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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: ИС Зоопарка

Студентка гр. 9382 _____ Балаева М.О.

Студентка гр. 9382 _____ Пя С.

Студентка гр. 9383 _____ Карпекина А.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2022

ЗАДАНИЕ

Студенты

Карпекина А.А.

Балаева М.О.

Пя С.

Группа 9382, 9383

Тема работы: Разработка веб-приложения "зоопарк", где будет отображена информация о зоопарке.

Исходные данные:

Необходимо реализовать приложение для СУБД(MongoDB).

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:

Студентка гр. 9382		Балаева М.О.
Студентка гр. 9382		Пя С.
Студентка гр. 9383		Карпекина А.А.
Преподаватель		Заславский М.М.

АННОТАЦИЯ

В данной работе было разработано ИС Зоопарка. Для разработки использовались memcached[1] для базы данных, Python[2] для бэкенда и React[3] для фронтенда. Был создан первый прототип в котором реализована возможность просматривать содержимое БД с помощью таблиц. Кроме этого прототип позволяет добавить новые элементы данных в БД. В приложении была создана авторизация, в которой по умолчанию присутствуют по одному отладочному пользователю на две роли - админ и не админ. Были добавлены видео о популярных животных.

SUMMARY

In this paper we developed the Zoo IS. For the development we used memcahed for the database, python for the backend and react for the frontend. The first prototype was created in which the ability to view the contents of the database using tables was implemented. In addition, the prototype allows you to add new data items to the database. Authorization was created in the application, with one debug user per two roles - admin and non-admin by default. Videos about popular animals have been added.

СОДЕРЖАНИЕ

1. Введение

Цель работы - создать веб-приложение для хранения, просмотра и добавления данных о зоопарке, взаимодействующее с базой данных Memcached.

2. Качественные требования к решению

Требуется разработать веб-приложение с использованием СУБД Memcached.

3. Сценарии использования

Макет UI:

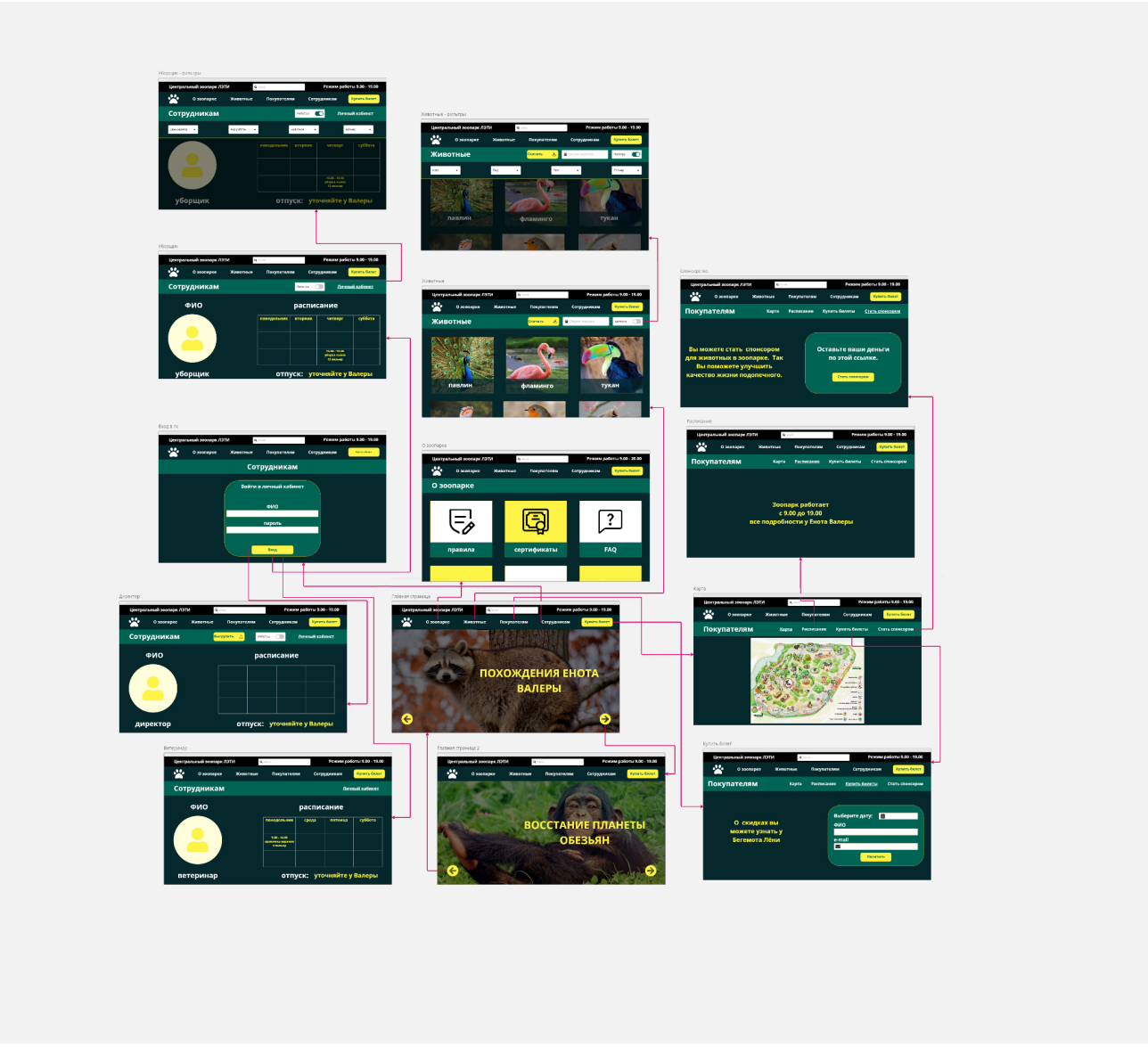

Рисунок 1 –Use Case

Описание сценариев использования:

1. **Посетитель:**

Посетитель оказывается на главной странице, где может листать новостную ленту и осмотреть основные разделы сайта. С главной страницы он может перейти в раздел

- "Животные", в котором располагается вся основная информация о животных зоопарка. Пользователь может найти интересующее животное с помощью поиска, отфильтровать запрос, скачать информацию обо всех животных.
- "О зоопарке", где может ознакомиться с правилами, сертификатами, историей и другой полезной информацией о зоопарке.
- "Посетителям", где может ознакомиться с расписанием и картой, купить билет и помочь зверюшкам.

Из любого раздела можно вернуться на главную страницу, нажав на лапку в левом верхнем углу.

2. **Сотрудник:**

Сотрудник может выполнять все действия, доступные обычному пользователю. В разделе "Сотрудникам", он может войти в личный кабинет. В зависимости от должности у сотрудника будет разное расписание и функционал. Все сотрудники видят информацию о ближайшем отпуске и свое рабочее расписание. Директор может загружать информацию на сайт.

4. Модель данных

Схема БД.

Схема БД представлена на рисунке 2.

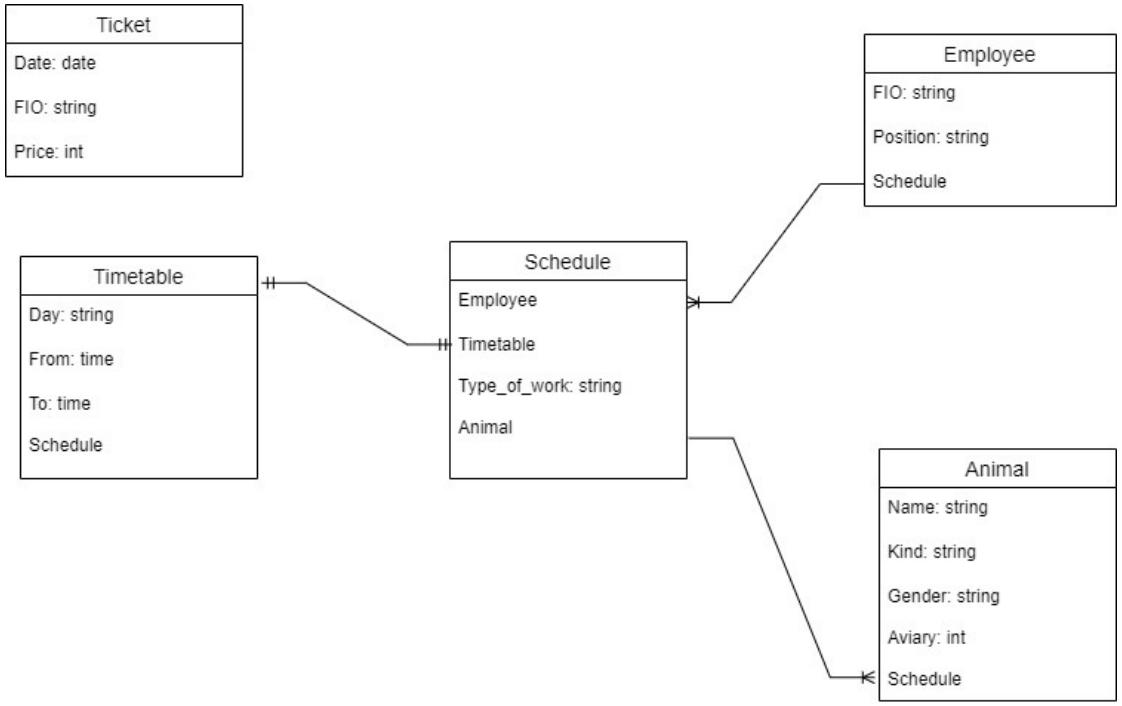

Рисунок 2 – Схема БД

Список сущностей модели:

1. Таблица Ticket:

- Date: date - дата покупки билета
- FIO: string - Фамилия Имя Отчество покупателя
- Price: int - цена билета

2. Таблица Timetable:

- Day: string - день недели
- From: time - время начала работы
- To: time - время окончания работы
- Schedule - график

3. Таблица Schedule:

- Employee - сотрудник
- Timetable - расписание

- Animal - животное

4. Таблица Animal:

- Name: string - имя животного
- Kind: string - вид животного
- Gender: string - пол животного
- Aviary: int - вольер
- Schedule - график

5. Таблица Employee:

- FIO: string - Фамилия Имя Отчество
- Position: string - должность
- Schedule - График

Нереляционная модель данных.

Графическое представление (см. рис. 3)

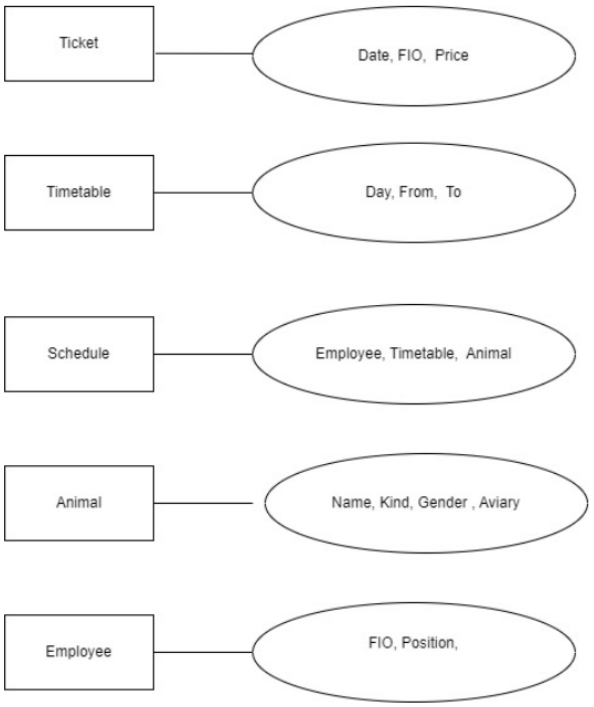

Рисунок 3 - Графическое представление

Список сущностей

БД содержит пять коллекций:

1. Коллекция "Ticket"
- Ключ: "Ticket"
- Значение:
    i. Date - дата покупки билета
    ii. FIO - Фамилия Имя Отчество покупателя
    iii. Price - цена билета

2. Коллекция "Timetable":

- Ключ:"Timetable":
- Значение:
    i. Day - день недели
    ii. From - время начала работы
    iii. To - время окончания работы

3. Коллекция "Schedule":

- Ключ: "Schedule"
- Значение:
    i. Employee - сотрудник
    ii. Timetable - расписание
    iii. Animal - животное

4. Коллекция "Animal":

- Ключ: "Animal"
- Значение:
    i. Name - имя животного
    ii. Kind - тип животного
    iii. Gender - пол животного
    iv. Aviary - вольер

5. Коллекция "Employee":

- Ключ: "Employee"

- Значение:
    i. FIO - Фамилия Имя Отчество
    ii. Position - должность

Оценка объема

1. Ключ "Ticket" - string, символ занимает 1 байт, количество символов N в среднем.
2. Значение - байт-строка, состоящая из
    i. Date - примерно 10 символов + размер ключа 10
    ii. FIO - количество N символов + размер ключа 10
    iii. Price - примерно 10 символов + размер ключа 10 Тогда объем записи одного билета составляет (20 + 2 * N + 30) байт Коллекция "Timetable":
3. Ключ:"Timetable" - string, символ занимает 1 байт, количество символов N в среднем.
4. Значение - байт-строка, состоящая из
    i. Day - примерно N символов + размер ключа 10
    ii. From - примерно N символов + размер ключа 10
    iii. To - примерно 8 символов + размер ключа 10 Тогда объем записи одного расписания составляет (8 + 3 * N + 30) байт Коллекция "Schedule":
5. Ключ: "Schedule" - string, символ занимает 1 байт, количество символов N в среднем.
6. Значение - байт-строка, состоящая из
    i. Employee - примерно 10 символов + размер ключа 10
    ii. Timetable - примерно 10 символов + размер ключа 10
    iii. Animal - примерно 10 символов + размер ключа 10 Тогда объем записи одного графика составляет (30 + N + 30) байт Коллекция "Animal":

7. Ключ: "Animal" - string, символ занимает 1 байт, количество символов N в среднем.
8. Значение - байт-строка, состоящая из
    i. Name - примерно N символов + размер ключа 10
    ii. Kind - примерно N символов + размер ключа 10
    iii. Gender - примерно N символов + размер ключа 10
    iv. Aviary - примерно 10 символов + размер ключа 10 Тогда объем записи одного животного составляет (10 + 4 * N + 40) байт
    Коллекция "Employee":
9. Ключ: "Employee" - string, символ занимает 1 байт, количество символов N в среднем.
10. Значение - байт-строка, состоящая из
    i. FIO - примерно N символов + размер ключа 10
    ii. Position - примерно N символов + размер ключа 10
    Тогда объем записи одного сотрудника составляет (3 * N + 20) байт

Общий объем базы данных можно оценить следующим образом:

N_Ticket * (20 + 2 * N + 30) + N_Schedule * (38 + 4 * N + 60) + N_Employee * (3*N + 20) + N_Animal * (10 + 4 * N + 40).

Если принять за среднее значения N_Ticket = 1000 (за неделю), N_Employee = 50, N_Schedule = 5 * N_Employee (если сотрудник работает пять дней в неделю) = 5 * 50 = 250, N_Animal = 100, N = 30 (символов в строке), тогда

1000 * (20 + 2 * 30 + 30) + 250 * (38 + 4 * 30 + 60) + 50* (330 + 20) + 100 * (10 + 4 * 30 + 40) = 187000 байт за неделю.

Выразим объем модели через количество билетов, тогда получим линейную зависимость равную

N * (20 + 2 * 30 + 30) + N / 4 * (38 + 4 * 30 + 60) + N / 20 * (330 + 20) + N/10 * (10 + 4 * 30 + 40) = 187N

Тогда объем чистых данных:

N_Ticket * (20 + N) + N_Schedule * (38 + 3 * N) + N_Employee * (2N) + N_Animal * (10 + 3 * N) 1000 * (20 + 30) + 250 * (38 + 3 * 30) + 50 * (230) + 100 * (10 + 3 * 30) = 95000 байт за неделю. Выразим объем модели через количество билетов, тогда получим линейную зависимость равную

N * (20 + 30) + N/4 * (38 + 3 * 30) + N/20 * (2*30) + N/10 * (10 + 3 * 30) = 95N

Избыточность модели.

187N / 95N = 1.96842105263

Как видно, модель сильно увеличивает объем данных, поскольку вместе с увеличением основного объема дублируются ключи в значениях.

Направление роста модели при увеличении количества объектов каждой сущности.

По количеству сотрудников:
N * (110 * 5 + 218) = N * 768
По животным:
N * 170

Запросы

1. Запрос на добавление билета:

```
add ticket 0 0 44
{ "Date": date, "FIO": fio, "Price": price }
```

2. Запрос на добавление расписания:

```
add timetable 0 0 66
{ "id_schedule": id_schedule, "Day": day, "From": from, "To": to }
```

3. Запрос на добавление сотрудника:

```
add animal 0 0 66
{ "Name": name, "Kind": kind, "Gender": gender, "Aviary": aviary }
```

4. Запрос на добавление животного:

```
add animal 0 0 66
{ "Name": name, "Kind": kind, "Gender": gender, "Aviary": aviary }
```

5. Запрос на добавление графика:

```
add schedule 0 0 112
{ "id_schedule": id_schedule, "id_employee": id_employee, "id_timetable": id_timetable, "id_animal": id_animal }
```

6. Запрос на обновление билета:

```
replace ticket 0 0 47
{ "Date": dateR, "FIO": fioR, "Price": priceR }
```

7. Запрос на обновление расписания:

```
replace timetable 0 0 70
{ "id_schedule": id_scheduleR, "Day": dayR, "From": fromR, "To": toR }
```

8. Запрос на обновление сотрудника:

```
replace employee 0 0 38
{ "FIO": fioR, "Position": positionR }
```

9. Запрос на обновление животного:

```
replace animal 0 0 70
{ "Name": nameR, "Kind": kindR, "Gender": genderR, "Aviary": aviaryR }
```

10. Запрос на обновление графика:

```
add schedule 0 0 116
{ "id_schedule": id_scheduleR, "id_employee": id_employeeR, "id_timetable": id_timetableR, "id_animal": id_animalR }
```

Реляционная модель данных.

Графическое представление.

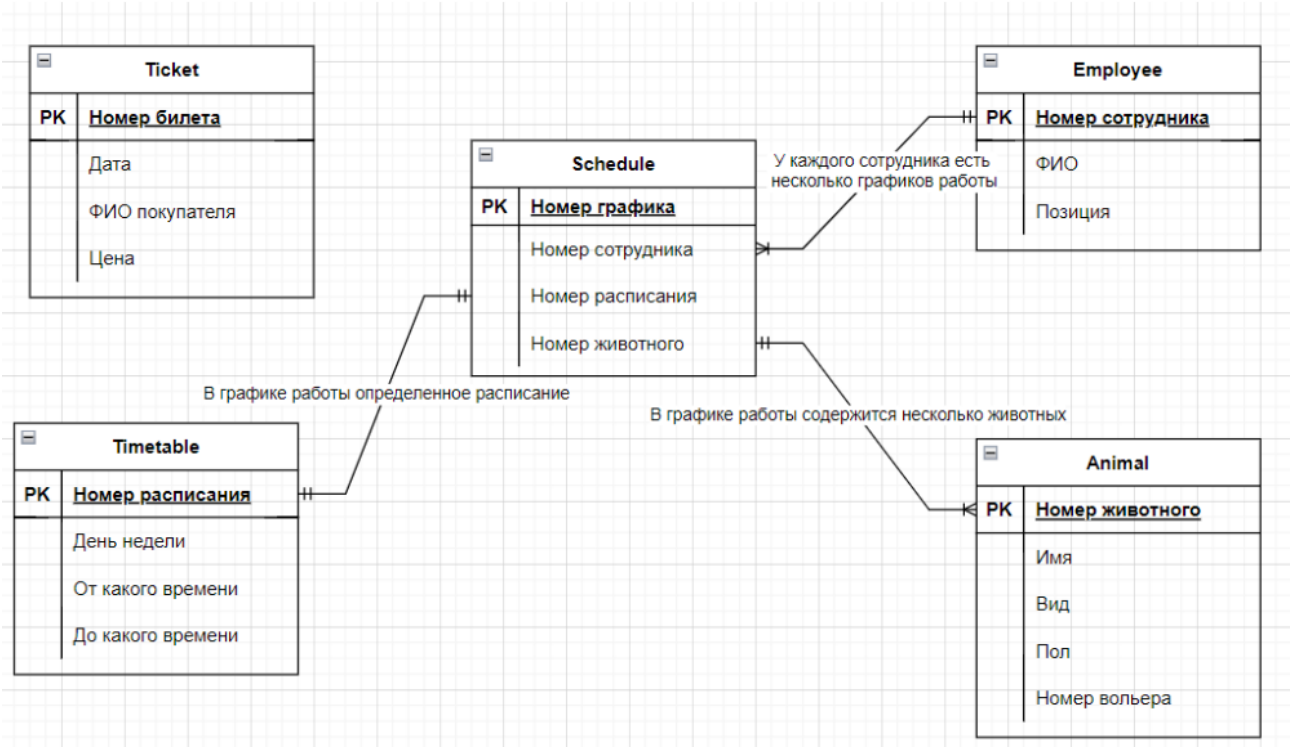

Рисунок 4 - Графическое представление

Оценка объема

- Коллекция "Ticket":
    1. id_ticket - тип int, занимает 4 байта
    2. Date - тип date, занимает 3 байта
    3. FIO - тип string, символ занимает 1 байт, количество символов N в среднем
    4. Price - тип int, занимает 4 байта
       Тогда объем записи одного билета составляет (11 + N) байт
- Коллекция "Timetable":
    1. id_schedule - тип int, занимает 4 байта
    2. Day - тип string, символ занимает 1 байт, количество символов N в среднем
    3. From - тип time, занимает 3 байта
    4. To - тип time, занимает 3 байта
       Тогда объем записи одного расписания составляет (10 + N) байт
- Коллекция "Schedule":
    1. id_schedule - тип int, занимает 4 байта
    2. id_employee - тип int, занимает 4 байта

3. id_timetable - тип int, занимает 4 байта
4. id_animal - тип int, занимает 4 байта
   Тогда объем записи одного графика работы составляет 16 байт

- Коллекция "Employee":
  1. id_employee - тип int, занимает 4 байта
  2. FIO - тип string, символ занимает 1 байт, количество символов N в среднем
  3. Position - тип string, символ занимает 1 байт, количество символов N в среднем
     Тогда объем записи одного работника составляет (4 + 2*N) байт
- Коллекция "Animal":
  1. id_animal - тип int, занимает 4 байта
  2. Name - тип string, символ занимает 1 байт, количество символов N в среднем
  3. Kind - тип string, символ занимает 1 байт, количество символов N в среднем
  4. Gender - тип string, символ занимает 1 байт, количество символов N в среднем
  5. Aviary - тип int, занимает 4 байта
     Тогда объем записи одного животного составляет (8 + 3*N) байт

Общий объем базы данных можно оценить следующим образом:

N_Ticket * (11 + N) + N_Schedule * (16 + 10 + N) + N_Employee * (4 + 2N) + N_Animal * (8 + 3N).

Если принять за среднее значения N_Ticket = 1000 (за неделю), N_Employee = 50, N_Schedule = 5 * N_Employee (если сотрудник работает пять дней в неделю) = 5 * 50 = 250, N_Animal = 100, N = 30 (символов в строке), тогда

1000 * (11 + 30) + 250 * (16 + 10 + 30) + 50 * (4 + 230) + 100(8 + 330) = 68000 байт за неделю.

Выразим объем модели через количество билетов, тогда получим линейную зависимость равную N * (11 + 30) + N/4 (16 + 10 + 30) + N/20* (4 + 230) + N/10 (8 + 3*30) = 68N

Тогда объем чистых данных:

N_Ticket * (7 + N) + N_Schedule * (6 + N) + N_Employee * (2N) + N_Animal * (4 + 3N).

1000 * (7 + 30) + 250 * (6 + 30) + 50 * (230) + 100(4 + 330) = 41000 + 14000 + 3200 = 58400 байт за неделю.

Выразим объем модели через количество билетов, тогда получим линейную зависимость: N * (7 + 30) + N/4 * (6 + 30) + N/20 * (230) + N/10 * (4 + 3*30) = 58.4N

Избыточность модели.

Вычислим отношение фактического и «чистого» объемов данных:
68N / 58.4N = 1.16438356
Таким образом, нереляционная модель более избыточна вследствие хранения идентификаторов и ключей в каждой записи. Данная реализация на SQL выигрывает NoSQL реализации по избыточности.

Направление роста модели при увеличении количества объектов каждой сущности.

По количеству сотрудников:
N * (56 * 5 + 64) = N * 344
По животным:
N * 98

Запросы:

1. Запрос на добавление билета:

```
INSERT Ticket(id_ticket, Date, FIO, Price)
    VALUES (id_ticket1, Date1, FIO1, Price1), (id_ticket2, Date2, FIO2, Price2);
```

2. Запрос на добавление расписания:

```
INSERT Timetable(id_schedule, Day, From, To)
    VALUES (id_schedule1, Day1, From1, To1), (id_schedule2, Day2, From2, To2);
```

3. Запрос на добавление сотрудника:

```
INSERT Employee(id_employee, FIO, Position)
    VALUES (id_employee1, FIO1, Position1), (id_employee2, FIO2, Position2);
```

4. Запрос на добавление животного:

```
INSERT Animal(id_animal, Name, Kind, Gender, Aviary)
    VALUES (id_animal1, Name1, Kind1, Gender1, Aviary1), (id_animal2, Name2, Kind2, Gender2, Aviary2);
```

5. Запрос на добавление графика:

```
INSERT Schedule(id_schedule, id_employee, id_timetable, id_animal)
    VALUES (id_schedule1, id_employee1, id_timetable1, id_animal1), (id_schedule2, id_employee2, id_timetable2, id_ani
```

6. Запрос на обновление билета:

```
UPDATE Ticket
SET Date = date,
    FIO = fio,
    Price = price
WHERE id_ticket = id;
```

7. Запрос на обновление расписания:

```
UPDATE Timetable
SET Day = day,
    From = from,
    To = to
WHERE id_schedule = id;
```

8. Запрос на обновление сотрудника:

```
UPDATE Employee
SET FIO = fio,
    Position = position
WHERE id_employee = id;
```

9. Запрос на обновление животного:

```
UPDATE Animal
SET Name = name,
    Kind = kind,
    Gender = gender,
    Aviary = aviary
WHERE id_animal = id;
```

10. Запрос на обновление графика:

```
UPDATE Schedule
SET id_employee = id_e,
    id_timetable = id_t,
    id_animal = id_a
WHERE id_schedule = id;
```

Сравнение моделей.

NoSQL требует меньше памяти. В то же время SQL выигрывает по количеству запросов, однако Memcached работает быстрее, т.е на небольших БД использовать целесообразнее.

Пример хранения данных:

key	value	type	delete
Animal	[{"id": 2, "Name": "Sjjkj", "Kind": "lion", "Gender": "m", "Aviary": 1256}]	string	X
Employee	[{"id": 3, "FIO": "Jfjfjfj Kkdkdkd Akklkl ", "Position": "cleaner"}]	string	X
Schedule	[[{"id": "0", "Employee": "1415", "Timetable": "2014", "Animal": "4015"}]]	string	X
Ticket	[[{"id": 4, "Date": "2022-12-14", "FIO": "LLLL Pooo Koooo", "Price": 1234}]]	string	X
Timetable	[{"id": 1, "Day": "Monday", "From": "09:00:00", "To": "19:00:00"}]	string	X

Рисунок 5 - Пример хранения данных

5. Разработанное приложение

Краткое описание

Back-end представляет из себя python-приложение. Реализация представлена следующим образом:

1. Данные из файла экспортируются в Memcahed, и далее работа идет с СУБД. Из нее данные экспортируются в промежуточный формат(json).
2. Выполняется импорт данных из промежуточного формата в СУБД.
3. Для взаимодействия с back-end был использован фреймворк Flask.

Front-end представляет из себя web-приложение, использующее React. С его помощью можно удобно взаимодействовать с базой данных.

Схема экранов приложения

Экраны приложения и переходы между ними представлены на рисунке .

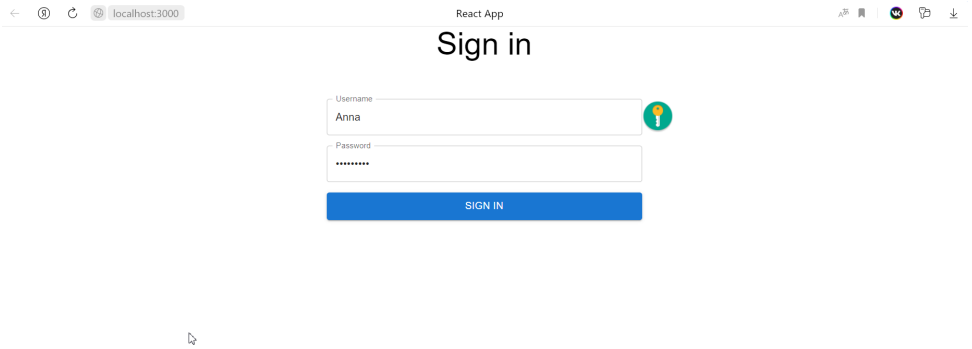

Рисунок 6 - Страница авторизации

Рисунок 7 - Домашняя страница(администратор)

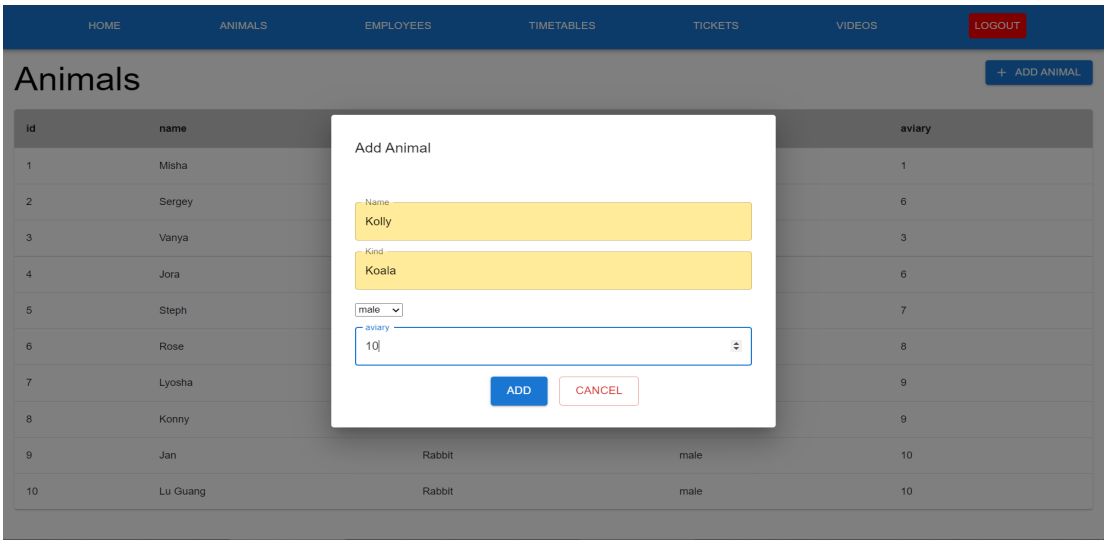

Рисунок 8 - Добавление животного

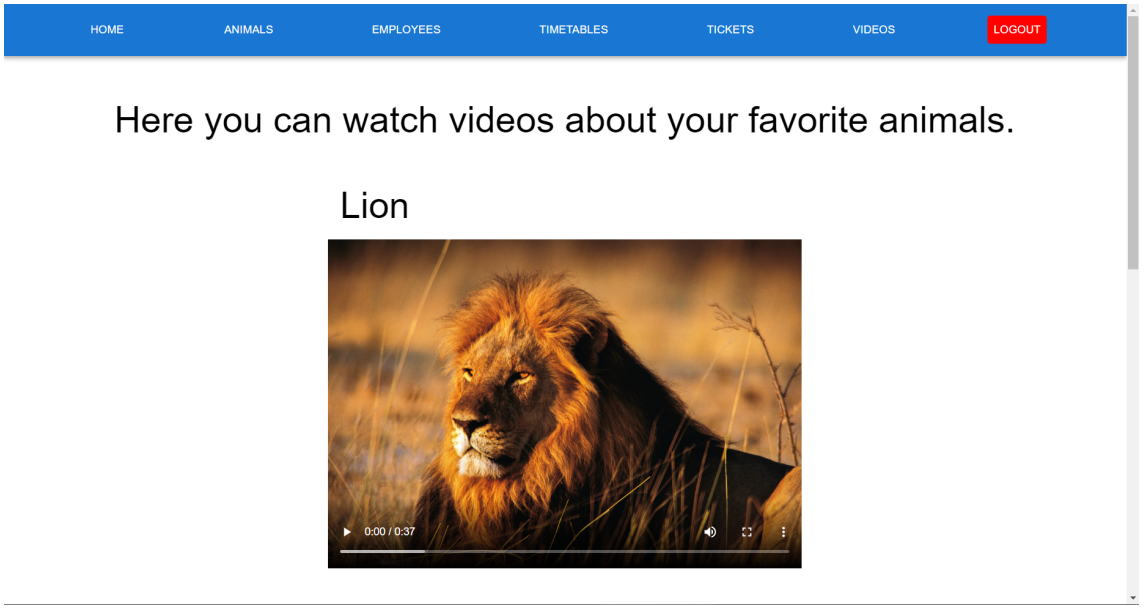

Рисунок 9 - Вкладка Videos

Использованные технологии

СУБД: Memcached

Backend: Python, Flask

Frontend: HTML, React.

Ссылки на приложение

Ссылка на github: https://github.com/moevm/nosql2h22-zoo

ВЫВОД.

Результаты

В ходе работы было разработано web-приложение “Зоопарк”, позволяющее пользователям взаимодействовать с базой данных: просмотр содержимого СУБД с помощью таблиц, добавление новых элементов, также была реализована авторизация.

Недостатки и пути для улучшения полученного решения

В данный момент экспорт происходит из файла, так как СУБД не работает на отдельном сервере постоянно. Реализован только базовый функционал - добавление элемента и просмотр данных. Реализованы базовые элементы веб-приложения, не соответствующие макету.

Будущее развитие решения

Планируется доработать приложение до состояния макета с полным функционалом: сортировка, фильтрация, импорт и экспорт данных.

ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА

1. Документация Memcached: https://github.com/memcached/memcached/wiki
2. Документация Python: https://docs.python.org/3/
3. Документация React: https://reactjs.org/

ПРИЛОЖЕНИЕ

Документация по сборке и развертыванию приложения

1. Скачать проект из репозитория (указан в ссылках на приложение)
2. Запустить в папке backend app.py, открыть приложение в браузере по адресу 127.0.0.1:6969, убедиться в работоспособности backend.
3. Запустить в папке frontend npm start