Лабораторная работа № 9. Разработка мобильных приложений с хранением информации в базе данных и в файле настроек

Цели лабораторной работы:

- получить навыки работы с разными типами хранилищ, включая SharedPreferences и базу данных;

Задачи лабораторной работы:

- изучить возможности по использованию SharedPreferences, DataStore;
- использовать абстрактные методы onCreate, OnUpgrade для работы с базой данных, выполнять выборки данных.

Table of Contents

Критерии оценивания

4-5 — приложения на основе примера 1, задания 1, 3, задание 4 для самостоятельной работы (4.1 на Java) из лабораторной работы и ответы на контрольные вопросы;

6-7 — приложения на основе примера 1, задания 1-3, задание 4 для самостоятельной работы (4.1-4.3 на Java) из лабораторной работы и ответы на контрольные вопросы;

8 — приложения на основе примера 1, задания 1-3, задание 4 для самостоятельной работы (4.1 на Java, 4.2-4.3 на Kotlin, 4.4 на Java) из лабораторной работы и ответы на контрольные вопросы. **Генерация кода на языке Kotlin из java-проекта не разрешается**;

9 — приложения на основе примера 1, задания 1-3, задание 4 для самостоятельной работы (4.1-4.4 на Java и Kotlin) из лабораторной работы, дополнительный функционал в заданиях для самостоятельной работы и ответы на контрольные вопросы. **Генерация кода на языке Kotlin из java-проекта не разрешается**.

9.1. Использование SharedPreferences

Теоретические сведения

SharedPreferences — постоянное хранилище на платформе Android, используемое приложениями для хранения своих настроек, например. Это хранилище является относительно постоянным, пользователь может зайти в

настройки приложения и очистить данные приложения, тем самым очистив все данные в хранилище.

В Android **Shared Preferences** используются для сохранения и извлечения данных примитивных типов данных (целочисленные, плавающие, логические, строковые, длинные) в виде пары ключ-значение из файла в файловой структуре приложения.

Обычно объект **Shared Preferences** указывает на файл, который содержит пары ключ-значение и предоставляет простые методы чтения и записи для сохранения и извлечения пар ключ-значение из файла.

Файл общих настроек управляется платформой Android, и к нему можно получить доступ из любого места приложения для чтения или записи данных в файл, но получить доступ к файлу из любого другого приложения невозможно, поэтому он защищен.

Общие настройки полезны для хранения небольшой коллекции значений ключей, таких как информация для входа пользователя в систему, настройки приложения, связанные с пользователями и т. Д., Для поддержания состояния приложения в следующий раз, когда они снова войдут в приложение.

Для взаимодействия с данными применяется SharedPreferemces API, классы которого наследуются от Preference API. В 29-ой версии API Preference API был переведён в статус deprecated. На данный момент используется новая версия API – `androidx.preference:preference` (https://developer.android.com/develop/ui/views/components/settings).

Чтобы создать новый файл общих настроек или получить доступ к существующему, нужно вызвать один из двух методов: **getSharedPreferences()** или **getPreferences()**.

- `getSharedPreferences()` — Используйте этот метод, если вам потребуется несколько общих файлов настроек, идентифицируемых по имени, указанном в первом параметре. Для работы с данными постоянного хранилища нам понадобится экземпляр класса **SharedPreferences**, который можно получить у любого объекта, унаследованного от класса *android.content.Context* (например, **Activity** или **Service**). У объектов этих классов (унаследованных от **Context**) есть метод **getSharedPreferences**, который принимает 2 параметра:
    - **name** — выбранный файл настроек. Если файл настроек с таким именем не существует, он будет создан при вызове метода edit().
    - **mode** — режим работы. Возможные значения:
        - **MODE_PRIVATE** — используется в большинстве случаев для приватного доступа к данным приложением-владельцем
        - **MODE_WORLD_READABLE** — только для чтения
        - **MODE_WORLD_WRITEABLE** — только записи
        - **MODE_MULTI_PROCESS** — несколько процессов

совместно используют один файл **SharedPreferences**.

- `getPreferences()` — Используйте метод из `Activity`, если вам нужен только один файл общих настроек для операции. Поскольку при этом получается файл общих настроек по умолчанию, относящийся к операции, вам не нужно указывать имя.

Если мы используем один общий файл настроек для нашей Activity, то нам необходимо инициализировать объект SharedPreferences с помощью метода getPreferences ():

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
```

Если у нас несколько файлов настроек, то инициализировать объект SharedPreferences можем следующим образом:

```
SharedPreferences sharedPref =
getSharedPreferences("filename1",Context.MODE_PRIVATE);
```

filename1 в данном случае — это файл настроек, из которого читаем значения. Режим доступа MODE_PRIVATE, что означает файл доступен только приложению, к которому относится.

Чтобы получить значение необходимой переменной, используйте следующие методы объекта **SharedPreferences**:

- **getBoolean**(String key, boolean defValue),
- **getFloat**(String key, float defValue),
- **getInt**(String key, int defValue),
- **getLong**(String key, long defValue),
- **getString**(String key, String defValue),
- **getStringSet**(String key, Set defValues).

Второй параметр — значение, которое вернется в случае если значение по ключу key отсутствует в **SharedPreferences**. Также, методом **getAll**() можно получить все доступные значения.

Записать в общие настройки

Для записи в файл общих настроек создайте объект `SharedPreferences.Editor` посредством вызова `edit()` в `SharedPreferences`.

Передавать ключи и значения, которые хотите записать, с помощью методов:

- **putBoolean**(String key, boolean value),
- **putFloat**(String key, float value),
- **putInt**(String key, int value),
- **putLong**(String key, long value),
- **putString**(String key, String value),
- **putStringSet**(String key, Set values)

Рассмотрим пример (см. код ниже). Чаще всего для передачи ключей и значений с помощью таких методов, как `putInt()` и `putString()`. Затем вызовите `commit()` для сохранения изменений. Например:

пример1

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putBoolean("keyname",true);
editor.putString("keyname","string value");
editor.putInt("keyname","int value");
editor.putFloat("keyname","float value");
editor.putLong("keyname","long value");
editor.commit();
```

пример 2

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score), newHighScore);
editor.commit();
```

Чтение общих настроек

Для получения значений из файла общих настроек следует вызвать такие, методы как `getInt()` и `getString()`, предоставляя ключ для нужного вам значения, а также при желании значение по умолчанию, которое будет выводиться при отсутствии ключа. Например:

пример 1

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
pref.getString("keyname",null);
pref.getInt("keyname",0);
pref.getFloat("keyname",0);
pref.getBoolean("keyname",true);
pref.getLong("keyname",0);
```

пример 2

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue = getResources().getInteger(R.string.saved_high_score_default);
long highScore = sharedPref.getInt(getString(R.string.saved_high_score),
defaultValue);
```

Удаление данных из общих настроек

Мы можем удалить все данные из файла общих настроек, используя метод `clear ()`, как показано ниже.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.clear();
editor.commit();
```

Используя фрагмент кода выше, мы удаляем все значения из общих настроек, используя метод `clear ()`, и фиксируем изменения в файле общих настроек,

используя метод `commit ()`.

Теперь рассмотрим, как хранить и извлекать пары ключ-значение примитивного типа данных в файле общих настроек, используя объект SharedPreferences в приложении для Android.

ПРИМЕР 1. Хранение параметров приложения в Shared preferences

Ниже приведен пример хранения и извлечения значений примитивных типов данных из файла общих настроек с использованием SharedPreferences.

Изучить документацию https://developer.android.com/develop/ui/views/components/settings.

Создайте новое приложение из одной Activity в Android Studio и сохраните под именем **SharedPreferences Example.**

После того, как создадите приложение, откройте файл **activity_main.xml** в папке **\res\layout** и напишите код, как показано ниже:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <EditText
        android:id="@+id/txtName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dp"
        android:ems="10" />

    <EditText
        android:id="@+id/txtPwd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dp"
        android:ems="10"
        android:inputType="textPassword" />

    <Button
        android:id="@+id/btnLogin"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:text="Login" />

    <TextView
        android:id="@+id/fstTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="150dp"
        android:text="UserName" />

    <TextView
        android:id="@+id/secTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
```

```
        android:text="Password" />
</LinearLayout>
```

Создайте другой файл разметки с именем **details.xml** в папке **\res\layout**., щелкнув правой кнопкой мыши по папке **layout** и выбрав их контекстного меню **New** команду **Layout Resource File**.

Пример кода для файла details.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/resultView"
        android:layout_gravity="center"
        android:layout_marginTop="170dp"
        android:textSize="20dp"/>
    <Button
        android:id="@+id/btnLogOut"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:text="Log Out" />
</LinearLayout>
```

Теперь откройте файл **MainActivity.java** и добавьте код ниже:

```
package com.example.myapplication;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import static android.content.Context.MODE_PRIVATE;

public class MainActivity extends AppCompatActivity {
    EditText uname, pwd;
    Button loginBtn;
    SharedPreferences pref;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        uname = (EditText)findViewById(R.id.txtName);
        pwd = (EditText)findViewById(R.id.txtPwd);
        loginBtn = (Button)findViewById(R.id.btnLogin);
        pref = getSharedPreferences("user_details",MODE_PRIVATE);
        intent = new Intent(MainActivity.this,DetailsActivity.class);
        if(pref.contains("username") && pref.contains("password")){
            startActivity(intent);
        }
        loginBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                String username = uname.getText().toString();
                String password = pwd.getText().toString();
                if(username.equals("fpmi-student") && password.equals("student")){
                    SharedPreferences.Editor editor = pref.edit();
                    editor.putString("username",username);
                    editor.putString("password",password);
                    editor.commit();
                    Toast.makeText(getApplicationContext(), "Login
Successful",Toast.LENGTH_SHORT).show();
                    startActivity(intent);
                }
                else
                {
                    Toast.makeText(getApplicationContext(),"Credentials are not
valid",Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

В данном коде проверяем, соответствуют ли введенные данные имени пользователя и пароля или нет. Если введены верно, то сохраняем данные в файл общих настроек и перенаправляем пользователя на другую Activity. Обратите внимание, что в коде задано имя пользователя **`fpmi-student`** и пароль **`student`**.

Создайте новый файл DetailsActivity.java, который будет выводить данные из общих настроек. Код файла ниже:

```
ackage com.example.myapplication;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import static android.content.Context.MODE_PRIVATE;

public class DetailsActivity extends AppCompatActivity {
    SharedPreferences prf;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.details);
        TextView result = (TextView)findViewById(R.id.resultView);
        Button btnLogOut = (Button)findViewById(R.id.btnLogOut);
        prf = getSharedPreferences("user_details",MODE_PRIVATE);
        intent = new Intent(DetailsActivity.this,MainActivity.class);
        result.setText("Hello, "+prf.getString("username",null));
        btnLogOut.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SharedPreferences.Editor editor = prf.edit();
                editor.clear();
                editor.commit();
                startActivity(intent);
            }
        });
    }
}
```

Добавим описание созданной Activity в файл AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".DetailsActivity" android:label="Shared Preferences -
Details"></activity>
    </application>

</manifest>
```

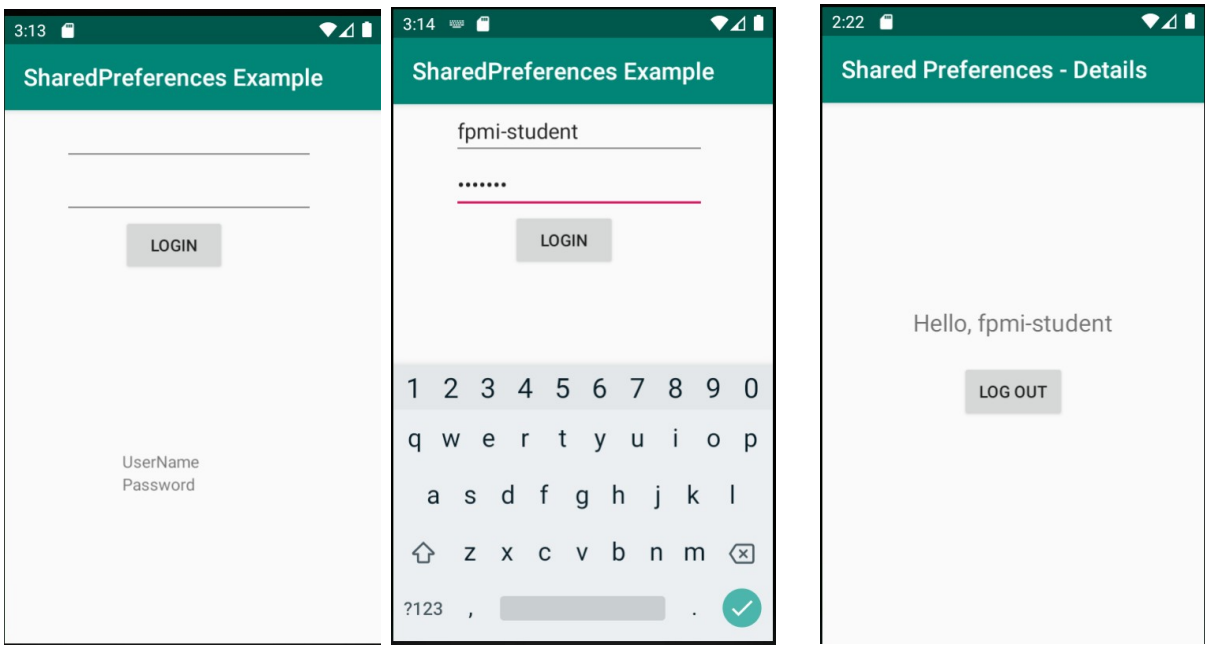

Рис. 1. а) Макет Activity.xml б) Макет details.xml

ЗАДАНИЕ 1. Хранение параметров приложения в Android Jetpack Preferences DataStore

Реализовать приложение из примера 1 на языке Java, заменив **LinearLayout** на **ConstraintLayout** и добавив поля в файл string.xml и внеся изменения в код приложения.

9.2. Использование Android Jetpack Preferences DataStore

Теоретические сведения

В отличие от SharedPreference, DataStore работает асинхронно. Вся работа с

библиотекой выполняется с помощью Kotlin Coroutines и Flow. DataStore позволяет нам хранить данные двумя способами:

- По принципу «ключ — значение», аналогично SharedPreference.
- Хранить типизированные объекты, основанные на protocol buffers.

Все взаимодействие с DataStore происходит через интерфейс `DataStore<T>`, который содержит в себе всего два элемента:

```
interface DataStore<T> {
   val data: Flow<T>
   suspend fun updateData(transform: suspend (t: T) -> T): T
}
```

Типы DataStore

- **Preferences DataStore** — хранит данные по принципу *«ключ — значение»* и не предоставляет нам никакой типобезопасности.
- **Proto DataStore** — хранит данные в объектах. Это дает нам типобезопасноть, но описывать схему нужно с помощью protocol buffers.

Подробнее:

https://habr.com/ru/company/tinkoff/blog/525010/

https://android-developers.googleblog.com/2020/09/prefer-storing-data-with-jetpack.html

https://www.raywenderlich.com/18348259-datastore-tutorial-for-android-getting-started

ЗАДАНИЕ 2. Использование Android Jetpack Preferences DataStore

Изучить статьи из раздела 9.2. Использование Android Jetpack Preferences DataStore и пример https://c1ctech.com/android-jetpack-preferences-datastore-example/. Реализовать приложение из примера в статье.

9.3. Работа с базой данных

Теоретические сведения

SQLite

• SQLite—компактная встраиваемая реляционная база данных с открытым исходным кодом. Поддерживает динамическое типизированиеданных.

• Возможные типы полей: INTEGER, REAL, TEXT, BLOB.

В приложении, при подключении к БД мы указываем **имя** БД и **версию**. При этом могут возникнуть следующие ситуации:

1) БД **не существует**. Это может быть, например, в случае первичной установки программы. В этом случае приложение должно само **создать** БД и все

таблицы в ней. И далее оно уже работает с только что созданной БД.

2) БД **существует**, но ее версия **устарела**. Это может быть в случае обновления программы. Например, новой версии программы нужны дополнительные поля в старых таблицах или новые таблицы. В этом случае приложение должно **апдейтить** существующие таблицы и создать новые, если это необходимо.

3) БД **существует** и ее версия **актуальна**. В этом случае приложение успешно **подключается** к БД и работает.

Для обработки описанных выше ситуаций нам надо создать **класс**, являющийся наследником для SQLiteOpenHelper. Назовем его DBHelper. Этот класс предоставит нам методы для **создания** или **обновления** БД в случаях ее **отсутствия** или **устаревания**:

SQLiteOpenHelper

Абстрактные методы

onCreate - метод, который будет вызван, если БД, к которой мы хотим подключиться – не существует

onUpgrade - будет вызван в случае, если мы пытаемся подключиться к БД более новой версии, чем существующая

Реализация метода onCreate

```
public void onCreate( SQLiteDatabasedb) {
dЬ.execSQL( "CREATE TABLE"+ TABLE_NAME
+ "(_id INTEGER PRIMARY KEYAUTOINCREMENT ,"
+ COL NAME + " TEXT ,"+ COL PHONE + " TEXT ) ; " );
```

Реализация метода **onUpgrade**

```
@ Override
puЬlicvoid onUpgrade( SQLiteDatabasedb, intoldVersion, intnewVersion)
{
db.execSQL( "DROP TABLE IF EXISTS"+ TABLE_NAME );
onCreate( db);
}
```

Чтение из базы данных

Cursor query(String table, String [ ] c5. Приведите фрагменты кода для записи, чтения данных из файла общих настроек и удаления данных.

```
olumns,
String selection, String [ ] selectionArgs,
String groupBy, String having , String sortOrder)
```

Позиционирование курсора

- moveToFirst( ) -перемещает курсор в первую запись в выборке;
- moveToLast( ) -перемешает курсор в последнюю запись в выборке;
- moveToNext( ) -перемешает курсор в следующую запись и одновременно определяет, существует ли эта запись. Метод moveToNext( ) возвращает true, если курсор указывает на другую строку после перемещения, и false,если текущая запись была последней в выборке;
- moveToPrevious( ) -перемешает курсор в предыдущую запись;
- moveToPosition( ) -перемешает курсор в указанную позицию;
- getPosition( ) -возвращает текущий индекс позиции курсора.
- isFirst( );
- isLast( );
- • isBeforeFirst( );
- • isAfterLast( ).

Обновление и удаление записей

• intupdate(String table, ContentValuesvalues, String whereClause, String [

] whereArgs)

• intdelete (String table, String whereClause, String [ ] whereArgs)

ЗАДАНИЕ 3. Приложение для работы с базой данных SQLite

Создадим простое приложение – справочник контактов, которое будет хранить **имя** и **email**. **Вводить** данные будем на **экране** приложения, а для **отображения** информации используем **логи**. Обычно для вывода используется List (список).

Открываем **MainActivity.java** и пишем:

```
public class MainActivity extends Activity implements OnClickListener {

  final String LOG_TAG = "myLogs";

  Button btnAdd, btnRead, btnClear;
  EditText etName, etEmail;

  DBHelper dbHelper;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        btnAdd = (Button) findViewById(R.id.btnAdd);
        btnAdd.setOnClickListener(this);

        btnRead = (Button) findViewById(R.id.btnRead);
        btnRead.setOnClickListener(this);

        btnClear = (Button) findViewById(R.id.btnClear);
        btnClear.setOnClickListener(this);

        etName = (EditText) findViewById(R.id.etName);
        etEmail = (EditText) findViewById(R.id.etEmail);

        // создаем объект для создания и управления версиями БД
        dbHelper = new DBHelper(this);
    }


    @Override
    public void onClick(View v) {

        // создаем объект для данных
        ContentValues cv = new ContentValues();

        // получаем данные из полей ввода
        String name = etName.getText().toString();
        String email = etEmail.getText().toString();

        // подключаемся к БД
        SQLiteDatabase db = dbHelper.getWritableDatabase();


        switch (v.getId()) {
        case R.id.btnAdd:
          Log.d(LOG_TAG, "--- Insert in mytable: ---");
           // подготовим данные для вставки в виде пар: наименование столбца - значение

          cv.put("name", name);
          cv.put("email", email);
          // вставляем запись и получаем ее ID
          long rowID = db.insert("mytable", null, cv);
```

```
            Log.d(LOG_TAG, "row inserted, ID = " + rowID);
            break;
          case R.id.btnRead:
            Log.d(LOG_TAG, "--- Rows in mytable: ---");
            // делаем запрос всех данных из таблицы mytable, получаем Cursor
            Cursor c = db.query("mytable", null, null, null, null, null, null);

            // ставим позицию курсора на первую строку выборки
            // если в выборке нет строк, вернется false
            if (c.moveToFirst()) {

              // определяем номера столбцов по имени в выборке
              int idColIndex = c.getColumnIndex("id");
              int nameColIndex = c.getColumnIndex("name");
              int emailColIndex = c.getColumnIndex("email");

              do {
                // получаем значения по номерам столбцов и пишем все в лог
                Log.d(LOG_TAG,
                    "ID = " + c.getInt(idColIndex) +
                    ", name = " + c.getString(nameColIndex) +
                    ", email = " + c.getString(emailColIndex));
                // переход на следующую строку
                 // а если следующей нет (текущая - последняя), то false - выходим из
цикла
              } while (c.moveToNext());
            } else
              Log.d(LOG_TAG, "0 rows");
            c.close();
            break;
          case R.id.btnClear:
            Log.d(LOG_TAG, "--- Clear mytable: ---");
            // удаляем все записи
            int clearCount = db.delete("mytable", null, null);
            Log.d(LOG_TAG, "deleted rows count = " + clearCount);
            break;
          }
          // закрываем подключение к БД
          dbHelper.close();
        }

        class DBHelper extends SQLiteOpenHelper {

          public DBHelper(Context context) {
```

```
        // конструктор суперкласса
        super(context, "myDB", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        Log.d(LOG_TAG, "--- onCreate database ---");
        // создаем таблицу с полями
        db.execSQL("create table mytable ("
            + "id integer primary key autoincrement,"
            + "name text,"
            + "email text" + ");");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
  }

}
```

В методе Activity - **onCreate** мы определяем объекты, присваиваем обработчики и создаем объект dbHelper класса **DBHelper** для управления БД. Сам класс будет описан ниже.

Далее смотрим метод Activity – **onClick**, в котором обрабатываем нажатия на кнопки.

Класс ContentValues используется для указания **полей** таблицы и **значений**, которые в эти поля будем вставлять. Создаем объект **cv**, и позже его используем. Далее записываем в переменные значения из **полей** ввода. Затем, с помощью метода getWritableDatabase подключаемся к БД и получаем объект SQLiteDatabase. Он позволит нам работать с БД. Будем использовать его методы **insert** – вставка записи, **query** – чтение, **delete** – удаление.

Далее смотрим, какая кнопка была нажата:

btnAdd – добавление записи в таблицу *mytable*. Заполняем объект **cv** парами: **имя** поля и **значение**. И (при вставке записи в таблицу) в указанные поля будут вставлены соответствующие значения. Заполняем поля *name* и *email*. *id* у нас заполнится автоматически (primary key autoincrement). Вызываем метод insert – передаем ему **имя таблицы** и объект **cv** с вставляемыми значениями. Второй аргумент метода используется, при вставке в таблицу пустой строки. Нам это сейчас не нужно, поэтому передаем null. Метод **insert** возвращает **ID** вставленной строки, мы его сохраняем в **rowID** и

выводим в **лог**.

btnRead – чтение всех записей из таблицы *mytable*. Для чтения используется метод query. На вход ему подается **имя таблицы**, список запрашиваемых полей, условия выборки, группировка, сортировка. Т.к. нам нужны все данные во всех полях без сортировок и группировок - мы используем везде **null**. Только имя таблицы указываем. Метод возвращает нам объект класса Cursor. Его можно рассматривать как таблицу с данными. Метод moveToFirst – делает **первую** запись в Cursor **активной** и заодно проверяет, есть ли вообще записи в нем (т.е. выбралось ли что-либо в методе **query**). Далее мы получаем порядковые номера столбцов в Cursor по их именам с помощью метода getColumnIndex. Эти номера потом используем для чтения данных в методах getInt и getString и выводим данные в лог. С помощью метода moveToNext мы перебираем все строки в Cursor пока не добираемся до последней. Если же записей не было, то выводим в лог соответствующее сообщение – *0 rows*. В конце закрываем курсор (освобождаем занимаемые им ресурсы) методом close, т.к. далее мы его нигде не используем.

btnClear – очистка таблицы. Метод delete удаляет записи. На вход передаем **имя таблицы** и **null** в качестве условий для удаления, а значит удалится все. Метод возвращает **кол-во удаленных** записей.

После этого закрываем соединение с БД методом close.

Класс **DBHelper** является **вложенным** в **MainActivity** и описан в конце кода. Этот класс должен наследовать класс **SQLiteOpenHelper**.

В **конструкторе** мы вызываем конструктор суперкласса и передаем ему:

context - контекст

mydb - название базы данных

null – объект для работы с курсорами, нам пока не нужен, поэтому null

1 – версия базы данных

В методе **onCreate** этого класса мы используем метод execSQL объекта SQLiteDatabase для выполнения SQL-запроса, который создает таблицу. Этот метод вызывается, если БД не существует и ее надо создавать. По запросу видно, что мы создаем таблицу *mytable* с полями *id*, *name* и *email*.

Метод **onUpgrade** пока не заполняем, т.к. используем одну версию БД и менять ее не планируем.

Важно понять, что для работы с БД мы использовали два класса:

- **DBHelper**, наследующий **SQLiteOpenHelper**. В его **конструкторе** мы вызываем конструктор супер-класса и указываем имя и версию БД. Метод **getWritableDatabase** выполняет подключение к базе данных и возвращает нам объект **SQLiteDatabase** для работы с ней. Метод **close** закрывает подключение к БД. В случае, когда БД отсутствует или устарела, класс предоставляет нам самим реализовать создание или обновление в методах **onCreate** и **onUpgrate**.

- **SQLiteDatabase**. Содержит методы для работы с данными – т.е. **вставка**, **обновление**, **удаление** и **чтение**.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

ЗАДАНИЕ 4.1. Хранение параметров приложения в Shared preferences или DataStore

Изучить статью «Android: SharedPreferences (http://developer.alexanderklimov.ru/android/theory/sharedpreferences.php)», материалы и рекомендации из раздела 9.2. Использование Android Jetpack Preferences DataStore, примеры для Kotlin и Java из документации https://developer.android.com/training/data-storage/shared-preferences и изменить приложение «Угадай число», разработанное в лабораторной работе № 6 и дополненное в лабораторной работе 7.

Сохранить в один файл настроек, используя для хранения перечисленных ниже параметров SharedPrefrences или DataStore:

- счетчик — количество запусков приложения;
- уникальный ID приложения, генерируемый при первом запуске;
- настройки пользователя (количество попыток, набранных баллов, лучший результат и др.);

  Изучить пример https://abhiandroid.com/programming/shared-preference, добавить Activity авторизации (ввода адреса электронной почты и пароля) и сохранить в другой файл настроек данные для авторизации пользователя:

- адрес электронной почты и пароль.

  Приложение реализовать или на Java, или на Kotlin в зависимости от реализации приложения в лабораторных работах 6 и 7.

ЗАДАНИЕ 4.2. ПРИЛОЖЕНИЕ С ХРАНЕНИЕМ ДАННЫХ В БД SQLITE

Необходимо создать приложение, взаимодействующее с базой данных и с учетом критериев оценивания на Java и/или Kotlin. Первая операция (activity) должна содержать три кнопки. При нажатии на первую кнопку должна открываться новая операция (activity), выводящая информацию из таблицы согласно варианту задания в удобном для вывода формате.

При запуске приложения необходимо:

1. Создать БД, если ее не существует.

2. Создать таблицу согласно варианту

3. Удалить все записи из БД (если существует), а затем добавить 5 записей.

При нажатии на вторую кнопку необходимо внести еще одну запись в таблицу.

При нажатии на третью кнопку необходимо заменить данные в любой записи.

ЗАДАНИЕ 4.3. ОБНОВЛЕНИЕ СХЕМЫ БАЗЫ ДАННЫХ SQLITE

Создать новое отдельное приложение на основе приложения, созданного в упражнении 4.2.

Переопределить функцию onUpgrade. При изменении версии БД необходимо удалить таблицу с данными, создать таблицу содержащую два дополнительных поля следующих форматов: integer, char.

Обязательным условием при изменении структуры таблицы (базы данных) является изменение версию базы данных и сохранение данных, т. е. основная цель данного задания — проверить как правильно выполнять обновление приложения, если изменяется структура базы данных.

Перед выполнением задания изучить пример по адресу — http://startandroid.ru/ru/uroki/vse-uroki-spiskom/79-urok-39-onupgrade-obnovljaem-bd-v-sqlite.html.

ЗАДАНИЕ 4.4. ПРИЛОЖЕНИЕ С ХРАНЕНИЕМ ДАННЫХ В БД SQLITE И ВОЗМОЖНОСТЯМИ ФИЛЬТРАЦИИ, СОРТИРОВКИ, ГРУППИРОВКИ И АГРЕГАЦИИ

Создать приложение согласно варианту и с учетом критериев оценивания на Java и/или Kotlin. Реализовать в приложении следующие функции:

- вывод всех записей.
- вывод значения агрегирующей функции (SUM, MIN, MAX, COUNT, AVG)
- вывод строк большего определённого значения, которое задаётся или в текстовом поле, или списком выбора или другим способом.
- группировка строк по определённому признаку (признак может выбираться из списка).
- сортировка по различным полям.

Варианты

Вариант 1.

4.2. Создать таблицу «Участники конференции», содержащую поля:

- ID;
- ФИО;
- Время регистрации на конференции.

4.3. Добавить поля «Компания», «Должность», «Адрес электронной почты».

4.4. База данных «Справочник Минска» состоит как минимум из 2 таблиц. Для районов Минска найти данные по микрорайонам количеству жителей, предприятий, школ, вузов, бюджету района.

Вариант 2.

4.2. Создать таблицу «Студенты», содержащую поля:

- ID;
- Фамилия;
- Имя;
- Дата зачисления.

4.3. Добавить поля «Место рождения», «Дата рождения», «Номер телефона».

4.4. База данных «Специальности ФПМИ» состоит как минимум из 2 таблиц. Хранить данные о курсах (1, 2, 3, 4), специальностях (ПМ, ПИ, ИН, КБ, АМ), предметах, читаемых на специальностях, и количестве студентов на каждой специальности каждого курса.

Вариант 3.

4.2. Создать таблицу «Книги», содержащую поля:

- ID;
- Автор;
- Название;
- Год выпуска.

4.3. Добавить поля «Город (где издана книга)», «Издательство», «Формат издания» (печатная или электронная книга).

4.4. База данных «Вузы Минска» состоит как минимум из 2 таблиц. Для вузов, хранить данные о названии, количеству факультетов, количеству специальностей, специальности и коды специальностей, количество студентов на специальности, а так же в каком вузе и на каком факультете обучают такой специальности, например студенты на специальности «Прикладная информатика» обучаются на факультетах ФПМИ, факультете социальнокультурных коммуникаций, факультете радиофизики и электроники в БГУ.

Вариант 4.

4.2. Создать таблицу «Специальность», содержащую поля:

- ID;
- Название специальности;
- Год открытия специальности.

4.3. Добавить поля «Курс обучения», «Количество студентов», «Специализация».

4.4. База данных «Издательство» состоит как минимум из 2 таблиц. Содержит сведения об авторах (фамилия, имя, отчество, емейл), гонорары (месяц, размер гонорара выплаченного автора, id книги, за которую выплачен гонорар), книги (название, автора, год издания, количество страниц) и др. данные.

Вариант 5.

4.2. Создать таблицу «Машины», содержащую поля:

- ID;

- Модель;
- Производитель;
- Год выпуска.

4.3. Добавить поля «Объем двигателя», «Количество лошадиных», «Тип кузова».

4.4. База данных «Магазин» состоит как минимум из 2 таблиц. Содержит данные о товарах, продавцах, стоимости товара, количестве проданных единиц, день продажи каждого товара, выручке для каждого продавца за каждый день.

Вариант 6.

4.2. Создать таблицу «Экзамен», содержащую поля:

- ID;
- Название предмета (дисциплины);
- Дата проведения экзамена

4.3. Добавить поля «Преподаватель», «Время начала экзамена», «Группа».

4.4. База данных «Страны» состоит как минимум из 2 таблиц. Содержит данные об общей площади, количестве населения, столице, городах (не менее 3), достопримечательностях и в каком городе находится достопримечательность.

Вариант 7.

4.2. Создать таблицу «Контакты», содержащую поля:

- ID;
- Фамилия;
- Имя
- Дата добавления записи.

4.3. Добавить поля «Отчество», «Адрес электронной почты», «Номер телефона».

4.4. База данных «Интернет магазин» состоит как минимум из 2 таблиц. Содержит сведения о товарах, стоимости товара, признаке наличия товара, заказах (номер, заказанные товары, способ оплаты), адресе доставки, курьере, статусе доставки.

Вариант 8.

4.2. Создать таблицу «Расписание», содержащую поля:

- ID;
- Название предмета;
- Время начала лекции/занятия.

4.3. Добавить поля «Время окончания занятия», «ФИО преподавателя», «Аудитория».

4.4. База данных «Сотрудники» состоит как минимум из 2 таблиц. Содержит

сведения о личных данных сотрудниках (фио, дата рождения, место рождения, номер телефона, электронной почты), паспортах, местах работы, занимаемых должностях, трудовом стаже и средней заработной плате по каждому месту работы.

Вариант 9.

4.2. Создать таблицу «Ноутбуки», содержащую поля:

- ID;
- Модель;
- Производитель.

4.3. Добавить поля «Время выпуска», «Процессор», «Объем оперативной памяти».

4.4. База данных «Студенты» состоит как минимум из 2 таблиц. Содержит сведения о личных данных студента (фио, дата рождения, место рождения, номер телефона, электронной почты), паспортные данные, результаты на момент поступления, результаты сдачи по сессиям (как минимум 2 сессии и 4 предмета по каждой сессии)

Вариант 10.

4.2. Создать таблицу «Музеи Минска», содержащую поля:

- ID;
- Название музея;
- Дата основания.

4.3. Добавить поля «Адрес», «Площадь музея», «Выставка (название выставки)».

4.4. База данных «Ресторан» состоит как минимум из 2 таблиц. Содержит сведения о блюдах, их стоимости, времени на готовку блюда, включено или нет в обеденное меню, заказы, включая заказы с доставкой и данные о доставке, включая стоимость заказа, адрес доставки, способ оплаты и курьер и способ доставки.

Вариант 11.

4.2. Создать таблицу «Товар», содержащую поля:

- ID;
- наименование;
- стоимость.

4.3. Добавить поля «срок хранения», «сорт», «дата выпуска», «срок годности».

4.4. База данных «Музеи и библиотеки Минска» состоит как минимум из 2 таблиц. Для районов Минска найти данные по количеству музеев, библиотек, суммарное количеству посетителей для музеев и библиотек по году, среднее количество посетителей по музеям и по библиотекам, максимальное и минимальное количество посетилей и др. данные.

Вариант 12.

4.2. Создать таблицу «Автомобиль», содержащую поля:

- ID;
- Марка;
- Серийный номер;
- Год выпуска.

4.3. Добавить поля «Цвет», «Год техосмотра», «Цена».

4.4. База данных «Кафедры» состоит как минимум из 2 таблиц. Для кафедр хранить сведения о заведующем, преподавателях, читаемых дисциплинах, количестве часов (лекционных и лабораторных). Учесть, что один преподаватель может читать несколько дисциплин и несколько преподавателей могут читать одну дисциплину. Вывести данные о суммарном количестве читаемых часов на преподавателя, с делением на лекционные и лабораторные, среднее количество часов на преподавателя по кафедре, минимальное и максимальное количество часов на преподавателя.

Вариант 13.

4.2. Создать таблицу «Государства», содержащую поля:

- ID;
- Название страны;
- Столица;
- Государственный язык.

4.3. Добавить поля «Население», «Площадь территории», «Денежная единица».

4.4. База данных «Учебники» состоит как минимум из 2 таблиц. Содержит сведения названия учебника, авторы, тираж, год издания, стоимость, язык учебника, класс обучения, издательство. Найти данные по количеству учебников для отдельного автора, суммарный тираж по одному учебнику в зависимости от языка, года издания, минимальный и максимальный тираж.

Вариант 14.

4.2. Создать таблицу «Абитуриент», содержащую поля:

- ID;
- Фамилия;
- Имя;
- Отчество;
- Дата рождения (год, месяц число).

4.3. Добавить поля «Гражданство», «Год поступления», «ЦТ (количество баллов)», «Аттестат (количество баллов)».

4.4. База данных «Издательство» состоит как минимум из 2 таблиц. Содержит сведения об видах печатных изделий (название, формат, тип бумаги, стоимость), заказы (дата заказа, количество экземпляров, количество страниц, стоимость, формат печати (одно/двухсторонняя)) и другие данные.

Вариант 15.

4.2. Создать таблицу «Рабочий», содержащую поля:

- ID;
- ФИО;
- Домашний адрес;
- Табельный номер;
- Должность.

4.3. Добавить поля «Образование», «Год поступления на работу», «Дата рождения (год, месяц число)».

4.4. База данных «БГУ» состоит как минимум из 2 таблиц. Для факультетов БГУ найти данные по количеству студентов, количеству специальностей, количеству студентов на бюджетной и платной форме обучения, количеству студентов по курсам (году обучения), средний, максимальный и минимальный балл студентов.

Вариант 16.

4.2. Создать таблицу «Владелец телефона», содержащую поля:

- ID;
- Фамилия;
- Имя;
- Отчество;
- Дата рождения;
- Номер телефона.

4.3. Добавить поля «Адрес», «Год заключения договора», «Название тарифа», «Стоимость тарифа».

4.4. База данных «Зоомагазин» состоит как минимум из 2 таблиц. Выводить список кормов с сортировкой, максимальную и минимальную стоимость одного корма в зависимости от производителя, среднюю стоимость одного корма, суммарное количество кормов.

Вариант 17.

4.2. Создать таблицу «Военнослужащий», содержащую поля:

- ID;
- ФИО;
- Домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира).

4.3. Добавить поля «Дата рождения», «Должность», «Звание», «Зарплата».

4.4. База данных «Пиццерия» состоит как минимум из 2 таблиц. Содержит сведения о видах блюд, их стоимости, акции, заказы, включая заказы с доставкой и данные о доставке, включая стоимость заказа и участие в акции, адрес доставки, способ оплаты, курьер и способ доставки.

Вариант 18.

4.2. Создать таблицу «Владелец автомобиля», содержащую поля:

- ID;
- ФИО;
- Номер телефона;
- Номер автомобиля.

4.3. Добавить поля «Марка автомобиля», «Номер техпаспорта», «Стоимость», «Год выпуска».

4.4. База данных «Аптека» состоит как минимум из 2 таблиц. Выводить количество лекарств в каждой аптеке, среднюю стоимость одного лекарства по аптекам, минимальную и максимальную стоимость одного лекарства по аптекам.

Вариант 19.

4.2. Создать таблицу «Пациент», содержащую поля:

- ID;
- ФИО;
- Рост;
- Вес;
- Номер медицинской карты.

4.3. Добавить поля «Диагноз», «Группа крови», «Домашний адрес», «Номер телефона».

4.4. База данных «Онлайн-курса» состоит как минимум из 2 таблиц. Для каждого курса выводить количество студентов на каждый поток, среднее количество студентов по каждому курсу, максимальное и минимальное количество студентов, количество обученных студентов по каждому курсу и по году обучения.

Вариант 20.

4.2. Создать таблицу «Покупатель», содержащую поля:

- ID;
- ФИО;
- Номер телефона;
- Номер скидочной карты.

4.3. Добавить поля «Дата рождения», «Домашний адрес», «Адрес электронной почты», «Средний чек».

4.4. База данных «Аудиотека» состоит как минимум из 2 таблиц. Добавить функционал, обеспечивающий вывод среднего и суммарного количества песен в альбомах, максимальное количество песец и минимальное количество песен в одном альбоме, количество альбомов по году выпуска.

Вариант 21.

4.2. Создать таблицу «Студент», содержащую поля:

- ID;
- ФИО;
- Год поступления;
- Специальность;
- Факультет.

4.3. Добавить поля «Группа», «Курс», «Средний балл».

4.4. База данных «Фильмотека» состоит как минимум из 2 таблиц. Найти данные по количеству сеансов, среднему и суммарному количеству зрителей, максимальному и минимальному количеству зрителей за один сеанс.

Вариант 22.

4.2. Создать таблицу «Школьник», содержащую поля:

- ID;
- ФИО;
- Дата рождения;
- Школа;
- Класс.

4.3. Добавить поля «Классный руководитель», «Средний балл», «Адрес электронной почты».

4.4. База данных «Библиотека» состоит как минимум из 2 таблиц. Для авторов и тем найти данные по количеству книг автора/ темы, суммарной и средней стоимости, среднем тираже, количеству проданных книг и полученному доходу.

Вариант 23.

4.2. Создать таблицу «Студенты колледжа», содержащую поля:

- ID;
- Фамилия;
- Имя;
- Дата зачисления.
- Программа, на которую зачислен.
- Длительность обучения.

4.3. Добавить поля «Место рождения», «Дата рождения», «Дата окончания», «Специальность», «Квалификация». В поля «Специальность», «Квалификация» добавить данные только, если студент окончил колледж.

4.4. База данных «Факультеты БГУ» состоит как минимум из 2 таблиц. Хранить данные о факультетах(как минимум 3 факультета), курсах (1, 2, 3, 4), специальностях, количестве студентов на каждой специальности каждого курса, количестве преподавателей на каждой должности с учетом ставки и на каждом факультете.

Вариант 24.

4.2. Создать таблицу «Сотрудники», содержащую поля:

- ID;
- Фамилия;
- Имя;
- Дата принятия на работу.

4.3. Добавить поля «Идентификационный номер», «Дата рождения», «Номер телефона», «Должность», «Дата увольнения».

4.4. База данных «Товары и заказы» состоит как минимум из 2 таблиц. Хранить данные о компании (производителе), моделях микроволновых печей, дате выпуска, стоимости, НДС, количестве суммарно, на каком складе или в каком магазине находятся и другие данные.

Вариант 25.

4.2. Создать таблицу «Книги», содержащую поля:

- ID;
- Автор (ФИО);
- Название книги;
- Год издания.

4.3. Добавить поля «Издательство», «Количество страниц», «Стоимость».

4.4. База данных «Программное обеспечение» состоит как минимум из 2 таблиц. Содержит сведения о видах ПО, их стоимости, лицензии, акции, заказы, включая стоимость заказа, количество и типы лицензий, участие в акции, способ оплаты и др. данные.

Контрольные вопросы

1. Как называется хранилище настроек в Android и какие типы данных в нем можем хранить?

2. Какой метод используется в случае одного файла общих настроек?

3. Какой метод используется в случае нескольких файлов настроек?

4. Какие режимы работы (доступа) задаются при использовании SharedPreferences и какой из них самый надежный?

5. Приведите фрагменты кода для записи, чтения данных из файла общих настроек и удаления данных.

6. Дайте характеристику методу хранения данных DataStore, его типах и в каких случаях его рекомендуется использовать.

7. Приведите фрагменты кода для записи, чтения данных из файла DataStore в случае хранения данных по принципу «ключ — значение» по аналогии с

SharedPreferences.

8. Какой класс используется для создания базы данных SQLite и управления версиями?

9. Какой метод, который будет вызван, если БД, к которой подключаемся, не существует? Приведите его реализацию.

10. Какой методы, будет вызван в случае, если подключаемся к БД более новой версии, чем существующая?

11. Приведите фрагмент кода, описывающий вставку строк, вывод всех строк из одной таблицы, вывод значений из таблицы в зависимости от условий.

12. Приведите фрагмент кода, описывающий обновление и удаление записей из базы данных.