****

**Частное учреждение профессионального образования**

**«Высшая школа предпринимательства»**

**(ЧУПО «ВШП»)**

КУРСОВОЙ ПРОЕКТ

«Разработка базы данных для частного охранного предприятия»

Выполнил:

студент 3-го курса специальности

09.02.07 «Информационные системы и программирование»

Павлов Илья Витальевич

подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

преподаватель дисциплины,  
преподаватель ЧУПО «ВШП»,  
к.ф.н. Ткачев П.С.

оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

Введение................................................................................................................ 3

Глава 1. Теория о разработке базы данных........................................................ 5

1.1 Анализ предметной области................................................................... 5

1.2 Анализ существующих решений............................................................ 6

1.3 Требования для базы данных.................................................................. 6

1.4 Проектировка схемы базы данных......................................................... 7

1.5 Анализ перспектив базы данных............................................................ 9

1.6 Подбор наиболее подходящей СУБД................................................... 10

Глава 2. Разработка базы данных....................................................................... 13

2.1 Таблицы и типы значений...................................................................... 12

2.2 Хранимые процедуры............................................................................. 17

2.3 Типовые запросы..................................................................................... 21

2.4 Триггеры................................................................................................... 24

2.5 Функции.................................................................................................... 24

2.6 Представление таблиц............................................................................. 25

2.7 Роли........................................................................................................... 25

Заключение............................................................................................................ 27

Список литературы............................................................................................... 28

Приложение 1. Хранимая процедура.................................................................. 29

Приложение 2. Триггер........................................................................................ 30

Приложение 3. Представление таблиц............................................................... 31

Приложение 4. Функции...................................................................................... 31

Приложение 5. Создание ролей........................................................................... 32

Приложение 6. Ссылка на репозиторий.............................................................. 33

Приложение 7. Антиплагиат................................................................................ 34

**Введение**

**Актуальность**

База данных для частного охранного предприятия является ключевым инструментом для эффективного управления информацией о клиентах, сотрудниках, объектах охраны и инцидентах. Она помогает упорядочить данные, обеспечивает быстрый доступ к необходимой информации, улучшает контроль над процессами и способствует обеспечению безопасности. В базе данных для частного охранного предприятия могут содержаться различные типы информации, такие как персональные данные клиентов, контактная информация, данные о контрактах и услугах, информация о расписании службы охраны, данные о системах безопасности на объектах, а также детали о прошлых инцидентах и их расследовании. Использование базы данных позволяет эффективно управлять этой информацией, автоматизировать процессы регистрации, отслеживания и анализа данных, оптимизировать планирование ресурсов и повышать общую эффективность работы предприятия. Важно также учитывать нормативные требования к хранению и защите персональных данных, а также обеспечение безопасности базы данных от несанкционированного доступа. Итак, база данных для частного охранного предприятия является неотъемлемым инструментом, способствующим улучшению оперативности, безопасности и качества предоставляемых услуг.

**Цель**

Целью данной курсовой работы является разработка базы данных для частного охранного предприятия, которая предназначена для автоматизации работы сотрудников с патрулями, инцидентами и клиентами.

**Задачи**

Для достижения цели необходимо правильно составить и решить задачи:

1. Анализ потребностей ЧОП: изучить основные потребности и требования ЧОП.
2. Создать схему базу данных, отражающую основную деятельность ЧОП.
3. Разработать базу данных.
4. Заполнить данными таблицы в базе данных.
5. Проверить базу данных на возможные ошибки, недочеты и логику.
6. Завершить работу.

**Объект исследования**

Объектом исследования является база данных для частного охранного предприятия в среде разработки MySQL Workbench

**Предмет исследования**

Предметом исследования является база данных для частного охранного предприятия в среде разработки MySQL Workbench.

**Метод исследования**

Методом исследования является:

1. Первым шагом будет понимание требований к базе данных. Это включает сбор требований из интернета.
2. Выбор и изучение СУБД — выбор и изучение СУБД для создания базы данных для ЧОП.
3. Проектирование ERD для базы данных — это включает определение таблиц, связей, ключей и типов данных.
4. Изучение и составление логики базы данных — включает взаимодействия сотрудников, клиентов, охраны и т.д.

**Глава 1. Теория о разработке базы данных.**

**1.1 Анализ предметной области**

Для анализа предметной области можно включить такие аспекты:

1. Сотрудники и их данные: Таблица для хранения информации о сотрудниках компании: ФИО, должность, уровень доступа, контактные данные.
2. Данные о клиентом, с которыми может вестись взаимодействие в рамках оказания услуг.
3. Объекты охраны: Хранение информации об охраняемых объектах: адрес, географические координаты, особенности, технические средства безопасности и т.д.
4. События и инциденты, данные об инцидентах, произошедших на охраняемых объектах.
5. Связи между сотрудниками, клиентами и инцидентами.
6. Связи локаций и инцидентов.

Также, поскольку я буду работать в среде MySQL Workbench, на языке программирования SQL.

1. MySQL — одна из наиболее используемых систем управления базами данных. MySQL управляет реляционными базами данных, то есть такими, в которых таблицы связаны между собой.[1]
2. MySQL Workbench — это мощный и удобный инструмент для работы с базами данных MySQL. Он предоставляет широкий спектр функций, которые помогают управлять базами данных, создавать и изменять таблицы, выполнять запросы и многое другое. Более того, он является бесплатным и открытым исходным кодом, что делает его доступным для всех пользователей. [1]

**1.2 Анализ существующих решений**

В процессе разработки базы данных для частной охранной организации могут возникать проблемы и ошибки, которые нужно исправлять.

Чтобы их избежать нужно:

1. Изучить и спроектировать логику базы данных для ЧОП.
2. Использовать MySQL Workbench как инструмент для создания базы данных для ЧОП.
3. Смотреть руководства и исправление ошибок в различных источниках[5].
4. Основательно делать функционал базы данных для ЧОП.

**1.3 Требования для базы данных**

Требования для базы данных включают хранение информации о клиентах, сотрудниках и объектах охраны, обеспечение безопасности и защиты персональных данных, предоставление необходимого функционала, удобства доступа и соответствие стандартам. Используя эти требования, проектирование и завершение проекта можно будет выполнить более эффективно и качественно.

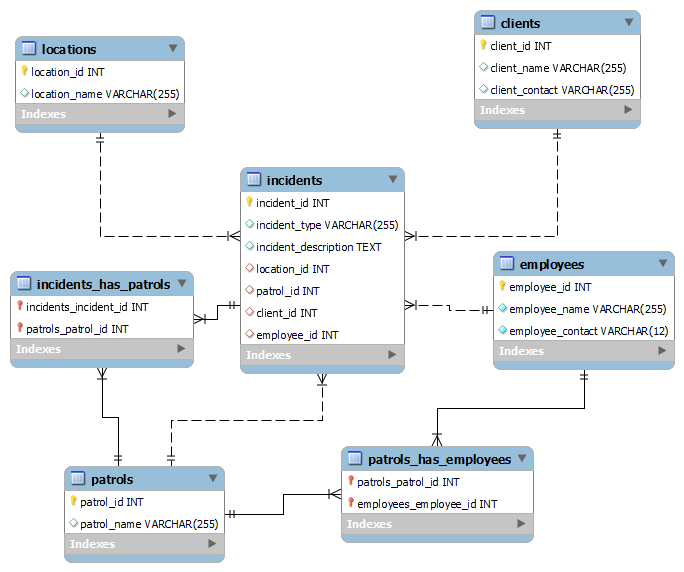
1. Исходя из ранних исследований и суждений, Хранить информацию о клиентах, сотрудниках и объектах охраны.
2. Обеспечить безопасность данных и защиту персональной информации от посторонних лиц.
3. Обеспечить нужным функционалом.
4. Обеспечить возможность удобного доступа и пользования.
5. Соответствовать принятыми требованиями.

Исходя из данных требований спроектировать и завершить проект можно будет быстрее и качественнее. В общем понимании, нужно будет сделать: полезную функцию, полезную процедуру с использованием переменных, условия и т.д., сделать логику связями и типами данных, также добавить роли для будущей безопасности БД в использовании.

**1.4 Проектировка схемы базы данных**

Перед началом создания самой базы данных, я создал ER-Диаграмму[2] для базы данных, составил там основную логику проекта. На данном этапе я спроектировал связи между таблицами и типы данных для них, которые понадобятся для создания самой БД и её логики.

Для составления ERD базы данных я воспользуюсь функцией создания моделей в MySQL Workbench.



**(рис.1)**

В данной ER-Диаграмме(рис. 1) таблицы, которые будут использоваться в базе данных и связи между ними.

В ER-Диаграмме(рис. 1) всего 7 таблиц:

Исходя из предоставленных вами SQL-запросов, я могу сделать вывод о предполагаемой структуре вашей базы данных и связях между таблицами:

1. Таблица "clients": - Содержит информацию о клиентах, включая их имена и контактную информацию.
2. Таблица "employees": - Хранит данные о сотрудниках вашей организации, такие как их имена и контактная информация.
3. Таблица "locations": - Содержит информацию о местоположениях, возможно, где происходят инциденты или расположены объекты охраны.
4. Таблица "patrols": - Информация о патрулях или охранном персонале, возможно, ответственных за наблюдение за определенными местоположениями.
5. Таблица "incidents": - Хранит данные о произошедших инцидентах, включая их тип, описание, связанные местоположения, патрули, клиентов и сотрудников.
6. Таблица "incidents\_has\_patrols": - Таблица соответствия между инцидентами и патрулями, устанавливая связь между конкретными инцидентами и патрулями, ответственными за их расследование или реагирование на них.
7. Таблица "patrols\_has\_employees": - Таблица, устанавливающая связь между патрулями и сотрудниками, указывая, какие сотрудники относятся к каким патрулям.

Эта структура позволяет эффективно управлять данными и связями между ними, обеспечивая легкость отслеживания информации и оперативное реагирование на происшествия. Кроме того, благодаря таблицам, отвечающим за соответствия и связи, система позволяет устанавливать взаимосвязи между различными аспектами работы организации, что является важным для эффективного управления безопасностью и охраной. Благодаря такой структуре данных, организация может также проводить анализ эффективности патрулирования и реагирования на инциденты, оптимизировать распределение персонала и ресурсов, а также улучшать общее понимание потенциальных угроз и паттернов инцидентов. Это позволяет создать более безопасную среду как для клиентов, так и для сотрудников, улучшить реагирование на чрезвычайные ситуации и оптимизировать общую производительность организации в области безопасности.

**1.5 Анализ перспектив базы данных**

База данных системы безопасности и охраны предоставляет широкий спектр функциональности для управления клиентами, сотрудниками, местоположениями и инцидентами.

1. Управление данными клиентов база данных предоставляет возможность добавлять новых клиентов, редактировать их данные, а также осуществлять поиск и фильтрацию информации о клиентах для удобного доступа к необходимым данным.
2. Учет сотрудников и их обязанностей с помощью базы данных можно эффективно управлять персоналом, включая добавление новых сотрудников, назначение им обязанностей, отслеживание рабочего времени и оценку их производительности.
3. Мониторинг местоположений база данных позволяет добавлять новые местоположения, вносить изменения в существующие записи и проводить анализ охраняемых территорий для оптимизации патрулирования.
4. Регистрация и анализ инцидентов в базе данных фиксируются все инциденты, а также их тип, местоположение, причины и последующие действия. Таким образом, система обеспечивает историю и аналитику произошедших событий.
5. Назначение и отслеживание патрулей база данных позволяет назначать патрули на контроль определенных местоположений и инцидентов, а также отслеживать их активности, маршруты и результаты выполненной работы.

К тому же в будущем ее можно будет расширить для более удобного использования и автоматизировать несколько процессов. В нынешнее время не проблема сделать много функций, но главное полезность функций и их значимость в программе, т.к. чем больше структура проекта, тем выше его нестабильность. Анализ перспектив базы данных системы безопасности и охраны указывает на широкий спектр функциональности, предоставляемой данной системой.Возможность управления клиентами, сотрудниками, местоположениями и инцидентами через базу данных обеспечивает удобный доступ к ключевым данным. Эффективное управление данными клиентов, включая добавление, редактирование, поиск и фильтрацию информации, является основополагающим аспектом работы базы данных. Точно также, учет сотрудников и их обязанностей через базу данных повышает уровень контроля и управления персоналом.  
Важной функциональностью является мониторинг местоположений и регистрация инцидентов, что позволяет системе обеспечивать историю и аналитику.  
С учетом быстро развивающихся технологий, базу данных можно будет расширить для более удобного использования и автоматизации процессов. Важно помнить, что полезность функций имеет преимущество перед их количеством, поскольку увеличение структуры проекта может повлечь за собой риск нестабильности. Однако с соблюдением правильной балансировки, база данных системы безопасности и охраны своевременно сможет адаптироваться к изменяющимся потребностям и оставаться полезным инструментом для обеспечения безопасности.

**1.6 Подбор наиболее подходящей СУБД.**

Я выбрал MySQL в качестве СУБД(Приложение 7) для разработки по нескольким причинам. Во-первых, MySQL - это мощная и гибкая открытая СУБД с отличной производительностью, надежностью и масштабируемостью. Ее широкое распространение и активное сообщество разработчиков обеспечивают обширную поддержку, документацию и возможности обучения.  
Во-вторых, MySQL является экономически выгодным выбором, поскольку является открытым программным обеспечением с бесплатной лицензией в большинстве случаев. Это позволяет существенно сократить издержки на лицензии и обслуживание по сравнению с коммерческими СУБД.  
Кроме того, MySQL обладает огромной функциональностью, включая поддержку различных типов данных, хорошую интеграцию с различными языками программирования, возможности репликации данных, а также многое другое. Это позволяет эффективно реализовывать разнообразные проекты с различными требованиями к базе данных.  
В заключение, выбор MySQL основан на сочетании надежности, производительности, экономической выгоды и обширных возможностей настройки, что делает его превосходным выбором для разработки баз данных в различных проектах. Принятие решения в пользу MySQL также обусловлено текущим обучением в колледже на этой программе. Работа с MySQL в рамках проекта разработки базы данных для автосервиса даст дополнительный практический опыт и позволит закрепить ваши знания. Это помогает мне лучше понимать материал и применять его на практике. Проект становится не просто заданием, а возможностью применить полученные знания на практике.

Таким образом, я уверен, что выбор MySQL в качестве СУБД обеспечит не только успешное завершение текущего проекта разработки базы данных, но и будет

являться надежным фундаментом для будущих проектов и карьерного роста.

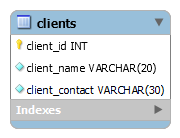
**Глава 2. Разработка базы данных.**

**2.1 Таблицы и типы значений**

В MySQL существуют различные типы данных, такие как целочисленные, с плавающей запятой, символьные, дата/время и многие другие. Выбор подходящего типа данных для каждого столбца в таблице важен для оптимизации хранения информации и обеспечения ее целостности. Например, для хранения целых чисел без знака можно выбрать тип данных UNSIGNED INTEGER, а для хранения текста - тип данных VARCHAR или TEXT. Важно также учитывать размеры и ограничения типов данных при проектировании базы данных, чтобы избежать излишнего расхода ресурсов и обеспечить эффективность операций с данными.

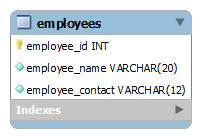
Перейдем к созданию таблиц по схеме (рис.1).

1. Таблица [clients] содержит столбцы: client\_id, client\_name, client\_contact. Для колонки client\_id используется тип данных INT, используются Primary Key, Not Null и Auto Increment. Тип данных INT для хранения идентификатора клиента. Primary Key используется для уникальной идентификации каждой записи в таблице. Auto Increment используется для создания уникального идентификатора для каждой записи в таблице. Not Null – используется потому что client\_id не может быть null. Для client\_name используется тип данных varchar(100) – покрывает строку, где максимальная длина не превышает 100 символов, применяется Not Null. Для client\_contact используется тип данных varchar(36) – позволяет хранить номера в предпочитаемом формате, позволяет хранить несколько любых номеров в международных форматах, применяется Not Null. Готовая таблица указана на рисунке 2.



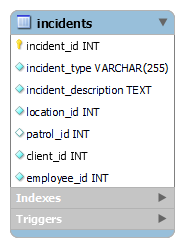
(рис.2)

1. Таблица [employees] содержит столбцы: employee\_id, employee\_name, employee\_contact. Для колонки employee\_id используется тип данных INT, используются Primary Key, Not Null и Auto Increment – потому что employee\_id используется для идентификации записи в таблице. Для employee\_name используется тип данных varchar(100) – покрывает строку, где максимальная длина не превышает 100 символов, применяется Not Null. Для employee\_contact используется тип данных varchar(36) – позволяет хранить номера в предпочитаемом формате, позволяет хранить несколько любых номеров в международных форматах, применяется Not Null. Готовая таблица указана на рисунке 3.



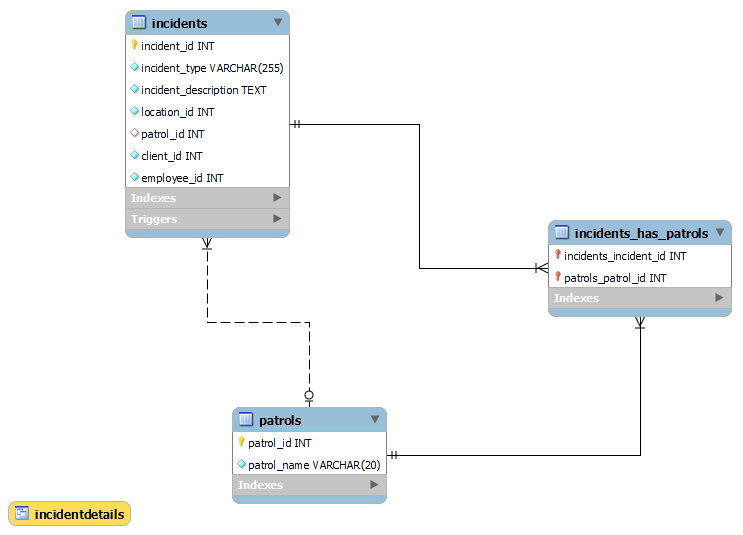
(рис.3)

1. Таблица [incidents] содержит столбцы: incident\_id, incident\_type, incident\_description, location\_id, patrol\_id, client\_id, employee\_id. Для колонки incident\_id используется тип данных INT, используются Primary Key, Not Null и Auto Increment – потому что incident\_id используется для идентификации записи в таблице. Для колонки incident\_type используется тип данных varchar(30) – позволяет хранить тип инцидента для быстрого определения. Для колонки incident\_description используется тип данных TEXT – позволяет хранить текст с большим объёмом данных для точного и полного описания инцидента. Для колонки location\_id используется тип данных INT, используется для связи инцидента с локацией. Использование связи 1:n один ко многим показывает, что на одной локации может быть много инцидентов. Для колонки patrol\_id используется тип данных INT, используется для связи инцидента с патрулем. Использование связи 1:n один ко многим показывает, что у одного патруля может быть несколько участий в инцидентах. Для колонки client\_id используется тип данных INT, используется для связи клиента с инцидентом. Использование связи 1:n один ко многим показывает, что у одного клиента может быть много инцидентов. Для колонки employee\_id используется тип данных INT, используется для связи сотрудника с инцидентом. Использование связи 1:n показывает, что у одного сотрудника может быть несколько инцидентов. Готовая таблица указана на рисунке 4.



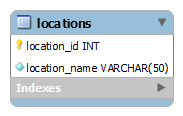
(рис.4)

1. Таблица [incidents\_has\_patrols] содержит столбцы: incidents\_incident\_id, patrols\_patrol\_id, которые имеют типы данных INT и Primary Key, которые также являются Foreign Keys для связей n:m . Используется для связи таблиц incidents и patrols. Используется тип связи n:m – многие ко многим, потому что у одного патруля может быть много завершенных инцидентов, и у одного инцидента может быть много патрулей в участии, но как правило один. Готовая таблица указага на рисунке 5.

(рис.5)

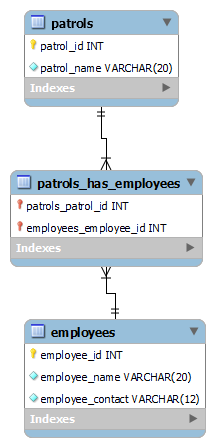
1. Таблица [locations] содержит столбцы: location\_id, location\_address. Для колонки location\_id используется тип данных INT, используются Primary Key, Not Null и Auto Increment – потому что location\_id используется для идентификации записи в таблице. Для location\_address используется тип данных varchar(64) – позволяет хранить полные адреса локации, применяется Not Null.

Готовая таблица указана на рисунке 6.



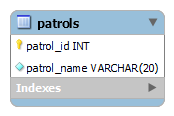
(Рис.6)

1. Таблица [patrols\_has\_employees] содержит столбцы: patrols\_patrol\_id, employees\_employee\_id которые имеют типы данных INT и Primary Key, которые также являются Foreign Keys для связей n:m . Используется для связи таблиц employees и patrols. Используется тип связи n:m – многие ко многим, потому что у одного патруля может быть много сотрудников по работе, но как правило это один и тот же сотрудник, и у одного сотрудника может быть много патрулей в работе.Готовая таблица указана на рисунке 7.



(рис.7)

1. Таблица [patrols] содержит столбцы: patrol\_id, patrol\_name. Для колонки patrol\_id используется тип данных INT, используются Primary Key, Not Null и Auto Increment – потому что patrol\_id используется для идентификации записи в таблице. Для location\_address используется тип данных varchar(30) – позволяет хранить полные имена патрулей, применяется Not Null. Готовая таблица указана на рисунке 8.



(рис.8)

**2.2 Хранимые процедуры**

MySQL 8 предоставляет возможность создания и использования хранимых процедур, которые являются набором инструкций, объединенных в единую единицу. Хранимые процедуры позволяют упростить и структурировать работу с базой данных, а также повысить безопасность и эффективность выполнения операций.

Хранимая процедура MySQL представляет собой подпрограмму, хранящуюся в базе данных. Она содержит имя, список параметров и операторы SQL. Все популярные системы управления базами данных поддерживают хранимые процедуры. Они были введены в MySQL 5.

Существует два вида подпрограмм: хранимые процедуры и функции, возвращающие значения, которые используются в других операторах SQL.

Основное отличие заключается в том, что функции могут использоваться, как любое другое выражение в операторах SQL, а хранимые процедуры должны вызываться с помощью оператора CALL.

Хранимые процедуры MySQL работают быстро. Преимущество сервера MySQL заключается в том, что он использует кэширование, а также заранее заданные операторы. Основной прирост скорости дает сокращение сетевого трафика. Если есть повторяющиеся задачи, которые требуют проверки, обработки циклов, нескольких операторов, и при этом не требуют взаимодействия с пользователем, это можно реализовать с помощью одного вызова процедуры, которая хранится на сервере.

Исходный код хранимых процедур всегда доступен в базе данных. Это эффективная практика связать данные с процессами, которые их обрабатывают.

Условия – в MySQL условия используются для фильтрации данных в запросах. Они позволяют выбирать определенные строки или группы строк, сортировать данные и применять условия к результатам. Например, с помощью условий можно составить запросы, которые выберут данные только из определенного диапазона значений, с учетом определенных условий, или выполнят операции над данными в зависимости от их свойств.[4]

Обработчик исключений в MySQL 8 способны генерировать специальные ошибки, которые могут указать на каком этапе была допущена ошибка.

Транзакции - это механизм, который позволяет группировать несколько операций базы данных в одну логическую единицу работы. Они обеспечивают целостность данных и поддерживают принципы ACID.

ACID - это аббревиатура, которая описывает четыре основных принципа транзакций:

Атомарность: Транзакция считается атомарной, если все ее операции выполняются либо все успешно, либо ни одна из них не выполняется. Если хотя бы одна операция транзакции не может быть выполнена, то все изменения отменяются.

Согласованность: Транзакция должна приводить базу данных из одного согласованного состояния в другое согласованное состояние. Это означает, что все ограничения целостности должны быть соблюдены во время выполнения транзакции.

Изолированность: Каждая транзакция должна быть выполнена изолированно от других транзакций. Это означает, что результаты одной транзакции не должны быть видны другим транзакциям до тех пор, пока первая транзакция не будет завершена.

Долговечность: Результаты успешно завершенной транзакции должны быть постоянными и доступными даже в случае сбоя системы. Это достигается путем записи изменений в постоянное хранилище, такое как жесткий диск.[3]

MySQL 8 обеспечивает поддержку всех этих принципов для транзакций. Он предоставляет различные уровни изоляции, которые позволяют контролировать видимость изменений другим транзакциям. Кроме того, MySQL 8 включает механизмы для обработки сбоев и восстановления данных, чтобы гарантировать долговечность транзакций.

Использование транзакций в MySQL 8 позволяет обеспечить надежность и целостность данных в приложениях, особенно в ситуациях, когда несколько операций должны быть выполнены атомарно. Это делает их полезными в различных сценариях, таких как финансовые операции, онлайн-покупки или любые другие операции, требующие точности и надежности.

Локальные переменные в хранимых процедурах способны держать в себе значение, проводить с этим значением различные операции.

Хранимая процедура используемая в моей базе данных – эта процедура предназначена для проверки того, что хотя бы один патруль был назначен на какой-либо инцидент. Если условие не выполняется (т.е., если нет связи между патрулями и инцидентами), процедура вызывает исключение и откатывает транзакцию, чтобы сохранить целостность данных. В противном случае она подтверждает успешное выполнение и выводит сообщение о наличии хотя бы одного патруля на инциденте.

DELIMITER //

CREATE DEFINER=`root`@`localhost` PROCEDURE `CheckPatrolsSentToIncidents`()

BEGIN

DECLARE patrol\_count INT;

DECLARE error\_message TEXT;

...

В этом блоке кода создается процедура и объявляются переменные и тип данных к ним

...

DECLARE CONTINUE HANDLER FOR NOT FOUND

BEGIN

SET error\_message = 'Как минимум один патруль должен быть отправлен на инцидент';

ROLLBACK;

SELECT error\_message;

END;

...

«DECLARE CONTINUE HANDLER FOR NOT FOUND»: обработчик, который срабатывает, если не найдено ни одной строки при выполнении запроса в блоке «BEGIN».  
- «BEGIN ... END;»: В этом блоке кода устанавливается сообщение об ошибке, выполняется откат транзакции «ROLLBACK», и затем выводится сообщение об ошибке.

...

START TRANSACTION;

SELECT COUNT(\*) INTO patrol\_count

FROM incidents\_has\_patrols;

...

«START TRANSACTION;»: начало транзакции перед выполнением запросов к базе данных. Затем был выполнен запрос для подсчета количества записей в таблице “incidents\_has\_patrols”. Результат этого запроса был сохранен в переменную “patrol\_count”. Это позволяет хранить количество связей между инцидентами и патрулями для дальнейшего использования в программе.

...

IF patrol\_count < 1 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Как минимум один патруль должен быть отправлен на инцидент';

END IF;

...

Этот фрагмент кода выполняет проверку значения переменной patrol\_count. Если значение этой переменной меньше 1, то генерируется ошибка с текстом “Как минимум один патруль должен быть отправлен на инцидент”. Данная конструкция позволяет обеспечить контроль за значением переменной и принимать соответствующие действия в зависимости от условия.

...

COMMIT;

SELECT 'Хотя бы один патруль отправлен на инцидент.';

END //

DELIMITER ;

...

Этот фрагмент кода представляет завершение SQL-транзакции с использованием команды «COMMIT;», что фиксирует изменения в базе данных. После этого идет SQL-запрос, возвращающий текст "Хотя бы один патруль отправлен на инцидент.", затем встречается «END;;», что указывает на завершение процедуры. Полный код всей хранимой процедуры см. Приложение 1.

**2.3 Типовые запросы**

Типовые запросы к базе данных необходимы для выполнения различных операций с данными, таких как извлечение информации, обновление, удаление или добавление новых данных. Они позволяют взаимодействовать с базой данных и получать нужную информацию. Основные цели типовых запросов:

1. Извлечение данных: Получение информации из базы данных в соответствии с определенными критериями для отображения, анализа или использования в приложениях.
2. Обновление данных: Внесение изменений в существующие записи базы данных, такие как обновление информации о клиентах или товарах.
3. Удаление данных: Удаление устаревших или ненужных записей, чтобы оптимизировать хранилище данных.
4. Создание отчётов: Формирование сводных отчётов, статистики или аналитики на основе данных из базы для анализа бизнес-процессов и принятия управленческих решений.

Типовые запросы в базах данных играют важную роль в обеспечении эффективной работы с данными. Они позволяют извлекать, обновлять, удалять данные и управлять структурой базы данных.

Первый типовой запрос – получение списка всех клиентов.

SELECT \* FROM clients;

Этот SQL запрос выполняет выборку всех столбцов из таблицы "clients" в базе данных.

Второй типовой запрос – получение информации об инцидентах с описанием их расположения.

SELECT i.incident\_type, i.incident\_description, l.location\_name

FROM incidents i

JOIN locations l ON i.location\_id = l.location\_id;

Этот SQL запрос извлекает данные из двух таблиц: "incidents" и "locations". Он выбирает три столбца: "incident\_type" из таблицы "incidents", "incident\_description" также из таблицы "incidents" и "location\_name" из таблицы "locations". Затем запрос использует оператор JOIN для объединения этих двух таблиц на основе условия, что значение столбца "location\_id" в таблице "incidents" равно значению столбца "location\_id" в таблице "locations". Это позволяет объединить информацию об инцидентах с информацией о местоположениях, связанных с этими инцидентами.

Третий типовой запрос – получение информации о сотрудниках и патрулях, с которыми они связаны.

SELECT e.employee\_name, p.patrol\_name

FROM employees e

JOIN patrols\_has\_employees pe ON e.employee\_id = pe.employees\_employee\_id

JOIN patrols p ON pe.patrols\_patrol\_id = p.patrol\_id;

Этот SQL запрос извлекает данные из трех таблиц: "employees", "patrols\_has\_employees" и "patrols". Он соединяет их, используя идентификаторы сотрудников, патрулей и идентификаторы принадлежности сотрудников к патрулям. В результате запроса вы получите имена сотрудников и названия патрулей, к которым они относятся.

Четвёртый типовой запрос – подсчет количества инцидентов, связанных с каждым клиентом.

SELECT c.client\_name, COUNT(\*) AS total\_incidents

FROM incidents i

JOIN clients c ON i.client\_id = c.client\_id

GROUP BY c.client\_name;

Этот SQL запрос извлекает данные из двух таблиц: "incidents" и "clients". Он выбирает два столбца: "client\_name" из таблицы "clients" и COUNT(\*) как "total\_incidents". COUNT - это функция агрегирования, которая подсчитывает количество строк, которые удовлетворяют условиям запроса. Далее запрос использует оператор JOIN для объединения таблиц. Он объединяет "incidents" и "clients" на основе условия, что значение столбца "client\_id" в таблице "incidents" равно значению столбца "client\_id" в таблице "clients". Затем, используя оператор GROUP BY, результат запроса группируется по столбцу "client\_name". Это означает, что для каждого уникального значения "client\_name" будет вычислено количество инцидентов, связанных с данным клиентом.

Пятый типовой запрос – Получение списка инцидентов, где тип инцидента – "Vandalism".

SELECT \*

FROM incidents

WHERE incident\_type = 'Vandalism';

Этот SQL запрос означает "выбрать все столбцы из таблицы 'incidents', где тип инцидента равен 'Вандализм'". Используя оператор SELECT \*, мы запрашиваем все столбцы из таблицы 'incidents'. Затем оператор WHERE используется для фильтрации результатов, где только строки с типом инцидента "Vandalism" будут выбраны.

**2.4 Триггеры**

Триггеры в MySQL 8 представляют собой специальные объекты базы данных, которые автоматически выполняются при определенных событиях или действиях с данными. Они позволяют программистам определить пользовательские действия, которые должны быть выполнены перед или после выполнения операции в базе данных. Иными словами, это определяемая пользователем SQL-команда, которая автоматически вызывается во время операций INSERT, DELETE или UPDATE. Код триггера связан с таблицей и уничтожается после удаления таблицы. В моей базе данных используется один триггер с названием AssignPatrolToIncident(Приложение 2)

Этот триггер будет запускаться после вставки новой записи в таблицу incidents. Он найдет первый доступный патруль, который еще не назначен на другой инцидент, и автоматически назначит его на новый инцидент.

**2.5 Функции**

Функции в MySQL представляют собой блоки кода, которые могут принимать параметры, выполнять определенные операции и возвращать результат. Они предоставляют удобный способ для организации повторно используемых операций, выполнения вычислений и обработки данных прямо в базе данных.[6]

В моей базе данных используется одна функция. При вызове функции с названием CountSentPatrols(Приложение 4) она выполнит запрос, чтобы подсчитать количество различных патрулей, зарегистрированных в связи с инцидентами, и вернет этот результат. Код см приложение.

**2.6 Представление таблиц**

Представление таблиц в MySQL используются для создания виртуальных "таблиц" на основе результатов выполнения запросов к одной или более реальным таблицам. Они позволяют сократить повторяющиеся запросы, облегчают доступ к данным и упрощают сложные запросы. Я в свою базу данных добавил одно представление. Это представление с названием IncidentDetails(Приложение 3) создает виртуальную таблицу, которая содержит подробные сведения о происшествиях, включая ключевую информацию о клиенте, сотруднике, местоположении и патруле. Оно объединяет данные из нескольких таблиц incidents, clients, employees, locations и patrols.

Представление IncidentDetails содержит следующие поля данных о происшествиях:

1. incident\_id – идентификатор происшествия;
2. incident\_type – тип происшествия;
3. incident\_description – описание происшествия;
4. client\_name – имя клиента, связанного с происшествием;
5. employee\_name – имя сотрудника, связанного с происшествием;
6. location\_name – название местоположения, связанного с происшествием;
7. patrol\_name – название патруля, связанного с происшествием;

**2.7 Роли**

В MySQL роли используются для упрощения управления правами доступа к базе данных. Вместо того чтобы настраивать права для каждого пользователя отдельно, можно создать роли, которые включают нужные права, а затем назначить эти роли пользователям. Таким образом, можно легко управлять доступом и избежать повторения одних и тех же настроек для разных пользователей. Это также помогает в обеспечении безопасности базы данных, потому что можно точно контролировать, кто имеет доступ к каким данным.

В моей базе данных есть две отдельные роли:

1. Администратор(“admin”) – обладает всеми правами. Это дает полный контроль над данными и структурой этой базы данных.
2. Пользователь(“user”) – ограничен правами на SELECT, INSERT и UPDATE в базе данных.

Таким образом , можно не беспокоится о безопасности и работоспособности будущих пользователей. Код для создания ролей см. Приложение 5.

**Заключение**

В заключение, создание базы данных для частного охранного предприятия представляет собой важный шаг к обеспечению нужной для работы эргономикой и удобством. Разработанная база данных предоставляет эффективный инструмент для управления информацией о сотрудниках, клиентах, объектах охраны, событиях и инцидентах, что способствует оптимизации оперативной работы и повышению качества предоставляемых работы. Адекватное проектирование базы данных позволяет эффективно управлять данными, обеспечивает точность и целостность информации, а также улучшает доступность данных для пользователей. Продуманные связи между таблицами позволяют строить сложные запросы для получения ценной информации, необходимой для принятия эффективных решений. Кроме того, база данных обладает потенциалом для дальнейшего развития базы данных для частного охранного предприятия, что позволит улучшить её функциональность и эффективность.

**Список литературы**

1. СУБД MySQL. MySQL Workbench[Электронный ресурс]

Режим доступа:

<https://it.vshp.online/#/pages/op08/op08_03_lab>

2. Entity relationship diagram (ERD)[Электронный ресурс]

Режим доступа:

<https://www.techtarget.com/searchdatamanagement/definition/entity-relationship-diagram-ERD>

3. Введение в транзакции[Электронный ресурс]

Режим доступа:

<https://it.vshp.online/#/pages/mdk1101/mdk1101_lab_15>

4. Условия в SQL[Электронный ресурс]

Режим доступа:

<https://sky.pro/wiki/sql/usloviya-v-sql-pravilnoe-ispolzovanie-if-begin-end/>

5. Руководства и помощь при ошибке[Электронный ресурс]

Режим доступа:

<https://dev.mysql.com/doc/refman/8.0/en/help.html>

6. Введение в функции[Электронный ресурс]

Режим доступа:

<https://oracleplsql.ru/functions-mysql.html>

7. Система управления базами данных[Электронный ресурс]

Режим доступа:

https://ru.wikipedia.org/wiki/Система\_управления\_базами\_данных

6

**Приложение 1. Хранимая процедура**

DELIMITER //  
  
CREATE PROCEDURE CheckPatrolsSentToIncidents()  
BEGIN  
DECLARE patrol\_count INT;  
DECLARE error\_message TEXT;  
  
DECLARE CONTINUE HANDLER FOR NOT FOUND  
BEGIN  
SET error\_message = 'Как минимум один патруль должен быть отправлен на инцидент';  
ROLLBACK;  
SELECT error\_message;  
END;  
  
START TRANSACTION;  
  
SELECT COUNT(\*) INTO patrol\_count  
FROM incidents\_has\_patrols;  
  
IF patrol\_count < 1 THEN  
SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Как минимум один патруль должен быть отправлен на инцидент';  
END IF;  
  
COMMIT;  
  
SELECT 'Хотя бы один патруль отправлен на инцидент.';  
END//  
  
DELIMITER ;

**Приложение 2. Триггер**

DELIMITER //  
  
CREATE TRIGGER AssignPatrolToIncident  
AFTER INSERT ON incidents  
FOR EACH ROW  
BEGIN  
DECLARE available\_patrol\_id INT;  
  
-- Находим доступный патруль, который еще не назначен на другой инцидент  
SELECT patrols.patrol\_id INTO available\_patrol\_id  
FROM patrols  
WHERE patrols.patrol\_id NOT IN (  
SELECT patrols\_patrol\_id  
FROM incidents\_has\_patrols  
)  
LIMIT 1;  
  
-- Назначаем найденный патруль на текущий инцидент  
INSERT INTO incidents\_has\_patrols (incidents\_incident\_id, patrols\_patrol\_id)  
VALUES (NEW.incident\_id, available\_patrol\_id);  
END//  
  
DELIMITER ;

**Приложение 3. Представление таблиц**

CREATE VIEW IncidentDetails AS  
SELECT  
i.incident\_id,  
i.incident\_type,  
i.incident\_description,  
c.client\_name,  
e.employee\_name,  
l.location\_name,  
p.patrol\_name  
FROM incidents i  
JOIN clients c ON i.client\_id = c.client\_id  
JOIN employees e ON i.employee\_id = e.employee\_id  
JOIN locations l ON i.location\_id = l.location\_id  
JOIN patrols p ON i.patrol\_id = p.patrol\_id;

**Приложение 4. Функция**

DELIMITER //  
  
CREATE FUNCTION CountSentPatrols()  
RETURNS INT  
DETERMINISTIC  
BEGIN  
DECLARE sent\_patrols\_count INT;  
  
SELECT COUNT(DISTINCT patrols\_patrol\_id) INTO sent\_patrols\_count  
FROM incidents\_has\_patrols;  
  
RETURN sent\_patrols\_count**;**  
END//  
  
DELIMITER ;

**Приложение 5. Создание ролей**

-- Создание ролей  
CREATE ROLE admin;  
CREATE ROLE user;  
  
-- Назначение прав для администратора  
GRANT ALL PRIVILEGES ON security\_system.\* TO admin;  
  
-- Назначение ограниченных прав для пользователя  
GRANT SELECT, INSERT, UPDATE ON security\_system.\* TO user;  
  
-- Создание пользователей и назначение ролей  
CREATE USER 'admin\_user'@'localhost' IDENTIFIED BY 'password';  
GRANT admin TO 'admin\_user'@'localhost';  
  
CREATE USER 'regular\_user'@'localhost' IDENTIFIED BY 'password';  
GRANT user TO 'regular\_user'@'localhost'**;**

**Приложение 6. Ссылка на репозиторий**

с этой ролю смогут выполнять только эти три операции в указанной базе данных, что ог213р3213213ан

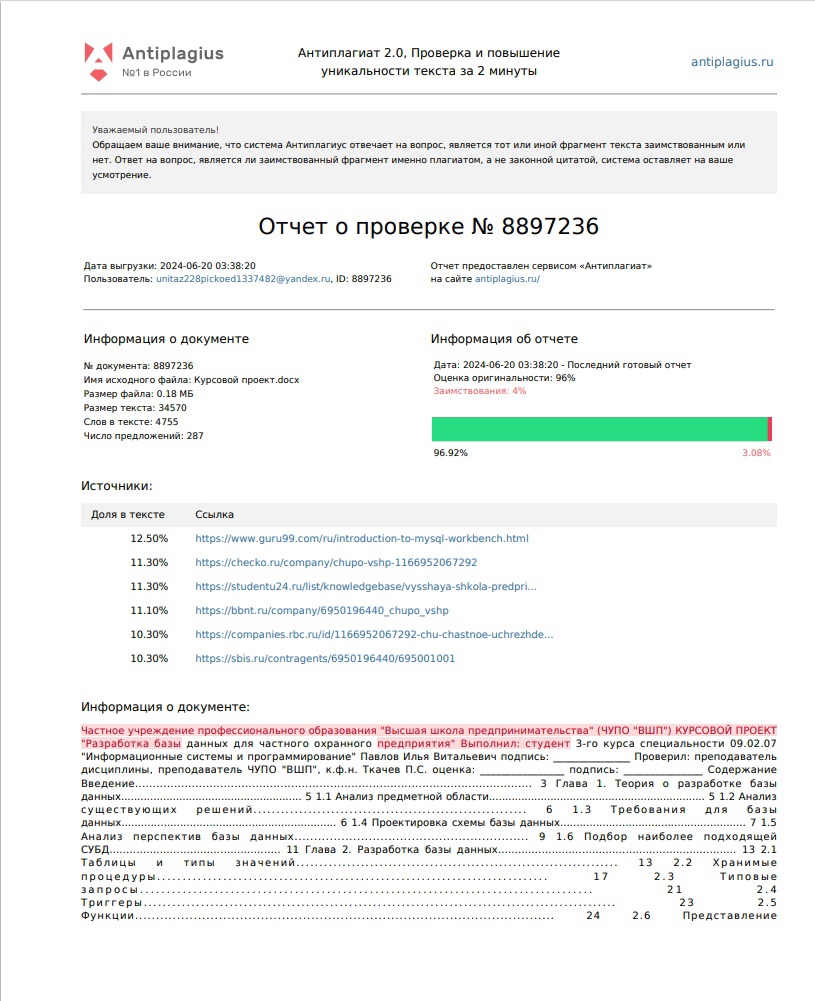
Цу313

Ссылка для ручного ввода:

|  |
| --- |
| https://github.com/obdrystish/chop.git |

**Приложение 7.** Антиплагиат

21



их доступ и возможности в сравнении с администратором