

Факультет программной инженерии и компьютерной техники

Информационные системы и базы данных

Курсовая работа: информационная система для частной военной компании

Преподаватель: Харитонова Анастасия Евгеньевна

Выполнил: Кульбако Артемий Юрьевич Р33112

# Предметная область

**База данных для частной военной компании**

Частная военная компания имеет штаб СОТРУДНИКОВ (у работников необходимо узнать ИМЯ, ФАМИЛИЮ, ДАТУ\_РОЖДЕНИЯ, ОБРАЗОВАНИЕ и текущий СЕМЕЙНЫЙ\_СТАТУС, а также хранить его ДАТУ\_ЗАЧИСЛЕНИЯ на службу), которые могут занимать различные военные и невоенные ДОЛЖНОСТИ (НАЗВАНИЕ\_ДОЛЖНОСТИ, ЗАРПЛАТУ ВОИНСКОЕ\_ЗВАНИЕ если есть, номер комплекта ЭКИПИРОВКИ, и тип ВООРУЖЕННЫХ\_СИЛ (также работники могут занимать гражданские должности)).

Типы ВС: СВ, ВМФ, ВКС.

Каждый сотрудник имеет МЕДКАРТУ (с информацией о РОСТЕ в см, ВЕСЕ в кг, ГРУППЕ\_КРОВИ (по системе AB0), ПЕРЕНЕСЁННЫХ\_ТРАВМАХ/ЗАБОЛЕВАНИЯХ, БИОЛОГИЧЕСКОМ\_ПОЛЕ) и закрепленную за ним БАЗУ, являющуюся его основным местом прибывания (с информацией о МЕСТОПОЛОЖЕНИИ базы и её СТАТУСЕ).

Сотрудники могут отправляться на МИССИИ (нужно хранить НАЗВАНИЕ, ДАТУ\_И\_ВРЕМЯ\_СТАРТА и ЗАВЕРШЕНИЯ, ЮРИДИЧЕСКИЙ\_СТАТУС, МЕСТО\_ОТПРАВЛЕНИЯ и ПРИБЫТИЯ, ВРАГОВ), а также историю миссий сотрудников.

Миссия является большой частью военной КАМПАНИИ (должно содержать НАЗВАНИЕ, ЗАКАЗЧИКА, ПРИБЫЛЬ, ЗАТРАТЫ и СТАТУС\_ВЫПОЛЕНИЯ) на ТРАНСПОРТЕ (НАЗВАНИЕ, ТИП, необходимо также знать, когда СОСТОЯНИЕ находится), принадлежащем чвк (транспорт, естественно не утилизируется после миссии и может быть использован повторно).

Для безопасности, стоит хранить историю всех ТЕХОСМОТРОВ (с номерами ТРАНСПОРТА, ОБСЛУЖИВАЮЩЕГО, ДАТЫ\_ОСМОТРА).

Людям с военной должностью должен выдаваться комплект ЭКИПИРОВКИ (где может быть (но необязательно) КАМУФЛЯЖ, СРЕДСТВА\_КОММУНИКАЦИИ, РАЗВЕДКИ, МЕДИКАМЕНТЫ и ПРОЧЕЕ).

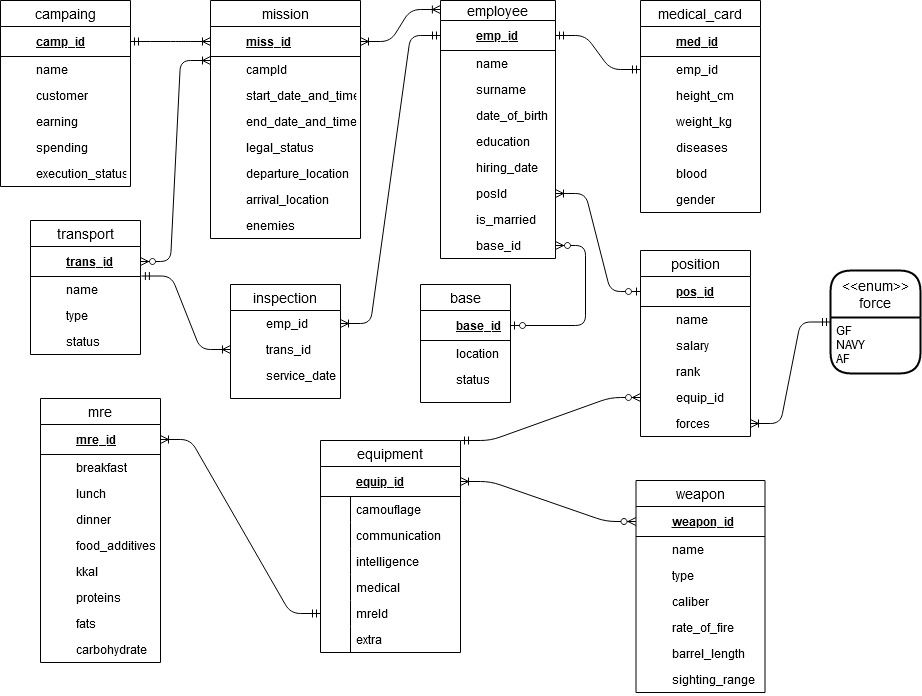
Экипировка в обязательном порядке должна включать один из ИРП (с описание о БЕЛКАХ, ЖИРАХ, УГЛЕВОДАХ и КАЛОРИЙНОСТИ, блюдах ЗАВТРАКА, ОБЕДА, УЖИНА, ПИЩЕВЫХ\_ДОБАВКАХ).

Экипировка может содержать одно или несколько ОРУЖИЙ (с подробными техническими характеристиками, будь то НАЗВАНИЕ, ТИП, КАЛИБР, СКОРОСТРЕЛЬНОСТЬ, ДЛИНА\_СТВОЛА, ПРИЦЕЛЬНАЯ\_ДАЛЬНОСТЬ).

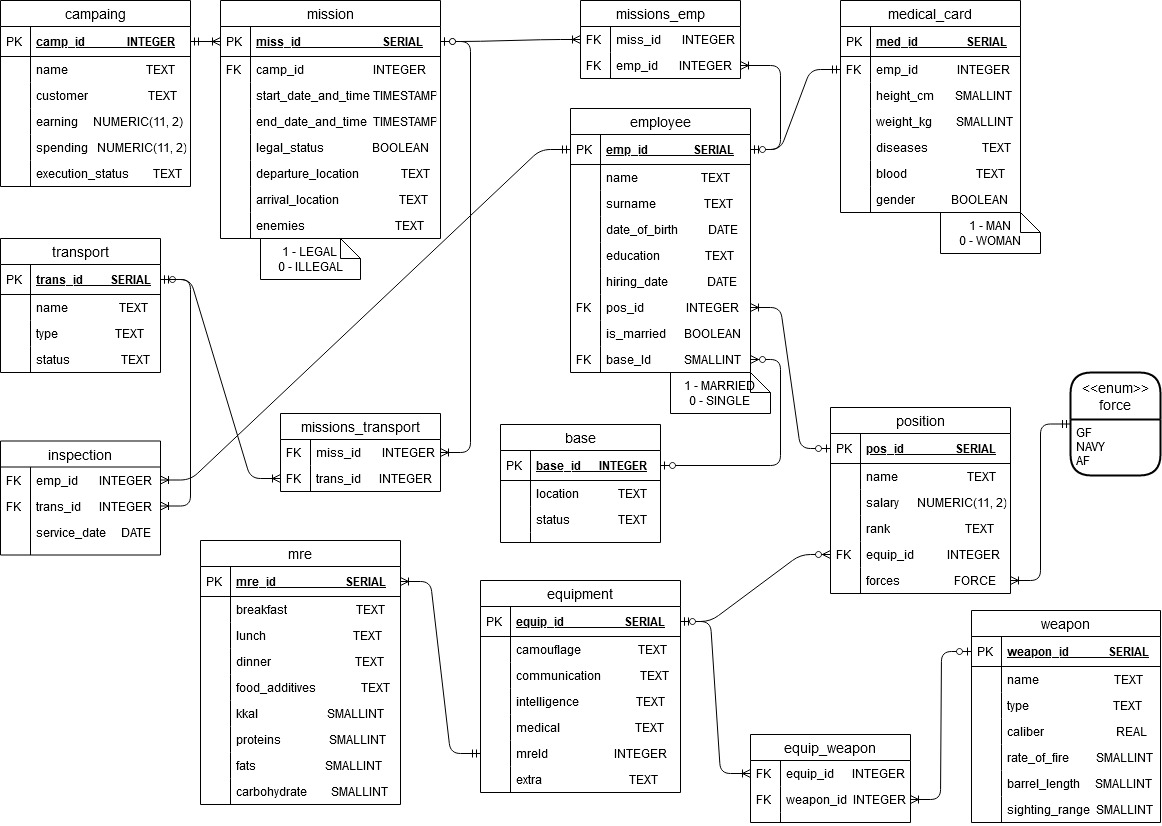
**Процессы**

1. Тех, кто не имеет воинских званий, нельзя отправлять на боевые миссии.
2. Информационная система должна учитывая какие сотрудники отправились на миссии (один и тот же сотрудник не может находиться на двух миссиях одновременно).
3. Работников неподходящих по физическим данным запрещено устраивать как военных сотрудников.
4. Необходимо хранить историю инспекций транспорта, а транспорт со статусами «сломан» или «в ремонте» нельзя использовать в операциях.
5. Если за базой не закреплён ни один сотрудник, стоит закрыть её.
6. Стараться отправлять на боевые операции при прочих равных в первую очередь неженатых военных, давно не участвовавших в миссиях, имеющих большой опыт работы.

# Инфологическая модель



# Даталогическая модель



# Индексы

Естественно, большая часть обращений будут происходит к таблицам, связанным с бизнес-процессами информационной системы, поэтому при оптимизации нужно делать ставку на них.

1-ую функцию оптимизировать смысла нет, т.к. при обращении к 2 из 3 трёх таблиц поиск и так происходит через индексы, ибо условие отбора работает с первичными ключами таблиц.

Во 2-ой функции, для избежание пересечения временных промежутков добавляемой миссии и существующих, нам необходимо получить этот самый интервал для каждой записи, и индекс здесь действительно будет к месту.

*CREATE INDEX* mission\_period *ON* mission *USING* btree(start\_date\_and\_time, end\_date\_and\_time);

|  |  |  |
| --- | --- | --- |
|  | -> |  |

**Выигрыш в треть.**

В 3-ем необходимо сопоставлять работников и их некоторые параметры из их медицинской карты. Т.к. выборка на «нижнем уровне» дерева запроса осуществляет по ключу таблицы, индекс не нужен.

4-ый: аналогично 3-ему.

В 5-ой функции стоит сделать материализованное представление, которое будет содержать базы и количество их сотрудников, а логику обновления этой таблицы запускать по запросу на изменение данных Employee.

*CREATE MATERIALIZED VIEW* base\_count\_emp *AS* (*SELECT* base\_id *FROM* base *JOIN* employee *USING* (base\_id) *GROUP BY* base\_id *HAVING* COUNT(emp\_id) = 0);  
  
*CREATE FUNCTION* update\_base\_count\_emp() *RETURNS trigger AS* $$  
 *BEGIN  
 REFRESH MATERIALIZED VIEW* base\_count\_emp;  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE TRIGGER* update\_base\_count\_emp *BEFORE INSERT OR UPDATE OR DELETE ON* employee  
 *FOR EACH ROW EXECUTE FUNCTION* update\_base\_count\_emp();

В последней функции присутствует условие отбора записей, не по ключу, и в таблице (правда записей в таблице не так уж и много). Создадим индекс:

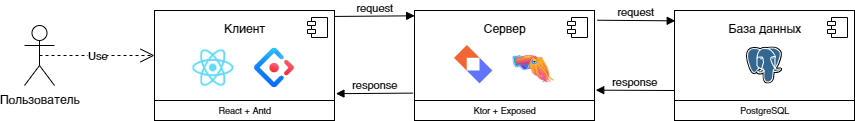
*CREATE INDEX* pos\_rank *ON position USING* btree(rank);

|  |  |  |
| --- | --- | --- |
|  | -> |  |

**Время незначительно сократилось.**

# Информационная система

## Компонентная модель



## api

Аpi до безобразия банален. Для каждой сущности заведён свой url, а http-метод определяет операцию. Все данные передаётся в json.

|  |  |
| --- | --- |
| GET (получение) | Отправляется ключ “selectedIds” для получения записей, значением является массив (если пустой, то вернуть все записи, иначе с запрошенными id).  Примеры:  *GET* http://localhost:9090/base?ids={ "selectedIds": [] }  *GET* http://localhost:9090/base?ids={ "selectedIds": [3, 5] } |
| POST (добавление) | Нужно передать всю сущность, вместо уникального ключа необходимо передавать null.  Пример:  *POST* http://localhost:9090/base *Content-Type*: application/json  {  "baseId" : null,  "location" : "LUNA",  "status" : "TEST-BASE" } |
| PUT (модификация) | Обязательно передаётся id сущности, а также вся сущности или только те поля, которые будут отличаться.  Пример:  *PUT* http://localhost:9090/base *Content-Type*: application/json  {  "baseId": 419,  "location": "MARS" } |
| DELETE (удаление) | Передаётся массив с id удаляемых записей.  Пример:  *DELETE* http://localhost:9090/base *Content-Type*: application/json  {  "droppedIds": [414, 415] } |

Также есть уникальный путь god, для целей разработки:

*### получить страницу godmode  
GET* http://localhost:9090/god  
  
*### создать таблицы  
PUT* http://localhost:9090/god  
  
*### заполнить таблицы случайными данными  
POST* http://localhost:9090/god  
  
*### уничтожить таблицы и все записи  
DELETE* http://localhost:9090/god

## Реализация

Стек технологий выбирал по двум параметрам: возможность развернуть на сервере ВТ, при этом это должно было быть что-то, с чем я не работал. Бэкенд первоначально планировался на [Micronaut](https://micronaut.io/) (отказался из-за не самой подробной документации, возможно в будущем попробую ещё раз), потом [Node.js](https://nodejs.org/ru/) (на Гелиосе запустить его не удалось). Тогда было решено вернуться к чему-то на JVM. Я знаю Kotlin, поэтому сначала решил попробовать Spring на Kotlin, но после перешёл на связку [Ktor](https://ktor.io/) и [Exposed](https://github.com/JetBrains/Exposed) – с нуля созданных на Kotlin для него фреймворка и ORM. Ktor чудесен – написать REST-API приложение можно быстро, и в отличие от Java + Spring количество бойлерплейт кода невелико. С Exposed оказалось сложнее (для взаимодействия с бд можно использовать DSL или DAO подход, и каждый из них имеет свои плюсы и минусы. В целом, Exposed мне понравился, но работе с ним всё же принесла немного боли.

Фронт решил попробовать делать на [React](https://ru.reactjs.org/) (до этого имел опыт с Vue, поэтому думал будет несложно). Несмотря на общую компонентную концепцию обоих, а также общие методы создание этих самых компонентов, JSX (язык React), показался мне чуть сложнее, ввиду не такого явного как у Vue разделения на разметку, логику и стиль. Также мне не удалось избежать вечной проблемы JavaScript – «ада коллбеков», и даже рефакторинг кода, с заменой части коллбеков на промисы не сильно спас ситуацию. Понравилось работать с библиотекой компонентов [Antd](https://ant.design/), которая сильно сократила время работы над фронтендом. Подводя итог, я всё же в будущем предпочту Vue, хотя уже после разработки вижу, что некоторых проблем можно было избежать, если бы на стадии проектирования я уже знал React.

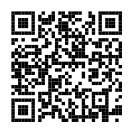
Пробела была и с REST-API. Во время разработки бэкенда, я не вспомнил, что в http-GET запросе в теле метода не рекомендуется передавать данные, но явно это не запрещается. В время тестирования, я удачно посылал GET с json-контентом в теле, а позже обнаружил, что js-овский метод fetch, предназначенный для ajax-запросов, не позволяет этого делать, поэтому пришлось отправлять json-данные как параметр, прикреплённый к url-у.

С слоем базы данных проблем не возникало – [PostgreSQL](https://www.postgresql.org/) – проверенная временем база данных, информацию о который достаточно легче найти и понять.

# Вывод

В рамках курсовой работы я полностью создал информационные систему: от этапа формирования предметной области (где преподаватель выступал в качестве некого заказчика), до этапа внедрения на оборудовании этого самого заказчика (Helios). За все два с половиной курса – это самое полезное и наиболее близкое к настоящей работе задание, где мне «повезло» столкнуться даже с типичными проблемами реальных проектов, в виде неправильного спроектированного api, к примеру, поверхностного знания технологий, а также неудачному проектированию, ввиду нехватки времени (по-моему мнению, это моя самая большая ошибка).

**Полный код (включая конфигурационные файлы и ресурсы) доступен по ссылке:**



|  |  |
| --- | --- |
|  |  |

(файлы кликабельны)