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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные СУБД»

Тема: Информационная система для сети точек продаж спортивного питания

Студентка гр. 9303 _____ Москаленко Е.М.

Студент гр. 9303 _____ Камакин Д.В.

Студент гр. 9304 _____ Арутюнян С.Н.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2022

ЗАДАНИЕ НА ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

Студентка Москаленко Е.М. Группа 9303

Студент Камакин Д.В. Группа 9303

Студент Арутюнян С.Н. Группа 9304

Тема работы: Информационная система для сети точек продаж спортивного питания

Содержание пояснительной записки:

- Аннотация
- Содержание
- Введение
- Качественные требования к решению
- Сценарии использования
- Модель данных
- Разработанное приложение
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 40 страниц.

Дата выдачи задания: 01.09.2022

Дата сдачи реферата: 22.12.2022

Дата защиты реферата: 22.12.2022

Студентка	_____	Москаленко Е.М.
Студент	_____	Камакин Д.В.
Студент	_____	Арутюнян С.Н.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В данной работе разработана информационная система для точек продаж спортивного питания, позволяющая просматривать и фильтровать информацию о точках, сотрудниках, продуктах, запасах и поставщиках продукции, а также добавлять в систему новую информацию. Для frontend-части в системе используется следующий стек технологий: язык программирования TypeScript, библиотека React, библиотека компонентов Material UI; для backend-части: язык программирования Python, веб-фреймворк Sanic, система управления базами данных MongoDB, драйвером для которой выбран Motor. Также для запуска проекта используется Docker Compose.

SUMMARY

In this work, an information system has been developed for points of sale of sports nutrition, which allows you to view and filter information about points, employees, products, stocks and suppliers of products, as well as add new information to the system. For the frontend part, the system uses the following technology stack: TypeScript programming language, React library, Material UI component library; for the backend part: Python programming language, Sanic web framework, MongoDB database management system, for which Motor is selected as the driver. Docker Compose is also used to run the project.

СОДЕРЖАНИЕ

Введение 5

1. Качественные требования к решению 6

    1.1. Текущие требования 6

2. Сценарии использования 7

    2.1. Макет пользовательского интерфейса 7

    2.2. Сценарии использования 7

    2.3. Преобладающие операции 10

3. Модель данных 12

    3.1. Нереляционная модель данных 12

    3.2. Реляционный аналог модели данных 22

    3.3. Сравнение моделей 30

4. Разработанное приложение 31

    4.1. Описание приложения 31

    4.2. Использованные технологии 31

Заключение 32

Список использованных источников 33

Приложение А. Макет пользовательского интерфейса 34

Приложение Б. Документация по сборке и развёртыванию приложения 35

Приложение В. Снимки экрана приложения 36

Приложение Г. Запросы к NoSQL модели, с помощью которых реализуются сценарии использования 42

Приложение Д. Запросы к реляционной модели, с помощью которых реализуются сценарии использования 50

ВВЕДЕНИЕ

В настоящее время спортивное питание пользуется популярностью: согласно данным Grand View Research, объем мирового рынка спортивного питания в 2022 г. оценивается в $42,9 млрд. За период 2023–2030 гг. ожидается рост рынка со среднегодовым темпом 7,4%. В связи с этим актуально создать информационную систему для хранения и обработки всей необходимой информации, связанной с поставщиками, продукцией и работниками, которой может пользоваться та или иная сеть точек продаж.

Цель работы – разработать информационную систему для сети магазинов спортивного питания, которая позволит хранить и обрабатывать необходимую информацию для успешного функционирования бизнес-процессов.

Основные задачи:

1. Сформулировать основные сценарии использования приложения.
2. Разработать макет пользовательского интерфейса.
3. Разработать схему базы данных.
4. Подготовить прототип приложения.

В рамках данной работы решено реализовать web-приложение, позволяющее просматривать и фильтровать по определенным критериям, а также добавлять информацию о поставщиках и поставляемой продукции, существующих точках продаж, их сотрудниках и запасах.

1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ

1.1. Текущие требования

Текущие требования к решению выглядят следующим образом:

1. Имеется главная страница с приветствием и, возможно, кратким описанием сети продаж. Пользователю предоставляется возможность выбрать вкладку “Branches” или “Suppliers” для работы с той или иной информацией.
2. Предусмотрена возможность импортирования и экспортирования всех имеющихся данных в формате JSON.
3. На странице Suppliers пользователю позволяется просматривать общую информацию о поставщиках, список которых представлен в режиме пагинации. Пользователь может перейти на страницу того или иного поставщика по кликабельной ссылке, просмотреть информацию о нем (id, имя, телефон, почта), а также перейти на страницу с продуктами поставщика. На странице с продуктами предоставляется возможность добавить новый продукт, а также отфильтровать по критериям (id, имя, ценовой диапазон, id деструктора - сущности, являющейся абстракцией над одинаковыми продуктами от разных поставщиков).

   На общей странице поставщиков также возможно добавить нового поставщика, найти поставщиков по определенным критериям и найти конкретные продукты всех поставщиков.
4. На странице Branches пользователю позволяется просматривать общую информацию о точках продаж, список которых представлен в режиме пагинации. Пользователь может перейти на страницу той или иной точки по кликабельной ссылке, просмотреть информацию о ней (id, название, город), а также о сотрудниках и запасах. На странице с сотрудниками предоставляется возможность добавить нового сотрудника, а также отфильтровать по критериям. Страница

с запасами аналогична. На общей странице точек продаж также возможно добавить новую точку, найти точки по определенным критериям, а также найти и отфильтровать сотрудников всех точек.

5. Используемые на момент разработки данные - синтетические.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет пользовательского интерфейса

Разработан макет пользовательского интерфейса, представленный в приложении А.

2.2. Сценарии использования

Сценарий использования «Добавление работника в филиал»:

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь заходит на страницу филиалов(главная)
2. Пользователь выбирает нужный филиал в таблице, нажимает на кнопку получения информации о нём (details)
3. Происходит переход на страницу филиала (по умолчанию на вкладку с работниками)
4. Пользователь нажимает на кнопку добавления нового работника (+)
5. Открывается страница, на которой требуется ввести данные (обязательно: id работника, необязательно: его должность)
6. Пользователь вводит необходимые данные, нажимает на кнопку добавления
7. Происходит переадресация на обновлённую страницу с работниками

Альтернативный сценарий: Работник ещё не добавлен в систему, тогда пользователь одновременно создаст запись о сотруднике и прикрепит к нужному филиалу.

Сценарий использования «Добавление нового работника»:

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь заходит на страницу филиалов (главная)
2. Пользователь выбирает нужный филиал в таблице, нажимает на кнопку получения информации о нём (details)
3. Происходит переход на страницу филиала (по умолчанию на вкладку с работниками)
4. Пользователь нажимает на кнопку добавления нового работника (+)
5. Открывается страница, на которой требуется ввести данные (обязательно: id работника, необязательно: его должность)
6. Пользователь вводит необходимые данные, нажимает на кнопку добавления
7. Происходит переадресация на обновлённую страницу с работниками

Альтернативный сценарий: Работник ещё не добавлен в систему, тогда пользователь одновременно создаст запись о сотруднике и прикрепит к нужному филиалу

Сценарий использования «Добавление нового поставщика»:

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит на страницу поставщиков
2. Пользователь нажимает на кнопку добавления нового поставщика (+)
3. Открывается страница, на которой требуется ввести данные (обязательно: имя, id продукта, цена)
4. Пользователь вводит необходимые данные, нажимает на кнопку добавления
5. Происходит переадресация на обновлённую страницу с поставщиками

Альтернативный сценарий: Продукт ещё не добавлен в систему, тогда пользователь одновременно создаст запись о продукте и прикрепит к нужному поставщику.

Сценарий использования «Импорт данных»:

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь заходит на страницу Branches или Suppliers
2. Пользователь нажимает на кнопку IMPORT и выбирает файл в формате JSON с импортируемыми данными
3. Отобразится сообщение с результатом импорта данных

Сценарий использования «Экспорт данных»:

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь заходит на страницу Branches или Suppliers
2. Пользователь нажимает на кнопку EXPORT
3. На панели браузера появляется загруженный файл формата JSON

Альтернативный сценарий: выводится сообщение об ошибке экспорта.

2.3. Преобладающие операции

На странице Suppliers преобладают операции чтения, так как просматривать продукты поставщиков придется каждый раз при обновлении запасов в точках продаж, а добавлять новых поставщиков придется нечасто. Операция добавление новых поставляемых товаров также не требует ежедневного пользования.

На странице Branches преобладают операции записи, так как каждую покупку или пополнение запасов нужно учесть в информационной системе и обладать актуальными данными о товарах в наличии. Также нужно вести учет сотрудников, обновлять информацию об отработанных сменах, отпусках и изменении заработной платы.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

Разработана схема нереляционной базы данных (рисунок 1).

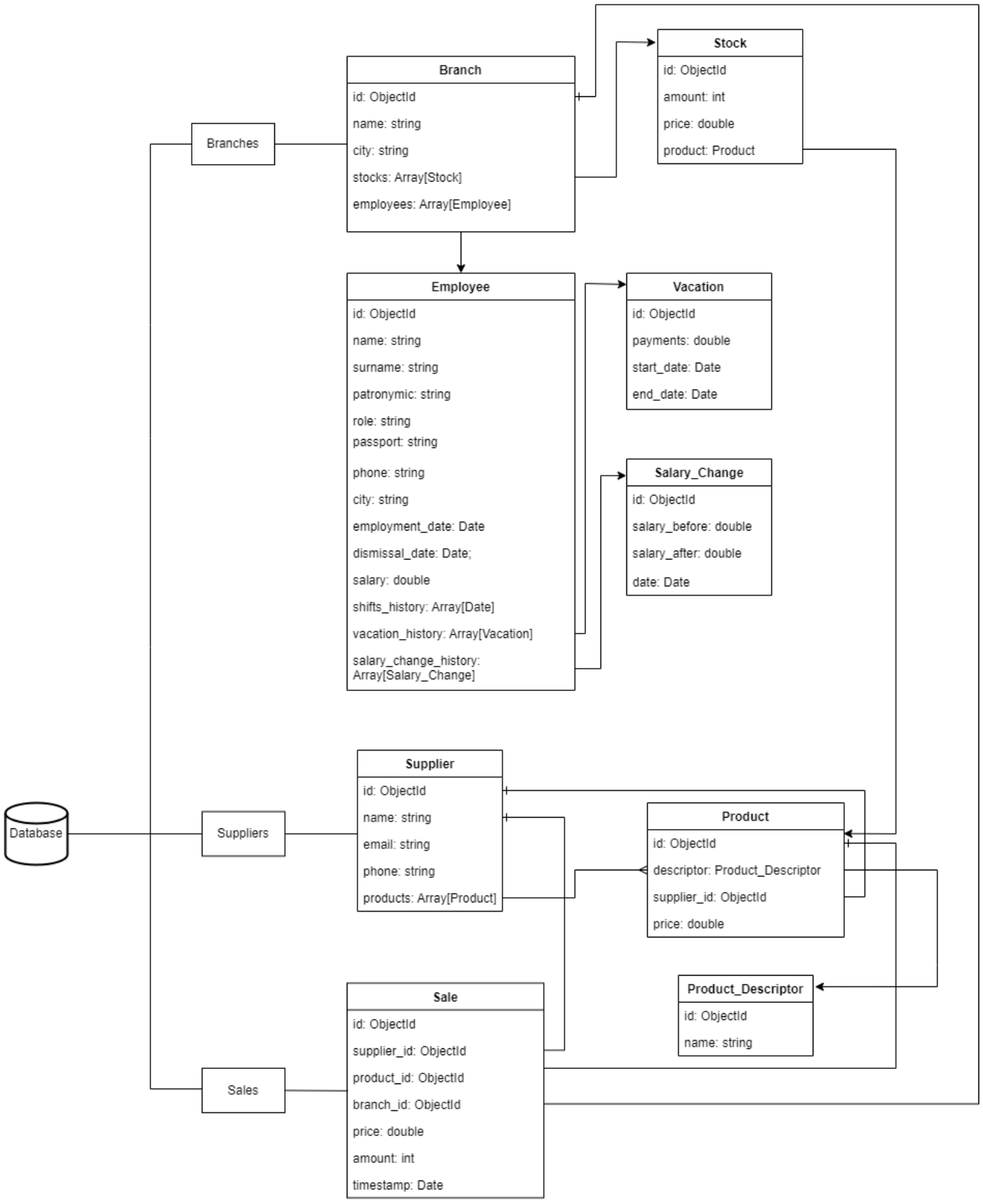

Рисунок 1 – Графическое представление нереляционной базы данных

Описание назначений коллекций, типов данных и сущностей:

БД содержит 3 коллекции: Филиалы (Branches), Поставщики (Suppliers), Продажи (Sales).

Обоснуем такой выбор коллекций:

Продажи вынесены в отдельную коллекцию для их выборки, например, за определённый период у определенного поставщика. Вся требуемая информация хранится в отдельном документе и нет необходимости искать по разным уровням вложенности разных коллекций.

Филиалы хранят достаточное количество информации для получения общей справки о конкретной точке продажи. Эта информация содержит в том числе товары, при этом получать один и тот же продукт возможно от разных поставщиков. В случае добавления объекта Supplier в товар было бы избыточное количество дублирования, которое требовало бы обслуживания, например, в случае изменения контактного телефона поставщика. Поэтому решено ввести коллекцию поставщиков, на которую ссылается товар.

Коллекция Branches:

Branches содержит документы со следующими полями:

- name - string, название
- city - string, город
- stocks - массив объектов типа Stock
- employees - массив объектов типа Branch_Employee
- id - ObjectId

Stock - объект, представляющий собой запас филиала. Включает следующие поля:

- amount - int, количество определенного товара
- price - double, цена для продажи
- product - документ типа Product (см. коллекция Products)
- id - ObjectId

Employee - объект с информацией о сотруднике.

- name - string, имя
- surname - string, фамилия
- patronymic - string, отчество
- role - string, должность сотрудника в рассматриваемом филиале
- city - string, город
- passport - string, паспортные данные
- phone - string, номер телефона
- employment_date - Date, дата трудоустройства
- dismissal_date - Date, дата увольнения
- salary - double, текущая зарплата
- shifts_history - массив отработанных смен
- vacation_history - массив объектов типа Vacation, история отпусков
- salary_change_history - массив объектов типа Salary_Change, история изменений зарплаты
- id - ObjectId

Объект Vacation включают следующие поля:

- payments - double, отпускные выплаты
- start_date - Date, дата начала отпуска
- end_date - Date, дата окончания отпуска
- id - ObjectId

Объект типа Salary_Change включают следующие поля:

- salary_before - double, зарплата до ее изменения
- salary_after - double, зарплата после изменения
- date - Date, дата изменения зарплаты
- id - ObjectId

Коллекция Suppliers:

Содержит документы типа Supplier:

- id - ObjectId
- name - string, имя
- email - string, почта
- phone - string, номер телефона
- products - массив документов типа Product

Объекты Product_Descriptor описывают тот или иной товар и позволяют одинаковым товарам от разных поставщиков ссылаться на общий описатель.

- name - string, название товара
- id - ObjectId

Объекты типа Product содержат описания товара, получаемого от поставщика:

- id - ObjectId
- descriptor - объект типа Product_Descriptor
- supplier_id - ObjectId поставщика
- price - double, закупочная цена

Коллекция Sales:

Содержит документы типа Sale - продажа:

- id - ObjectId
- supplier_id - id-ссылка на поставщика
- product_id - id-ссылка на товар
- branch_id - id-ссылка на филиал
- price - double, выручка
- amount - int, количество проданного товара
- timestamp - Date, дата продажи

Оценка удельного объема информации, хранимой в модели

Будем считать, что все имена, фамилии, отчества и города в среднем занимают по 20 байт, а номера телефонов и должности - 15. Единицами измерения являются байты.

Объем памяти для одного филиала Branch:

- id - 8
- name - 20
- city - 20
- stock - Nstocks * 68
- employees - Nbranch-employees * (157 + Nshifts * 8 + Nvacation * 32 + Nsalarychg * 32)

В результате получаем следующий объем для документа Branch: 8 + 20 + 20 + Nstocks * 68 + Nemployees * 31 = 48 + Nstocks * 68 + Nemployees * (157 + Nshifts * 8 + Nvacation * 32 + Nsalarychg * 32)

Stock:

- id - 8
- amount - 4
- price - 8
- product - 48

Итого: 68 байт

Product:

- id - 8
- supplier_id - 8
- price - 8
- descriptor - 28

Итого: 52 байта

Product_Descriptor:

- id - 8
- name - 20

Итого: 28 байт

Vacation:

- start_date - 8
- end_date - 8
- payments - 8
- id - 8

Итого: 32 байта

Salary_Change:

- id - 8
- salary_before - 8
- salary_after - 8
- date - 8

Итого: 32 байта

Employee:

- id - 8
- name - 20
- surname - 20
- patronymic - 20
- passport - 20
- role - 15
- phone - 15
- city - 15
- employment_date - 8

- dismissal_date - 8
- salary - 8
- shifts_history - Nshifts * 8
- vacations - Nvacation * 32
- salary_changes - Nsalarychg * 32

В результате получаем следующий объем для объекта Employee: 8 + 20 + 20 + 20 + 20 + 15 + 15 + 15 + 8 + 8 + 8 + Nshifts * 8 + Nvacation * 32 + Nsalarychg * 32 = 157 + Nshifts * 8 + Nvacation * 32 + Nsalarychg * 32

Supplier:

- id - 8
- name - 20
- email - 20
- phone - 15
- products - Nsupplier-products * 52

В результате получаем следующий объем для документа Supplier: 8 + 20 + 20 + 15 + Nsupplier-products * 52 = 63 + Nsupplier-products * 52

Sale:

- id - 8
- supplier_id - 8
- product_id - 8
- branch_id - 8
- price - 8
- amount - 4
- timestamp - 8

Итого: 52 байта

Таким образом, объем данных, необходимый для хранения Nbranch, Nsale, Nsupplier:

V(Nbranch, Nsale, Nsupplier) = Nbranch * (48 + Nstocks * 68 + Nbranch-employees * (157 + Nshifts * 8 + Nvacation * 32 + Nsalarychg * 32)) + Nsale * 52 + Nsupplier * (63 + Nsupplier-products * 52)

Для приблизительной оценки будем считать, что в каждом филиале в среднем по 200(Nstocks=200) позиций товаров и работают там 10(Nbranch-employees=10) человек. При этом каждый сотрудник отработал 400(Nshifts=400) смен, взял 10(Nvacations=10) отпусков и зарплата менялась 3(Nsalarychg=3) раза. Примем среднее количество позиций от поставщика - 100(Nsupplier-products=100). Тогда, для 10(Nbranch=10) филиалов, 5(Nsupplier=5) поставщиков, 10000(Nsales=10000) продаж, будем иметь следующий объем:

10 * (48 + 200 * 68 + 10 * (157 + 400 * 8 + 10 * 32 + 3 * 32)) + 10000 * 52 + 5 * (63 + 100 * 52) = 1060095 байт = 1 мб

Направление роста модели при увеличении количества объектов каждой сущности.

Для оценки направления роста выразим, учитывая эмпирические допущения, зависимость объёма от параметров.

При добавлении филиала предполагаем найм новых сотрудников и поставку уже существующих в БД продуктов.

При добавлении поставщика предполагаем получаемое от него количество позиций - 100, распределенные на 10 филиалов (Nstocks = 10, Nsupplier-products = 100, Nbranch = 10).

Vadd(N) - объём информации, добавляемый в БД при увеличении количества объектов сущности.

Vadd(Nbranch) = Nbranch * (48 + Nstocks * 68 + Nbranch-employees * (157 + Nshifts * 8 + Nvacation * 32 + Nsalarychg * 32)) = Nbranch * (48 + 200 * 68 + 10 * (142 + 400 * 8 + 10 * 32 + 3 * 32)) = Nbranch * 51228 байт

Vadd(Nsale) = Nsale * 52 байт

Vadd(Nsupplier) = Nbranch * (68 * Nstocks) + Nsupplier * (63 + Nsupplier-products * 52) = 10 * (68 * 10) + Nsupplier * (63 + 100 * 52) = Nsupplier * 5263 + 6800 байт

При N = 1 и V = 1060095 байт имеем:

Сущность	**Увеличения объёма БД (%)**
Branches	4.83
Suppliers	1.14
Sales	0.004

Таблица 1 - Направление роста модели при увеличении количества объектов каждой сущности.

Вывод: ожидаемо, из-за большого количества вложенных документов, сильнее всего на объём БД повлияет добавление нового филиала. Однако даже такая объёмная операция имеет влияние менее 5 процентов, что оставляет простор для возможных изменений структуры с целью повышения производительности при помощи дублирования информации.

Избыточность модели (отношение между фактическим объемом модели и “чистым” объемом данных)

В Stock и Supplier не учитываем Product, в самом Product не учитываем Product_Descriptor. Получаем:

V(Nbranch, Nsale, Nsupplier) = Nbranch * (48 + Nstocks * 28 + Nemployees* (157 + Nshifts * 8 + Nvacation * 32 + Nsalarychg * 32)) + Nsale * 52 + Nsupplier* 63 = 10 * (48 + 200 * 28 + 10 * (157 + 400 * 8 + 10 * 32 + 3 * 32)) + 10000 * 52 + 5 * 63 = 954095 байт = 931 кб

Отношение “чистого” объема данных к реальному = 0.931

Запросы к модели, с помощью которых реализуются сценарии использования, указаны в приложении Г.

3.2. Аналог модели данных для SQL СУБД

Разработана схема реляционной базы данных (рисунок 2).

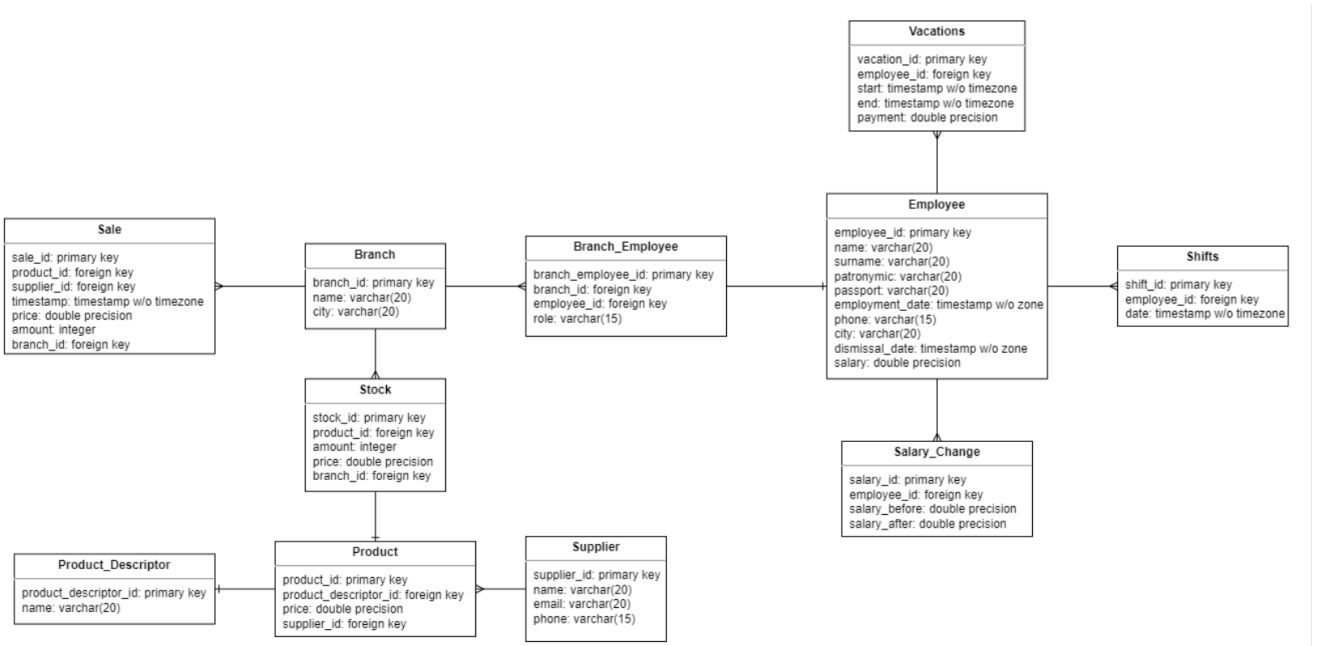

Рисунок 2 – Графическое представление реляционной базы данных

Описание сущностей

В спроектированной реляционной базе данных 11 сущностей, в полях та же информация, что в документах нереляционной базы данных:

Branch:

- branch_id - primary key
- name - varchar(20)
- city - varchar(20)

Branch_Employee:

- branch_employee_id - primary key
- branch_id - foreign key
- employee_id - foreign key
- role - varchar(15)

Stock:

- stock_id - primary key
- product_id - foreign key
- amount - int
- price - double
- branch_id - foreign key (многие к одному)

Employee:

- employee_id - primary key
- name - varchar(20)
- surname - varchar(20)
- patronymic - varchar(20)
- passport - varchar(20)
- phone - varchar(15)
- city - varchar(15)
- employment_date - timestamp w/o zone
- dismissal_date - timestamp w/o zone
- salary - double

Vacation:

- start_date - timestamp w/o zone

- employee_id - foreign key (многие к одному)
- end_date - timestamp w/o zone
- payments - double
- vacation_id - primary_key

Supplier:

- name - varchar(20)
- email - varchar(20)
- phone - varchar(15)
- supplier_id - primary key

Product_Descriptor:

- product_descriptor_id - primary key
- name - varchar(20)

Product:

- product_id - primary key
- product_descriptor_id - foreign key
- supplier_id - foreign key (многие к одному)
- price - double

Salary_Change:

- salary_id - primary key
- employee_id - foreign key (многие к одному)
- salary_before - double

- salary_after - double
- date - timestamp w/o zone

Shifts:

- shift_id - primary key
- employee_id - foreign key (многие к одному)
- date - timestamp w/o zone

Sale:

- sale_id - primary key
- supplier_id - foreign key
- product_id - foreign key
- branch_id - foreign key
- price - double
- amount - int
- timestamp - timestamp w/o zone

Оценка удельного объема информации, хранимой в модели

Единица измерения - байты. Все остальные предположения аналогичны оценке удельного объема информации в NoSQL.

Branch:

- branch_id - 8
- name - 20
- city - 20

V = 8 + 20 + 20 = 48 байт

Branch_Employee:

- branch_employee_id - 8
- branch_id - 8
- employee_id - 8
- role - 15

V = 8 + 8 + 8 + 15 = 39 байт

Stock:

- stock_id - 8
- product_id - 8
- amount - 4
- price - 8
- branch_id - 8

V = 8 + 8 + 4 + 8 + 8 = 36 байт

Product:

- product_id - 8
- product_descriptor_id - 8
- supplier_id - 8
- price - 8

V = 8 + 8 + 8 + 8 = 32 байт

Product_Descriptor:

- id - 8
- name - 20

V = 8 + 20 = 28 байт

Employee:

- employee_id - 8
- name - 20
- surname - 20
- patronymic - 20
- passport - 20
- phone - 15
- city - 15
- employment_date - 8
- dismissal_date - 8
- salary - 8

V = 8 + 20 + 20 + 20 + 20 + 15 + 15 + 8 + 8 + 8 = 142 байт

Vacation:

- start_date - 8
- employee_id - 8
- end_date - 8
- payments - 8
- vacation_id - 8

V = 8 + 8 + 8 + 8 + 8 = 40 байт

Salary_Change:

- salary_id - 8
- employee_id - 8
- salary_before - 8
- salary_after - 8
- date - 8

V = 8 * 5 = 40 байт

Shifts:

- shift_id - 8
- employee_id - 8
- date - 8

V = 8 * 3 = 24 байт

Supplier:

- id - 8
- name - 20
- email - 20
- phone - 15

V = 8 + 20 + 20 + 15 = 63 байт

Sale:

- sale_id - 8
- supplier_id - 8
- product_id - 8
- branch_id - 8
- price - 8
- amount - 4
- timestamp - 8

V = 8 * 6 + 4 = 52 байт

Используем те же предположения, что и для расчета объема NoSQL.

Итого:

V = Branch * (48 + BranchEmployee * 39 + Stock * 36) + Product * 32 + ProductDescriptor * 28 + Employee * (142 + Vacation * 40 + SalaryChange * 40 +

Shifts * 24) + Supplier * 63 + Sale * 52 = 10 * (48 + 10 * 39 + 200 * 36) + 300 * 32 + 300 * 28 + 120 * (142 + 10 * 40 + 3 * 40 + 400 * 24) = 1325820 байт = 1.3 мб

Избыточность модели (отношение между фактическим объемом модели и “чистым” объемом данных)

Не будем учитывать поля branch_id в Stock, Branch_Employee, а также employee_id в Shifts, Vacations, Salary_Change

V = Branch * (48 + BranchEmployee * 31 + Stock * 28) + Product * 32 + ProductDescriptor * 28 + Employee * (142 + Vacation * 32 + SalaryChange * 32 + Shifts * 16) + Supplier * 63 + Sale * 52 = 10 * (48 + 10 * 31 + 200 * 28) + 300 * 32 + 300 * 28 + 120 * (142 + 10 * 32 + 3 * 32 + 400 * 16) = 912540 байт = 900 кб

Отношение “чистого” объема данных к реальному = 0.68

Запросы к модели, с помощью которых реализуются сценарии использования представлены в приложении Д.

3.3. Сравнение моделей

Исследования объема памяти реляционной и нереляционной баз данных показали, что NoSQL использует занимает меньший объем, чем SQL база данных (примерно на 0.2мб). Кроме этого, при использовании нереляционной модели при запросах используется меньше коллекций, чем таблиц при реляционной модели. Даже при простых сценариях использования запросы в реляционной базе данных не обходятся без как минимум одного оператора join, и в таком случае запросы могут выполняться медленнее, чем в NoSQL, у которой преимущество в скорости за счет вложенности данных. Однако NoSQL использует дублирование данных, в результате чего запросы на удаление/изменение могут также быть затратными по времени, а неправильные запросы - привести к потери информации. В результате, обе модели имеют свои преимущества и минусы. При наличии ограничений на типы данных и

структуру хранящейся информации, следует выбрать реляционную модель. При высоких требованиях к скорости работы и нечетких требованиях к типам данных следует выбрать нереляционную модель.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Описание приложения

Backend представляет собой сервер с тремя слоями: repository, service и web. Также при запуске приложения с использование Docker Compose база данных заполняется синтетическими данными.

Frontend представляет собой web-приложение. Разработанный интерфейс представлен в приложении Б.

Пользователь имеет возможность перейти на одну из двух вкладок: Branches или Suppliers. На вкладке Branches пользователь может просматривать, фильтровать и добавлять информацию о точках продаж, сотрудниках и запасах (товарах), на вкладке Suppliers производить те же действия с поставщиками и поставляемыми позициями.

Инструкция по сборке и развёртыванию приложения представлена в приложении Б.

Снимки экрана расположены в приложении В.

4.2. Использованные технологии

Для frontend-части используется следующий стек технологий:

- язык программирования TypeScript;
- библиотека React;
- библиотека компонентов Material UI

Для backend-части используется следующий стек технологий:

- язык программирования Python;
- веб-фреймворк Sanic;

- интеграция с БД Motor: Asynchronous Python driver for MongoDB;
- база данных MongoDB.

Для запуска проекта используется Docker Compose.

ЗАКЛЮЧЕНИЕ

В ходе данной работы был разработан макет информационной системы для точек продаж спортивного питания, на основе которого реализован прототип приложения, содержащий основной функционал. Приложение позволяет просматривать, фильтровать и добавлять информацию о точках продаж, сотрудниках, товарах, поставщиках и поставляемых продуктах.

В текущей реализации имеется ряд недостатков:

- Отдельные импорт и экспорт для поставщиков и точек продаж. Необходимо добавить возможность общего импорта и экспорта всей системы данных на главной странице.
- Отсутствие возможности проведения сделки, т.е. покупки товара. Необходимо доработать текущую функциональность приложения, добавить возможность анализа сделок сети точек и расчета полученной выручки.
- Незавершённая клиентская часть. Необходимо доработать frontend-часть приложения, стилизовать ее до приятного пользователю виду, добавить недостающие возможности.
- Ограниченность в ролях. На данный момент приложение работает лишь под администратора.

Также в дальнейшем можно доработать приложение, добавив в него различных возможности для новых ролей: например, личный кабинет пользователя или расчет статистики популярности продаваемой продукции.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Sanic Documentation [Электронный ресурс] URL: https://sanic.dev/en/guide (дата обращения: 20.10.2022)
2. MongoDB Documentation [Электронный ресурс] URL: https://www.mongodb.com/docs (дата обращения: 29.09.2022)
3. Motor Tutorial [Электронный ресурс] URL: https://motor.readthedocs.io/en/stable/tutorial-asyncio.html (дата обращения: 10.11.2022)
4. Material UI Documentation [Электронный ресурс]. URL: https://mui.com/material-ui/getting-started/overview (дата обращения 15.11.2022)
5. React Documentation [Электронный ресурс]. URL: https://reactjs.org/docs/getting-started.html (дата обращения 17.11.2022)
6. Docker Documentation [Электронный ресурс]. URL: https://docs.docker.com (дата обращения: 25.11.2022)

ПРИЛОЖЕНИЕ А

МАКЕТ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Изображение слишком велико. Просьба перейти по ссылке:
https://drive.google.com/file/d/1jj4TKaDo2yhr--6JbR-FBAs-r1DbLGay/view

ПРИЛОЖЕНИЕ Б

ДОКУМЕНТАЦИЯ ПО СБОРКЕ И РАЗВЁРТЫВАНИЮ ПРИЛОЖЕНИЯ

Инструкция по сборке и развёртыванию приложений:

1. Необходимо склонировать репозиторий проекта по данной ссылке: https://github.com/moevm/nosql2h22-sports-nutrition
2. Для запуска всех сервисов необходимо в корневой директории запустить Docker Compose с помощью команды *docker-compose up*.
3. После завершения локального развёртывания приложения необходимо перейти по ссылке: http://localhost:8080. Сервер будет запущен по адресу http://localhost:8008.

Для ознакомления с API сервера предлагается перейти в Postman проекта, указанный в README.md в репозитории проекта.

ПРИЛОЖЕНИЕ В
СНИМКИ ЭКРАНА ПРИЛОЖЕНИЯ

Рисунок 3 - Стартовая страница

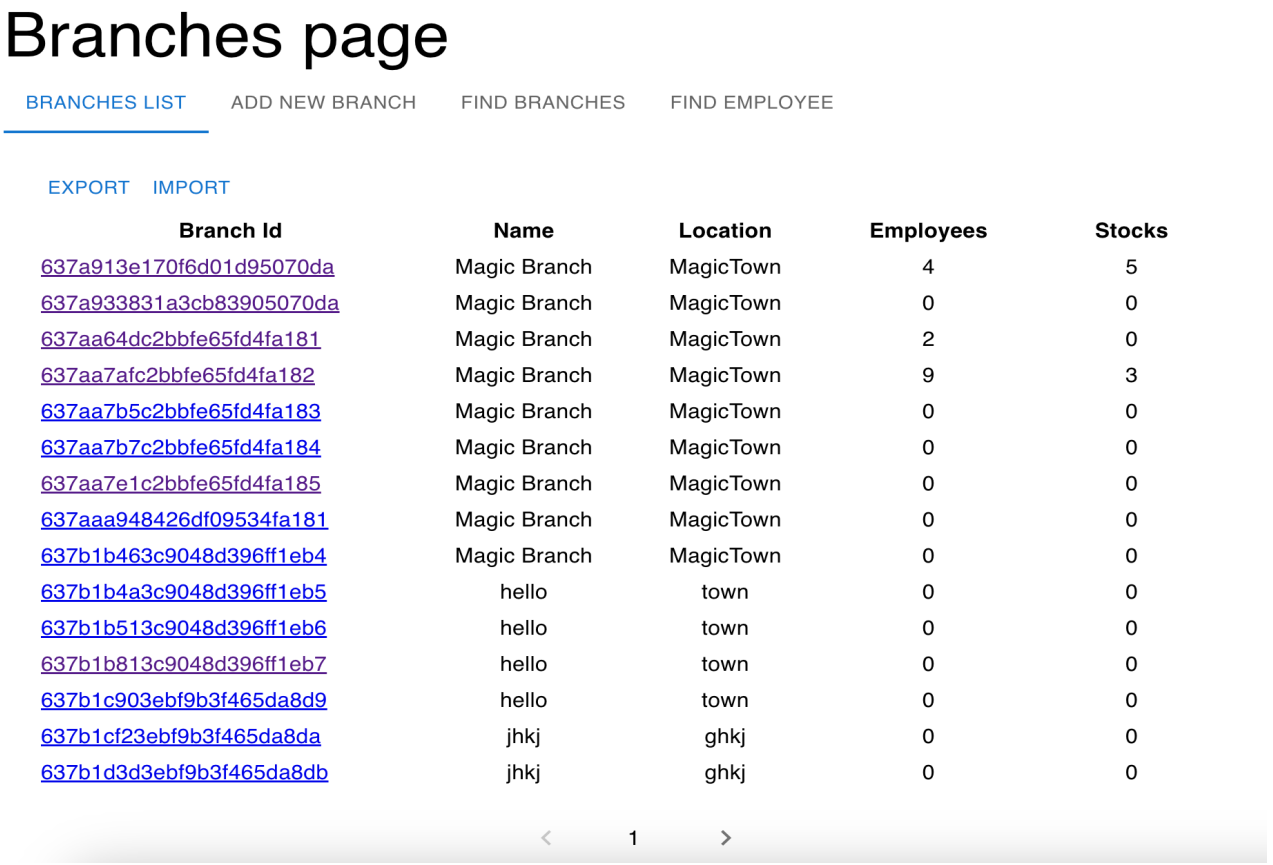

Рисунок 4 - Страница Branches. Пагинация точек продаж

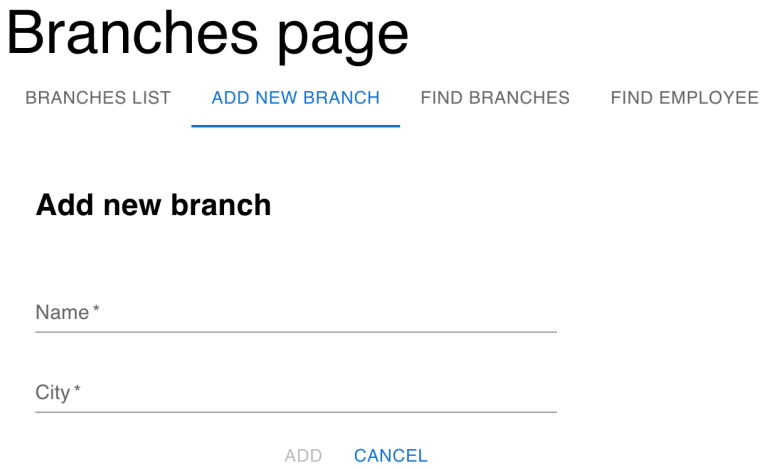

Рисунок 5 - Страница Branches. Добавление новой точки продаж

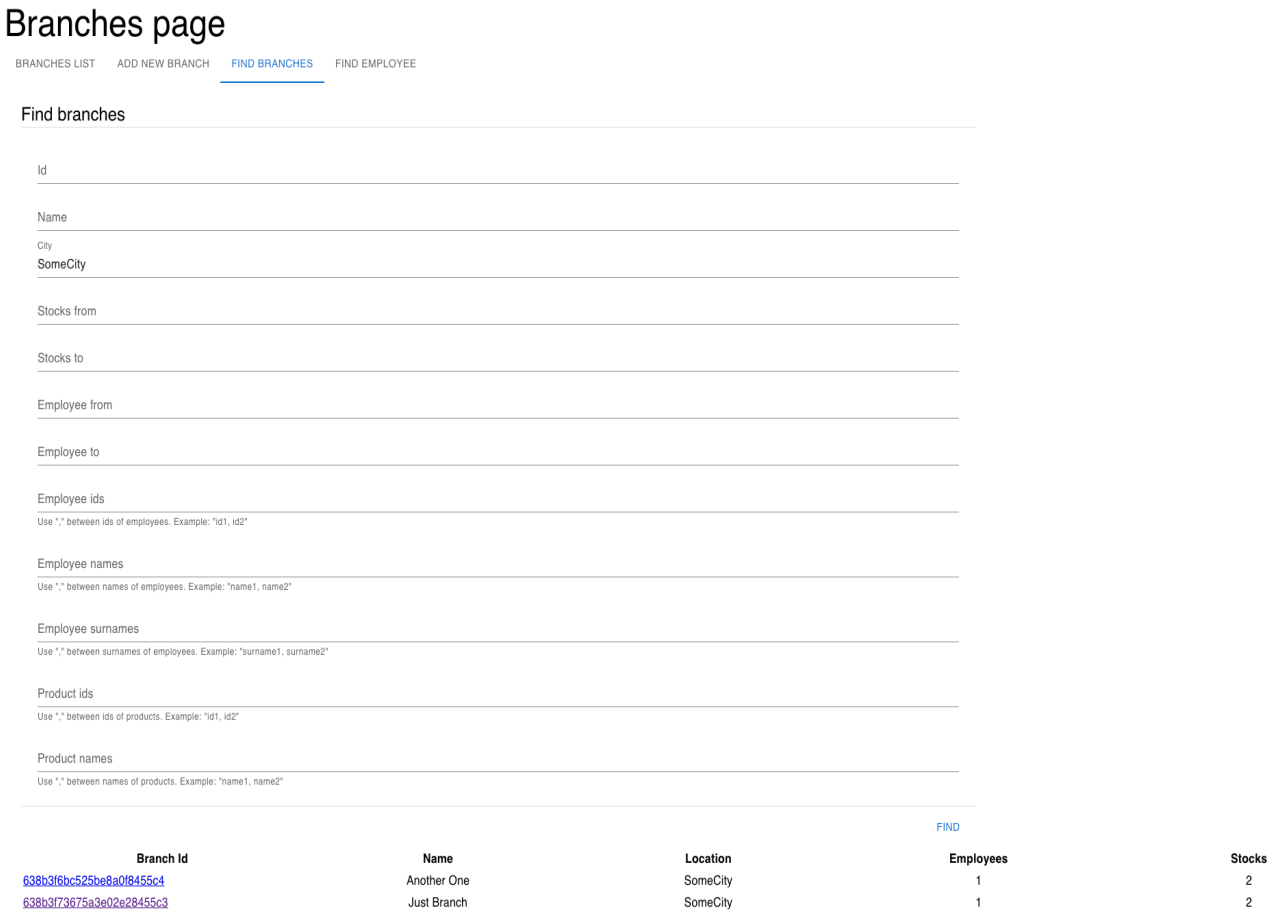

Рисунок 6 - Страница Branches. Поиск точек продаж по фильтрам

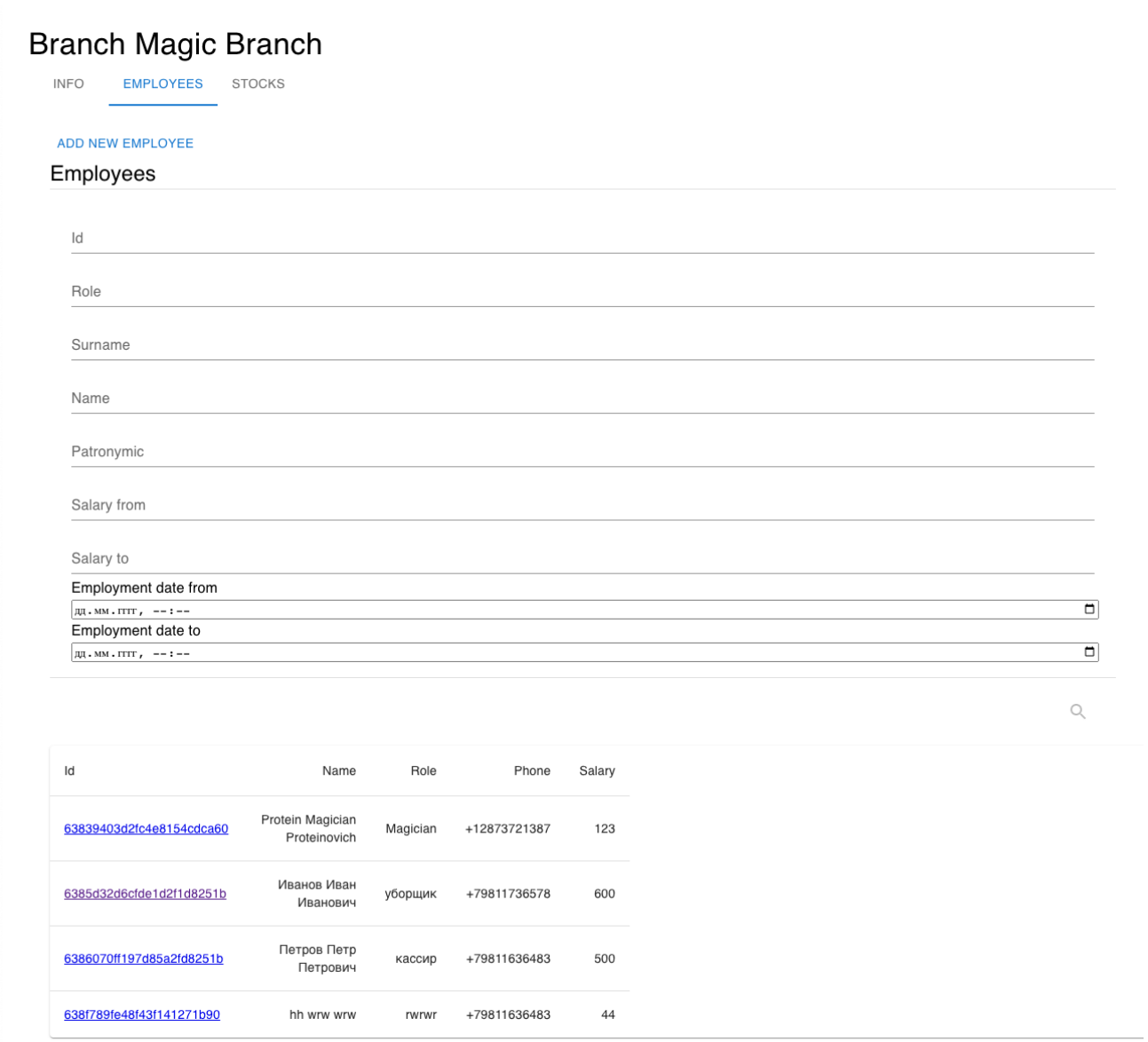

Рисунок 7 - Страница конкретной точки с сотрудниками

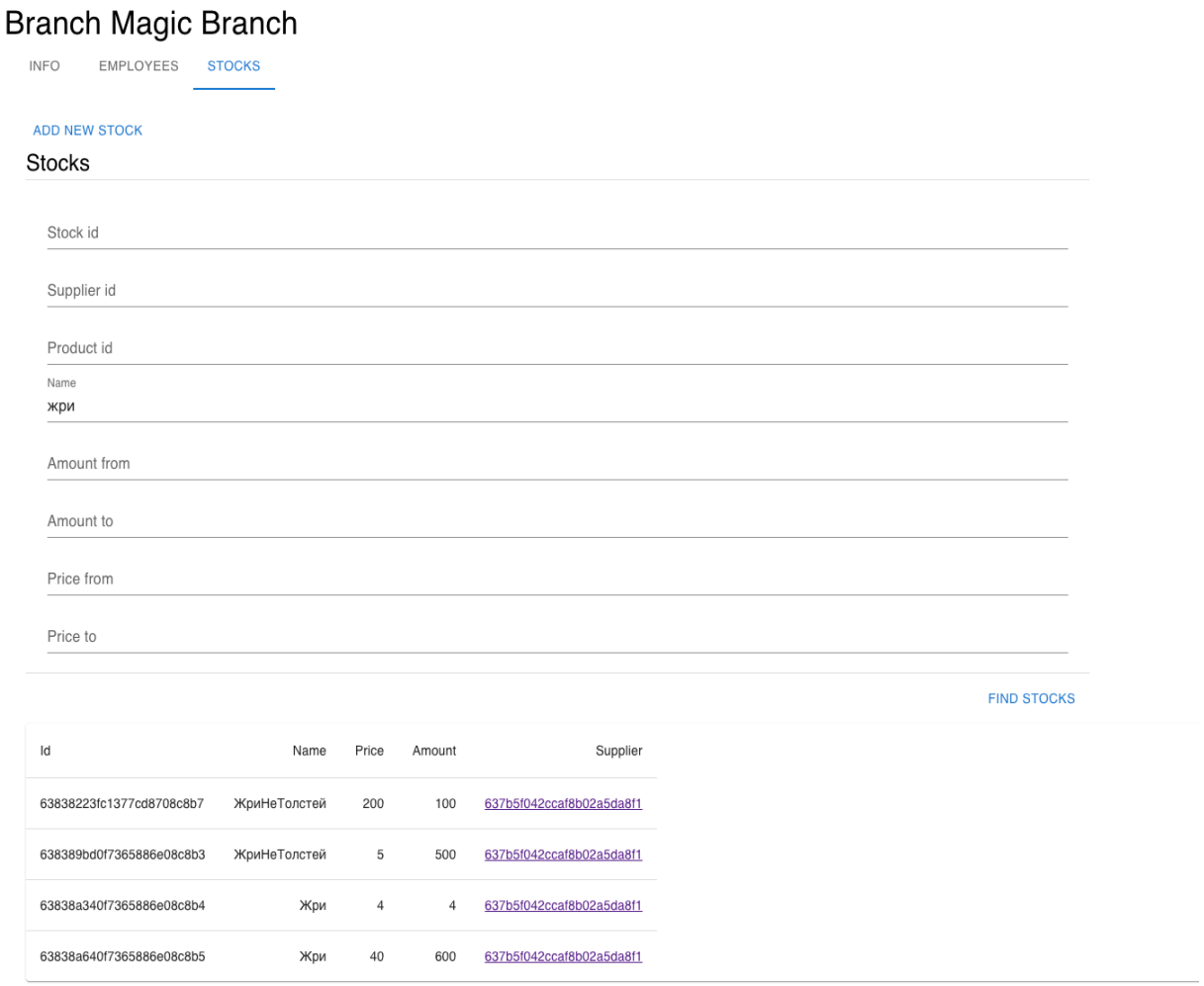

Рисунок 8 - Страница конкретной точки, фильтр по запасам

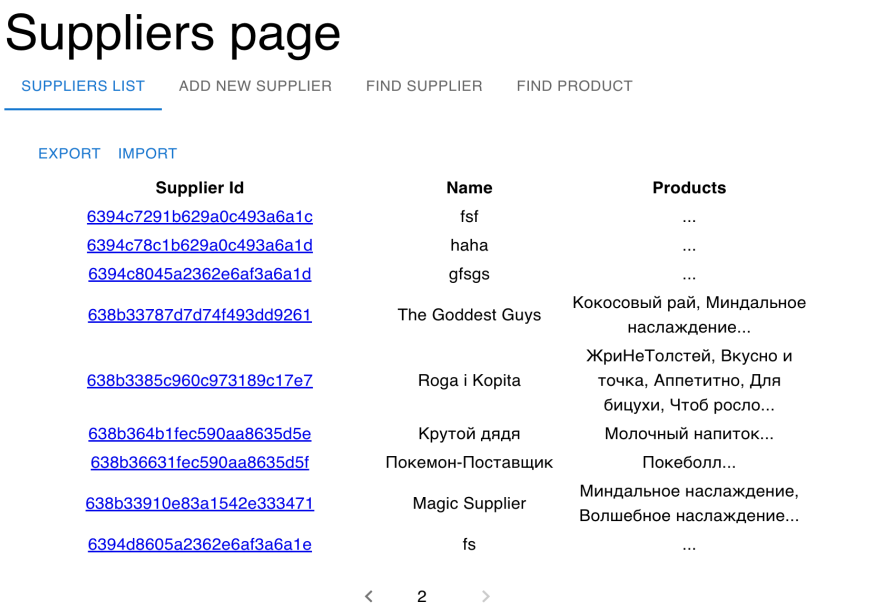

Рисунок 9 - Страница Suppliers. Пагинация поставщиков

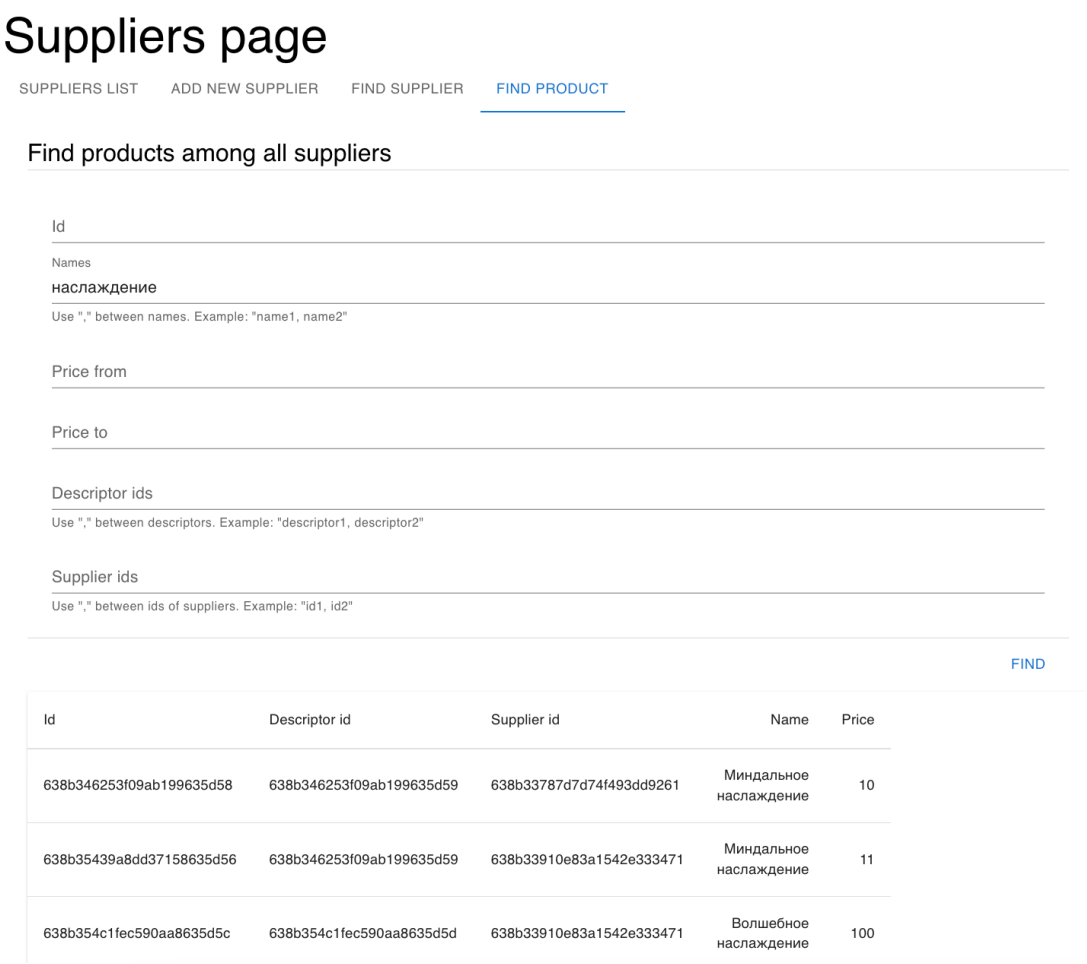

Рисунок 10 - Страница Suppliers. Поиск продукта среди всех поставщиков

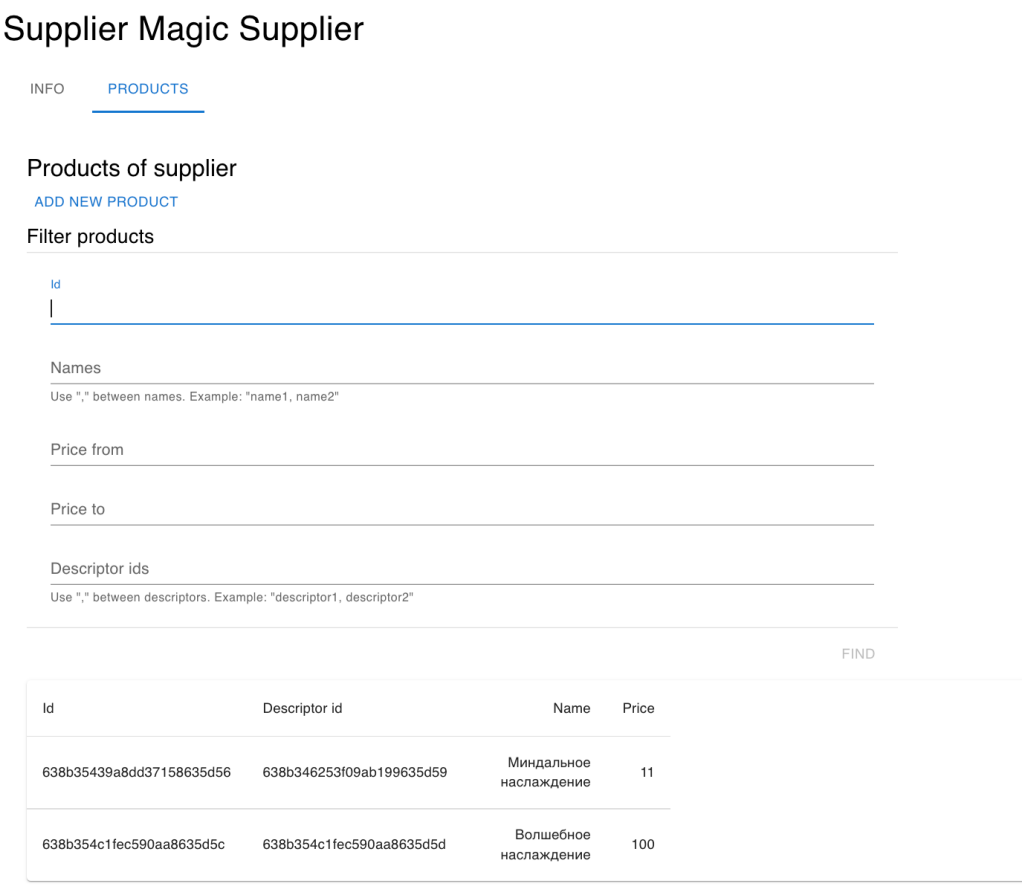

Рисунок 11 - Страница конкретного поставщика с продуктами

ПРИЛОЖЕНИЕ Г
ЗАПРОСЫ К NOSQL МОДЕЛИ, С ПОМОЩЬЮ КОТОРЫХ РЕАЛИЗУЮТСЯ СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1. Получение работников по имени филиала

```
db.branches.find({
  "name": <branch_name>
},
{
  employees: 1
})
```

Задействованных коллекций: 1

2. Получение по поставщику его продуктов

```
db.suppliers.find({
  "name": <supplier_name>
},
{
  "products": 1
})
```

Задействованных коллекций: 1

3. Поиск работника по минимальной дате трудоустройства и смене за определённый день

```
db.collection.find({
  "$and": [
    {
      "employees.employment_date": {
```

```
        "$gte": ISODate(<date>)
      }
    },
    {
      "employees.shifts_history": ISODate(<date>)
    }
  ]
},
{
  "employees.$": 1
})
```

Задействованных коллекций: 1

Пример хранения

Коллекция Branch:

```
[
  {
    "_id": ObjectId("5a934e000102030405000000"),
    "city": "MagicTown",
    "employees": [
      {
        "name": "Magician",
        "surname": "Protein",
        "patronymic": "Proteinovich",
```

```
"passport": "yes",
"phone": "12873721387",
"city": "no",
"employment_date": ISODate("2013-10-02T01:11:18.965Z"),
"dissmisal_date": ISODate("2015-10-02T01:11:18.965Z"),
"salary": 123,
"shifts_history": [
  ISODate("2013-10-02T01:11:18.965Z"),
  ISODate("2013-11-02T01:11:18.965Z")
],
"vacation_history": [
  {
    "payments": 100,
    "start_date": ISODate("2013-12-02T01:11:18.965Z"),
    "end_date": ISODate("2015-10-02T01:11:18.965Z")
  }
],
"salary_change_history": [
  {
    "salary_before": 100,
    "salary_after": -1,
    "date": ISODate("2013-11-02T01:11:18.965Z")
  }
],
```

```
      "role": "Magician"
    },
    {
      "name": "Second",
      "surname": "Magician",
      "patronymic": "Test",
      "passport": "yes",
      "phone": "34453345",
      "city": "no",
      "employment_date": ISODate("2013-11-02T01:11:18.965Z"),
      "dissmisal_date": ISODate("2016-10-02T01:11:18.965Z"),
      "salary": 123,
      "shifts_history": [
        ISODate("2013-12-02T01:11:18.965Z"),
        ISODate("2013-12-02T01:11:18.965Z")
      ],
      "vacation_history": [],
      "salary_change_history": [],
      "role": "Magician"
    }
  ],
  "name": "Magic Branch",
  "stocks": [
    {
```

```
        "amount": 100,
        "price": 1.2,
        "product": {
          "descriptor": {
            "name": "EatWontBeFat"
          },
          "price": 0.1,
          "supplier_id": ObjectId("635c08e696a1cf869f76566d")
        }
      },
      {
        "amount": 23,
        "price": 2.3,
        "product": {
          "descriptor": {
            "name": "MagicThing"
          },
          "price": 0.01,
          "supplier_id": ObjectId("635c08e696b1cf869f76566d")
        }
      }
    ]
  }
]
```

Коллекция Suppliers:

```
[
  {
    "id": 1,
    "name": "MagicianSupplier",
    "email": "magic@supplier.food",
    "phone": "12371231872",
    "products": [
      {
        "supplier_id": 1,
        "descriptor": {
          "name": "MagicProtein"
        },
        "price": 100
      }
    ]
  }
]
```

Коллекция Employees

```
[
  {
    "name": "Magician",
    "surname": "Protein",
    "patronymic": "Proteinovich",
```

```
"passport": "yes",
"phone": "12873721387",
"role": "Magician",
"city": "no",
"employment_date": ISODate("2013-10-02T01:11:18.965Z"),
 "dissmisal_date": ISODate("2015-10-02T01:11:18.965Z"),
"salary": 123,
"shifts_history": [
  ISODate("2013-10-02T01:11:18.965Z"),
  ISODate("2013-11-02T01:11:18.965Z")
],
"vacation_history": [
  {
    "payments": 100,
    "start_date": ISODate("2013-12-02T01:11:18.965Z"),
    "end_date": ISODate("2015-10-02T01:11:18.965Z")
  }
 ],
"salary_change_history": [
  {
    "salary_before": 100,
    "salary_after": -1,
    "date": ISODate("2013-11-02T01:11:18.965Z")
  }
```

```
    ]
  }
]
```

ПРИЛОЖЕНИЕ Д
ЗАПРОСЫ К РЕЛЯЦИОННОЙ МОДЕЛИ, С ПОМОЩЬЮ КОТОРЫХ РЕАЛИЗУЮТСЯ СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1. Получение работников по имени филиала

```
select
  be.role,
  e.name
from
  branch b
  left join branch_employee be using(branch_id)
  left join employee e using(employee_id)
where b.name = <branch_name>;
```

Задействованных сущностей: 3, оператор join используется 2 раза

2. Получение по поставщику его продуктов

```
select
  s.name,
  pd.name
from
  supplier s
  left join product p using(supplier_id)
  left join product_descriptor pd using(product_descriptor_id)
where
  supplier_name = 'supplier name'
```

Задействованных сущностей: 3, оператор join используется 2 раза

3. Поиск работника по минимальной дате трудоустройства и смене за определённый день

```
select
```

```
  e.name
from
  employee e
  left join shifts s using(employee_id)
where
  e.employment_date >= "employement_date"::date
  and s.date = "shift_date"
```

Задействованных сущностей: 2, оператор join используется 1 раз