Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Базы данных (БД)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**ПРИЛОЖЕНИЕ ПО УПРАВЛЕНИЮ АРСЕНАЛОМ**

БГУИР КП 1-40 01 01 007 ПЗ

Студент гр. 251004 Карась А.С.

Руководитель Кожемяко Е.А.

Минск 2025

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc193647996)

[1 АНАЛИЗ ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ 5](#_Toc193647997)

[1.1 Анализ существующих прототипов 5](#_Toc193647998)

[1.2 Требования к проектируемому программному средству 8](#_Toc193647999)

[1.3 Постановка задачи 11](#_Toc193648000)

[2 АНАЛИЗ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 12](#_Toc193648001)

[2.1 Используемые технологии при разработке 12](#_Toc193648002)

[2.2 Спецификация функциональных требований 13](#_Toc193648003)

[2.3 Ключевые особенности информационной системы 15](#_Toc193648004)

[2.4 Пользователи системы и их роли 15](#_Toc193648005)

[2.5 Параметры поиска 17](#_Toc193648006)

[3 ИНФОЛОГИЧЕСКАЯ МОДЕЛЬ ПРЕДМЕТНОЙ ОБЛАСТИ 19](#_Toc193648007)

[3.1 Модель базы данных 19](#_Toc193648008)

[3.2 Описание таблиц 20](#_Toc193648009)

[4 ПОДРОБНОЕ ОПИСАНИЕ БИЗНЕС-ЛОГИКИ 22](#_Toc193648010)

[4.1 Триггеры 22](#_Toc193648011)

[4.2 Хранимые процедуры 23](#_Toc193648012)

[4.3 Представления 24](#_Toc193648013)

[5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 25](#_Toc193648014)

[ЗАКЛЮЧЕНИЕ 26](#_Toc193648015)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc193648016)

[ПРИЛОЖЕНИЕ А. 28](#_Toc193648017)

# ВВЕДЕНИЕ

В современном мире, где технологии развиваются стремительными темпами, эффективное управление ресурсами и информацией становится ключевым фактором успеха в различных сферах деятельности. Одной из таких областей является управление арсеналом, будь то военные ресурсы, оборудование для предприятий или инструменты для специализированных организаций. В этом контексте создание специализированного приложения для управления арсеналом становится актуальной задачей, требующей использования современных технологий, среди которых центральное место занимают базы данных.

Базы данных представляют собой структурированные системы хранения и обработки информации, которые позволяют эффективно управлять большими объемами данных. В приложении для управления арсеналом базы данных играют ключевую роль, так как они обеспечивают хранение информации о ресурсах, их характеристиках, состоянии, истории использования и других важных параметрах. Это позволяет организациям оперативно получать доступ к необходимым данным, анализировать их и принимать обоснованные решения.

Актуальность использования баз данных в приложениях для управления арсеналом обусловлена несколькими факторами. Во-первых, с увеличением объема ресурсов и сложности их учета ручные методы управления становятся неэффективными. Базы данных позволяют автоматизировать процессы учета, минимизировать ошибки и сократить время на обработку информации. Во-вторых, современные требования к прозрачности и отчетности в управлении ресурсами делают базы данных незаменимым инструментом для отслеживания изменений и генерации отчетов. В-третьих, интеграция баз данных с другими системами, такими как системы планирования или аналитики, позволяет повысить общую эффективность управления арсеналом.

В приложении для управления арсеналом базы данных позволяют отслеживать состояние ресурсов, их распределение, сроки эксплуатации и другие параметры, что способствует оптимизации использования ресурсов и повышению эффективности работы организации.

Управление арсеналом предполагает работу с большими объемами данных, которые должны быть точными и непротиворечивыми. Консистентность данных обеспечивает корректность информации о ресурсах, их состоянии, распределении и истории использования. Необходимо уделить особое внимание механизмам валидации данных, транзакциям и контролю целостности, чтобы минимизировать риски потери или искажения информации.

Также управление арсеналом часто связано с конфиденциальной информацией, особенно в военной или промышленной сферах. Поэтому безопасность данных является критически важным требованием.

Приложение должно эффективно работать с большими объемами данных, обеспечивая быстрый доступ к информации и оперативную обработку запросов. Для этого необходимо использовать оптимизированные структуры базы данных, индексацию и кэширование. Также обязательно уделить внимание масштабируемости системы, чтобы она могла справляться с увеличением объема данных.

Целью данного проекта является разработка приложения для управления арсеналом. Такое приложение позволит не только упростить процессы учета и управления ресурсами, но и предоставит возможности для анализа данных, прогнозирования потребностей и улучшения стратегического планирования. Основное внимание будет уделено обеспечению консистентности данных, так как это ключевой фактор для корректного учета и управления ресурсами. Для этого будут использованы транзакционные механизмы, контроль целостности и валидация данных на всех этапах работы с системой.

# АНАЛИЗ ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ

## 1.1 Анализ существующих прототипов

**1.1.1** SAP ERP

SAP ERP (Enterprise Resource Planning) — это комплексная система управления ресурсами предприятия, которая широко используется в различных отраслях, включая управление арсеналом. Она предоставляет модули для управления финансами, логистикой, запасами, персоналом и другими аспектами бизнеса. SAP ERP позволяет интегрировать данные из разных отделов компании, обеспечивая единую платформу для управления ресурсами. Система поддерживает сложные процессы учета, планирования и анализа, что делает ее подходящей для крупных организаций с множеством хранилищ и сложной логистикой. Программное средство также включает следующие преимущества:

* высокая степень интеграции различных бизнес-процессов;
* поддержка масштабируемости для крупных предприятий;
* широкие возможности для аналитики и отчетности;
* гибкость в настройке под специфические нужды организации;
* надежность и проверенная временем репутация.

Однако у SAP ERP есть и недостатки:

* высокая стоимость внедрения и поддержки;
* сложность настройки и обучения персонала.

Внешний вид программного средства представлен на рисунке 1.1.



Рисунок 1.1 **–** интерфейс программного средства SAP ERP.

**1.1.2** GCSS-Army (Global Combat Support System-Army)

GCSS-Army — это специализированная система управления логистикой и запасами, разработанная для армии США. Она используется для управления военным арсеналом, включая оружие, технику, амуницию и другие ресурсы. Система обеспечивает полный контроль за перемещением, состоянием и использованием ресурсов, а также интегрируется с другими военными системами. GCSS-Army позволяет автоматизировать процессы учета, планирования и отчетности, что критически важно для эффективного управления военными ресурсами. Программное средство также включает следующие преимущества:

* специализированная система для военных нужд;
* высокая степень автоматизации процессов учета и управления;
* интеграция с другими военными системами;
* поддержка сложных логистических операций;
* удобство генерации отчетов и аналитики.

Однако у GCSS-Army есть и недостатки:

* ограниченная применимость вне военного контекста;
* высокие требования к инфраструктуре и обучению персонала.

Внешний вид программного средства представлен на рисунке 1.2.

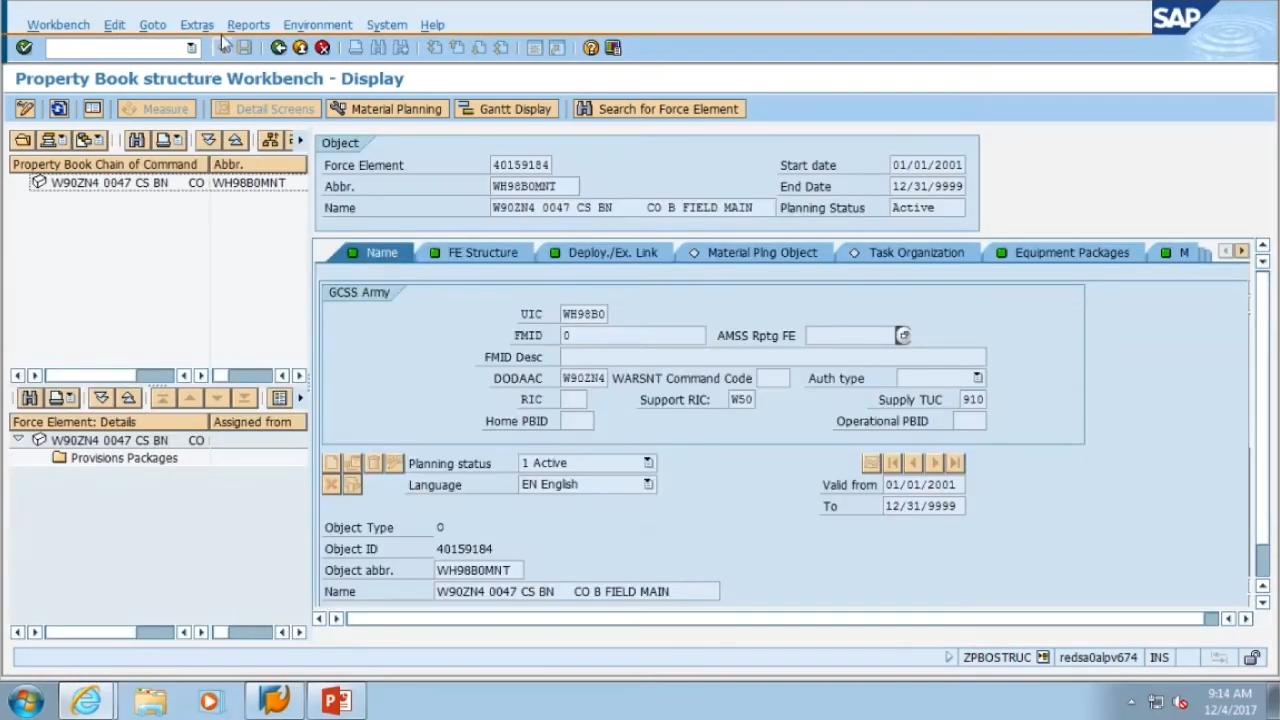


Рисунок 1.2 **–** интерфейс программного средства GCSS-Army.

**1.1.3** IBM Maximo

IBM Maximo — это система управления активами предприятия (EAM), которая используется для управления оборудованием, инвентарем и другими ресурсами. Она подходит для промышленных предприятий, логистических компаний и организаций, которым требуется контроль за состоянием и использованием активов. IBM Maximo предоставляет инструменты для планирования технического обслуживания, учета ресурсов и анализа данных. Система поддерживает мобильные приложения, что позволяет управлять ресурсами в режиме реального времени. Программное средство также включает следующие преимущества:

* широкие возможности для управления техническим обслуживанием;
* поддержка мобильных приложений для удобства работы в полевых условиях;
* гибкость в настройке под нужды предприятия;
* интеграция с другими системами IBM;
* удобство для анализа данных и прогнозирования потребностей.

Однако у IBM Maximo есть и недостатки:

* высокая стоимость лицензий и поддержки;
* требует значительных ресурсов для внедрения и настройки.

Внешний вид программного средства представлен на рисунке 1.3.

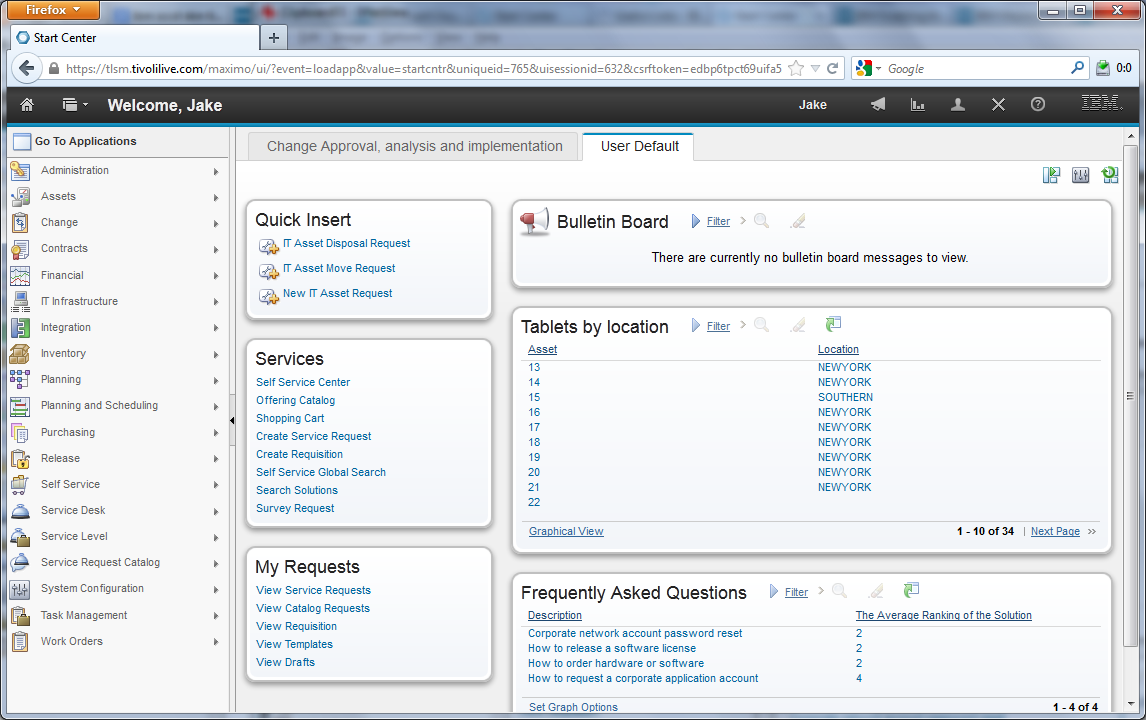


Рисунок 1.3 **–** интерфейс программного средства GCSS-Army.

## 1.2 Требования к проектируемому программному средству

**1.2.1** Назначение разработки

В рамках данной курсовой работы планируется разработать базу данных и приложение для взаимодействия с базой данных, которых будет достаточно для управления арсеналом. Таким образом, для выполнения поставленной задачи необходимо выполнить следующие задачи:

* разработать физическую модель базы данных для реляционной базы данных MySQL;
* разработать приложение с визуальным пользовательским интерфейсом для взаимодействия с физической базой данных;
* произвести тестирование базы данных;
* произвести тестирование приложения с визуальным пользовательским интерфейсом.

**1.2.2** Состав выполняемых функций

* Ведение базы данных ресурсов (наименование, тип, характеристики, состояние, количество, место хранения и т.п.).
* Возможность добавления, редактирования и удаления записей о ресурсах с использованием визуального пользовательского интерфейса.
* Ведение истории использования ресурсов.
* Учет технического состояния и сроков эксплуатации.
* Разграничение прав доступа в зависимости от ролей (гость, администратор, менеджер, пользователь).

**1.2.3** Входные данные

* Наименование, тип, характеристики, количество, место хранения, состояние, срок эксплуатации.
* Логин, пароль, роль, контактная информация.
* Информация о назначении ресурсов подразделениям или сотрудникам.
* История использования, техническое состояние, даты последнего обслуживания.
* Поиск, фильтрация, сортировка данных.

**1.2.4** Выходные данные

* Отображение списка ресурсов, их состояния и распределения.
* Формы для добавления, редактирования и удаления данных.
* Таблицы и графики по наличию, распределению и состоянию ресурсов.
* Оповещения о необходимости обслуживания или замены ресурсов.
* Уведомления о критических изменениях в состоянии ресурсов.
* Лог изменений, внесенных пользователями.

**1.2.5** Требования к временным характеристикам

* Время выполнения стандартных операций (добавление, редактирование, удаление записей) не должно превышать 2 секунд.
* Время формирования отчетов при объеме данных до 10 000 записей не должно превышать 5 секунд.
* Загрузка основного интерфейса приложения должна занимать не более 3 секунд.
* Поиск и фильтрация данных должны выполняться за время, не превышающее 1 секунду при объеме данных до 10 000 записей.

**1.2.6** Требования к надежности

* Система должна обеспечивать бесперебойную работу при нагрузке до 10 одновременных пользователей.
* В случае сбоя система должна восстанавливать работоспособность в течение 1 минуты.
* Автоматическое резервное копирование данных должно выполняться ежедневно. Хранение резервных копий — 30 дней.
* Система должна обеспечивать целостность данных при сбоях (использование транзакций).
* В случае обнаружения ошибок в данных система должна уведомлять администратора.
* Ведение журнала ошибок для последующего анализа и устранения неисправностей.
* Проведение нагрузочного тестирования для проверки устойчивости системы при пиковых нагрузках.

**1.2.7** Требования к составу и параметрам технических и программных средств

Аппаратные требования для сервера (для развертывания системы):

* Процессор: Intel Core i5 2.2 ГГц или выше (или аналогичный процессор AMD).
* Оперативная память: 8 ГБ DDR3 1600 МГц или выше.
* Накопитель: HDD или SSD объемом 64 ГБ и более (рекомендуется SSD для повышения производительности).

Аппаратные требования для клиентских устройств (для работы с приложением):

* Процессор: Intel Core i3 1.8 ГГц или выше (или аналогичный процессор AMD).
* Оперативная память: 4 ГБ DDR3 или выше.
* Накопитель: HDD или SSD объемом 32 ГБ и более.
* Монитор: разрешение не менее 1280x720 пикселей.

Программные требования для сервера:

* Операционная система: Windows Server 2016/2019.
* СУБД: MySQL 8.0.
* Среда выполнения: .NET Core 9.0.

Программные требования для клиентских устройств:

* Операционная система: Windows 10 (64-битная версия) или выше.

**1.2.8** Требования к информационной и программной совместимости

* Приложение должно быть совместимо с Windows 10 (64-битная версия) и выше;
* Приложение должно быть совместимо с СУБД MySQL 8.0 или выше.
* Приложение должно корректно работать с устройствами ввода (клавиатура, мышь, сенсорные экраны).

**1.2.9** Обоснование выбора языка и сред разработки

Для разработки клиентской части приложения выбран язык программирования C# и технология WPF (Windows Presentation Foundation). WPF предоставляет широкие возможности для создания интуитивно понятных и визуально привлекательных пользовательских интерфейсов. Интеграция с платформой .NET обеспечивает высокую производительность, безопасность и совместимость с операционной системой Windows, что важно для клиентской части приложения.

Для разработки базы данных выбрана MySQL и инструмент MySQL Workbench. MySQL Workbench предоставляет удобный графический интерфейс для проектирования, разработки и администрирования баз данных. Также инструмент предоставляет функцию «реверсивного инжиниринга» – т.е. построение схема уже описанных языком SQL.

## 1.3 Постановка задачи

В рамках данной курсовой работы планируется разработать базу данных и приложение для взаимодействия с базой данных, которых будет достаточно для управления арсеналом. Таким образом, для выполнения поставленной задачи необходимо выполнить следующие задачи:

* разработать физическую модель базы данных для реляционной базы данных MySQL;
* разработать приложение с визуальным пользовательским интерфейсом для взаимодействия с физической базой данных;
* произвести тестирование базы данных;
* произвести тестирование приложения с визуальным пользовательским интерфейсом.

# АНАЛИЗ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## 2.1 Используемые технологии при разработке

В качестве СУБД была выбрана MySQL. Она обеспечивает высокую производительность даже при работе с большими объемами данных, что критически важно для приложения управления арсеналом, где требуется обработка и хранение значительного количества информации. Ниже перечислены основные преимущества этой СУБД:

* поддерживает транзакции, что гарантирует консистентность данных. Это особенно важно для приложений, где точность и целостность данных являются ключевыми требованиями;
* легко масштабируется как вертикально (увеличение мощности сервера), так и горизонтально (распределение нагрузки между несколькими серверами);
* имеет огромное сообщество разработчиков, что упрощает поиск решений для возникающих проблем. Также существует множество документации, учебных материалов и форумов;
* интегрируется с большинством современных языков программирования, включая C#.

В качестве платформы была выбрана платформа .NET, а языком разработки клиентского приложения был выбран C# в связке с технологией WPF.

C# является современным, строго типизированным языком программирования, который поддерживает объектно-ориентированное программирование (ООП), функциональное программирование и асинхронные операции. Это позволяет писать чистый, поддерживаемый и расширяемый код.

.NET предоставляет мощную среду выполнения, которая обеспечивает высокую производительность, безопасность и кроссплатформенность (с появлением .NET Core и .NET 5+).

WPF (Windows Presentation Foundation) позволяет создавать современные, визуально привлекательные интерфейсы с использованием XAML. Это разделяет логику приложения и его интерфейс, что упрощает разработку и поддержку. Также WPF поддерживает сложные анимации, стилизацию и привязку данных, что делает его идеальным выбором для создания десктопных приложений с богатым пользовательским интерфейсом.

## Спецификация функциональных требований

1. Выбор таблицы:

1.1. Пользователь может выбрать конкретную таблицу для работы (например, "Хранилища") через выпадающий список в интерфейсе.

1.2. После выбора таблицы система отображает её содержимое в виде таблицы с данными.

2. Описание вида таблицы:

2.1. Таблица отображается в виде сетки с колонками и строками.

2.2. Каждая колонка соответствует атрибуту ресурса (например, "Наименование", "Тип", "Количество", "Место хранения").

2.3. Каждая строка представляет собой запись о ресурсе.

2.4. Над таблицей расположены кнопки для операций:

* "Добавить" (для создания новой записи).
* "Редактировать" (для изменения выбранной записи).
* "Удалить" (для удаления выбранной записи).

2.5. Под таблицей отображается строка состояния с информацией о количестве записей (например, "Всего записей: 50").

3. Добавление записи:

3.1. Пользователь нажимает кнопку "Добавить".

3.2. Система открывает модальное окно с формой для ввода данных.

3.3. Форма содержит поля, соответствующие атрибутам ресурса:

* "Наименование" (текстовое поле, обязательное).
* "Тип" (выпадающий список с предопределенными значениями, обязательное).
* "Количество" (числовое поле, обязательное).
* "Место хранения" (выпадающий список с предопределенными значениями, обязательное).
* "Состояние" (выпадающий список с предопределенными значениями, обязательное).
* "Срок эксплуатации" (поле выбора даты, необязательное).

3.4. Пользователь заполняет форму и нажимает кнопку "Сохранить".

3.5. Система проверяет корректность введенных данных:

* Обязательные поля должны быть заполнены.
* Числовые поля должны содержать только положительные числа.

3.6. Если данные корректны, система добавляет новую запись в таблицу и обновляет её отображение.

3.7. Если данные некорректны, система отображает сообщение об ошибке с указанием проблемных полей.

4. Редактирование записи:

4.1. Пользователь выбирает запись в таблице (щелчок по строке).

4.2. Пользователь нажимает кнопку "Редактировать".

4.3. Система открывает модальное окно с формой, предзаполненной данными выбранной записи.

4.4. Пользователь изменяет необходимые поля и нажимает кнопку "Сохранить".

4.5. Система проверяет корректность введенных данных (аналогично добавлению).

4.6. Если данные корректны, система обновляет запись в таблице и обновляет её отображение.

4.7. Если данные некорректны, система отображает сообщение об ошибке.

5. Удаление записи:

5.1. Пользователь выбирает запись в таблице (щелчок по строке).

5.2. Пользователь нажимает кнопку "Удалить".

5.3. Система отображает диалоговое окно с подтверждением удаления (например, "Вы уверены, что хотите удалить эту запись?").

5.4. Если пользователь подтверждает удаление, система проверяет, не используется ли запись в других таблицах (например, в заявках или распределениях).

5.4.1. Если запись не используется, система удаляет её из таблицы и обновляет отображение.

5.4.2. Если запись используется, система отображает сообщение об ошибке (например, "Невозможно удалить запись, так как она используется в других данных").

5.5. Если пользователь отменяет удаление, система закрывает диалоговое окно без изменений.

Дополнительные требования:

6.1. Пользователь должен иметь возможность отменить операцию добавления или редактирования до сохранения (кнопка "Отмена" в модальном окне).

6.2. После успешного добавления, редактирования или удаления записи система должна отображать уведомление об успешном выполнении операции.

6.3. Все изменения в таблице должны немедленно отражаться в интерфейсе без необходимости перезагрузки страницы.

6.4. При работе с большими объемами данных таблица должна поддерживать пагинацию (разбиение на страницы) и поиск по ключевым словам.

## Ключевые особенности информационной системы

Особенностями разрабатываемой информационной системы по сравнению с аналогами являются:

* централизованное хранение данных;
* высокая консистентность данных;
* безопасность данных;
* удобство использования;
* отказоустойчивость и надежность.

Метриками качества будут выступать оценки следующих показателей:

* Время отклика системы: ≤ 2 секунды для стандартных операций.
* Время формирования отчетов: ≤ 5 секунд при объеме данных до 10 000 записей.
* Время восстановления после сбоя: ≤ 1 минута.
* Частота резервного копирования: ежедневно.
* Возможность увеличения объема данных до 1 млн записей.

## Пользователи системы и их роли

Всего в системе будет 4 роли для пользователей. Возможности и ограничения каждой роли представлены ниже в сводных таблицах.

1. Гость (неавторизованный пользователь):

|  |  |
| --- | --- |
| Возможности | Ограничения |
| Просмотр общей информации о системе (например, описание функционала, контакты поддержки). | Нет доступа к данным об арсенале. |
| Доступ к форме авторизации. | Невозможность выполнения каких-либо операций с ресурсами. |

1. Пользователь (обычный сотрудник):

|  |  |
| --- | --- |
| Возможности | Ограничения |
| Просмотр информации о ресурсах, доступных для его подразделения или личного использования. | Нет доступа к данным других подразделений. |
| Подача заявок на получение ресурсов. | Невозможность редактирования или удаления данных. |
| Просмотр истории использования ресурсов, за которые он ответственен. |  |

1. Менеджер (руководитель подразделения):

|  |  |
| --- | --- |
| Возможности | Ограничения |
| Просмотр и управление ресурсами, закрепленными за его подразделением. | Нет доступа к данным других подразделений. |
| Утверждение или отклонение заявок на ресурсы от сотрудников. | Невозможность изменения системных настроек. |
| Формирование отчетов по использованию ресурсов в своем подразделении. |  |

1. Администратор (системный администратор):

|  |  |
| --- | --- |
| Возможности | Ограничения |
| Полный доступ ко всем данным и функциям системы. | Нет ограничений (полный доступ). |
| Управление пользователями (добавление, редактирование, удаление учетных записей). |  |
| Настройка ролей и прав доступа. |  |
| Мониторинг и устранение сбоев в работе системы. |  |
| Резервное копирование и восстановление данных. |  |

## Параметры поиска

На основе ранее описанных входных данных, параметры поиска могут включать:

* Данные о ресурсах:
  + Наименование ресурса.
  + Тип ресурса.
  + Характеристики ресурса.
  + Состояние ресурса.
  + Место хранения.
  + Срок годности.
* Данные о пользователях:
  + ФИО пользователя.
  + Роль пользователя.
  + Подразделение пользователя.
* Данные о распределении ресурсов:
  + Информация о назначении ресурсов
  + Дата выдачи ресурса.
  + Дата возврата ресурса.
  + Данные о состоянии ресурсов.
  + Дата последнего обслуживания.
  + Критические изменения в состоянии.
* Дополнительные параметры:
  + Период времени.
  + Количество ресурсов.

Алгоритм реализации поиска будет состоять из следующих шагов:

1. Пользователь выбирает параметры поиска через интерфейс (например, dropdown-меню, текстовые поля, чекбоксы).
2. Система преобразует выбранные параметры в SQL-запрос.
3. Запрос отправляется в базу данных (MySQL).
   1. Если запрос не возвращает результатов, система уведомляет пользователя (например, "По вашему запросу ничего не найдено").
   2. В случае ошибки в запросе (например, синтаксическая ошибка) система логирует ошибку и уведомляет администратора.
4. База данных возвращает результаты, соответствующие критериям поиска.
5. Результаты отображаются в табличном виде с возможностью сортировки и фильтрации

# 3 ИНФОЛОГИЧЕСКАЯ МОДЕЛЬ ПРЕДМЕТНОЙ ОБЛАСТИ

## 3.1 Модель базы данных

Модель базы данных была спроектирована в MySQL Workbench. При построении схемы инфологическая и даталогическая модели были объединены, поскольку проектирование велось для конкретной базы данных. На рисунке 3.1 представлена даталогическая модель.

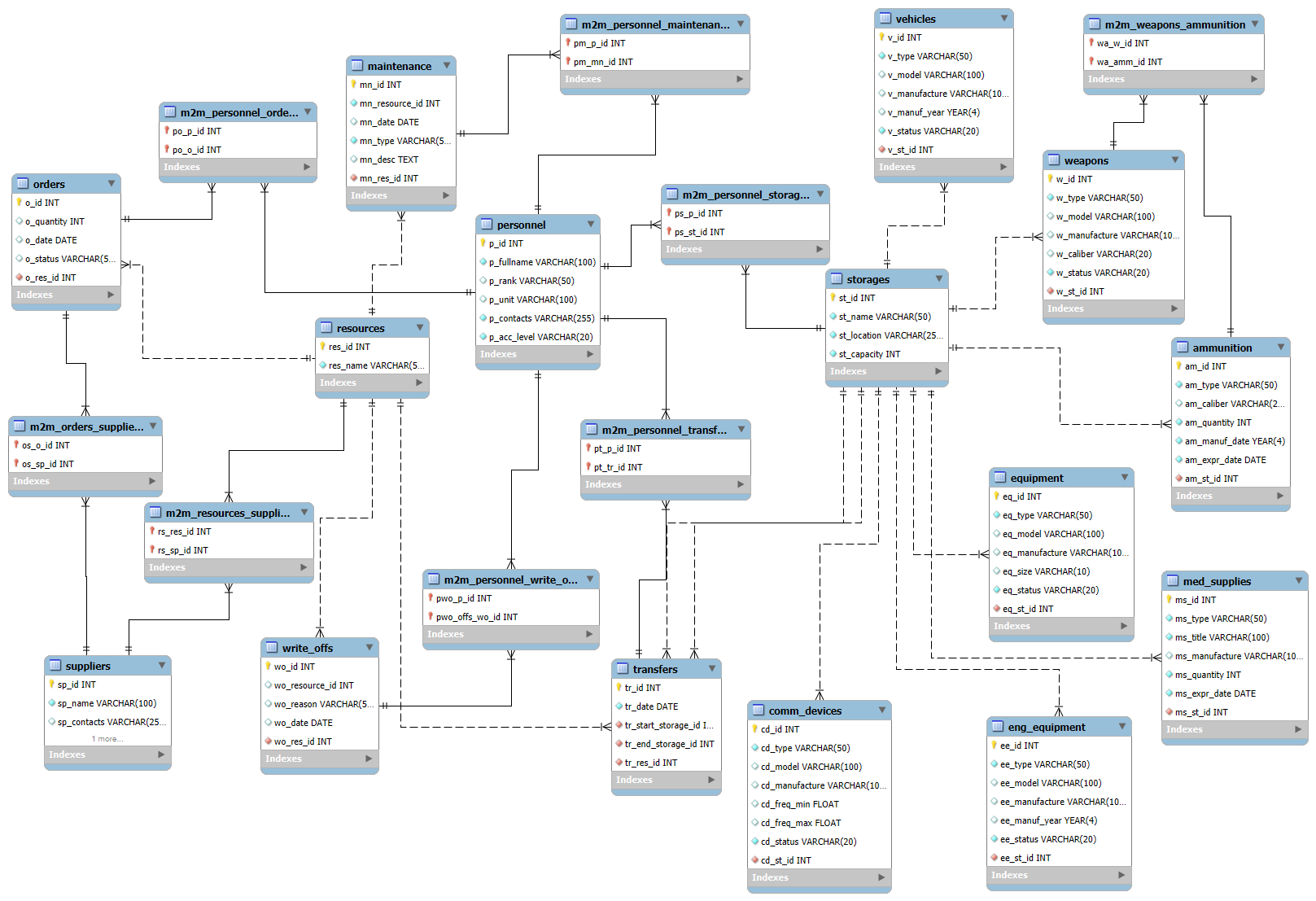


Рисунок 3.1 – Даталогическая модель базы данных

При нормализации базы данных была выбрана 2-ая нормальная форма, т.к. для достижения заданной скорости обработки данных нужны были простые первичные ключи. Переход к более высокой нормальной форме является нецелесообразным, т.к. полученные транзитивные зависимости редко меняют свои значения, а более высокая нормальная форма усложнит запросы и может понизить производительность в определенных сценариях.

## 3.2 Описание таблиц

Для разработки базы данных были выбраны следующие таблицы с определенными полями:

* боеприпасы (ammunition);
* средства связи (comm\_devices);
* инженерное оборудование (eng\_equipment);
* амуниция (equipment);
* техническое обслуживание (maintenance);
* медицинские припасы (med\_supplies);
* заказы (orders);
* сотрудники (personnel);
* ресурсы (resources);
* хранилища (storages);
* поставщики (suppliers);
* перемещения (transfers);
* техника и транспорт (vehicles);
* оружие (weapons);
* списания (write\_offs);

Всего 15 таблиц, однако некоторые таблицы имеют связь «многие ко многим». Для реализации связей были добавлены следующие промежуточные таблицы:

* m2m\_orders\_suppliers;
* m2m\_personnel\_maintenance;
* m2m\_personnel\_orders;
* m2m\_personnel\_storages;
* m2m\_personnel\_transfers;
* m2m\_personnel\_write\_offs;
* m2m\_resources\_suppliers;
* m2m\_weapons\_ammunition;

Всего 23 таблицы. Описание всех таблиц базы данных приложения представлено сводной таблицей ниже.

Таблица 3.1 – сводная таблица с описанием таблиц базы данных

|  |  |  |
| --- | --- | --- |
| № | Название | Описание |
| 1 | ammunition | Хранит информацию о боеприпасах, включая тип, калибр, количество, состояние и т.д. Используется для учета и контроля расхода боеприпасов. Связана с таблицей оружия через промежуточную таблицу. |
| 2 | comm\_devices | Содержит данные о средствах связи, таких как рации, телефоны и другие устройства. Учитывает диапазон частот, состояние, модель и т.д. Используется для управления доступностью и техническим состоянием средств связи. |
| 3 | eng\_equipment | Включает информацию об инженерном оборудовании, таком как инструменты, генераторы и спецтехника. Учитывает тип, состояние и местонахождение оборудования. Необходимо для контроля инженерного оборудования. |
| 4 | equipment | Хранит данные о снаряжении, таком как бронежилеты, каски и униформа. Учитывает тип, размер, состояние и количество. Используется для учета снаряжения. |
| 5 | maintenance | Содержит записи о техническом обслуживании техники и оборудования. Учитывает дату, тип обслуживания и ответственных сотрудников. Связана с персоналом через промежуточную таблицу. |
| 6 | med\_supplies | Включает информацию о медицинских запасах, таких как аптечки, лекарства и перевязочные материалы. Учитывает срок годности, количество и место хранения. Используется для контроля медицинских припасов и своевременного пополнения. |
| 7 | orders | Хранит данные о заказах на поставку ресурсов, оборудования или техники. Учитывает дату заказа, поставщика и ответственных сотрудников. Связана с поставщиками и персоналом через промежуточные таблицы. |
| 8 | personnel | Содержит информацию о сотрудниках, включая их должности, контактные данные и доступы. Используется для управления персоналом и назначения задач. Связана с несколькими таблицами через промежуточные таблицы. |
| 9 | resources | Хранит данные о различных ресурсах, таких как топливо, запчасти и строительные материалы. Учитывает тип, количество и поставщиков. Используется для контроля запасов и планирования закупок. |
| 10 | storages | Содержит информацию о местах хранения, таких как склады и хранилища. Учитывает местоположение, вместимость и ответственных сотрудников. Используется для управления хранилищами. |
| 11 | suppliers | Включает данные о поставщиках оборудования, ресурсов и техники. Учитывает контактную информацию и историю поставок. Связана с заказами и ресурсами через промежуточные таблицы. |
| 12 | transfers | Хранит записи о перемещении оборудования, техники или ресурсов между хранилищами. Учитывает дату, тип перемещения и ответственных сотрудников. Используется для отслеживания движения ресурсов. |
| 13 | vehicles | Содержит информацию о транспортных средствах и технике, включая тип, состояние и местонахождение. Используется для учета и контроля использования техники. |
| 14 | weapons | Хранит данные о вооружении, включая тип, серийный номер и состояние. Учитывает количество и место хранения. Связана с таблицей боеприпасов через промежуточную таблицу. |
| 15 | write\_offs | Включает записи о списании оборудования, техники или ресурсов. Учитывает причину списания, дату и ответственных сотрудников. Используется для контроля утилизации. |
| 16 | m2m\_orders\_suppliers | Промежуточная таблица для связи заказов и поставщиков. Учитывает, какие поставщики участвуют в выполнении заказов. Используется для управления цепочками поставок. |
| 17 | m2m\_personnel\_maintenance | Промежуточная таблица для связи сотрудников и технического обслуживания. Учитывает, кто отвечает за проведение обслуживания. Используется для контроля за обслуживанием. |
| 18 | m2m\_personnel\_orders | Промежуточная таблица для связи сотрудников и заказов. Учитывает, кто отвечает за оформление и выполнение заказов. Используется для управления заказами. |
| 19 | m2m\_personnel\_storages | Промежуточная таблица для связи сотрудников и хранилищ. Учитывает, кто отвечает за управление конкретными хранилищами. Используется для контроля доступа. |
| 20 | m2m\_personnel\_transfers | Промежуточная таблица для связи сотрудников и перемещений. Учитывает, кто отвечает за организацию и контроль перемещений. Используется для контроля за перемещением ресурсов. |
| 21 | m2m\_personnel\_write\_offs | Промежуточная таблица для связи сотрудников и списаний. Учитывает, кто отвечает за процесс списания. Используется для контроля списаний. |
| 22 | m2m\_resources\_suppliers | Промежуточная таблица для связи ресурсов и поставщиков. Учитывает, какие ресурсы поставляются конкретными поставщиками. Используется для управления закупками. |
| 23 | m2m\_weapons\_ammunition | Промежуточная таблица для связи оружия и боеприпасов. Учитывает, какие боеприпасы подходят для конкретного оружия. Используется для управления совместимостью. |

# 4 ПОДРОБНОЕ ОПИСАНИЕ БИЗНЕС-ЛОГИКИ

## 4.1 Триггеры

Триггер – специальный объект базы данных, описывающий перечень действий, которые необходимо автоматически выполнить при наступлении указанного события.

В базе данных присутствует типовая связака триггеров, изменяющая текущее доступное место в конкретном хранилище, когда происходит добавление ресурса в одну из зависимых таблиц.

Один из триггеров проверяет, достаточна ли текущая вместимость конкретного хранилища для добавления ресурса. В случае, если вместимость недостаточна, происходит генерация соответствующего сигнала. Код данного тригера для одной из таблиц представлен ниже:

CREATE TRIGGER before\_ammunition\_insert

BEFORE INSERT ON ammunition

FOR EACH ROW

BEGIN

DECLARE current\_capacity INT;

SELECT st\_capacity INTO current\_capacity

FROM storages

WHERE st\_id = NEW.am\_st\_id;

IF current\_capacity < NEW.am\_quantity THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Not enough capacity for chosen storage';

END IF;

END;

Другой триггер уменьшает доступное в хранилище место. Код данного триггера представлен ниже:

CREATE TRIGGER after\_ammunition\_insert

AFTER INSERT ON ammunition

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity - NEW.am\_quantity

WHERE st\_id = NEW.am\_st\_id;

END;

## 4.2 Хранимые процедуры

Хранимая процедура – подпрограмма (возможно, параметризованная), предназначенная для выполнения ряда операций с данными и структурами базы данных, хранимая на стороне базы данных и доступная как для вызова из кода других процедур и триггеров, так и для непосредственного исполнения.

Одна из процедур получает минимальный порог количества для всех ресурсов и проверяет текущее количество каждого ресурса. В случае, если количество ресурса меньше указанного, процедура создает заказ на пополнение до указанного количества. Код указанной процедуры представлен ниже:

CREATE PROCEDURE auto\_replenish\_resources(IN p\_threshold INT)

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE resource\_id INT;

DECLARE resource\_name VARCHAR(255);

DECLARE current\_quantity INT;

DECLARE supplier\_id INT;

DECLARE cur CURSOR FOR

SELECT

r.res\_id,

r.res\_name,

r.res\_quantity,

s.sp\_id AS supplier\_id

FROM

resources r

JOIN

m2m\_resources\_suppliers mrs ON r.res\_id = mrs.rs\_res\_id

JOIN

suppliers s ON mrs.rs\_sp\_id = s.sp\_id

WHERE

r.res\_quantity < p\_threshold;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO resource\_id, resource\_name, current\_quantity, supplier\_id;

IF done THEN

LEAVE read\_loop;

END IF;

INSERT INTO orders (o\_res\_id, o\_quantity, o\_status, o\_date)

VALUES (resource\_id, p\_threshold - current\_quantity, 'Processing', NOW());

INSERT INTO m2m\_orders\_suppliers (os\_o\_id, os\_sp\_id)

VALUES (LAST\_INSERT\_ID(), supplier\_id);

END LOOP;

CLOSE cur;

END

## 4.3 Представления

Представление – виртуальная производная переменная отношения, значением которой является результат вычисления реляционного выражения (выполнения запроса), заданного при создании представления (такое выражение должно ссылаться хотя бы на одну переменную отношения).

Одно из представлений показывает все заказы, которые еще не выполнены. Код данного представления указан ниже:

CREATE OR REPLACE VIEW active\_orders AS

SELECT

o.o\_id AS order\_id,

o.o\_date,

s.sp\_name AS supplier\_name,

o.o\_status

FROM

orders o

JOIN

m2m\_orders\_suppliers mos ON o.o\_id = mos.os\_o\_id

JOIN

suppliers s ON mos.os\_sp\_id = s.sp\_id

WHERE

o.o\_status = 'Processing';

# 5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Тестирование программного обеспечения играет неотъемлемую роль в обеспечении качества, позволяя выявить и исправить ошибки, убедиться в правильном функционировании, соответствии требованиям и ожиданиям пользователей, а также повысить надежность и стабильность программного продукта. Поскольку в ходе выполнения курсового проекта разрабатывалась как база данных, так и приложение, тестированию подвергаются оба компонента системы. Проведённые тесты описываются в таблице 5.1.

Таблица 5.1 – Таблица тест-кейсов для базы данных

|  |  |  |
| --- | --- | --- |
| **Действия** | **Ожидаемый результат** | **Результат** |
| Вставка нового поставщика (suppliers) с корректными данными | Кортеж успешно добавлен в таблицу suppliers. Ошибок нет. | Кортеж добавлен, ошибок нет. |
| Попытка вставить поставщика с уже существующим supplier\_id | Ошибка: нарушение уникальности первичного ключа. Данные не добавлены. | Ошибка: UNIQUE constraint failed. |
| Обновление количества боеприпасов (ammunition) для существующей записи | Значение AmQuantity успешно изменено. | Данные обновлены. |
| Попытка удаления склада (storages), на который ссылаются боеприпасы | Ошибка: нарушение внешнего ключа. Удаление отменено. | Ошибка: FOREIGN KEY constraint failed. |
| Вызов хранимой процедуры auto\_replenish\_resources(100) | Все ресурсы с количеством < 100 пополнены до минимального порога. | Данные обновлены согласно условию. |

Таблица 5.2 – Таблица тест-кейсов для визуального пользовательского интерфейса

|  |  |  |
| --- | --- | --- |
| **Действия** | **Ожидаемый результат** | **Результат** |
| Добавление нового боеприпаса через форму с корректными данными | Запись появляется в таблице без ошибок. Статус: "Item added successfully". | Данные отображаются, статус подтвержден. |
| Попытка ввода отрицательного количества боеприпасов (AmQuantity = -5) | Ошибка валидации: поле подсвечивается красным. Кнопка "Save" неактивна. | Форма блокирует отправку, выводит сообщение. |
| Выбор сущности Ammunition в ComboBox и нажатие "Refresh" | Таблица обновляется, отображая актуальные данные из БД. | Данные загружены, интерфейс реагирует. |
| Попытка удаления записи без подтверждения (нажатие "Cancel" в диалоге) | Диалог закрывается. Данные остаются неизменными. | Удаление отменено. |

# ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта были разработаны база данных и приложение с графическим интерфейсом, которые позволят управлять арсеналом. При разработке данного средства были успешно выполнены следующие задачи:

* разработка физической модель базы данных для реляционной базы данных MySQL;
* разработка приложение с визуальным пользовательским интерфейсом для взаимодействия с физической базой данных;
* тестирование базы данных;
* тестирование приложения с визуальным пользовательским интерфейсом.

Для успешного выполнения поставленных задач потребовалось изучить принципы проектирование реляционных БД, правила взаимодействия с MySQL, архитектуру MVVM для WPF приложений.

Существует множество способов улучшить данное программное средство:

* возможность создавать собственные хранимые процедуры через визуальный пользовательский интерфейс;
* выбор подключаемой базы данных и сервера;
* формирование отчетов на основе текущего наполнения базы данных.

Программное средство «To Arms Inc.» с базой данных позволяют пользователям легко и безопасно управлять арсеналом.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Куликов Святослав Святославович «Реляционные базы данных в примерах» (2021) – 422 с.

[2] Куликов Святослав Святославович «Работа с MySQL, MSSQL Server и Oracle» (2021) – 602 с.

[3] Бен Форта «SQL за 10 минут» (2021) – 320 с.

[4] Род Стивенс «Проектирование баз данных. Полное руководство» (2019) – 528 с.

[5] MySQL 8 для больших данных [Электронный ресурс]. – Режим доступа: <https://dev.mysql.com/tech-resources/articles/mysql8-big-data/>. Дата доступа: 10.04.2025

[6] Основы проектирования реляционных баз данных [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/sql/relational-databases/database-design/>. Дата доступа: 12.04.2025

[7] WPF и MySQL: создание клиент-серверных приложений [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/wpf/>. Дата доступа: 15.04.2025

[8] Entity Framework Core: работа с базами данных в .NET [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/ef/core/>. Дата доступа: 18.04.2025

[9] Мэтью Мак-Дональд «WPF: Windows Presentation Foundation в .NET 4.5» (2013) – 1024 с.

[10] Эндрю Троелсен «Язык программирования C# 10 и платформа .NET 6» (2022) – 1320 с.

ПРИЛОЖЕНИЕ А.

**(обязательное)**

**Исходный код программы**

-- MySQL Workbench Forward Engineering

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 arsenal

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

DROP SCHEMA IF EXISTS `arsenal` ;

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

-- Schema arsenal

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

CREATE SCHEMA IF NOT EXISTS `arsenal` DEFAULT CHARACTER SET utf8 ;

USE `arsenal` ;

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

-- Table `arsenal`.`storages`

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

DROP TABLE IF EXISTS `arsenal`.`storages` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`storages` (

`st\_id` INT NOT NULL AUTO\_INCREMENT,

`st\_name` VARCHAR(50) NOT NULL,

`st\_location` VARCHAR(255) NOT NULL,

`st\_capacity` INT NOT NULL,

PRIMARY KEY (`st\_id`),

UNIQUE INDEX `st\_id\_UNIQUE` (`st\_id` ASC) VISIBLE)

ENGINE = InnoDB;

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

-- Table `arsenal`.`weapons`

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

DROP TABLE IF EXISTS `arsenal`.`weapons` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`weapons` (

`w\_id` INT NOT NULL AUTO\_INCREMENT,

`w\_type` VARCHAR(50) NOT NULL,

`w\_model` VARCHAR(100) NULL,

`w\_manufacture` VARCHAR(100) NULL,

`w\_caliber` VARCHAR(20) NULL,

`w\_status` VARCHAR(20) NOT NULL,

`w\_st\_id` INT NOT NULL,

PRIMARY KEY (`w\_id`),

INDEX `fk\_weapons\_storages\_idx` (`w\_st\_id` ASC) VISIBLE,

UNIQUE INDEX `w\_id\_UNIQUE` (`w\_id` ASC) VISIBLE,

CONSTRAINT `fk\_weapons\_storage`

FOREIGN KEY (`w\_st\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`ammunition`

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

DROP TABLE IF EXISTS `arsenal`.`ammunition` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`ammunition` (

`am\_id` INT NOT NULL AUTO\_INCREMENT,

`am\_type` VARCHAR(50) NOT NULL,

`am\_caliber` VARCHAR(20) NULL,

`am\_quantity` INT UNSIGNED NOT NULL,

`am\_manuf\_date` YEAR(4) NOT NULL,

`am\_expr\_date` DATE NOT NULL,

`am\_st\_id` INT NOT NULL,

PRIMARY KEY (`am\_id`),

INDEX `fk\_ammunition\_storages1\_idx` (`am\_st\_id` ASC) VISIBLE,

UNIQUE INDEX `am\_id\_UNIQUE` (`am\_id` ASC) VISIBLE,

CONSTRAINT `fk\_ammunition\_storage`

FOREIGN KEY (`am\_st\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`vehicles`

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

DROP TABLE IF EXISTS `arsenal`.`vehicles` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`vehicles` (

`v\_id` INT NOT NULL AUTO\_INCREMENT,

`v\_type` VARCHAR(50) NOT NULL,

`v\_model` VARCHAR(100) NULL,

`v\_manufacture` VARCHAR(100) NULL,

`v\_manuf\_year` YEAR(4) NULL,

`v\_status` VARCHAR(20) NOT NULL,

`v\_st\_id` INT NOT NULL,

PRIMARY KEY (`v\_id`),

INDEX `fk\_vehicles\_storages1\_idx` (`v\_st\_id` ASC) VISIBLE,

UNIQUE INDEX `v\_id\_UNIQUE` (`v\_id` ASC) VISIBLE,

CONSTRAINT `fk\_vehicles\_storage`

FOREIGN KEY (`v\_st\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`equipment`

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

DROP TABLE IF EXISTS `arsenal`.`equipment` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`equipment` (

`eq\_id` INT NOT NULL AUTO\_INCREMENT,

`eq\_type` VARCHAR(50) NOT NULL,

`eq\_model` VARCHAR(100) NULL,

`eq\_manufacture` VARCHAR(100) NULL,

`eq\_size` VARCHAR(10) NULL,

`eq\_status` VARCHAR(20) NOT NULL,

`eq\_st\_id` INT NOT NULL,

PRIMARY KEY (`eq\_id`),

INDEX `fk\_equipment\_storages1\_idx` (`eq\_st\_id` ASC) VISIBLE,

UNIQUE INDEX `eq\_id\_UNIQUE` (`eq\_id` ASC) VISIBLE,

CONSTRAINT `fk\_equipment\_storage`

FOREIGN KEY (`eq\_st\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`comm\_devices`

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

DROP TABLE IF EXISTS `arsenal`.`comm\_devices` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`comm\_devices` (

`cd\_id` INT NOT NULL AUTO\_INCREMENT,

`cd\_type` VARCHAR(50) NOT NULL,

`cd\_model` VARCHAR(100) NULL,

`cd\_manufacture` VARCHAR(100) NULL,

`cd\_freq\_min` FLOAT UNSIGNED NULL,

`cd\_freq\_max` FLOAT UNSIGNED NULL,

`cd\_status` VARCHAR(20) NOT NULL,

`cd\_st\_id` INT NOT NULL,

PRIMARY KEY (`cd\_id`),

INDEX `fk\_comm\_devices\_storages1\_idx` (`cd\_st\_id` ASC) VISIBLE,

UNIQUE INDEX `cd\_id\_UNIQUE` (`cd\_id` ASC) VISIBLE,

CONSTRAINT `fk\_comm\_devices\_storage`

FOREIGN KEY (`cd\_st\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`med\_supplies`

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

DROP TABLE IF EXISTS `arsenal`.`med\_supplies` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`med\_supplies` (

`ms\_id` INT NOT NULL AUTO\_INCREMENT,

`ms\_type` VARCHAR(50) NOT NULL,

`ms\_title` VARCHAR(100) NOT NULL,

`ms\_manufacture` VARCHAR(100) NULL,

`ms\_quantity` INT UNSIGNED NOT NULL,

`ms\_expr\_date` DATE NOT NULL,

`ms\_st\_id` INT NOT NULL,

PRIMARY KEY (`ms\_id`),

INDEX `fk\_med\_supplies\_storages1\_idx` (`ms\_st\_id` ASC) VISIBLE,

UNIQUE INDEX `ms\_id\_UNIQUE` (`ms\_id` ASC) VISIBLE,

CONSTRAINT `fk\_med\_supplies\_storage`

FOREIGN KEY (`ms\_st\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`eng\_equipment`

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

DROP TABLE IF EXISTS `arsenal`.`eng\_equipment` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`eng\_equipment` (

`ee\_id` INT NOT NULL AUTO\_INCREMENT,

`ee\_type` VARCHAR(50) NOT NULL,

`ee\_model` VARCHAR(100) NULL,

`ee\_manufacture` VARCHAR(100) NULL,

`ee\_manuf\_year` YEAR(4) NULL,

`ee\_status` VARCHAR(20) NOT NULL,

`ee\_st\_id` INT NOT NULL,

PRIMARY KEY (`ee\_id`),

INDEX `fk\_eng\_equipment\_storages1\_idx` (`ee\_st\_id` ASC) VISIBLE,

UNIQUE INDEX `ee\_id\_UNIQUE` (`ee\_id` ASC) VISIBLE,

CONSTRAINT `fk\_eng\_equipment\_storage`

FOREIGN KEY (`ee\_st\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`personnel`

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

DROP TABLE IF EXISTS `arsenal`.`personnel` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`personnel` (

`p\_id` INT NOT NULL AUTO\_INCREMENT,

`p\_fullname` VARCHAR(100) NOT NULL,

`p\_rank` VARCHAR(50) NULL,

`p\_unit` VARCHAR(100) NULL,

`p\_contacts` VARCHAR(255) NOT NULL,

`p\_acc\_level` VARCHAR(20) NOT NULL,

PRIMARY KEY (`p\_id`),

UNIQUE INDEX `p\_id\_UNIQUE` (`p\_id` ASC) VISIBLE)

ENGINE = InnoDB;

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

-- Table `arsenal`.`resources`

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

DROP TABLE IF EXISTS `arsenal`.`resources` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`resources` (

`res\_id` INT NOT NULL AUTO\_INCREMENT,

`res\_name` VARCHAR(50) NOT NULL,

`res\_quantity` BIGINT NULL,

PRIMARY KEY (`res\_id`),

UNIQUE INDEX `res\_name\_UNIQUE` (`res\_name` ASC) VISIBLE,

UNIQUE INDEX `res\_id\_UNIQUE` (`res\_id` ASC) VISIBLE)

ENGINE = InnoDB;

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

-- Table `arsenal`.`transfers`

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

DROP TABLE IF EXISTS `arsenal`.`transfers` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`transfers` (

`tr\_id` INT NOT NULL AUTO\_INCREMENT,

`tr\_date` DATE NOT NULL,

`tr\_start\_storage\_id` INT NOT NULL,

`tr\_end\_storage\_id` INT NOT NULL,

`tr\_res\_id` INT NOT NULL,

PRIMARY KEY (`tr\_id`),

INDEX `fk\_transfers\_storages1\_idx` (`tr\_start\_storage\_id` ASC) VISIBLE,

INDEX `fk\_transfers\_storages2\_idx` (`tr\_end\_storage\_id` ASC) VISIBLE,

INDEX `fk\_transfers\_resources1\_idx` (`tr\_res\_id` ASC) VISIBLE,

UNIQUE INDEX `tr\_id\_UNIQUE` (`tr\_id` ASC) VISIBLE,

CONSTRAINT `fk\_transfers\_start\_storage`

FOREIGN KEY (`tr\_start\_storage\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_transfers\_end\_storages`

FOREIGN KEY (`tr\_end\_storage\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_transfers\_resource`

FOREIGN KEY (`tr\_res\_id`)

REFERENCES `arsenal`.`resources` (`res\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`maintenance`

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

DROP TABLE IF EXISTS `arsenal`.`maintenance` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`maintenance` (

`mn\_id` INT NOT NULL AUTO\_INCREMENT,

`mn\_date` DATE NULL,

`mn\_type` VARCHAR(50) NOT NULL,

`mn\_desc` TEXT NULL,

`mn\_res\_id` INT NOT NULL,

PRIMARY KEY (`mn\_id`),

INDEX `fk\_maintenance\_resources1\_idx` (`mn\_res\_id` ASC) VISIBLE,

UNIQUE INDEX `mn\_id\_UNIQUE` (`mn\_id` ASC) VISIBLE,

CONSTRAINT `fk\_maintenance\_resource`

FOREIGN KEY (`mn\_res\_id`)

REFERENCES `arsenal`.`resources` (`res\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`write\_offs`

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

DROP TABLE IF EXISTS `arsenal`.`write\_offs` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`write\_offs` (

`wo\_id` INT NOT NULL AUTO\_INCREMENT,

`wo\_reason` VARCHAR(50) NULL,

`wo\_date` DATE NULL,

`wo\_res\_id` INT NOT NULL,

PRIMARY KEY (`wo\_id`),

INDEX `fk\_write\_offs\_resources1\_idx` (`wo\_res\_id` ASC) VISIBLE,

CONSTRAINT `fk\_write\_offs\_resource`

FOREIGN KEY (`wo\_res\_id`)

REFERENCES `arsenal`.`resources` (`res\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`suppliers`

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

DROP TABLE IF EXISTS `arsenal`.`suppliers` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`suppliers` (

`sp\_id` INT NOT NULL AUTO\_INCREMENT,

`sp\_name` VARCHAR(100) NOT NULL,

`sp\_contacts` VARCHAR(255) NULL,

`sp\_rating` DECIMAL(3,2) NULL,

PRIMARY KEY (`sp\_id`),

UNIQUE INDEX `sp\_id\_UNIQUE` (`sp\_id` ASC) VISIBLE)

ENGINE = InnoDB;

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

-- Table `arsenal`.`orders`

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

DROP TABLE IF EXISTS `arsenal`.`orders` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`orders` (

`o\_id` INT NOT NULL AUTO\_INCREMENT,

`o\_quantity` INT NULL,

`o\_date` DATE NULL,

`o\_status` VARCHAR(50) NULL,

`o\_res\_id` INT NOT NULL,

PRIMARY KEY (`o\_id`),

INDEX `fk\_orders\_resources1\_idx` (`o\_res\_id` ASC) VISIBLE,

UNIQUE INDEX `o\_id\_UNIQUE` (`o\_id` ASC) VISIBLE,

CONSTRAINT `fk\_orders\_resource`

FOREIGN KEY (`o\_res\_id`)

REFERENCES `arsenal`.`resources` (`res\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`m2m\_weapons\_ammunition`

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

DROP TABLE IF EXISTS `arsenal`.`m2m\_weapons\_ammunition` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`m2m\_weapons\_ammunition` (

`wa\_w\_id` INT NOT NULL,

`wa\_am\_id` INT NOT NULL,

PRIMARY KEY (`wa\_w\_id`, `wa\_am\_id`),

INDEX `fk\_weapons\_has\_ammunition\_ammunition1\_idx` (`wa\_am\_id` ASC) VISIBLE,

INDEX `fk\_weapons\_has\_ammunition\_weapons1\_idx` (`wa\_w\_id` ASC) VISIBLE,

CONSTRAINT `fk\_m2m\_weapons\_ammunition\_weapons`

FOREIGN KEY (`wa\_w\_id`)

REFERENCES `arsenal`.`weapons` (`w\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_m2m\_weapons\_ammunition\_ammunition`

FOREIGN KEY (`wa\_am\_id`)

REFERENCES `arsenal`.`ammunition` (`am\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`m2m\_personnel\_storages`

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

DROP TABLE IF EXISTS `arsenal`.`m2m\_personnel\_storages` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`m2m\_personnel\_storages` (

`ps\_p\_id` INT NOT NULL,

`ps\_st\_id` INT NOT NULL,

PRIMARY KEY (`ps\_p\_id`, `ps\_st\_id`),

INDEX `fk\_personnel\_has\_storages\_storages1\_idx` (`ps\_st\_id` ASC) VISIBLE,

INDEX `fk\_personnel\_has\_storages\_personnel1\_idx` (`ps\_p\_id` ASC) VISIBLE,

CONSTRAINT `fk\_m2m\_personnel\_storages\_personnel`

FOREIGN KEY (`ps\_p\_id`)

REFERENCES `arsenal`.`personnel` (`p\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_m2m\_personnel\_storages\_storages`

FOREIGN KEY (`ps\_st\_id`)

REFERENCES `arsenal`.`storages` (`st\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`m2m\_orders\_suppliers`

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

DROP TABLE IF EXISTS `arsenal`.`m2m\_orders\_suppliers` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`m2m\_orders\_suppliers` (

`os\_o\_id` INT NOT NULL,

`os\_sp\_id` INT NOT NULL,

PRIMARY KEY (`os\_o\_id`, `os\_sp\_id`),

INDEX `fk\_orders\_has\_suppliers\_suppliers1\_idx` (`os\_sp\_id` ASC) VISIBLE,

INDEX `fk\_orders\_has\_suppliers\_orders1\_idx` (`os\_o\_id` ASC) VISIBLE,

CONSTRAINT `fk\_m2m\_orders\_suppliers\_orders`

FOREIGN KEY (`os\_o\_id`)

REFERENCES `arsenal`.`orders` (`o\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_m2m\_orders\_suppliers\_suppliers`

FOREIGN KEY (`os\_sp\_id`)

REFERENCES `arsenal`.`suppliers` (`sp\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`m2m\_personnel\_transfers`

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

DROP TABLE IF EXISTS `arsenal`.`m2m\_personnel\_transfers` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`m2m\_personnel\_transfers` (

`pt\_p\_id` INT NOT NULL,

`pt\_tr\_id` INT NOT NULL,

PRIMARY KEY (`pt\_p\_id`, `pt\_tr\_id`),

INDEX `fk\_personnel\_has\_transfers\_transfers1\_idx` (`pt\_tr\_id` ASC) VISIBLE,

INDEX `fk\_personnel\_has\_transfers\_personnel1\_idx` (`pt\_p\_id` ASC) VISIBLE,

CONSTRAINT `fk\_m2m\_personnel\_transfers\_personnel`

FOREIGN KEY (`pt\_p\_id`)

REFERENCES `arsenal`.`personnel` (`p\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_m2m\_personnel\_transfers\_transfers`

FOREIGN KEY (`pt\_tr\_id`)

REFERENCES `arsenal`.`transfers` (`tr\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`m2m\_personnel\_write\_offs`

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

DROP TABLE IF EXISTS `arsenal`.`m2m\_personnel\_write\_offs` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`m2m\_personnel\_write\_offs` (

`pwo\_p\_id` INT NOT NULL,

`pwo\_wo\_id` INT NOT NULL,

PRIMARY KEY (`pwo\_p\_id`, `pwo\_wo\_id`),

INDEX `fk\_personnel\_has\_write\_offs\_write\_offs1\_idx` (`pwo\_wo\_id` ASC) VISIBLE,

INDEX `fk\_personnel\_has\_write\_offs\_personnel1\_idx` (`pwo\_p\_id` ASC) VISIBLE,

CONSTRAINT `fk\_m2m\_personnel\_write\_offs\_personnel`

FOREIGN KEY (`pwo\_p\_id`)

REFERENCES `arsenal`.`personnel` (`p\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_m2m\_personnel\_write\_offs\_write\_offs`

FOREIGN KEY (`pwo\_wo\_id`)

REFERENCES `arsenal`.`write\_offs` (`wo\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`m2m\_resources\_suppliers`

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

DROP TABLE IF EXISTS `arsenal`.`m2m\_resources\_suppliers` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`m2m\_resources\_suppliers` (

`rs\_res\_id` INT NOT NULL,

`rs\_sp\_id` INT NOT NULL,

PRIMARY KEY (`rs\_res\_id`, `rs\_sp\_id`),

INDEX `fk\_resources\_has\_suppliers\_suppliers1\_idx` (`rs\_sp\_id` ASC) VISIBLE,

INDEX `fk\_resources\_has\_suppliers\_resources1\_idx` (`rs\_res\_id` ASC) VISIBLE,

CONSTRAINT `fk\_m2m\_resources\_suppliers\_resources`

FOREIGN KEY (`rs\_res\_id`)

REFERENCES `arsenal`.`resources` (`res\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_m2m\_resources\_suppliers\_suppliers`

FOREIGN KEY (`rs\_sp\_id`)

REFERENCES `arsenal`.`suppliers` (`sp\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`m2m\_personnel\_orders`

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

DROP TABLE IF EXISTS `arsenal`.`m2m\_personnel\_orders` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`m2m\_personnel\_orders` (

`po\_p\_id` INT NOT NULL,

`po\_o\_id` INT NOT NULL,

PRIMARY KEY (`po\_p\_id`, `po\_o\_id`),

INDEX `fk\_m2m\_personnel\_order\_order\_idx` (`po\_o\_id` ASC) VISIBLE,

CONSTRAINT `fk\_m2m\_personnel\_order\_personnel`

FOREIGN KEY (`po\_p\_id`)

REFERENCES `arsenal`.`personnel` (`p\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_m2m\_personnel\_order\_order`

FOREIGN KEY (`po\_o\_id`)

REFERENCES `arsenal`.`orders` (`o\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `arsenal`.`m2m\_personnel\_maintenance`

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

DROP TABLE IF EXISTS `arsenal`.`m2m\_personnel\_maintenance` ;

CREATE TABLE IF NOT EXISTS `arsenal`.`m2m\_personnel\_maintenance` (

`pm\_p\_id` INT NOT NULL,

`pm\_mn\_id` INT NOT NULL,

PRIMARY KEY (`pm\_p\_id`, `pm\_mn\_id`),

INDEX `fk\_personnel\_has\_maintenance\_maintenance1\_idx` (`pm\_mn\_id` ASC) VISIBLE,

INDEX `fk\_personnel\_has\_maintenance\_personnel1\_idx` (`pm\_p\_id` ASC) VISIBLE,

CONSTRAINT `fk\_m2m\_personnel\_maintenance\_personnel`

FOREIGN KEY (`pm\_p\_id`)

REFERENCES `arsenal`.`personnel` (`p\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_m2m\_personnel\_maintenance\_maintenance`

FOREIGN KEY (`pm\_mn\_id`)

REFERENCES `arsenal`.`maintenance` (`mn\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

USE `arsenal` ;

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

-- Placeholder table for view `arsenal`.`active\_orders`

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

CREATE TABLE IF NOT EXISTS `arsenal`.`active\_orders` (`order\_id` INT, `o\_date` INT, `supplier\_name` INT, `o\_status` INT);

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

-- Placeholder table for view `arsenal`.`transfer\_history`

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

CREATE TABLE IF NOT EXISTS `arsenal`.`transfer\_history` (`transfer\_id` INT, `tr\_date` INT, `resource\_name` INT, `from\_storage` INT, `to\_storage` INT, `responsible\_person` INT);

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

-- Placeholder table for view `arsenal`.`order\_history`

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

CREATE TABLE IF NOT EXISTS `arsenal`.`order\_history` (`order\_id` INT, `o\_date` INT, `resource\_name` INT, `supplier\_name` INT, `responsible\_person` INT, `o\_status` INT);

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

-- Placeholder table for view `arsenal`.`weapon\_ammunition\_compatibility`

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

CREATE TABLE IF NOT EXISTS `arsenal`.`weapon\_ammunition\_compatibility` (`weapon\_id` INT, `weapon\_name` INT, `ammunition\_id` INT, `ammunition\_type` INT);

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

-- procedure create\_order

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

USE `arsenal`;

DROP procedure IF EXISTS `arsenal`.`create\_order`;

DELIMITER $$

USE `arsenal`$$

CREATE PROCEDURE create\_order(

IN p\_resource\_id INT,

IN p\_quantity INT,

IN p\_supplier\_id INT,

IN p\_personnel\_id INT

)

BEGIN

DECLARE available\_quantity INT;

SELECT res\_quantity INTO available\_quantity

FROM resources

WHERE res\_id = p\_resource\_id;

IF available\_quantity < p\_quantity THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Not enough resources';

ELSE

INSERT INTO orders (o\_res\_id, o\_quantity, o\_status, o\_date)

VALUES (p\_resource\_id, p\_quantity, 'Processing', NOW());

INSERT INTO m2m\_orders\_suppliers (os\_o\_id, os\_sp\_id)

VALUES (LAST\_INSERT\_ID(), p\_supplier\_id);

INSERT INTO m2m\_personnel\_orders (po\_p\_id, po\_o\_id)

VALUES (p\_personnel\_id, LAST\_INSERT\_ID());

UPDATE resources

SET res\_quantity = res\_quantity - p\_quantity

WHERE res\_id = p\_resource\_id;

END IF;

END$$

DELIMITER ;

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

-- procedure auto\_replenish\_resources

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

USE `arsenal`;

DROP procedure IF EXISTS `arsenal`.`auto\_replenish\_resources`;

DELIMITER $$

USE `arsenal`$$

CREATE PROCEDURE auto\_replenish\_resources(IN p\_threshold INT)

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE resource\_id INT;

DECLARE resource\_name VARCHAR(255);

DECLARE current\_quantity INT;

DECLARE supplier\_id INT;

DECLARE cur CURSOR FOR

SELECT

r.res\_id,

r.res\_name,

r.res\_quantity,

s.sp\_id AS supplier\_id

FROM

resources r

JOIN

m2m\_resources\_suppliers mrs ON r.res\_id = mrs.rs\_res\_id

JOIN

suppliers s ON mrs.rs\_sp\_id = s.sp\_id

WHERE

r.res\_quantity < p\_threshold;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO resource\_id, resource\_name, current\_quantity, supplier\_id;

IF done THEN

LEAVE read\_loop;

END IF;

INSERT INTO orders (o\_res\_id, o\_quantity, o\_status, o\_date)

VALUES (resource\_id, p\_threshold - current\_quantity, 'Processing', NOW());

INSERT INTO m2m\_orders\_suppliers (os\_o\_id, os\_sp\_id)

VALUES (LAST\_INSERT\_ID(), supplier\_id);

END LOOP;

CLOSE cur;

END$$

DELIMITER ;

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

-- View `arsenal`.`active\_orders`

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

DROP TABLE IF EXISTS `arsenal`.`active\_orders`;

DROP VIEW IF EXISTS `arsenal`.`active\_orders` ;

USE `arsenal`;

CREATE OR REPLACE VIEW active\_orders AS

SELECT

o.o\_id AS order\_id,

o.o\_date,

s.sp\_name AS supplier\_name,

o.o\_status

FROM

orders o

JOIN

m2m\_orders\_suppliers mos ON o.o\_id = mos.os\_o\_id

JOIN

suppliers s ON mos.os\_sp\_id = s.sp\_id

WHERE

o.o\_status = 'Processing';

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

-- View `arsenal`.`transfer\_history`

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

DROP TABLE IF EXISTS `arsenal`.`transfer\_history`;

DROP VIEW IF EXISTS `arsenal`.`transfer\_history` ;

USE `arsenal`;

CREATE OR REPLACE VIEW transfer\_history AS

SELECT

t.tr\_id AS transfer\_id,

t.tr\_date,

r.res\_name AS resource\_name,

s1.st\_name AS from\_storage,

s2.st\_name AS to\_storage,

p.p\_fullname AS responsible\_person

FROM

transfers t

JOIN

resources r ON t.tr\_res\_id = r.res\_id

JOIN

storages s1 ON t.tr\_start\_storage\_id = s1.st\_id

JOIN

storages s2 ON t.tr\_end\_storage\_id = s2.st\_id

JOIN

m2m\_personnel\_transfers mpt ON t.tr\_id = mpt.pt\_tr\_id

JOIN

personnel p ON mpt.pt\_p\_id = p.p\_id;

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

-- View `arsenal`.`order\_history`

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

DROP TABLE IF EXISTS `arsenal`.`order\_history`;

DROP VIEW IF EXISTS `arsenal`.`order\_history` ;

USE `arsenal`;

CREATE OR REPLACE VIEW order\_history AS

SELECT

o.o\_id AS order\_id,

o.o\_date,

r.res\_name AS resource\_name,

s.sp\_name AS supplier\_name,

p.p\_fullname AS responsible\_person,

o.o\_status

FROM

orders o

JOIN

resources r ON o.o\_res\_id = r.res\_id

JOIN

m2m\_orders\_suppliers mos ON o.o\_id = mos.os\_o\_id

JOIN

suppliers s ON mos.os\_sp\_id = s.sp\_id

JOIN

m2m\_personnel\_orders mpo ON o.o\_id = mpo.po\_o\_id

JOIN

personnel p ON mpo.po\_o\_id = p.p\_id;

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

-- View `arsenal`.`weapon\_ammunition\_compatibility`

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

DROP TABLE IF EXISTS `arsenal`.`weapon\_ammunition\_compatibility`;

DROP VIEW IF EXISTS `arsenal`.`weapon\_ammunition\_compatibility` ;

USE `arsenal`;

CREATE OR REPLACE VIEW weapon\_ammunition\_compatibility AS

SELECT

w.w\_id AS weapon\_id,

w.w\_model AS weapon\_name,

a.am\_id AS ammunition\_id,

a.am\_type AS ammunition\_type

FROM

weapons w

JOIN

m2m\_weapons\_ammunition mwa ON w.w\_id = mwa.wa\_w\_id

JOIN

ammunition a ON mwa.wa\_am\_id = a.am\_id;

USE `arsenal`;

DELIMITER $$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_weapons\_insert` $$

USE `arsenal`$$

CREATE TRIGGER before\_weapons\_insert

BEFORE INSERT ON weapons

FOR EACH ROW

BEGIN

DECLARE current\_capacity INT;

-- Получаем текущую вместимость склада

SELECT st\_capacity INTO current\_capacity

FROM storages

WHERE st\_id = NEW.w\_st\_id;

IF current\_capacity <= 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Not enough capacity for chosen storage';

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`after\_weapons\_insert` $$

USE `arsenal`$$

CREATE TRIGGER after\_weapons\_insert

AFTER INSERT ON weapons

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.w\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_weapons\_update` $$

USE `arsenal`$$

CREATE TRIGGER before\_weapons\_update

BEFORE UPDATE ON weapons

FOR EACH ROW

BEGIN

IF OLD.w\_st\_id != NEW.w\_st\_id THEN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.w\_st\_id;

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.w\_st\_id;

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_weapons\_delete` $$

USE `arsenal`$$

CREATE TRIGGER before\_weapons\_delete

BEFORE DELETE ON weapons

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.w\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_ammunition\_insert` $$

USE `arsenal`$$

CREATE TRIGGER before\_ammunition\_insert

BEFORE INSERT ON ammunition

FOR EACH ROW

BEGIN

DECLARE current\_capacity INT;

SELECT st\_capacity INTO current\_capacity

FROM storages

WHERE st\_id = NEW.am\_st\_id;

IF current\_capacity < NEW.am\_quantity THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Not enough capacity for chosen storage';

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`after\_ammunition\_insert` $$

USE `arsenal`$$

CREATE TRIGGER after\_ammunition\_insert

AFTER INSERT ON ammunition

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity - NEW.am\_quantity

WHERE st\_id = NEW.am\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_ammunition\_update` $$

USE `arsenal`$$

CREATE TRIGGER before\_ammunition\_update

BEFORE UPDATE ON ammunition

FOR EACH ROW

BEGIN

IF OLD.am\_st\_id != NEW.am\_st\_id OR OLD.am\_quantity != NEW.am\_quantity THEN

UPDATE storages

SET st\_capacity = st\_capacity + OLD.am\_quantity

WHERE st\_id = OLD.am\_st\_id;

UPDATE storages

SET st\_capacity = st\_capacity - NEW.am\_quantity

WHERE st\_id = NEW.am\_st\_id;

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_ammunition\_delete` $$

USE `arsenal`$$

CREATE TRIGGER before\_ammunition\_delete

BEFORE DELETE ON ammunition

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity + OLD.am\_quantity

WHERE st\_id = OLD.am\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`after\_vehicle\_status\_update` $$

USE `arsenal`$$

CREATE TRIGGER after\_vehicle\_status\_update

AFTER UPDATE ON vehicles

FOR EACH ROW

BEGIN

IF NEW.v\_status = 'requires maintenance' THEN

INSERT INTO maintenance (vehicle\_id, maintenance\_date, mn\_desc)

VALUES (NEW.v\_id, NOW(), 'Automatically added: vehicle requires maintenance');

END IF;

END$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_vehicles\_insert` $$

USE `arsenal`$$

CREATE TRIGGER before\_vehicles\_insert

BEFORE INSERT ON vehicles

FOR EACH ROW

BEGIN

DECLARE current\_capacity INT;

SELECT st\_capacity INTO current\_capacity

FROM storages

WHERE st\_id = NEW.v\_st\_id;

IF current\_capacity <= 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Not enough capacity for chosen storage';

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`after\_vehicles\_insert` $$

USE `arsenal`$$

CREATE TRIGGER after\_vehicles\_insert

AFTER INSERT ON vehicles

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.v\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_vehicles\_delete` $$

USE `arsenal`$$

CREATE TRIGGER before\_vehicles\_delete

BEFORE DELETE ON vehicles

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.v\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_vehicles\_update` $$

USE `arsenal`$$

CREATE TRIGGER before\_vehicles\_update

BEFORE UPDATE ON vehicles

FOR EACH ROW

BEGIN

IF OLD.v\_st\_id != NEW.v\_st\_id THEN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.v\_st\_id;

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.v\_st\_id;

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_equipment\_insert` $$

USE `arsenal`$$

CREATE TRIGGER before\_equipment\_insert

BEFORE INSERT ON equipment

FOR EACH ROW

BEGIN

DECLARE current\_capacity INT;

SELECT st\_capacity INTO current\_capacity

FROM storages

WHERE st\_id = NEW.eq\_st\_id;

IF current\_capacity <= 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Not enough capacity for chosen storage';

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`after\_equipment\_insert` $$

USE `arsenal`$$

CREATE TRIGGER after\_equipment\_insert

AFTER INSERT ON equipment

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.eq\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_equipment\_delete` $$

USE `arsenal`$$

CREATE TRIGGER before\_equipment\_delete

BEFORE DELETE ON equipment

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.eq\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_equipment\_update` $$

USE `arsenal`$$

CREATE TRIGGER before\_equipment\_update

BEFORE UPDATE ON equipment

FOR EACH ROW

BEGIN

IF OLD.eq\_st\_id != NEW.eq\_st\_id THEN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.eq\_st\_id;

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.eq\_st\_id;

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_comm\_devices\_insert` $$

USE `arsenal`$$

CREATE TRIGGER before\_comm\_devices\_insert

BEFORE INSERT ON comm\_devices

FOR EACH ROW

BEGIN

DECLARE current\_capacity INT;

SELECT st\_capacity INTO current\_capacity

FROM storages

WHERE st\_id = NEW.cd\_st\_id;

IF current\_capacity <= 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Not enough capacity for chosen storage';

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`after\_comm\_devices\_insert` $$

USE `arsenal`$$

CREATE TRIGGER after\_comm\_devices\_insert

AFTER INSERT ON comm\_devices

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.cd\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_comm\_devices\_update` $$

USE `arsenal`$$

CREATE TRIGGER before\_comm\_devices\_update

BEFORE UPDATE ON comm\_devices

FOR EACH ROW

BEGIN

IF OLD.cd\_st\_id != NEW.cd\_st\_id THEN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.cd\_st\_id;

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.cd\_st\_id;

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_comm\_devices\_delete` $$

USE `arsenal`$$

CREATE TRIGGER before\_comm\_devices\_delete

BEFORE DELETE ON comm\_devices

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.cd\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_med\_supplies\_insert` $$

USE `arsenal`$$

CREATE TRIGGER before\_med\_supplies\_insert

BEFORE INSERT ON med\_supplies

FOR EACH ROW

BEGIN

DECLARE current\_capacity INT;

SELECT st\_capacity INTO current\_capacity

FROM storages

WHERE st\_id = NEW.ms\_st\_id;

IF current\_capacity < NEW.ms\_quantity THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Not enough capacity for chosen storage';

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`after\_med\_supplies\_insert` $$

USE `arsenal`$$

CREATE TRIGGER after\_med\_supplies\_insert

AFTER INSERT ON med\_supplies

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity - NEW.ms\_quantity

WHERE st\_id = NEW.ms\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_med\_supplies\_update` $$

USE `arsenal`$$

CREATE TRIGGER before\_med\_supplies\_update

BEFORE UPDATE ON med\_supplies

FOR EACH ROW

BEGIN

IF OLD.ms\_st\_id != NEW.ms\_st\_id OR OLD.ms\_quantity != NEW.ms\_quantity THEN

UPDATE storages

SET st\_capacity = st\_capacity + OLD.ms\_quantity

WHERE st\_id = OLD.ms\_st\_id;

UPDATE storages

SET st\_capacity = st\_capacity - NEW.ms\_quantity

WHERE st\_id = NEW.ms\_st\_id;

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_med\_supplies\_delete` $$

USE `arsenal`$$

CREATE TRIGGER before\_med\_supplies\_delete

BEFORE DELETE ON med\_supplies

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity + OLD.ms\_quantity

WHERE st\_id = OLD.ms\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_eng\_equipment\_insert` $$

USE `arsenal`$$

CREATE TRIGGER before\_eng\_equipment\_insert

BEFORE INSERT ON eng\_equipment

FOR EACH ROW

BEGIN

DECLARE current\_capacity INT;

SELECT st\_capacity INTO current\_capacity

FROM storages

WHERE st\_id = NEW.ee\_st\_id;

IF current\_capacity <= 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Not enough capacity for chosen storage';

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`after\_eng\_equipment\_insert` $$

USE `arsenal`$$

CREATE TRIGGER after\_eng\_equipment\_insert

AFTER INSERT ON eng\_equipment

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.ee\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_eng\_equipment\_delete` $$

USE `arsenal`$$

CREATE TRIGGER before\_eng\_equipment\_delete

BEFORE DELETE ON eng\_equipment

FOR EACH ROW

BEGIN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.ee\_st\_id;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_eng\_equipment\_update` $$

USE `arsenal`$$

CREATE TRIGGER before\_eng\_equipment\_update

BEFORE UPDATE ON eng\_equipment

FOR EACH ROW

BEGIN

IF OLD.ee\_st\_id != NEW.ee\_st\_id THEN

UPDATE storages

SET st\_capacity = st\_capacity + 1

WHERE st\_id = OLD.ee\_st\_id;

UPDATE storages

SET st\_capacity = st\_capacity - 1

WHERE st\_id = NEW.ee\_st\_id;

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_m2m\_weapons\_ammunition\_insert` $$

USE `arsenal`$$

CREATE TRIGGER before\_m2m\_weapons\_ammunition\_insert

BEFORE INSERT ON m2m\_weapons\_ammunition

FOR EACH ROW

BEGIN

DECLARE weapon\_caliber VARCHAR(20);

DECLARE ammo\_caliber VARCHAR(20);

SELECT w\_caliber INTO weapon\_caliber

FROM weapons

WHERE w\_id = NEW.wa\_w\_id;

-- Получаем калибр боеприпасов

SELECT am\_caliber INTO ammo\_caliber

FROM ammunition

WHERE am\_id = NEW.wa\_am\_id;

IF weapon\_caliber != ammo\_caliber THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Calibers does not match';

END IF;

END;$$

USE `arsenal`$$

DROP TRIGGER IF EXISTS `arsenal`.`before\_m2m\_orders\_suppliers\_insert` $$

USE `arsenal`$$

CREATE TRIGGER before\_m2m\_orders\_suppliers\_insert

BEFORE INSERT ON m2m\_orders\_suppliers

FOR EACH ROW

BEGIN

DECLARE supplier\_count INT;

SELECT COUNT(\*) INTO supplier\_count

FROM m2m\_orders\_suppliers

WHERE os\_o\_id = NEW.os\_o\_id AND os\_sp\_id = NEW.os\_sp\_id;

IF supplier\_count > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Supplier already connected to order.';

END IF;

END$$

DELIMITER ;

SET SQL\_MODE=@OLD\_SQL\_MODE;

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

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