Министерство образования Республики Беларусь

Учреждение образования

“Белорусский государственный университет

информатики и радиоэлектроники”

Факультет информационных технологий и управления

Кафедра интеллектуальных информационных технологий

**Лабораторная работа №2**

по дисциплине «Проектирование баз знаний»

Вариант №26

Выполнил: Якубович Н.И.

Группа: 021703

Проверил: Марковец В. С.

Минск 2022

**Содержание:**

1. Постановка задачи 3

2. Концептуальное проектирование 4

3. Логическое проектирование 5

4. Физическое проектирование 6

5. Тексты основных запросов, функций, процедур и триггеров 7

6. Интерфейс приложения: Основные окна с описанием 9

7. Инструментальные средства, которые использованы при разработке 10

1. **Постановка задачи:**

**Предметная область:** Обучение на курсах

**Текстовое описание предметной области:** Организация (код, название, адрес, телефон, электронный адрес) проводит курсы. Каждый курс имеет код, название, тип (информационные технологии, менеджмент и т.п.), количество дней обучения, количество обучаемых, цену, цену с учетом 20% НДС. Цена со временем может меняться. Цена устанавливается соответствующим документом (номер, дата, цена). Занятия проводят преподаватели (номер, ФИО, дата рождения, пол, образование, категория – высшая, первая, вторая). Закрепление преподавателей за курсами осуществляется с помощью документа, в котором указано какой курс будет проводить данный преподаватель, дата начала обучения, дата окончания. В выходные дни – суббота, воскресение – занятия не проводятся. Организация принимает заявки на обучение (обучение проводится только по заявкам и только для организаций), в которой д.б. указаны: название организации, отправляющей сотрудников на обучение, адрес, телефон, электронная почта: количество человек, о каждом сообщается ФИО, должность; на какие курсы и в какой срок необходимо обучение.

**Необходимо реализовать выполнения следующих функций**:

* Добавление/редактирование/удаление информации об организациях
* Добавление/редактирование/удаление информации о курсах.
* Добавление информации об изменении стоимости обучения на курсах.
* Добавление/редактирование/удаление информации о преподавателях.
* Добавление и редактирование заявки на обучение.
* Просмотр прайс-листа организации на заданную дату – перечень курсов, количество дней, цена, цена с учетом НДС.
* Просмотр расписания заданного преподавателя за заданный период – название курса, дата начала, дата окончания.
* Просмотр для заданного (выбранного) курса, за заданный период наполнение групп – полностью ли набраны?

1. **Концептуальное проектирование.**

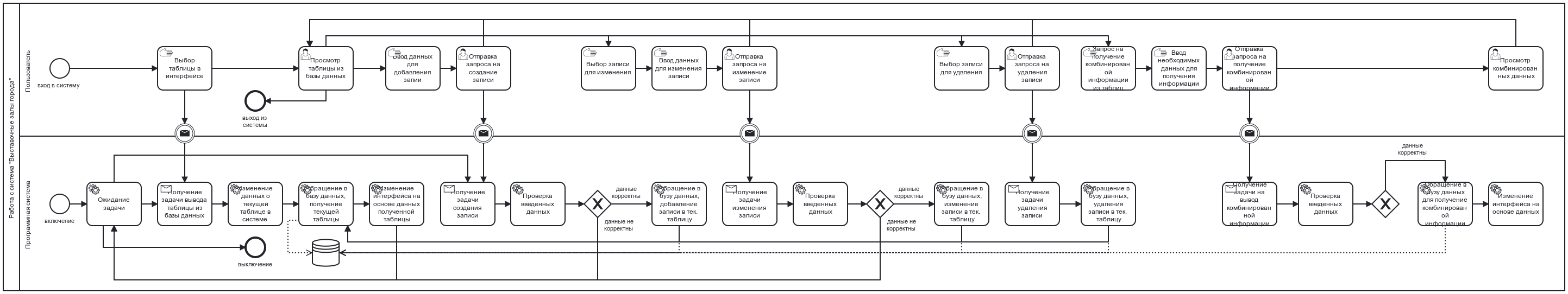
[](https://drive.google.com/file/d/1mNzCya-7y40-oYH8Fd-1NsQz-Pi_l2qU/view?usp=sharing)

Диаграмма бизнес-процессов (нотация BPMN). Рисунок 1.

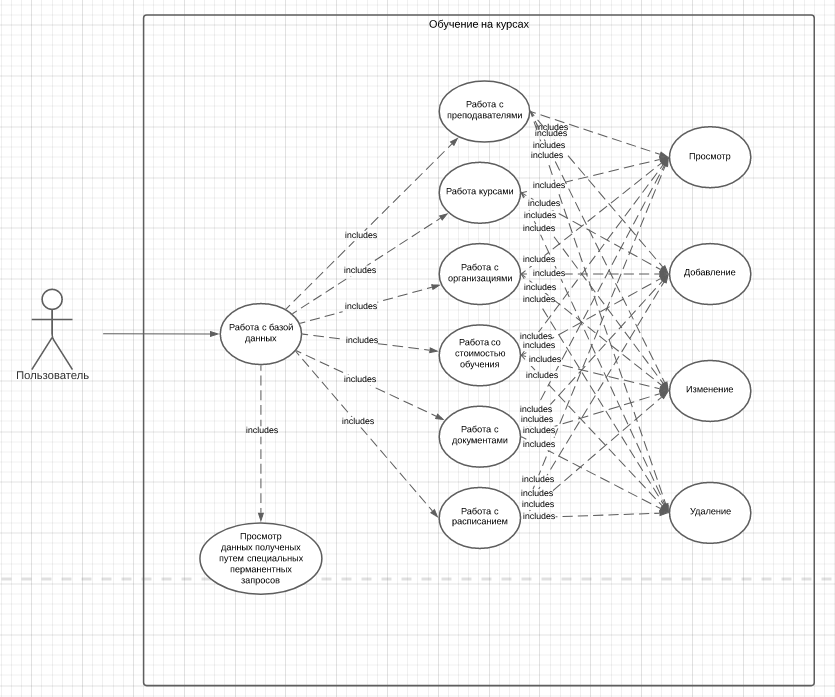
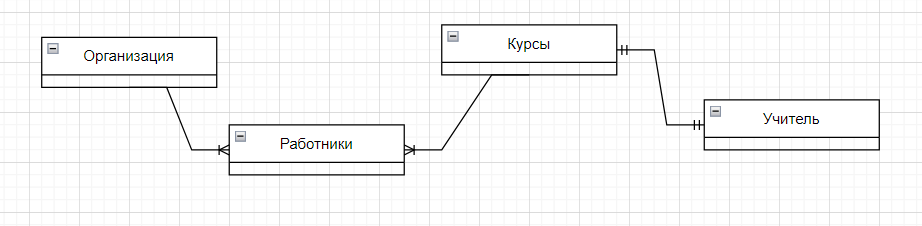


Диаграмма вариантов использования (Use Case). Рисунок 2.

1. **Логическое проектирование.**

**Логическое проектирование: Диаграмма сущность-связь (ER-диаграмма)**



.Рисунок 3.

1. **Физическое проектирование**.

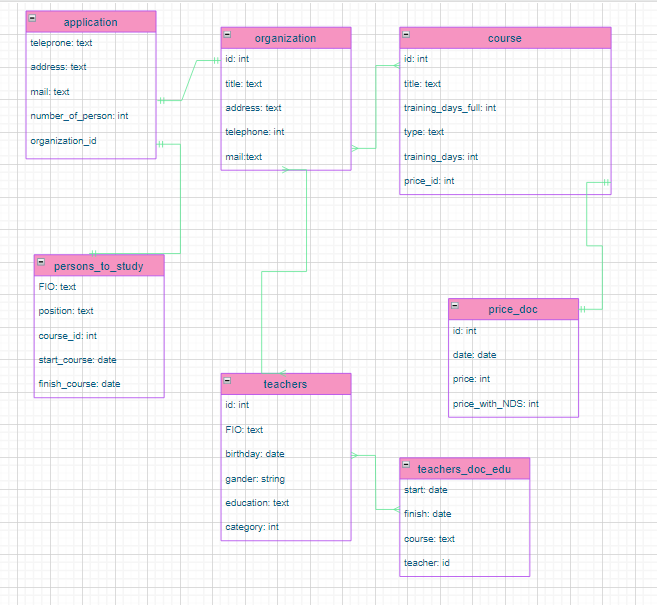


Схема БД. Рисунок 4.

1. **Тексты основных запросов, функций, процедур и триггеров.**

Лабораторная работа была реализована на основе паттерна MVC.

В виде Model выступает класс SQLiteDataBase, связывающий программу с базой данной и выполняющий запросы к ней.

def executeSQLiteQuery(self, query):  
 try:  
 self.cursor.execute(query)  
 record = self.cursor.fetchall()  
 return record  
 except sqlite3.Error as error:  
 print("The request cannot be executed:", error)

def saveChanges(self):  
 try:  
 self.sqlite\_connection.commit()  
 except sqlite3.Error as error:  
 print("The request cannot be executed:", error)  
  
def \_\_del\_\_(self):  
 if self.cursor:  
 self.cursor.close()  
 if self.sqlite\_connection:  
 self.sqlite\_connection.close()  
 print("Connection to SQLite is closed.")

В виде Controller выступает класс AppController, в котором находятся все основные функции приложения.

def getTables(self, \*args):  
 query = "SELECT name FROM sqlite\_master WHERE type ='table' AND name NOT LIKE 'sqlite\_%';"  
 self.tables = self.database.executeSQLiteQuery(query=query)

def updateCurrentRecords(self):  
 self.current\_columns = self.database.executeSQLiteQuery(f"PRAGMA table\_info({self.current\_table\_name})")  
 self.current\_records = self.database.executeSQLiteQuery(f"SELECT \* FROM {self.current\_table\_name}")

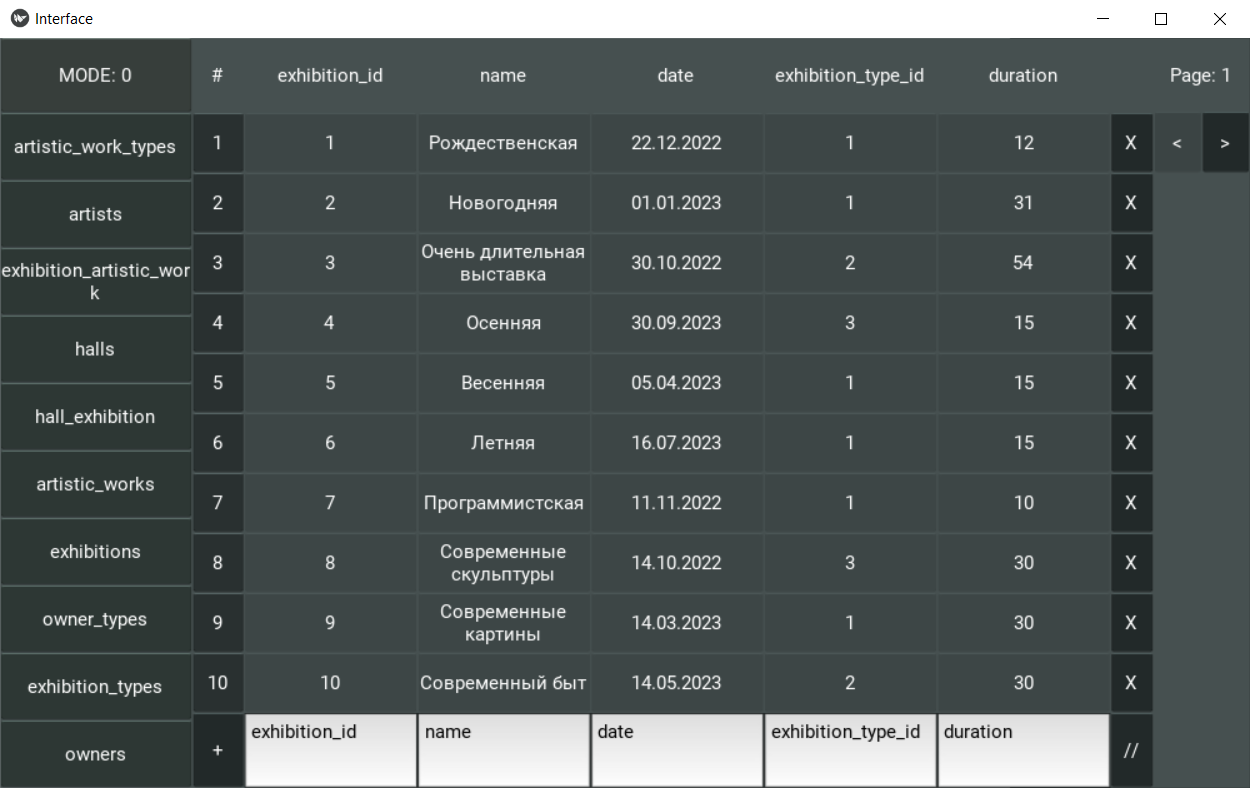
def addNewRecord(self, \*args):  
 if not self.current\_table\_name:  
 return  
 columns\_str = ""  
 values\_str = ""  
 for index in range(0, len(self.current\_columns)):  
 value = f"{args[0][index].text}"  
 try:  
 if self.current\_columns[index][2] == "INTEGER":  
 value = int(value)  
 if self.current\_columns[index][2] == "REAL":  
 value = float(value)  
 except Exception as ex:  
 print(ex)  
 return  
 value = f"'{value}'"  
 values\_str += f"{str(value)}, "  
 columns\_str += f"{self.current\_columns[index][1]}, "  
 values\_str = values\_str[0:-2]  
 columns\_str = columns\_str[0:-2]  
 query = f"INSERT INTO {self.current\_table\_name} ({columns\_str}) VALUES({values\_str});"  
 self.database.executeSQLiteQuery(query=query)  
 self.saveChangesInTable()  
 self.updateCurrentRecords()

def deleteRecord(self, \*args):  
 if not self.current\_table\_name:  
 return  
 conditions\_str = ""  
 for index in range(0, len(self.current\_columns)):  
 conditions\_str += f"{self.current\_columns[index][1]}='{args[0][index].text}' AND "  
 query = f"DELETE FROM {self.current\_table\_name} WHERE {conditions\_str[0: -5]};"  
 self.database.executeSQLiteQuery(query=query)  
 self.saveChangesInTable()  
 self.updateCurrentRecords()

def changeRecord(self, \*args):  
 if not self.current\_table\_name:  
 return  
 previous\_data = args[1]  
 if len(previous\_data) == 0:  
 return  
 # conditions  
 conditions\_str = ""  
 for index in range(0, len(previous\_data)):  
 conditions\_str += f"{self.current\_columns[index][1]}='{previous\_data[index]}' AND "  
 # change to  
 change\_str = ""  
 for index in range(0, len(self.current\_columns)):  
 value = f"{args[0][index].text}"  
 try:  
 if self.current\_columns[index][2] == "INTEGER":  
 value = int(value)  
 if self.current\_columns[index][2] == "REAL":  
 value = float(value)  
 except Exception as ex:  
 print(ex)  
 return  
 value = f"'{value}'"  
 change\_str += f"{self.current\_columns[index][1]} = {value}, "  
 query = f"UPDATE {self.current\_table\_name} SET {change\_str[0: -2]} WHERE {conditions\_str[0: -5]}"  
 self.database.executeSQLiteQuery(query=query)  
 self.saveChangesInTable()  
 self.updateCurrentRecords()

1. **Интерфейс приложения: Основные окна с описанием.**

Взаимодействие пользователя с базой данных осуществляется посредством интерфейса, представленного в виде двух основных окон.



Окно интерфейса MODE:0. Рисунок 5.



Окно интерфейса MODE:1. Рисунок 6.

Если интерфейс находится в режиме 0 (Рисунок 5), пользователь может просматривать таблицы в базе данных. Добавлять/редактировать/удалять записи в любой из таблиц.

Для выбора таблицы требуется нажать на одну из кнопок, находящихся в левом столбце окна. Сама таблица занимает все оставшееся пространство. Каждая запись имеет номер. При нажатии на номер пользователь выбирает данную запись и может ее редактировать посредством окон ввода, находящихся внизу таблицы. После того как пользователь ввел необходимые данные для редактирования он должен нажать на кнопку, находящуюся справа от окон ввода “//”, запись будет отредактировано и окно обновлено. Кроме редактирования записи можно добавлять и удалять. Для добавления надо ввести нужные данные и нажать на кнопку, находящуюся слева от окон ввода “+”. Для удаления достаточно нажать на кнопку находящуюся справа от записи в таблице “x”. Так как все записи не помещаются на одной странице, было реализовано постраничное отображение таблицы. Для смены таблицы требуется нажать на одну из кнопок сверху справа таблицы “<” или “>”.

После заполнения таблиц данными пользователь может произвести специальные перманентные запросы (текст запросов записан в коде приложения). Для этого требуется сменить режим интерфейса, для этого надо нажать на кнопку сверху слева окна “MODE”.

Если интерфейс находится в режиме 1 (Рисунок 6), пользователь получает доступ к дополнительным функциям приложения. Посредством кнопок, находящихся в левом столбце окна. Так же в левом столбце находится окно ввода аргументов, требующихся для выполнения запросов. В данном режиме интерфейса пользователь не имеет возможности работы с таблицами, он может их только просматривать. Постраничное отображение таблиц так же реализовано.

1. **Инструментальные средства, которые использованы при разработке.**

При разработке приложения был использован язык Python, для написания интерфейса использовалась библиотека Kivy. Данные приложения хранятся в базе данных SQLite.