

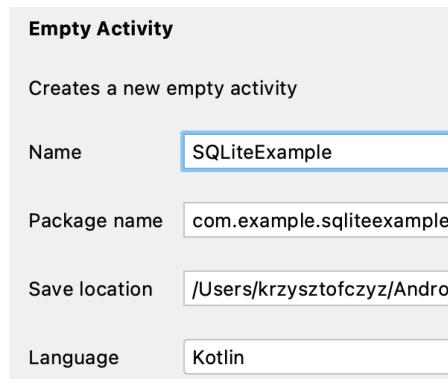
Spis treści

OBSŁUGA WŁASNEJ BAZY DANYCH – CZ.1	2
OBSŁUGA WŁASNEJ BAZY DANYCH – CZ.2	7

Obsługa własnej bazy danych – cz.1

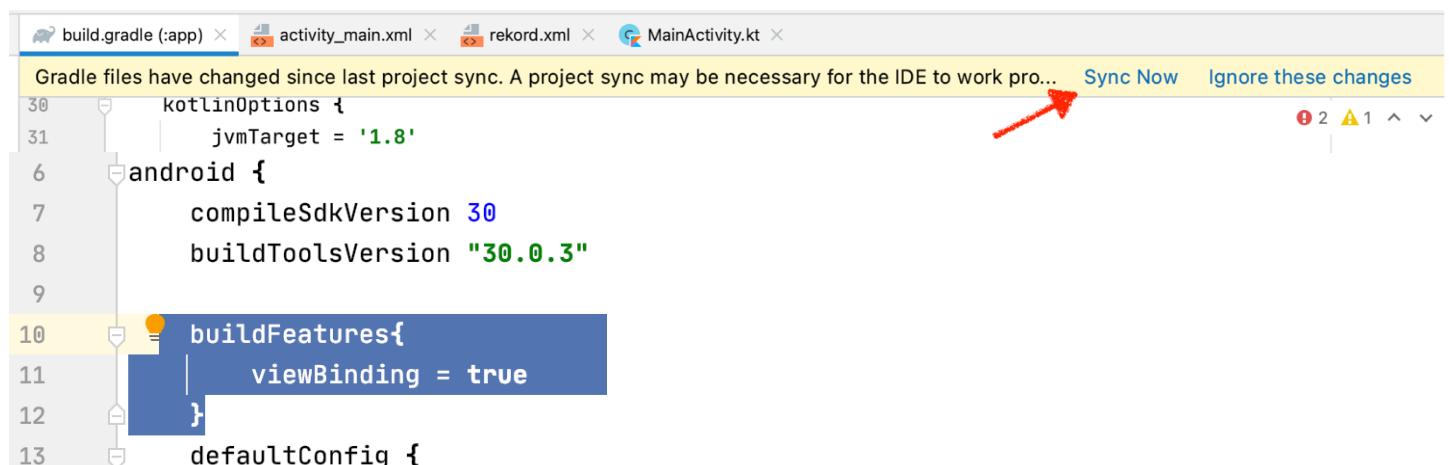
Utworzenie bazy danych w AS ma na celu zapis danych z aplikacji oraz ich odczyt. Przydatne to jest w szczególności w trakcie wyłączenia i ponownego włączenia aplikacji, która ma działać tak jak w momencie wyłączenia a nie podczas pierwszego startu aplikacji. Jednym słowem przechowujemy dane aplikacji w lokalnej bazie danych. Lokalne dane mogą być przechowywane na karcie SD lub w bazie danych.

Tworzymy aplikację – nowy kotlinowy projekt:

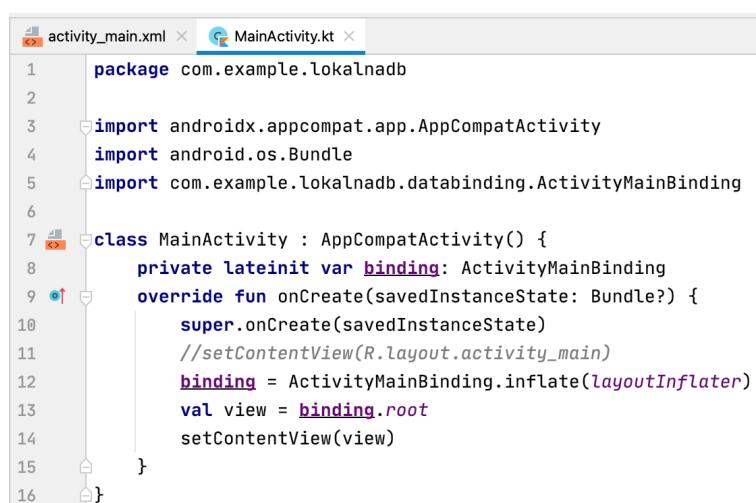


Wybieramy pustą aktywność.

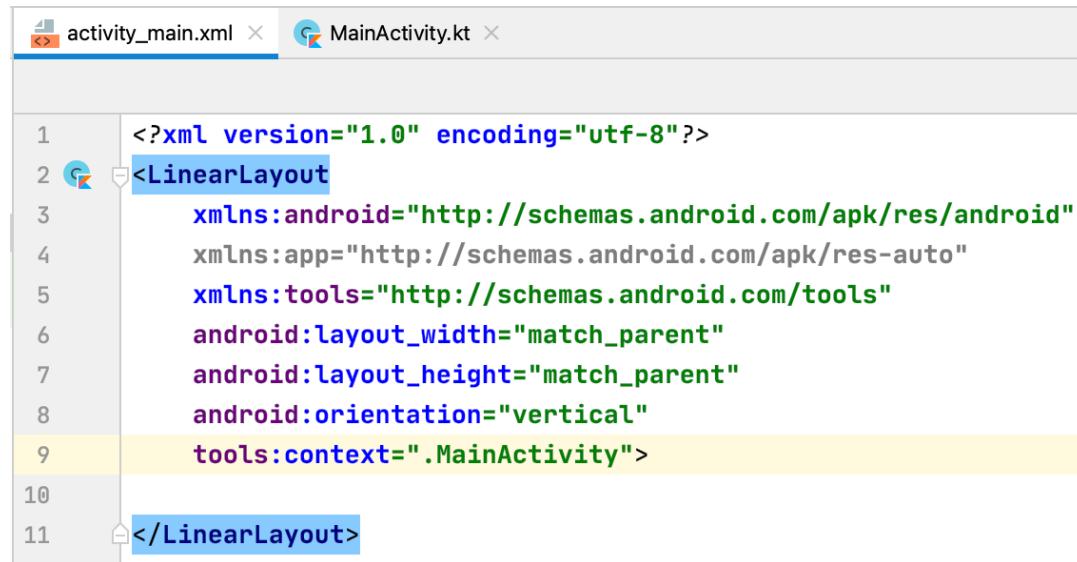
W pliku build.gradle (dla Modules) należy dopisać linie i zsynchronizować z projektem:



Tradycyjnie w MainActivity.kt należy zmodyfikować zawartość



Tworzymy LinearLayout typu vertical (oczywiście w pliku activity_main.xml):



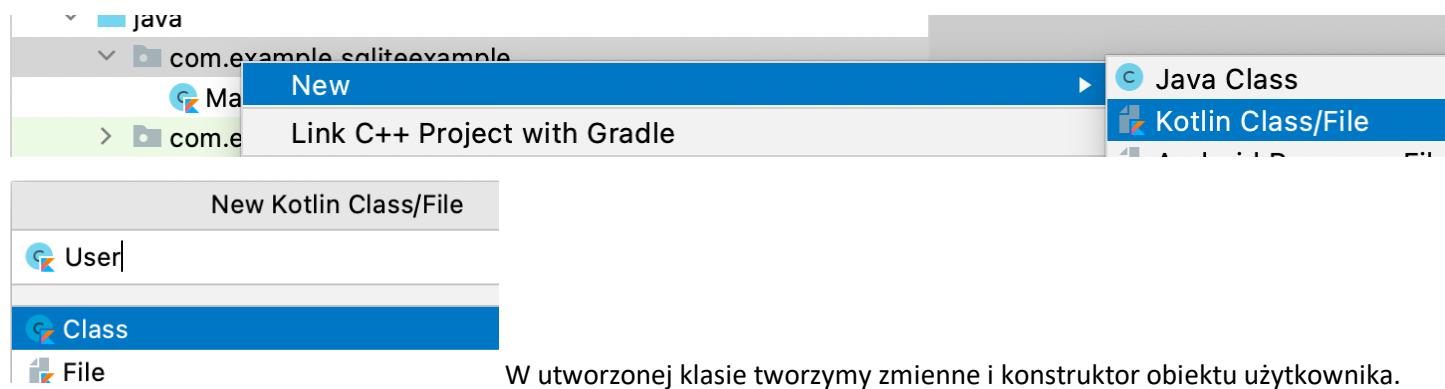
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     android:orientation="vertical"
9     tools:context=".MainActivity">
10
11 </LinearLayout>
```

W layoutie tworzymy kolejny LinearLayout: z polami edycyjnymi oraz przyciskiem zapisu.



```
10
11 <LinearLayout
12     android:layout_width="match_parent"
13     android:layout_height="wrap_content"
14     android:orientation="vertical">
15
16     <EditText
17         android:id="@+id/etName"
18         android:layout_width="match_parent"
19         android:layout_height="wrap_content"
20         android:layout_margin="10dp"
21         android:padding="10dp"/>
22
23     <EditText
24         android:id="@+id/etAge"
25         android:layout_width="match_parent"
26         android:layout_height="wrap_content"
27         android:layout_margin="10dp"
28         android:padding="10dp"
29         android:inputType="number"/>
30
31     <Button
32         android:id="@+id/btnZapisz"
33         android:layout_width="match_parent"
34         android:layout_height="wrap_content"
35         android:layout_margin="10dp"
36         android:padding="10dp"
37         android:text="@string/zapisz"/>
38
39 </LinearLayout>
```

Tworzymy klasę **User** reprezentującą dane użytkownika:

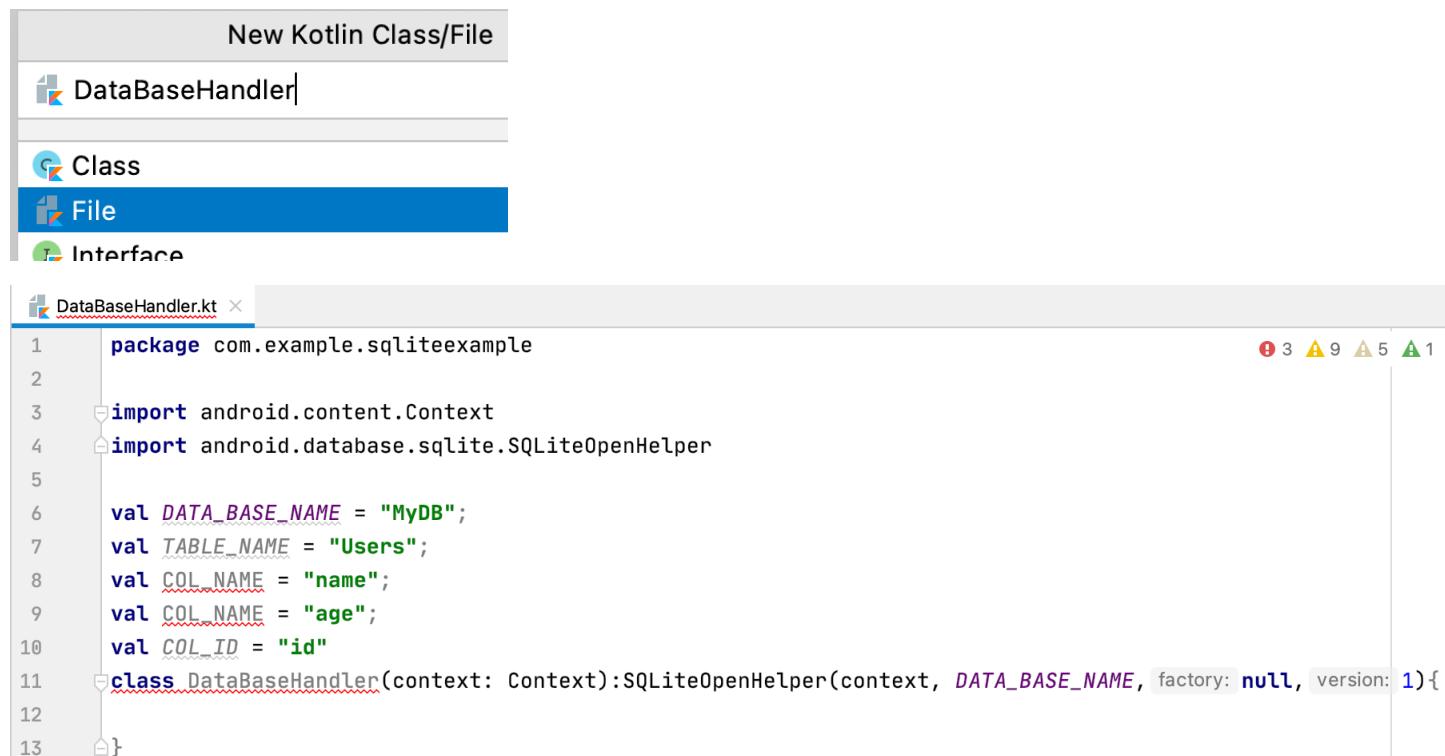


```

1 package com.example.sqliteexample
2
3 class User {
4     var id: Int = 0;
5     var name: String = "";
6     var age: Int = 0;
7     constructor(name:String,age:Int){
8         this.name = name;
9         this.age = age;
10    }
11 }

```

W podobny sposób tworzymy klasę tworzącą bazę danych:



The screenshot shows the 'New' dialog again. The 'File' category is selected. The 'DataBaseHandler' class is listed under the 'Class' category. The 'File' category is also visible. The status bar at the bottom right shows code quality metrics: 1 error (red), 3 warnings (yellow), 9 code smells (orange), 5 bugs (green), and 1 info message (blue).

```

1 package com.example.sqliteexample
2
3 import android.content.Context
4 import android.database.sqlite.SQLiteOpenHelper
5
6 val DATA_BASE_NAME = "MyDB";
7 val TABLE_NAME = "Users";
8 val COL_NAME = "name";
9 val COL_NAME = "age";
10 val COL_ID = "id"
11 class DataBaseHandler(context: Context):SQLiteOpenHelper(context, DATA_BASE_NAME, factory: null, version: 1){
12
13 }

```

Importujemy metody:

Implement Members

android.database.sqlite.SQLiteOpenHelper

- onCreate(db: SQLiteDatabase!): Unit
- onUpgrade(db: SQLiteDatabase!, oldVersion: Int, newVersion: Int): Unit

```

12  class DataBaseHandler(context: Context):SQLiteOpenHelper(context, DATA_BASE_NAME, factory: null, version: 1){
13
14     override fun onCreate(db: SQLiteDatabase?) {
15         TODO("Not yet implemented")
16     }
17
18     override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
19         TODO("Not yet implemented")
20     }
21 }

```

Metoda onCreate

```

14  class DataBaseHandler(var context: Context):SQLiteOpenHelper(context, DATA_BASE_NAME, factory: null, version: 1){
15      //tworzy bazę danych
16      override fun onCreate(db: SQLiteDatabase?) {
17          val createTable = "CREATE TABLE $TABLE_NAME (" +
18              "$COL_ID INTEGER PRIMARY KEY AUTOINCREMENT," +
19              "$COL_NAME VARCHAR(256)," +
20              "$COL_AGE INTEGER)";
21          db?.execSQL(createTable)
22      }

```

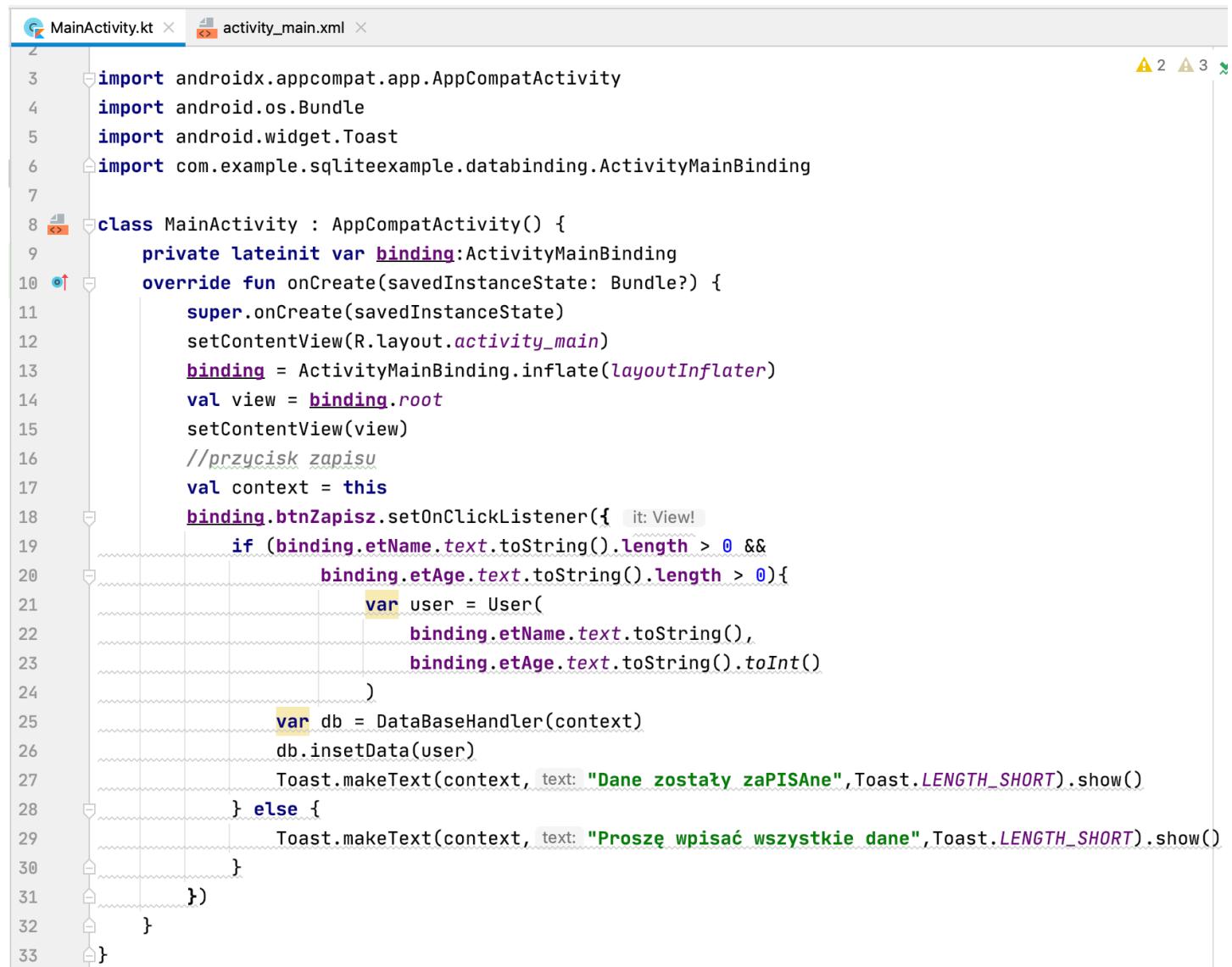
Budujemy też funkcję obsługującą przycisk i wpisującą dane do tabeli:

```

27  //funkcja wstawiajaca dane
28  fun insetData(user:User){
29      val db = this.writableDatabase
30      var cv = ContentValues()
31      cv.put(COL_NAME,user.name)
32      cv.put(COL_AGE,user.age)
33      var result = db.insert(TABLE_NAME, nullColumnHack: null, cv)
34      if (result == -1.toLong()){
35          Toast.makeText(context, text: "Failed",Toast.LENGTH_SHORT).show()
36      } else{
37          Toast.makeText(context, text: "Success",Toast.LENGTH_SHORT).show()
38      }
39  }

```

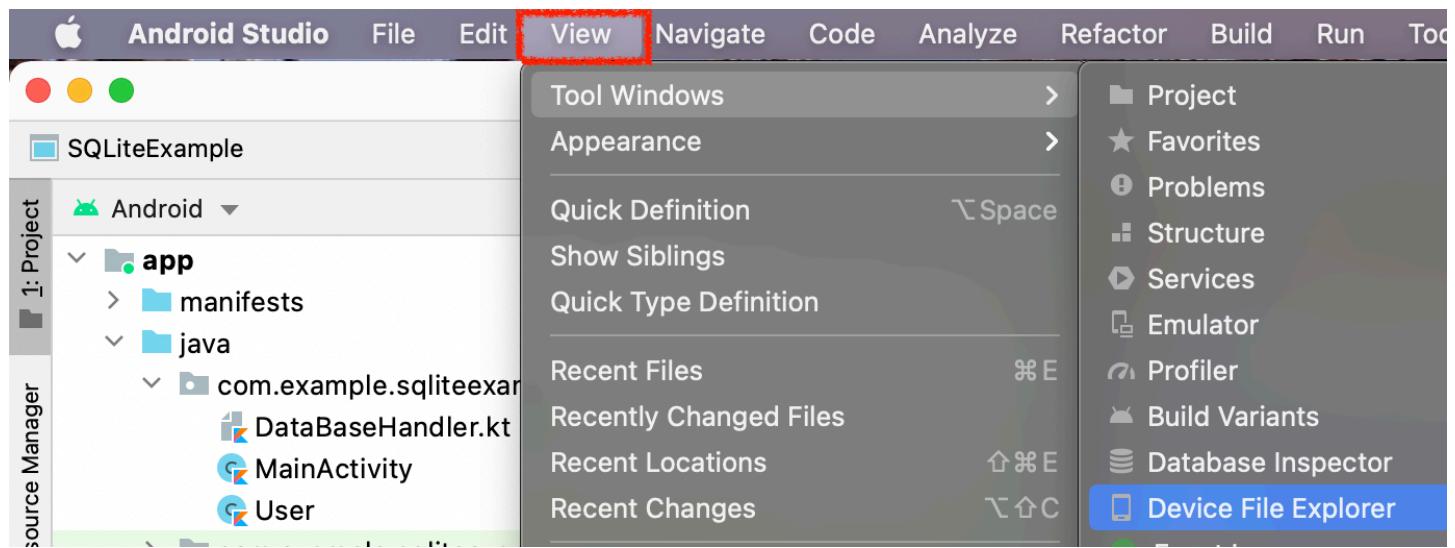
Zmieniamy zachowanie aplikacji po kliknięciu w przycisk wpisujący dane. Przechodzimy do MainActivity.kt :



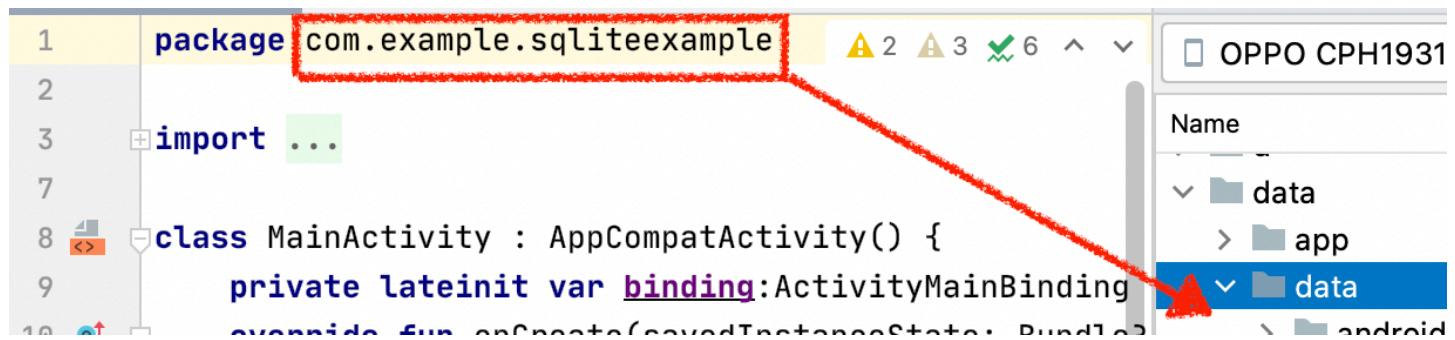
```
2
3 import androidx.appcompat.app.AppCompatActivity
4 import android.os.Bundle
5 import android.widget.Toast
6 import com.example.sqliteexample.databinding.ActivityMainBinding
7
8 class MainActivity : AppCompatActivity() {
9     private lateinit var binding: ActivityMainBinding
10    override fun onCreate(savedInstanceState: Bundle?) {
11        super.onCreate(savedInstanceState)
12        setContentView(R.layout.activity_main)
13        binding = ActivityMainBinding.inflate(layoutInflater)
14        val view = binding.root
15        setContentView(view)
16        //przycisk zapisu
17        val context = this
18        binding.btnZapisz.setOnClickListener{ it: View!
19            if (binding.etName.text.toString().length > 0 &&
20                binding.etAge.text.toString().length > 0){
21                var user = User(
22                    binding.etName.text.toString(),
23                    binding.etAge.text.toString().toInt()
24                )
25                var db = DataBaseHandler(context)
26                db.insertData(user)
27                Toast.makeText(context, text: "Dane zostały zaPISane", Toast.LENGTH_SHORT).show()
28            } else {
29                Toast.makeText(context, text: "Proszę wpisać wszystkie dane", Toast.LENGTH_SHORT).show()
30            }
31        }
32    }
33}
```

Obsługa własnej bazy danych – cz.2

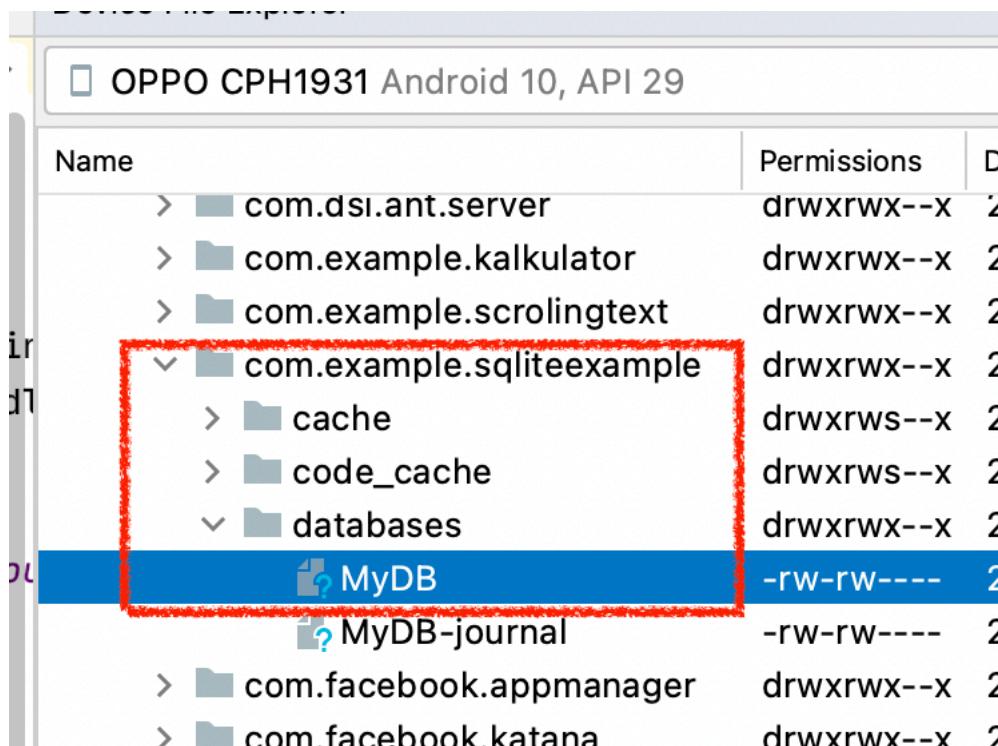
Aby sprawdzić, czy udało się utworzyć bazę danych wybieramy:



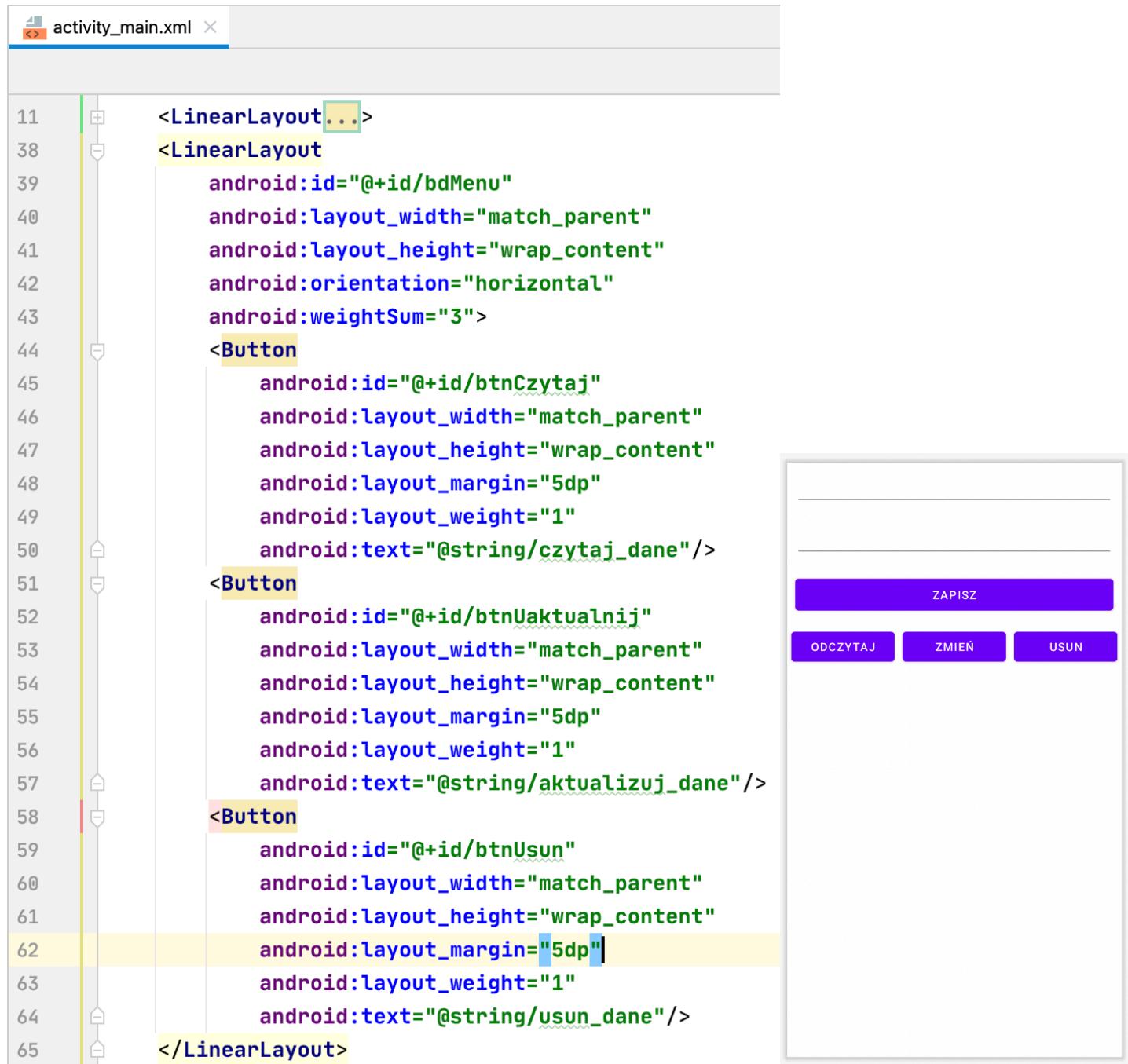
Odszukujemy nasz projekt:



Tam powinna być widoczna baza danych o nadanej przez nas nazwie:



Utworzymy Layout na kolejne przyciski.

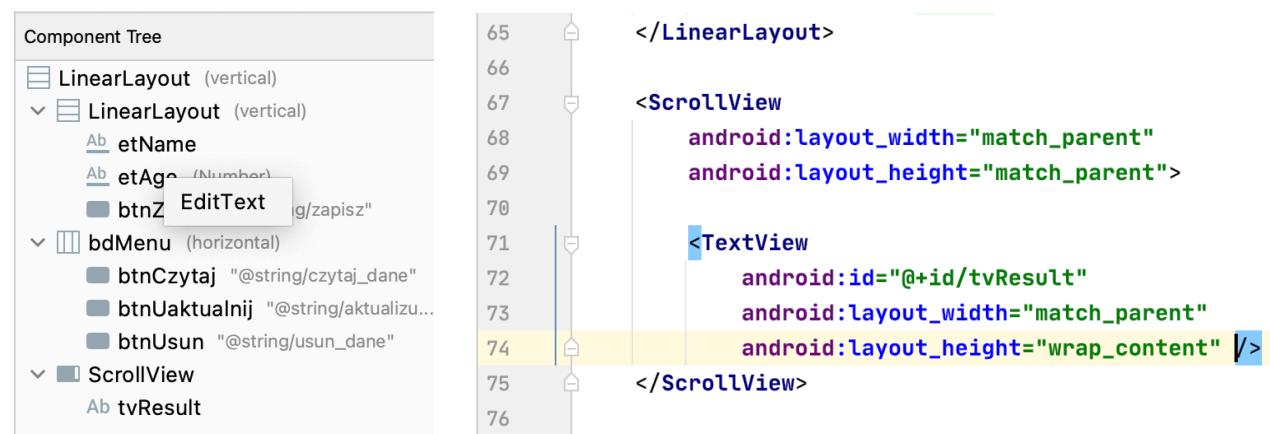


```

11 <LinearLayout...>
38 <LinearLayout
39     android:id="@+id/bdMenu"
40     android:layout_width="match_parent"
41     android:layout_height="wrap_content"
42     android:orientation="horizontal"
43     android:weightSum="3">
44     <Button
45         android:id="@+id/btnCzytaj"
46         android:layout_width="match_parent"
47         android:layout_height="wrap_content"
48         android:layout_margin="5dp"
49         android:layout_weight="1"
50         android:text="@string/czytaj_dane"/>
51     <Button
52         android:id="@+id/btnUaktualnij"
53         android:layout_width="match_parent"
54         android:layout_height="wrap_content"
55         android:layout_margin="5dp"
56         android:layout_weight="1"
57         android:text="@string/aktualizuj_dane"/>
58     <Button
59         android:id="@+id/btnUsun"
60         android:layout_width="match_parent"
61         android:layout_height="wrap_content"
62         android:layout_margin="5dp"
63         android:layout_weight="1"
64         android:text="@string/usun_dane"/>
65     </LinearLayout>

```

Stworzymy w naszym Layoucie pole wyświetlania zawartości bazy danych. Umożliwimy przewijanie tej części aplikacji za pomocą ScrollView:



```

Component Tree
  LinearLayout (vertical)
    LinearLayout (vertical)
      etName
      etAge
      btnZ
    bdMenu (horizontal)
      btnCzytaj
      btnUaktualnij
      btnUsun
    ScrollView
      tvResult

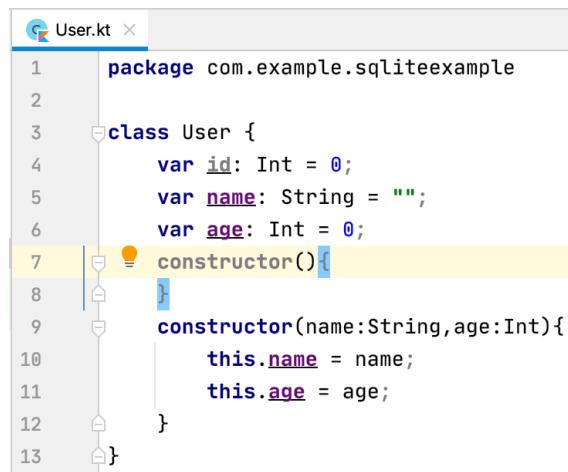
```

```

65 </LinearLayout>
66
67 <ScrollView
68     android:layout_width="match_parent"
69     android:layout_height="match_parent">
70
71     <TextView
72         android:id="@+id/tvResult"
73         android:layout_width="match_parent"
74         android:layout_height="wrap_content" />
75     </ScrollView>
76

```

Dodamy do klasy User konstruktor bez atrybutów – wprowadzimy przeciążenie konstruktora:



```

1 package com.example.sqliteexample
2
3 class User {
4     var id: Int = 0;
5     var name: String = "";
6     var age: Int = 0;
7     constructor(id: Int, name: String, age: Int) {
8     }
9     constructor(name: String, age: Int) {
10        this.name = name;
11        this.age = age;
12    }
13 }

```

Teraz trzeba utworzyć funkcje obsługujące zdarzenia nowych przycisków:



```

28     fun insertData(user:User){...}
29
30     fun readData(): MutableList<User>{
31         var list : MutableList<User> = ArrayList()
32         val db = this.readableDatabase
33         val query = "SELECT * FROM $TABLE_NAME"
34         val result = db.rawQuery(query, selectionArgs: null)
35         if(result.moveToFirst()){
36             do {
37                 var user = User()
38                 user.id = result.getString(result.getColumnIndex(COL_ID)).toInt()
39                 user.name = result.getString(result.getColumnIndex(COL_NAME))
40                 user.id = result.getString(result.getColumnIndex(COL_ID)).toInt()
41                 list.add(user)
42             } while (result.moveToNext());
43         }
44         result.close()
45         db.close()
46         return list
47     }

```

Do funkcji potrzebne jest zachowanie przycisku wywołujące tą funkcję. W MainActivity.kt dopisujemy metodę:

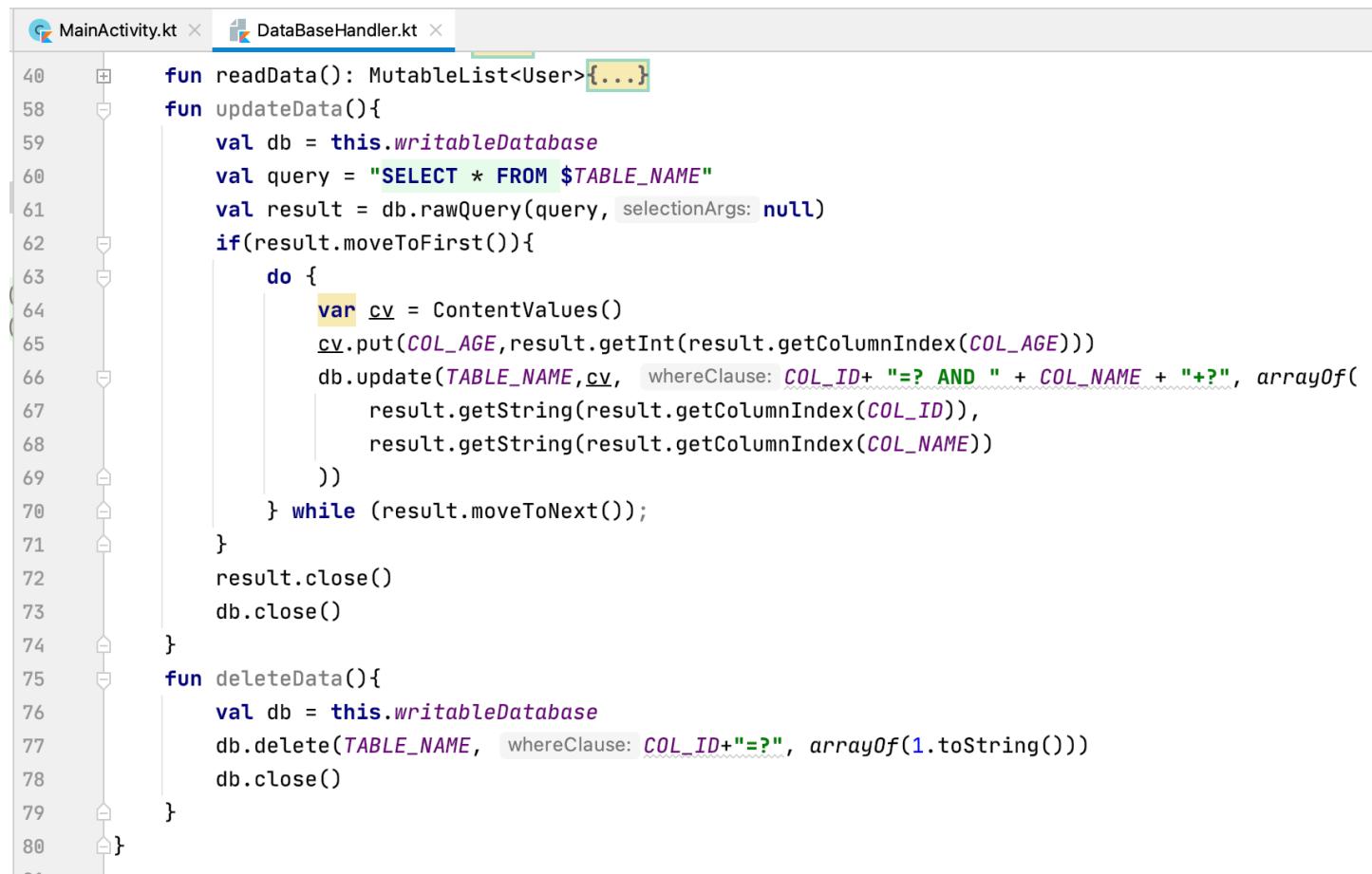


```

31     }
32     binding.btnCzytaj.setOnClickListener{ it: View!
33         var db = DataBaseHandler(context)
34         var data = db.readData()
35         binding.tvResult.text = ""
36         for(i in 0..(data.size-1)){
37             binding.tvResult.append(data.get(i).id.toString() + " " + data.get(i).name.toString() + " "
38             + data.get(i).age.toString() + "\n")
39         }
40     }
41 }
42

```

Utworzymy funkcje: usuwającą rekordy w bazie danych oraz modyfikującą:

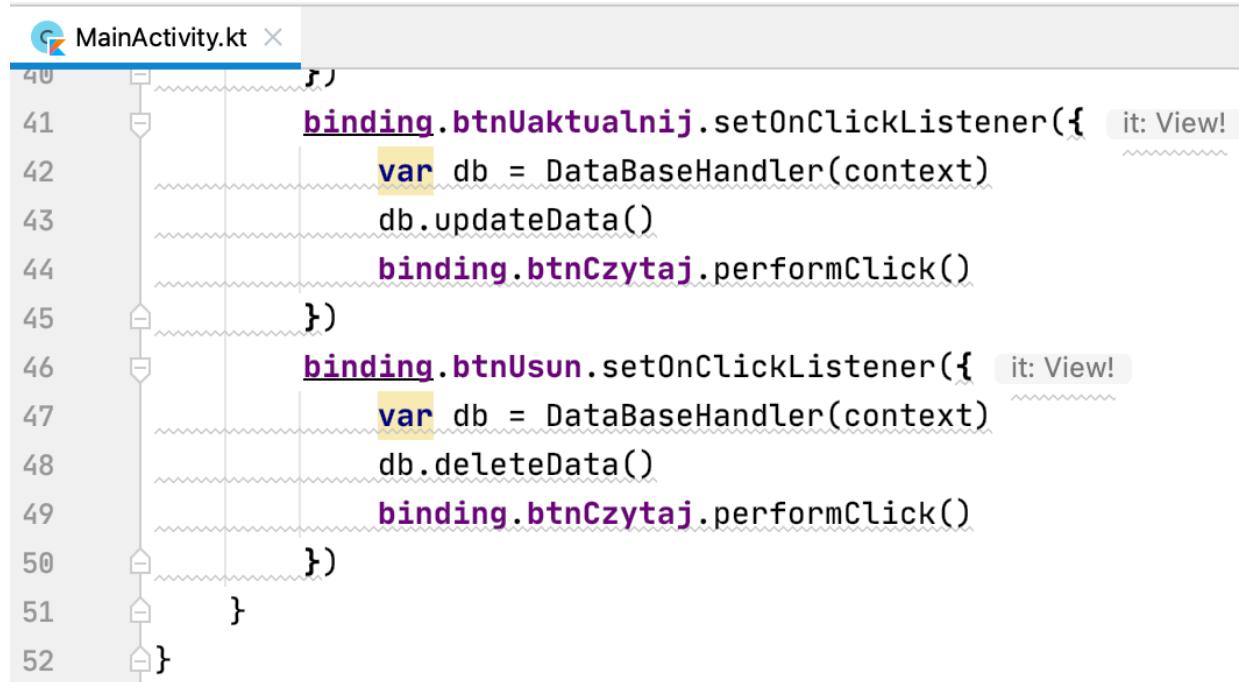


```

40     fun readData(): MutableList<User>{...}
58     fun updateData(){
59         val db = this.writableDatabase
60         val query = "SELECT * FROM $TABLE_NAME"
61         val result = db.rawQuery(query, selectionArgs: null)
62         if(result.moveToFirst()){
63             do {
64                 var cv = ContentValues()
65                 cv.put(COL_AGE, result.getInt(result.getColumnIndex(COL_AGE)))
66                 db.update(TABLE_NAME, cv, whereClause: COL_ID+ "=? AND " + COL_NAME + "+?", arrayOf(
67                     result.getString(result.getColumnIndex(COL_ID)),
68                     result.getString(result.getColumnIndex(COL_NAME))
69                 ))
70             } while (result.moveToNext());
71         }
72         result.close()
73         db.close()
74     }
75     fun deleteData(){
76         val db = this.writableDatabase
77         db.delete(TABLE_NAME, whereClause: COL_ID+"=?", arrayOf(1.toString()))
78         db.close()
79     }
80 }

```

Oraz zachowanie aplikacji (MainActivity.kt), czyli pozostałynych przycisków:



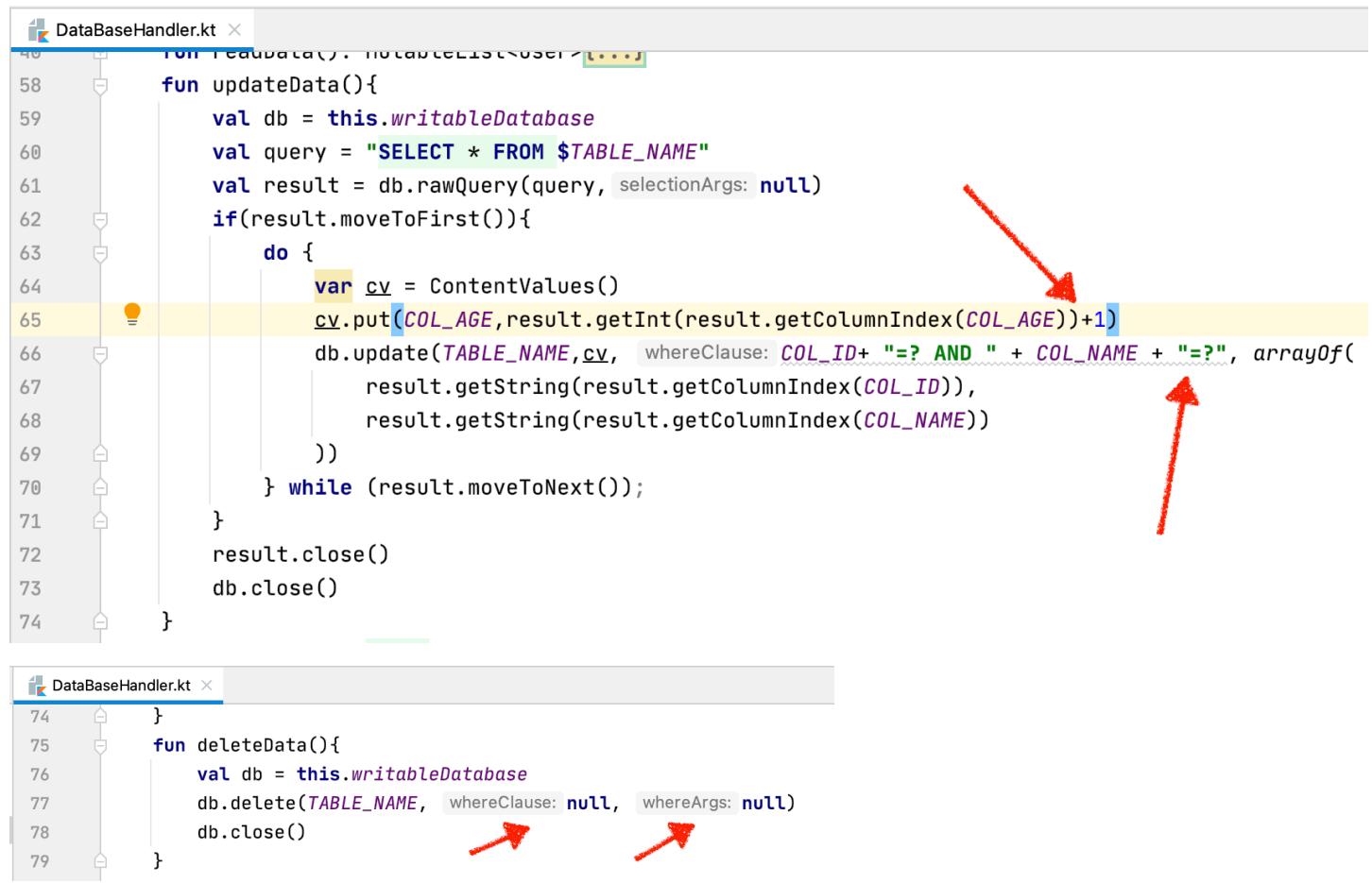
```

40
41     binding.btnUaktualnij.setOnClickListener{ it: View!
42         var db = DataBaseHandler(context)
43         db.updateData()
44         binding.btnCzytaj.performClick()
45     }
46     binding.btnUsun.setOnClickListener{ it: View!
47         var db = DataBaseHandler(context)
48         db.deleteData()
49         binding.btnCzytaj.performClick()
50     }
51 }
52

```

Aplikacja powinna już działać. Brakuje jeszcze inkrementacji rekordów bazy.

Modyfikujemy zatem plik DataBaseHandler:



```

40 DataBaseHandler.kt
41
42     fun updateData(){
43
44         val db = this.writableDatabase
45
46         val query = "SELECT * FROM $TABLE_NAME"
47
48         val result = db.rawQuery(query, selectionArgs: null)
49
50         if(result.moveToFirst()){
51
52             do {
53
54                 val cv = ContentValues()
55
55                 cv.putInt(COL_AGE, result.getInt(result.getColumnIndex(COL_AGE))+1)
56
57                 db.update(TABLE_NAME, cv, whereClause: COL_ID+ "=? AND "+ COL_NAME + "=?", arrayOf(
58
59                     result.getString(result.getColumnIndex(COL_ID)),
60                     result.getString(result.getColumnIndex(COL_NAME)))
61
62             } while (result.moveToNext());
63
64         }
65
66         result.close()
67         db.close()
68
69     }
70
71
72
73
74

```



```

74     }
75
76     fun deleteData(){
77
78         val db = this.writableDatabase
79
80         db.delete(TABLE_NAME, whereClause: null, whereArgs: null)
81
82         db.close()
83
84     }
85
86
87
88
89

```