UNIWERSYTET RADOMSKI
im. Kazimierza Pułaskiego

PROGRAMOWANI APLIKACJI MOBILNYCH

Wykład

dr Artur Bartoszewski

Baza danych SQLite

SQLite

SQLite jest lekką, wbudowaną bazą danych, która nie wymaga instalacji serwera i działa bezpośrednio na urządzeniu. Jest idealnym rozwiązaniem dla aplikacji mobilnych na Androida, gdzie wymagane jest przechowywanie danych lokalnie.

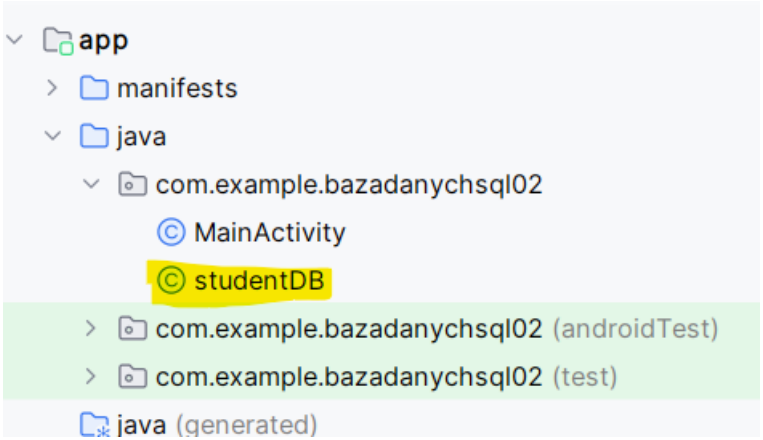
Kluczowe klasy i narzędzia w Androidzie:

- SQLiteOpenHelper: Klasa służąca do zarządzania bazą danych. Pomaga w tworzeniu, aktualizowaniu oraz usuwaniu tabel.
- SQLiteDatabase: Reprezentuje bazę danych i zapewnia metody do wykonywania operacji SQL.

Klasa obsługująca bazę danych

Pierwszym krokiem jest utworzenie klasy, która dziedziczy po SQLiteOpenHelper.

W niej definiujemy schemat bazy danych oraz jej inicjalizację.

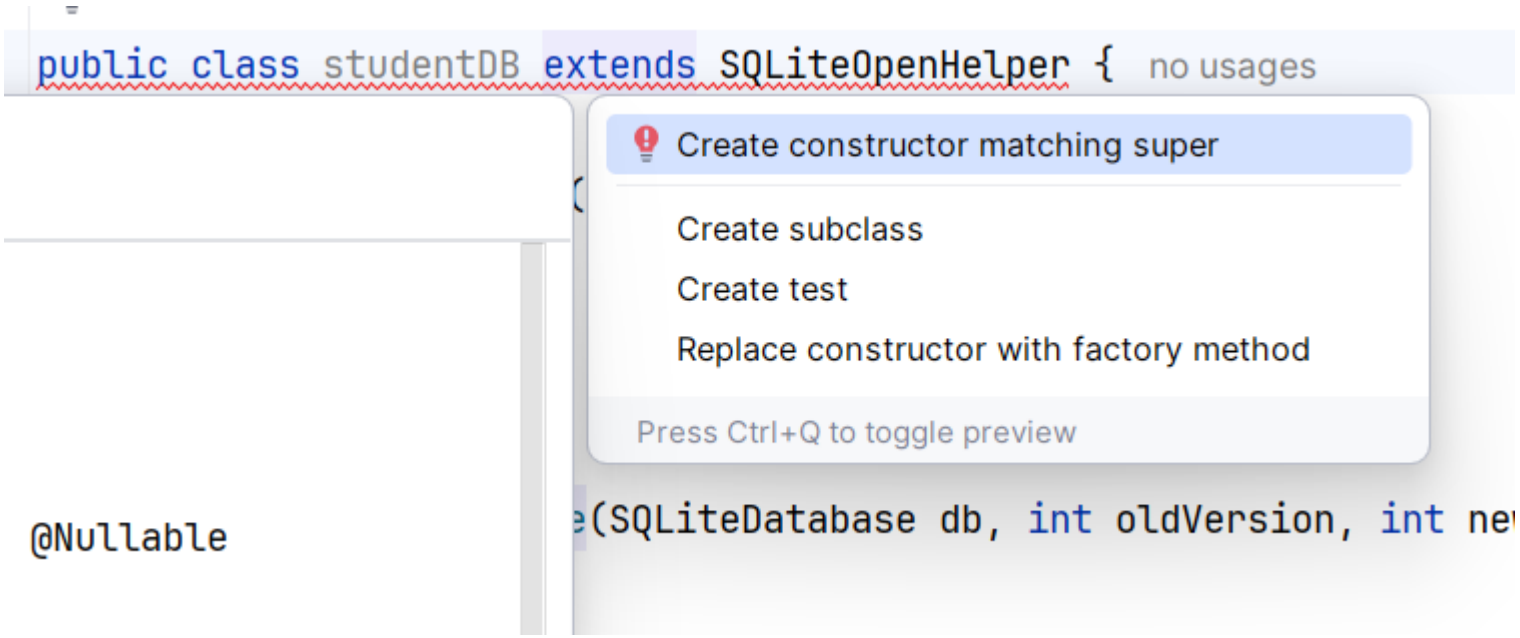



```
1  package com.example.bazadanychsql02;
2
3  import android.database.sqlite.SQLiteOpenHelper;
4
5  public class studentDB extends SQLiteOpenHelper
6
7
```

Implement methods
Make 'studentDB' abstract
Safe delete 'com.example.bazadanychsql02.studentDB'
Create subclass
Create test
Make 'studentDB' package-private
Replace constructor with factory method
Add Javadoc
Press Ctrl+Q to toggle preview

Klasa obsługująca bazę danych

Jeżeli stworzymy klasę rozszerzającą interfejs SQLiteOpenHelper to środowisko automatycznie zaproponuje nam najpierw implementację metod a potem konstruktorów.



Klasa obsługująca bazę danych

W tej klasie definiujemy schemat bazy danych oraz jej inicjalizację

```
//nazwa bazy danych
private static final String DATABASE_NAME = "students.db";
private static final int DATABASE_VERSION = 1;

// Definiowanie struktury tabeli
private static final String TABLE_STUDENCI = "students";
private static final String COLUMN_ID = "_id";
private static final String COLUMN_IMIE_NAZWISKO = "name";
private static final String COLUMN_SEMESTR = "sem";
```

Klasa obsługująca bazę danych

Następnie warto przygotować bardziej skomplikowane zapytania SQL w postaci stałych. Żeby nie definiować ich w kodzie metod.

```
CREATE TABLE students (
        _id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT,
        sem INTEGER
);
```

students

_id	INTEGER
name	TEXT
sem	INTEGER

```
// tworzenie zapytania SQL które stworzy tabelę
private static final String TABLE_CREATE =
        "CREATE TABLE " + TABLE_STUDENCI + " (" +
                COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                COLUMN_IMIE_NAZWISKO + " TEXT, " +
                COLUMN_SEMESTR + " INTEGER);";
```

Klasa obsługująca bazę danych

Kolejnym krokiem jest modyfikacja konstruktora. Konstruktor domyślnie utworzony przez środowisko przystosowany jest do tego, aby w parametrach otrzymać dane które w naszym przykładzie umieściliśmy wewnątrz klasy jako stałe (nazwa bazy itp.).

```
public studentDB(@Nullable Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

Konstruktor uruchamiany będzie tylko z parametrem kontekst.

Klasa obsługująca bazę danych

Następnie wypełniamy utworzone przez środowisko metody onCreate i onUpgrade

```
@Override
//tworzenie tabeli
public void onCreate(SQLiteDatabase db) {
    db.execSQL(TABLE_CREATE); // Tworzenie tabeli
}
```

Do stworzenia tabeli używamy wcześniej przygotowanego zapytania.

```
@Override
//modyfikowanie tabeli
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Usuwanie tabeli jeśli istnieje
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_STUDENCI);
    onCreate(db); // Tworzenie tabeli na nowo
}
```

Metoda otrzymuje w parametrze zmienną db która jest nową, zmodyfikowaną tabelę bazą.

Realizacja operacji CRUD

CRUD to akronim opisujący cztery podstawowe operacje wykonywane na danych:

- Create (tworzenie) – dodawanie nowych danych do bazy.
- Read (odczyt) – pobieranie danych z bazy.
- Update (aktualizacja) – modyfikacja istniejących danych.
- Delete (usuwanie) – usuwanie danych z bazy.

W operacjach CRUD przydatne są dwa typy zmiennych złożonych:

- Cursor
- ContentValues.

Realizacja operacji CRUD

Cursor to interfejs używany do reprezentowania zestawu wyników zapytania z bazy danych SQL.

Umożliwia przechodzenie po wierszach (rekordach) i dostęp do poszczególnych kolumn w każdym wierszu.

```
 Cursor cursor = db.rawQuery(query, null);
if (cursor.moveToFirst()) {     //ustawienie kursora na pierwszy rekord
      do {
         String name = cursor.getString(cursor.getColumnIndex("name"));
         int age = cursor.getInt(cursor.getColumnIndex("age"));
         // Przetwarzanie danych...
      } while (cursor.moveToNext());   // przesunięcie kursora na kolejny rekord
   }
cursor.close();  // Należy zamknąć Cursor po zakończeniu pracy
```

Realizacja operacji CRUD

ContentValues to klasa w Androidzie używana do przechowywania par klucz-wartość, które są wykorzystywane przy operacjach na bazach danych SQLite, takich jak insert(), update().

Jest to wygodny sposób na przekazywanie danych do zapytań SQL bez potrzeby bezpośredniego pisania SQL-owego kodu.

ContentValues służy do mapowania kolumn na wartości, które mają być zapisane w bazie danych.

- kluczami są nazwy kolumn (typu String),
- wartościami mogą być różne typy danych, takie jak String, Integer, Boolean, Float, itp.

```
ContentValues values = new ContentValues();
values.put("name", "John Doe");
values.put("age", 25);

SQLiteDatabase db = this.getWritableDatabase();
db.insert("students", null, values);
db.close();
```

Realizacja operacji CRUD

Do realizacji podstawowych operacji CRUD na obiekcie klasy SQLiteDatabase używamy następujących metod:

- Insert: insert()
- Read: query(), rawQuery()
- Update: update()
- Delete: delete()
- Inne operacje: execSQL()

Realizacja operacji CRUD

insert(String table, String nullColumnHack, ContentValues values)

Wstawia nowy wiersz do tabeli.

Parametry:

- table: nazwa tabeli.
- nullColumnHack: kolumna, która zostanie wypełniona NULL, jeśli wartości są puste.
- values: obiekt ContentValues, zawierający dane do wstawienia

```
db.insert("students", null, contentValues);
```

Realizacja operacji CRUD

update(String table, ContentValues values, String whereClause, String[] whereArgs)

Aktualizuje istniejące wiersze w tabeli.

Parametry:

- table: nazwa tabeli.
- values: obiekt ContentValues, zawierający nowe wartości.
- whereClause: opcjonalny warunek WHERE dla aktualizacji.
- whereArgs: tablica argumentów dla warunku WHERE.

```
db.update("students", contentValues, "id = ?", new String[] {String.valueOf(studentId)});
```

Realizacja operacji CRUD

UWAGA: Wyjaśnienie dotyczące składni polecenia.

W wyrażeniu "id = ?", znak zapytania (?) jest symbolem zastępczym (tzw. placeholder) dla wartości, która zostanie dostarczona później.

Jest to technika stosowana w zapytaniach SQL w Androidzie (np. w metodach update(), delete() lub rawQuery())

- Wartości są przekazywane osobno od struktury zapytania, co zmniejsza ryzyko ataków SQL Injection.
- Wartości dynamiczne są wstawiane w prosty i bezpieczny sposób.

Przykład:

```
db.delete("students", "id = ?", new String[] { String.valueOf(studentId) });
```

- "id = ?" to warunek WHERE, który mówi, że operacja ma dotyczyć wiersza, gdzie kolumna id jest równa podanemu studentId.
- new String[] { String.valueOf(studentId) }: dostarcza rzeczywistą wartość dla symbolu zastępczego ?, w tym przypadku wartość identyfikatora studenta.

Realizacja operacji CRUD

delete(String table, String whereClause, String[] whereArgs)

Usuwa wiersze z tabeli na podstawie warunku WHERE.

Parametry:

- table: nazwa tabeli.
- whereClause: opcjonalny warunek WHERE określający, które wiersze usunąć.
- whereArgs: tablica argumentów dla warunku WHERE

```
db.delete("students", "id = ?", new String[] {String.valueOf(studentId)});
```

Realizacja operacji CRUD

query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)

Wykonuje zapytanie **SELECT** na bazie danych, zwracając dane w obiekcie Cursor.

- Parametry: table: nazwa tabeli.
- columns: lista kolumn, które mają być pobrane.
- selection: opcjonalny warunek WHERE.
- selectionArgs: tablica argumentów dla warunku WHERE.
- groupBy: opcjonalna klauzula GROUP BY.
- having: opcjonalna klauzula HAVING.orderBy: opcjonalna klauzula ORDER BY.

```
Cursor cursor = db.query("students", new String[]{"name", "age"}, "age > ?", new String[]{"18"},
null, null, "name ASC");
```

Realizacja operacji CRUD

rawQuery(String sql, String[] selectionArgs)

Wykonuje dowolne zapytanie SQL, zwracając dane w obiekcie Cursor.

Parametry:

- sql: pełne zapytanie SQL.
- selectionArgs: argumenty dla warunku w zapytaniu SQL.

```
Cursor cursor = db.rawQuery("SELECT * FROM students WHERE age > ?", new String[] {"18"});
```

Realizacja operacji CRUD

execSQL(String sql)

Wykonuje dowolne zapytanie SQL, które nie zwraca wyników (np. CREATE, UPDATE, DELETE, INSERT).

Parametry:

- sql: pełne zapytanie SQL.

```
db.execSQL("DELETE FROM students WHERE age < 18");
```

Realizacja operacji CRUD

Create (tworzenie) – dodawanie nowych danych do bazy.

```
public void insertStudent(String name, int sem) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COLUMN_IMIE_NAZWISKO, name);
    values.put(COLUMN_SEMESTR, sem);
    db.insert(TABLE_STUDENCI, null, values);
    db.close();
}
```

Realizacja operacji CRUD

Read (odczyt) – pobieranie danych z bazy.

```
public Cursor getAllStudents() {
    SQLiteDatabase db = this.getReadableDatabase();
    String query = "SELECT * FROM " + TABLE_STUDENCI;
    return db.rawQuery(query, null);
}
```

Realizacja operacji CRUD

Update (aktualizacja) – modyfikacja istniejących danych.

```
// zmiana danych istniejącego rekordu
    public int updateStudent(int id, String name, int age) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(COLUMN_IMIE_NAZWISKO, name);
        values.put(COLUMN_SEMESTR, age);
        return db.update(TABLE_STUDENCI, values, COLUMN_ID + " = ?", new String[]{String.valueOf(id)});
    }
```

Realizacja operacji CRUD

Delete (usuwanie) – usuwanie danych z bazy.

```
public void deleteStudent(int id) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_STUDENCI, COLUMN_ID + " = ?", new String[]{String.valueOf(id)});
    db.close();
}

public void deleteAllStudents() {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_STUDENCI, null, null);
    db.close();
}
```

Realizacja operacji CRUD

Obsługa wyjątków i transakcji

Warto dodać obsługę wyjątków oraz transakcje, aby zapobiec częściowym aktualizacjom danych w przypadku błędów.

Transakcje SQLite pozwalają na grupowanie wielu operacji bazodanowych (np. INSERT, UPDATE, DELETE) w jeden logiczny zestaw, który może być albo w całości zatwierdzony, albo cofnięty w przypadku błędu.

1. beginTransaction() - rozpoczyna transakcję.
2. Wykonujemy operacje na bazie danych (np. wstawianie, aktualizowanie danych).
3. Jeśli wszystkie operacje zakończą się sukcesem, metoda setTransactionSuccessful() jest wywoływana, aby oznaczyć transakcję jako poprawną.
4. endTransaction() - kończy transakcję. Jeśli wywołano setTransactionSuccessful(), zmiany są zatwierdzane. W przeciwnym razie zostają cofnięte (rollback).

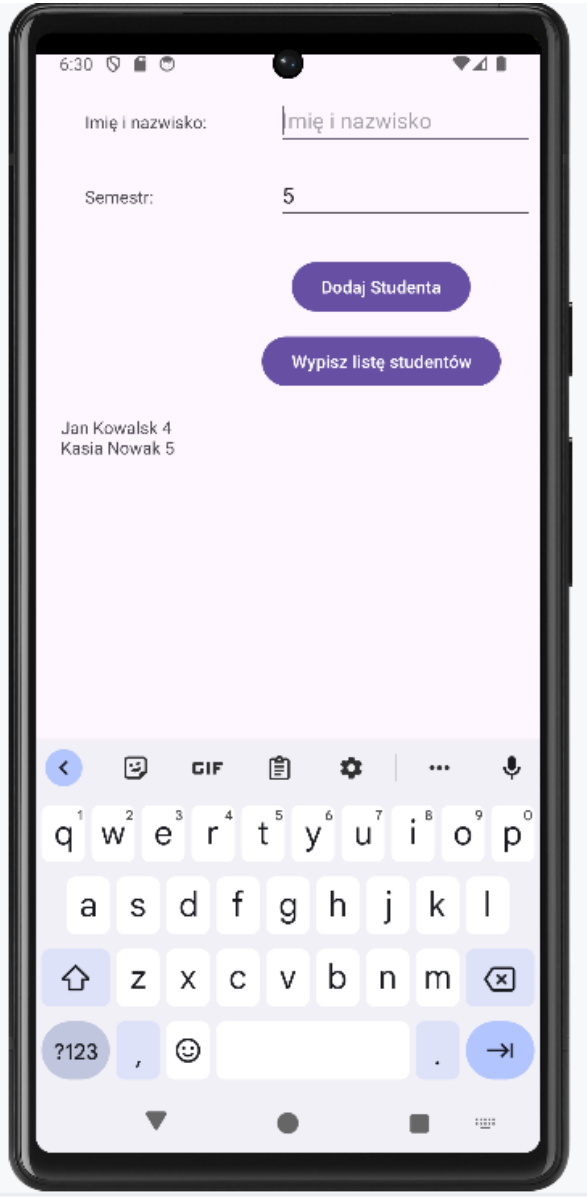
Realizacja operacji CRUD

Przykład użycia transakcji:

```
 SQLiteDatabase db = this.getWritableDatabase();
db.beginTransaction();
try {
        // Operacje na bazie danych, np. wstawianie danych
        db.insert("students", null, values1);
        db.insert("courses", null, values2);
        // Oznacz transakcję jako udaną
        db.setTransactionSuccessful();
    } catch (Exception e) {
        // W przypadku błędu zmiany zostaną cofnięte
        e.printStackTrace();
    } finally {
        // Kończymy transakcję
        db.endTransaction();
    }
```

Przykład:

Pełny kod przykładu, którego fragmenty zastosowano w prezentacji



Przykład

```
public class studentDB extends SQLiteOpenHelper {
    //nazwa bazy danych
    private static final String DATABASE_NAME = "students.db";
    private static final int DATABASE_VERSION = 1;

    // Definiowanie struktury tabeli
    private static final String TABLE_STUDENCI = "students";
    private static final String COLUMN_ID = "_id";
    private static final String COLUMN_IMIE_NAZWISKO = "name";
    private static final String COLUMN_SEMESTR = "sem";

    // tworzenie zapytania SQL które stworzy tabelę
    private static final String TABLE_CREATE =
            "CREATE TABLE " + TABLE_STUDENCI + " (" +
                    COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    COLUMN_IMIE_NAZWISKO + " TEXT, " +
                    COLUMN_SEMESTR + " INTEGER);";

    public studentDB(@Nullable Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    //tworzenie tabeli
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(TABLE_CREATE); // Tworzenie tabeli
    }
    @Override
    //modyfikowanie tabeli
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Usuwanie tabeli jesli istnieje
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_STUDENCI);
        onCreate(db); // Tworzenie tabeli na nowo
    }
    // dodawanie rekordu
    public void insertStudent(String name, int sem) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(COLUMN_IMIE_NAZWISKO, name);
        values.put(COLUMN_SEMESTR, sem);
        db.insert(TABLE_STUDENCI, null, values);
        db.close();
    }
    public Cursor getAllStudents() {
        SQLiteDatabase db = this.getReadableDatabase();
        String query = "SELECT * FROM " + TABLE_STUDENCI;
        return db.rawQuery(query, null);
    }
    // zmiana danych istniejącego rekordu
    public int updateStudent(int id, String name, int age) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(COLUMN_IMIE_NAZWISKO, name);
        values.put(COLUMN_SEMESTR, age);
        return db.update(TABLE_STUDENCI, values, COLUMN_ID + " = ?", new String[]{String.valueOf(id)});
    }
    public void deleteStudent(int id) {
        SQLiteDatabase db = this.getWritableDatabase();
        db.delete(TABLE_STUDENCI, COLUMN_ID + " = ?", new String[]{String.valueOf(id)});
        db.close();
    }
    public void deleteAllStudents() {
        SQLiteDatabase db = this.getWritableDatabase();
        db.delete(TABLE_STUDENCI, null, null);
        db.close();
    }
}
```

Przykład


```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_begin="152dp" />
    <EditText
        android:id="@+id/editText01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:ems="10"
        android:inputType="text"
        android:hint="Imię i nazwisko"
        android:layout_marginStart="40dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="@+id/guideline"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/textView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="40dp"
        android:text="Imię i nazwisko:"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintBaseline_toBaselineOf="@id/editText01"
        />
```

Przykład

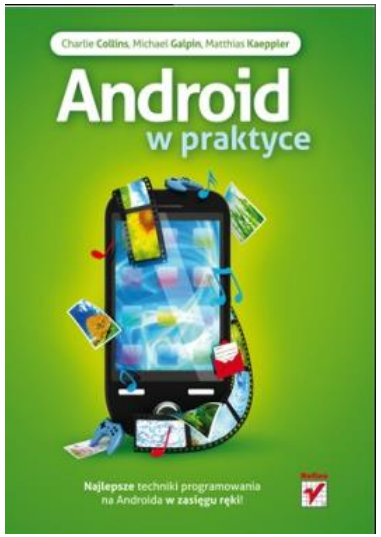
```
<EditText
    android:id="@+id/editText02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:ems="10"
    android:inputType="number"
    android:hint="Semestr"
    app:layout_constraintEnd_toEndOf="@+id/editText01"
    app:layout_constraintStart_toStartOf="@+id/editText01"
    app:layout_constraintTop_toBottomOf="@+id/editText01" />

 <TextView
    android:id="@+id/textView02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Semestr:"
    app:layout_constraintStart_toStartOf="@+id/textView01"
    app:layout_constraintBaseline_toBaselineOf="@id/editText02" />


 <Button
    android:id="@+id/button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="28dp"
    android:text="Dodaj Studenta"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="@+id/guideline"
    app:layout_constraintTop_toBottomOf="@+id/editText02" />
```

```
<Button
    android:id="@+id/button02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="13dp"
    android:text="Wypisz listę studentów"
    app:layout_constraintEnd_toEndOf="@+id/button01"
    app:layout_constraintStart_toStartOf="@+id/button01"
    app:layout_constraintTop_toBottomOf="@+id/button01" />

 <TextView
    android:id="@+id/textView3"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_margin="20dp"
    android:text="TextView"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button02"
    app:layout_constraintBottom_toBottomOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

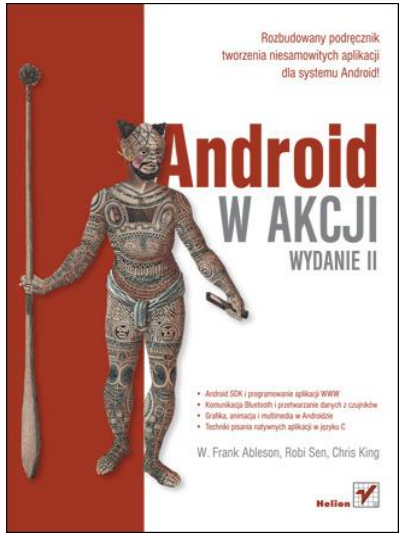
Przykład

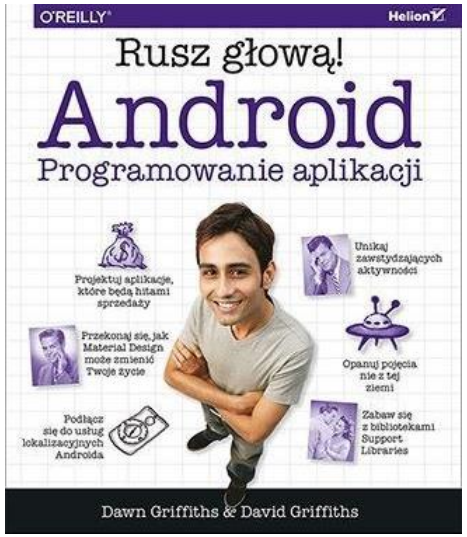
```
public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding binding;
    studentDB studenci;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        studenci = new studentDB(this);

        // Przykład wstawienia rekordu
        //  studenci.insertStudent("John Doe", 21);
        // przykład usuania całej zawartości bazy
        // studenci.deleteAllStudents();

        binding.button01.setOnClickListener(v -> {
            // Przykład wstawienia rekordu
            String name = binding.editText01.getText().toString();
            int sem = Integer.parseInt(binding.editText02.getText().toString());
            studenci.insertStudent(name, sem);
        });

        binding.button02.setOnClickListener(v -> {
            binding.textView3.setText("");
            // Przykład odczytu danych
            Cursor cursor = studenci.getAllStudents();
            while (cursor.moveToNext()) {
                String name = cursor.getString(1);
                int age = cursor.getInt(2);
                binding.textView3.append(name + " " + age + "\n");
            }
        });
    }
}
```

Literatura



https://developer.android.com





https://javastart.pl/baza-wiedzy/android/

https://forum.android.com.pl