

6. Вывод

Результаты

В ходе работы было разработано приложение управления библиотечными карточками, которое позволяет пользователям удобно и быстро добавлять, удалять и редактировать их. Приложение предоставляет возможность выбирать одну из нереляционных СУБД для хранения данных, тем самым позволяя использовать их индивидуальные преимущества.

Недостатки и пути для улучшения полученного решения

На данный момент переключение между базами данных реализовано как полное удаление данных с одной БД и перенесение их в другую БД. Данный процесс занимает достаточно много времени и памяти.

Одним из возможных путей решения данной проблемы является разработка системы таким образом, что данные читались и писались одновременно во все СУБД.

Будущее развитие решения

Планируется разработка нативных приложений для OS Windows и MacOS.

7. Приложения

Документация по сборке и развертыванию приложения

1. Скачать проект из репозитория (указан в ссылках на приложение)
2. Запустить main.py
3. Открыть приложение в браузере по адресу 127.0.0.1

8. Используемая литература

1. Документация MongoDB: <https://docs.mongodb.com/manual/>
2. Документация к Memcached: <https://github.com/memcached/memcached/wiki>
3. Документация к Neo4j: <https://neo4j.com/docs/>