

**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: Райдшеринг**

Студент гр. 9381

Судаков Е.В.

Студент гр. 9304

Сорин А.В.

Преподаватель

Заславский М.М.

Санкт-Петербург

2022

## ЗАДАНИЕ

Студенты:

Судаков Е.В.

Сорин А.В.

Группа 9381 и 9304

Тема работы: Ridesharing

Исходные данные:

Сервис для поиска попутчиков. Необходимые фичи: поиск попутчиков, поиск поездок, расчет длительности и стоимости, запись, профили, рейтинги, агрегация и статистика поездок как индивидуально, так и в масштабе сервиса.

<https://www.dictionary.com/browse/ridesharing#:~:text=noun,a%20usually%20privately%20owned%20vehicle>

Содержание пояснительной записки:

“Введение”, “Качественные требования к решению”, “Сценарии использования”, “Модель данных”, “Разработанное приложение”, “Заключение”, “Список литературы”

Предполагаемый объем пояснительной записки:

Не менее 19 страниц.

Дата выдачи задания: 01.09.2022

Дата сдачи реферата: 26.12.2022

Дата защиты реферата: 26.12.2022

Студент		Судаков Е.В.
Студент		Сорин А.В.
Преподаватель		Заславский М.М.

## **АННОТАЦИЯ**

Был разработан прототип приложения по поиску поездок и попутчиков на основе ridesharing. Прототип позволяет создавать аккаунты, поездки, отслеживать историю статусов поездки, приглашать в поездки других пользователей, удалять поездки и отмечать их как исполненные. Доступна история прошлых поездок, а также возможность ответить на приглашение в поездку. При разработке использовался Neo4j, GraphQL, Express, React.js. Приложение может быть запущено с помощью docker.

## **SUMMARY**

A prototype of an application for finding rides and fellow travelers based on ridesharing was developed. The prototype allows you to create accounts, trips, track the history of trip statuses, invite other users to trips, delete trips and mark them as completed. The history of past trips is available, as well as the ability to respond to an invitation to a trip. During development, Neo4j, GraphQL, Express were used, React.js. The application can be launched using docker.

## СОДЕРЖАНИЕ

	Введение	6
1.	Качественные требования к решению	6
2.	Сценарии использования	7
3.	Модель данных	12
4.	Разработанное приложение	19
	Заключение	
	Список литературы	

## **ВВЕДЕНИЕ**

Цель:

- Разработать прототип приложения поиска попутчиков и поездок.

Задачи:

- Описать основные сценарии использования и макет.
- Разработать модель данных.
- Разработать прототип хранения и представление
- Разработать прототип анализ

### **1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ**

- 1) Реализовать поиск по названию.
- 2) Реализовать фильтры и сортировки результатов поиска.
- 3) Добавить возможность просмотра профиля, поездки.
- 4) Добавить импорт/экспорт данных в формате json.
- 5) Добавить возможность запуска приложения в docker.

## 2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

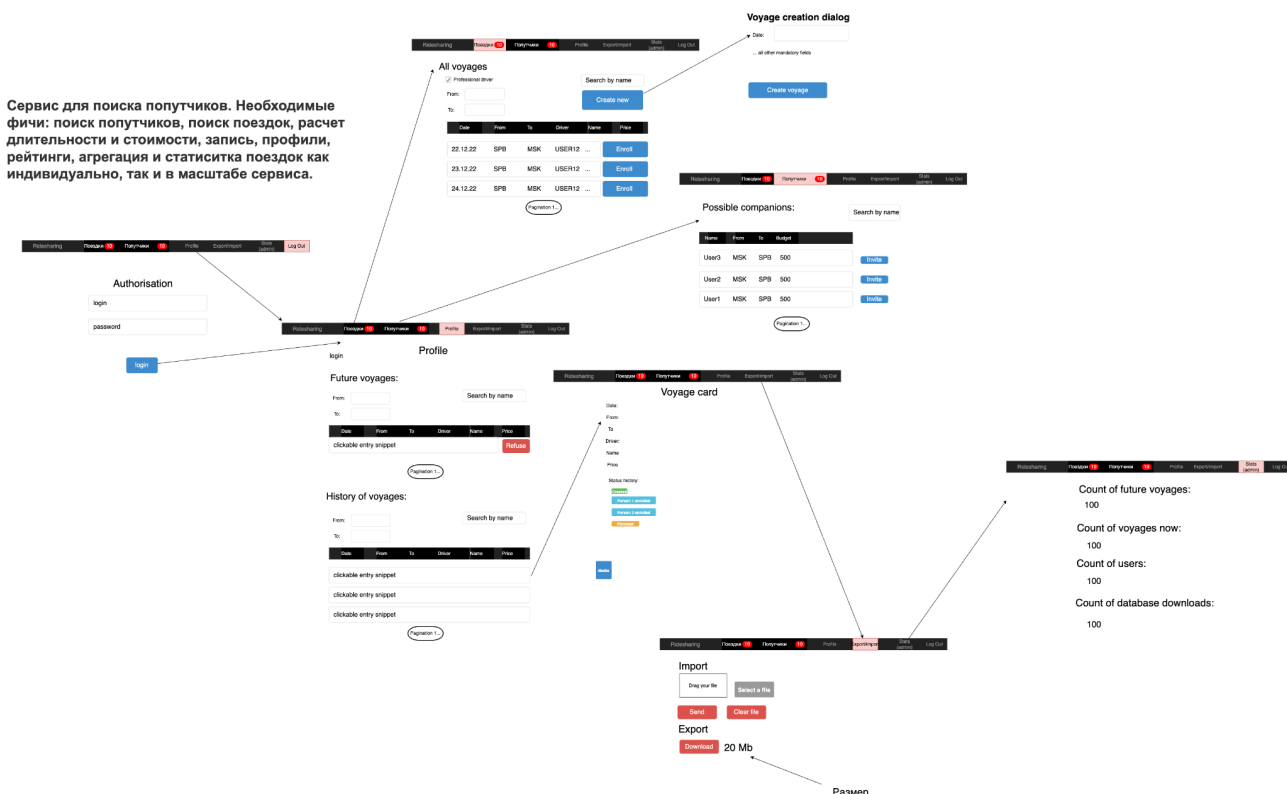


РИС 1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1. Регистрация / Авторизация
2. Пользователь может зайти в свой профиль, где отображаются предстоящие и прошлые поездки. От будущей поездки есть возможность отказаться.
3. Пользователь заходит на страницу поездок, где ему предлагается выбрать из ближайших существующих поездок. Есть возможность добавиться в поездку
4. Пользователь может создать свою собственную поездку как водитель.
5. На странице попутчиков пользователь может пригласить попутчика в свою поездку.
6. На странице импорта / экспорта можно выгрузить всю базу данных в текстовом виде, а также наоборот, загрузить.
7. На странице статистики отображается вся основная информацию о текущем состоянии БД





### 3. Модель данных

#### Нереляционная модель данных (Neo4j)

##### Графическое представление

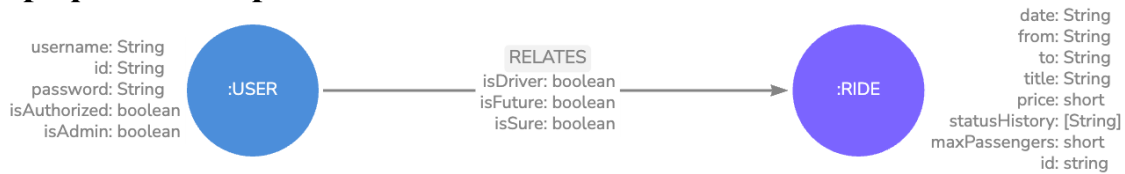


Рис 2. Графическое представление нереляционной модели данных

##### Описание назначений коллекций, типов данных и сущностей

- Узел USER - представляет сущность пользователя
- Узел RIDE - представляет сущность поездки

**Оценка удельного объема информации, хранимой в модели (сколько потребуется памяти, чтобы сохранить объекты, как объем зависит от количества объектов)**

##### Узел USER

У узла есть следующие атрибуты: \* username: String -  $V_u n = 2b * 30 = 60b$  \* id: String -  $V_i d = 2b * 30 = 60b$  \* password: String -  $V_p = 2b * 30 = 60b$  \* isAuthorized: boolean -  $V_i sAu = 1b$  \* isAdmin: boolean -  $V_i sAd = 1b$

Суммарный объем памяти, занимаемый одним узлом USER:

$$V_u = 60b + 60b + 60b + 1b + 1b = 182b$$

##### Узел RIDE

У узла есть следующие атрибуты: \* date: String -  $V_d = 2b * 30 = 60b$  \* from: String -  $V_f = 2b * 60 = 120b$  \* to: String -  $V_t = 2b * 60 = 120b$  \* title: String -  $V_t i = 2b * 30 = 60b$  \* price: short -  $V_s = 2b$  \* statusHistory: [String] -  $V_s h = 2b * 60 * 10 = 1200b$  (если предположить что есть 10 записей длиной в 60 символов) \* maxPassengers: short -  $V_m p = 2b$  \* id: String -  $V_i d = 2b * 30 = 60b$

Суммарный объем памяти, занимаемый одним узлом RIDE:

$$V_r = 60b + 120b + 120b + 60b + 2b + 1200b + 2b + 60b = 1624b$$

**Есть следующие связи:**

- RELATES с узлом RIDE -  $V_r el = 1b + 1b + 1b = 3b$

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

Пусть: -  $W$  - число пользователей; -  $Z$  - число связей; -  $X$  - число поездок;

**“Чистый” объем данных:**

$$V_{\text{чист}} = W * 182b + X * 1624b$$

При  $W = 300$ ,  $Z = 1200$ ,  $X = 200$ , количество данных, занимаемой “чистыми” данными =  $54600 + 324800 = 379400$  **Фактический объем данных для модели Neo4j:**

$$V_{\text{факт}} = W * 182b + X * 1624 + Z * 3b$$

При  $W = 300$ ,  $Z = 1200$ ,  $X = 200$ , количество данных, занимаемой “фактическими” данными =  $54600 + 324800 + 3600 = 382800$

$$\frac{V_{\text{факт}}}{V_{\text{чист}}} = 1.01$$

**Запросы к модели, с помощью которых реализуются сценарии использования**

1. Добавление пользователя

```
CREATE (a:USER {id: $id, username: $username, password: $password,
isAuthorized: $isAuthorized,
isAdmin: $isAdmin }) RETURN a
```

2. Редактирование информации о пользователе

```
MATCH(
  user: USER {
    id: "234682637"
  }
)
SET user.username = "kopito",
  user.password = "123456"
```

3. Получение всех пользователей

```
MATCH(user: USER) RETURN user
```

4. Получение количества всех пользователей

```
MATCH (u: USER) RETURN count(u)
```

5. Удаление пользователя

```
MATCH (
  user: USER {id: used_id}
) DELETE user
```

#### 6. Удаление пользователя и его связей

```
MATCH (u: USER {id: user_id})  
-[edge:RELATES]  
-(r: RIDE)  
DELETE edge  
DELETE u
```

#### 7. Авторизация

```
MATCH(u : USER {username: $username, password: $password }) SET  
u.isAuthorized = $authorized RETURN u
```

#### 8. Выход из системы

```
MATCH(u : USER {username: $username}) SET u.isAuthorized = $authorized  
RETURN u
```

#### 9. Создание поездки пользователя

```
MATCH (u: USER {username: $username}) CREATE (u)-[r: RELATES {isDriver:  
$isDriver,  
isFuture: $isFuture, isSure: $isSure}]->  
(:RIDE { id: $id, date: $date, from: $from, to: $to, title: $title,  
price: $price, statusHistory: $statusHistory, maxPassengers: $maxPassengers  
});
```

#### 10. Получение всех поездок пользователя (с пагинацией)

```
MATCH (u: USER {username: $username}) -[edge:RELATES] - (r: RIDE)  
RETURN r, edge  
ORDER BY r.id SKIP (page_no - 1) * MAX_PAGE_SIZE LIMIT  
MAX_PAGE_SIZE
```

#### 11. Получение всех поездок приложения (с пагинацией)

```
MATCH (r:RIDE) RETURN r ORDER by r.id SKIP (page_no - 1) *  
MAX_PAGE_SIZE LIMIT MAX_PAGE_SIZE
```

#### 12. Получение всех пользователей с поездками с заданным направлением (с пагинацией)

```
MATCH (u: USER) -[edge:RELATES] -(r: RIDE {from: from, to: to})  
RETURN u  
ORDER BY u.id  
SKIP (page_no - 1) * MAX_PAGE_SIZE  
LIMIT MAX_PAGE_SIZE
```

#### 13. Добавление пользователя в поездку

```
MATCH (u: USER {id: user_id})  
MATCH (r: RIDE {id: ride_id})  
CREATE (u)-[:RELATES {isDriver: false, isFuture: true, isSure: true}]->(r)  
RETURN u
```

14. Получение прошлых поездок пользователя

```
MATCH (u: USER {id: user_id})  
-[edge:RELATES {isFuture: false}]  
-(r: RIDE)  
RETURN r  
ORDER BY r.id  
SKIP (page_no - 1) * MAX_PAGE_SIZE  
LIMIT MAX_PAGE_SIZE
```

15. Получение количества прошлых поездок

```
MATCH (u: USER {id: user_id})  
-[edge:RELATES {isFuture: false}]  
-(r: RIDE)  
RETURN count(r)
```

16. Удаление пользователя из поездки

```
MATCH (u: USER {id: user_id})  
-[edge:RELATES]  
-(r: RIDE {id: ride_id})  
DELETE edge
```

17. Удаление поездки без связей

```
MATCH (r: RIDE {id: ride_id})  
DELETE r
```

18. Удаление поездки и ее связей

```
MATCH (u: USER)  
-[edge:RELATES]  
-(r: RIDE {id: ride_id})  
DELETE edge  
DELETE r
```

19. Удаление пользователя из всех поездок

```
MATCH (u: USER {id: user_id})  
-[edge:RELATES]  
-(r: RIDE)  
DELETE edge
```

20. Удаление всех пользователей из поездки

```
MATCH (u: USER)  
-[edge:RELATES]  
-(r: RIDE {id: ride_id})  
DELETE edge
```

## Реляционная модель данных (SQL)

### Графическое представление

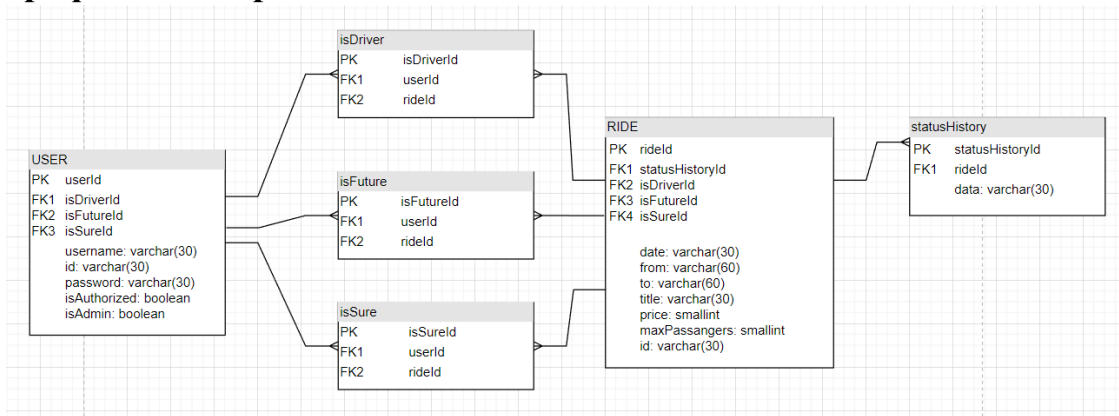


Рис 3. Графическое представление реляционной модели данных

### Описание назначений коллекций, типов данных и сущностей

- Таблица USER - таблица пользователей
- Таблица RIDE - таблица поездок
- Таблица isDriver - вспомогательная таблица для обеспечения связи многие ко многим по одному из свойств
- Таблица isFuture - вспомогательная таблица для обеспечения связи многие ко многим по одному из свойств
- Таблица isSure - вспомогательная таблица для обеспечения связи многие ко многим по одному из свойств
- Таблица statusHisroty - вспомогательная таблица для хранения массива строк

**Оценка удельного объема информации, хранимой в модели (сколько потребуется памяти, чтобы сохранить объекты, как объем зависит от количества объектов)**

### Таблица USER

Данная таблица имеет следующие поля:

- userId - UNSIGNED INT -  $V_{userId} = 4b$
- isDriverId - UNSIGNED INT -  $V_{isDIId} = 4b$
- isFutureId - UNSIGNED INT -  $V_{isFIId} = 4b$
- isSureId - UNSIGNED INT -  $V_{isSIId} = 4b$
- username - VARCHAR(30) -  $V_u = 30b$
- id - VARCHAR(30) -  $\{V_{id}\} = 30b$
- password - VARCHAR(30) -  $\{V_{p}\} = 30b$

- isAuthorized - BOOLEAN -  $\{V_{isAu} = 1b\}$
- isAdmin - BOOLEAN -  $\{V_{isAd} = 1b\}$

Суммарный объем памяти, занимаемый одной строкой в таблице USER:

$$\sum_x^y = 4b + 4b + 4b + 4b + 30b + 30b + 30b + 1b + 1b = 108b$$

### Таблица RIDE

Данная таблица имеет следующие поля:

- rideId - UNSIGNED INT -  $V_{rideId} = 4b$
- isDriverId - UNSIGNED INT -  $V_{isDId} = 4b$
- isFutureId - UNSIGNED INT -  $V_{isFId} = 4b$
- isSureId - UNSIGNED INT -  $V_{isSId} = 4b$
- statusHistoryId - UNSIGNED INT -  $V_{sHId} = 4b$
- date - VARCHAR(30) -  $V_d = 30b$
- from - VARCHAR(60) -  $V_{from} = 60b$
- to - VARCHAR(60) -  $V_{to} = 60b$
- title - VARCHAR(30) -  $V_t = 30b$
- price - SMALLINT -  $V_p = 2b$
- maxPassangers - SMALLINT -  $V_{mP} = 2b$
- id - VARCHAR(30) -  $\{V_{id} = 30b\}$

Суммарный объем памяти, занимаемый одной строкой в таблице RIDE:

$$\sum_x^y = 4b + 4b + 4b + 4b + 4b + 30b + 60b + 60b + 30b + 2b + 2b + 30b = 234b$$

### Таблица isDriver

Данная таблица имеет следующие поля:

- isDriverId - UNSIGNED INT -  $V_{isDId} = 4b$
- rideId - UNSIGNED INT -  $V_{rideId} = 4b$
- userId - UNSIGNED INT -  $V_{userId} = 4b$

Суммарный объем памяти, занимаемый одной строкой в таблице isDriver:

$$\sum_x^y = 4b + 4b + 4b = 12b$$

### Таблица isFuture

Данная таблица имеет следующие поля:

- isFutureId - UNSIGNED INT -  $V_{isFId} = 4b$
- rideId - UNSIGNED INT -  $V_{rideId} = 4b$
- userId - UNSIGNED INT -  $V_{userId} = 4b$

Суммарный объем памяти, занимаемый одной строкой в таблице isFuture:

$$\sum_x^y = 4b + 4b + 4b = 12b$$

### Таблица isSure

Данная таблица имеет следующие поля:

- isSureId - UNSIGNED INT -  $V_{isSIId} = 4b$
- rideId - UNSIGNED INT -  $V_{rideId} = 4b$
- userId - UNSIGNED INT -  $V_{userId} = 4b$

Суммарный объем памяти, занимаемый одной строкой в таблице isSure:

$$\sum_x^y = 4b + 4b + 4b = 12b$$

### Таблица statusHistory

Данная таблица имеет следующие поля:

- statusHistoryId - UNSIGNED INT -  $V_{sHIId} = 4b$
- rideId - UNSIGNED INT -  $V_{rideId} = 4b$
- data - VARCHAR(30) -  $V_{data} = 30b$  (предположим что записей 10 тогда выходит 300b)

Суммарный объем памяти, занимаемый одной строкой в таблице statusHistory:

$$\sum_x^y = 4b + 4b + 30b * 10 = 308b$$

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

Пусть: -  $W$  - число пользователей; -  $Z1$  - число связей с таблицей isDriver; -  $Z2$  - число связей с таблицей isFuture; -  $Z3$  - число связей с таблицей isSure; -  $X$  - число поездок; -  $V_i^U$  - объем данных, занимаемый полем  $i$  таблицы USER; -  $V_i^R$  - объем данных, занимаемый полем  $i$  таблицы RIDE; -  $V^i sD_i$  - объем данных, занимаемый полем  $i$  таблицы isDriver; -  $V^i sF_i$  - объем данных, занимаемый полем  $i$  таблицы isFuture; -  $V^i sS_i$  - объем данных, занимаемый полем  $i$  таблицы isSure; -  $V^s H_i$  - объем данных, занимаемый полем  $i$  таблицы statusHistory;

**“Чистый” объем данных.**

Чистый объем данных  $\$V_{\text{чист}} = 92b * W + 214b * X + 300b * X$

При  $W = 300$ ,  $X = 200$ , количество данных, занимаемой “чистыми” данными =  $300 * 92 + 514 * 200 = 130400$

**Фактический объем данных для модели SQL:**

$$V_{\text{факт}} = (92b + 16b) * W + (214b + 20b) * X + (300b + 8b) * X + Z1 * 12b + \dots$$

При  $W = 300$ ,  $X = 200$ ,  $Z1 = Z2 = Z3 = 400$ , количество данных, занимаемой SQL моделью =  $108 * 300 + 200 * 542 + 400 * 36 = 155200$

$$\frac{V_{\text{факт}}}{V_{\text{чист}}} = 1.19$$

**Запросы к модели, с помощью которых реализуются сценарии использования**

1. Создание таблицы USER

```
CREATE TABLE USER (  
    userId INTEGER PRIMARY KEY AUTOINCREMENT,  
    FOREIGN KEY (isDriverId) REFERENCES isDriver (isDriverId),  
    FOREIGN KEY (isFutureId) REFERENCES isFuture (isFutureId),  
    FOREIGN KEY (isSureId) REFERENCES isSure (isSureId),  
    username VARCHAR(30) NOT NULL UNIQUE,  
    id VARCHAR(30) NOT NULL UNIQUE,  
    password VARCHAR(30) NOT NULL,  
    isAuthorized BOOLEAN,  
    isAdmin BOOLEAN  
);
```

2. Создание таблицы RIDE



```
CREATE TABLE RIDE (
    rideId INTEGER PRIMARY KEY AUTOINCREMENT,
    FOREIGN KEY (isDriverId) REFERENCES isDriver (isDriverId),
    FOREIGN KEY (isFutureId) REFERENCES isFuture (isFutureId),
    FOREIGN KEY (isSureId) REFERENCES isSure (isSureId),
    FOREIGN KEY (statusHistoryId) REFERENCES isSure (isSureId),
    date VARCHAR(30) NOT NULL,
    from VARCHAR(60) NOT NULL,
    to VARCHAR(60) NOT NULL,
    title VARCHAR(30) NOT NULL,
    price SMALLINT,
    maxPassangers SMALLINT,
    id VARCHAR(30) NOT NULL
);
```

### 3. Создание таблицы isDriver

```
CREATE TABLE isDriver (
    isDriverId INTEGER PRIMARY KEY AUTOINCREMENT,
    FOREIGN KEY (userId) REFERENCES USER (userId),
    FOREIGN KEY (rideId) REFERENCES RIDE (rideId)
);
```

### 4. Создание таблицы isFuture

```
CREATE TABLE isFuture (
    isFutureId INTEGER PRIMARY KEY AUTOINCREMENT,
    FOREIGN KEY (userId) REFERENCES USER (userId),
    FOREIGN KEY (rideId) REFERENCES RIDE (rideId)
);
```

### 5. Создание таблицы isSure

```
CREATE TABLE isSure (
    isSureId INTEGER PRIMARY KEY AUTOINCREMENT,
    FOREIGN KEY (userId) REFERENCES USER (userId),
    FOREIGN KEY (rideId) REFERENCES RIDE (rideId)
);
```

### 6. Создание таблицы statusHistory

```
CREATE TABLE statusHistory (
    statusHistoryId INTEGER PRIMARY KEY AUTOINCREMENT,
    FOREIGN KEY (rideId) REFERENCES RIDE (rideId),
    data VARCHAR(30)
);
```

### 7. Добавление пользователя

```
INSERT INTO USER (
    username,
```

```

    id,
    password,
    isAuthorized,
    isAdmin,
) VALUES (
    'ivanovich',
    '4376473646',
    'qwerty123',
    FALSE,
    FALSE
);

```

7. Редактирование информации пользователя

```

UPDATE USER
SET username = 'petrov',
    isAdmin = TRUE
WHERE id = '4376473646'

```

8. Получение пользователя по id

```

SELECT * FROM WORKER where id = '4376473646'

```

9. Получение всех пользователей

```

SELECT * FROM USER

```

10. Получение всех админов

```

SELECT * FROM USER
WHERE USER.isAdmin == TRUE

```

11. Удаление пользователя по id

```

DELETE FROM USER where id = '4376473646'

```

12. Авторизация

```

UPDATE USER
SET USER.isAuthorized = TRUE
WHERE USER.username == "john" AND USER.password == "qwerty"

```

13. Выход из системы

```

UPDATE USER
SET USER.isAuthorized = FALSE
WHERE USER.username == "john"

```

14. Добавление поездки

```

INSERT INTO RIDE (
    date,
    from,
    to,

```

```

        title,
        price,
        maxPassengers,
        id
    ) VALUES (
        '24-12-2022',
        'Moscow',
        'Saint-Petersburg',
        'Ride from M to S',
        500,
        4,
        '231423423434'
    );

```

15. Получение всех будущих поездок

```

SELECT RIDE.id FROM USER
INNER JOIN isFuture ON USER.isFutureId == isFuture.isFutureId
INNER JOIN RIDE ON isFuture.rideId == RIDE.rideId

```

16. Получение всех поездок где пользователь водитель 23-04-2023 числа

```

SELECT RIDE.id FROM USER
INNER JOIN isDriver ON USER.isDriverId == isDriver.isDriverId
INNER JOIN RIDE ON isDriver.rideId == RIDE.rideId
WHERE RIDE.date == '23-04-2023'

```

17. Получение количества всех будущих поездок

```

SELECT COUNT(RIDE.id) FROM USER
INNER JOIN isFuture ON USER.isFutureId == isFuture.isFutureId
INNER JOIN RIDE ON isFuture.rideId == RIDE.rideId

```

18. Добавление пользователя в водители поездки

```

INSERT INTO isDriver (
    isDriverId,
    userId,
    rideId
) VALUES (
    '35465',
    '47355',
    '6776776'
)

```

19. Получение всех поездок (с пагинацией)

```

SELECT * FROM RIDE ORDER by r.id SKIP (page_no - 1) * MAX_PAGE_SIZE
LIMIT MAX_PAGE_SIZE

```

20. Удаление поездок с определенной датой

DELETE FROM WORKER WHERE date = '29-12-2022'

21. Удаление пользователя из водителей поездки

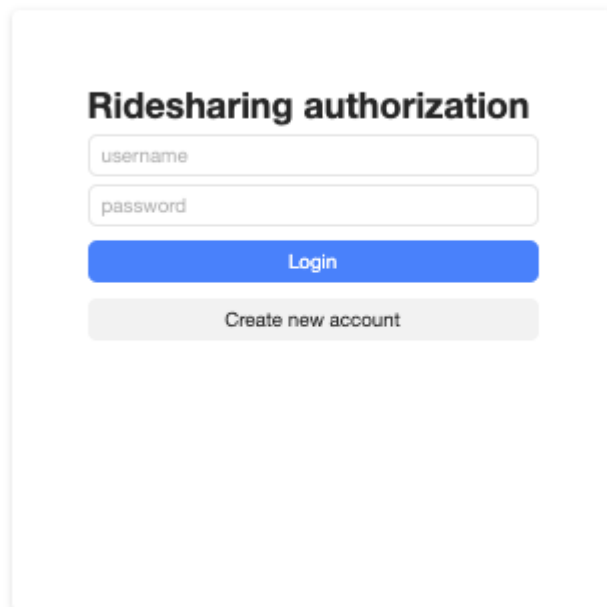
DELETE FROM isDriver WHERE userId == '34243424234'

### **Сравнение SQL и NoSQL**

- Отношение фактических данных к “чистым” дали следующие результаты: для SQL - 1.19, для NoSQL - 1.01. Таким образом, получили, что NoSQL модель более экономична по памяти, нежели SQL.
- Neo4j имеет преимущество в виде графовой структуры. Данная структура позволяет избежать создание дополнительных таблиц для связей путем создания связей между узлами данных.
- Количество запросов, необходимых для выполнения сценариев использования в SQL модели больше.

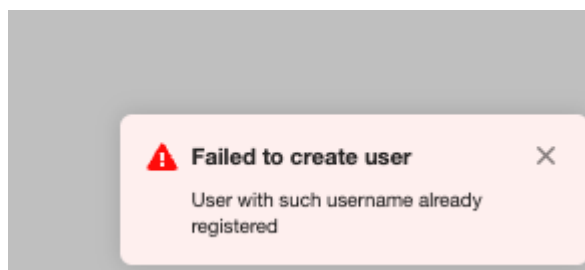
Вывод: для данной задачи определенно лучше использовать NoSQL модель. Она занимает меньше места, и с ней проще работать.

## 4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ



The image shows a login form titled "Ridesharing authorization". It contains two input fields: "username" and "password". Below these fields are two buttons: a blue "Login" button and a grey "Create new account" button.

*Рис 4. Форма логина*



*Рис 5. Нотификация при попытке создать уже существующего пользователя*

# Statistics

## All users

Username	Id
admin	1164ff6a-368a-4da0-b08c-93ee4530ca37
test5	16309071-ca7e-4e82-a2a9-7014b51d049e
test3	1bbd4f47-e6a1-4679-a852-1af8e2c13562
test	320a7d2f-f94a-420a-957f-05a0f6e40874
test2	362f682d-5277-4433-808b-ea67c937270f

< 1 2 >

## All rides

Title	From	To	Price	MaxPassengers	
test ride	spb	msk	150	10	Status history
test ride 2	spb	msk	100	150	Status history

< 1 >

Download DB

Рис 6. Страница статистики для администратора - с возможностью выгрузки данных.



Рис 7. История статусов поездки

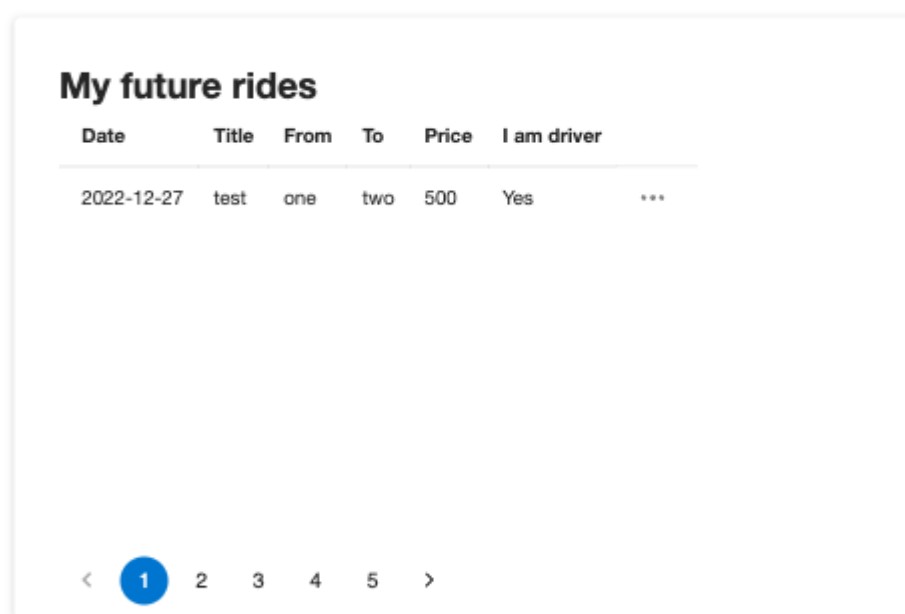


Рис 8. Список предстоящих поездок

# Companions

Username	Id	Action
admin	1164ff6a-368a-4da0-b08c-93ee4530ca37	Add user to your ride
test5	16309071-ca7e-4e82-a2a9-7014b51d049e	Add user to your ride
test3	1bbd4f47-e6a1-4679-a852-1af8e2c13562	Add user to your ride
test	320a7d2f-f94a-420a-957f-05a0f6e40874	Add user to your ride
test2	362f682d-5277-4433-808b-ea67c937270f	Add user to your ride

< 1 2 >

Рис 9. Список пользователей которых можно пригласить в поездки

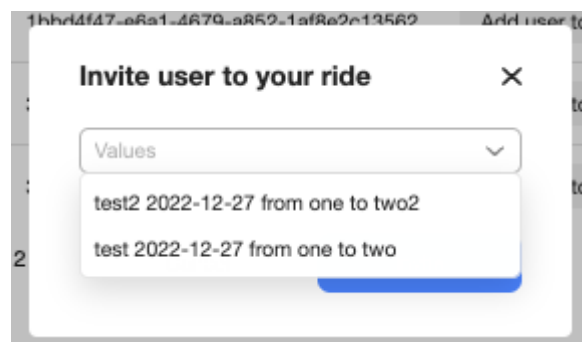


Рис 10. Диалог приглашения пользователя в поездку



Invitations						
Date	Title	From	To	Price	I am driver	
2022-12-27	test	one	two	500	No	...
						Accept invitation

< 1 2 3 4 5 >

Рис 11. Таблица приглашений с возможностью согласиться

test		
23:19:48 GMT+0300 (Moscow Standard Time)	<b>ACCEPTED_BY_320a7d2f-f94a-420a-957f-05a0f6e40874</b>	ACCEPTED_BY_320a7d2f-f94a-420a-957f-05a0f6e40874
23:19:05 GMT+0300 (Moscow Standard Time)	<b>INVITED_USER320a7d2f-f94a-420a-957f-05a0f6e40874</b>	INVITED_USER320a7d2f-f94a-420a-957f-05a0f6e40874
23:16:30 GMT+0300 (Moscow Standard Time)	<b>CREATED</b>	CREATED

Рис 12. Обновленная история статусов поездки после приглашения и принятия

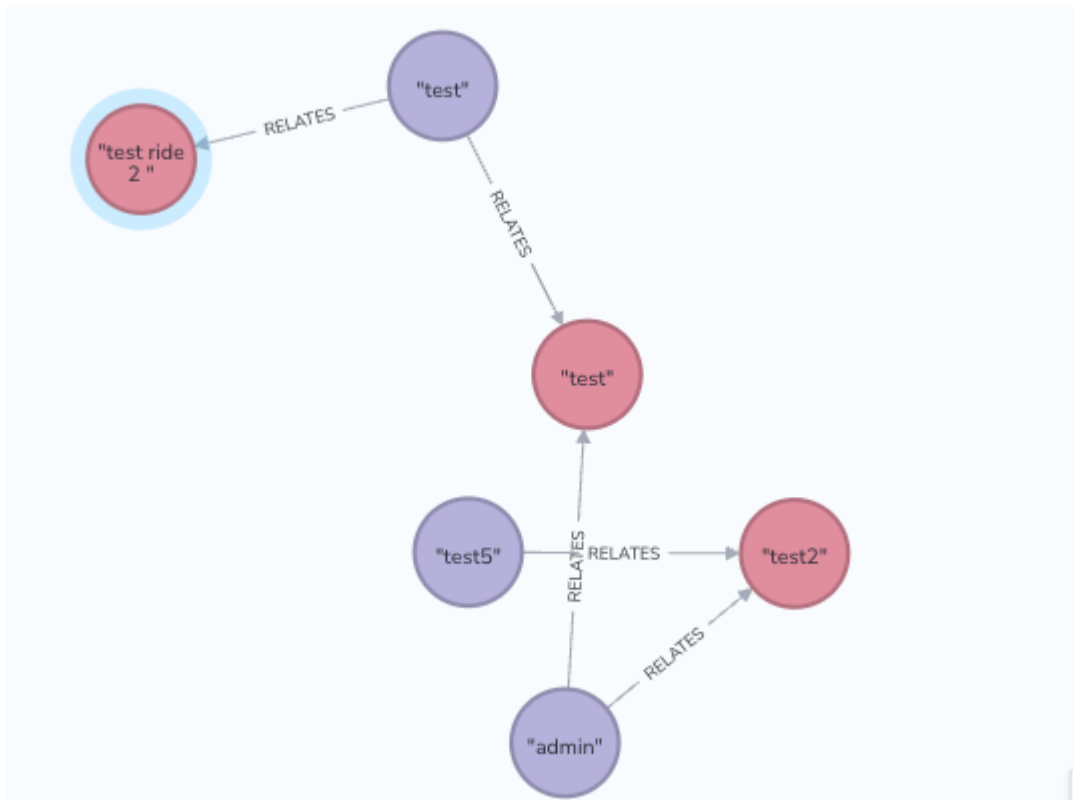


Рис 13. Узлы и ребра графа базы данных.

### Использованные технологии

БД: neo4j

Бэкенд: Express, Neo4j-driver, GraphQL, TypeScript

Фронтенд: Typescript, React.js, GraphQL

Ссылка на приложение: <https://github.com/moevm/nosql2h22-ridesharing>

## **5. ВЫВОДЫ**

Был разработан прототип приложения для поиска попутчиков и поездок, реализующий костяк необходимого функционала. Недостатком решения является отсутствие фильтров для таблиц, однако разработанный бекенд и фронтенд позволят достаточно просто добавить этот функционал. В дальнейшем решение может быть улучшено с помощью добавления интерактивной карты, визуализирующей поездки и желаемые направления пользователей.

## ПРИЛОЖЕНИЕ

Чтобы развернуть приложение необходимо ввести команду  
**docker-compose up --build**

### Список литературы.

1. <https://reactjs.org/docs/getting-started.html> - документация React.js
2. <https://neo4j.com/> - сайт базы данных