

**Частное учреждение профессионального образования**

**«Высшая школа предпринимательства»**

**(ЧУПО «ВШП»)**

КУРСОВОЙ ПРОЕКТ

«Разработка базы данных для ТСЖ»

Выполнил:

студент 3-го курса специальности

09.02.07 «Информационные системы и программирование»  
Жуков Кирилл Николаевич

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

Проверил:

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

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

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

# **Введение**

**Актуальность**

В настоящее время информационные технологии играют ключевую роль в управлении и оптимизации процессов, в том числе управление недвижимостью чем в свою очередь занимается ТСЖ.

Что такое ТСЖ? Товарищество Собственников Жилья (ТСЖ) - это некоммерческая организация, созданная собственниками помещений в многоквартирном доме для обслуживания своего дома. Основная цель ТСЖ - эффективное и качественное содержание общего имущества в доме, включая проведение ремонта, уборки территорий, поддержка работоспособности лифтов и многое другое.   
[https://ru.wikipedia.org/wiki/Товарищество\_собственников\_жилья]

Разработка базы данных для ТСЖ актуальна по трем причинам. Во-первых, это обеспечение минимизации человеческого фактора в ошибках, учете и расчётах. Во-вторых, это прозрачность и доступность информации для участников управления ТСЖ. В-третьих, это улучшение качества и эффективности управления сразу несколькими объектами.

**Цель**

Целью данной курсовой работы является разработка базы данных для ТСЖ, которая предназначена для автоматизации расчётов и учетов в товариществе собственников жилья.

**Задачи**

Для достижения цели необходимо решить несколько задач:

1. Изучить существующие решения и проанализировать потребность ТСЖ в информационных системах
2. Спроектировать схему базы данных, отражающую основную деятельность ТСЖ
3. Разработать базу данных
4. Заполнить базу данных
5. Проверить базу на недочеты и ошибки
6. Разработка рекомендаций по внедрению и использованию базы данных

**Объект исследования**

Объектом является база данных

**Предмет исследования**

Предметом исследования является база данных для товарищества собственников жилья

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

**1.1 Анализ предметной области**

Базы данных можно разделить на несколько основных типов. Вот основные виды баз данных:

1. **Реляционные базы данных.**   
   Такие базы данных представляют собой базы данных, которые используются для хранения и предоставления доступа к взаимосвязанным элементам информации. Реляционные базы данных основаны на реляционной модели — интуитивно понятном, наглядном табличном способе представления данных. Каждая строка, содержащая в таблице такой базы данных, представляет собой запись с уникальным идентификатором, который называют ключом. Столбцы таблицы имеют атрибуты данных, а каждая запись обычно содержит значение для каждого атрибута, что дает возможность легко устанавливать взаимосвязь между элементами данных. [<https://www.oracle.com/cis/database/what-is-a-relational-database/>]
2. **Нереляционные базы данных.**  
   Данные в них хранятся в формате, отличном от реляционных таблиц. Однако к нереляционным базам данным можно обращаться с помощью идиоматических языковых API, языков декларативно структурированных запросов и языков запросов по примеру, поэтому их также называют базами данных «не только SQL».  
   [<https://www.oracle.com/cis/database/nosql/what-is-nosql/>]
3. **Иерархические базы данных**  
   Здесь между хранимыми объектами устанавливаются связи. Объекты делятся на родителей (основные классы или категории объектов) и потомков (экземпляры этих классов или категорий). При этом у каждого потомка может быть не более одного родителя.  
   [<https://selectel.ru/blog/databases-types/>]
4. **Объектно-ориентированные базы данных**  
   базы данных, в которых информация представлена в виде объектов, как в объектно-ориентированных языках программирования.  
   [<https://habr.com/ru/articles/56399/>]

Существует также не мало СУБД, но можно выделить лишь несколько интересных:

1. **MySQL -** свободно распространяемая серверная реляционная СУБД. В настоящее время является наиболее популярной в веб-программировании. [«PHP и MySQL. 25 уроков для начинающих» — В. Дронов / стр194 - 13.2 ]
2. **PostgreSQL(postgres) -** объектно-реляционная система управления базами данных с открытым исходным кодом. На первом месте в ней стоит расширяемость, техническое совершенство и совместимость.  
   [«Изучаем PostgreSQL 10» — С. Джуба, А. Волков стр 46]
3. **MongoDB -** это ориентированная на документы база данных NoSQL с открытым исходным кодом, которая использует для хранения структуру JSON. Модель данных MongoDB позволяет представлять иерархические отношения, проще хранить массивы и другие более сложные структуры. [https://yandex.cloud/ru/blog/posts/2020/12/managed-mongodb-overview]
4. **SQLite -** это встраиваемая система управления базами данных для клиент-серверных приложений. Эта СУБД отличается компактностью — она занимает один файл, благодаря чему встраивается в приложения, чтобы выполнять функции СУБД автономно, без использования сервера. Также SQLite применяется в веб-разработке небольших проектов. [https://workspace.ru/tools/database/sqllite/]

**1.2 Анализ существующих решений**

Во время разработки и внедрения системы для Товарищества Собственников Жилья (ТСЖ) возникают различные проблемы, которые требуют внимания и анализа. Решение этих проблем способствует эффективности решений и оптимизации управления имуществом. На первый взгляд можно выделить несколько основных проблем:

1. **Безопасность данных.** Разработка базы данных подразумевает работу с большим объемом личной информации о собственниках. Обеспечение надежной защиты данных от несанкционированного доступа является одной из проблем.
2. **Масштабируемость системы.** Необходимо заранее предусмотреть возможность масштабирования системы для обеспечения ее эффективной работы с увеличением объема данных и количества пользователей.
3. **Учет законодательных требований.** Система должна соответствовать всем действующим законодательным и нормативным требованиям, включая правила обработки и защиты данных

Уже существуют готовые решения проблем. Вот несколько из них:

1. 1С:Учет в управляющих компаниях ЖКХ, ТСЖ и ЖСК — популярное решение, которое предлагает широкий спектр функциональных возможностей для автоматизации процессов учета и управления в ТСЖ. Система позволяет вести учет финансов, контролировать платежи жильцов, планировать бюджет и отслеживать выполнение ремонтных работ.   
   [https://solutions.1c.ru/catalog/jkh-tsj/features]
2. Домовой — это еще одна система, которая ориентирована на упрощение взаимодействия между управляющими компаниями и жителями многоквартирных домов. Предлагает инструменты для онлайн-голосований, обсуждений и опросов среди жильцов, а также функции для учета и контроля платежей.   
   [https://infinnity.solutions/project/upravdom]
3. ЖКХ Контроль — сервис, акцентирующий внимание на прозрачности и контроле расходов на жилищно-коммунальные услуги. Предоставляет жильцам детальную отчетность по использованным ресурсам и выполненным работам.   
   [http://gkhkontrol.ru/]

При анализе существующих решений можно выявить у этих программ ряд недостатков:

1. Многие системы имеют заранее готовые набор функций, которые не всегда полностью соответствуют специфике работы ТСЖ
2. Некоторые программные продукты требуют значительных усилий для внедрения и обучения персонала, что может стать препятствием для маленьких ТСЖ
3. Цена на лицензии и обслуживание программного обеспечения может быть достаточно высока, что делает его недоступным для ТСЖ с ограниченным бюджетом.

**1.3 Требования для базы данных**

Исходя из ранее исследованных типов баз данных, я выберу реляционные базы данных. Они достаточно популярные, подойдут для создания таблиц и связей между сущностями.

В базе данных будет храниться информация о домах, квартирах, счетах и жильцах. Одной из важных особенностей является масштабируемость базы. Так как внутри ТСЖ может быть не один дом, а несколько поэтому база данных должна быть подготовлена для этого

База данных должна отображать основные характеристики дома, квартиры, жильца. У каждого дома есть общие данные – это количество этажей, количество подъездов, количество квартир и год постройки дома. Данные связанные с наличием подвала, площади целого дома не будут учитываться.

Удобное отслеживание и управление домом – это один из главных критериев для разработки базы данных. Такая база данных рассчитана на использование ее персоналом жилого фонда. Применение базы данных позволит повысить эффективность работы персонала по эксплуатации жилого фонда, снизить затраты времени на получение и обработку данных по жилому фонду и повысить эффективность работы. Правильная организация работы персонала обеспечивает точные и достоверные данные которые способствуют снижению затрат на содержание жилого фонда, точные расчеты материалов, время и стоимость работ по содержанию жилищного фонда.

Так же необходимо создать роли для управления этой базой данных. Какие-то пользователи смогут управлять, а какие-то только просматривать. Без создания ролей любой пользователь сможет не санкционированно вносить изменения в базу данных, что не сходится с одной из основных проблем. Для того чтобы исключить возможность упустить какое-то действие персонала потребуется отдельная таблица для отслеживания действий.

**1.4 Проектировка схемы базы данных**

Перед началом создания самой базы данных мы построим схему, отражающая «сущность-связь». Такую схему называют ER-диаграмма или ERD. ER-диаграмма - это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы [https://it.vshp.online/#/pages/op08/op08\_10\_lec?id=Что-такое-er-модель-и-er-диаграмма]

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

**Рис. 1**

В данной схеме (рис. 1) готовая спроектированная база данных на основе требований, составленных ранее.

В схеме находится 10 таблиц, каждая из которых реализует или помогает реализовать поставленную задачу:

1. buildings [здания] – таблица которая содержит в себе здания, которые принадлежат ТСЖ.
2. apartments [квартиры] – таблица которая содержит в себе квартиры, которые находятся в здании.
3. storerooms [кладовки] – таблица которая содержит в себе данные про кладовку.
4. payments [платежи] – таблица которая содержит в себе данные про выставленные платежи.
5. residents [жильцы] – таблица которая содержит в себе жильцов, которые живут в определённой квартире.
6. car [машина] – таблица которая содержит в себе информацию про автомобили.
7. carspots [парковочные места] – таблица которая содержит в себе информацию про парковочные места.
8. logs [логи/аудит] – таблица которая содержит в себе информацию про действия пользователей.

Такой набор таблиц необходим для покрытия всех требований базы данных для ТСЖ.

**Глава 2. Разработка базы данных.**

**1.1 Таблицы и типы значений**

В каждой таблице есть столбцы, которые имеют тип данных. В MySQL существует не мало типов данных, поэтому следует думать наперед – какой тип выбирать. Каждый тип данных занимает разное количество памяти. Для примера возьмем INT и TINYINT. Числовой тип INT предоставляет целые числа от -2147483648 до 2147483647, занимает 4 байта, а TINYINT представляет целые числа от -128 до 127, занимает 1 байт. [https://metanit.com/sql/mysql/2.3.php] Исходя из этого таблица с изначально непродуманным типом данных может весить в 3 и более раз больше, по сравнению с правильным выбором типа.

Перейдем к созданию таблиц по схеме (рис.1) которую мы разработали.

1. Таблица buildings [здания] содержит столбцы: id, address, number, floors, aparts, building\_year, building\_enters.   
   1. Для колонки id используем тип данных INT, используем Primary Key, так же Not Null и Auto Increment. Тип данных INT используется чтобы в будущем не было упора в диапазон другого типа данных. Primary Key используется для как уникальный идентификатор каждой записи в таблице, так же для корректной работы включаем Auto Increment – он самостоятельно присваивает порядковый номер каждой записи. Not null – используется потому что id не может быть null   
   2. Для address используем тип данных VARCHAR(265) – он сможет покрыть длину символов вплоть до 265. Так же применяем Not Null   
   3. Для number используем CHAR(15) – выбор такого типа данных обеспечивает работу с номерами домов которые имеют номер «2/4» или «д.6 к.2». Если использовать INT то будут проблемы с домами и не все дома смогут корректно состоять в базе данных  
   4. Для floors используем тип данных TINYINT Unsigned– для уменьшения размера базы данных в будущем, при большом количестве записей. Тип TINYINT покроет целые числа от 0 до 255 занимая минимально места. Unsigned указывает что могут быть только положительные значения поля  
   5. Для aparts используем SMALLINT UNSIGNED – такой тип имеет больше чем 255, ведь существуют дома с количеством квартир более 255. Исходя из этого следующий по размеру будет SMALLINT – он предоставляет до 65535 при использовании UNSIGNED   
   6. Для building\_year используем тип DATE – нам нужно знать дату постройки и ввода в эксплуатацию дома. Тип DATE хранит даты с 1 января 1000 года до 31 декабря 9999 года по умолчанию используется формат yyyy-mm-dd [https://metanit.com/sql/mysql/2.3.php]   
   7. Для building\_enters используем TINYINT UNSIGNED – маловероятно что будет здание с 256 подъездами, поэтому SMALLINT будет достаточно.  
     
   Так же все поля имеют Not Null – в доме не может быть значение null в любом из полей

  
**Рис. 2**

Так должна выглядеть готовая таблица buildings (рис. 2)

1. Таблица apartments [квартиры] содержит столбцы: id, buildings\_id, rooms, area.  
   1. Для колонки id используется Primary Key, Not Null, Auto Increment – так как ID используется для идентификации записи  
   2. buildings\_id требуется для создании связи с таблицей buildings. Связь один ко многим (1:n), где buildings это 1, а apartments n.   
   3. Для rooms используем SMALLINT UNSIGNED чтобы не занимать много места в будущем, в квартире не может быть больше чем 255 комнат.  
   4. Для area используем FLOAT так как площадь квартиры может быть не целой, а с плавающей точкой  
      
   Все поля в таблице имеют значение Not Null



**Рис. 3**

Так выглядит готовая таблица apartments (рис.3)

1. Таблица storerooms [кладовки] содержит столбцы: id, apartments\_id, area, number.  
   1. Для id – используется INT, Primary Key, Auto Increment – поскольку id используется для идентификации записи в таблице  
   2.apartments\_id тип данных INT. Используется для связи с apartments. Использование связи 1:1 один к одному показывает что у 1 квартиры может быть 1 кладовка либо не быть ее вообще.  
   3. Для area используется FLOAT так как площадь имеет плавающую запятую  
   4. Для number используется INT поскольку номера кладовок могут быть больше 255 и иметь любой набор цифр.



**Рис. 4**

Так выглядит готовая таблица storerooms (рис. 4)

1. Таблица payments [платежи] имеет столбцы id, apartments\_id, date, amount, type.  
   1. Для id – используется INT, Primary Key, Auto Increment – поскольку id используется для идентификации записи в таблице  
   2. apartments\_id тип данных INT. Используется для связи с apartments. Использование связи 1:n один ко многим показывает что у 1 квартиры может быть большое количество платежей.  
   3. Для date используется DATETIME такой тип сможет сохранять не только дату платежа, но и время этого платежа. Такой тип сможет по запросу более конкретно выводить платежи, которые поступали и в случае ошибки поможет быстрее определить неверный платеж.  
   4. Для amount используется DECIMAL(10,2) такой тип хранит в себе 8 цифр до запятой и 2 после. Я думаю что это лучший тип данных для работы со значениями, которые связаны с оплатой.  
   5. Для type используем ENUM('свет', 'газ', 'кап.ремонт'). Такой тип гарантирует что не появится лишних типов оплаты. Доступны только 3 - свет, газ, кап. ремонт.



**Рис. 5**

Так выглядит готовая таблица payments (рис. 5)

1. Таблица residents [жильцы] имеет столбцы id, full\_name, birth\_date, registration\_date, phone\_number.  
   1. Для id – используется INT, Primary Key, Auto Increment – поскольку id используется для идентификации записи в таблице.  
   2. Для full\_name используется VARCHAR (255) с таким типом данных маловероятно возникнет проблема с длинной ФИО   
   3. Для birth\_date используется DATE поскольку неважно в какое время человек родился, интересует только дата.   
   4. Для registration\_date используется DATE поскольку неважно в какое время человек зарегистрировался, интересует только дата регистрации.  
   5. Для phone\_number используется VARCHAR 20 – 20 символов будет достаточно для телефонного номера, меньше может быть, больше – нет.  
   Все столбцы имеют значение Not Null



**Рис. 6**Так выглядит готовая таблица residents (рис. 6)

1. Таблица carspots [парковочные места] имеет столбцы id, car\_id, residents\_id, description  
   1. Для id – используется INT, Primary Key, Auto Increment – поскольку id используется для идентификации записи в таблице.  
   2. car\_id тип данных INT. Используется для связи с таблицей с машинами. Связь 1 к 1 говорит о том что у 1 машины 1 место.   
   3. residents\_id тип данных INT. Используется для связи с жильцом. У жильца может быть много парковочных мест поэтому связь 1:n один ко многим.  
   4. Для description используется VARCHAR 255. Такого типа данных хватит для описания места где находится парковочное место, например «под деревом у подъезда» или для своих обозначений «первая парковка второе место»



**Рис. 7**

Так выглядит готовя таблица carspots (рис. 7)

1. Таблица car содержит столбцы id, car\_plate.  
   1. Для id – используется INT, Primary Key, Auto Increment – поскольку id используется для идентификации записи в таблице.  
   2. Для car\_plate используется VARCHAR 9 поскольку длина номера автомобиля – 9 символов, например А000АА777



**Рис. 8**Так выглядит готовая таблица car (рис. 8)

1. Таблица apartments\_has\_residents используется для связи таблиц apartments и residents. Столбцы apartments\_id и residents\_id имеют типы INT и Primary Key. Так же они являются Foreign Keys для связей n:m – многие ко многим поскольку у одного жильца может быть несколько квартир и у одной квартиры может быть несколько жильцов



1. Таблица payments\_has\_residents используется для связи таблиц apartments и residents. Столбцы apartments\_id и payments\_id имеют типы INT и Primary Key. Так же они являются Foreign Keys для связей n:m – многие ко многим поскольку у одного жильца может быть несколько платежей и у одного платежа может быть несколько жильцов



1. Таблица residents\_has\_car используется для связи таблиц apartments и residents. Столбцы residents\_id и car\_id имеют типы INT и Primary Key. Так же они являются Foreign Keys для связей n:m – многие ко многим поскольку у одного жильца может быть несколько машин и у одной машины может быть несколько владельцев



1. Таблица logs используется для срабатывания тригера. В таблице хранятся значения id, type, date, desctiprion  
   1. Для id – используется INT, Primary Key, Auto Increment – поскольку id используется для идентификации записи в таблице.  
   2. Для type – используется VARCAHR 45. Тип хранит значение какое действие было сделано  
   3. Для data используется DATETIME – в этом поле используется не только дата, но и время, для более точного определения действия  
   4. Для description используется VARCAHAR 255 чего будет достаточно для описания действия «что было сделано и кем»



**2.1 Хранимые процедуры.**

Хранимые процедуры могут содержать в себе готовый код, который будет выполнен при вызове процедуры. Хранимая процедура MySQL представляет собой подпрограмму, хранящуюся в базе данных. Она содержит имя, список параметров и операторы SQL. 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. [https://it.vshp.online/#/pages/mdk1101/mdk1101\_lab\_18]

В моей базе данных данные процедуры могут исключить ошибку записи в неверном виде и защитить базу от непредвиденных выражений. Хранимая процедура может содержать в себе транзакцию, условия и обработчик исключений, а также локальные переменные.

Транзакции позволяет обеспечить контроль над выполнением операций. Транзакция содержит в себе группу операций как одно целое, при этом все изменения можно или откатить, или подтвердить, если все прошло успешно.

Условия – это операторы для выполнения различных действий в зависимости от условий. В MySQL 8 IF, ELSEIF, ELSE, в моей работе они используются для обработчика исключений.

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

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

Первая хранимая процедура – добавление нового жильца в существующую квартиру. Она принимает 5 значений, которые будут записаны в базу данных. В ней содержится: условие, транзакция и обработчик исключений. В данной хранимой процедуре существует особенность – ID квартиры — это порядковый номер квартиры в базе данных. Он не связан с номером дома и номером квартиры в этом доме.

CREATE DEFINER=`root`@`localhost` PROCEDURE `add\_resident\_to\_apartment`(

IN p\_full\_name VARCHAR(255),

IN p\_birth\_date DATE,

IN p\_registration\_date DATE,

IN p\_phone\_number VARCHAR(45),

IN p\_apartment\_id INT

)

...

В этом блоке кода задается название хранимой процедуры и указывается какие значения она будет принимать и типы данных этих значений.

...

BEGIN

DECLARE v\_resident\_id INT;

DECLARE v\_exists INT DEFAULT 0;

-- Проверка на существование апартамента

SELECT COUNT(\*) INTO v\_exists FROM apartments WHERE id = p\_apartment\_id;

IF v\_exists = 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Такой квартиры не существует';

END IF;

...

В этом блоке кода мы создаем локальные переменные «v\_resident\_id» и «v\_exists» с указанием типа данных. Далее идет проверка на существование квартиры через SELECT-запрос и если в ответе значение 0, то выдает ошибку «Такой квартиры не существует»

...

-- Начало транзакции

START TRANSACTION;

-- Добавление жителя

INSERT INTO residents (full\_name, birth\_date, registration\_date, phone\_number)

VALUES (p\_full\_name, p\_birth\_date, p\_registration\_date, p\_phone\_number);

...

В этом месте начинается транзакция и добавления пользователя путем добавления данных, которые были получены при вызове хранимой процедуре

...

-- Получение ID только что добавленного жителя

SET v\_resident\_id = LAST\_INSERT\_ID();

-- Связывание жителя с уже существующей квартирой

INSERT INTO apartments\_has\_residents (apartments\_id, residents\_id)

VALUES (p\_apartment\_id, v\_resident\_id);

...

В этом месте создается связь между новым жителем и квартирой

...

-- Проверка успешности операций и завершение транзакции

IF ROW\_COUNT() > 0 THEN

COMMIT;

ELSE

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Ошибка при добавлении жителя';

END IF;

END

Тут заканчивается хранимая процедура и транзакция. Если все прошло успешно, то транзакция подтверждается и сохраняется, иначе выдается ошибка «Ошибка при добавлении жителя» и транзакция откатывает все изменения внутри себя. Полный код всей хранимой см. Приложение 1.

Вторая хранимая процедура ускоряет получение всей информации про жителя. Она не содержит в себе транзакции поскольку не затрагивает изменения данных в базе, а лишь выводит их для просмотра

CREATE DEFINER=`root`@`localhost` PROCEDURE `get\_resident\_details`(IN residentID INT)

BEGIN

DECLARE vFullName VARCHAR(255);

DECLARE vTotalApartments INT;

DECLARE vTotalArea FLOAT;

...

В этом блоке кода создается процедура, которая принимает ID пользователя о котором требуется информация, задаются 3 локальные переменные и тип данных к ним

...

-- Получаем полное имя жителя

SELECT full\_name INTO vFullName

FROM residents

WHERE id = residentID;

-- Получаем количество квартир, связанных с жителем

SELECT COUNT(\*) INTO vTotalApartments

FROM apartments\_has\_residents

WHERE residents\_id = residentID;

...

В данном блоке через SELECT узнается ФИО жильца и записывается в переменную «vFullName». Получение всех квартир одного человека происходит через агрегатную функцию COUNT которая считает количество строк. Отличие SUM от COUNT – первое считает сумму значений, а второе количество строк в запросе.

...

-- Получаем общую площадь всех квартир жителя

SELECT SUM(a.area) INTO vTotalArea

FROM apartments a

JOIN apartments\_has\_residents ahr ON a.id = ahr.apartments\_id

WHERE ahr.residents\_id = residentID;

-- Возвращаем результаты

SELECT vFullName AS 'Full Name', vTotalApartments AS 'Total Apartments', vTotalArea AS 'Total Area';

END

В последнем блоке используется агрегатная функция «SUM» для получения суммы значений и записи в переменную. В конце выводятся все данные про жильца через переменные. Полный код см. Приложение 2.

**2.2 Триггеры**

В данной базе данных триггеры используются для сохранения действий пользователей в базе данных в отдельной таблице. Такой способ позволит улучшить модерацию внутри базы.

DELIMITER $$

CREATE TRIGGER `AfterResidentToApartmentInsert`

AFTER INSERT ON `apartments\_has\_residents`

FOR EACH ROW

BEGIN

DECLARE vFullName VARCHAR(255);

-- Получение полного имени жителя по его ID

SELECT full\_name INTO vFullName FROM residents WHERE id = NEW.residents\_id;

-- Формирование описания действия с использованием полного имени вместо ID

SET @Description = CONCAT('Житель ', vFullName, ' добавлен в квартиру с ID ', NEW.apartments\_id);

-- Добавление записи в лог

INSERT INTO `logs` (`type`, `description`)

VALUES ('Добавление', @Description);

END$$

DELIMITER ;

Данный код срабатывает после добавления записи в таблицу «apartments\_has\_residents», затем добавляется запись в таблицу «logs» ФИО нового жителя и id квартиры куда был добавлен, тип «Добавление» обозначает добавление новой записи в базу данных.

**2.3 Типовые запросы**

Типовые запросы состоят из кода, который не состоит в хранимых процедурах и выполняется вручную, с возможностью быстрого изменения поскольку никуда не сохраняются.

Первый типовой запрос – получение списка всех квартир, площадь квартиры и количество комнат в квартире.

SELECT id AS ApartmentID, rooms AS NumberOfRooms, area AS AreaInSqM

FROM apartments

ORDER BY rooms, area;

В данном коде выбирается id квартиры и задается название «ApartmentID», выбирается rooms (комнаты) и задается название «NumberOfRooms», area (площадь) и задается название «AreaInSqM». Данные из таблицы apartments и сортируется квартирам и площади. После выполнения кода выводится список всех квартир, количество комнат в квартире и площадь квартиры.

Второй запрос – получение информации о жителях и количество их квартир

SELECT

b.addres,

r.full\_name,

r.birth\_date,

r.phone\_number,

COUNT(ahr.apartments\_id) AS NumberOfApartments

FROM residents r

JOIN apartments\_has\_residents ahr ON r.id = ahr.residents\_id

JOIN apartments a ON ahr.apartments\_id = a.id

JOIN buildings b ON a.buildings\_id = b.id

GROUP BY r.id, b.addres, r.full\_name, r.birth\_date, r.phone\_number

ORDER BY NumberOfApartments DESC, r.full\_name;

После SELECT из таблицы buildings получаем address, из таблицы residents получаем ФИО, дату рождения, телефонный номер. При помощи COUNT подсчитывается количество квартир. Первый JOIN это соединение с таблицей «apartments\_has\_residents» которая является связующей таблицей между жителями и квартирами. Соединение по столбцу id с столбцом residents\_id. Второй JOIN соединение квартирами и информацией о ней. Третий JOIN соединение с таблицей buildings. Таким образом, результатом запроса будет таблица, отображающая адрес здания, информацию о жителях (имя, дата рождения, номер телефона), а также количество апартаментов, которыми владеет каждый житель, с сортировкой по количеству апартаментов и имени.

Третий типовой запрос – подсчет квартир владельцы которых имеют автомобиль.

SELECT COUNT(DISTINCT a.id) AS ApartmentsWithCarOwners

FROM apartments a

JOIN apartments\_has\_residents ahr ON a.id = ahr.apartments\_id

JOIN residents\_has\_car rhc ON ahr.residents\_id = rhc.residents\_id;

Используется COUNT(DISTINCT) для подсчета только уникальных значений квартир. DISTINCT гарантирует, что будут учтены только уникальные квартира, даже если в одной и той же квартира проживает несколько владельцев автомобилей. Запрос не использует группировку GROUP BY, поскольку агрегирующая функция COUNT(DISTINCT) сама по себе обеспечивает необходимую обработку данных для получения желаемого результата — количество жителей с автомобилями.

Четвертый запрос – список платежей за определенный месяц.

SELECT a.id AS ApartmentID, p.date AS PaymentDate, p.amount AS Amount, p.type AS PaymentType

FROM payments p

JOIN apartments a ON p.apartments\_id = a.id

WHERE MONTH(p.date) = **3** AND YEAR(p.date) = **2024**; -- Пример для **марта 2024**

Через SELECT получаем необходимые данные из таблицы payments. Через JOIN соединение платежа с квартирой. Последняя часть запроса - это фильтр MONTH(p.date) = 3 ограничивает выборку только платежами, совершенными в марте (третий месяц), а YEAR(p.date) = 2024 ограничивает выборку платежами, совершенными в 2024 году.

Пятый запрос – получение информации о наличии кладовки у квартиры.

SELECT a.id AS ApartmentID, IF(ISNULL(s.id), 'Нет', 'Да') AS StoreroomAvailable

FROM apartments a

LEFT JOIN storerooms s ON a.id = s.apartments\_id

ORDER BY ApartmentID;

В данном коде происходить «левое соединение» LEFT JOIN таблицы с квартирами с таблицей кладовок. Левое соединение выбрано, чтобы в результат включались все апартаменты, даже если для них нет кладовых, в этом случае соответствующие поля таблицы storerooms будут содержать значения NULL. В конце кода результаты сортируются по ID квартиры, это упорядочивает вывод, делая его более удобным для понимания.

**2.4 Представление таблиц**

Виртуальные таблицы, которые основаны на результате выполнения SELECT. Они представляют собой логические структуры данных, которые могут быть использованы для упрощения и оптимизации запросов к базе данных. [https://it.vshp.online/#/pages/mdk1101/mdk1101\_lab\_23]

В моей базе данных представление выводит всю информацию про все записи в базе данных. Все данные идут в таком порядке: id жителя, ФИО, дата рождения, дата регистрации, телефонный номер, квартира в которой живет, количество комнат и площадь квартиры, так же id дома, его адрес, количество этажей, количество квартир, год постройки и количество подъездов. Код представления см. Приложение 3. В его коде нету никаких новых интересных выражений. Некоторые из них уже были приведены выше и объяснены (см. Типовые запросы)

**2.5 Функции**

Функции могут быть использованы для выполнения вычислений, преобразования данных, агрегации и других задач. В моей базе данных есть функция подсчета сумм всех платежей определённого жителя за определённый период (от даты до даты). Принимает 3 значения: id пользователя, начало отсчета, конец отсчета. Код см. Приложение 4

**2.6 Роли**

В информационных системах безопасность данных является одним из важнейших аспектов. При работе с базой данных MySQL, рекомендуется следовать принципу наименьших привилегий и создавать пользователей с ограниченным доступом к базе данных. Администратор MySQL должен создавать такие учетные записи для "обычных" пользователей и определять их права доступа. Это позволяет предоставлять пользователям только те привилегии, которые им действительно необходимы для работы, и обеспечивать безопасность данных. Кроме того, проведение аудита полномочий и корректировка их при необходимости также являются важными шагами для обеспечения безопасности базы данных [https://it.vshp.online/#/pages/mdk1101/mdk1101\_lab\_25]

В моей базе есть 3 роли: администратор, модератор и рядовой пользователь. Все они различны. Например, роли модератор и пользователь не видят таблицу logs, хранимые процедуры, представления и функции. Такой способ защищает базу данных от злоумышленников. Роль пользователя может только просматривать таблицу. Роль модератора может изменять, удалять и добавлять данные в таблицы. Роль администратора имеет доступ ко всем таблицам, хранимым процедурам и функциям. Рассмотрим создание ролей на примере роли «пользователь»

-- создание роли

CREATE ROLE IF NOT EXISTS basic\_user\_role;

В данном месте происходит создание роли, если такой не существует

-- присвоение прав

GRANT SELECT ON housing.buildings TO basic\_user\_role;

...

Таким выражение мы даем доступ к SELECT в таблице «housing.buildings»

...

-- создание пользователя, если он еще не существует

CREATE USER IF NOT EXISTS 'user'@'localhost' IDENTIFIED BY '123';

Создание пользователя с названием user и паролем 123, для примера используется простой пароль, но в рабочем варианте стоит задуматься над сильным паролем

-- назначение роли пользователю

GRANT basic\_user\_role TO 'user'@'localhost';

-- активация роли для пользователя

SET DEFAULT ROLE basic\_user\_role TO 'user'@'localhost';

-- применение изменений прав

FLUSH PRIVILEGES;

В этом блоке мы назначим роль пользователю, активируем роль и сохраняем изменения. Последняя строка используется для упрощения изменения прав, когда можно убрать лишнее или добавить новые и права и просто запустить код, без удаления роли или пользователя

**Заключение.**

**Приложение 1**. Хранимая процедура добавления жителя в квартиру

CREATE DEFINER=`root`@`localhost` PROCEDURE `add\_resident\_to\_apartment`(

IN p\_full\_name VARCHAR(255),

IN p\_birth\_date DATE,

IN p\_registration\_date DATE,

IN p\_phone\_number VARCHAR(45),

IN p\_apartment\_id INT

)

BEGIN

DECLARE v\_resident\_id INT;

DECLARE v\_exists INT DEFAULT 0;

-- Проверка на существование апартамента

SELECT COUNT(\*) INTO v\_exists FROM apartments WHERE id = p\_apartment\_id;

IF v\_exists = 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Такой квартиры не существует';

END IF;

-- Начало транзакции

START TRANSACTION;

-- Добавление жителя

INSERT INTO residents (full\_name, birth\_date, registration\_date, phone\_number)

VALUES (p\_full\_name, p\_birth\_date, p\_registration\_date, p\_phone\_number);

-- Получение ID только что добавленного жителя

SET v\_resident\_id = LAST\_INSERT\_ID();

-- Связывание жителя с уже существующей квартирой

INSERT INTO apartments\_has\_residents (apartments\_id, residents\_id)

VALUES (p\_apartment\_id, v\_resident\_id);

-- Проверка успешности операций и завершение транзакции

IF ROW\_COUNT() > 0 THEN

COMMIT;

ELSE

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Ошибка при добавлении жителя';

END IF;

END

**Приложение 2**. Получение информации про жильца

CREATE DEFINER=`root`@`localhost` PROCEDURE `get\_resident\_details`(IN residentID INT)

BEGIN

DECLARE vFullName VARCHAR(255);

DECLARE vTotalApartments INT;

DECLARE vTotalArea FLOAT;

-- Получаем полное имя жителя

SELECT full\_name INTO vFullName

FROM residents

WHERE id = residentID;

-- Получаем количество квартир, связанных с жителем

SELECT COUNT(\*) INTO vTotalApartments

FROM apartments\_has\_residents

WHERE residents\_id = residentID;

-- Получаем общую площадь всех квартир жителя

SELECT SUM(a.area) INTO vTotalArea

FROM apartments a

JOIN apartments\_has\_residents ahr ON a.id = ahr.apartments\_id

WHERE ahr.residents\_id = residentID;

-- Возвращаем результаты

SELECT vFullName AS 'Full Name', vTotalApartments AS 'Total Apartments', vTotalArea AS 'Total Area';

END

**Приложение 3**. Представлении таблицы

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `residents\_apartments\_view` AS

SELECT

`r`.`id` AS `resident\_id`,

`r`.`full\_name` AS `full\_name`,

`r`.`birth\_date` AS иirth\_date,

r.registration\_date AS registration\_date,

r.phone\_number AS phone\_number,

a.id AS apartment\_id,

a.rooms AS rooms,

a.area AS area,

b.id AS building\_id,

b.addres AS addres,

b.number AS building\_number,

b.floors AS floors,

b.aparts AS aparts,

b.building\_year AS building\_year,

b.building\_enters AS building\_enters

FROM

(((residents r

JOIN apartments\_has\_residents ahr ON ((r.id = ahr.residents\_id)))

JOIN apartments a ON ((ahr.apartments\_id = a.id)))

JOIN buildings b ON ((a.buildings\_id = b.id)))

Приложение 4. Функция подсчета

DELIMITER $$

CREATE FUNCTION `GetTotalPaymentsByResident`(

residentID INT,

startDate DATE,

endDate DATE

) RETURNS DECIMAL(10,2)

READS SQL DATA

BEGIN

DECLARE total DECIMAL(10,2);

SELECT SUM(p.amount) INTO total

FROM housing.payments p

JOIN housing.apartments\_has\_residents ar ON p.apartments\_id = ar.apartments\_id

WHERE ar.residents\_id = residentID

AND p.date >= startDate

AND p.date <= endDate;

RETURN IFNULL(total, 0);

END$$

DELIMITER ;