**Федеральное Государственное Бюджетное**

**Образовательное Учреждение**

**Высшего Профессионального Образования**

**Национальный Исследовательский Университет «МЭИ»**

Институт информационных и вычислительных технологий

Кафедра Прикладной Математики и Искусственного Интеллекта

**«Реализация базы данных учеников кружка по рисованию»**

**Курсовой проект**

по учебной дисциплине

«Базы данных»

Выполнил:

**Крылов К.С.**

Преподаватель:

**Сидорова Н.П.**

**Москва 2021**

**Оглавление**

1. Введение……………………………………………………………………………...3

2. Описание предметной области……………………………………………………...3

3. Проектирование БД…………………………………………………………………..4

4. Реализация базы данных……………………………………………………………..5

5. Программная реализация……………………………………………………………8

6. Вывод………………………………………………………………………………..11

Список литературы……………………………………………………………………12

# Введение.

**Цель** курсового проекта - разработка модели базы данных и интерфейсных средств для поддержки процессов учета и ведение бухгалтерии курсов по рисованию.

**Этапы** выполнения курсового проекта:

* анализ предметной области;
* проектирование БД в AllFusion ERwin Data Modeler.**[1]**
* реализация БД средствами SQLite.**[2]**
* программная реализация android интерфейса для управления БД на языке kotlin.**[3]**

БД должна поддерживать выполнение следующих функций:

* Вывод списка тем, рисунки на которые ученик делал.
* Получение информации о текущем балансе ученика.
* Вывод списка всех учеников заданного возраста для формирования группы.
* Составление списка учеников, имеющих задолженности(отрицательный баланс на счету).

## 1. Описание предметной области.

### Предметная область представляет из себя школу рисования: учителя, ученики и их работы. Учителя имеют контактные данные, ФИО и уникальный идентификатор, ученики имеют идентификатор своего учителя, свой идентификатор, баланс, возраст и список работ, каждая из которых имеет название и тему.

## 2. Проектирование БД.

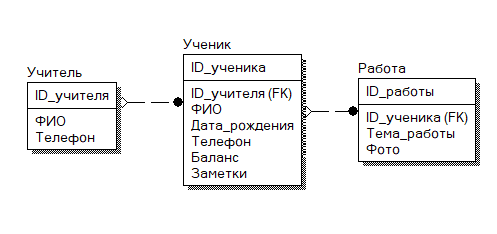


Таблица Учитель

Учитель (ID\_ учителя, ФИО, Телефон)

Ключ: ID\_ учителя

ID\_ учителя > ФИО

ID\_ учителя > Телефон

Таблица Ученик

Ученик (ID\_ ученика, ID\_ учителя, ФИО, Дата\_рождения, Телефон, Баланс, Заметки)

Ключ: ID\_ ученика

ID\_ ученика > ФИО

ID\_ ученика > Дата\_рождения

ID\_ учителя > Телефон

ID\_ учителя > Баланс

ID\_ учителя > Заметки

Таблица Работа

Учитель (ID\_ работы, ID\_ ученика, Тема\_работы, Фото)

Ключ: ID\_ работы

ID\_ учителя > Тема\_работы

ID\_ учителя > Фото

Все поля таблиц содержат атомарные значения, а не ключевые атрибуты функционально полно и не транзитивно зависят от ключей, значит база данных находится в 3НФ.

База данных будет реализована на SQLite – СУБД, встраиваемой в приложение. Выбор был сделан благодаря тому, что android приложения имеют возможность создания БД SQLite «из коробки» при помощи класса SQLiteOpenHelper, а библиотека room**[4]** делает работу с ним удобной, логически стройной и понятной. Также SQLite обладает следующими немаловажными преимуществами:

* Высокая скорость.
* Малый расход памяти.
* Отсутствие необходимости в отдельной программе – сервере (стоит отметить, что отсутствие необходимости в сервере не означает отсутствие возможности создать сервер с SQLite, например можно разместить БД в облаке при помощи Google Firebase**[5]**)

Из недостатков SQLite стоит отметить:

* Небольшой по меркам БД максимальный размер базы (хотя для android приложения это, в некоторых ситуациях может стать преимуществом).
* Небольшое разнообразие поддерживаемых типов данных, в данном случае приходится хранить дату в переменной типа long integer.
* Отсутствие репликации (в данном случае, в общем случае возможна).
* Отсутствие пользовательских прав (в данном случае недостаток не существенный, т.к. пользователь всего один).
* Простейшая изоляция транзакций – «записываем по одному» (недостаток так же несущественный, по той же причине что и пункт выше)
* Отсутствие архивации.

**3. Реализация базы данных.**

База данных написаны на языке kotlin при помощи библиотеки room, которая создает экземпляр базы данных SQLite.

**Сущности:**

*@Entity  
data class Teacher(  
 @PrimaryKey(autoGenerate = true)  
 val teacherId: Int,  
 val name: String,  
 val phone: String  
 )*

*@Entity(  
 foreignKeys = arrayOf(  
 ForeignKey(  
 entity = Teacher::class,  
 parentColumns = arrayOf("teacherId"),  
 childColumns = arrayOf("teacherId"),  
 onDelete = CASCADE,  
 onUpdate = CASCADE  
 )  
 )  
)*   
data class Student(  
 @PrimaryKey(autoGenerate = true)  
 val studentId: Int,  
 val teacherId: Int,  
 val name: String,  
 val birthDate: String,val phone: String,  
 val balance: Int,  
 val notes: String  
)

@Entity(  
 foreignKeys = arrayOf(  
 ForeignKey(  
 entity = Student::class,  
 parentColumns = arrayOf("studentId"),  
 childColumns = arrayOf("studentId"),  
 onDelete = CASCADE,  
 onUpdate = CASCADE  
 )  
 )  
)  
data class Work(  
 @PrimaryKey(autoGenerate = true)  
 val workId: Int,  
 val studentId: Int,  
 val theme: String,  
 val photo: Int  
)

**Запросы к базе данных осуществляются посредством формирования функций специальных классов DAO(Data Access Object):**

*@Dao  
interface StudentDao {  
 @Query("SELECT \* FROM Student")  
 fun getStudents(): MutableList<Student>  
  
 @Query("SELECT \* FROM Student WHERE teacherId = :id")  
 fun getByTeacherId(id: Int): Student?  
  
 @Query("SELECT \* FROM Student WHERE balance > :balance")  
 fun getByBalance(balance: Int): Student?  
  
 @Insert(onConflict = OnConflictStrategy.REPLACE)  
 fun insert(student: Student?): Long  
  
 @Delete  
 fun delete(student: Student?)  
  
 @Query("SELECT \* FROM Student WHERE birthDate > :birthdaymin and birthDate < :birthdaymax and balance >:balancemin and balance < :balancemax ")  
 fun getByDate(  
 @TypeConverters(  
 DateConverter::class  
 ) birthdaymin: Date,*

*@TypeConverters(  
 DateConverter::class  
 ) birthdaymax: Date,*

*balancemax: Int,*

*balancemin: Int,  
 ): Student?*

*@Query("SELECT \* FROM Student WHERE teacherID = :teacherID")  
fun getByTeacher(teacherID: Int): Student?*

*}*

*@Dao  
interface TeacherDao {  
 @get:Query("SELECT \* FROM Teacher")  
 val all: List<Any?>?  
  
 @Insert  
 fun insert(student: Teacher?)  
  
 @Update  
 fun update(student: Teacher?)  
  
 @Delete  
 fun delete(student: Teacher?)  
}*

*@Dao  
interface WorkDao {  
 @get:Query("SELECT \* FROM Work")  
 val all: List<Any?>?  
  
 @Query("SELECT \* FROM Work WHERE studentId = :id")  
 fun getByStudentId(id: Int): Work?  
  
 @Insert  
 fun insert(student: Work?)  
  
 @Update  
 fun update(student: Work?)  
  
 @Delete  
 fun delete(student: Work?)  
}*

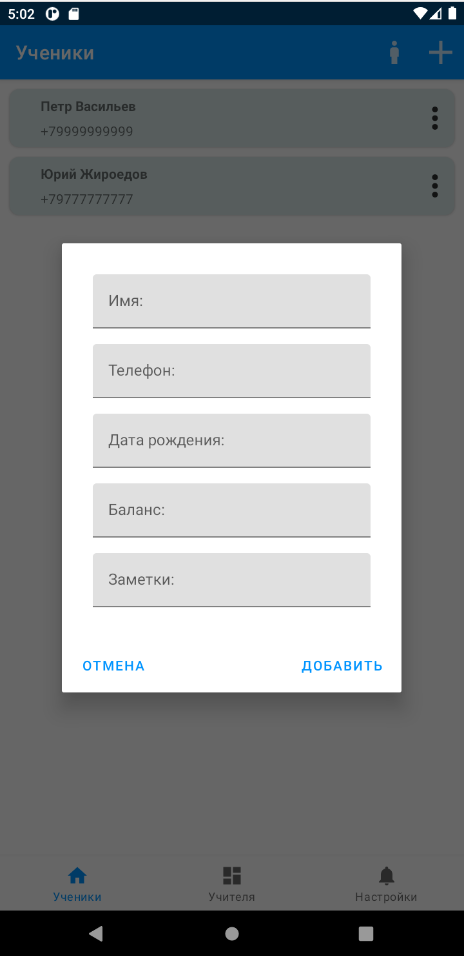
# 4. Программная реализация.

Интерфейс был реализован в среде android studio.**[6]** Интерфейс фрагментов описан в xml файлах, а логика фрагментов описана в специальных классах, наследующих класс Fragment. Логика взаимодействия фрагменты с базой данных описана в классе AppStateViewModel.



В разделе «ученики» выводится список учеников. При нажатии на три вертикальные точки появляется контекстное меню, предлагающее удалить ученика или узнать более подробную информацию о нём (или редактировать). Добавление ученика происходит при нажатии «+» на верхней панели навигации. При нажатии на пиктограмму с изображением человека открывается окно с фильтрацией данных. При добавлении ученика необходимо выбрать в окне со списком обучающего его учителя.

**Рис 1. Список учеников.**



В окне добавления ученика есть возможность введения имени, телефона, даты рождения, Баланса на счету и заметок. Внизу окна находятся кнопки подтверждения добавления ученика в базу данных и отмены.

Рис 2. Окно добавления ученика.

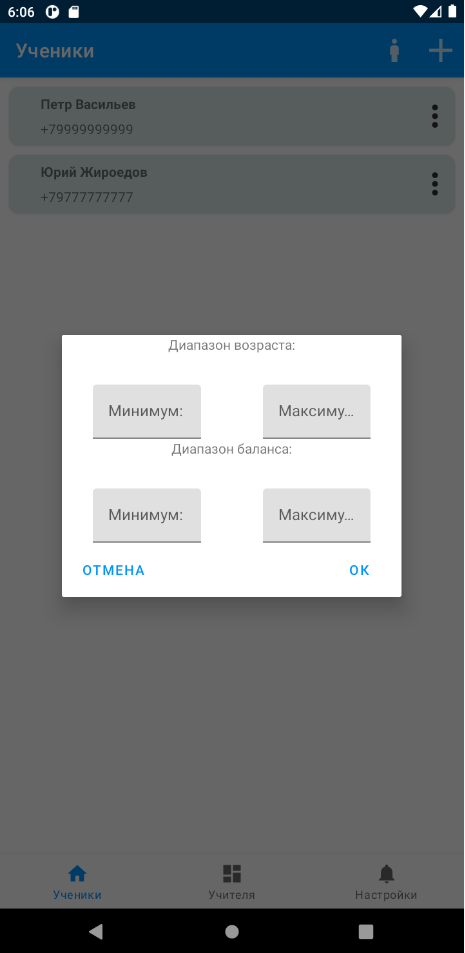
В окне фильтрации можно добавить значения минимума и максимума для баланса и возраста, при этом баланс, в отличие от возраста может быть отрицательным.

Рис 3. Окно фильтрации.

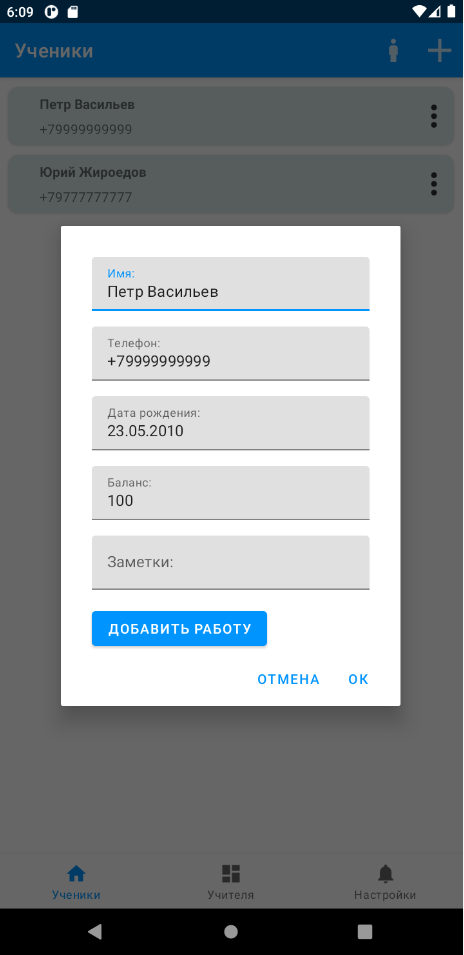
В окне редактирования отображается вся информация об ученике, как и при добавлении, но есть кнопка добавить работу. При ее нажатии открывается окно добавлении работы. Все работы ученика отображаются в окне редактирования в виде списка, как на главном экране.

Рис 4. Окно редактирования ученика.

На экране со списком учителей интерфейс идентичен списку учеников, но нет возможности отфильтровать учителей по признакам. Интерфейс добавления и редактирования учителя в целом аналогичен таковому у учеников.

Рис 4. Список учителей.

Вывод.

В результате выполнения курсового проекта был проведен анализ процесса организации процессов учета программного и технического обеспечения компьютеров компании, спроектирована БД из трех таблиц, реализованы указанные в задании функции и разработан интерфейс для работы с БД.

Список литературы

[1] Официальный сайт Erwin(сдесь и далее дата обращения 18.02.2022):

https://www.erwin.com/products/erwin-data-modeler/

[2] Официальная документация SQLite:

https://www.sqlite.org/index.html

[3] Официальная документация языка kotlin:

https://kotlinlang.org/docs/home.html

[4] Официальная документация библиотеки room:

https://developer.android.google.cn/jetpack/androidx/releases/room?hl=en

[5] Сайт Google Firebase:

https://firebase.google.com/

[6] Сайт для скачивания android studio:

https://developer.android.google.cn/studio