|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Мытищинский филиал**  **Федерального государственного бюджетного образовательного учреждения**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МФ МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | Космический |
| КАФЕДРА | К1 «Системы автоматического управления» |

**КУРСОВАЯ РАБОТА**

***НА ТЕМУ:***

|  |
| --- |
| Разработка прикладного программного обеспечения |
| деятельности отдела учёта квартир |
| "Бюро технической инвентаризации" |
|  |
|  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Студент |  |  | **К1-61Б** |  |  |  | **Тимофеев К.А.** |
|  |  |  | (группа) |  | (подпись, дата) | (И.О. Фамилия) |
| Руководитель курсовой | |  |  |  |  |  |
| работы | |  |  |  |  | **Батырев Ю.П.** |
|  | |  |  |  | (подпись, дата) | (И.О. Фамилия) |
| Консультант | |  |  |  |  | **Корнеев Ф.В.** |
|  | |  |  |  | (подпись, дата) | (И.О. Фамилия) |

Мытищи, 2024 г.

**Министерство науки и высшего образования Российской Федерации**

**Мытищинский филиал**

**Федерального государственного бюджетного образовательного учреждения**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МФ МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой К1

(индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Беляев М.Ю.

(И.О. Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение курсовой работы**

|  |  |  |
| --- | --- | --- |
| по дисциплине: | | Системы управления базами данных |
| Студент группы | | К1-61Б |
| (Фамилия, имя, отчество) | | |
| Тема курсовой работы: Разработка прикладного программного обеспечения деятельности | |
| отдела учёта квартир "Бюро технической инвентаризации" | |
|  | |

Направленность КР (учебная, исследовательская, практическая, производственная, др.)

|  |  |
| --- | --- |
| учебная | |
|  | |
| Источник тематики (кафедра, предприятие, НИР): | | кафедра К1 |
| Разработать прикладное программное обеспечение деятельности отдела учета квартир | |
| "Бюро технической инвентаризации" | |
|  | |
|  | |
| ***Оформление курсовой работы:*** | |
|  | |
|  | |

Расчетно-пояснительная записка (Отчет по КР) на 10 листах формата А4.

|  |  |
| --- | --- |
|  | |
|  | |
| Дата выдачи задания | | 01.02.24 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Руководитель курсовой** |  |  |  |  |  |  |
| **работы** |  |  |  |  | **Батырев Ю.П.** |
|  |  |  |  | (подпись, дата) | (И.О. Фамилия) |
| Студент |  |  |  |  | **Тимофеев К.А.** |
|  |  |  |  | (подпись, дата) | (И.О. Фамилия) |

**Министерство науки и высшего образования Российской Федерации**

**Мытищинский филиал**

**Федерального государственного бюджетного образовательного учреждения**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МФ МГТУ им. Н.Э. Баумана)**

**КАЛЕНДАРНЫЙ ПЛАН**

**на выполнение курсовой работы**

|  |  |  |
| --- | --- | --- |
| по дисциплине: | | Системы управления базами данных |
| Студент группы: | | К1-61Б |
| (Фамилия, имя, отчество) | | |
| Тема курсовой работы:  Разработка прикладного программного обеспечения деятельности отдела учета квартир | |
| "Бюро технической инвентаризации" | |
|  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Наименование этапов  курсовой работы** | **Сроки выполнения**  **этапов** | | **Отметка о выполнении** | |
| **план** | **факт** | **Руководитель КР** | **Куратор** |
|  | Задание на выполнение курсовой  работы | 01.02.24 | 01.02.24 |  |  |
|  | 1 модуль | 01.04.24  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *Планируемая дата* | 01.04.24 |  |  |
|  | 2 модуль | 01.05.24  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *Планируемая дата* | 01.05.24 |  |  |
|  | Оформление РПЗ | 01.06.24  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *Планируемая дата* | 01.06.24 |  |  |
|  | Подготовка доклада и презентации (при необходимости) | -  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *Планируемая дата* | - |  |  |
|  | Защита курсовой работы | 06.06.24  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *Планируемая дата* | 06.06.24 |  |  |

Студент \_\_\_\_\_ Руководитель работы \_\_\_\_

(подпись, дата) (подпись, дата)

**Содержание**

[Введение 5](#_Toc21126)

[Формирование абстрактной модели 6](#_Toc28060)

[Реализация в SQLiteStudio 8](#_Toc11434)

[• Запросы на выборку 10](#_Toc4289)

[• Запросы на добавление/удаление записи 11](#_Toc18978)

[Выводы 13](#_Toc20673)

[Список литературы 14](#_Toc14737)

[Приложение 15](#_Toc2629)

# Введение

Задача работы заключается в построении прикладного программного обеспечения деятельности отдела учёта квартир "Бюро технической инвентаризации". Для этого составляется база данных, которая определит структуру и взаимосвязи между данными, которыми оперирует организация "Бюро технической инвентаризации". По логике вещей в задачи рассматриваемой системы необходимо включить учёт n-ого количества квартир в k-ом количестве домов и т.д. и т.п. Основательное описание структуры связей таблиц будет изложено далее.

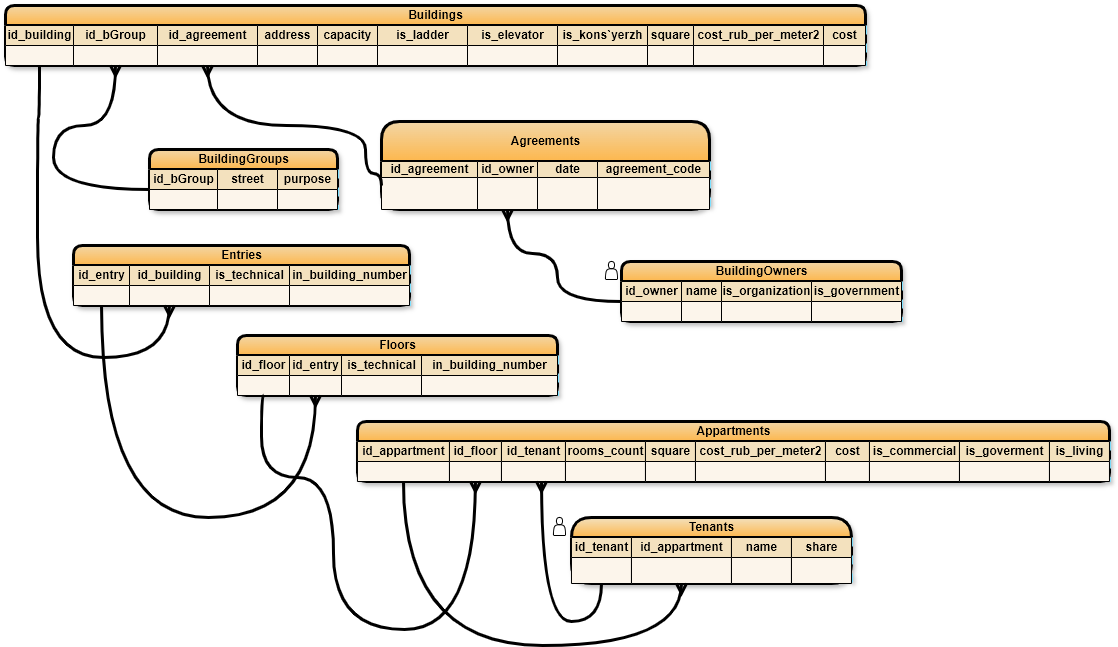
**Итоговое решение должно отвечать следующим требованиям:**

* База должна содержать не менее 8 таблиц.
* Схемы в виде рисунков к предметной области.
* Не менее 2 запросов на выборку (с сортировкой данных там, где это необходимо).
* Не менее 1 запроса на изменение.
* Не менее 2 запросов на добавление/удаление записей.
* Таблицы должны содержать не менее 10 записей.
* Запросы приводятся на SQL.

Для решения задачи было выбрано использовать в качестве СУБД SQLite – компактная встраиваемая СУБД. Движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. SQLite поддерживает динамическое типизирование данных. Возможные типы значений: **INTEGER**, **REAL**, **TEXT** и **BLOB**. Также поддерживается специальное значение **NULL**.

# Формирование абстрактной модели

Абстрактно представить любую базу данных можно как набор нескольких тем или иным образом связанных между собой таблиц данных. В свою очередь таблицы с данными представляют собой несколько столбцов и несколько строк, в заголовке каждой таблицы укажем осмысленные зависимости и характеристики объектов, которые будет описывать таблица. На нижеприведённой схеме представлена структурная схема базы данных для "Бюро технической инвентаризации".

Здесь стоит отметить принципы, по которым связываются между собой таблицы и соответственно данные в них. Связи в базе данных рождаются из нужд пользователя и от входных данных. В нашем случае существует восемь таблиц, в каждой есть некий набор данных, причём к каждой записи цепляется идентификатор, выделенный в отдельном столбце **primary key**. Из каждого идентификатора «ветками» выходят абстрактные связи таблиц, и как видно из иллюстрации, каждый первичный ключ находит себе в пару такой же идентификатор, который в свою очередь называют **foreign key** или внешний ключ. По смыслу применения первичный ключ в одной таблице не повторяется и как правило просто увеличивается на единицу с каждой новой записью. Внешние ключи ограничени не имеют, их значения могут повторяться. Отсюда и появляются связи связи таблицами.

Они бывают четырёх видов:

* ***ОДИН-к-ОДНОМУ*** (там, где идентификатор одной таблицы соответствует идентификатору другой. По сути такую связь всегда можно реализовать через объединение таблиц.)
* ***МНОГИЕ-к-ОДНОМУ*** (множество записей одной таблицы связаны с одной записью другой таблицы)
* ***ОДИН-ко-МНОГИМ*** (то же, что и МНОГИЕ-к-ОДНОМУ, но наоборот)
* ***МНОГИЕ-ко-МНОГИМ*** ( в обоих таблицах есть внешние ключи на друг друга, поэтому для одной квартиры может быть несколько жильцов, среди которых может быть кто-то, владеющий ещё одной ( или более) квартирой).

Внутри таблицы также можно составить множество условий построения данных. Банально нужно для каждой колонки указывать тип данных, хотя в самой SQLite это необязательно, если значения могут быть вычислены из введённых ранее данных, это можно реализовать внутри SQL-запроса с помощью математических функций, если данные загружаются неравномерно для случаев упущения значений можно указать специальный тип NULL или не допускать таких записей вообще, записи могут повторяться или не повторяться, …, и т.п. Иными словами, способов настройки таблицы существует великое множество. Приведём в пример один из запросов на создание таблицы Tenants:

**CREATE TABLE** Tenants **(**

id\_tenant **INTEGER PRIMARY KEY ASC AUTOINCREMENT**

**REFERENCES** Appartments **(**id\_tenant**) UNIQUE,**

id\_appartment **INTEGER REFERENCES** Appartments**(**id\_appartment**),**

name\_tenant **TEXT,**

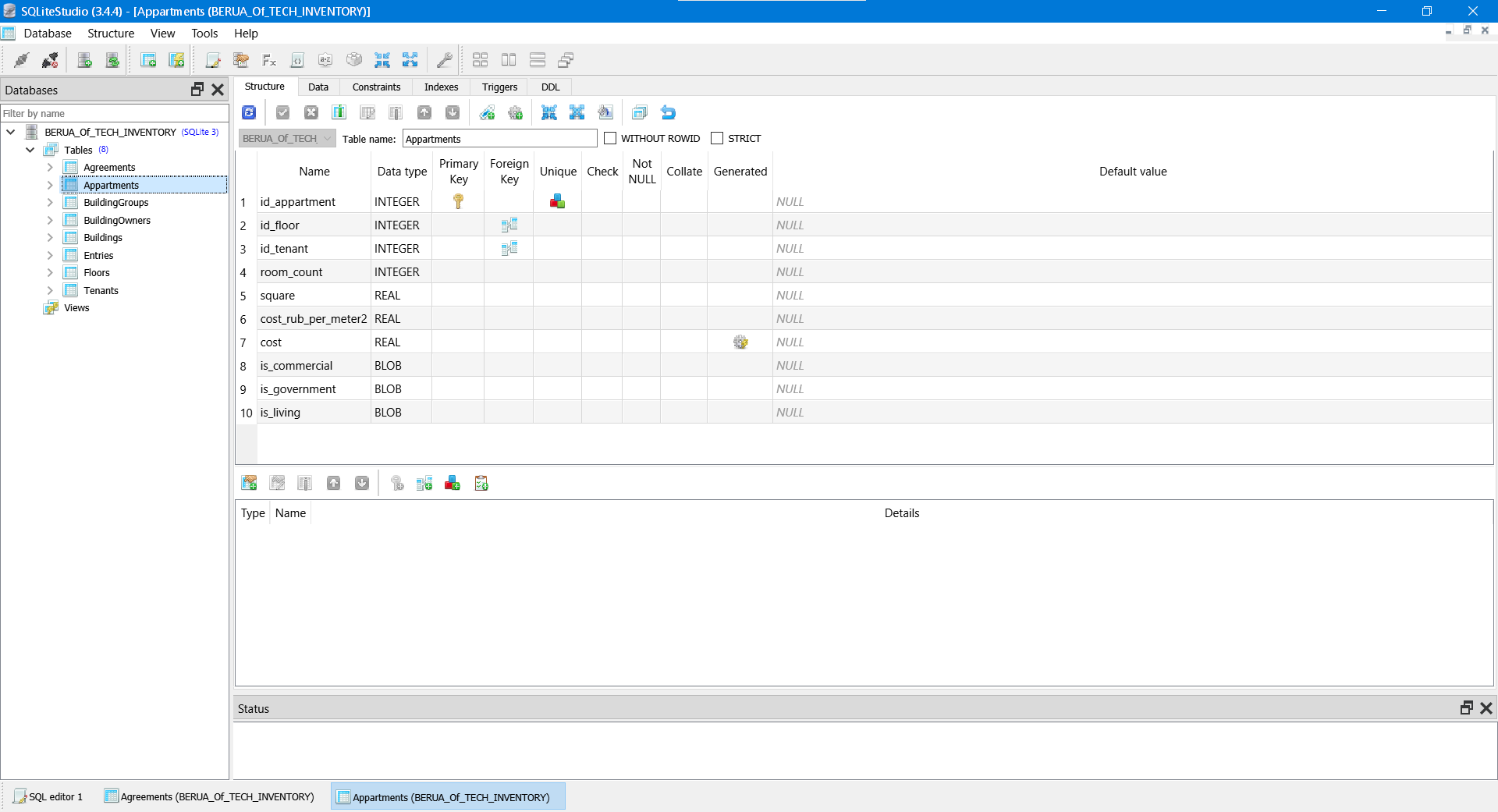
share **INTEGER);**

\*здесь заглавными буквами набраны встроенные функции языка SQL запросов, прописными ссылки на элементы базы данных.

Следуя вышеописанным указаниям, менеджер базы данных создаёт таблицу Tenants, связанную с таблицей Appartments, как МНОГИЕ-ко-МНОГИМ, по первичному ключу id\_tenant и вторичному id\_appartment. Также в Tenants создаются столбцы имени владельца квартиры и его доли на этой жилплощади.

# Реализация в SQLiteStudio

**SQLiteStudio** – это настольное приложение для просмотра и редактирования файлов базы данных SQLite. Он предназначен для людей, которые знают, что такое SQLite или что такое реляционные базы данных в целом. Иными словами для простоты взаимодействия с базой данных на низком уровне будем использовать именно этот менеджер баз данных SQLite.



8 таблиц БД

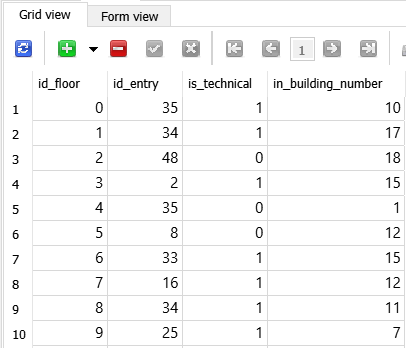
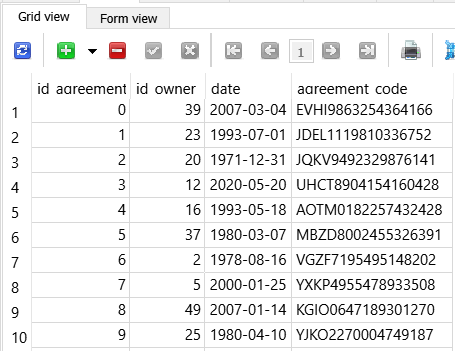
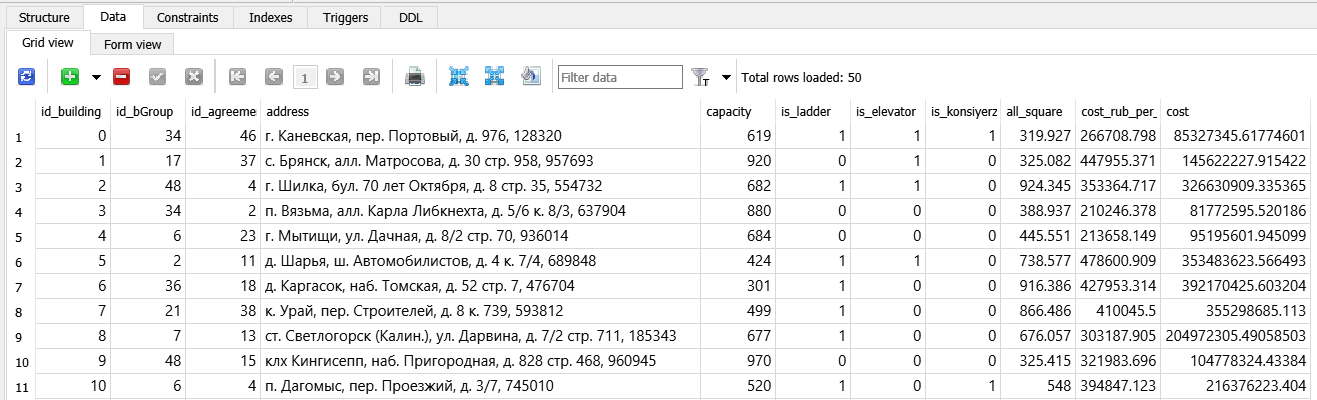
БД «Тех. инвентаризации»

Таблица квартир (апартаментов)

В SQLiteStudio используя соответствующий интерфейс создаём базу данных согласно схеме, указанной в предыдущем рассуждении, и соблюдая иерархию связей.

Более интересной задачей является заполнение базы данных. Очевидно, что в реальных задачах данные не берут из воздуха, однако в случае данной работы данные необходимо как-то искусственно сгенерировать и загрузить в базу. Для этого напишем генератор данных на языке Python. Листинг программы будет приведён в приложении к курсовой работе. Здесь же можно сказать, что взаимодействие с базами данных реализовано в библиотечном виде на всех популярных языках программирования. В случае Python используется модуль **sqlite3**. Необходимо создать соединение с базой данных, затем создать курсор, которым будем посылать запросы на загрузку сгенерированных данных. Генерацией данных займётся модуль **faker** – свободно распространяемый генератор псевдоосмысленной информации по типу: имён, адресов, банковских счетов и т.п. Модуль не слишком хорошо работает над русифицированной информацией, поэтому смоделировать очень много данных не получиться, но всё же это быстрее, чем вручную.

Некоторые заполненные таблицы из готовой базы данных:



В тексте уже были упомянуты SQL запросы на создание таблиц базы данных и заполнение ячеек данных, распространим рассуждения на эту тему примерами.

Как правило, непосредственно с базой данных никто не работает, люди, в работе которых необходима возможность получать данные из базы, не прочёсывают таблицу за таблицей и строку за строкой в поисках нужной информации, для выборок, или отчётов, существуют запросы **SELECT**. С их помощью можно реализовать определённой условием выбора просмотр информации из базы данных SQL.

Укажем общий синтаксис запросов **SELECT**:

**SELECT ('столбцы или \* для выбора всех столбцов; обязательно')**

**FROM ('таблица; обязательно')**

**WHERE ('условие/фильтрация, например, city = 'Moscow'; необязательно')**

**GROUP BY ('столбец, по которому хотим сгруппировать данные; необязательно')**

**HAVING ('условие/фильтрация на уровне сгруппированных данных; необязательно')**

**ORDER BY ('столбец, по которому хотим отсортировать вывод; необязательно')**

В требованиях к работе указаны обязательные запросы SQL, ниже по пунктам обсудим и продемонстрируем их реализацию применительно к инициализированной базе данных.

* **Запросы на выборку**

Следующий запрос создаёт выборку для сводной таблицы из стоимости квартиры и адресов домов, в котором они расположены, по совместительству сортируя по убыванию стоимости.

**SELECT** Buildings.address, Appartments.square, Appartments.cost

**FROM (**Buildings **INNER JOIN**

**(**Entries **INNER JOIN**

**(**Floors **INNER JOIN**

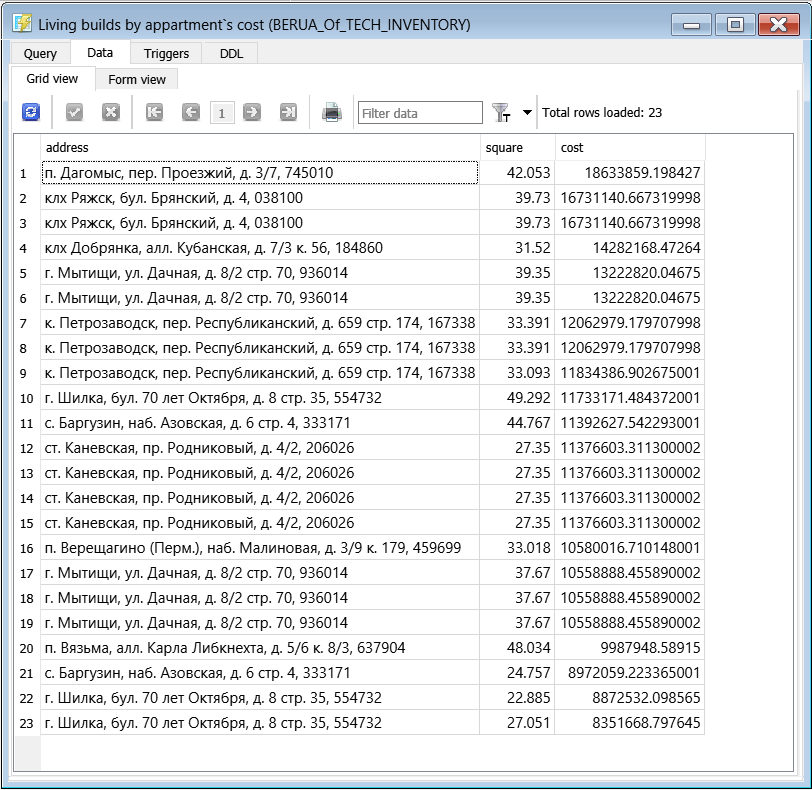
Appartments **ON** Floors.id\_floor **=** Appartments.id\_floor**)**

floor **ON** Entries.id\_entry **=** floor.id\_entry**)**

entry **ON** Buildings.id\_building **=** entry.id\_building**)**

**WHERE** Appartments.is\_living **=** **True**

**ORDER BY** Appartments.cost **DESC;**



Запрос на выборку информации по имени владельца квартиры.

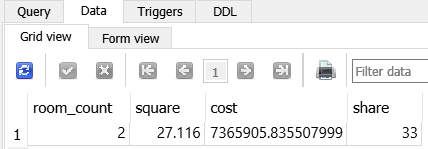
**SELECT** Appartments.room\_count**,** Appartments.square**,**

Appartments.cost**,** Tenants.share

**FROM (**Appartments **INNER JOIN** Tenants **ON**

Appartments.id\_tenant **=** Tenants.id\_tenant**)**

**WHERE** Tenants.name\_tenant **=** **"Шилов Евстафий Фролович";**



* **Запросы на добавление/удаление записи**

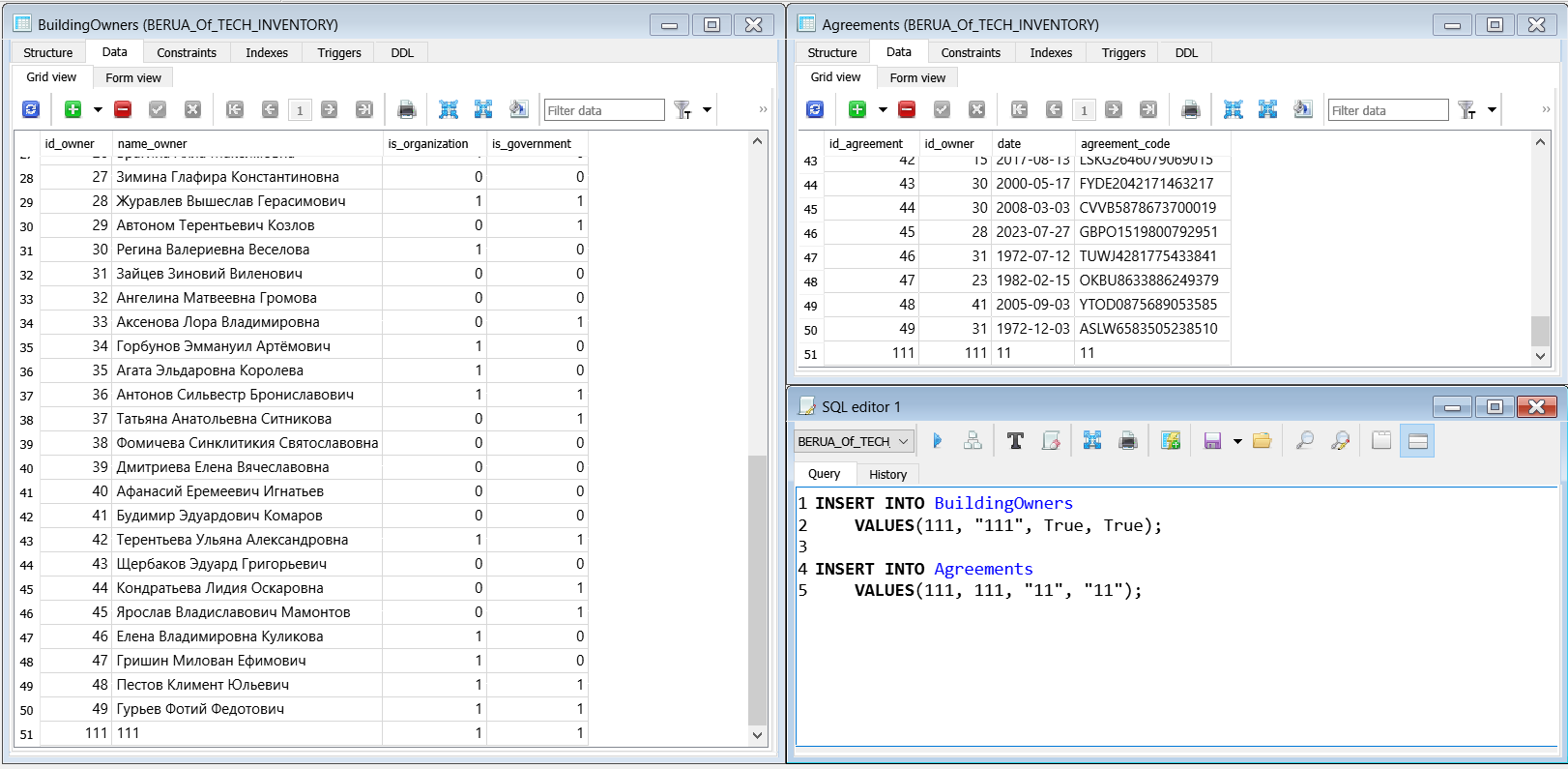
*Образец добавления записи.*

**INSERT INTO** BuildingOwners

**VALUES(111**, "111", True, True**);**

**INSERT INTO** Agreements

**VALUES(111**, **111**, "11", "11"**);**



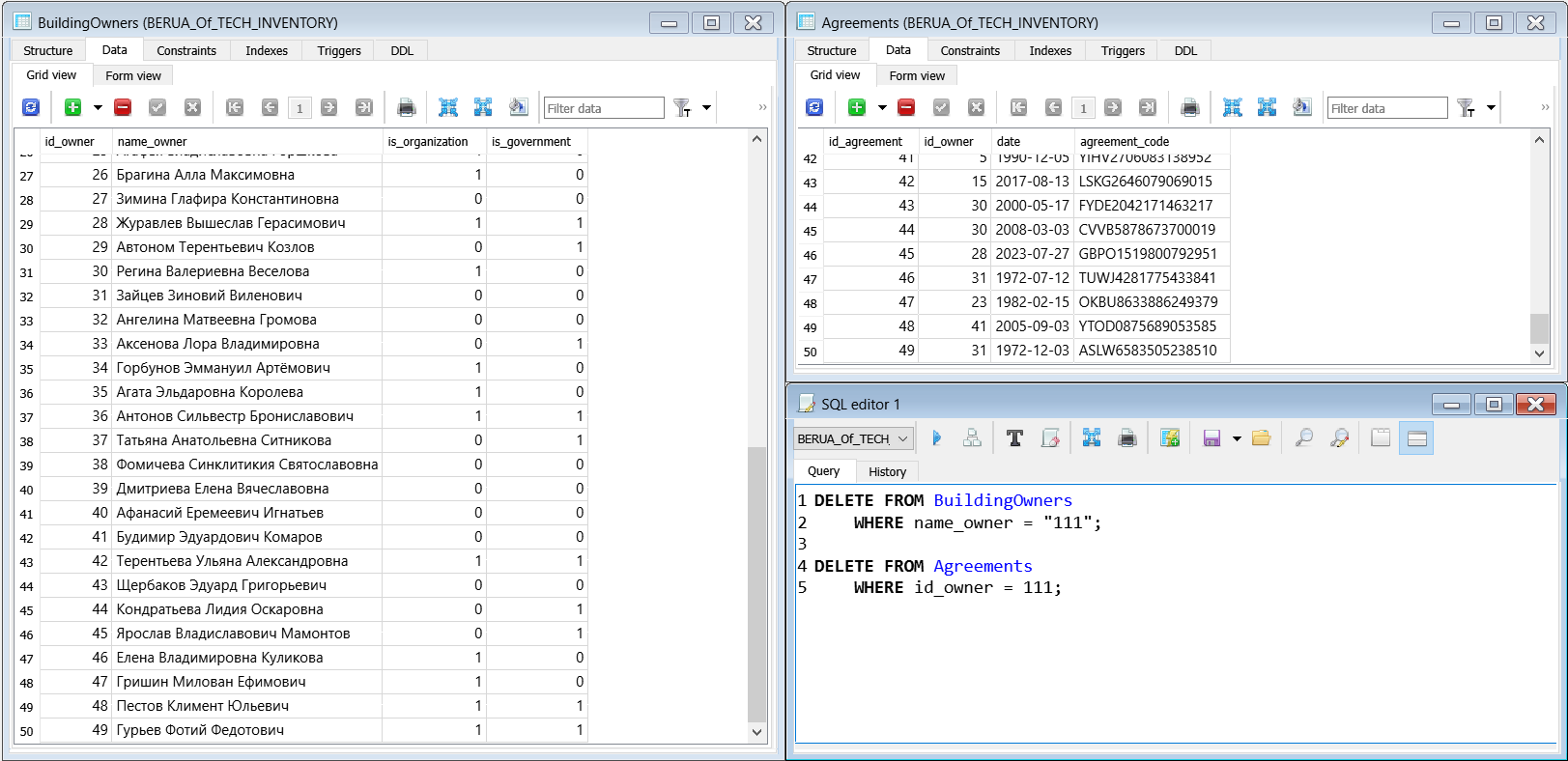
*Образец удаления записи.*

**DELETE FROM** BuildingOwners

**WHERE** name\_owner **= "111";**

**DELETE FROM** Agreements

**WHERE** id\_owner **= 111;**

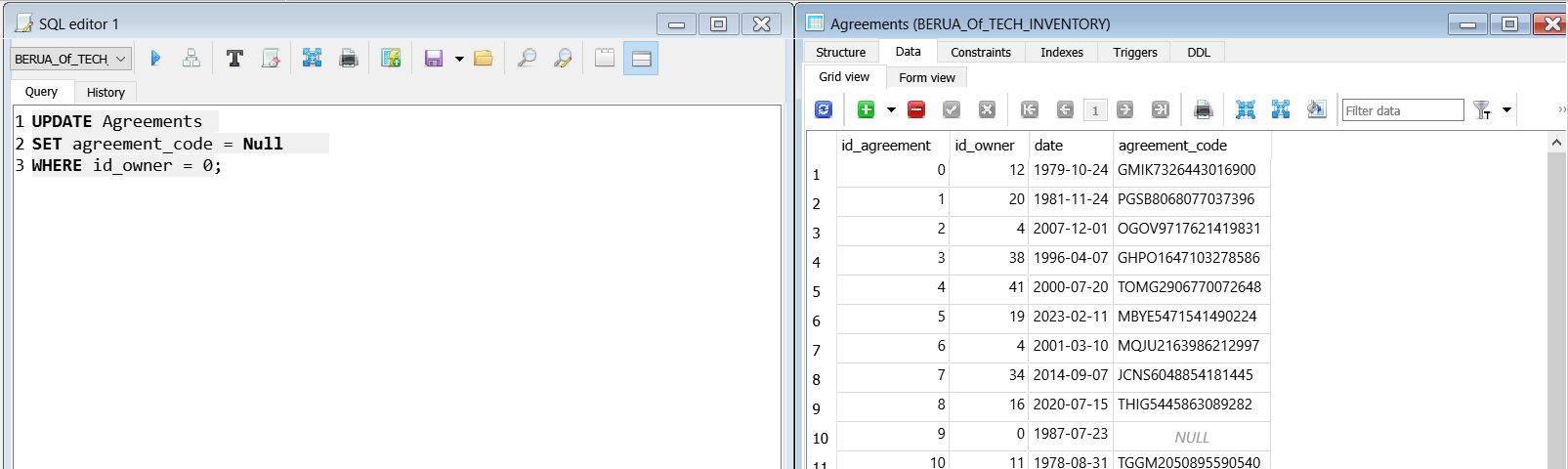


*Обновление значений.*

**UPDATE** Agreements

**SET** agreement\_code **= Null**

**WHERE** id\_owner **= 0;**



# Выводы

Вышеописанная работа показывает процессы создания, редактирования и настройки базы данных «Бюро технической инвентаризации». Стоит отметить, что с объективной стороны заданным требованиям проект отвечает. По итогу мы имеем макет СУБД, представляющий интерфейс для взаимодействия с достаточно большим объёмом данных, однако в выводе не следует повторять тезисы уточнённые в самом тексте работы, вместо этого скажем, что поскольку основные принципы работы с базами данных, такие как отношения таблиц, типы данных базы, представление простейших отчётов и т.д., следующим шагом в смысле практического применения полученных навыков и работы с СУБД будет создание программного обеспечение, специального для конкретной базы данных. Так, например, для исследуемого случая можно реализовать приложение, ограничивающее взаимодействие пользователя с базой данных посредством заранее “разрешённых” отчётов. Программа должна будет иметь ряд определённых SQL-запросов, параметры которых пользователь сможет изменять в некоторой степени свободы. Именно такие приложения в итоге используют на практике в организациях. В реальной ситуации никто не взаимодействует с SQL напрямую, так как иначе даже рядовым бухгалтерам пришлось бы учить программирование. В данной работе такую программу мы не описали, однако полученных сведений уже достаточно для её реализации.

# Список литературы

1. Диго, С. М. Базы данных: проектирование и использование : учебник для

вузов / С. М. Диго. – М.: Финансы и статистика, 2005.

1. Базы данных. Учебник для высших учебных заведений / под редакцией

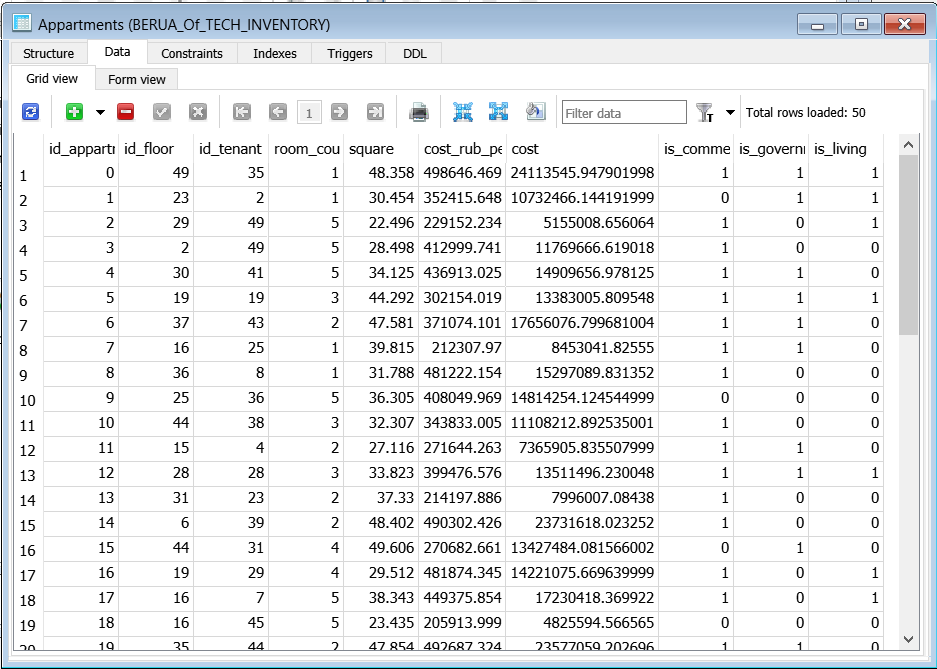
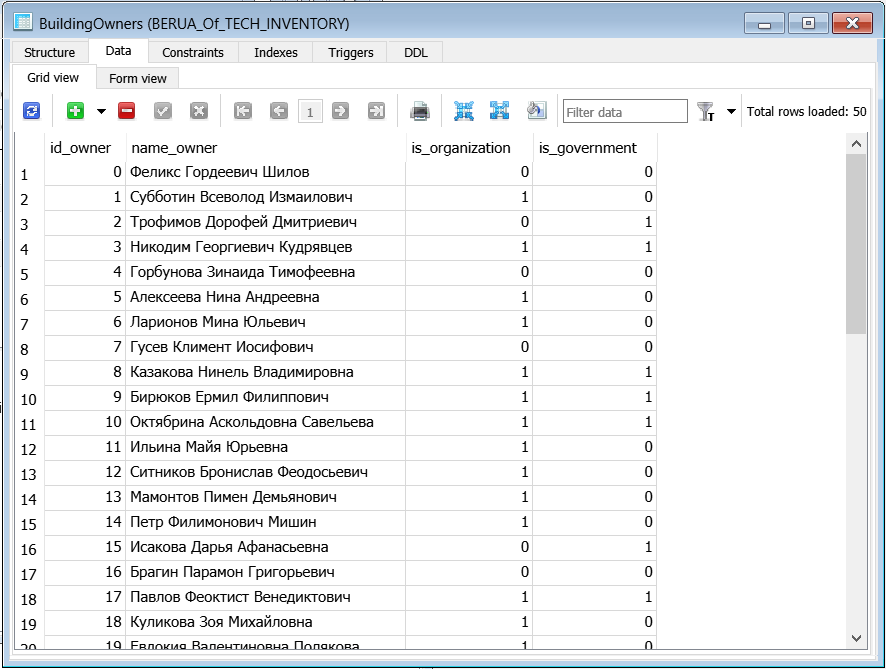
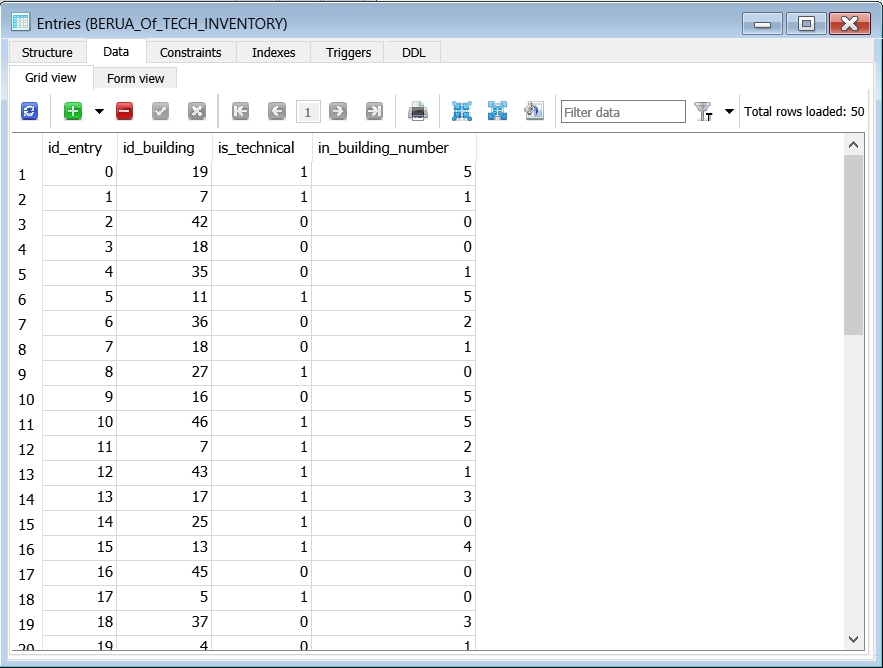
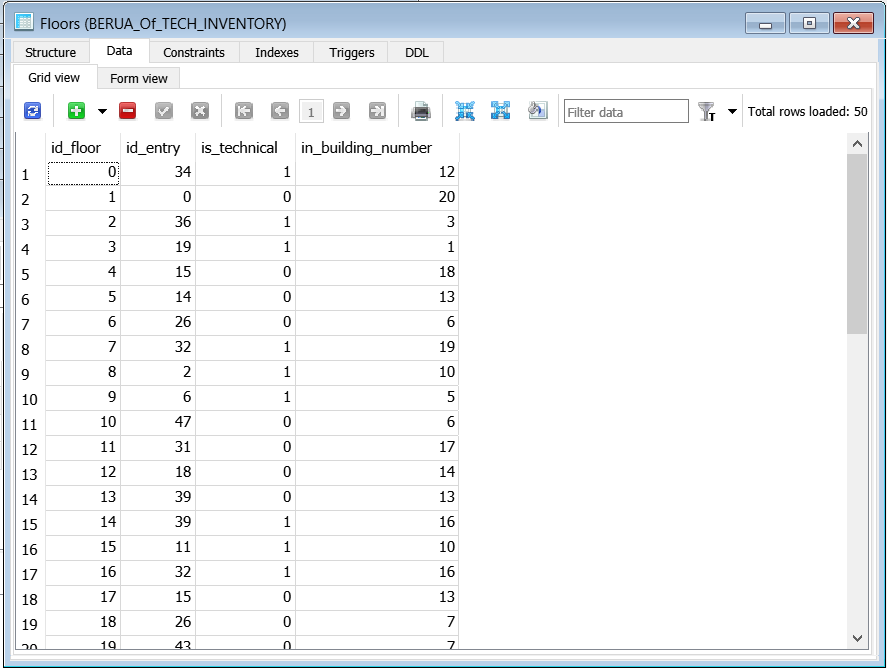
проф. А.Д. Хомоненко, СПб.: Корона, 2004.

1. Дейт, К.Дж. Введение в системы баз данных : [пер. с англ.] / Дейт К.Дж. – 8-е изд. - Москва ; СПб. ; Киев : Издательский дом «Вильямс», 2005. – 1327 с.
2. Гурвиц, Г.А. Microsoft Access 2010. Разработка приложений на реальном примере / Гурвиц Г.А. – СПб.: БХВ-Петербург, 2010.
3. Крёнке, Д. Теория и практика построения баз данных. 9-е изд./ Крёнке Д. – СПб.: Издательский дом «Питер», 2005.

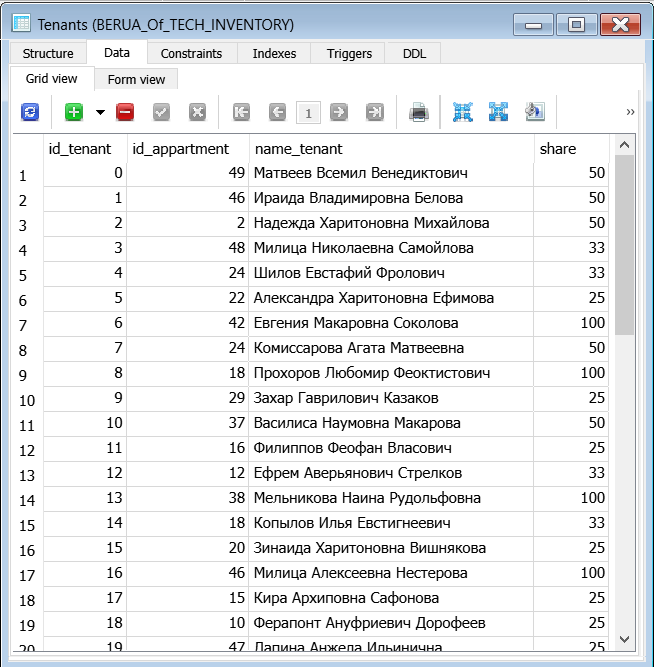
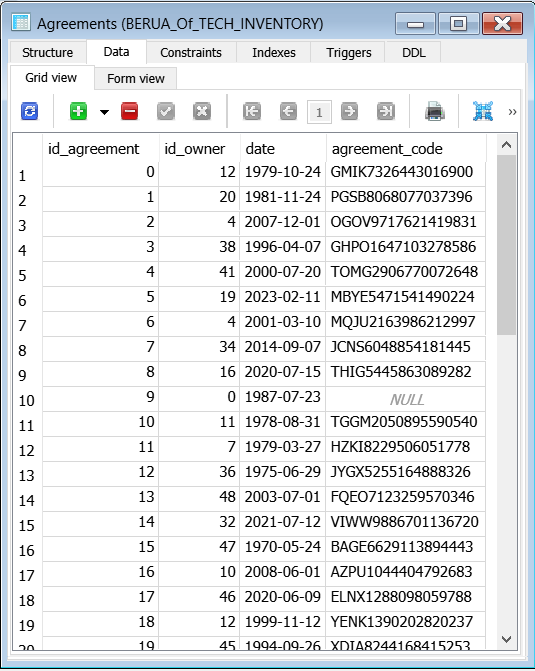
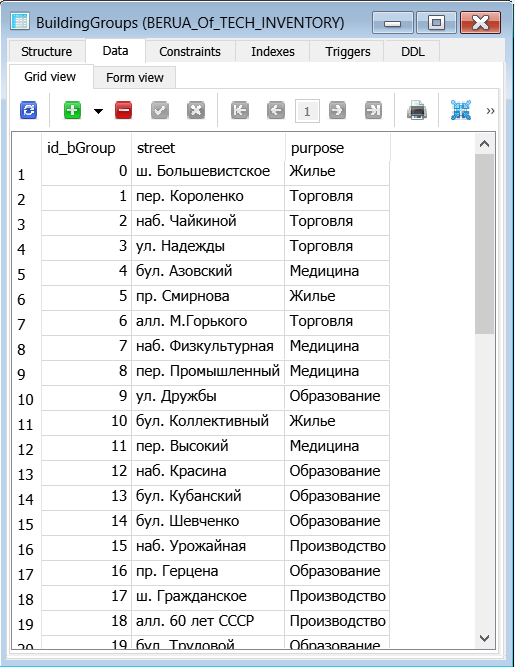
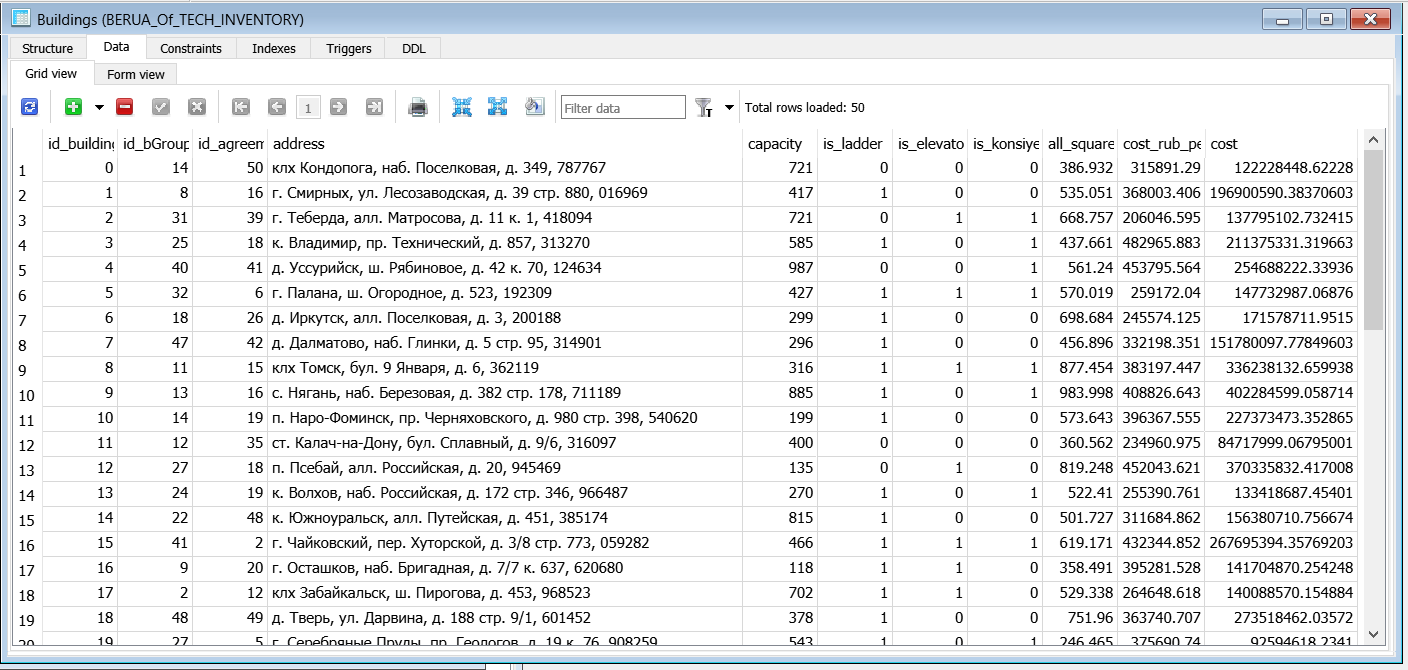
# Приложение

* **Представление всех составленных таблиц в режиме оконного просмотра в SQLiteStudio.**

Таблицы, в обрезанном виде: Группы зданий, Квартиросъёмщики, Договора, Здания...



..., Собственники Зданий, Квартиры, Подъезды, Этажи.



* **Код программы заполнения полей данных на Python.**

import pandas as pd

from tabulate import tabulate

from faker import \*

import random

import sqlite3

conn = sqlite3.connect("C:\\Qt\\KursovayaDB\_Timofeev\_K161B\\DB\\test\\BERUA\_Of\_TECH\_INVENTORY.sqlite")

c = conn.cursor()

tables = {

    'Appartments':[],

    'Agreements':[],

    'Buildings':[],

    'BuildingOwners': [],

    'BuildingGroups': [],

    'Tenants':[],

    'Floors': [],

    'Entries' : []

}

str\_columns = [] # columns with text data type

for i in tables.keys():

    c.execute(f'PRAGMA table\_info({i})')

    column\_names = [ i[1] for i in c.fetchall()]

    c.execute(f'PRAGMA table\_info({i})')

    column\_types = [ i[2] for i in c.fetchall()]

    c.execute(f'PRAGMA table\_info({i})')

    column\_isPkey = [ i[5] for i in c.fetchall()]

    for k, j, m in zip(column\_names, column\_types, column\_isPkey):

        tables[i].append(tuple([k, j, m]))

        if j == 'TEXT':

            str\_columns.append(k)

types = {'INTEGER' : int, 'REAL' : float, # helps to convert sql types to python types

         'TEXT' : str, 'BLOB' : bool}

n : int = 50 # rows count

fake = Faker('ru\_RU')

id = 0

for tab\_name in tables.keys():

    conn.commit()

    list\_of\_values : list[tuple] = []

    questions = '?'

    c.execute("DELETE FROM " + tab\_name)

    id = 0

    for \_ in range(n):

        value\_row : tuple = ()

        for col\_tup in tables[tab\_name]: # col\_tup = (col\_name, col\_dtype)

            match col\_tup[1]: # match column types

                case 'TEXT' : # match text

                    if 'name' in col\_tup[0]: value\_row += (fake.name(), )

                    elif 'address' in col\_tup[0]:  value\_row += (fake.address(), )

                    elif 'street' in col\_tup[0]:  value\_row += (fake.street\_name(), )

                    elif 'date' in col\_tup[0]: value\_row += (fake.date(), )

                    elif 'agreement' in col\_tup[0]: value\_row += (fake.bban(), )

                    elif 'purpose' in col\_tup[0]: value\_row += (random.choice(['Образование', 'Производство', 'Жилье', 'Медицина', 'Торговля']), )

                case 'BLOB' :

                    value\_row += (random.choice([True, False]), )

                case 'REAL' :

                    if 'square' == col\_tup[0]: value\_row += (round(random.uniform(20.0, 50.0), 3), )

                    elif 'all\_square' == col\_tup[0]: value\_row += (round(random.uniform(200.0, 1000.0), 3), )

                    else : value\_row += (round(random.uniform(200000.0, 500000.0), 3), )

                case 'INTEGER' :

                    if 'id' in col\_tup[0] and col\_tup[2] == 1:

                        value\_row += (id, )

                        id += 1

                    elif 'id' in col\_tup[0]: value\_row += (random.randint(0, n), )

                    elif 'capacity' == col\_tup[0]: value\_row += (random.randint(100, 1000), )

                    elif 'share' == col\_tup[0] : value\_row += (random.choice([50, 100, 33, 25]), )

                    elif 'Etries' == tab\_name and  'in\_building\_number' == col\_tup[0] or 'room\_count' == col\_tup[0]: value\_row += (random.randint(1, 5), )

                    elif 'Floors' == tab\_name and  'in\_building\_number' == col\_tup[0]: value\_row += (random.randint(0, 20), )

                    else : value\_row += (random.randint(0, 5), )

                case \_ :

                    value\_row += (random.randint(0, n), )

        list\_of\_values.append(value\_row)

    print(tab\_name)

    print(list\_of\_values)

    for \_ in range(len(tables[tab\_name])-1):

        questions+=', ?'

    print(questions)

    c.executemany(f'INSERT INTO {tab\_name} VALUES ({questions})', list\_of\_values)

conn.commit()

Скрипт вернёт полностью заполненную БД (50 записей для каждой таблицы) и консольный листинг из подобных сообщений о введённых в базу значениях .

?, ?, ?, ?

Floors

[(0, 29, False, 18), (1, 22, False, 2), (2, 20, False, 18), (3, 40, True, 20), (4, 29, True, 18), (5, 23, False, 2), (6, 9, True, 14), (7, 44, False, 15), (8, 28, False, 10), (9, 30, False, 1), (10, 2, False, 4), (11, 47, False, 17), (12, 19, False, 2), (13, 29, True, 19), (14, 25, False, 2), (15, 32, False, 18), (16, 50, True, 11), (17, 43, False, 14), (18, 21, True, 12), (19, 35, False, 12), (20, 44, True, 1), (21, 24, False, 18), (22, 8, False, 9), (23, 13, True, 11), (24, 25, False, 4), (25, 38, True, 16), (26, 10, False, 12), (27, 3, True, 2), (28, 31, True, 15), (29, 40, True, 16), (30, 36, False, 7), (31, 21, True, 16), (32, 38, False, 0), (33, 40,

True, 7), (34, 48, False, 12), (35, 26, True, 5), (36, 24, False, 14), (37, 50, True, 7), (38, 31, False, 12), (39, 19, False, 12), (40, 49, False, 12), (41, 42, False, 19), (42, 37, True, 4), (43, 29, False, 20), (44, 37, True, 19), (45, 38, True, 8), (46, 43, True, 18), (47, 34, True, 2), (48, 1, False, 17), (49, 26, False, 14)]?, ?, ?, ?

Entries

[(0, 1, False, 0), (1, 19, True, 5), (2, 5, True, 0), (3, 38, False, 5), (4, 6, False, 3), (5, 36, False, 0), (6, 10, False, 5), (7, 17, True, 0), (8, 1, False, 2), (9, 25, False, 1), (10, 30, True, 3), (11, 5, False, 0), (12, 39, True, 5), (13, 13, True, 5), (14, 24, False, 2), (15, 42, False, 3), (16, 25, True, 1), (17, 35, False, 5), (18, 19, True, 1), (19, 49, False, 3), (20, 28, False, 4), (21, 14, True, 5), (22, 1, False, 4), (23, 42, False, 2), (24, 2, True, 3), (25, 20, True, 2), (26,

45, False, 3), (27, 4, False, 1), (28, 9, False, 3), (29, 20, True, 1), (30, 11, True, 0), (31, 21, True, 0), (32, 48, False, 0), (33, 2, False, 5), (34, 37, True, 4), (35, 25, False, 3), (36, 37, True, 4), (37, 37, True, 3), (38, 44, True, 3), (39, 31, False, 1), (40, 37, True, 0), (41, 20, True, 5), (42, 23, True, 0), (43, 12, False, 4), (44, 39, True, 1), (45, 35, False, 4), (46, 22, True, 4), (47, 40, True, 5), (48, 29, True, 4), (49, 38, False, 5)]

?, ?, ?, ?