Министерство образования и науки РФ  
Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

**Базы данных**

**Отчет по курсовому проекту**

Тема: «Информационная система военного округа»

Выполнил: студент группы РИС-19-1б

Мазитов Т.Э.

Проверил: доцент кафедры ИТАС Петренко А.А

г.Пермь – 2021

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 2

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 3

1.1 Границы исследования 3

1.2 Анализ средств разработки 6

2 ТЕХНОЛОГИЯ РЕАЛИЗАЦИИ СТРУКТУРЫ БАЗЫ ДАННЫХ 8

2.1 Концептуальная модель 8

2.2 Логическая модель 9

2.3 Физическая модель 12

3 ТЕХНОЛОГИЯ РЕАЛИЗАЦИИ ВЕБ-ПРИЛОЖЕНИЯ 15

3.1 Создание базы данных 15

3.2 Создание бизнес-логики 17

3.2.1 Схема MVC 17

3.2.2 Создание моделей 18

3.2.3 Создание контроллеров 21

Создание пользовательского интерфейса 24

ЗАКЛЮЧЕНИЕ 27

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29

ПРИЛОЖЕНИЯ 30

**ВВЕДЕНИЕ**

Одним из ключевых этапов учебного процесса на втором курсе по специальности “Разработка программно-информационных систем” является курсовая работа по базам данных по теме, назначенной преподавателем дисциплины.

Цель работы – систематизация и углубление теоретических знаний, полученных за время обучения, приобретение и закрепление навыков самостоятельной работы.

Курсовая работа носит исследовательский характер. В ходе ее выполнения студент должен перейти от выполнения нормативных инструкций, которые получал ранее на практических и лабораторных занятиях, к исследовательской деятельности и принятию самостоятельных решений.

Работа предусматривает: обследование заданной предметной области, систематизацию полученных результатов и представление их в виде концептуальной модели (в ручном варианте или с помощью case-средств); преобразование концептуальной модели в логическую, затем в физическую с использованием нормализации; создание таблиц и первоначальное заполнение базы; разработку интерфейса приложения и базы данных, решение практических задач с использованием созданной базы данных; разработку проектной и пользовательской документации на разработанную систему (база данных + приложение).

**1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

**1.1 Границы исследования**

**Подробное описание темы**

Научным руководителем студента была назначена тема “Информационная система военного округа”.

Полное описание реализуемой информационной системы:

*“Военные части округа расквартированы по местам дислокации, причем в одном месте могут располагаться несколько частей. Каждая воинская часть состоит из рот, роты из взводов, взводы из отделений, в свою очередь, воинские части объединяются в дивизии, корпуса или бригады, а те – в армии.*

*Военный округ представлен офицерским составом (генералы, полковники, подполковники, майоры, капитаны, лейтенанты) и рядовым и сержантским составом (старшины, сержанты, прапорщики, ефрейторы, рядовые). Каждая из перечисленных категорий военнослужащих может иметь характеристики, присущие только этой категории: для генералов это может быть дата окончания академии, дата присвоения генеральского звания и т.д. Каждое из подразделений имеет командира, причем военнослужащие офицерского состава могут командовать любым из вышеперечисленных подразделений, а военнослужащие рядового и сержантского состава только взводом и отделением. Все военнослужащие имеют одну или несколько воинских специальностей.*

*Каждой воинской части придана боевая и транспортная техника: БМП, тягачи, автотранспорт и пр. и вооружение: карабины, автоматическое оружие, артиллерия, ракетное вооружение и т.д. Каждая из перечисленных категорий боевой техники и вооружения также имеет специфические, присущие только ей атрибуты и по каждой категории может быть несколько видов техники и вооружения. Инфраструктура военной части представлена набором сооружений (сооружение 1, сооружение 2 ... ), некоторые из которых предназначены для дислокации подразделений части.”*

Дополнительно, пользователю должна быть предоставлена возможность отправлять запросы к базе данных:

1. Получите перечень всех частей военного округа, указанной армии, дивизии, корпуса и их командиров.
2. Получите данные по офицерскому составу в целом и по офицерскому составу указанного звания всех частей военного округа, отдельной армии, дивизии, корпуса, военной части.
3. Получите данные по рядовому и сержантскому составу в целом и с учетом указанного звания всех частей военного округа, отдельной армии, дивизии, корпуса, военной части.
4. Получите цепочку подчиненности снизу доверху для указанного военнослужащего.
5. Получите перечень мест дислокации всех частей военного округа, отдельной армии, дивизии, корпуса, военной части.
6. Получите данные о наличии боевой технике в целом и с учетом указанной категории или вида во всех частях военного округа, в отдельной армии, дивизии, корпусе, военной части.
7. Получите перечень сооружений указанной военной части, перечень сооружений, где дислоцировано более одного подразделения, где не дислоцировано ни одного подразделения.
8. Получите перечень военных частей, в которых число единиц указанного вида боевой техники больше 5 (нет указанной боевой техники).
9. Получите данные о наличии вооружения в целом и с учетом указанной категории или вида во всех частях военного округа, в отдельной армии, дивизии, корпусе, военной части.
10. Получите перечень военных специальностей, по которым в округе, в отдельной армии, дивизии, корпусе, военной части более пяти специалистов (нет специалистов).
11. Получите перечень военнослужащих указанной специальности в округе, в отдельной армии, дивизии, корпусе, военной части, в указанном подразделении некоторой военной части.
12. Получите перечень военных частей, в которых число единиц указанного вида вооружения больше 10 (нет указанного вооружения).
13. Получите данные об армии, дивизии, корпусе, в которые входит больше всего (меньше всего) военных частей.

Разработанная система, помимо отправки запросов, должна иметь возможности чтения, редактирования, создания и удаления записей из таблиц.

Пользователю должен быть предоставлен графический интерфейс для взаимодействия с базой данных.

**Первичный анализ темы**

В ходе подробного анализа темы и перспектив сложности реализации запросов было проведено представление и осмысление взаимодействия планируемых сущностей, что привело к созданию итоговой структуры базы данных.

Итоговую структуру БД можно условно разделить на 4 группы сущностей:

* относящиеся к войсковой части,
* относящиеся к военнослужащему,
* относящиеся к вооружению и технике,
* относящиеся к подразделениям.

Каждая группа инкапсулирует некоторое количество таблиц.

Всего было выделено 16 сущностей, главными из которых являются “Войсковая часть” и “Военнослужащий".

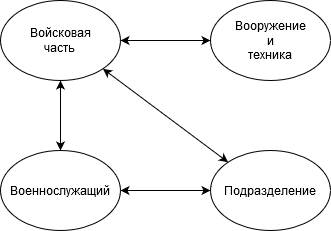


Рисунок – Отношение групп сущностей

Моделирование базы данных будет вестись с использованием нотации Мартина (“воронья лапка”) и нотации Питера Чена, как одни из самых популярных и легких в восприятии.

Таким образом, в разделе была подробно описана поставленная задача и видение ее реализации.

**1.2 Анализ средств для решения задачи**

Поставленная задачи может быть решена с помощью приложения, которое использует клиент-серверную архитектуру. Клиентская часть отвечает за отображение информации и отлавливание действий пользователя. Серверная часть состоит из базы данных и сервера, содержащего бизнес-логику.

Главным плюсом данного подхода является модульность приложения, что позволяет иметь одну серверную часть и несколько клиентских.

Для реализации клиентской части можно использовать один или несколько из типов приложения: локальное устанавливаемое приложение, веб-приложение, мобильное приложение.

В качестве основного подхода будет использоваться веб-приложение, т.к. не требует большого объема знаний и применим на практике.

Также был выбран высокоуровневый язык программирования C#, главным плюсом последних версий которого является кроссплатформенность и мощная библиотеке классов, СУБД MySQL для хранения данных, среда JetBrains Rider для написания кода, среда DataGrip для создания и редактирования объектов базы данных, платформа ASP.NET для создания веб-приложения.

**Дополнительные инструменты**

.NET 5 - модульная платформа для разработки программного обеспечения с открытым исходным кодом. К основным достоинствам платформы относят кроссплатформенность, мощную библиотеку классов, разнообразие технологий, производительность.

MySQL Workbench - инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое бесшовное окружение для системы баз данных MySQL.

ASP .NET Core - кроссплатформенной, высокопроизводительной средой с открытым исходным кодом для создания современных облачных приложений, подключенных к Интернету.

Entity Framework Core — современный модуль сопоставления "объект — база данных" для .NET. Он поддерживает запросы LINQ, отслеживание изменений, обновления и миграции схемы. EF Core работает с многими базами данных, включая базы данных SQL (локальные и в Azure), SQLite, MySQL, PostgreSQL и Azure Cosmos DB.

**вывод**

**2 ТЕХНОЛОГИЯ РЕАЛИЗАЦИЯ СТРУКТУРЫ БАЗЫ ДАННЫХ**

Во время проектирования баз данных перед программистом всегда стоит перечень задач, а само проектирование разделено на несколько этапов.

Основные задачи, которые стоят перед программистом:

* Обеспечение хранения в БД всей необходимой информации.
* Обеспечение возможности получения данных по всем необходимым запросам.
* Сокращение избыточности и дублирования данных.
* Обеспечение целостности данных.

Моделирование БД состоит из трех этапов, на каждом из которых создают конкретную модель данных: концептуальную, логическую и физическую.

**2.1 Концептуальная модель**

Концептуальное (инфологическое) проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных.

Конкретный вид и содержание концептуальной модели базы данных определяется выбранным для этого формальным аппаратом. Обычно используются графические нотации, подобные ER-диаграммам.

Чаще всего концептуальная модель базы данных включает в себя описание информационных объектов или понятий предметной области и связей между ними и ограничения целостности, то есть требований к допустимым значениям данных и к связям между ними.

Концептуальная модель представлена в приложении А.

Ключевой сущностью в информационной системе военного округа является “Войсковая часть”. Здесь будут храниться части всех типов, которые будут хранить в отдельной таблице “Тип войсковой части”. У каждой части может быть несколько мест дисклокаций, которые можно найти через связующую сущность “Место дислокации в/ч”. Также на территории в/ч могут находиться различный сооружения, в которых может проживать личный состав подразделений. Любая в/ч, кроме военного округа, должна быть в составе какой-либо другой, высшей по рангу, войсковой части.

К в/ч привязаны категории техники и вооружения, которые в свою очередь разделяются на отдельные виды.

В войсковую часть могут входить различные подразделения различный типов (“Тип подразделения”), который будут храниться в сущности “Подразделение”.

В в/ч служат военнослужащие, который записаны в таблице “Военнослужащий”. У каждого есть воинское звание, одна или несколько специальностей, карточка с дополнительной информацией о в/с. В/с проходит службу в одной в/ч и может командовать ей, а также быть в составе или командовать каким-либо подразделением. Военнослужащий также может находиться в подчинении у другого в/с.

**2.2 Логическая модель**

Логическое (даталогическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных даталогическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

Преобразование концептуальной модели в логическую модель, как правило, осуществляется по формальным правилам. Этот этап может быть в значительной степени автоматизирован.

Для каждой сущности придумаем название с использование латинских букв. Например, войсковая часть – unit, военнослужащий – soldier, и т.д. Полный список названий приведен в таблице ниже.

Таблица - Листинг названий сущностей

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Описывает | Название | Описывает |
| building | Сооружение | unit | Войсковая часть |
| location\_type | Тип места дисклокации | unit\_location | Место дислокации в/ч |
| military\_rank | Звание | unit\_type | Тип в/ч |
| occupation | Специальнось | vehicle | Техника |
| soldier | Военнослужащего | vehicle\_category | Категория техники |
| soldier\_occupation | Специальность в/с | weapon | Вооружение |
| soldier\_rank\_info | Инф. о звании в/с | weapon\_category | Категория вооружения |
| subunit | Подразделение | subunit\_type | Тип подразделения |

Следующим шагом определяем атрибуты сущностей.

“Войсковая часть” имеет идентификатор unit\_id, номер unit\_number, название name, тип войсковая части как внешний ключ unit\_type\_id, ссылающийся на сущность unit\_type, родительская войсковая часть как внешний ключ parent\_unit\_id, ссылающий на unit\_id этой же сущности и командир как внешний ключ leader\_id, ссылающий на soldier\_id в soldier.

“Тип войсковой части” имеет идентификатор unit\_type\_id, название name и является справочной сущностью.

“Тип места дислокации” имеет идентификатор location\_type\_id, название name и является справочной сущностью.

“Место дислокации в/ч” является связующей сущностью, которая разрешает отношение многие ко многим. Сущность имеет название name, внешние ключи unit\_id и location\_type\_id.

“Сооружение” имеет идентификатор building\_id, внешний ключ unit\_id и число проживающих подразделений number\_of\_living\_subunits.

“Категория техники” содержит первичный ключ vehicle\_category\_id, внешний ключ unit\_id и название категории name. На эту сущность ссылается “Техника”, которая содержит ключ vehicle\_id, внешний ключ vehicle\_category\_id, название техники name и количество amount.

“Категория вооружения” и “Вооружение” построены аналогично.

“Подразделение” имеет идентификатор subunit\_id, внешний ключ типа подразделения subunit\_type\_id, номер командира как внешний ключ leader\_id, родительская войсковая часть как внешний ключ parent\_unit\_id и название name.

“Тип подразделение” содержит идентификатор subunit\_type\_id, название name и является справочной сущностью.

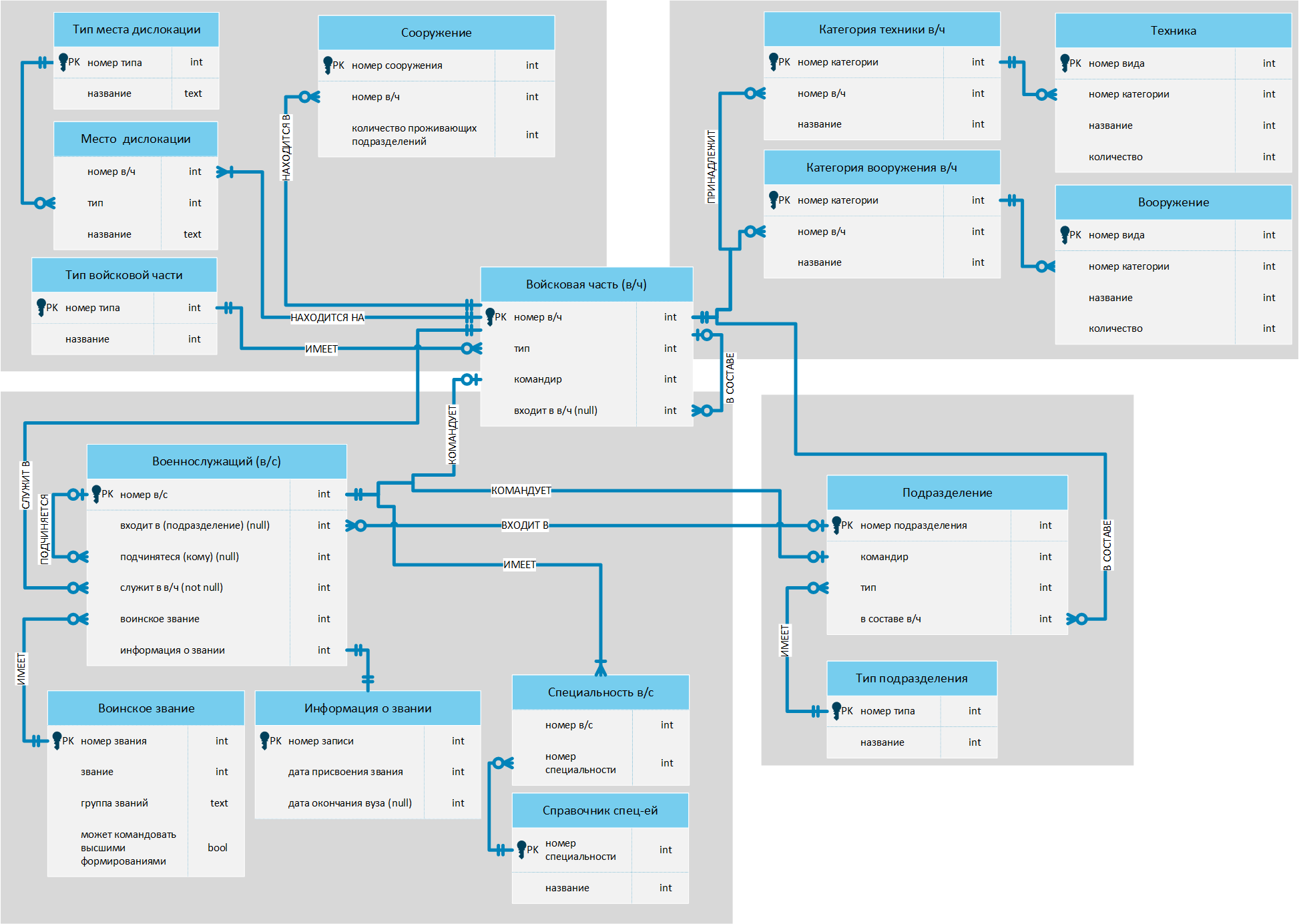


Рисунок – Логическая модель, военнослужащий

Для “Военнослужащего” определены идентификатор soldier\_id, места службы как внешние ключи serving\_unit\_id и serving\_subunit\_id, командир как внешний ключ на атрибут soldier\_id этой же сущности, имя name, воинское звание как внешний ключ military\_rank\_id и номер карточки rank\_info\_id с дополнительной информацией о в/с.

“Специальность” имеет идентификатор occupation\_id и название name и является справочником.

“Специальности военнослужащего” является связующей сущностью, которая разрешает отношение многие ко многим. Имеет внешние ключи soldier\_id и occupation\_id.

“Воинское звание” определено идентификатором military\_rank\_id, название name, названия группы званий rank\_group и поля command\_higher\_unit, указывающее, может ли военнослужащий с данным званием командовать высшим войсковыми образованиями.

**2.3 Физическая модель**

Физическое проектирование — создание схемы базы данных для конкретной СУБД. Специфика конкретной СУБД может включать в себя ограничения на именование объектов базы данных, ограничения на поддерживаемые типы данных и т. п. Кроме того, специфика конкретной СУБД при физическом проектировании включает выбор решений, связанных с физической средой хранения данных (выбор методов управления дисковой памятью, разделение БД по файлам и устройствам, методов доступа к данным), создание индексов и т. д.

Для всех таблиц определим общие параметры:

* Идентификатор и внешний ключ – тип INTEGER,
* Текстовые поля – тип VARCHAR
* Для поля command\_higher\_unit таблицы military\_rank – тип BOOLEAN.

Выбранная СУБД позволяет использовать различные типы данных, дополнять поля атрибутами, объявлять индексы и ограничения.

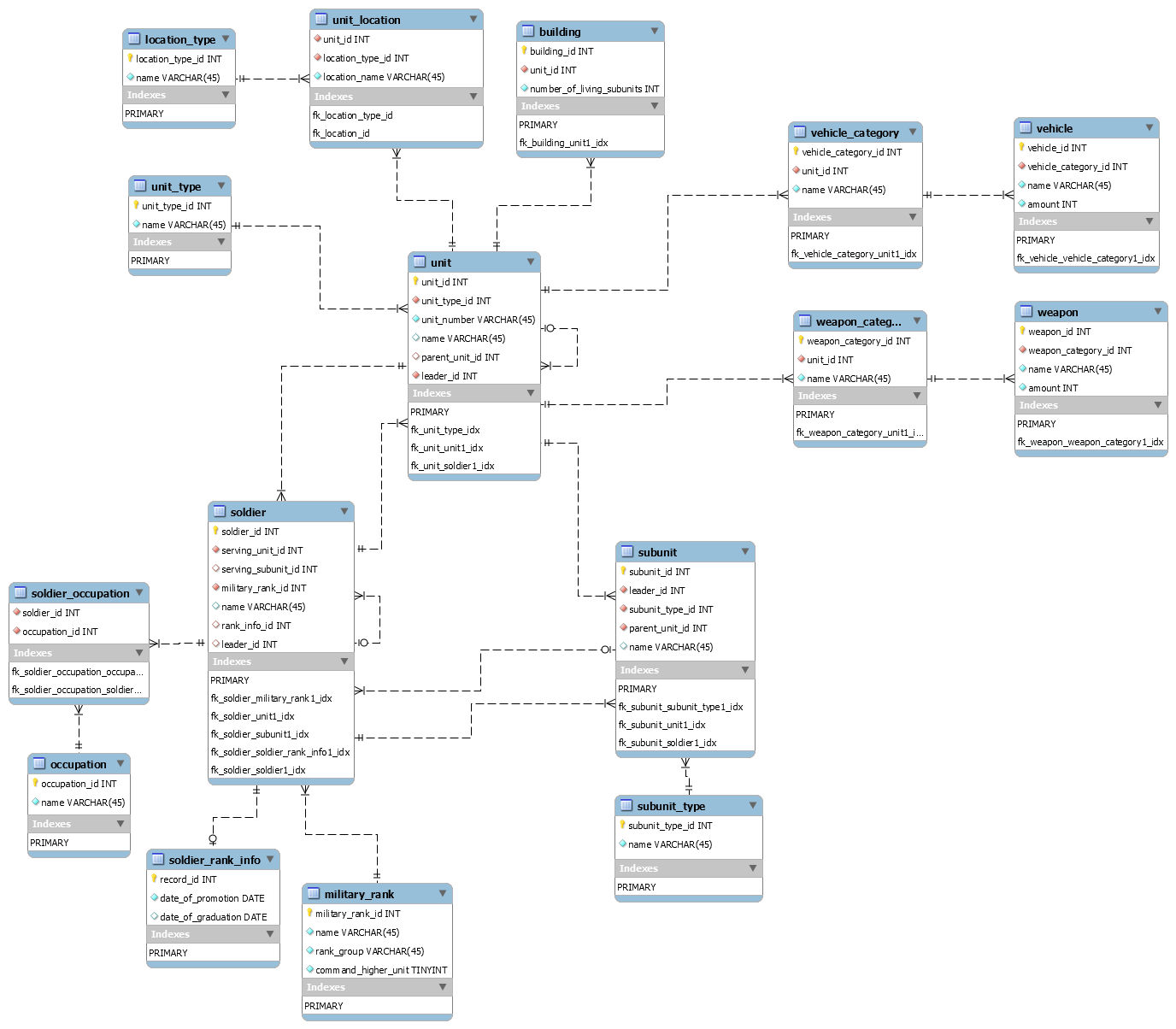


Рисунок – Физическая модель, войсковая часть

Для некоторых полей таблиц добавлены ограничения null или nut null, указывающие, может ли поле не иметь значения или иметь. Например, для поля leader\_id из таблицы unit укажет атрибут NULL. Так, не имея записей в таблице soldier мы сможем создать новую войсковую часть. Затем, добавив военнослужащего, его soldier\_id можно указать как leader\_id в таблице unit.

Индексы — важнейший инструмент баз данных, ускоряющий поиск. Он не бесплатен, создавать много индексов без лишней необходимости не стоит — индексы занимают дополнительную память, и при любом обновлении проиндексированных данных СУБД должна выполнять дополнительную работу по поддержанию индекса в актуальном состоянии.

Использование индексов не ограничено, поэтому они установлены по умолчанию на первичные ключи и внешние ключи во всех таблицах.

Таблицы, предполагающие добавление в них новых записей, имеют первичный ключ с атрибутом AUTO\_INCREMENT, позволяющей базе данных самостоятельно устанавливать значений в поле.

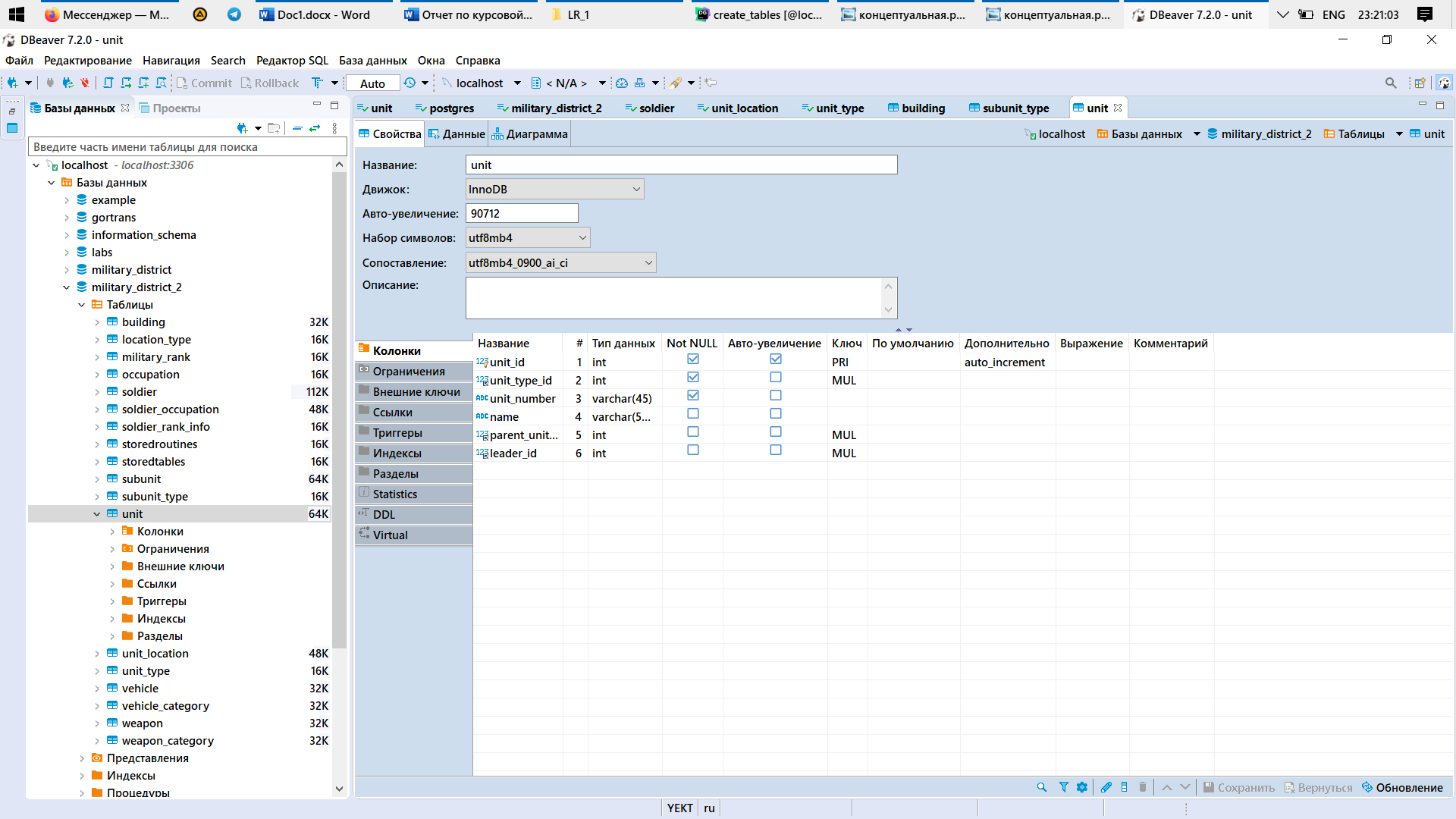


Рисунок – Поля таблицы unit

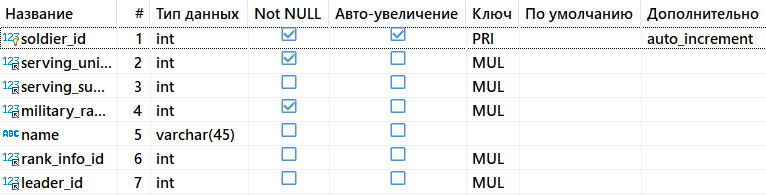


Рисунок – Поля таблицы soldier

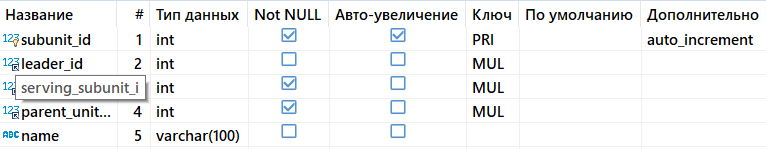


Рисунок – Поля таблицы subunit

На рисунках выше представлено описание ключевых таблиц в базе данных.

**Вывод**, в разделе были подробно описаны шаги по проектированию схемы данных с использованием концептуальной, логической и физической моделей. Реализация схемы будет производиться на основе СУБД MySQL.

**3 ТЕХНОЛОГИЯ РЕАЛИЗАЦИИ ВЕБ-ПРИЛОЖЕНИЯ**

Разработка веб-приложения включает в себя три этапа:

* 1. Создание базы данных: таблицы, представления, хранимые функции и процедуры, роли и пр.,
  2. Создание бизнес-логики: проект на языке C#, содержащий классы, контролирующие ход работы приложения,
  3. Создание графического интерфейса пользователя: изменение содержимого файлов, отвечающих за отображение информации в браузере.

Этапы 2 и 3 тесно связаны, т.к. выбранный подход к разработке и используемые инструменты позволяют моментально генерировать файлы, используемые браузером для формирования графического интерфейса.

**3.1 Создание базы данных**

Сперва, для изолирования таблиц от других системных данных базы данных, создадим новую схему military\_district с права администратора. Так мы сможем создавать различные объекты, не беспокоясь об конфликте прав или имен.

Далее следуют таблицы, которые создаются на основе разработанной ранее физической модели. В каждой таблице будет присутствовать один первичный ключ с атрибутом AUTO\_INCREMENT, один или несколько индексов.

Для обеспечения целостности базы используем внешние ключи, связывающие поле из одной таблицы с полем из второй таблицы. Некоторые ключи имеют атрибут “ON DELETE SET NULL”, который указывает на то, что поле записи станет null, если внешний ключ не найдет запись в связанной таблице.

DML операции к таблицам не требуют какой-либо инкапсуляции, т.к. они будут осуществляться средствами языка C#, точнее ORM (объектно-реляционное представление) фреймворком платформы .NET Entity Framework. Подробнее об использовании фреймворка речь пойдет в следующем разделе.

Следующим шагом создаем процедуры, принимающие аргументы для и возвращающие результат внутреннего запроса с параметрами. Процедуры инкапсулируют заложенный в их теле запрос и позволяют много кратно вызываться себя.

Некоторые запросы комплексны по своему описанию и подразумевают несколько версий реализации. Решением стало разбиение на составные запросы и создание для них отдельных процедур. Из-за такого подхода, конечное количество процедур вдвое больше количества запросов.

Помимо основных таблиц, была создана таблица storedroutines, которая обеспечивает связность хранимых процедур с запросами, которые они реализуют, и хранит поясняющую информацию о процедуре, например название и количество аргументов.

Таблица – Неполый листинг процедур

|  |  |
| --- | --- |
| Название | Аргументы |
| Show\_ChainOfSubunitsAndLeaders | номер в/ч |
| Show\_Soldiers\_WithGroupRank | номер в/ч:группа званий |
| Show\_Soldiers\_WithRank | номер в/ч:звание |
| Show\_ChainOfSoldierLeaders | номер в/с |
| Show\_Locations | номер в/ч |
| Show\_Vehicles | номер в/ч |
| Show\_Vehicles\_WithCategory | номер в/ч:категория машин |
| Show\_Vehicles\_WithName | номер в/ч:вид машины |

Стоит отметить, что большинство объектов БД MySQL необходимо создавать вручную или через средства среды разработки. Для создания таблиц же можно воспользоваться уже упомянутым программой MySQL WorkBench, а точнее функцией создания базы данных из уже разработанной физической модели (“Forward engineer to database”). Среда сгенерирует код для создания таблиц, ограничений, индексов и внешних ключей, или создаст таблицу в указанной схеме.

Таким образом, в базе данных были созданы таблицы на основе физической модели и процедуры с параметрами для выполнения запросов. На следующем этапе к коде программы мы будем явно указывать имя таблицы или процедуры, с которой мы хотим работать.

**3.2 Создание бизнес-логики**

**3.2.1 Схема MVC**

За бизнес-логику, т.е. то, как приложение обрабатывает запросы от пользователя и генерирует ему ответы, будет отвечать приложение на языке программирования C#.

Стартом послужит шаблонный проект ASP.NET MVC с использованием Entity Framework для операций CRUD на платформе APS.NET Core.

Model-View-Controller (MVC) — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

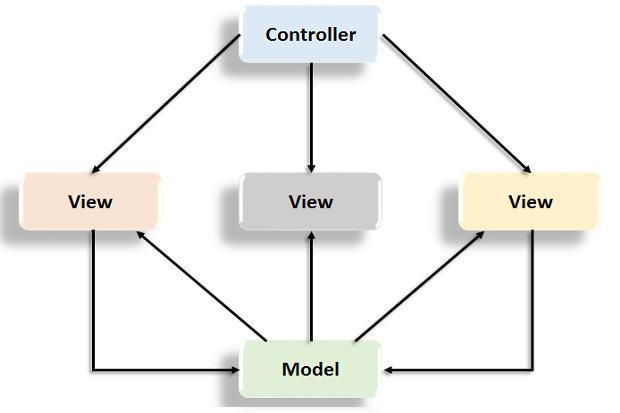


Рисунок – Схема разделение данных MVC

Подход MVC подразумевает приложение как множество взаимодействующих между собой контроллеров, к каждому из которых привязаны определенные модели и представления.

Тогда цепочка вызовов после действия пользователя будет выглядеть:

*“Действие пользователя – отправка запроса серверу – сервер передает управление контроллеру, возможно с моделью – контроллер производит манипуляции над данными и наполняет модель новыми данными – контроллер формирует ответ, возможно с данными и передает управление представлению на сервере – представление на сервере создает интерфейс в соответствии с моделью – ответ отправляется пользователю – браузер получает ответ и обрабатывает его”.*

**3.2.2 Создание моделей**

Модель предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

**Модели таблиц**

Взаимодействие программы и базы данных будет происходить с помощью ORM - технологии Entity Framework. Она позволяет представить схему базы данных в виде классов с различными отношениями. Фреймворк позволяет использовать парадигмы:

* Code First – создание базы данных и таблиц на основе описанной в программе модели,
* Database First – создание кода приложения на основе созданной базы данных.

В проекте используется второй подход, т.к. на данном этапе имеется готовая база данных.

Компания, разрабатывающая базу данных, также поддерживает пакет Entity Framework, нацеленный только на их продукт. Такой пакет от компании Oracle, владеющей MySQL, называется MySql.EntityFrameworkCore. С помощью классов этого пакеты происходит взаимодействие кода программы на C# с базой данных MySQL.

Процесс создания моделей заключается в описании классов, каждый из которых будет соответствует одной таблице в базе данных. Это трудоемкий процесс, в котором неопытный разработчик может совершить ошибки, тормозящие разработку. Поэтому лучше применим автоматизацию к созданию моделей. Для этого воспользуемся пакетом Microsoft.EntityFrameworkCore.Tools, инструмент которого проведет сканирование схемы базы данных и создаст по модели на каждую таблицу. Сделать это можно с помощью следующей команды:

“*Scaffold-DbContext строка\_подключения MySql.EntityFrameworkCore -OutputDir scaffolded -f”*

Команду необходимо использовать в терминале среды разработки и только один раз.

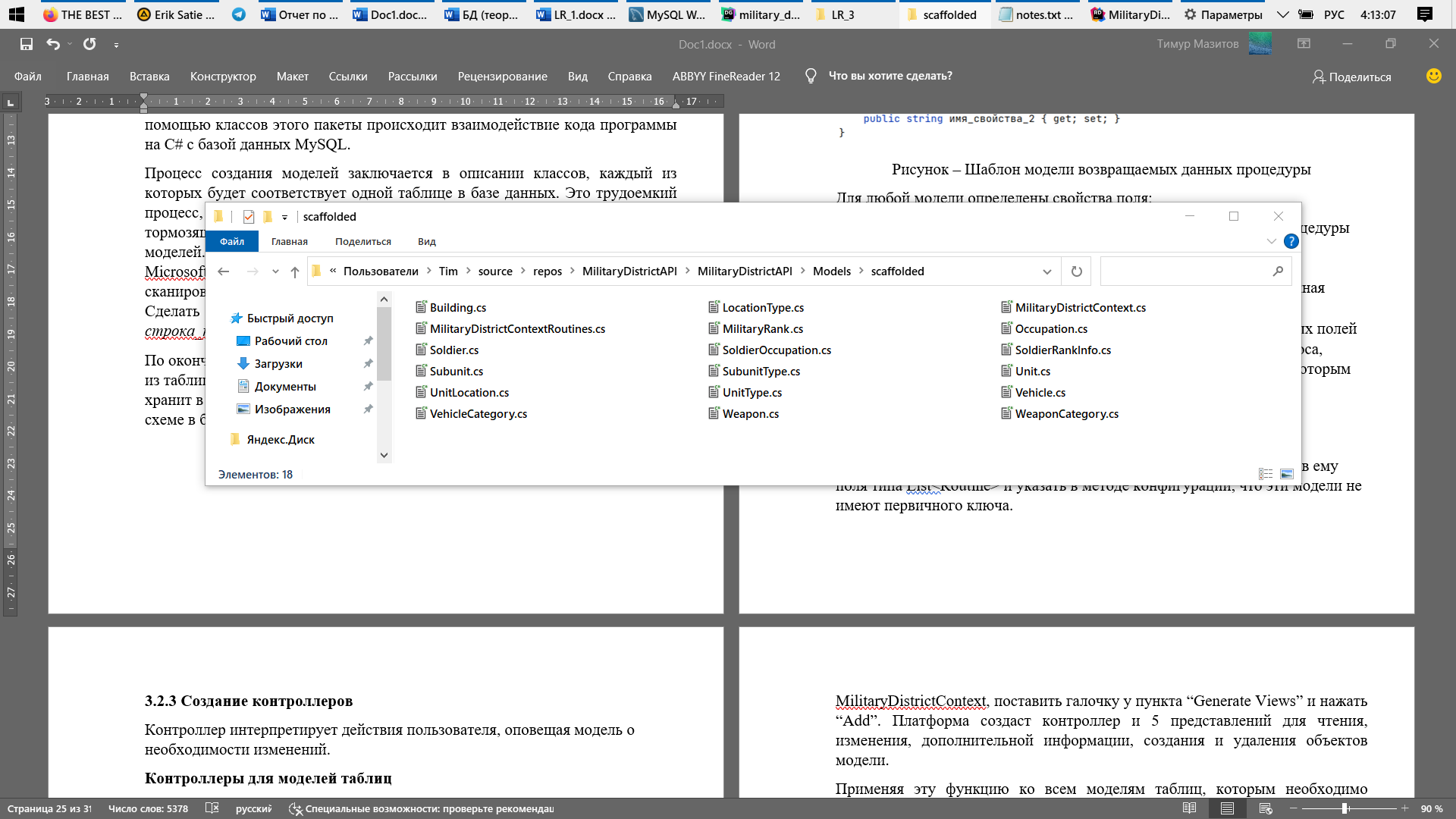


Рисунок – Содержание папки scaffolded

По окончанию процесса, в папке scaffolded создадутся 16 моделей на каждую из таблиц и класс MilitaryDistrict - наследник от класса DbContext, который хранит в своих свойства объекты моделей таблиц и сопоставляется всей схеме в базе данных.

**Модели процедур**

Вызов и обработка результатов хранимой процедуры осуществляется через соответствующую модель. Каждая модель описывает столько свойств, сколько полей возвращает процедура, и такого типа, какого типа возвращаемые данные процедуры. Поэтому необходимо были описаны 13 различных моделей по следующему шаблону класса, в котором определены свойства поля:

* SQL – строка sql-запроса, где вызывается процедура. Имя процедуры указывается именно здесь. Количество пар фигурных скобок соответствует количеству аргументов процедуры.
* NAME – порядковый номер запроса. По нему выбирается нужная модель.
* Свойства, количество которых равно количеству возвращаемых полей запроса. В атрибуте Column указывается точно имя поля запроса, которому соответствует свойство, JsonProperty – название, с которым поле будет отображается в браузере.

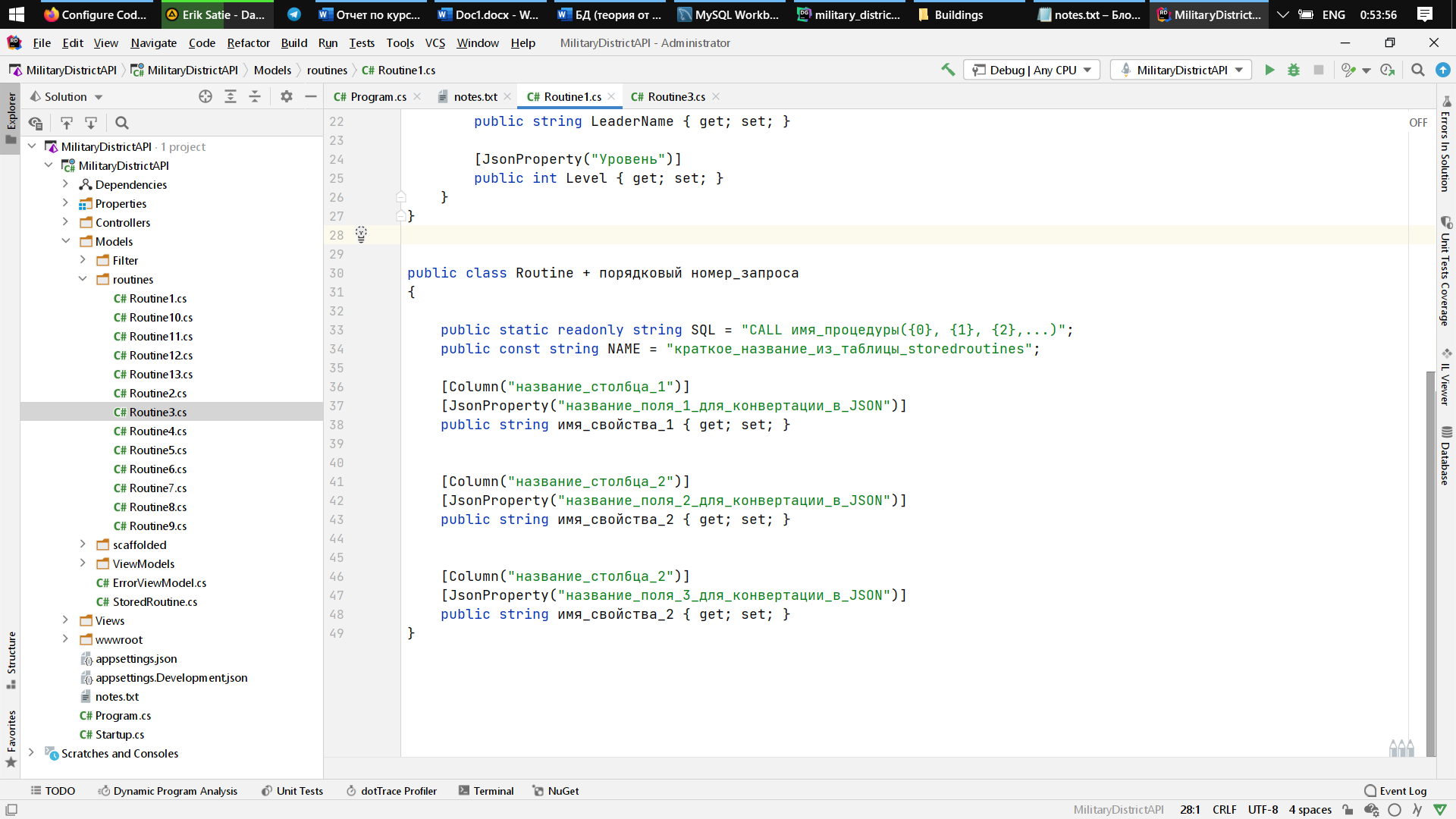


Рисунок – Шаблон модели возвращаемых данных процедуры

После выполнения рутинной работы по описанию моделей в папке Model/routines находятся 13 классов: по одному классу на запрос.

Также необходимо внести изменения в класс MilitaryDistrict, добавив ему поля типа List<Routine> и указать в методе конфигурации, что эти модели не имеют первичного ключа.

**3.2.3 Создание контроллеров**

Контроллер интерпретирует действия пользователя, оповещая модель о необходимости изменений.

**Контроллеры для моделей таблиц**

Для каждой не ассоциативной модели создается класс-контроллер со следующими методами:

* Конструктор, принимающий объект класса MilitaryDistrictContext
* Index без параметров, который будет возвращать список всех объектов модели из базы данных,
* Details с параметром id, который возвращает подробную информацию о модели с указанным id,
* Create без параметров, который возвращает параметры для создания нового объекта модели,
* Create c параметром object класса модели, который добавляет новый объект в базу данных,
* Edit с параметром id, который возвращает данные модели для редактирования,
* Edit с параметров object класса модели, который сохраняет изменения в базе,
* Delete с параметром id, который удаляет объект с id из базы данных.

Полученные контроллеры крайне схожи по своему содержанию, поэтому их создание нужно делегировать встроенной функции создания.

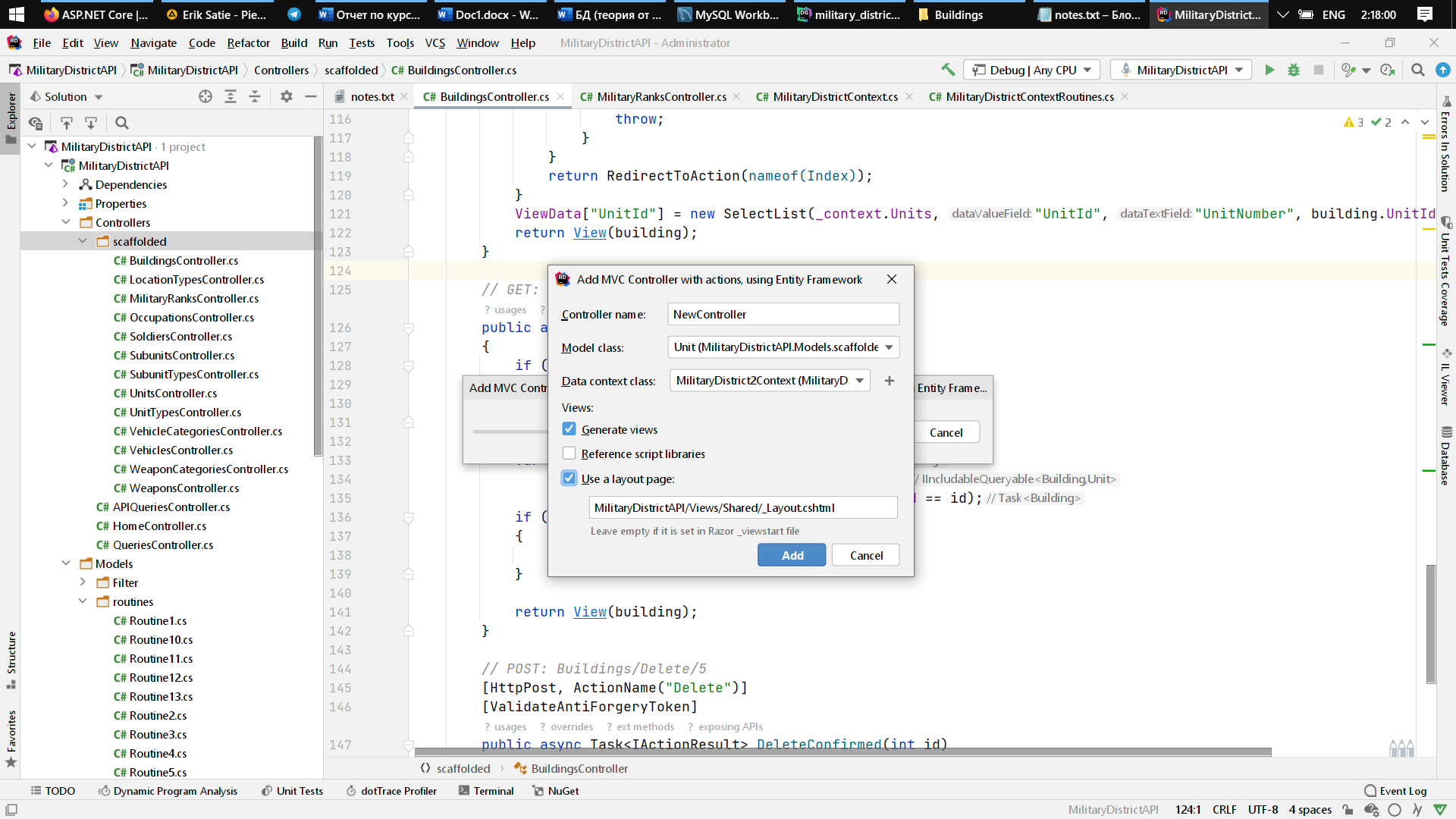


Рисунок – Шаблонизация модели

Для ее вызова необходимо правой кнопкой мыши кликнуть на папку “Controllers”, выбрать пункт “Add -> Scaffolding item”, написать имя в формате “МодельController”, указать класс модели и класс-контекст MilitaryDistrictContext, поставить галочку у пункта “Generate Views” и нажать “Add”. Платформа создаст контроллер и 5 представлений для чтения, изменения, дополнительной информации, создания и удаления объектов модели.

Применяя эту функцию ко всем моделям таблиц, которым необходимо взаимодействие с пользователем, можно получить готовую реализацию запросов DML, что значительно ускорит ход разработки приложения.

**Контроллер для моделей процедур**

Сущность запросов подразумевает не хранение результатов их выполнения, а мгновенную передачу их пользователю. Поэтому контроллер APIQueries для моделей процедур содержит только один публичный метод, который будет принимать JSON документ с номером запроса и аргументами и возвращать JSON документ с результатом выполнения запроса.

Для такой задачи лучше всего подойдет API-контроллер, который оперирует именно JSON документами.

Также необходимо определить метод ExecuteQuery для класса MiltaryDistrictContext, который будет определять номер запроса, выбирать модель процедуры, выполнять запрос и возвращать его результат.

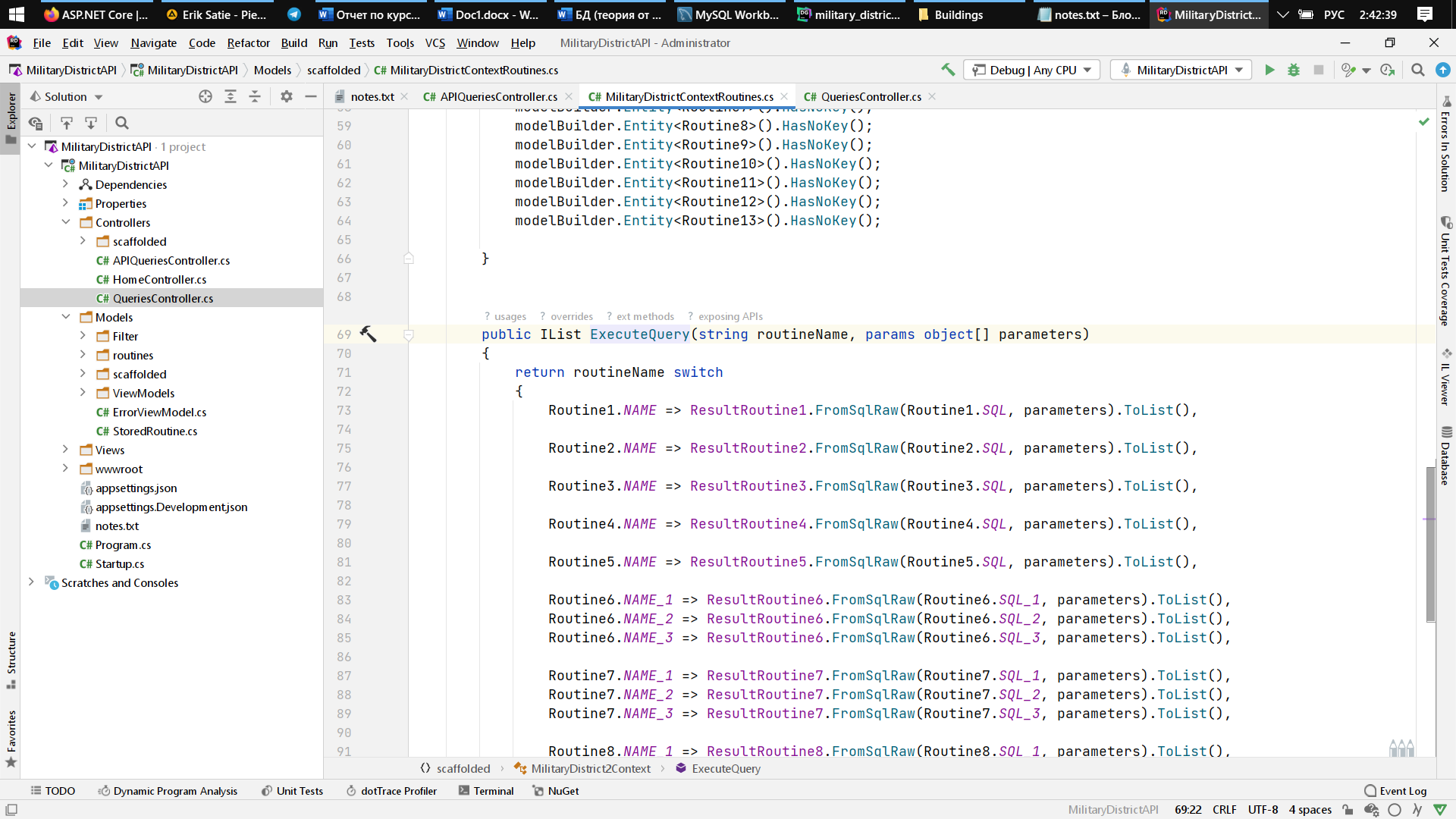


Рисунок – Вызов процедур в коде

В результате формируется список объектов выбранной модели, который с помощью пакета NewtonSoft конвертируется в JSON документ и возвращает его как результат запроса.

**Прочие контроллеры**

Помимо контроллеров для таблиц и процедур, необходимо 2 навигационных контроллера:

* Контроллер Home не обрабатывает никаких данных из базы, а только переключает между представлениями запросов и таблиц.
* Контроллер Queries отвечает за получение данных о доступных запросах. Оперирует данными из таблицы storedroutines.

Все не CRUD-контроллеры расположены в папке Controllers, а контроллеры для таблиц в подпапке scaffolded.

**3.3 Создание графического интерфейса пользователя**

Визуальной компонентой в схеме MVC является представление. Оно отвечает за отображение данных модели пользователю, реагируя на изменения модели.

**Особенности разработки представлений**

Интуитивный и приятный интерфейс вносит огромный вклад в опыт работе пользователя с приложением. Поэтому за оформление страницы будет отвечать библиотека Bootstrap. В ней содержится большое многообразие готовых классов и стилей, который можно применять к различным элементам страницы. По умолчанию, библиотека встроена в проект ASP.NET и может быть использована из коробки.

В проекте ASP.NET используются файлы с расширение .html и .cshtml. Особенностью последних является возможность использования кода языка C#, предваренного символом “@”. Данная конструкция позволяет избежать повторяемости языка разметки и позволяет удобно работать с моделями, переданными из контроллеров.

Все файлы, отвечающие за итоговое представление в браузере, находятся в папке “Views”. В ней расположены папки, имена которых совпадают с именами ранее описанных контроллеров. Содержание папок разнится: если модель предполагает изменение данных, то необходимо иметь страницы, отвечающие за добавление, изменение и удаление данных; если модель дает возможность только просматривать данные, то хватит одного файла для отображения списка элементов.

Также, технология Razor-pages предлагает использовать шаблоны страниц. Суть такого подхода заключается в том, что код из файла представления будет вставляться в код шаблона, и только затем отсылаться клиенту. Данная особенность позволяет избежать повторяемости кода и упрощает процесс внесения глобальных изменений.

**Конструирование интерфейса**

В ходе анализа темы и поставленных задач, было определено, что веб приложение будет включать два раздела:

* Таблицы, где пользователю будет доступно содержание таблиц и возможность их редактирования,
* Запросы, где пользователь сможет найти необходимый запрос и выполнить его.

Ссылки на разделы находятся на верхней навигационной ленте, через которую пользователь может попасть в желаемый раздел.

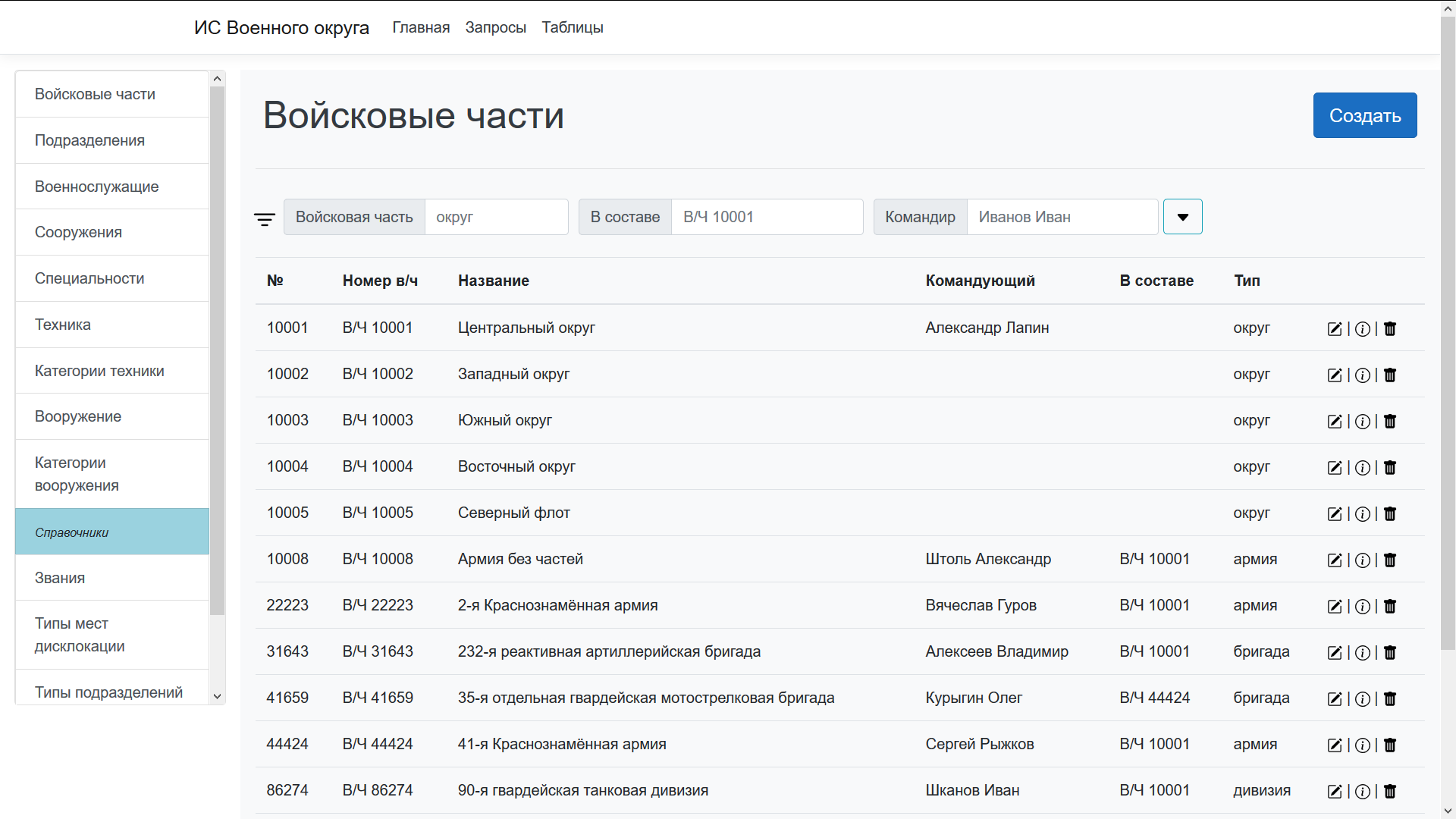


Рисунок – Страница “Войсковые части” в разделе Таблицы

Страница раздела поделена на две части: список элементов и взаимодействия с пользователем. Кликнув по элементу списка, пользователь направляется на соответствующий подраздел.

В разделе таблицы пользователю доступны редактируемые таблицы и справочники, данные в которых нельзя изменить. Перейдя в подраздел выбранной таблицы, пользователь видит название таблицы, кнопку “Создать” для добавления нового элемента в таблицу и список записей в таблице. Для каждой записи есть три функции: изменить, показать подробности, удалить. Для некоторых таблиц также доступна функция фильтрации записей по полю данных.

В разделе запросы пользователю доступен список запросов. На странице каждого запроса подробно указано содержание запроса, поле для ввода данных и кнопка для отправки. После выполнения запроса пользователю предоставляется неизменяемая таблица, содержащая результат выполнения запроса.

**Вывод**, итоговым результатом данного этапа является успешно сконструированная база данных, веб приложения к ней на языке C# на платформе ASP.NET, с пользовательским интерфейсом для работы с таблицами и отправления запросов.

**ЗАКЛЮЧЕНИЕ**

Для выполнения проекты были использованы MS Visio для визуального моделирования, СУБД MySQL для хранения данных, MySQL Workbench для моделирования базы данных, среда DataGrip для редактирования объектов базы данных, среда Rider для написания кода на C# и редактирования файлов различных расширений, платформа ASP.NET для разработки веб-приложения.

В ходе выполнения курсовой работы были выполнены все поставленные перед разработчиком задачи: смоделирована и создана база данных, успешно заполнена валидными данными, реализована работа с содержимым таблиц, реализованы запросы к таблицам.

Цель курсовой работы была достигнута – разработана информационная система военного округа.

Во время моделирования были использованы абстрагированные сущности, у которых выделены ключевые свойства, значимые для выполнения запросов. Поэтому в будущем сущности могут быть расширены и дополнены.

Более детальное рассмотрение некоторых сущностей может привести к их разбиению на подсущности, которые вместе образуют новую базу данных. Так, например, может быть встроена база данных отдела кадров или склада с вооружением и техникой.

Неиспользованные возможности базы данных, такие как роли и распределение прав, могут быть успешно включены в текущую концепцию информационной системы и внедрены на уровне пользователя.

Разработанная система имеет возможности расширения, поэтому может считаться перспективной.

За время выполнения работы были отточены навыки взаимодействия с базой данных MySQL через технологию EntityFramework, расширены представления о разработке клиент-серверного приложения с использованием API-методов на платформе ASP.NET, вынесены ценные уроки о возникающих ошибках и способах их избежать, получен незаменимый опыт поиска необходимой информации в печатных издания и в сети Интернет.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Документация по ASP.NET [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-5.0>. (дата обращения: 1.05.21).
2. MySQL Documentation [Электронный ресурс]. URL: <https://dev.mysql.com/doc> .(дата обращения: 1.05.21).
3. Грабер М. Введение в SQL : пер. с англ. Москва : Лори, 2011. 379 с., 25 усл. печ. л.

**ПРИЛОЖЕНИЕ A**

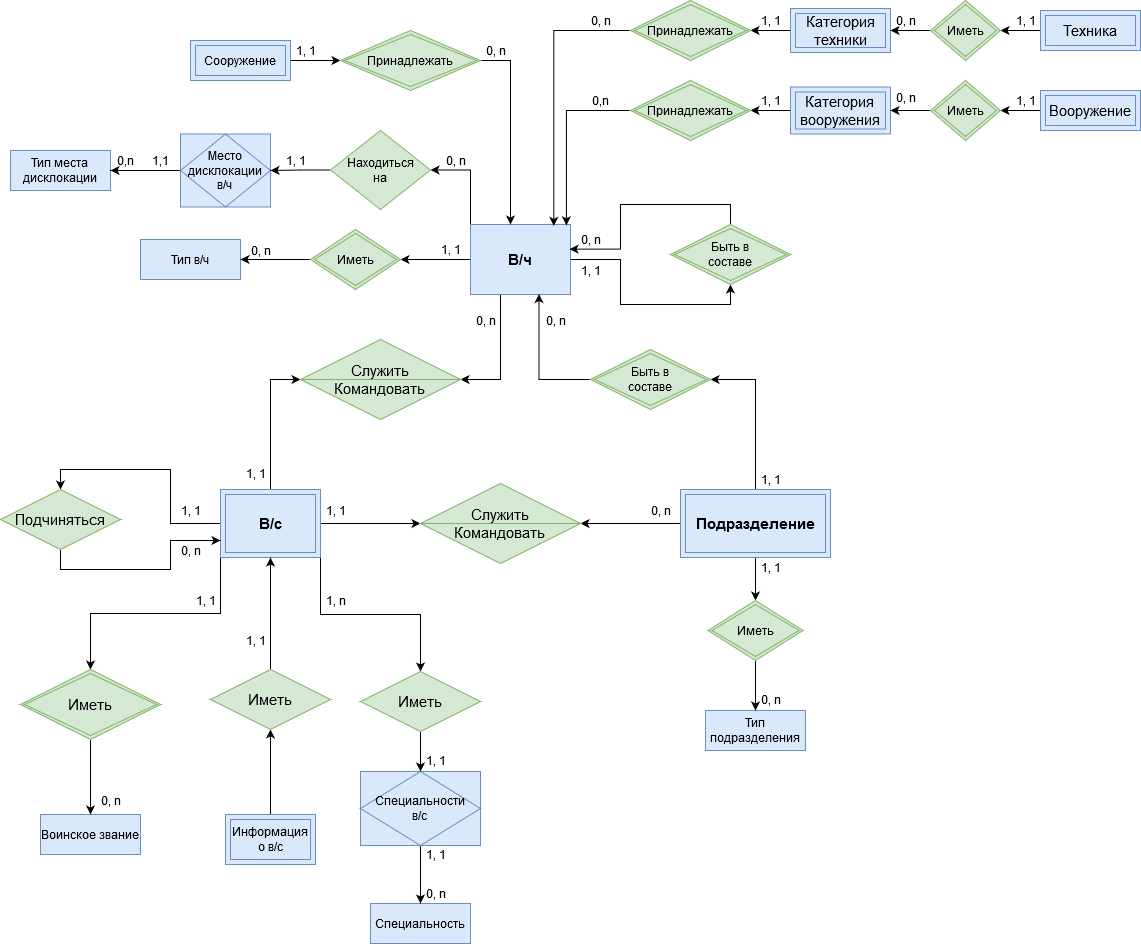


Рисунок – Концептуальная модель

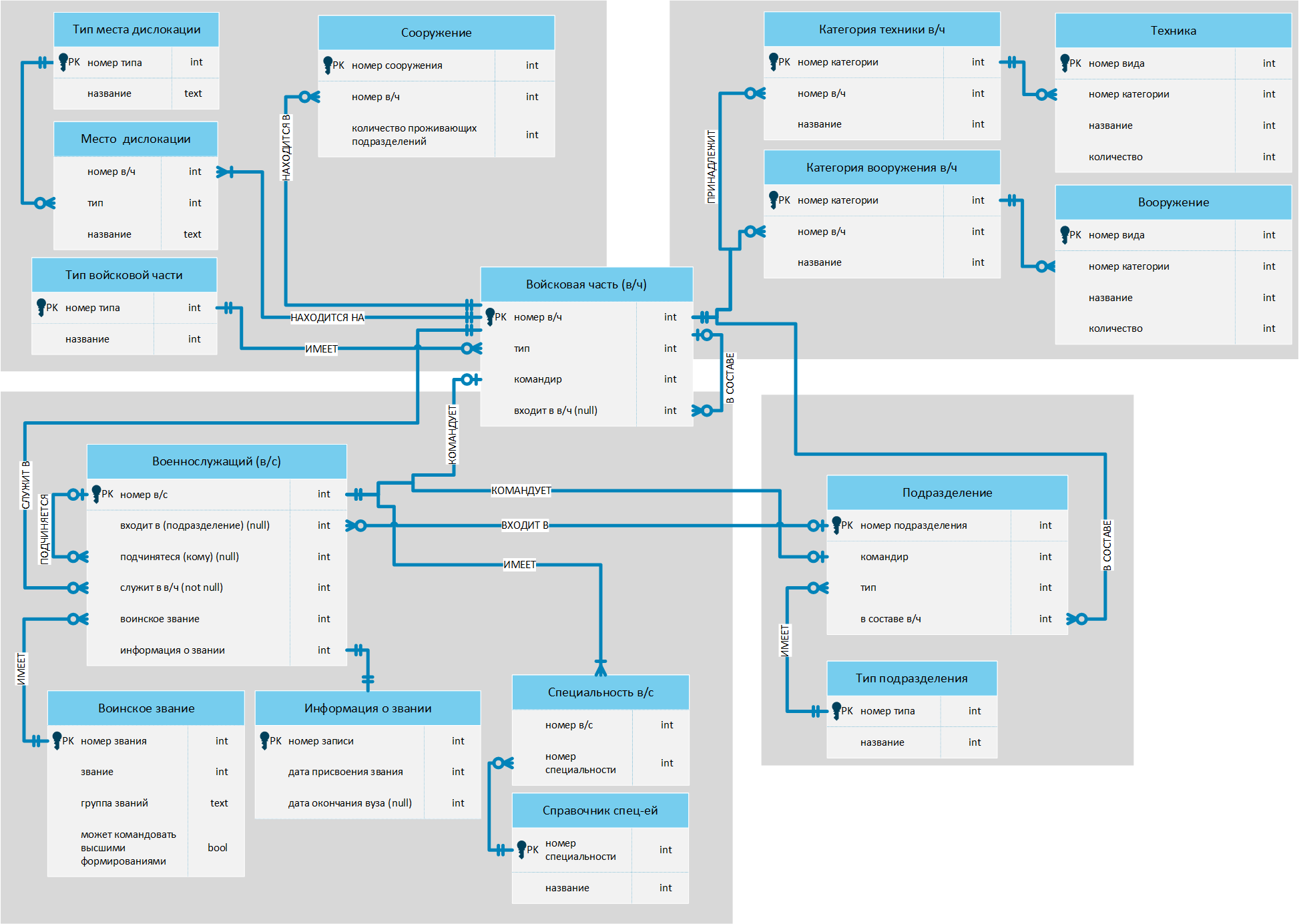


Рисунок – Логическая модель

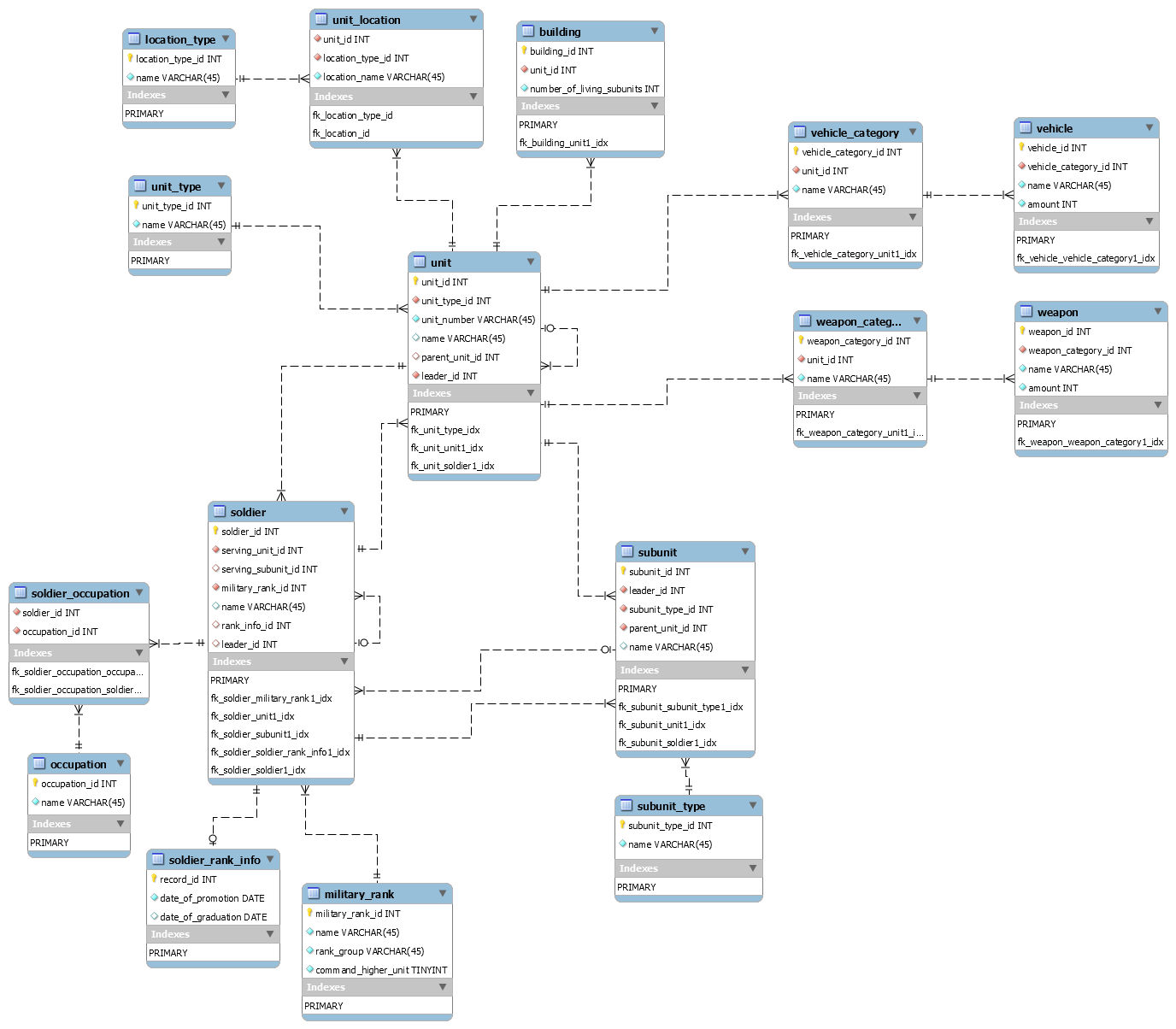


Рисунок – Физическая модель

Таблица – листинг процедур

|  |  |
| --- | --- |
| Название | Аргументы |
| Show\_ChainOfSubunitsAndLeaders | номер в/ч |
| Show\_Soldiers\_WithGroupRank | номер в/ч:группа званий |
| Show\_Soldiers\_WithRank | номер в/ч:звание |
| Show\_ChainOfSoldierLeaders | номер в/с |
| Show\_Locations | номер в/ч |
| Show\_Vehicles | номер в/ч |
| Show\_Vehicles\_WithCategory | номер в/ч:категория машин |
| Show\_Vehicles\_WithName | номер в/ч:вид машины |
| Show\_Buildings | номер в/ч |
| Show\_Buildings\_LivingMoreThan | номер в/ч:число подразделений |
| Show\_Buildings\_NonLiving | номер в/ч |
| Show\_Units\_WithMoreVehiclesThan | вид машины:количество |
| Show\_Units\_WithNoVehicle | вид техники |
| Show\_Weapons | номер в/ч |
| Show\_Weapons\_WithCategory | номер в/ч:категория вооружения |
| Show\_Weapon\_WithName | номер в/ч:вид вооружения |
| Show\_Occupations\_WithMoreSpecialistsThan | номер в/ч:количество специалистов |
| Show\_Occupations\_WithNoSpecialists | номер в/ч |
| Show\_Soldiers\_WithOccupation | номер в/ч:номер специальности |
| Show\_SoldiersSubunit\_WithOccupation | номер подразделения:номер специальности |
| Show\_Units\_WithMoreWeaponsThan | вид вооружения:количество единиц |
| Show\_Units\_WithNoWeapon | вид вооружения |
| Show\_Units\_WithMostSubunits | номер типа в/ч |
| Show\_Units\_WithLeastSubunits | номер типа в/ч |

**ПРИЛОЖЕНИЕ B – код на языке SQL**

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema military\_district\_2

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema military\_district\_2

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `military\_district\_2` ;

USE `military\_district\_2` ;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`unit\_type`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`unit\_type` (

  `unit\_type\_id` INT NOT NULL,

  `name` VARCHAR(45) NOT NULL,

  PRIMARY KEY (`unit\_type\_id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`military\_rank`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`military\_rank` (

  `military\_rank\_id` INT NOT NULL,

  `name` VARCHAR(45) NOT NULL,

  `rank\_group` VARCHAR(45) NOT NULL,

  `command\_higher\_unit` BOOL NOT NULL,

  PRIMARY KEY (`military\_rank\_id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`subunit\_type`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`subunit\_type` (

  `subunit\_type\_id` INT NOT NULL,

  `name` VARCHAR(45) NOT NULL,

  PRIMARY KEY (`subunit\_type\_id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`subunit`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`subunit` (

  `subunit\_id` INT NOT NULL,

  `leader\_id` INT NOT NULL,

  `subunit\_type\_id` INT NOT NULL,

  `parent\_unit\_id` INT NOT NULL,

  `name` VARCHAR(45) NULL,

  PRIMARY KEY (`subunit\_id`),

  INDEX `fk\_subunit\_subunit\_type1\_idx` (`subunit\_type\_id` ASC) VISIBLE,

  INDEX `fk\_subunit\_unit1\_idx` (`parent\_unit\_id` ASC) VISIBLE,

  INDEX `fk\_subunit\_soldier1\_idx` (`leader\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_subunit\_subunit\_type1`

    FOREIGN KEY (`subunit\_type\_id`)

    REFERENCES `military\_district\_2`.`subunit\_type` (`subunit\_type\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_subunit\_unit1`

    FOREIGN KEY (`parent\_unit\_id`)

    REFERENCES `military\_district\_2`.`unit` (`unit\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_subunit\_soldier1`

    FOREIGN KEY (`leader\_id`)

    REFERENCES `military\_district\_2`.`soldier` (`soldier\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`soldier\_rank\_info`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`soldier\_rank\_info` (

  `record\_id` INT NOT NULL,

  `date\_of\_promotion` DATE NOT NULL,

  `date\_of\_graduation` DATE NULL,

  PRIMARY KEY (`record\_id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`soldier`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`soldier` (

  `soldier\_id` INT NOT NULL,

  `serving\_unit\_id` INT NOT NULL,

  `serving\_subunit\_id` INT NULL,

  `military\_rank\_id` INT NOT NULL,

  `name` VARCHAR(45) NULL,

  `rank\_info\_id` INT NULL,

  `leader\_id` INT NULL,

  PRIMARY KEY (`soldier\_id`),

  INDEX `fk\_soldier\_military\_rank1\_idx` (`military\_rank\_id` ASC) VISIBLE,

  INDEX `fk\_soldier\_unit1\_idx` (`serving\_unit\_id` ASC) VISIBLE,

  INDEX `fk\_soldier\_subunit1\_idx` (`serving\_subunit\_id` ASC) VISIBLE,

  INDEX `fk\_soldier\_soldier\_rank\_info1\_idx` (`rank\_info\_id` ASC) VISIBLE,

  INDEX `fk\_soldier\_soldier1\_idx` (`leader\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_soldier\_military\_rank1`

    FOREIGN KEY (`military\_rank\_id`)

    REFERENCES `military\_district\_2`.`military\_rank` (`military\_rank\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_soldier\_unit1`

    FOREIGN KEY (`serving\_unit\_id`)

    REFERENCES `military\_district\_2`.`unit` (`unit\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_soldier\_subunit1`

    FOREIGN KEY (`serving\_subunit\_id`)

    REFERENCES `military\_district\_2`.`subunit` (`subunit\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_soldier\_soldier\_rank\_info1`

    FOREIGN KEY (`rank\_info\_id`)

    REFERENCES `military\_district\_2`.`soldier\_rank\_info` (`record\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_soldier\_soldier1`

    FOREIGN KEY (`leader\_id`)

    REFERENCES `military\_district\_2`.`soldier` (`soldier\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`unit`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`unit` (

  `unit\_id` INT NOT NULL,

  `unit\_type\_id` INT NOT NULL,

  `unit\_number` VARCHAR(45) NOT NULL,

  `name` VARCHAR(45) NULL,

  `parent\_unit\_id` INT NULL,

  `leader\_id` INT NOT NULL,

  PRIMARY KEY (`unit\_id`),

  INDEX `fk\_unit\_type\_idx` (`unit\_type\_id` ASC) INVISIBLE,

  INDEX `fk\_unit\_unit1\_idx` (`parent\_unit\_id` ASC) VISIBLE,

  INDEX `fk\_unit\_soldier1\_idx` (`leader\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_unit\_type\_id`

    FOREIGN KEY (`unit\_type\_id`)

    REFERENCES `military\_district\_2`.`unit\_type` (`unit\_type\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_unit\_unit1`

    FOREIGN KEY (`parent\_unit\_id`)

    REFERENCES `military\_district\_2`.`unit` (`unit\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_unit\_soldier1`

    FOREIGN KEY (`leader\_id`)

    REFERENCES `military\_district\_2`.`soldier` (`soldier\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`location\_type`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`location\_type` (

  `location\_type\_id` INT NOT NULL,

  `name` VARCHAR(45) NOT NULL,

  PRIMARY KEY (`location\_type\_id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`unit\_location`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`unit\_location` (

  `unit\_id` INT NOT NULL,

  `location\_type\_id` INT NOT NULL,

  `location\_name` VARCHAR(45) NOT NULL,

  INDEX `fk\_location\_type\_id` (`location\_type\_id` ASC) VISIBLE,

  INDEX `fk\_location\_id` (`unit\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_location\_location\_type`

    FOREIGN KEY (`location\_type\_id`)

    REFERENCES `military\_district\_2`.`location\_type` (`location\_type\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_location\_unit`

    FOREIGN KEY (`unit\_id`)

    REFERENCES `military\_district\_2`.`unit` (`unit\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`occupation`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`occupation` (

  `occupation\_id` INT NOT NULL,

  `name` VARCHAR(45) NOT NULL,

  PRIMARY KEY (`occupation\_id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`soldier\_occupation`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`soldier\_occupation` (

  `soldier\_id` INT NOT NULL,

  `occupation\_id` INT NOT NULL,

  INDEX `fk\_soldier\_occupation\_occupation1\_idx` (`occupation\_id` ASC) VISIBLE,

  INDEX `fk\_soldier\_occupation\_soldier1\_idx` (`soldier\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_soldier\_occupation\_occupation1`

    FOREIGN KEY (`occupation\_id`)

    REFERENCES `military\_district\_2`.`occupation` (`occupation\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_soldier\_occupation\_soldier1`

    FOREIGN KEY (`soldier\_id`)

    REFERENCES `military\_district\_2`.`soldier` (`soldier\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`vehicle\_category`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`vehicle\_category` (

  `vehicle\_category\_id` INT NOT NULL,

  `unit\_id` INT NOT NULL,

  `name` VARCHAR(45) NOT NULL,

  PRIMARY KEY (`vehicle\_category\_id`),

  INDEX `fk\_vehicle\_category\_unit1\_idx` (`unit\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_vehicle\_category\_unit1`

    FOREIGN KEY (`unit\_id`)

    REFERENCES `military\_district\_2`.`unit` (`unit\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`vehicle`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`vehicle` (

  `vehicle\_id` INT NOT NULL,

  `vehicle\_category\_id` INT NOT NULL,

  `name` VARCHAR(45) NOT NULL,

  `amount` INT NOT NULL,

  PRIMARY KEY (`vehicle\_id`),

  INDEX `fk\_vehicle\_vehicle\_category1\_idx` (`vehicle\_category\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_vehicle\_vehicle\_category1`

    FOREIGN KEY (`vehicle\_category\_id`)

    REFERENCES `military\_district\_2`.`vehicle\_category` (`vehicle\_category\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`building`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`building` (

  `building\_id` INT NOT NULL,

  `unit\_id` INT NOT NULL,

  `number\_of\_living\_subunits` INT NOT NULL,

  PRIMARY KEY (`building\_id`),

  INDEX `fk\_building\_unit1\_idx` (`unit\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_building\_unit1`

    FOREIGN KEY (`unit\_id`)

    REFERENCES `military\_district\_2`.`unit` (`unit\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`weapon\_category`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`weapon\_category` (

  `weapon\_category\_id` INT NOT NULL,

  `unit\_id` INT NOT NULL,

  `name` VARCHAR(45) NOT NULL,

  PRIMARY KEY (`weapon\_category\_id`),

  INDEX `fk\_weapon\_category\_unit1\_idx` (`unit\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_weapon\_category\_unit1`

    FOREIGN KEY (`unit\_id`)

    REFERENCES `military\_district\_2`.`unit` (`unit\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `military\_district\_2`.`weapon`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `military\_district\_2`.`weapon` (

  `weapon\_id` INT NOT NULL,

  `weapon\_category\_id` INT NOT NULL,

  `name` VARCHAR(45) NOT NULL,

  `amount` INT NOT NULL,

  PRIMARY KEY (`weapon\_id`),

  INDEX `fk\_weapon\_weapon\_category1\_idx` (`weapon\_category\_id` ASC) VISIBLE,

  CONSTRAINT `fk\_weapon\_weapon\_category1`

    FOREIGN KEY (`weapon\_category\_id`)

    REFERENCES `military\_district\_2`.`weapon\_category` (`weapon\_category\_id`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

<!-- Default unchecked -->

<div class="custom-control custom-checkbox">

    <input type="checkbox" class="custom-control-input" id="defaultUnchecked">

    <label class="custom-control-label" for="defaultUnchecked">Default unchecked</label>

</div>

-- ПРОЦЕДУРЫ –

CREATE

    DEFINER = root@localhost FUNCTION military\_district\_2.GetNumberOfSubunits(unitId INT) RETURNS INT DETERMINISTIC

BEGIN

    DECLARE x INT;

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT COUNT(\*) - 1

    INTO x

    FROM units;

    RETURN x;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Buildings(IN unitId INT)

BEGIN

    SELECT \* FROM building WHERE unit\_id = unitId;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Buildings\_LivingMoreThan(IN unitId INT, IN numberOfLivingSubunits INT)

BEGIN

    SELECT \* FROM building WHERE unit\_id = unitId AND number\_of\_living\_subunits > numberOfLivingSubunits;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Buildings\_NonLiving(IN unitId INT)

BEGIN

    SELECT \* FROM building WHERE unit\_id = unitId AND number\_of\_living\_subunits = 0;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_ChainOfSoldierLeaders(IN id INT)

BEGIN

    WITH RECURSIVE leaders AS (

        SELECT soldier\_id, name, leader\_id

        FROM soldier

        WHERE soldier\_id = id

        UNION ALL

        SELECT s.soldier\_id, s.name, s.leader\_id

        FROM soldier s

                 INNER JOIN leaders ON leaders.leader\_id = s.soldier\_id

    )

    SELECT \*

    FROM leaders;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_ChainOfSubunitsAndLeaders(IN id INT)

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, leader\_id, 1 level

                           FROM unit

                           WHERE unit\_id = id

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, u.leader\_id, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                           UNION ALL

                           SELECT su.subunit\_id, su.parent\_unit\_id, su.name, su.leader\_id, level + 1

                           FROM subunit su

                                    INNER JOIN units ON units.unit\_id = su.parent\_unit\_id

                       )

    SELECT units.unit\_id unit\_id, units.name unit\_name, soldier.name leader\_name, level

    FROM units

             LEFT JOIN soldier ON soldier.soldier\_id = units.leader\_id;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Locations(IN unitId INT)

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT units.unit\_id, units.name, ul.location\_name, level

    FROM units

             JOIN unit\_location ul ON units.unit\_id = ul.unit\_id;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Occupations\_WithMoreSpecialistsThan(IN unitId INT, IN numberOfSpecialists INT)

BEGIN

    SELECT occupation.name, COUNT(\*) specialists

    FROM soldier

             JOIN unit ON soldier.serving\_unit\_id = unit.unit\_id

             JOIN soldier\_occupation ON soldier.soldier\_id = soldier\_occupation.soldier\_id

             JOIN occupation ON soldier\_occupation.occupation\_id = occupation.occupation\_id

    WHERE unit.unit\_id = unitId

    GROUP BY occupation.occupation\_id

    HAVING specialists > numberOfSpecialists;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Occupations\_WithNoSpecialists(IN unitId INT)

BEGIN

    SELECT name, 0 specialists

    FROM occupation

    WHERE occupation\_id NOT IN

          (

              SELECT occupation.occupation\_id specialists

              FROM soldier

                       JOIN unit ON soldier.serving\_unit\_id = unit.unit\_id

                       JOIN soldier\_occupation ON soldier.soldier\_id = soldier\_occupation.soldier\_id

                       JOIN occupation ON soldier\_occupation.occupation\_id = occupation.occupation\_id

              WHERE unit.unit\_id = unitId

              GROUP BY occupation.occupation\_id

          );

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_SoldiersSubunit\_WithOccupation(IN subunitId INT, IN occupationId INT)

BEGIN

    SELECT soldier.soldier\_id 'soldier\_id', soldier.name soldier\_name, serving\_unit\_id, occupation\_id

    FROM soldier

             JOIN subunit s ON soldier.serving\_subunit\_id = s.subunit\_id

             JOIN soldier\_occupation so ON soldier.soldier\_id = so.soldier\_id

    WHERE serving\_subunit\_id = subunitId

      AND occupation\_id = occupationId;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Soldiers\_WithGroupRank(IN unitId INT, IN rankGroupName VARCHAR(45))

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, leader\_id, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, u.leader\_id, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT soldier.name soldier\_name, military\_rank.name rank\_name, units.name unit\_name

    FROM units

             JOIN soldier ON soldier.serving\_unit\_id = units.unit\_id

             JOIN military\_rank ON soldier.military\_rank\_id = military\_rank.military\_rank\_id

    WHERE military\_rank.rank\_group = rankGroupName;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Soldiers\_WithOccupation(IN unitId INT, IN occupationId INT)

BEGIN

    SELECT soldier.soldier\_id soldier\_id, soldier.name soldier\_name, serving\_unit\_id, occupation\_id

    FROM soldier

             JOIN unit u ON soldier.serving\_unit\_id = u.unit\_id

             JOIN soldier\_occupation so ON soldier.soldier\_id = so.soldier\_id

    WHERE serving\_unit\_id = unitId

      AND occupation\_id = occupationId;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Soldiers\_WithRank(IN unitId INT, IN rankName VARCHAR(45))

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, leader\_id, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, u.leader\_id, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT soldier.name soldier\_name, military\_rank.name rank\_name, units.name unit\_name

    FROM units

             JOIN soldier ON soldier.serving\_unit\_id = units.unit\_id

             JOIN military\_rank ON soldier.military\_rank\_id = military\_rank.military\_rank\_id

    WHERE military\_rank.name = rankName;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Units\_WithLeastSubunits(IN unitTypeId INT)

BEGIN

    WITH units AS (

        SELECT unit\_id, name, GetNumberOfSubunits(unit\_id) subunits

        FROM unit

        WHERE unit\_type\_id = unitTypeId

    )

    SELECT unit\_id, name, subunits

    FROM units

             INNER JOIN (SELECT MIN(subunits) min\_number FROM units) s ON s.min\_number = units.subunits;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Units\_WithMoreVehiclesThan(IN vehicleName VARCHAR(30), IN vehicleAmount INT)

BEGIN

    SELECT unit.unit\_id, unit.name unit\_name, vehicle.name, vehicle.amount

    FROM unit

             JOIN vehicle\_category vc ON unit.unit\_id = vc.unit\_id

             JOIN vehicle ON vc.vehicle\_category\_id = vehicle.vehicle\_category\_id

    WHERE vehicle.name = vehicleName

      AND vehicle.amount > vehicleAmount;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Units\_WithMoreWeaponsThan(IN weaponName VARCHAR(30), IN weaponAmount INT)

BEGIN

    SELECT unit.unit\_id, weapon.name, weapon.amount

    FROM unit

             JOIN weapon\_category wc ON unit.unit\_id = wc.unit\_id

             JOIN weapon ON wc.weapon\_category\_id = weapon.weapon\_category\_id

    WHERE weapon.name = weaponName

      AND weapon.amount > weaponAmount;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Units\_WithMostSubunits(IN unitTypeId INT)

BEGIN

    WITH units AS (

        SELECT unit\_id, name, GetNumberOfSubunits(unit\_id) subunits

        FROM unit

        WHERE unit\_type\_id = unitTypeId

    )

    SELECT unit\_id, name, subunits

    FROM units

             INNER JOIN (SELECT MAX(subunits) max\_number FROM units) s ON s.max\_number = units.subunits;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Units\_WithNoVehicle(IN vehicleName VARCHAR(45))

BEGIN

    SELECT unit.unit\_id, unit.name unit\_name, vehicle.name, vehicle.amount

    FROM unit

             JOIN vehicle\_category vc ON unit.unit\_id = vc.unit\_id

             JOIN vehicle ON vc.vehicle\_category\_id = vehicle.vehicle\_category\_id

    WHERE vehicle.name = vehicleName

      AND vehicle.amount = 0;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Units\_WithNoWeapon(IN weaponName VARCHAR(45))

BEGIN

    SELECT unit.unit\_id, weapon.name, weapon.amount

    FROM unit

             JOIN weapon\_category wc ON unit.unit\_id = wc.unit\_id

             JOIN weapon ON wc.weapon\_category\_id = weapon.weapon\_category\_id

    WHERE weapon.name = weaponName

      AND weapon.amount = 0;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Vehicle\_WithName(IN unitId INT, IN vehicleName VARCHAR(45))

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT category\_name, v\_name.vehicleName vehicle\_name, IF(sub.amount IS NULL, 0, sub.amount) amount

    FROM (SELECT vehicleName) v\_name

             LEFT JOIN

         (

             SELECT vc.name category\_name, v.name vehicle\_name, SUM(v.amount) amount

             FROM units u

                      JOIN vehicle\_category vc ON u.unit\_id = vc.unit\_id

                      JOIN vehicle v ON vc.vehicle\_category\_id = v.vehicle\_category\_id

             WHERE v.name = vehicleName

             GROUP BY v.name

         ) sub

         ON sub.vehicle\_name = v\_name.vehicleName;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Vehicles(IN unitId INT)

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT vc.name category\_name, v.name vehicle\_name, SUM(v.amount) amount

    FROM units u

             JOIN vehicle\_category vc ON u.unit\_id = vc.unit\_id

             JOIN vehicle v ON vc.vehicle\_category\_id = v.vehicle\_category\_id

    GROUP BY v.name;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Vehicles\_WithCategory(IN unitId INT, IN categoryName VARCHAR(45))

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT c\_name.categoryName category\_name, vehicle\_name, IF(sub.amount IS NULL, 0, sub.amount) amount

    FROM (SELECT categoryName) c\_name

             LEFT JOIN

         (

             SELECT vc.name category\_name, v.name vehicle\_name, SUM(v.amount) amount

             FROM units u

                      JOIN vehicle\_category vc ON u.unit\_id = vc.unit\_id

                      JOIN vehicle v ON vc.vehicle\_category\_id = v.vehicle\_category\_id

             WHERE vc.name = categoryName

             GROUP BY v.name

         ) sub

         ON c\_name.categoryName = sub.category\_name;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Weapon\_WithName(IN unitId INT, IN weaponName VARCHAR(45))

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT wc.name category\_name, w.name weapon\_name, SUM(w.amount) amount

    FROM units

             JOIN weapon\_category wc ON wc.unit\_id = units.unit\_id

             JOIN weapon w ON wc.weapon\_category\_id = w.weapon\_category\_id

    WHERE w.name = weaponName

    GROUP BY w.name;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Weapons(IN unitId INT)

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT wc.name category\_name, w.name weapon\_name, SUM(w.amount) amount

    FROM units

             JOIN weapon\_category wc ON wc.unit\_id = units.unit\_id

             JOIN weapon w ON wc.weapon\_category\_id = w.weapon\_category\_id

    GROUP BY w.name;

END;

CREATE

    DEFINER = root@localhost PROCEDURE military\_district\_2.Show\_Weapons\_WithCategory(IN unitId INT, IN categoryName VARCHAR(45))

BEGIN

    WITH RECURSIVE units AS

                       (

                           SELECT unit\_id, parent\_unit\_id, name, 1 level

                           FROM unit

                           WHERE unit\_id = unitId

                           UNION ALL

                           SELECT u.unit\_id, u.parent\_unit\_id, u.name, level + 1

                           FROM unit u

                                    INNER JOIN units ON units.unit\_id = u.parent\_unit\_id

                       )

    SELECT wc.name category\_name, w.name weapon\_name, SUM(w.amount) amount

    FROM units

             JOIN weapon\_category wc ON wc.unit\_id = units.unit\_id

             JOIN weapon w ON wc.weapon\_category\_id = w.weapon\_category\_id

    WHERE wc.name = categoryName

    GROUP BY w.name;

END;

**ПРИЛОЖЕНИЕ C – код на языке C#**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class Building

    {

        [Display(Name = "Номер здания")]

        public int BuildingId { get; set; }

        [Display(Name = "Номер в/ч")]

        public int UnitId { get; set; }

        [Display(Name = "Кол-во дислоцированных подразделений")]

        public int NumberOfLivingSubunits { get; set; }

        [Display(Name = "Войсковая часть")]

        public virtual Unit Unit { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class LocationType

    {

        [Display(Name =  "Номер")]

        public int LocationTypeId { get; set; }

        [Display(Name = "Название")]

        public string Name { get; set; }

    }

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata;

using MilitaryDistrictAPI.Models.routines;

using MilitaryDistrictAPI.Models;

using MilitaryDistrictAPI.Models.scaffolded;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class MilitaryDistrict2Context : DbContext

    {

        public MilitaryDistrict2Context()

        {

        }

        public MilitaryDistrict2Context(DbContextOptions<MilitaryDistrict2Context> options)

            : base(options)

        {

        }

        public virtual DbSet<Building> Buildings { get; set; }

        public virtual DbSet<LocationType> LocationTypes { get; set; }

        public virtual DbSet<MilitaryRank> MilitaryRanks { get; set; }

        public virtual DbSet<Occupation> Occupations { get; set; }

        public virtual DbSet<Soldier> Soldiers { get; set; }

        // public virtual DbSet<SoldierOccupation> SoldierOccupations { get; set; }

        public virtual DbSet<SoldierRankInfo> SoldierRankInfos { get; set; }

        public virtual DbSet<Subunit> Subunits { get; set; }

        public virtual DbSet<SubunitType> SubunitTypes { get; set; }

        public virtual DbSet<Unit> Units { get; set; }

        public virtual DbSet<UnitLocation> UnitLocations { get; set; }

        public virtual DbSet<UnitType> UnitTypes { get; set; }

        public virtual DbSet<Vehicle> Vehicles { get; set; }

        public virtual DbSet<VehicleCategory> VehicleCategories { get; set; }

        public virtual DbSet<Weapon> Weapons { get; set; }

        public virtual DbSet<WeaponCategory> WeaponCategories { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

        {

            if (!optionsBuilder.IsConfigured)

            {

                optionsBuilder.UseMySQL("server=localhost;uid=root;pwd=1234;database=military\_district\_2");

            }

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            ConfigureRoutines(modelBuilder);

            modelBuilder.Entity<Building>(entity =>

            {

                entity.ToTable("building");

                entity.HasIndex(e => e.UnitId, "fk\_building\_unit1\_idx");

                entity.Property(e => e.BuildingId).HasColumnName("building\_id");

                entity.Property(e => e.NumberOfLivingSubunits).HasColumnName("number\_of\_living\_subunits");

                entity.Property(e => e.UnitId).HasColumnName("unit\_id");

                entity.HasOne(d => d.Unit)

                    .WithMany(p => p.Buildings)

                    .HasForeignKey(d => d.UnitId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_building\_unit1");

            });

            modelBuilder.Entity<LocationType>(entity =>

            {

                entity.ToTable("location\_type");

                entity.Property(e => e.LocationTypeId).HasColumnName("location\_type\_id");

                entity.Property(e => e.Name)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("name");

            });

            modelBuilder.Entity<MilitaryRank>(entity =>

            {

                entity.ToTable("military\_rank");

                entity.Property(e => e.MilitaryRankId).HasColumnName("military\_rank\_id");

                entity.Property(e => e.CommandHigherUnit).HasColumnName("command\_higher\_unit");

                entity.Property(e => e.Name)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("name");

                entity.Property(e => e.RankGroup)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("rank\_group");

            });

            modelBuilder.Entity<Occupation>(entity =>

            {

                entity.ToTable("occupation");

                entity.Property(e => e.OccupationId).HasColumnName("occupation\_id");

                entity.Property(e => e.Name)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("name");

            });

            modelBuilder.Entity<Soldier>(entity =>

            {

                entity.ToTable("soldier");

                entity.HasIndex(e => e.MilitaryRankId, "fk\_soldier\_military\_rank1\_idx");

                entity.HasIndex(e => e.LeaderId, "fk\_soldier\_soldier1\_idx");

                entity.HasIndex(e => e.RankInfoId, "fk\_soldier\_soldier\_rank\_info1\_idx");

                entity.HasIndex(e => e.ServingSubunitId, "fk\_soldier\_subunit1\_idx");

                entity.HasIndex(e => e.ServingUnitId, "fk\_soldier\_unit1\_idx");

                entity.Property(e => e.SoldierId).HasColumnName("soldier\_id").ValueGeneratedOnAdd();

                entity.Property(e => e.LeaderId).HasColumnName("leader\_id");

                entity.Property(e => e.MilitaryRankId).HasColumnName("military\_rank\_id");

                entity.Property(e => e.Name)

                    .HasMaxLength(45)

                    .HasColumnName("name");

                entity.Property(e => e.RankInfoId).HasColumnName("rank\_info\_id");

                entity.Property(e => e.ServingSubunitId).HasColumnName("serving\_subunit\_id");

                entity.Property(e => e.ServingUnitId).HasColumnName("serving\_unit\_id");

                entity.HasMany(p => p.Occupations)

                    .WithMany(p => p.Soldiers)

                    .UsingEntity<Dictionary<string, object>>(

                        "soldier\_occupation",

                        j => j

                            .HasOne<Occupation>()

                            .WithMany()

                            .HasForeignKey("occupation\_id")

                            .OnDelete(DeleteBehavior.Cascade),

                        j => j

                            .HasOne<Soldier>()

                            .WithMany()

                            .HasForeignKey("soldier\_id")

                            .OnDelete(DeleteBehavior.ClientCascade));

                entity.HasOne(d => d.Leader)

                    .WithMany(p => p.InverseLeader)

                    .HasForeignKey(d => d.LeaderId)

                    .HasConstraintName("fk\_soldier\_soldier1");

                entity.HasOne(d => d.MilitaryRank)

                    .WithMany(p => p.Soldiers)

                    .HasForeignKey(d => d.MilitaryRankId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_soldier\_military\_rank1");

                entity.HasOne(d => d.RankInfo)

                    .WithMany(p => p.Soldiers)

                    .HasForeignKey(d => d.RankInfoId)

                    .HasConstraintName("fk\_soldier\_soldier\_rank\_info1");

                entity.HasOne(d => d.ServingSubunit)

                    .WithMany(p => p.Soldiers)

                    .HasForeignKey(d => d.ServingSubunitId)

                    .HasConstraintName("fk\_soldier\_subunit1");

                entity.HasOne(d => d.ServingUnit)

                    .WithMany(p => p.Soldiers)

                    .HasForeignKey(d => d.ServingUnitId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_soldier\_unit1");

            });

            modelBuilder.Entity<SoldierRankInfo>(entity =>

            {

                entity.HasKey(e => e.RecordId)

                    .HasName("PRIMARY");

                entity.ToTable("soldier\_rank\_info");

                entity.Property(e => e.RecordId).HasColumnName("record\_id");

                entity.Property(e => e.DateOfGraduation)

                    .HasColumnType("date")

                    .HasColumnName("date\_of\_graduation");

                entity.Property(e => e.DateOfPromotion)

                    .HasColumnType("date")

                    .HasColumnName("date\_of\_promotion");

            });

            modelBuilder.Entity<Subunit>(entity =>

            {

                entity.ToTable("subunit");

                entity.HasIndex(e => e.LeaderId, "fk\_subunit\_soldier1\_idx");

                entity.HasIndex(e => e.SubunitTypeId, "fk\_subunit\_subunit\_type1\_idx");

                entity.HasIndex(e => e.ParentUnitId, "fk\_subunit\_unit1\_idx");

                entity.Property(e => e.SubunitId).HasColumnName("subunit\_id").ValueGeneratedOnAdd();

                entity.Property(e => e.LeaderId).HasColumnName("leader\_id");

                entity.Property(e => e.Name)

                    .HasMaxLength(45)

                    .HasColumnName("name");

                entity.Property(e => e.ParentUnitId).HasColumnName("parent\_unit\_id");

                entity.Property(e => e.SubunitTypeId).HasColumnName("subunit\_type\_id");

                entity.HasOne(d => d.Leader)

                    .WithMany(p => p.Subunits)

                    .HasForeignKey(d => d.LeaderId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_subunit\_soldier1");

                entity.HasOne(d => d.ParentUnit)

                    .WithMany(p => p.Subunits)

                    .HasForeignKey(d => d.ParentUnitId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_subunit\_unit1");

                entity.HasOne(d => d.SubunitType)

                    .WithMany(p => p.Subunits)

                    .HasForeignKey(d => d.SubunitTypeId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_subunit\_subunit\_type1");

            });

            modelBuilder.Entity<SubunitType>(entity =>

            {

                entity.ToTable("subunit\_type");

                entity.Property(e => e.SubunitTypeId).HasColumnName("subunit\_type\_id");

                entity.Property(e => e.Name)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("name");

            });

            modelBuilder.Entity<Unit>(entity =>

            {

                entity.ToTable("unit");

                entity.HasIndex(e => e.LeaderId, "fk\_unit\_soldier1\_idx");

                entity.HasIndex(e => e.UnitTypeId, "fk\_unit\_type\_idx");

                entity.HasIndex(e => e.ParentUnitId, "fk\_unit\_unit1\_idx");

                entity.Property(e => e.UnitId).HasColumnName("unit\_id").ValueGeneratedOnAdd();

                entity.Property(e => e.LeaderId).HasColumnName("leader\_id");

                entity.Property(e => e.Name)

                    .HasMaxLength(500)

                    .HasColumnName("name");

                entity.Property(e => e.ParentUnitId).HasColumnName("parent\_unit\_id");

                entity.Property(e => e.UnitNumber)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("unit\_number");

                entity.Property(e => e.UnitTypeId).HasColumnName("unit\_type\_id");

                entity.HasOne(d => d.Leader)

                    .WithMany(p => p.Units)

                    .HasForeignKey(d => d.LeaderId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_unit\_soldier1");

                entity.HasOne(d => d.ParentUnit)

                    .WithMany(p => p.InverseParentUnit)

                    .HasForeignKey(d => d.ParentUnitId)

                    .HasConstraintName("fk\_unit\_unit1");

                entity.HasOne(d => d.UnitType)

                    .WithMany(p => p.Units)

                    .HasForeignKey(d => d.UnitTypeId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_unit\_type\_id");

            });

            modelBuilder.Entity<UnitLocation>(entity =>

            {

                entity.HasNoKey();

                entity.ToTable("unit\_location");

                entity.HasIndex(e => e.UnitId, "fk\_location\_id");

                entity.HasIndex(e => e.LocationTypeId, "fk\_location\_type\_id");

                entity.Property(e => e.LocationName)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("location\_name");

                entity.Property(e => e.LocationTypeId).HasColumnName("location\_type\_id");

                entity.Property(e => e.UnitId).HasColumnName("unit\_id");

                entity.HasOne(d => d.LocationType)

                    .WithMany()

                    .HasForeignKey(d => d.LocationTypeId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_location\_location\_type");

                entity.HasOne(d => d.Unit)

                    .WithMany()

                    .HasForeignKey(d => d.UnitId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_location\_unit");

            });

            modelBuilder.Entity<UnitType>(entity =>

            {

                entity.ToTable("unit\_type");

                entity.Property(e => e.UnitTypeId).HasColumnName("unit\_type\_id");

                entity.Property(e => e.Name)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("name");

            });

            modelBuilder.Entity<Vehicle>(entity =>

            {

                entity.ToTable("vehicle");

                entity.HasIndex(e => e.VehicleCategoryId, "fk\_vehicle\_vehicle\_category1\_idx");

                entity.Property(e => e.VehicleId).HasColumnName("vehicle\_id");

                entity.Property(e => e.Amount).HasColumnName("amount");

                entity.Property(e => e.Name)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("name");

                entity.Property(e => e.VehicleCategoryId).HasColumnName("vehicle\_category\_id");

                entity.HasOne(d => d.VehicleCategory)

                    .WithMany(p => p.Vehicles)

                    .HasForeignKey(d => d.VehicleCategoryId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_vehicle\_vehicle\_category1");

            });

            modelBuilder.Entity<VehicleCategory>(entity =>

            {

                entity.ToTable("vehicle\_category");

                entity.HasIndex(e => e.UnitId, "fk\_vehicle\_category\_unit1\_idx");

                entity.Property(e => e.VehicleCategoryId).HasColumnName("vehicle\_category\_id");

                entity.Property(e => e.Name)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("name");

                entity.Property(e => e.UnitId).HasColumnName("unit\_id");

                entity.HasOne(d => d.Unit)

                    .WithMany(p => p.VehicleCategories)

                    .HasForeignKey(d => d.UnitId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_vehicle\_category\_unit1");

            });

            modelBuilder.Entity<Weapon>(entity =>

            {

                entity.ToTable("weapon");

                entity.HasIndex(e => e.WeaponCategoryId, "fk\_weapon\_weapon\_category1\_idx");

                entity.Property(e => e.WeaponId).HasColumnName("weapon\_id");

                entity.Property(e => e.Amount).HasColumnName("amount");

                entity.Property(e => e.Name)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("name");

                entity.Property(e => e.WeaponCategoryId).HasColumnName("weapon\_category\_id");

                entity.HasOne(d => d.WeaponCategory)

                    .WithMany(p => p.Weapons)

                    .HasForeignKey(d => d.WeaponCategoryId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_weapon\_weapon\_category1");

            });

            modelBuilder.Entity<WeaponCategory>(entity =>

            {

                entity.ToTable("weapon\_category");

                entity.HasIndex(e => e.UnitId, "fk\_weapon\_category\_unit1\_idx");

                entity.Property(e => e.WeaponCategoryId).HasColumnName("weapon\_category\_id");

                entity.Property(e => e.Name)

                    .IsRequired()

                    .HasMaxLength(45)

                    .HasColumnName("name");

                entity.Property(e => e.UnitId).HasColumnName("unit\_id");

                entity.HasOne(d => d.Unit)

                    .WithMany(p => p.WeaponCategories)

                    .HasForeignKey(d => d.UnitId)

                    .OnDelete(DeleteBehavior.ClientSetNull)

                    .HasConstraintName("fk\_weapon\_category\_unit1");

            });

            OnModelCreatingPartial(modelBuilder);

        }

        partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

    }

}

using System;

using System.Collections;

using System.Linq;

using Microsoft.EntityFrameworkCore;

using MilitaryDistrictAPI.Models.routines;

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class MilitaryDistrict2Context

    {

        public DbSet<Routine1> ResultRoutine1 { get; set; }

        public DbSet<Routine2> ResultRoutine2 { get; set; }

        public DbSet<Routine3> ResultRoutine3 { get; set; }

        public DbSet<Routine4> ResultRoutine4 { get; set; }

        public DbSet<Routine5> ResultRoutine5 { get; set; }

        public DbSet<Routine6> ResultRoutine6 { get; set; }

        public DbSet<Routine7> ResultRoutine7 { get; set; }

        public DbSet<Routine8> ResultRoutine8 { get; set; }

        public DbSet<Routine9> ResultRoutine9 { get; set; }

        public DbSet<Routine10> ResultRoutine10 { get; set; }

        public DbSet<Routine11> ResultRoutine11 { get; set; }

        public DbSet<Routine12> ResultRoutine12 { get; set; }

        public DbSet<Routine13> ResultRoutine13 { get; set; }

        protected virtual void ConfigureRoutines(ModelBuilder modelBuilder)

        {

            modelBuilder.Entity<Routine1>().HasNoKey();

            modelBuilder.Entity<Routine1>(entity =>

                {

                    entity.Property(e => e.UnitId).HasColumnName("unit\_id");

                    entity.Property(e => e.UnitName).HasColumnName("unit\_name");

                    entity.Property(e => e.LeaderName).HasColumnName("leader\_name");

                    entity.Property(e => e.Level).HasColumnName("level");

                }

            );

            modelBuilder.Entity<Routine2>().HasNoKey();

            modelBuilder.Entity<Routine3>().HasNoKey();

            modelBuilder.Entity<Routine4>().HasNoKey();

            modelBuilder.Entity<Routine4>(entity =>

                {

                    entity.Property(e => e.SoldierId).HasColumnName("soldier\_id");

                    entity.Property(e => e.Name).HasColumnName("name");

                    entity.Property(e => e.LeaderId).HasColumnName("leader\_id");

                }

            );

            modelBuilder.Entity<Routine5>().HasNoKey();

            modelBuilder.Entity<Routine6>().HasNoKey();

            modelBuilder.Entity<Routine7>().HasNoKey();

            modelBuilder.Entity<Routine8>().HasNoKey();

            modelBuilder.Entity<Routine9>().HasNoKey();

            modelBuilder.Entity<Routine10>().HasNoKey();

            modelBuilder.Entity<Routine11>().HasNoKey();

            modelBuilder.Entity<Routine12>().HasNoKey();

            modelBuilder.Entity<Routine13>().HasNoKey();

        }

        public IList ExecuteQuery(string routineName, params object[] parameters)

        {

            return routineName switch

            {

                Routine1.NAME => ResultRoutine1.FromSqlRaw(Routine1.SQL, parameters).ToList(),

                Routine2.NAME => ResultRoutine2.FromSqlRaw(Routine2.SQL, parameters).ToList(),

                Routine3.NAME => ResultRoutine3.FromSqlRaw(Routine3.SQL, parameters).ToList(),

                Routine4.NAME => ResultRoutine4.FromSqlRaw(Routine4.SQL, parameters).ToList(),

                Routine5.NAME => ResultRoutine5.FromSqlRaw(Routine5.SQL, parameters).ToList(),

                Routine6.NAME\_1 => ResultRoutine6.FromSqlRaw(Routine6.SQL\_1, parameters).ToList(),

                Routine6.NAME\_2 => ResultRoutine6.FromSqlRaw(Routine6.SQL\_2, parameters).ToList(),

                Routine6.NAME\_3 => ResultRoutine6.FromSqlRaw(Routine6.SQL\_3, parameters).ToList(),

                Routine7.NAME\_1 => ResultRoutine7.FromSqlRaw(Routine7.SQL\_1, parameters).ToList(),

                Routine7.NAME\_2 => ResultRoutine7.FromSqlRaw(Routine7.SQL\_2, parameters).ToList(),

                Routine7.NAME\_3 => ResultRoutine7.FromSqlRaw(Routine7.SQL\_3, parameters).ToList(),

                Routine8.NAME\_1 => ResultRoutine8.FromSqlRaw(Routine8.SQL\_1, parameters).ToList(),

                Routine8.NAME\_2 => ResultRoutine8.FromSqlRaw(Routine8.SQL\_2, parameters).ToList(),

                Routine9.NAME\_1 => ResultRoutine9.FromSqlRaw(Routine9.SQL\_1, parameters).ToList(),

                Routine9.NAME\_2 => ResultRoutine9.FromSqlRaw(Routine9.SQL\_2, parameters).ToList(),

                Routine9.NAME\_3 => ResultRoutine9.FromSqlRaw(Routine9.SQL\_3, parameters).ToList(),

                Routine10.NAME\_1 => ResultRoutine10.FromSqlRaw(Routine10.SQL\_1, parameters).ToList(),

                Routine10.NAME\_2 => ResultRoutine10.FromSqlRaw(Routine10.SQL\_2, parameters).ToList(),

                Routine11.NAME\_1 => ResultRoutine11.FromSqlRaw(Routine11.SQL\_1, parameters).ToList(),

                Routine11.NAME\_2 => ResultRoutine11.FromSqlRaw(Routine11.SQL\_2, parameters).ToList(),

                Routine12.NAME\_1 => ResultRoutine12.FromSqlRaw(Routine12.SQL\_1, parameters).ToList(),

                Routine12.NAME\_2 => ResultRoutine12.FromSqlRaw(Routine12.SQL\_2, parameters).ToList(),

                Routine13.NAME\_1 => ResultRoutine13.FromSqlRaw(Routine13.SQL\_1, parameters).ToList(),

                Routine13.NAME\_2 => ResultRoutine13.FromSqlRaw(Routine13.SQL\_2, parameters).ToList(),

                \_ => throw new ArgumentOutOfRangeException(nameof(routineName), routineName, null)

            };

        }

    }

}using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class MilitaryRank

    {

        public MilitaryRank()

        {

            Soldiers = new HashSet<Soldier>();

        }

        [Display(Name = "Номер")]

        public int MilitaryRankId { get; set; }

        [Display(Name = "Название")]

        public string Name { get; set; }

        [Display(Name = "Группа званий")]

        public string RankGroup { get; set; }

        [Display(Name = "Может командовать высшими формированиями")]

        public bool CommandHigherUnit { get; set; }

        public virtual ICollection<Soldier> Soldiers { get; set; }

    }

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class Occupation

    {

        public Occupation()

        {

            Soldiers = new List<Soldier>();

        }

        [Column("occupation\_id")]

        [Display(Name = "Номер")]

        public int OccupationId { get; set; }

        [Display(Name = "Название")]

        public string Name { get; set; }

        public ICollection<Soldier> Soldiers { get; set; }

    }

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Runtime.CompilerServices;

using Google.Protobuf.WellKnownTypes;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class Soldier

    {

        public Soldier()

        {

            InverseLeader = new HashSet<Soldier>();

            Subunits = new HashSet<Subunit>();

            Units = new HashSet<Unit>();

            Occupations = new List<Occupation>();

        }

        [Display(Name = "№")]

        public int SoldierId { get; set; }

        [Display(Name = "Номер в/ч")]

        public int ServingUnitId { get; set; }

        [Display(Name = "Номер подразделения")]

        public int? ServingSubunitId { get; set; }

        [Display(Name = "Номер звания")]

        public int MilitaryRankId { get; set; }

        [Display(Name = "Имя")]

        [Required(AllowEmptyStrings = false, ErrorMessage = "Имя не может быть пустым")]

        public string Name { get; set; }

        [Display(Name = "Номер записи о звании")]

        public int? RankInfoId { get; set; }

        [Display(Name = "Номер командующего")]

        public int? LeaderId { get; set; }

        [Display(Name = "Командир")]

        public virtual Soldier Leader { get; set; }

        [Display(Name = "Звание")]

        public virtual MilitaryRank MilitaryRank { get; set; }

        [Display(Name = "Запись о звании")]

        public virtual SoldierRankInfo RankInfo { get; set; }

        [Display(Name = "Подразделение")]

        public virtual Subunit ServingSubunit { get; set; }

        [Display(Name = "Войсковая часть")]

        public virtual Unit ServingUnit { get; set; }

        [Display(Name = "InverseLeader")]

        public virtual ICollection<Soldier> InverseLeader { get; set; }

        [Display(Name = "Subunits")]

        public virtual ICollection<Subunit> Subunits { get; set; }

        [Display(Name = "Units")]

        public virtual ICollection<Unit> Units { get; set; }

        [Display(Name = "Специальности")]

        public virtual List<Occupation> Occupations { get; set; }

        [NotMapped]

        public string Occop { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Org.BouncyCastle.Asn1.Pkcs;

// #nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public class SoldierOccupation

    {

        // [Key]

        // public long Id { get; set; }

        public int SoldierId { get; set; }

        public int OccupationId { get; set; }

        public virtual Soldier Soldier { get; set; }

        public virtual Occupation Occupation { get; set; }

    }

}

using System;

using System.Collections.Generic;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class SoldierRankInfo

    {

        public SoldierRankInfo()

        {

            Soldiers = new HashSet<Soldier>();

        }

        public int RecordId { get; set; }

        public DateTime DateOfPromotion { get; set; }

        public DateTime? DateOfGraduation { get; set; }

        public virtual ICollection<Soldier> Soldiers { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class Subunit

    {

        public Subunit()

        {

            Soldiers = new HashSet<Soldier>();

        }

        [Display(Name = "№")]

        public int SubunitId { get; set; }

        [Display(Name = "Командир")]

        public int? LeaderId { get; set; }

        [Display(Name = "Номер типа")]

        public int SubunitTypeId { get; set; }

        [Display(Name = "B составе")]

        public int ParentUnitId { get; set; }

        [Display(Name = "Название")]

        public string Name { get; set; }

        [Display(Name = "Командир")]

        public virtual Soldier Leader { get; set; }

        [Display(Name = "В составе")]

        public virtual Unit ParentUnit { get; set; }

        [Display(Name = "Тип")]

        public virtual SubunitType SubunitType { get; set; }

        public virtual ICollection<Soldier> Soldiers { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.CodeAnalysis.Diagnostics;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class SubunitType

    {

        public SubunitType()

        {

            Subunits = new HashSet<Subunit>();

        }

        [Display(Name = "Номер")]

        public int SubunitTypeId { get; set; }

        [Display(Name = "Название")]

        public string Name { get; set; }

        public virtual ICollection<Subunit> Subunits { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class Unit

    {

        public Unit()

        {

            Buildings = new HashSet<Building>();

            InverseParentUnit = new HashSet<Unit>();

            Soldiers = new HashSet<Soldier>();

            Subunits = new HashSet<Subunit>();

            VehicleCategories = new HashSet<VehicleCategory>();

            WeaponCategories = new HashSet<WeaponCategory>();

        }

        [Display(Name = "№")]

        public int UnitId { get; set; }

        [Display(Name = "Тип в/ч")]

        public int UnitTypeId { get; set; }

        [Display(Name = "Номер в/ч")]

        [Required(AllowEmptyStrings = false, ErrorMessage = "Номер в/ч не может быть пустым")]

        public string UnitNumber { get; set; }

        [Display(Name = "Название")]

        public string Name { get; set; }

        [Display(Name = "В составе")]

        public int? ParentUnitId { get; set; }

        [Display(Name = "Командующий")]

        public int? LeaderId { get; set; }

        [Display(Name = "Командующий")]

        public virtual Soldier Leader { get; set; }

        [Display(Name = "В составе")]

        public virtual Unit ParentUnit { get; set; }

        [Display(Name = "Тип")]

        public virtual UnitType UnitType { get; set; }

        public virtual ICollection<Building> Buildings { get; set; }

        public virtual ICollection<Unit> InverseParentUnit { get; set; }

        public virtual ICollection<Soldier> Soldiers { get; set; }

        public virtual ICollection<Subunit> Subunits { get; set; }

        public virtual ICollection<VehicleCategory> VehicleCategories { get; set; }

        public virtual ICollection<WeaponCategory> WeaponCategories { get; set; }

    }

}

using System;

using System.Collections.Generic;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class UnitLocation

    {

        public int UnitId { get; set; }

        public int LocationTypeId { get; set; }

        public string LocationName { get; set; }

        public virtual LocationType LocationType { get; set; }

        public virtual Unit Unit { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class UnitType

    {

        public UnitType()

        {

            Units = new HashSet<Unit>();

        }

        [Display(Name = "Номер")]

        public int UnitTypeId { get; set; }

        [Display(Name = "Название")]

        public string Name { get; set; }

        public virtual ICollection<Unit> Units { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Configuration;

using Org.BouncyCastle.Crypto.Tls;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class Vehicle

    {

        public int VehicleId { get; set; }

        [Display(Name = "Номер категории")]

        public int VehicleCategoryId { get; set; }

        [Display(Name = "Название")]

        [Required(AllowEmptyStrings = false, ErrorMessage = "Название не может быть пустым")]

        public string Name { get; set; }

        [Display(Name = "Количество")]

        [Required(ErrorMessage = "Укажите количество")]

        [RegularExpression("([1-9][0-9]\*)", ErrorMessage = "Только положительные числа")]

        [Range(typeof(int), "0", "100", ErrorMessage = "Только положительные числа")]

        public int Amount { get; set; }

        [Display(Name = "Категория техники")]

        public virtual VehicleCategory VehicleCategory { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class VehicleCategory

    {

        public VehicleCategory()

        {

            Vehicles = new HashSet<Vehicle>();

        }

        public int VehicleCategoryId { get; set; }

        [Display(Name = "Номер в/ч")]

        public int UnitId { get; set; }

        [Display(Name = "Название")]

        [Required(AllowEmptyStrings = false, ErrorMessage = "Название не может быть пустым")]

        public string Name { get; set; }

        [Display(Name = "Войсковая часть")]

        public virtual Unit Unit { get; set; }

        public virtual ICollection<Vehicle> Vehicles { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class Weapon

    {

        public int WeaponId { get; set; }

        [Display(Name = "Номер категории")]

        public int WeaponCategoryId { get; set; }

        [Display(Name = "Название")]

        [Required(AllowEmptyStrings = false, ErrorMessage = "Название не может быть пустым")]

        public string Name { get; set; }

        [Display(Name = "Количество")]

        [Required(ErrorMessage = "Укажите количество")]

        [Range(typeof(int), "0", "100", ErrorMessage = "Только положительные числа")]

        [RegularExpression("([1-9][0-9]\*)", ErrorMessage = "Только положительные числа")]

        public int Amount { get; set; }

        [Display(Name = "Категория вооружения")]

        public virtual WeaponCategory WeaponCategory { get; set; }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

#nullable disable

namespace MilitaryDistrictAPI.Models.scaffolded

{

    public partial class WeaponCategory

    {

        public WeaponCategory()

        {

            Weapons = new HashSet<Weapon>();

        }

        public int WeaponCategoryId { get; set; }

        [Display(Name = "Номер в/ч")]

        public int UnitId { get; set; }

        [Display(Name = "Название")]

        [Required(AllowEmptyStrings = false, ErrorMessage = "Название не может быть пустым")]

        public string Name { get; set; }

        [Display(Name = "Войсковая часть")]

        public virtual Unit Unit { get; set; }

        public virtual ICollection<Weapon> Weapons { get; set; }

    }

}