Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЧУВАШСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.Н. УЛЬЯНОВА»

Факультет информатики и вычислительной техники

Кафедра вычислительной техники

Курсовая работа

по дисциплине «Базы данных»

на тему

«Дачный кооператив»

Выполнил**:** студент группы ИВТ-41-22

Иванов Владимир Сергеевич

Проверил**:** кандидат технических наук

Ржавин Вячеслав Валентинович

Чебоксары 2024 г.

**Содержание**

1.Описание предметной области БД……………………………………… 3

2.Список всех возможных атрибутов БД…………………………………. 5

3.ER-диаграмма……………………………………………………………. 6

4.Диаграмма функциональных зависимостей…………………………. 7

5.Структура всех таблиц ...……………………………………………… 8

5.1.Правила валидации……………………………………………… 12

5.2.Описание подастановок…………………………………………... 14

6.Перечень запросов к базе данных……………………………………. 16

7.Название и содержание созданных форм…………………………….. 22

8.Название и содержание созданных отчетов………………………….. 25

9.Название и содержание созданных макросов………………………… 26

10.Название и описание созданных функций и процедур…………….. 27

11.Доступ к данным и веб-приложение…………………………………. 30

12.Описание процедур для интеграции ACCESS с компонентами Office.. 33

13.Описание сводной таблицы и диаграммы…………………………… 33

14.Описание схемы хранилища данных OLAP…………………………. 35

15.Описание XML-документов для импорта и экспорта данных……… 36

16.Выводы………………………………………………………………… 37

17.Список использованной литературы…………………………………. 38

18.Приложение с текстами программы…………………………………… 39

1. **Описание предметной области БД.**

Дачный кооператив - это объединение людей, которые владеют дачными участками на территории одного или нескольких соседних садовых товариществ. Кооператив создается для того, чтобы решать общие вопросы и проблемы, связанные с использованием и обслуживанием земельных участков, строительством и ремонтом дорог, организацией охраны территории, проведением коммуникаций и другими задачами, требующими совместных усилий и средств.

Основная цель дачного кооператива - обеспечить комфортное и безопасное проживание на территории дачного поселка, а также защитить права и интересы его членов. Кооперативы могут заниматься такими вопросами, как благоустройство территории, организация досуга жителей, создание условий для отдыха и занятий спортом, строительство и эксплуатация объектов инфраструктуры (например, детских площадок, спортивных комплексов, зон отдыха).

**Общее описание предметной области.**

База данных создается для дачного кооператива.

Входные, внутренние и выходные документы:

1. Заявление о вступлении в дачный кооператив от нового члена.
2. Договор купли-продажи земельного участка, расположенного на территории дачного кооператива.
3. Заявление о вступлении в дачный кооператив от нового члена.
4. Договор купли-продажи земельного участка, расположенного на территории дачного кооператива.
5. Документы об оплате членских взносов.
6. Свидетельства о праве собственности членов на земельные участки.
7. Бухгалтерская отчетность кооператива.

Сведения участника дачного кооператива:

1. ФИО
2. Телефон
3. Дата рождения
4. Паспортные данные
5. Сведения о земельном участке
6. Статус участника в дачном кооперативе
7. Дата вступления в кооператив

Участники так же имеют свой статус:

1. Действительный член
2. Ассоциированный член
3. Индивидуальный садовод
4. Почетный член

С базой данных дачного кооператива могут работать:

1. Председатель кооператива
2. Члены кооператива
3. Бухгалтер

Председатель кооператива должен иметь возможность решать следующие задачи:

1. Управлять членством в кооперативе, дачными участками, платежами: добавлять, изменять и удалять информацию о членах кооператива, их дачных участках и платежах.
2. Организовывать мероприятия: добавлять, изменять и удалять информацию о мероприятиях, проводимых в кооперативе.
3. Решать проблемы и заявки: обрабатывать заявки от членов кооператива, решать возникающие проблемы и предоставлять отчеты о проделанной работе.
4. Предоставлять отчеты о деятельности кооператива, анализ данных о платежах, участниках и мероприятиях.
5. **Список всех возможных атрибутов БД.**

*Таблица 1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Элемент данных | Описание | Структура или тип данных | Длина | Значения |
| дата |  | гггг.мм.дд | 10 |  |
| Кадастровый номер | Номер земельного участка кооператива | целое |  |  |
| Фамилия | Фамилия владельца/арендатора участка | Строка |  |  |
| Имя | Имя владельца/арендатора участка | Строка |  |  |
| Телефон | Номер телефона владельца/арендатора участка | Строка | 20 |  |
| Площадь участка | Площадь участка в сотках | Десятичное число |  |  |
| Статус участка | Статус участка(сдача в аренду/ пустой/владелец) | Строка |  |  |
| Арендодатель | Владелец участка, сдающегося в аренду | Строка, целое число, дата |  |  |
| Арендатор | Человек, который хочет арендовать участок | Строка, целое число, дата |  |  |
| Договор аренды участка | Содержит всю информацию об аренде, включая оговоренную сумму оплаты, сроки бронирования участка. | Строка, целое число, дата |  |  |

1. **ER-диаграмма.**

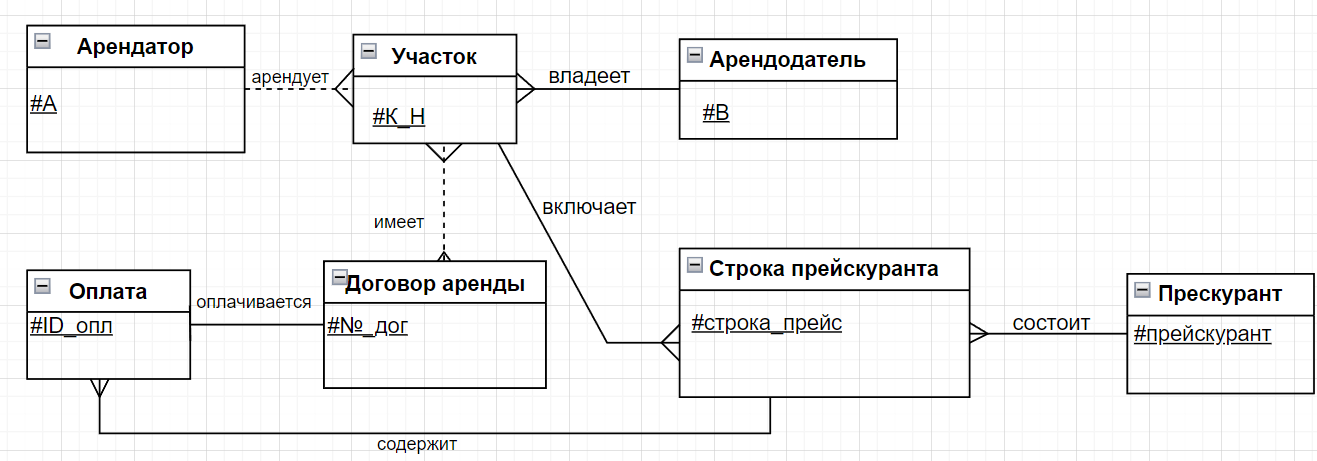


Рис.1 Концептуальная модель для предметной области

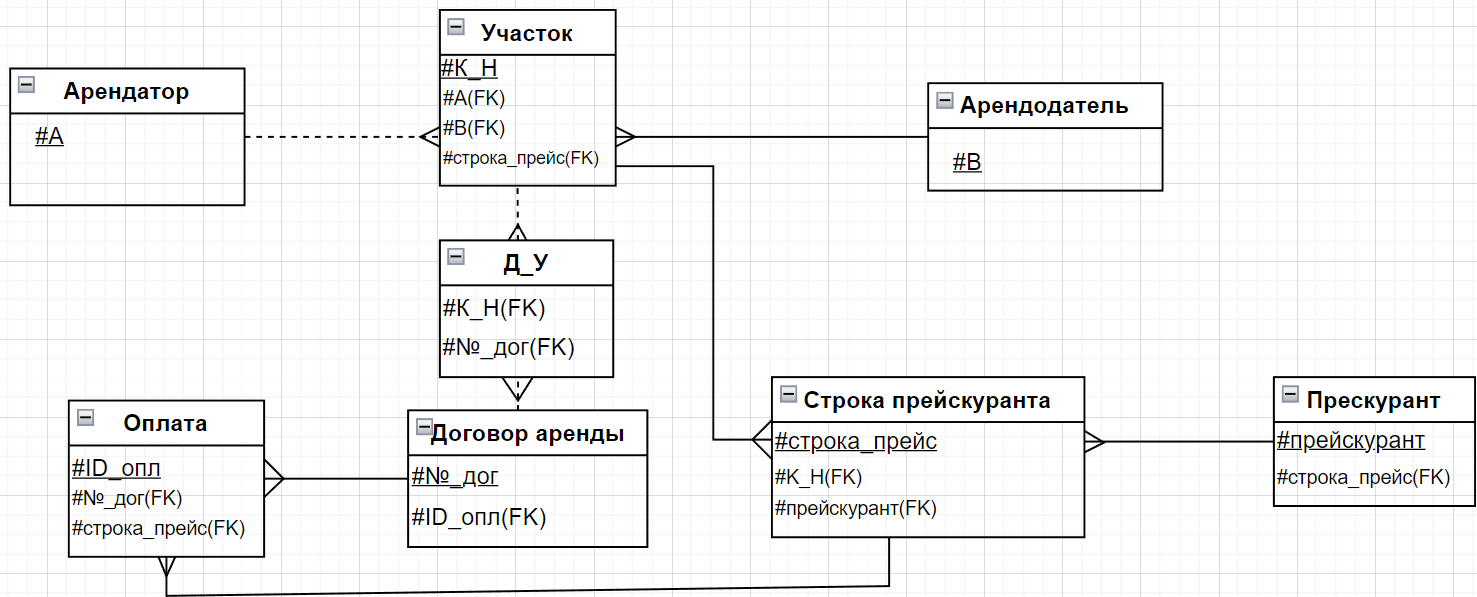


Рис. 2. Логическая модель для рассмотренной предметной области

1. **Диаграмма функциональных зависимостей.**

Функциональные зависимости:

-{ID\_Арендодателя} → {ФИО}

-{ID\_Арендодателя} → {Дата\_Регистрации}

-{ID\_Арендодателя} → {Email}

-{ID\_Арендодателя} → {Номер\_Телефона}

Правила Армстронга:

-Рефлексивность. Если {ID\_Арендодателя} → {ФИО}, то можно сказать, что {ID\_Арендодателя} → {ID\_Арендодателя, ФИО}

-Дополнение. Если {ID\_Арендодателя} → {ФИО}, то {ID\_Арендодателя, Номер\_Телефона} → {ФИО, Номер\_Телефона}

-Транзитивность. Если, например, {ID\_Арендодателя} → {ФИО} и {ФИО} → {Имя}, то {ID\_Арендодателя} → {Имя}. Однако такой зависимости на деле нет.

Так как все атрибуты зависят от ID\_Арендодателя, можно объединить зависимости в одну: {ID\_Арендодателя}→{ФИО, Дата\_Регистрации, Email, Номер\_Телефона}

Так выглядит диаграмма функциональных зависимостей для сущности Арендодатель:

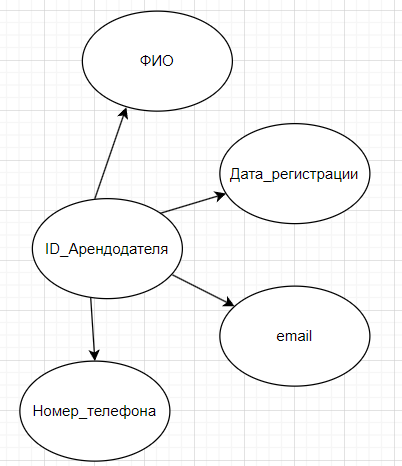


Рис. 3. Диаграмма функциональных зависимостей для сущности Арендодатель

1. **Структура всех таблиц.**

Состав базы данных

*Таблица 2*

|  |  |  |
| --- | --- | --- |
| **№** | **Название таблицы** | **Названия полей** |
| 1 | Арендатор | #ID\_Арендатора, ФИО,Дата\_Регистрации,Email,Номер\_Телефона |
| 2 | Арендодатель | #ID\_Арендодателя, ФИО,Дата\_Регистрации,Email,Номер\_Телефона |
| 3 | Д\_У | #Кадастровый\_Номер, #Номер\_Договора |
| 4 | Договор\_аренды | #Номер\_Договора,Дата\_Начала\_Договора,Дата\_Конца\_Договора,Сумма, #ID\_Арендатора, ID\_Оплаты |
| 5 | Оплата | #ID\_Оплаты,Дата\_Оплаты,Способ\_Оплаты,Статус\_Оплаты,#Номер\_Договора,#Строка\_Прейскуранта |
| 6 | Прейскурант | #Прейскурант, Имя\_Прейскуранта,От\_Какой\_Даты,До\_Какой\_Даты |
| 7 | Строка\_прейскуранта | #Строка\_Прейскуранта, Цена,#Кадастровый\_Номер,#Прейскурант |
| 8 | Участок | #Кадастровый\_Номер,Площадь,Статус\_Участка,Описание,Адрес\_Участка, #ID\_Арендодателя, Строка\_Прейскуранта |

Описание характеристик атрибутов

Арендатор:

*Таблица 3*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Имя атрибута** | **А1** | **А2** | **А3** | **А4** | **А5** | **А6** | **А7** |
| 1 | #ID\_Арендатора | Числовой | дц |  |  | 0 | да | да |
| 2 | ФИО | Текст | 255 |  |  |  | да | нет |
| 3 | Дата\_Регистрации | Д/В |  |  |  |  | нет | нет |
| 4 | Email | Текст | 255 |  |  |  | нет | нет |
| 5 | Номер\_Телефона | Текст | 15 |  |  |  | да | нет |

где

* Тип данных (А1).
* Размер поля (А2).
* Формат вывода значений данного поля (А3).
* Подпись поля в форме (А4).
* Значение по умолчанию (А5).
* Обязательное поле (А6).
* Индексированное поле (А7).

Арендодатель:

*Таблица 4*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Имя атрибута** | **А1** | **А2** | **А3** | **А4** | **А5** | **А6** | **А7** |
| 1 | #ID\_Арендодателя | Числовой | дц |  |  | 0 | да | да |
| 2 | ФИО | Текст | 255 |  |  |  | да | нет |
| 3 | Дата\_Регистрации | Д/В |  |  |  |  | нет | нет |
| 4 | Email | Текст | 255 |  |  |  | нет | нет |
| 5 | Номер\_Телефона | Текст | 15 |  |  |  | да | нет |

Д\_У:

*Таблица 5*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Имя атрибута** | **А1** | **А2** | **А3** | **А4** | **А5** | **А6** | **А7** |
| 1 | Кадастровый\_Номер | Числовой | дц |  |  |  | да | да |
| 2 | Номер\_Договора | Числовой | дц |  |  |  | да | нет |

Договор\_аренды:

*Таблица 6*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Имя атрибута** | **А1** | **А2** | **А3** | **А4** | **А5** | **А6** | **А7** |
| 1 | #ID\_Договора | Числовой | дц |  |  | 0 | да | да |
| 2 | Дата\_Начала\_Договора | Д/В |  |  |  |  | да | нет |
| 3 | Дата\_Конца\_Договора | Д/В |  |  |  |  | да | нет |
| 4 | Стоимость | Денежный |  |  |  | 0 | да | нет |
| 5 | ID\_Оплаты | Числовой | дц |  |  | 0 | да | нет |

Оплата:

*Таблица 7*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Имя атрибута** | **А1** | **А2** | **А3** | **А4** | **А5** | **А6** | **А7** |
| 1 | #ID\_Оплаты | Числовой | дц |  |  | 0 | да | да |
| 2 | Сумма\_Оплаты | Денежный |  |  |  | 0 | да | нет |
| 3 | Дата\_Оплаты | Д/В |  |  |  |  | да | нет |
| 4 | Способ\_Оплаты | Денежный |  |  |  | 0 | да | нет |
| 5 | Статус\_Оплаты | Числовой | дц |  |  | 0 | да | нет |
| 6 | Номер\_Договора | Числовой | Дц |  |  | 0 | Да | нет |
| 7 | Строка\_Прейскуранта | Числовой | дц |  |  | 0 | да | да |

Прейскурант:

*Таблица 8*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Имя атрибута** | **А1** | **А2** | **А3** | **А4** | **А5** | **А6** | **А7** |
| 1 | #Прейскурант | Числовой | дц |  |  | 0 | да | да |
| 2 | Имя\_Прейскуранта | Короткий текст |  |  |  | 0 | да | нет |
| 3 | От\_Какой\_Даты | Д/В |  |  |  |  | да | нет |
| 4 | До\_Какой\_Даты | Д/В |  |  |  | 0 | да | нет |

Строка прейскуранта:

*Талица 9*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Имя атрибута** | **А1** | **А2** | **А3** | **А4** | **А5** | **А6** | **А7** |
| 1 | #Строка\_Прейскуранта | Числовой | дц |  |  | 0 | да | да |
| 2 | Цена | Денежный |  |  |  | 0 | да | нет |
| 3 | Кадастровый\_Номер | Числовой | дц |  |  |  | да | нет |
| 4 | Прейскурант | Числовой | дц |  |  | 0 | да | нет |

Участок:

*Таблица 10*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Имя атрибута** | **А1** | **А2** | **А3** | **А4** | **А5** | **А6** | **А7** |
| 1 | #Кадастровый\_Номер | Числовой | дц |  |  | 0 | да | да |
| 2 | Площадь | Числовой | дц |  |  | 0 | да | нет |
| 3 | Статус\_Участка | Короткий текст | 255 |  |  |  | да | нет |
| 4 | Описание | Длинный текст |  |  |  |  | да | нет |
| 5 | Адрес\_Участка | Короткий текст | 255 |  |  |  | да | нет |
| 6 | ID\_Арендатора | Числовой | дц |  | ID Арендатора | 0 | нет | да |
| 7 | ФИО арендодателя | Числовой | дц |  | ФИО арендодателя | 0 | нет | да |
| 8 | Строка\_Прейскуранта | Числовой | дц |  |  | 0 | нет | нет |

* 1. **Правила валидации**

***Для полей***

*Таблица 11*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Отношение** | **Атрибут** | **Условие на значение** | **Сообщение об ошибке** |
| 1 | Арендатор/Арендодатель | Дата регистрации | < Today() | Дата регистрации в будущем |
| 2 | Участок | Площадь | > 0 | Площадь участка должна быть больше нуля |
| 3 | Оплата | Сумма оплаты | >0 | Сумма оплаты должна быть больше 0 |
| 4 | Строка прейскуранта | Цена | >0 | Цена должна быть больше 0 |
| 5 | Договор\_аренды | Дата\_Начала\_Договора | <Today() | Дата начала договора в будущем |
| 6 | Оплата | Дата\_Оплаты | <=Today() | Дата оплаты в будущем |

***Для записей***

*Таблица 12*

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Отношение** | **Условие на значение** | **Сообщение об ошибке** |
| 1 | Договор аренды | [Дата\_Начала\_Договора]>[Дата\_Конца\_Договора] | Дата начала договора должна быть раньше или такой же, как дата конца договора |
| 2 | Прейскурант | [От\_Какой\_Даты] <= [До\_Какой\_Даты] | Ошибка! Начальная дата прейскуранта позже конечной даты |

* 1. **Описание** **подстановок**

*Таблица 13*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица | Исходное поле | Поле подстановки | Тип элемента управления | Тип источника строк | Источник строк |
| Участок | Статус\_Участка | Статус\_Участка | Список | Список значений | **Арендуется;Свободен** |
| Участок | ID\_Арендодателя | ID\_Арендодателя | Список | Список значений | **1;2;3;4** |
| Участок | ФИО арендодателя | ФИО | список | Список значений | **Иванов В.С; Иванова А.А; Кожевников К.Ю; Ярдыков Э.Е.** |
| Участок | Строка\_Прейскуранта | Строка\_Прейскуранта | список | Список значений | **1;2;3;4** |
| Оплата | Способ\_Оплаты | Способ\_Оплаты | список | Список значений | **Наличные; Кредитная карта** |
| Оплата | Статус\_Оплаты | Статус\_Оплаты | Список | Список значений | **Оплачено; Частично оплачено; Не оплачено** |
| Оплата | Номер\_Договора | Номер\_Договора | Список | Список значений | **1; 2; 3** |
| Оплата | Строка\_Прейскуранта | Строка\_Прейскуранта | Список | Список значений | **1; 2; 3; 4** |
| Договор аренды | ID\_Оплаты | ID\_Оплаты | Список | Список значений | **1; 2; 3** |
| Д\_У | Кадастровый\_Номер | Кадастровый\_Номер | Список | Список значений | **1; 2; 3; 4** |
| Д\_У | Номер\_Договора | Номер\_Договора | Список | Список значений | **1; 2; 3** |

1. **Перечень запросов к базе данных.**

*Таблица 14*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Имя запроса** | **Текст запроса** | | **Запрос на SQL** | |
| 1 | Cписок арендодателей с числом участков, которые они сдают | | Для всех арендодателей выдать количество сдаваемых ими участков(одиночный) | | SELECT Арендодатель.ФИО, COUNT(Участок.Кадастровый\_Номер) AS Количество\_Участков  FROM Арендодатель INNER JOIN Участок ON Арендодатель.ID\_Арендодателя = Участок.ID\_Арендодателя  GROUP BY Арендодатель.ФИО; |
| 2 | Запрос с арифметическим оператором | | Для всех договоров выдать исходную сумму и сумму с учетом НДС | | SELECT Номер\_Договора, Сумма, Сумма\*1.2 AS Сумма\_НДС  FROM Договор\_аренды; |
| 3 | Запрос с опрератором сравнения | | Выдать все номера договоров где Сумма договора больше 10000 | | SELECT \*  FROM Договор\_аренды  WHERE Сумма > 10000; |
| 4 | Запрос с опретором LIKE | | Выдать все записи из таблицы Договор\_аренды , где номер договора выглядит как ‘1’ | | SELECT \*  FROM Договор\_аренды  WHERE Номер\_Договора LIKE '1'; |
| 5 | Запрос с предикатом TOP | | Выдать две первые записи таблицы Оплата, сортированные по сумме оплаты | | SELECT TOP 2 \*  FROM Оплата  ORDER BY Сумма\_оплаты; |
| 6 | Запрос Left Join | | Выдать на таблицу арендаторов записи таблицы Договор\_аренды с совпадением ID\_Арендатора | | SELECT Арендатор.ФИО,  Договор\_аренды.Сумма  FROM Арендатор  LEFT JOIN Договор\_аренды ON Арендатор.ID\_Арендатора = Договор\_аренды.ID\_Арендатора; |
| 7 | self соединение запрос | | Выдать записи таблицы оплата ,которые сперва были частично оплачены, но после были доплачены до конца | | SELECT a.ID\_Оплаты AS Не\_оплачено\_ID, a.Номер\_Договора AS Не\_оплачено\_договор, b.ID\_Оплаты AS Полностью\_оплачено\_ID, b.Номер\_Договора AS Полностью\_оплачено\_договор  FROM Оплата AS a INNER JOIN Оплата AS b ON a.Номер\_Договора = b.Номер\_Договора  WHERE a.Статус\_Оплаты = 'Не оплачено'  AND b.Статус\_Оплаты = 'Оплачено';WHERE a.Статус\_Оплаты = 'Частично оплачено'  AND b.Статус\_Оплаты = 'Оплачено'; |
| 8 | Запрос UNION | | Выдать списки ID\_Арендатора, общие с ID\_Арендодателя | | SELECT ID\_Арендатора FROM Арендатор  UNION SELECT ID\_Арендодателя FROM Арендодатель; |
| 9 | Получить все договора аренды для конкретного участка | | Выдать все договора аренды на участок с кадастровым номером 1 | | SELECT Договор\_аренды.\*  FROM Договор\_аренды INNER JOIN Д\_У ON Договор\_аренды.Номер\_Договора = Д\_У.Номер\_Договора  WHERE Д\_У.Кадастровый\_Номер = 1; |
| 10 | Запрос с расширенным фильтром | | Выдать все записи с таблицы Договор\_аренды,где ID\_Арендатора равен 1 и Дата\_конца\_договора после первого апреля 2024 | | SELECT \*  FROM Договор\_аренды  WHERE ID\_Арендатора = 1 AND Дата\_конца\_договора > #01/04/2024#; |
| 11 | Запрос Create Table | | Создание таблицы с заданными атрибутами сущности | | CREATE TABLE т\_РеестрЗапросов (  QueryID TEXT(255) PRIMARY KEY,  QueryTitle TEXT(255),  QueryDesc MEMO,  QueryTableDate DATE,  QueryExpose YESNO); |
| 12 | Запрос Insert Table | | Внесение новых данных в таблицу т\_РеестрЗапросов | | INSERT INTO т\_РеестрЗапросов ( QueryID, QueryTitle, QueryDesc, QueryTableDate, QueryExpose )  SELECT Name, ForeignName, NULL AS QueryDesc, DateCreate, IIf(Type=5,True,False) AS QueryExpose  FROM MSysObjects  WHERE Type = 5; |
| 13 | Запрос UPDATE | | Изменение значения QueryTitle на вводимое | | UPDATE т\_РеестрЗапросов SET QueryTitle = QueryTitle2; |
| 14 | Запрос DELETE | | Удаление строки в таблице с QueryId = ‘verw3vg’ | | DELETE FROM т\_РеестрЗапросов  WHERE QueryID = 'verw3vg'; |
| 15 | Запрос с внутренним неэквивалентьным соединением | | Вернуть все комбинации строк где ID\_Арендаторв больше и равен ID\_Арендодателя | | SELECT Арендатор.\*, Арендодатель.\*  FROM Арендатор INNER JOIN Арендодатель ON Арендатор.ID\_Арендатора >= Арендодатель.ID\_Арендодателя; |
| 16 | Запрос с оператором условный выражений | | Вернуть таблицу,для каждого арендатора добавить стату Новый если дата регистрации 2024, Старый в любом бругом случае | | SELECT ID\_Арендатора, ФИО, Дата\_Регистрации, Email, Номер\_Телефона, IIf(Year(Дата\_Регистрации) = 2024, 'Новый', 'Старый') AS Статус  FROM Арендатор; |
| 17 | Запрос с использованием статичстических функций по подмножеству | | Вернуть таблицу с четырьмя колоннами: Статус\_Оплаты, Общая\_Сумма, Средняя\_Сумма и Количество\_Платежей | | SELECT Статус\_Оплаты,  SUM(Сумма\_Оплаты) AS Общая\_Сумма,  AVG(Сумма\_Оплаты) AS Средняя\_Сумма,  COUNT(\*) AS Количество\_Платежей  FROM Оплата  GROUP BY Статус\_Оплаты; |
| 18 | Запрос с вложенным подзапросом | | Выдать все записи с таблицы Оплата где Сумма\_Оплаты больше среднего значения | | SELECT \*  FROM Оплата  WHERE Сумма\_Оплаты > (Select AVG(Сумма\_Оплаты) FROM Оплата); |
| 19 | Коррелированный подзапрос | | Выдать все записи с таблицы Оплата,сумма которых выше средней суммы для каждой оплаты | | SELECT \*  FROM Оплата AS o1  WHERE Сумма\_Оплаты > (SELECT AVG(Сумма\_Оплаты)  FROM Оплата AS o2  WHERE o1.Способ\_Оплаты = o2.Способ\_Оплаты); |
| 20 | Запрос с запросом как источником исходных данных | | Выдать сумму всех платежей, полученных из запроса [Получить все платежи по договору аренды] | | SELECT SUM(Сумма\_Оплаты) AS Общая\_Сумма FROM [Получить все платежи по договору аренды]; |
| 21 | Перекрестный запрос | | Выдать таблицу, где строки представляют ID\_Арендатора, а столбцы ID\_Договора.В ячейках отображать сумма оплат каждого арендатора по договору | | TRANSFORM SUM(Сумма) AS Общая\_Сумма  SELECT ID\_Арендатора  FROM Договор\_аренды  GROUP BY ID\_Арендатора  PIVOT Номер\_Договора; |
| 22 | Получение арендатора по ID | | Выдать таблицу с информацией об Арендаторе по введеному ID | | SELECT \*  FROM Арендатор  WHERE ID\_Арендатора = [ВВедите ID\_Арендатора]; |
| 23 | Договор\_аренды Итоговая сумма | | Выдать сумму платежей, указанных в договоре | | SELECT DISTINCTROW Sum([Договор\_аренды].[Сумма]) AS [Sum - Сумма]  FROM Договор\_аренды; |
| 24 | Группировка платежей по арендатору | | Выдать сумму платежа как Общая\_Сумма из таблицы Догово\_аренды сгруппированную по ID\_Арендатора | | SELECT ID\_Арендатора, SUM(Сумма) AS Общая\_Сумма  FROM Договор\_аренды  GROUP BY ID\_Арендатора; |

1. **Название и содержание созданных форм.**

*Таблица 15*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название формы** | **Тип формы** | **Назначение** | **Примечание** |
| 1 | Арендатор | Иерархическая | Показ карточек арендатора | На форме размещены таблица с общей суммой договоров и диаграмма |
| 2 | Арендодатель | Простая | Карточка арендодателя с его ID и ФИО. Остальная информация содержится в закладках | На форме размещены закладки |
| 3 | Д\_У | Простая | Карточка с кадастровым номером дома и номером договора на него, синхронизированная с формой участок | На форме размещена ссылка на указанный участок |
| 4 | Договор аренды | Простая | Карточка договора со всей информацией | На форме размещены данные с условным форматированием и диаграмма, а также колонититулы |
| 5 | Договор\_аренды\_DatasheetSub4 | Подчиненная | Высчитывает общую сумму, которую потратил арендатор на все аренды | - |
| 6 | Запрос по форме | Простая | Карточка, в которой можно ввести ID\_Арендатора и вывести договора, которые он заключил | На форме размещена таблица, соединенная с запросом и кнопка. Которая этот запрос и выполняет |
| 7 | Получение арендатора по ID | Простая | Форма на основе запроса | Выводит информацию о определенном арендаторе |
| 8 | Строка\_прейскуранта | Простая | Форма, синхронизированная с формой участок | Выводит данные о строке прейскуранта, с помощью кнопки можно перейти на форму Участок С помощью кадастрового номера |
| 9 | Т\_РеестрЗапросов | Простая | Показывает все созданные запросы, с возможностью их выполнить | На форме есть кнопка, которая позволяет выполнить выбранный запрос |
| 10 | Участок1 | Простая | Выводит информацию об участке | Форма, синхронизированная с формой Строка\_прейскуранта, Д\_У и Арендодатель1 |
| 11 | Оплата | Простая | Выводит информацию об оплатах |  |
| 12 | ЭУ\_Календарь | Простая | Форма в которой используется Элемент управления календарь | Работает как фильрт дат для талицы Оплата |
| 13 | Арендодатель1 | Простая | Форма синхронизированная с формами Д\_У, Участок1 и Арендодатель | Главная форма |

1. **Название и содержание созданных отчетов**

*Таблица 16*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название отчета | Тип отчета | Назначение | Примечание |
| 1 | Арендатор и договоры | Многотабличный отчет из таблиц Арендатор и Договор\_аренды | Вывод всех арендаторов в отчет и их договоров |  |
| 2 | Арендодатель | Однотабличный отчет с примечанием в виде счетчика и сортировкой | Вывод всех арендодателей с их общим числом и сортировкой по дате регистрации |  |
| 3 | Оплата | Однотабличный отчет | Вывод всех оплат |  |
| 4 | Участок без подробностей | Однотабличный отчет с группировкой без подробностей | Отчет по участкам с группировкой, но без подробностей. |  |
| 5 | Арендодатель1 | Однотабличный отчет из таблицы Арендодатель с подчиненным отчетом Участок | Отчет по арендодателям и участкам которые они сдают |  |
| 6 | Участок | Однотабличный отчет | Вывод в отчет всех участков |  |

1. **Название и содержание созданных макросов.**

*Таблица 17*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название макроса | Тип макроса | Назначение | Примечание |
| 1 | AutoExec | Макрос формы | Открытие главной формы при запуске БД |  |
| 2 | AutoKeys | Макрос горячих клавиш | Открытие окна сообщения на нажатие клавиши {F1} |  |
| 3 | ShortcutМеню | Макрос меню | Макрос для добавления в меню команд из макроса КомандыМеню |  |
| 4 | автономный для таблиц | Макрос таблицы | Открытие таблицы |  |
| 5 | автономный макрос формы + если+поиск | Макрос формы | Открытие формы, показ окна сообщения для макроса Если и Поиск по записям |  |
| 6 | автономный открыть отчет | Макрос отчета | Открытие отчета |  |
| 7 | вложенный макрос | Вложенный макрос | При ошибке вывести сообщение самого вложенного макроса |  |
| 8 | КомандыМеню | Макрос команды меню | Макрос для определения команд меню чтобы позже добавить их в форму в макросе ShortcutМеню |  |
| 9 | Окно сообщения | Макрос сообщения | Макрос вывода сообщения |  |
| 10 | Именованный макрос | Макрос данных в таблице Договор\_аренды | Вывод ошибки |  |
| 11 | Внедренный макрос | Макрос в форме | Вывод сообщения |  |
| 12 | Макрос применить фильтр | Фильтр | Применение указанного фильтра |  |

1. **Название и описание созданных функций и процедур.**

*Таблица 18*

|  |  |  |  |
| --- | --- | --- | --- |
| **Условие** | **Макрокоманда** | **Аргументы: Значения** | **Пояснение** |
|  | ОткрытьФорму | Имя формы: Арендодатель1 | Открывает форму при запуске БД |
|  |  | Режим: форма |
|  |  | Режим окна: обычное |
|  | Вложенный макрос | {F1} | Комбинация клавиши для открытия окна сообщения |
|  | Окно сообщения | Сообщение:Hello  Сигнал: Да  Тип: информационное  Заголовок: Это макрос бинда |
|  |  |
|  |  |
|  |  |
|  | Добавитьменю | Название меню: FormCommands  Имя макроса:КомандыМеню | добавление в меню команд из макроса КомандыМеню |
|  |  |
|  | ОткрытьТаблицу | Имя таблицы: Арендодатель | Открытие таблицы Арендодатель |
|  |  | Режим: Таблица |
|  |  | Режим данных: Только чтение |
|  | ОткрытьФорму | Имя формы: Арендодатель1  Режим: **Форма**  Имя фильтра:  Условие отбора:  Режим данных:  Режим окна: **Обычное** | Открывает форму «Арендодатель1» |
|  | Если | IsDate(CDate([Формы]!  [Арендодатель1]!  [Дата\_Регистрации])) | Если в форме Арендодатель1 поле даты является полем даты то рикпывается окно сообщения |
|  | НайтиЗапись | Образец поиска: Ива  Совпадение: С начала поля  С учетом регистра: Нет  Область поиска: Все  С учетом формата поля: Нет  Только в текущем поле: Нет  Первое вхождение: Да | Поиск записи по образцу Ива |
|  | ОткрытьОтчет | Имя отчета: Арендодатель  Режим: Отчет  Имя фильтра:  Условие отбора:  Режим окна: Обычное | Открывает отчет Арендодатель |
|  | ПриОшибке | Перейти: Имя макроса  Имя макроса: вложенный\_макрос | При ошибке БД переход к макросу с названием вложенный\_макрос |
|  | Вложенный макрос &Print | ЗапускКомандныМеню: ПечатьОбъекта | Макрос который запускает печать |
|  | Вложенный макрос &Refresh | ОбновитьЗапись | Обновление формы |
|  | Вложенный макрос &Close | ЗакрытьОкно  Тип объекта: Форма  Имя объекта: Арендодаетль1  Сохранение: Подсказка | Макрос добавления в меню команды закрыть которая закрывает форму |

1. **Доступ к данным и веб-приложение.**

В ACCESS для авторизации была создана форма авторизации(Рис. 3).

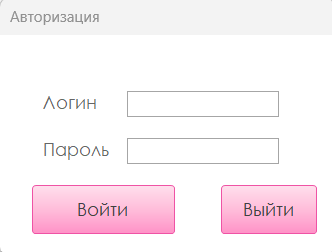


Рис. 4. Форма авторизации в Access

Была создана таблица Users, в которой хранятся логины и пароли пользователей с их ролями.

Так, для ролей были созданы ограничения в формах. Например, если роль авторизованного пользователя Admin, он имеет полную свободу в редактировании данных. Роль Director в свою очередь позволяет редактировать и создавать новые данные, но запрещает удалять данные. Роль User же может только добавлять новые данные.

Приложение разработано с помощью языка Python и использованием библиотеки tkinter.

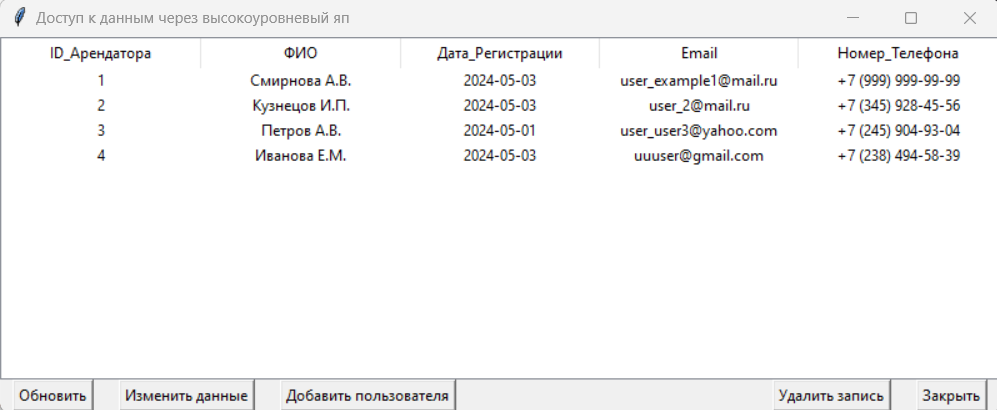


Рис. 5. Интерфейс основного окна приложения

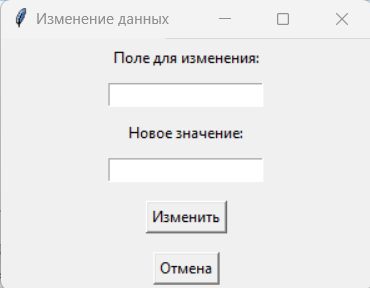


Рис. 6. Интерфейс окна изменения данных в поле

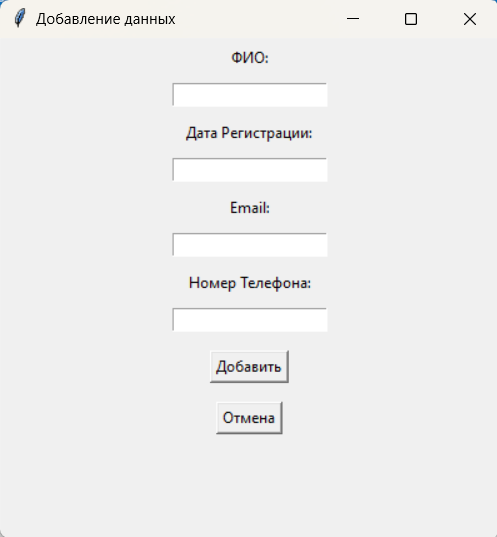


Рис. 7. Интерфейс добавления нового аренатора

Веб-приложение также написано на Python, но теперь с использованием Flask.

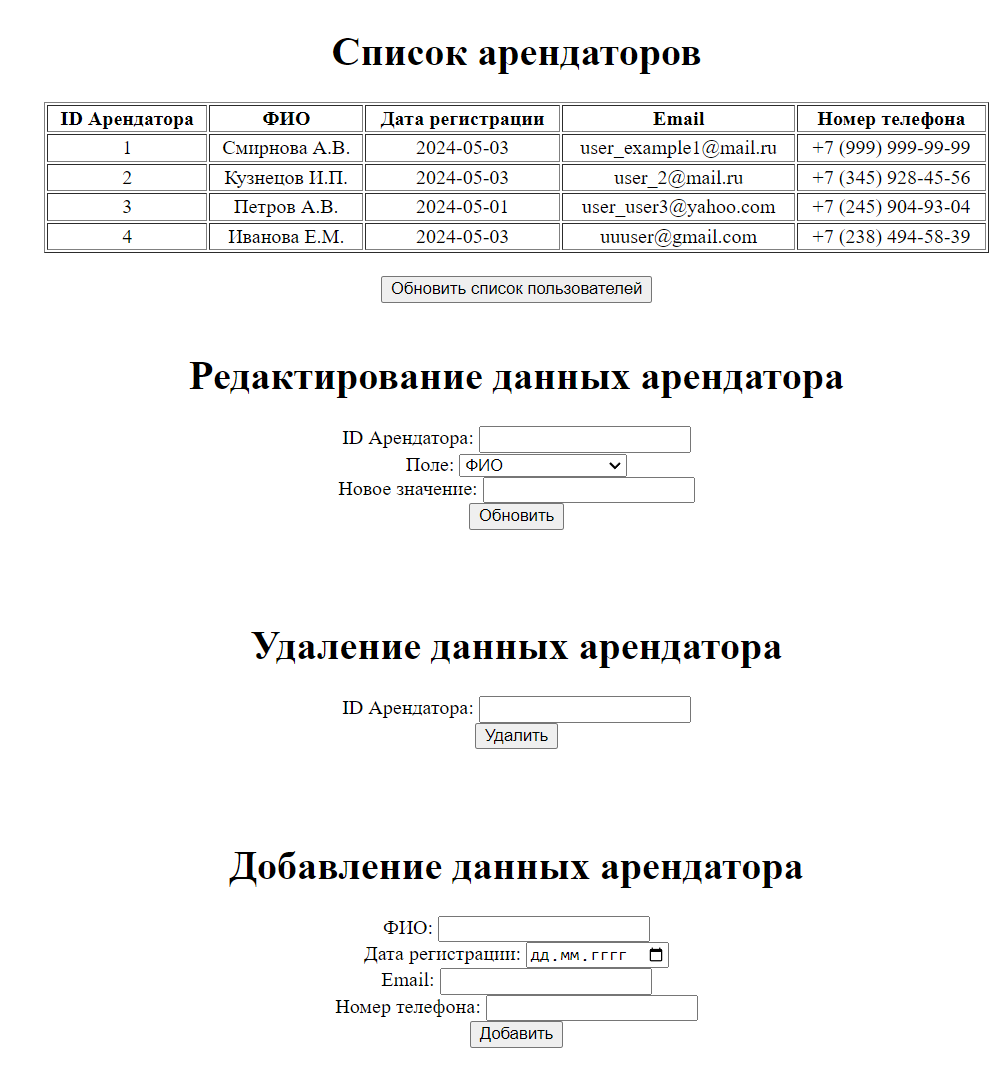
****

Рис. 8. Интерфейс веб-страницы.

1. **Описание процедур для интеграции ACCESS с компонентами Office.**

**Изображение выглядит как текст, Шрифт, число, линия

Автоматически созданное описание**

Рис. 9. Работа с таблицей, импортированной из Access в Excel

**Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание**

Рис. 10. Пример массовой рассылки в Word с использованием данных из Access

1. **Описание сводной таблицы и диаграммы.**

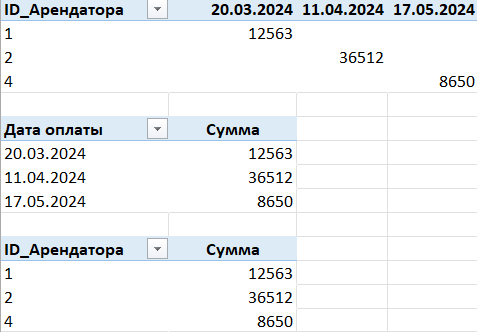
****

Рис. 11. Сводные таблицы

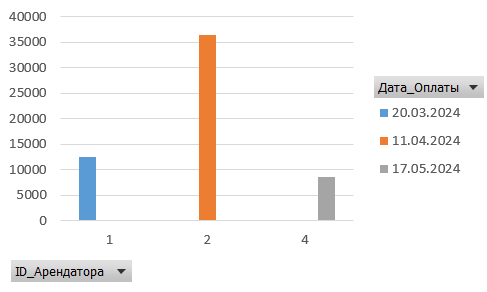
****

Рис. 12. Сводная диаграмма

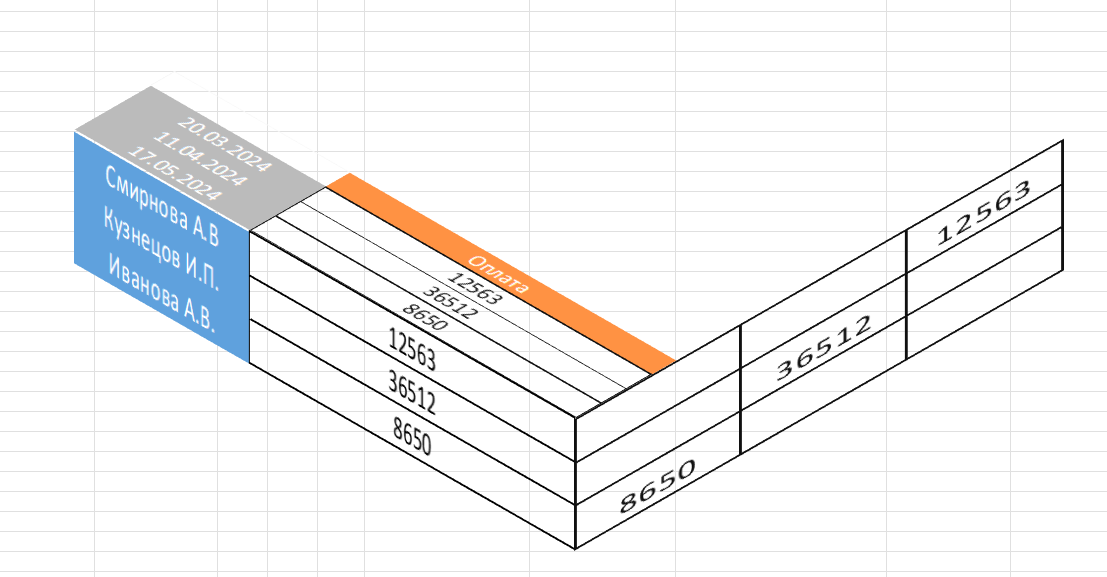


Рис. 13. Трехмерная таблица

1. **Описание схемы хранилища данных OLAP.**

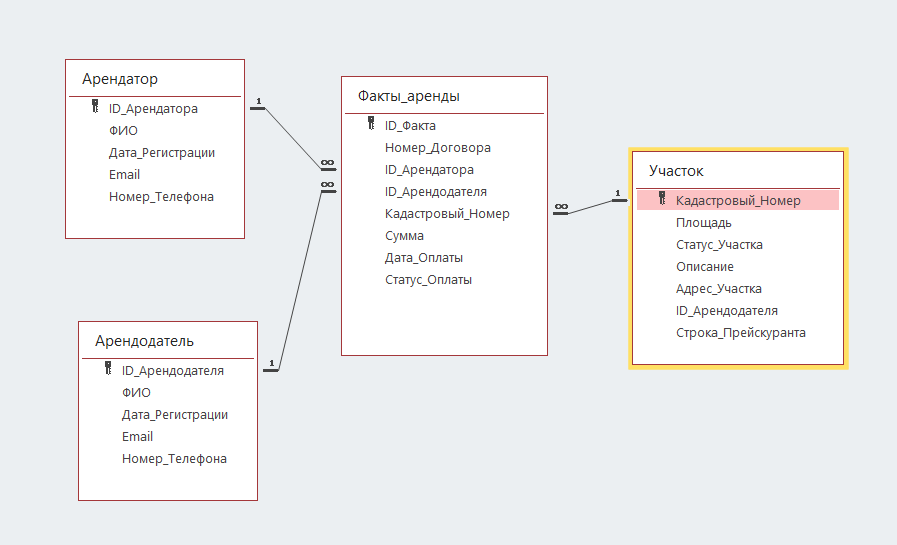
****

Рис. 14. Схема данных OLAP

1. **Описание XML-документов для импорта и экспорта данных.**

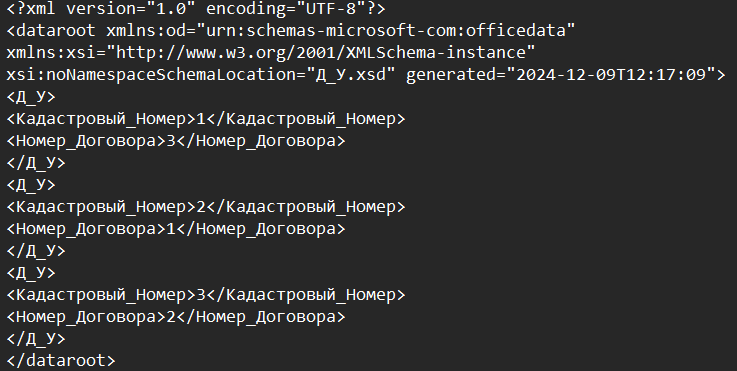
****

Рис. 15. Изначальный xml код таблицы Д\_У

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

Рис. 16. Xml код для добавления данных в таблицу

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание**

Рис. 17. Итог импорта новых данных в существующую таблицу

1. **Выводы.**

В ходе выполнения курсовой работы была проанализирована предметная область темы, разработаны таблицы по ней, Er-диаграмма, запросы к таблицам, созданы формы на основе полученных таблиц, отчеты, макросы, интегрировал Access с компонентами Office, создал сводную таблицу и диаграмму, разработал приложение и веб-страницу, изучил взаимодействие с xml-файлами.

**Список использованной литературы.**

1. «Самоучитель Access 2016. Самоучитель» Ю. Б. Бекаревич, Н. В. Пушкина., БХВ-Петербург, 2017, 475 с.
2. Справка и обучение по Access // URL: https://support.microsoft.com/ru-ru/access (дата обращения: 10.10.2024).

**Приложение**

* + 1. Код программы приложения:

import pyodbc

from tkinter import \*

from tkinter import ttk

from tkinter import messagebox

from datetime import datetime

def validate\_number(phone\_number):

# Убираем все нецифровые символы, кроме +

digits = ''.join(filter(lambda x: x.isdigit() or x == '+', phone\_number))

formatted\_number = ""

if len(digits) > 0:

formatted\_number += "+7 "

if len(digits) > 2:

formatted\_number += "(" + digits[2:5] # Код города

if len(digits) >= 5:

formatted\_number += ") " + digits[5:8] # Первая часть номера

if len(digits) >= 8:

formatted\_number += "-" + digits[8:10] # Вторая часть номера

if len(digits) >= 10:

formatted\_number += "-" + digits[10:12] # Третья часть номера

return formatted\_number

def key\_release(event, phone\_number\_entry):

# Получаем текущее значение и форматируем его

current\_value = phone\_number\_entry.get()

formatted\_value = validate\_number(current\_value)

phone\_number\_entry.delete(0, END)

phone\_number\_entry.insert(0, formatted\_value)

def connect\_to\_access\_db(db\_file: str) -> pyodbc.Connection:

"""Устанавливает соединение с базой данных Access."""

connection\_string = rf'DRIVER={{Microsoft Access Driver (\*.mdb, \*.accdb)}};DBQ={db\_file};'

return pyodbc.connect(connection\_string)

def fetch\_all\_users(conn: pyodbc.Connection) -> list:

"""Извлекает всех пользователей из базы."""

cursor = conn.cursor()

cursor.execute('SELECT \* FROM Арендатор')

rows = cursor.fetchall()

cursor.close()

return rows

def update\_table(conn: pyodbc.Connection, user\_id: int, field: str, new\_value: str):

"""Обновляет данные пользователя в базе."""

cursor = conn.cursor()

sql = f'UPDATE Арендатор SET {field} = ? WHERE ID\_Арендатора = ?'

cursor.execute(sql, (new\_value, user\_id))

conn.commit()

cursor.close()

def insert\_table(conn: pyodbc.Connection, full\_name: str, registration\_date: datetime, email: str, phone\_number: str):

"""Добавляет данные пользователя в базу."""

cursor = conn.cursor()

sql = f'INSERT INTO Арендатор (ФИО, Дата\_Регистрации, Email, Номер\_Телефона) VALUES ( ?, ?, ?, ?)'

cursor.execute(sql, (full\_name, registration\_date, email, phone\_number))

conn.commit()

cursor.close()

def delete\_table(conn: pyodbc.Connection, user\_id: int):

cursor = conn.cursor()

sql = f'DELETE FROM Арендатор WHERE ID\_Арендатора = ?'

cursor.execute(sql, (user\_id,))

conn.commit()

cursor.close()

update\_user\_table()

def format\_date(date\_value):

"""Форматирует дату в строку без времени."""

if isinstance(date\_value, datetime):

return date\_value.strftime('%Y-%m-%d') # Форматируем дату как 'ГГГГ-ММ-ДД'

return date\_value

def tuple\_to\_string(x):

formatted\_values = []

for item in x:

if isinstance(item, datetime):

formatted\_values.append(format\_date(item)) # Форматируем дату

else:

formatted\_values.append(str(item))

return tuple(formatted\_values)

def update\_user\_table():

"""Обновляет таблицу пользователей."""

# Очищаем текущие данные в Treeview

for item in userTable.get\_children():

userTable.delete(item)

# Получаем новых пользователей из базы данных

conn = connect\_to\_access\_db('./Database1.accdb')

users = fetch\_all\_users(conn)

conn.close()

# Заполняем Treeview новыми данными

for user in users:

userTable.insert("", END, values=tuple\_to\_string(user))

def open\_update\_window():

"""Открывает окно для изменения данных пользователя."""

selected\_item = userTable.selection()

if not selected\_item:

messagebox.showwarning("Предупреждение", "Пожалуйста, выберите пользователя для изменения.")

return

user\_id = userTable.item(selected\_item)['values'][0] # Получаем ID выбранного пользователя

# Создаем новое окно

update\_window = Toplevel(root)

update\_window.title("Изменение данных")

# Поля для ввода данных для изменения

Label(update\_window, text="Поле для изменения:").pack(pady=5)

field\_entry = Entry(update\_window)

field\_entry.pack(pady=5)

Label(update\_window, text="Новое значение:").pack(pady=5)

new\_value\_entry = Entry(update\_window)

new\_value\_entry.pack(pady=5)

def update\_data():

field\_to\_update = field\_entry.get()

new\_value = new\_value\_entry.get()

conn = connect\_to\_access\_db('./Database1.accdb')

if conn:

try:

update\_table(conn, user\_id, field\_to\_update, new\_value)

messagebox.showinfo("Успех", "Данные успешно изменены.")

update\_user\_table() # Обновляем таблицу после изменения

update\_window.destroy() # Закрываем окно изменения

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось изменить данные: {e}")

finally:

conn.close()

Button(update\_window, text="Изменить", command=update\_data).pack(pady=10)

Button(update\_window, text="Отмена", command=update\_window.destroy).pack(pady=5)

def open\_insert\_window():

insert\_window = Toplevel(root)

insert\_window.geometry("400x400")

insert\_window.title("Добавление данных")

Label(insert\_window, text="ФИО:").pack(pady=5)

full\_name\_entry = Entry(insert\_window)

full\_name\_entry.pack(pady=5)

Label(insert\_window, text="Дата Регистрации:").pack(pady=5)

registration\_date\_entry = Entry(insert\_window)

registration\_date\_entry.pack(pady=5)

Label(insert\_window, text="Email:").pack(pady=5)

email\_entry = Entry(insert\_window)

email\_entry.pack(pady=5)

Label(insert\_window, text="Номер Телефона:").pack(pady=5)

phone\_number\_entry = Entry(insert\_window)

phone\_number\_entry.pack(pady=5)

# Привязываем обработчик события нажатия клавиши

phone\_number\_entry.bind("<KeyRelease>", lambda event: key\_release(event, phone\_number\_entry))

def insert\_field():

full\_name = full\_name\_entry.get()

registration\_date = registration\_date\_entry.get()

email = email\_entry.get()

phone\_number = phone\_number\_entry.get() # Получаем значение из поля ввода

formatted\_phone\_number = validate\_number(phone\_number)

# Проверяем, корректен ли номер телефона

if formatted\_phone\_number != phone\_number:

messagebox.showwarning("Ошибка", "Неверный формат номера телефона.")

return

conn = connect\_to\_access\_db('./Database1.accdb')

if conn:

try:

insert\_table(conn, full\_name, registration\_date, email, formatted\_phone\_number)

messagebox.showinfo("Успех", "Данные успешно добавлены.")

update\_user\_table() # Обновляем таблицу после добавления

insert\_window.destroy() # Закрываем окно добавления

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось добавить данные: {e}")

finally:

conn.close()

Button(insert\_window, text="Добавить", command=insert\_field).pack(pady=10)

Button(insert\_window, text="Отмена", command=insert\_window.destroy).pack(pady=5)

def delete\_field():

selected\_item = userTable.selection()

if not selected\_item:

messagebox.showwarning("Предупреждение", "Пожалуйста, выберите поле для удаления.")

return

user\_id = userTable.item(selected\_item)['values'][0]

conn = connect\_to\_access\_db('./Database1.accdb')

if conn:

delete\_table(conn, user\_id)

update\_user\_table()

conn.close()

# Инициализация основного окна

root = Tk()

root.geometry("800x300")

root.title("Доступ к данным через высокоуровневый яп")

# Определяем колонки для Treeview

columns = ('ID\_Арендатора', 'ФИО', 'Дата\_Регистрации', 'Email', 'Номер\_Телефона')

# Создаем Treeview

userTable = ttk.Treeview(columns=columns, show='headings')

userTable.pack(fill=BOTH, expand=1)

# Настраиваем заголовки и ширину колонок

for col in columns:

userTable.heading(col, text=col)

userTable.column(col, anchor='center', width=100) # Устанавливаем ширину по умолчанию

# Первоначальное заполнение таблицы

update\_user\_table()

# Кнопка "Обновить"

Button(root, text="Обновить", command=update\_user\_table).pack(side=LEFT, padx=10)

# Кнопка "Изменить данные"

Button(root, text="Изменить данные", command=open\_update\_window).pack(side=LEFT, padx=10)

# Кнопка "Добавить пользователя"

Button(root, text="Добавить пользователя", command=open\_insert\_window).pack(side=LEFT, padx=10)

# Кнопка "Закрыть"

Button(root, text="Закрыть", command=root.quit).pack(side=RIGHT, padx=10)

#Кнопка "Удалить записть"

Button(root, text = "Удалить запись", command = delete\_field).pack(side = RIGHT, padx=10)

# Запускаем главный цикл приложения

root.mainloop()

* + 1. Код веб-страницы:

from flask import Flask, render\_template\_string, jsonify, request, redirect, flash

import pyodbc

from datetime import datetime

app = Flask(\_\_name\_\_)

def connect\_to\_access\_db(db\_file: str) -> pyodbc.Connection:

"""Устанавливает соединение с базой данных Access."""

connection\_string = rf'DRIVER={{Microsoft Access Driver (\*.mdb, \*.accdb)}};DBQ={db\_file};'

return pyodbc.connect(connection\_string)

def fetch\_all\_users(conn: pyodbc.Connection) -> list:

"""Извлекает всех пользователей из базы."""

cursor = conn.cursor()

cursor.execute('SELECT \* FROM Арендатор')

rows = cursor.fetchall()

formatted\_rows = []

for row in rows:

formatted\_row = list(row)

formatted\_row[2] = format\_date(formatted\_row[2])

formatted\_row[3] = str(formatted\_row[3])

formatted\_rows.append(formatted\_row)

cursor.close()

return formatted\_rows

def format\_date(date\_value):

"""Форматирует дату в строку без времени."""

if isinstance(date\_value, datetime):

return date\_value.strftime('%Y-%m-%d') # Форматируем дату как 'ГГГГ-ММ-ДД'

return date\_value

def update\_table(conn: pyodbc.Connection, user\_id: int, field: str, new\_value: str):

"""Обновляет данные пользователя в базе."""

cursor = conn.cursor()

# Проверка на допустимое поле

valid\_fields = ["ФИО", "Дата\_Регистрации", "Email", "Номер\_Телефона"]

if field not in valid\_fields:

raise ValueError(f"Недопустимое поле: {field}")

# Экранируем одинарные кавычки в new\_value

new\_value = new\_value.replace("'", "''")

# Формируем SQL-запрос

sql = f'UPDATE Арендатор SET {field} = ? WHERE ID\_Арендатора = ?'

# Выполняем запрос

cursor.execute(sql, (new\_value, user\_id))

conn.commit() # Не забудьте зафиксировать изменения

cursor.close()

def delete\_table(conn: pyodbc.Connection, user\_id: int):

cursor = conn.cursor()

sql = f'DELETE FROM Арендатор WHERE ID\_Арендатора = ?'

cursor.execute(sql, (user\_id,))

conn.commit()

cursor.close()

def insert\_table(conn: pyodbc.Connection, full\_name: str, registration\_date: datetime, email: str, phone\_number: str):

"""Добавляет данные пользователя в базу."""

cursor = conn.cursor()

sql = f'INSERT INTO Арендатор (ФИО, Дата\_Регистрации, Email, Номер\_Телефона) VALUES ( ?, ?, ?, ?)'

cursor.execute(sql, (full\_name, registration\_date, email, phone\_number))

conn.commit()

cursor.close()

@app.route('/')

def home():

return render\_template\_string('''

<body style = "display:flex; justify-content: center; align-items: center; flex-direction: column;">

<dis style = "width:75%; text-align: center;">

<h1 style="text-align: center;">Список арендаторов</h1>

<div id = user\_table></div>

<br>

<button onclick = "fetchUsers()">Обновить список пользователей</button>

<br><br>

<h1 style="text-align: center;">Редактирование данных арендатора</h1>

<form action="/update\_user" method="POST">

<label for="user\_id">ID Арендатора:</label>

<input type="number" id="user\_id" name="user\_id" required><br>

<label for="field">Поле:</label>

<select id="field" name="field" required>

<option value="ФИО">ФИО</option>

<option value="Дата\_Регистрации">Дата регистрации</option>

<option value="Email">Email</option>

<option value="Номер\_Телефона">Номер телефона</option>

</select><br>

<label for="new\_value">Новое значение:</label>

<input type="text" id="new\_value" name="new\_value" required pattern = ""><br>

<button type="submit">Обновить</button>

</form>

<br><br>

<h1 style="text-align: center;">Удаление данных арендатора</h1>

<form action="/delete\_user" method="POST">

<label for="user\_id">ID Арендатора:</label>

<input type="number" id="user\_id" name="user\_id" required><br>

<button type="submit">Удалить</button>

</form>

<br><br>

<h1 style="text-align: center;">Добавление данных арендатора</h1>

<form action="/insert\_user" method="POST">

<label for="full\_name">ФИО:</label>

<input type="text" id="full\_name" name="full\_name" required><br>

<label for="registration\_date">Дата регистрации:</label>

<input type="date" id="registration\_date" name="registration\_date" required><br>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required><br>

<label for="phone\_number">Номер телефона:</label>

<input type="tel" id="phone\_number" name="phone\_number" required><br>

<button type="submit">Добавить</button>

</form>

</body>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script>

function fetchUsers() {

fetch('/api/users')

.then(response => response.json())

.then(data => {

let table = '<table border="1" style="width:100%; margin:auto; text-align: center;">';

table += '<tr><th>ID Арендатора</th><th>ФИО</th><th>Дата регистрации</th><th>Email</th><th>Номер телефона</th></tr>';

data.users.forEach(user => {

table += '<tr>';

user.forEach(field => {

table += `<td style="text-align: center;">${field}</td>`;

});

table += '</tr>';

});

table += '</table>';

document.getElementById('user\_table').innerHTML = table; // Обновляем таблицу

})

.catch(error => console.error('Error:', error));

}

document.getElementById('field').addEventListener('change', function() {

var selectedField = this.value;

var newValueInput = document.getElementById('new\_value');

if (selectedField === 'Номер\_Телефона') {

$('#new\_value').mask("+7 (999) 999-99-99");

} else {

newValueInput.removeAttribute('pattern'); // Удаляем паттерн для других полей

}

});

window.onload = fetchUsers;

</script>

''')

@app.route('/api/users')

def get\_users():

conn = connect\_to\_access\_db('./Database1.accdb')

users = fetch\_all\_users(conn)

conn.close()

return jsonify(users = [list(user) for user in users])

@app.route('/update\_user', methods=['POST'])

def update\_user():

user\_id = int(request.form['user\_id'])

field = request.form['field']

new\_value = request.form['new\_value']

conn = connect\_to\_access\_db('./Database1.accdb')

try:

update\_table(conn, user\_id, field, new\_value)

except Exception as e:

return jsonify(error = str(e))

finally:

conn.close()

return redirect('/')

@app.route('/delete\_user', methods=['POST'])

def delete\_user():

user\_id = int(request.form['user\_id'])

conn = connect\_to\_access\_db('./Database1.accdb')

try:

delete\_table(conn, user\_id)

except Exception as e:

return jsonify(error = str(e))

finally:

conn.close()

return redirect('/')

@app.route('/insert\_user', methods=['POST'])

def insert\_user():

full\_name = request.form['full\_name']

registration\_date = request.form['registration\_date']

email = request.form['email']

phone\_number = request.form['phone\_number']

conn = connect\_to\_access\_db('./Database1.accdb')

try:

insert\_table(conn, full\_name, registration\_date, email, phone\_number)

except Exception as e:

return jsonify(error = str(e))

finally:

conn.close()

return redirect('/')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(port = 8000, debug = True)