



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ (ИУ)

КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПISKA

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

***Разработка базы данных внутреннего портала для
сотрудников предприятия***

Студент группы ИУ7-62Б

_____ Княжев А. В.

Руководитель курсовой работы

_____ Кострицкий А. С.

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 36 с., 9 рис., 4 табл., 26 источн., 4 прил.

Ключевые слова: база данных, интранет, социальная сеть, кеширование, Redis, PostgreSQL, сериализация.

В данной работе разработана база данных внутреннего портала для сотрудников предприятия. Для доступа к базе данных было реализовано приложение, работающее по модели клиент-сервер. Серверная часть реализована на языке Go, клиентская — на языке JavaScript с использованием фреймворка Vue.

Проведено исследование влияния кеширования на скорость поиска, а также влияние метода сериализации структур на скорость поиска и объем потребляемой закешированными данными памяти.

СОДЕРЖАНИЕ

РЕФЕРАТ	2
ОПРЕДЕЛЕНИЯ	5
ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	6
ВВЕДЕНИЕ	7
1 Аналитический раздел	8
1.1 Анализ предметной области	8
1.2 Сравнение с аналогичными решениями	8
1.3 Формализация задачи	9
1.4 Виды СУБД по модели данных	12
2 Конструкторский раздел	15
2.1 Описание сущностей базы данных	15
2.2 Разработка функции базы данных	17
2.3 Диаграмма сущностей базы данных	18
2.4 Ролевая модель	19
3 Технологический раздел	20
3.1 Выбор средств разработки	20
3.2 Реализация ролевой модели	20
3.3 Реализация программного интерфейса	21
3.4 Тестирование	24
3.5 Интерфейс	25
4 Исследовательский раздел	27
4.1 Исследование скорости работы поиска с кешированием и без .	27
4.2 Исследование скорости работы поиска с использованием различных методов сериализации при использовании кеширования в Redis	29
4.3 Исследование объема занимаемой закешированными данными памяти при использовании различных алгоритмов сериализации	31

ЗАКЛЮЧЕНИЕ	33
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	36

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

Система управления базами данных — это набор программ, которые управляют структурой БД и контролируют доступ к данным, хранящимся в БД [1].

Сериализация — процесс перевода структуры в текстовый формат, предназначенный для обмена информацией между существенно отличающимися друг от друга программными системами. [2]

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

БД — База Данных.

СУБД — Система Управления Базами Данных.

JSON — JavaScript Object Notation.

CBOR — Concise Binary Object Representation.

SQL — Structured Query Language.

ER — Entity Relationship.

ВВЕДЕНИЕ

Для организаций довольно актуальными являются проблемы коммуникации сотрудников, понимания иерархии, отделов, в которых распределены сотрудники [3]. Для решения таких проблем существуют внутренние порталы для предприятий.

Цель работы

Разработка базы данных внутреннего портала для сотрудников предприятия, а также приложения, позволяющего взаимодействовать с ней.

Задачи работы

- Формализовать задачу.
- Спроектировать базу данных.
- Выбрать СУБД.
- Реализовать базу данных, заполнить ее.
- Реализовать приложения для доступа к базе данных.

1 Аналитический раздел

1.1 Анализ предметной области

С увеличением числа сотрудников компании часто возникают проблемы внутренней коммуникации сотрудников, получения информации о коллегах и автоматизации процессов рекрутмента [4]. Для решения этих задач существуют внутренние порталы для сотрудников. Обычно они представляют функционал, схожий с соцсетями: отображение профилей пользователей, и др.

В основном, внутренние порталы для сотрудников представляют собой закрытые решения [5], которые разрабатываются в рамках одной компании [6], и не распространяются во внешний мир вследствие политики конфиденциальности [7].

1.2 Сравнение с аналогичными решениями

В данной части проводится сравнение предлагаемого решения с аналогами — наиболее активно используемыми и разрабатываемыми известными внутренними порталами для сотрудников [6].

Критерии сравнения

В качестве критериев сравнения решений были выбраны основные элементы функционала:

1. возможность просматривать профили пользователей;
2. хранение информации об иерархии сотрудников;
3. возможность хранить информацию о команде сотрудника;
4. отсутствие платы за использование;
5. открытость решения для использования.

Сравнение

В таблице 1.1 приведено сравнение решений для внутреннего портала для сотрудников организации.

Таблица 1.1 – Сравнение существующих решений с предложенным

Решение	1	2	3	4	5
Инtranет VK [3]	+	+	–	+	–
Avito People [4]	+	+	+	+	–
Битрикс24 [8]	+	+	–	–	+
Предлагаемое решение	+	+	+	+	+

1.3 Формализация задачи

Решение должно представлять собой портал, в котором будет производиться регистрация сотрудника менеджером по персоналу. Сотрудник сможет просматривать профили других сотрудников, производить поиск по организации.

Каждый сотрудник состоит в некоторой иерархии: у него есть руководитель и подразделение, в котором он находится. Кроме того, роль сотрудника может не совпадать с его положением в этой иерархии [4]. Например, по документам он может находиться в департаменте разработки, а фактически находиться в команде какого-либо проекта. Решение должно учитывать эту особенность.

Сотрудник может редактировать основные данные профиля: фотографию, описание профиля, информацию об отпуске, при этом такую информацию, как имя, должность и положение в иерархии компании может изменять только рекрутер.

Формализация сценариев использования

В рамках решения должно существовать три вида пользователей:

1. Сотрудник — сотрудник предприятия. Имеет базовые возможности по изменению части информации профиля, поиску среди пользователей системы.
2. Рекрутер — менеджер по персоналу. Может создавать профили сотрудников, удалять их, изменять базовую информацию, такую как имя, должность.

3. Администратор — главная роль в системе. Может делать то же, что может рекрутер. Кроме того, может редактировать базовую структуру организации, и управлять другими администраторами.

На рис. 1.1 изображена диаграмма сценариев использования предлагаемого решения.

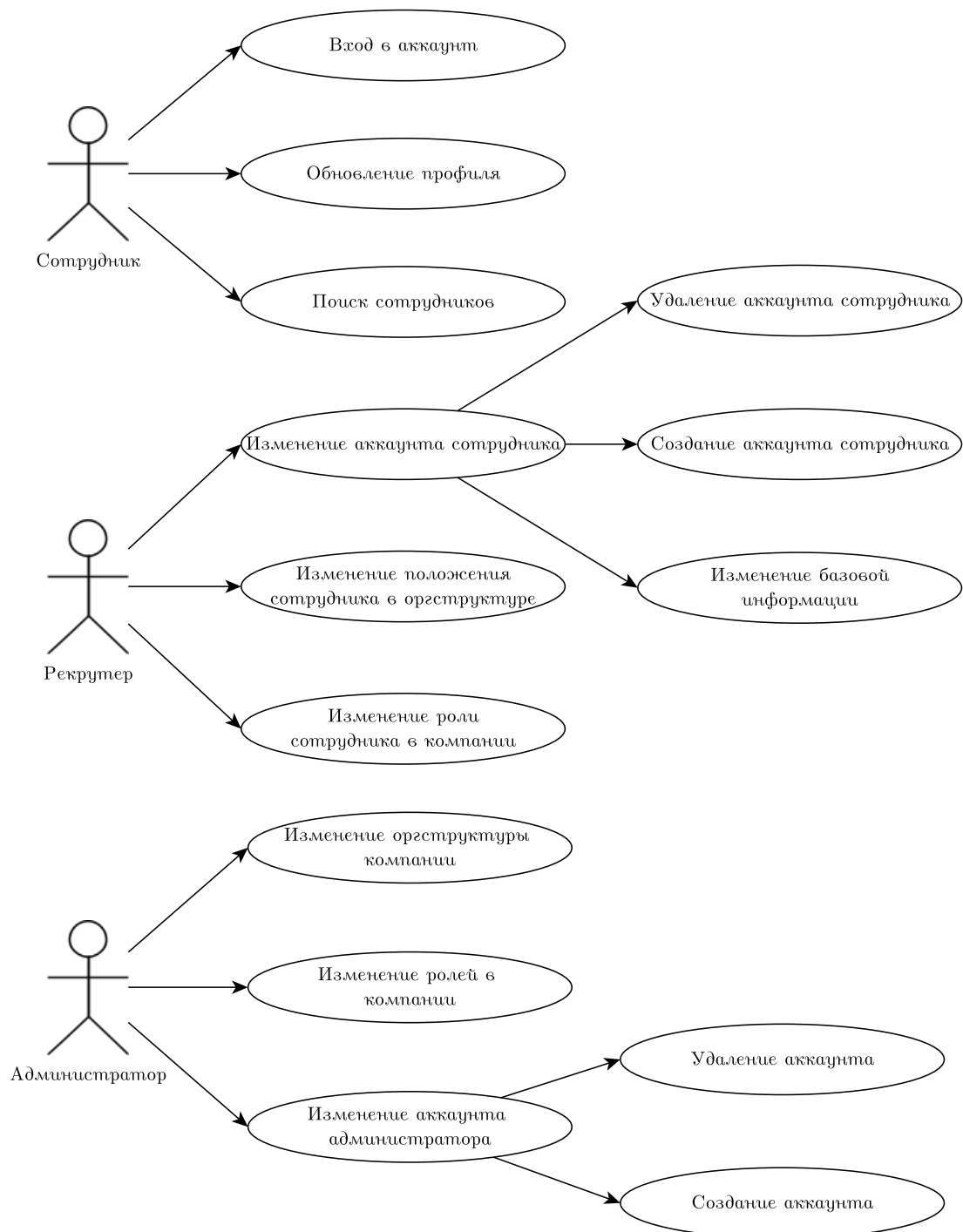


Рисунок 1.1 – Диаграмма сценариев использования (начало)

Формализация данных

Основные сущности, которые должны содержаться в проектируемой базе данных:

- сотрудник;
- команда — фактическая команда, в которой работает сотрудник;
- подписка;
- отпуск;
- отдел — официальный департамент, в который трудоустроен сотрудник.

На рис. 1.2 изображена ER-диаграмма сущностей проектируемой базы данных в нотации Чена.

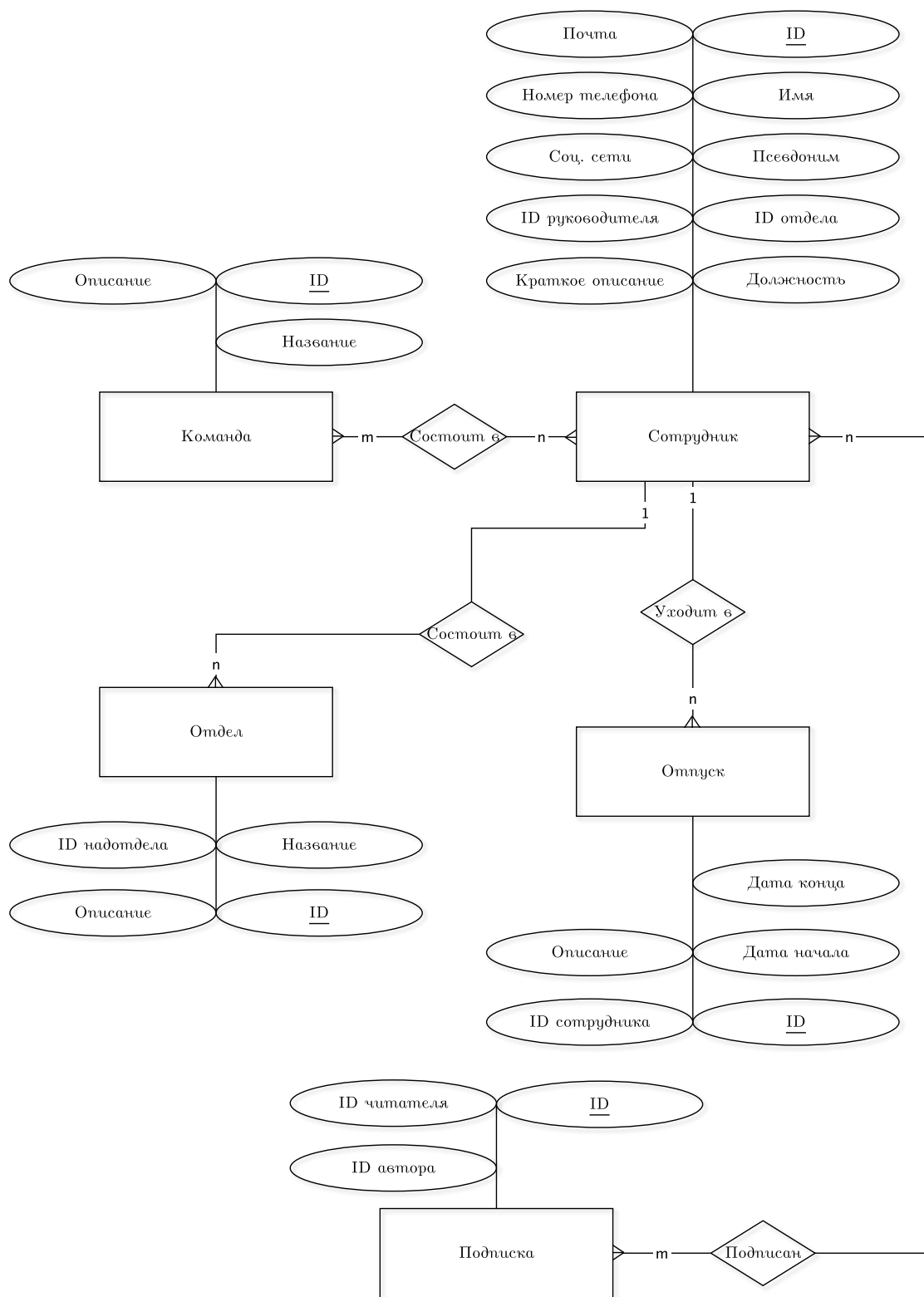


Рисунок 1.2 – ER-диаграмма сущностей в нотации Чена

1.4 Виды СУБД по модели данных

База данных представляет собой набор связанных данных некоторой предметной области, поэтому для ее организации важно определиться с

моделью данных. Модель данных представляет собой логическую структуру данных, хранящихся в базе данных [9]. На данный момент, в качестве основных моделей данных выделяют дореляционные, реляционные и нереляционные.

Дореляционные

Дореляционные базы данных не основываются на абстрактных моделях, доступ осуществляется на уровне записей [10].

Среди дореляционных моделей можно выделить [11]:

- Иерархическая — представлена в базе данных в виде дерева [12].
- Сетевая — является расширением иерархической БД. У потомка может быть больше одного родителя. Такая модель поддерживает отношение «многие-ко-многим» [12].

Среди особенностей дореляционных моделей можно отметить сложность использования и зависимость прикладных программ от физической реализации СУБД [10].

Реляционные

Реляционная база данных — совокупность отношений, содержащих всю информацию, которая должна храниться в базе данных [13]. В основном воспринимаются как базы, в которых данные хранятся в двумерных таблицах, а установленная между ними связь позволяет избегать повторения данных [14].

Таблица реляционной базы данных устроена следующим образом. Столбец таблицы представляет собой поле — значение атрибута отношения. Строка представляет собой конкретную запись, кортеж этого отношения. Такая структура позволяет создать наиболее простое и понятное пользователю представление базы данных, при такой организации данные хорошо структурированы [10].

Нереляционные

Нереляционные модели убирают строгие связи между данными в системе. Нереляционные базы данных бывают [10]:

- колоночные;

- документоориентированные;
- вида ключ-значение;
- графовые;
- с другие.

Основными отличиями от реляционной модели являются меньшая структуризация, отсутствие стандартизации [12].

В данном разделе была проведена формализация задачи: рассмотрена предметная область, формализованы данные и сценарии использования. Проведено сравнение с конкурентами по основным критериям.

Кроме того, рассмотрены виды СУБД по модели данных. В связи с отсутствием необходимости специфичной обработки данных и необходимостью структурировать данные, была выбрана реляционная модель.

2 Конструкторский раздел

2.1 Описание сущностей базы данных

При описании сущностей базы данных, после названия поле в скобках находится описание типа поля.

Сотрудник

`Employee` — хранит информацию о сотруднике организации. Содержит следующие поля:

- `uuid (integer)` — идентификатор сотрудника. Не может быть пустым.
- `name (text)` — имя сотрудника. Не может быть пустым.
- `nickname (text)` — псевдоним сотрудника. Не может быть пустым, длиной не менее четырех символов.
- `department_uuid (integer)` — идентификатор департамента, в котором состоит сотрудник. Не может быть пустым.
- `position (text)` — должность. Не может быть пустым.
- `email (text)` — адрес электронной почты. Не может быть пустым.
- `phone (text)` — номер телефона. Не может быть пустым.
- `social (text)` — контакты сотрудника в социальных сетях.
- `boss_uuid (integer)` — идентификатор непосредственного руководителя сотрудника.
- `description (text)` — описание профиля сотрудника.

Команда

`Team` — хранит информацию о команде. Содержит следующие поля:

- `uuid (integer)` — идентификатор команды. Не может быть пустым.
- `name (text)` — название команды. Не может быть пустым.
- `description (text)` — описание команды.

Отдел

Department — хранит информацию об отделе, юридической единицы деления организации. Содержит следующие поля:

- **uuid (integer)** — идентификатор отдела. Не может быть пустым.
- **parent_uuid (integer)** — идентификатор родительского отдела.
- **name (text)** — название отдела. Не может быть пустым.
- **description (text)** — описание отдела.

Отпуск

Vacation — хранит информацию об отпуске сотрудника. Содержит следующие поля:

- **uuid (integer)** — идентификатор отпуска. Не может быть пустым.
- **employee_uuid (integer)** — идентификатор сотрудника, уходящего в отпуск. Не может быть пустым.
- **start (timestamp)** — время начала отпуска. Не может быть пустым.
- **end (timestamp)** — время конца отпуска. Не может быть пустым.
- **description (text)** — описание отпуска.

Подписка

Subscription — хранит информацию о подписке сотрудника на другого сотрудника. Содержит следующие поля:

- **uuid (integer)** — идентификатор подписки. Не может быть пустым.
- **author_uuid (integer)** — идентификатор автора, то есть сотрудника, на которого происходит подписка. Не может быть пустым.
- **subscriber_uuid (integer)** — идентификатор подписчика. Не может быть пустым.

2.2 Разработка функции базы данных

Функция будет осуществлять получение полного положения сотрудника в организационной структуре компании, включая все вложенные отделы.

Алгоритм работы функции будет следующим:

1. Добавить текущий отдел к результату.
2. Если родительский отдел пуст, то возврат.
3. Если родительский отдел не пуст, то рекурсивно вызвать функцию поиска полного положения сотрудника в компании с аргументом — родительским отделом.

2.3 Диаграмма сущностей базы данных

На рисунке 2.1 изображена диаграмма сущностей базы данных в нотации Мартина.

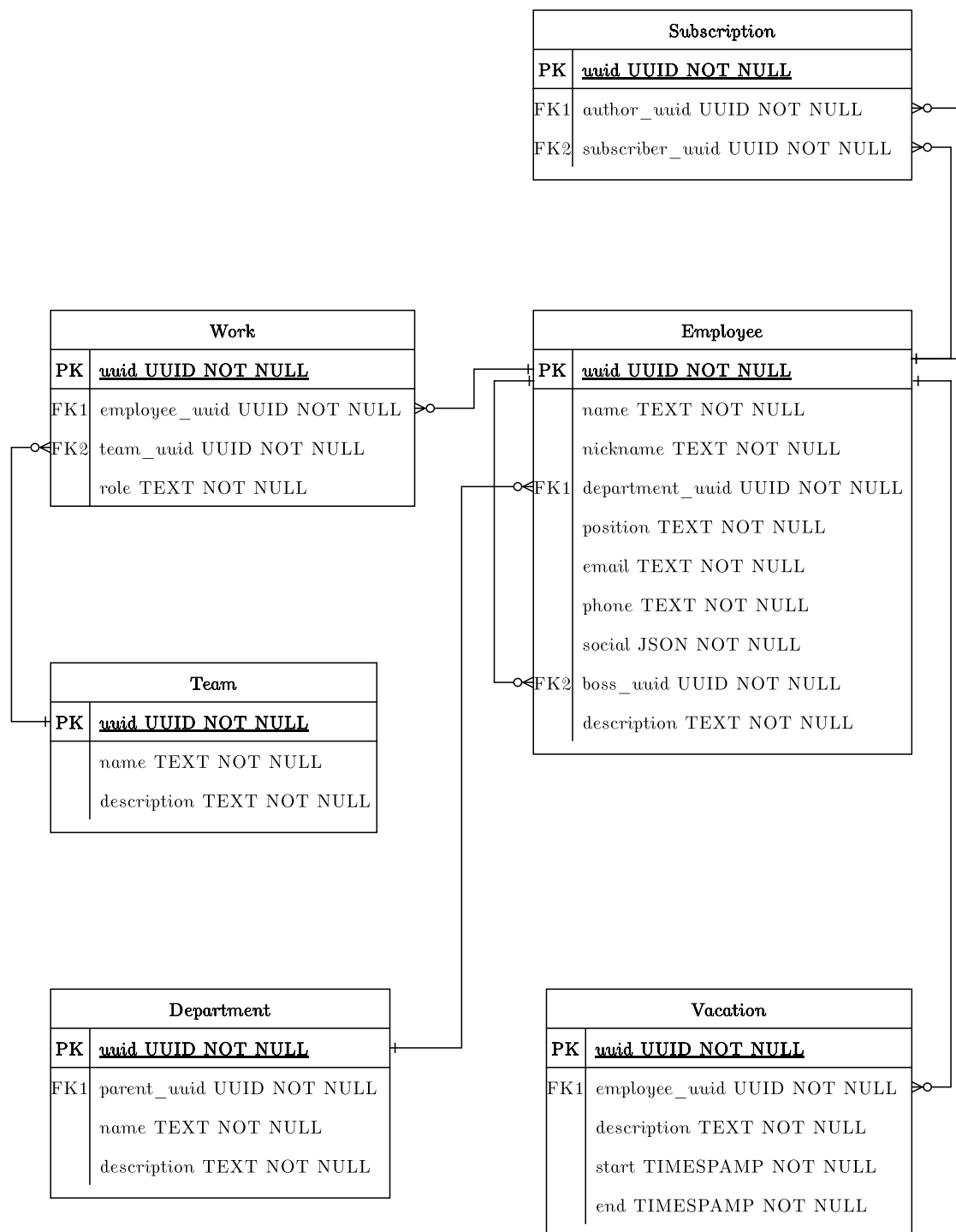


Рисунок 2.1 – Диаграмма сущностей в нотации Мартина

2.4 Ролевая модель

В базе данных должно существовать три роли — административная, сотрудника и рекрутера. Каждая роль наделена своим набором прав на внутренние структуры.

— Сотрудник:

- Subscription — SELECT, INSERT, DELETE.
- Team — SELECT.
- Employee — SELECT, UPDATE.
- Department — SELECT.
- Vacation — SELECT, INSERT.

— Рекрутер:

- Все права обычного сотрудника.
- Employee — INSERT.

— Администратор:

- Все права рекрутера.
- Team — INSERT, UPDATE, DELETE.
- Employee — DELETE.
- Department — INSERT, UPDATE, DELETE.

Каждый пользовательский сценарий должен использовать только соответствующую ему роль. Например, для просмотра информации о сотруднике необходимо использовать роль «Сотрудник».

3 Технологический раздел

3.1 Выбор средств разработки

Приложение разрабатывалось с использованием схемы взаимодействия клиент-сервер. В качестве языка разработки серверной части был выбран язык программирования Go [15], так как он имеет достаточный для выполнения работы набор библиотек, является строго типизированным и компилируемым. Кроме того, компилятор языка позволяет производить сборку программы под различные платформы.

В качестве языка разработки клиентской части был выбран язык программирования JavaScript и фреймворк Vue [16]. Выбор вызван достаточным набором функционала и библиотек для разработки браузерных web-приложений. Разработка клиентской части производилась в соответствии с паттерном проектирования MVVM.

В качестве системы управления базами данных была выбрана PostgreSQL [17]. Она предоставляет реляционную модель. Язык запросов имеет достаточный функционал для выполнения запросов, необходимых в рамках данной работы. Кроме того, PostgreSQL поддерживает систему ролей, позволяющую реализовать ролевую модель, описанную выше.

В качестве языка спецификации контракта между клиентской и серверной частями был использована спецификация OpenAPI [18]. Эта спецификация позволяет в полной мере реализовать необходимое в рамках поставленной задачи взаимодействие.

3.2 Реализация ролевой модели

Ролевая модель реализована на уровне бизнес-логики приложения, а также на уровне базы данных. Для второго варианта использована система ролей PostgreSQL. Реализованы три роли, для каждой роли создан пароль для доступа к базе данных. С использованием полученных в результате создания ролей параметров входа, приложение создает три подключения к базе данных, для каждой роли. Таким образом, каждый запрос использует свое подключение к базе данных с использованием необходимых для данного запроса прав. Скрипт создания ролей приведен в листинге 3.1.

Листинг 3.1 – Скрипт создания ролей для СУБД PostgreSQL

```
1 create role employer with login password 'employer_password';
2 grant select,update on employee to employer;
3 grant select on team to employer;
4 grant select, insert, delete on subscription to employer;
5 grant select on work to employer;
6 grant select on department to employer;
7 grant select, insert on vacation to employer;
8
9 create role recruiter with login password 'recruiter_password';
10 grant employer to recruiter;
11 grant insert on employee to recruiter;
12 grant insert,delete on work to recruiter;
13
14 create role admin with login password 'admin_password';
15 grant recruiter to admin;
16 grant delete on employee to admin;
17 grant insert,update,delete on team to recruiter;
18 grant insert,update,delete on department to recruiter;
```

3.3 Реализация программного интерфейса

Для описания контракта взаимодействия клиента и сервера был использован формат спецификации OpenAPI. Фрагмент файла спецификации приведен в листинге 3.2.

Листинг 3.2 – Вид файла спецификации OpenAPI

```
1 paths:
2   /login:
3     post:
4       operationId: Login
5       requestBody:
6         required: true
7         content:
8           application/json:
9             schema:
10               $ref: "#/components/schemas/LoginRequest"
11       responses:
12         "200":
13           description: Login info.
14           content:
```

```

15         application/json:
16             schema:
17                 $ref: '#/components/schemas/LoginResponse'
18         default:
19             description: Error response.
20             content:
21                 application/json:
22                     schema:
23                         $ref: '#/components/schemas/Error'
24 components:
25     schemas:
26         LoginRequest:
27             type: object
28             properties:
29                 login:
30                     type: string
31                 password:
32                     type: string
33
34         LoginResponse:
35             type: object
36             properties:
37                 token:
38                     type: string

```

На основе данного файла генерируются модели и заглушки сервера. Чтобы реализовать обработку http запросов по указанным путям достаточно реализовать заглушки. Основные запросы к серверу приведены ниже.

- POST /departments — создание нового департамента.
- GET /departments — получение списка департаментов.
- PUT /departments/id — изменение департамента.
- DELETE /departments/id — удаление департамента.
- GET /departments/id — получение департамента по идентификатору.
- POST /employee — создание нового пользователя.
- GET /employee — получение списка пользователей.

- PUT /employee/id — изменение пользователя.
- DELETE /employee/id/fire — увольнение пользователя. Пользователь лишается возможности пользоваться системой и помечается как деактивированный. При этом, его профиль можно просмотреть.
- DELETE /employee/id/delete — удаление пользователя из системы.
- GET /employee/id/id — получение пользователя по идентификатору.
- GET /employee/nickname/nickname — получение пользователя по псевдониму.
- GET /department/id/breadcrumbs — получение пути до текущего департамента от корневого в дереве иерархии департаментов.
- POST /login — вход в систему по логину и паролю.
- PUT /employee/id/subscribe — подписка на пользователя.
- GET /employee/id/subscribe — получение статуса подписки на пользователя.
- POST /me/vacations — создание отпуска.
- GET /employee/id/vacations — получение списка отпусков пользователя.
- GET /me — получение пользователя по токёну.
- POST /teams — создание новой команды.
- GET /teams — получение списка команд.
- GET /teams/id — получение команды по идентификатору.
- PUT /teams/id — изменение команды.
- DELETE /teams/id — удаление команды.
- POST /search — поиск сотрудников, департаментов и команд по заданной подстроке.

3.4 Тестирование

Для проверки корректности работы бизнес-логики были реализован набор модульных тестов. Для тестирования использовалась стандартная библиотека языка программирования Go. Для методов получения данных при тестировании были сгенерированы заглушки. Пример теста приведен в листинге 3.3.

Листинг 3.3 – Пример модульного теста

```
1 func TestDepartment_GetDepartments(t *testing.T) {
2     ...
3     tests := []struct {
4         ...
5     }{
6         {
7             name: "valid breadcrumbs authorized",
8             fields: fields{
9                 repo: mocks.NewDepartmentRepositoryMock(mc).
10                    BreadcrumbsMock.Return(ds, nil),
11             },
12             args: args{
13                 ctx: contextutils.SetEmployee(context.Background(),
14                    &models.Employee{}),
15             },
16             want:    ds,
17             wantErr: false,
18         },
19     }
20     for _, tt := range tests {
21         t.Run(tt.name, func(t *testing.T) {
22             m := &Department{
23                 repo: tt.fields.repo,
24             }
25             got, err := m.GetDepartments(tt.args.ctx, tt.args.id)
26             if (err != nil) != tt.wantErr {
27                 t.Errorf("GetDepartments() error = %v, wantErr %v",
28                    err, tt.wantErr)
29                 return
30             }
31             if !reflect.DeepEqual(got, tt.want) {
32                 t.Errorf("GetDepartments() got = %v, want %v", got,
```



```

30         tt.want)
31     })
32 }
33 }
```

3.5 Интерфейс

Интерфейс был реализован в формате одностраничного веб-приложения, в котором все отображение управляется на стороне клиента, в данном случае браузера. На рисунке 3.1 изображена главная страница интерфейса.

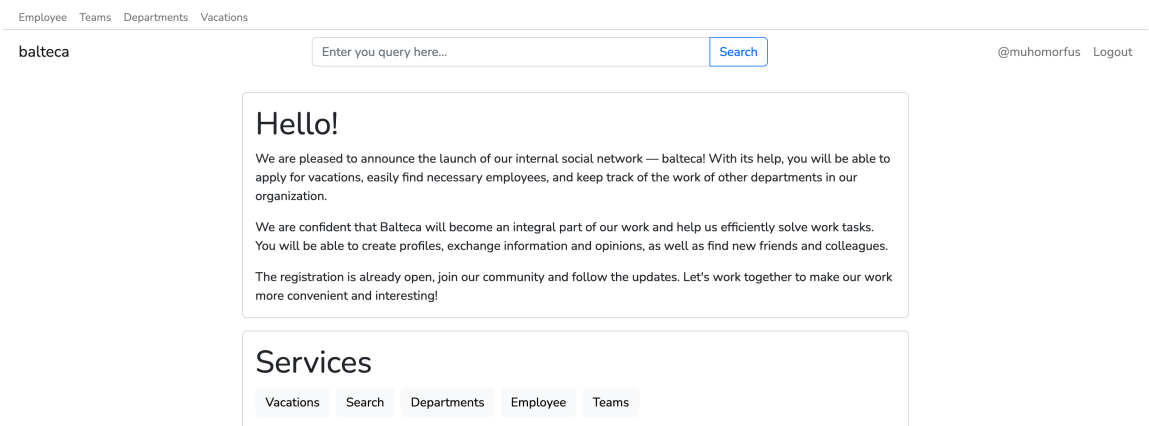


Рисунок 3.1 – Главная страница

На рис. 3.2 изображена страница отображения профиля пользователя.

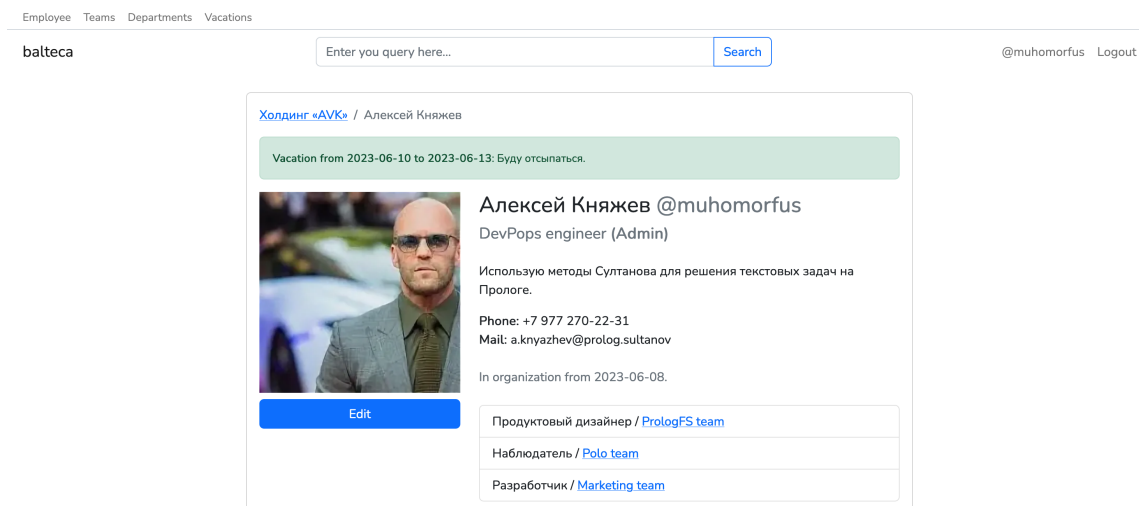


Рисунок 3.2 – Страница профиля

На рис. 3.3 изображена страница результатов поиска. Поисковая строка находится в меню навигации сверху, и доступно с любой страницы.

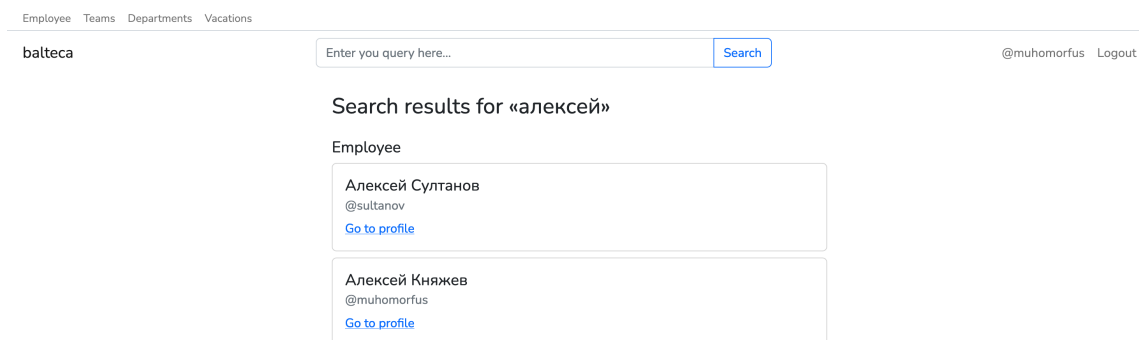


Рисунок 3.3 – Страница результатов поиска

4 Исследовательский раздел

В рамках данной работы было решено провести два исследования. Предметом исследований является поисковый запрос, а именно кеширование его результатов. Перед обращением к базе данных, сначала проверяется кеш, если ли в нем результат данного поискового запроса. Если есть, то запрос в базу данных не осуществляется, а пользователю возвращаются закешированные данные.

Данные о производительности измерялись на основе множественного запуска запросов поиска на десяти случайно выбранных заранее поисковых строках одинаковой длины. Для всех вариантов кеширования и сериализации в рамках одного тестирования, набор поисковых строк одинаков.

Характеристики устройства, на котором проводилось исследование, следующие [19]:

- оперативная память 8Гб;
- процессор Apple M2;
- операционная система macOS Ventura 13.0.

4.1 Исследование скорости работы поиска с кешированием и без

В рамках данного исследования проводилось сравнение скорости работы поиска без использования кеша и с использованием кеша двух видов: Redis [20] и кеша в оперативной памяти. Результаты замеров времени работы поиска от количества сотрудников, отделов и команд, которых, в рамках данного исследования, одинаковое количество, представлены в таблице 4.1

Таблица 4.1 – Зависимость времени выполнения запроса от количества записей с кешированием и без

Количество записей	Время без кеша, мкс	Время с кешем в Redis, мкс	Время с кешем в ОЗУ, мкс
5000	48099	1855	175
10000	47488	2286	686
15000	64490	3455	693
20000	72954	4173	706
25000	196815	4132	1047
30000	200044	4290	1236

Результаты замеров также представлены в графическом формате на рисунке 4.1.

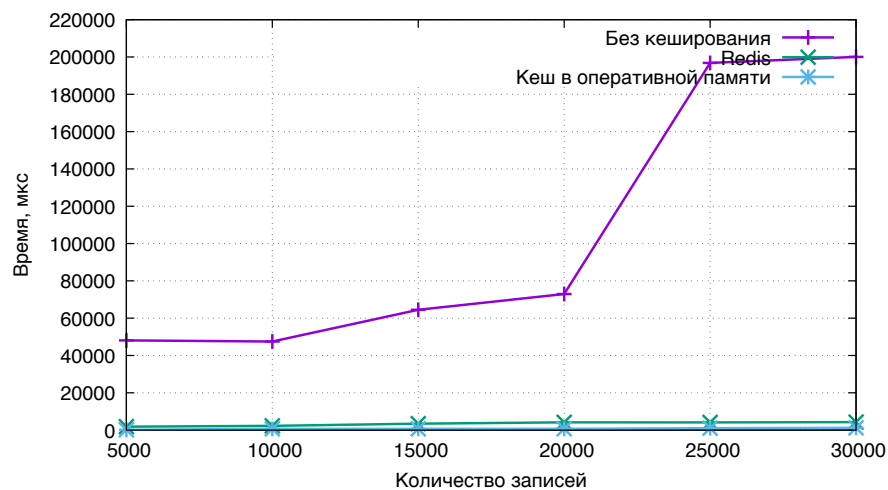


Рисунок 4.1 – Зависимость времени выполнения запроса от количества записей с кешированием и без

Исследуемая функция работает быстрее с кешированием. При этом, кеширование в оперативной памяти работает быстрее, чем кеширование в Redis на заданной выборке. Это связано в том числе с тем, что при использовании кеша в оперативной памяти, отсутствуют сетевые задержки при получении и отправке данных [20].

При объеме данных в 30 тысяч записей, запрос с использованием кеша в Redis работает примерно в 46 раз быстрее, чем без кеша. При этом, запрос с использованием кеша в оперативной памяти, работает быстрее реализации с кешем в Redis примерно в 3.4 раза.

4.2 Исследование скорости работы поиска с использованием различных методов сериализации при использовании кеширования в Redis

В рамках данного исследования проводилось сравнение скорости работы функции поиска с использованием различных алгоритмов сериализации при кешировании в Redis. В отличие от кеша в оперативной памяти, Redis хранит данные, в данном случае, в строковом формате, поэтому для хранения структуры необходимо сериализовать ее, затем преобразованную строку помещать в кеш.

В данном случае сравниваются популярные форматы сериализации JSON [21] и CBOR [22]. Кроме того, исследуется комбинация заданных форматов с алгоритмом сжатия Gzip [23]. Для реализации методов сериализации используются соответствующие библиотеки [24—26]. Результаты замеров времени для различных вариантов сериализации в зависимости от объема хранимых данных представлены в таблице 4.2 и на рисунке 4.2.

Таблица 4.2 – Зависимость времени выполнения запроса от количества записей для разных методов сериализации

Количество записей	JSON, мкс	CBOR, мкс	JSON+Gzip, мкс	CBOR+Gzip, мкс
5000	1447	1049	1208	1111
10000	2098	1527	1777	1467
15000	2487	1802	3056	1879
20000	3518	2297	3393	2402
25000	3861	2814	3634	2972
30000	4373	3595	3458	3267

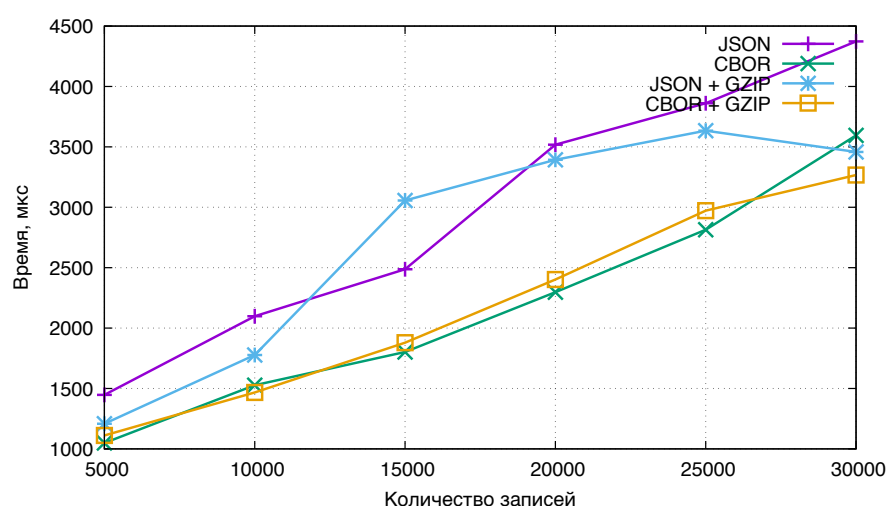


Рисунок 4.2 – Зависимость времени выполнения запроса от количества записей для разных методов сериализации

При объеме данных в 30 тысяч записей, самым долгим методом является сериализация в JSON, самым быстрым — сериализация в CBOR с использованием алгоритма сжатия Gzip. Разница во времени работы составляет примерно 1.3 раза. При этом, реализация с использованием комбинации JSON и Gzip работает также быстрее реализации JSON без архивации, несмотря на большее количество операций. Такое поведение может быть связано с тем, что заархивированные данные занимают меньше места и быстрее передаются по сети.

4.3 Исследование объема занимаемой закешированными данными памяти при использовании различных алгоритмов сериализации

В рамках данного исследование производились замеры суммарного объема занимаемой ключами в Redis памяти в зависимости от количества записей в базе данных. Результаты замеров представлены в таблице 4.3 и на рисунке 4.3.

Таблица 4.3 – Зависимость объема занимаемой памяти от количества записей для разных методов сериализации

Количество записей	JSON, Б	CBOR, Б	JSON+Gzip, Б	CBOR+Gzip, Б
5000	172880	111920	35104	30176
10000	297088	191856	56928	48544
15000	432624	276576	80576	68512
20000	562256	365536	103440	87760
25000	762032	490624	138416	117504
30000	898544	577760	162192	137536

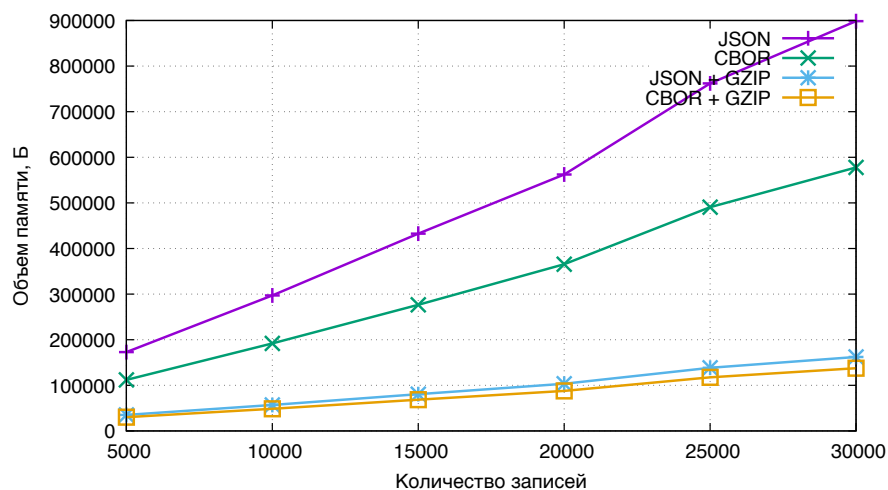


Рисунок 4.3 – Зависимость объема занимаемой памяти от количества записей для разных методов сериализации

При объеме данных в 30 тысяч записей, самым эффективным в плане потребляемой памяти является метод сериализации CBOR в комбинации с алгоритмом сжатия Gzip. Использование сжатия позволяет сократить потребление памяти примерно в 5.5 раз для JSON и примерно в 3.2 раза для

CBOR. Такое значительное сокращение объема данных позволяет ускорить получение данных из кеша из-за меньшего объема данных, передаваемого по сети, что и показано в предыдущем исследовании.

ЗАКЛЮЧЕНИЕ

В рамках выполнения работы была разработана база данных внутреннего портала для сотрудников организации. Все поставленные задачи были выполнены:

- формализована задача;
- спроектирована база данных;
- выбрана СУБД;
- реализована база данных;
- база данных заполнена данными;
- реализовано приложение для доступа к базе данных.

Цель работы — разработка базы данных внутреннего портала для сотрудников предприятия, и приложения, позволяющего взаимодействовать с ней, была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Швецов В. И.* Базы данных. — 2016.
2. *Погодин Г., Фиго Д., Васильев Э.* Сериализация структур данных для хранения и передачи в информационных системах. Методы и средства // Молодежь в науке: сборник докладов XVI научно-технической конференции. Саров: ФГУП «РФЯЦ-ВНИИЭФ. Т. 2. — 2018. — С. 231—236.
3. Что происходит в закрытой социальной сети сотрудников Mail.Ru Group [Электронный ресурс]. — Режим доступа: <https://vc.ru/insidevk/54334-intranet-mrg> (дата обращения: 02.03.2023).
4. Тортилья на вынос, виртуальный тур по офису и отпуск за пять секунд: что умеет портал для сотрудников Авито? [Электронный ресурс]. — Режим доступа: <https://vc.ru/avito/169490-tortilya-na-vynos-virtualnyy-tur-po-ofisu-i-otpusk-za-pyat-sekund-chto-umeet-portal-dlya-sotrudnikov-avito> (дата обращения: 27.02.2023).
5. *Глухих И.* Корпоративная информационная система университета на базе интернет/интранет-портала // Университетское управление: практика и анализ. — 2005. — № 5. — С. 68—76.
6. Best Intranet Russia-2019: победители и тренды [Электронный ресурс]. — Режим доступа: <https://abazhur.rivelty.ru/best-intranet-russia-2019> (дата обращения: 27.02.2023).
7. *Klee M. M.* The importance of having a non-disclosure agreement // IEEE engineering in medicine and biology magazine. — 2000. — Т. 19, № 3. — С. 120.
8. Битрикс24 [Электронный ресурс]. — Режим доступа: <https://www.bitrix24.ru> (дата обращения: 02.03.2023).
9. *Хомоненко А., Цыганков В., Мальцев М.* Базы данных // СПб.: КОРОНА принт. — 2000. — Т. 125.
10. *Бородин Д., Строганов Ю.* К задаче составления запросов к базам данных на естественном языке // Новые информационные технологии в автоматизированных системах. — 2016. — № 19. — С. 119—126.

11. *Дадян Э. Г., Зеленков Ю. А.* Методы, модели, средства хранения и обработки данных. — 2016.
12. *Корягин Д.* МОДЕЛИ БАЗ ДАННЫХ // Аллея науки. — 2020. — Т. 1, № 5. — С. 985—988.
13. *Кириллов В. В.* Введение в реляционные базы данных. — БХВ-Петербург, 2012.
14. *Жалолов О. И., Хаятов Х. У.* Понятие SQL и реляционной базы данных // Universum: технические науки. — 2020. — 6—1 (75). — С. 26—29.
15. Язык программирования Go [Электронный ресурс]. — Режим доступа: <https://go.dev> (дата обращения: 15.03.2023).
16. Vue.js — Прогрессивный JavaScript-фреймворк [Электронный ресурс]. — Режим доступа: <https://ru.vuejs.org> (дата обращения: 10.04.2023).
17. PostgreSQL — официальный сайт [Электронный ресурс]. — Режим доступа: <https://www.postgresql.org> (дата обращения: 01.04.2023).
18. Спецификация OpenAPI — официальный сайт [Электронный ресурс]. — Режим доступа: <https://swagger.io/specification> (дата обращения: 03.04.2023).
19. MacBook Air with M2 chip — технические характеристики [Электронный ресурс]. — Режим доступа: <https://www.apple.com/macbook-air-13-and-15-m2> (дата обращения: 27.04.2023).
20. Redis — официальный сайт [Электронный ресурс]. — Режим доступа: <https://redis.io> (дата обращения: 01.04.2023).
21. JSON — официальный сайт [Электронный ресурс]. — Режим доступа: <https://json.io> (дата обращения: 27.04.2023).
22. CBOR — официальный сайт [Электронный ресурс]. — Режим доступа: <https://cbor.io> (дата обращения: 27.04.2023).
23. GNU Gzip — официальный сайт [Электронный ресурс]. — Режим доступа: <https://www.gnu.org/software/gzip/manual/gzip.html> (дата обращения: 27.04.2023).

24. Документация по пакету json [Электронный ресурс]. — Режим доступа: <https://pkg.go.dev/encoding/json> (дата обращения: 27.04.2023).
25. Документация по пакету cbor [Электронный ресурс]. — Режим доступа: <https://pkg.go.dev/github.com/fxamacker/cbor/v2> (дата обращения: 27.04.2023).
26. Документация по пакету gzip [Электронный ресурс]. — Режим доступа: <https://pkg.go.dev/compress/gzip> (дата обращения: 27.04.2023).