Uniwersytet Wrocławski

Rafał Lewandków

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 6

- ViewBinding
- SharedPreferences
- SQLite

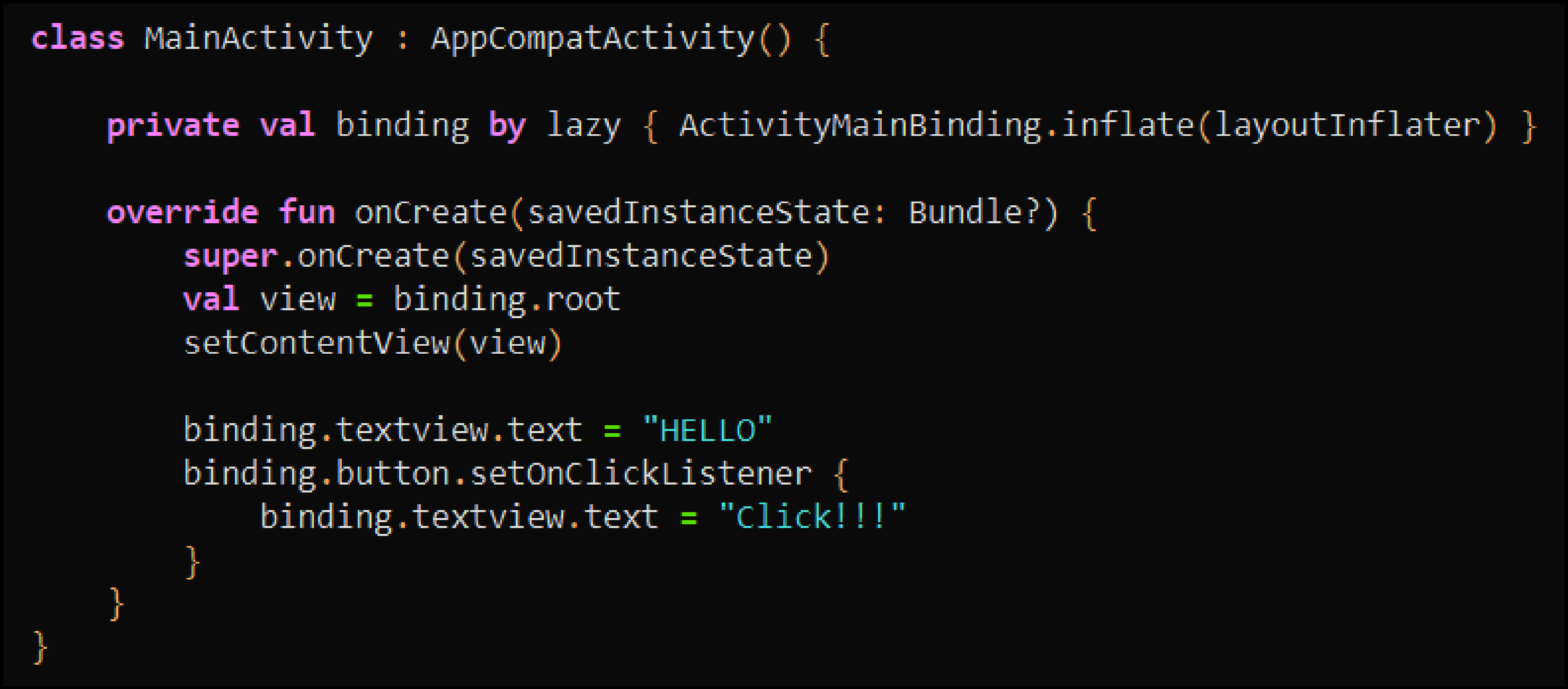
ViewBinding

```
android {
    ...
    buildFeatures {
        viewBinding = true
    }
}
```

```
private val binding by lazy { ActivityMainBinding.inflate(layoutInflater) }
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val view = binding.root
    setContentView(view)
}
```

ViewBinding



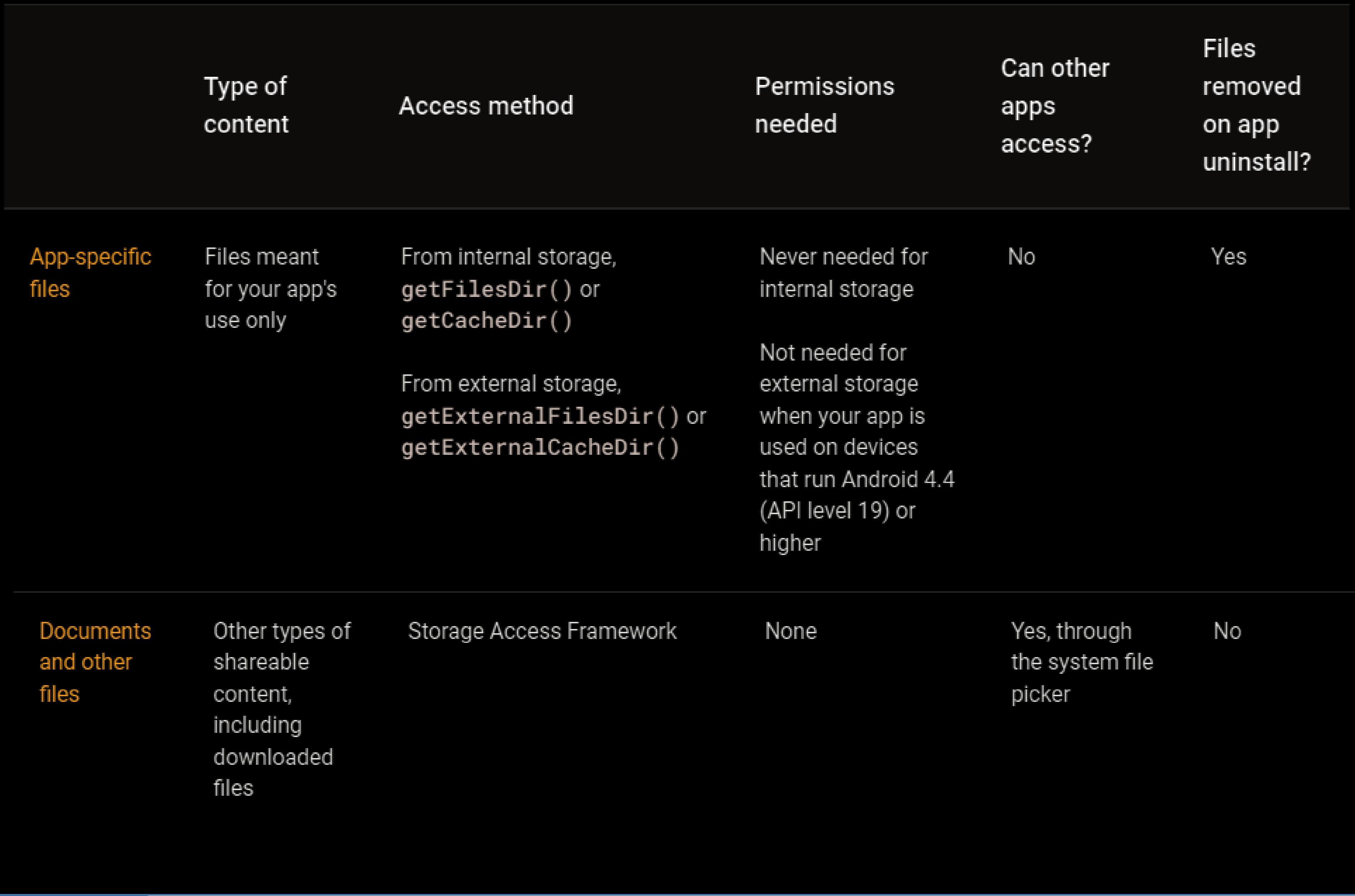
```kotlin
class MainActivity : AppCompatActivity() {

    private val binding by lazy { ActivityMainBinding.inflate(layoutInflater) }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val view = binding.root
        setContentView(view)

        binding.textview.text = "HELLO"
        binding.button.setOnClickListener {
            binding.textview.text = "Click!!!"
        }
    }
}
```

Przechowywanie danych

- Shared Preferences — Prywatne dane, pary klucz-wartość
- Internal Storage — Prywatne dane w pamięci urządzenia
- External Storage — Publiczne dane w pamięci urządzenia lub zewnętrznym dysku
- SQLite Databases — Prywatna baza danych
- ROOM — Prywatna baza danych

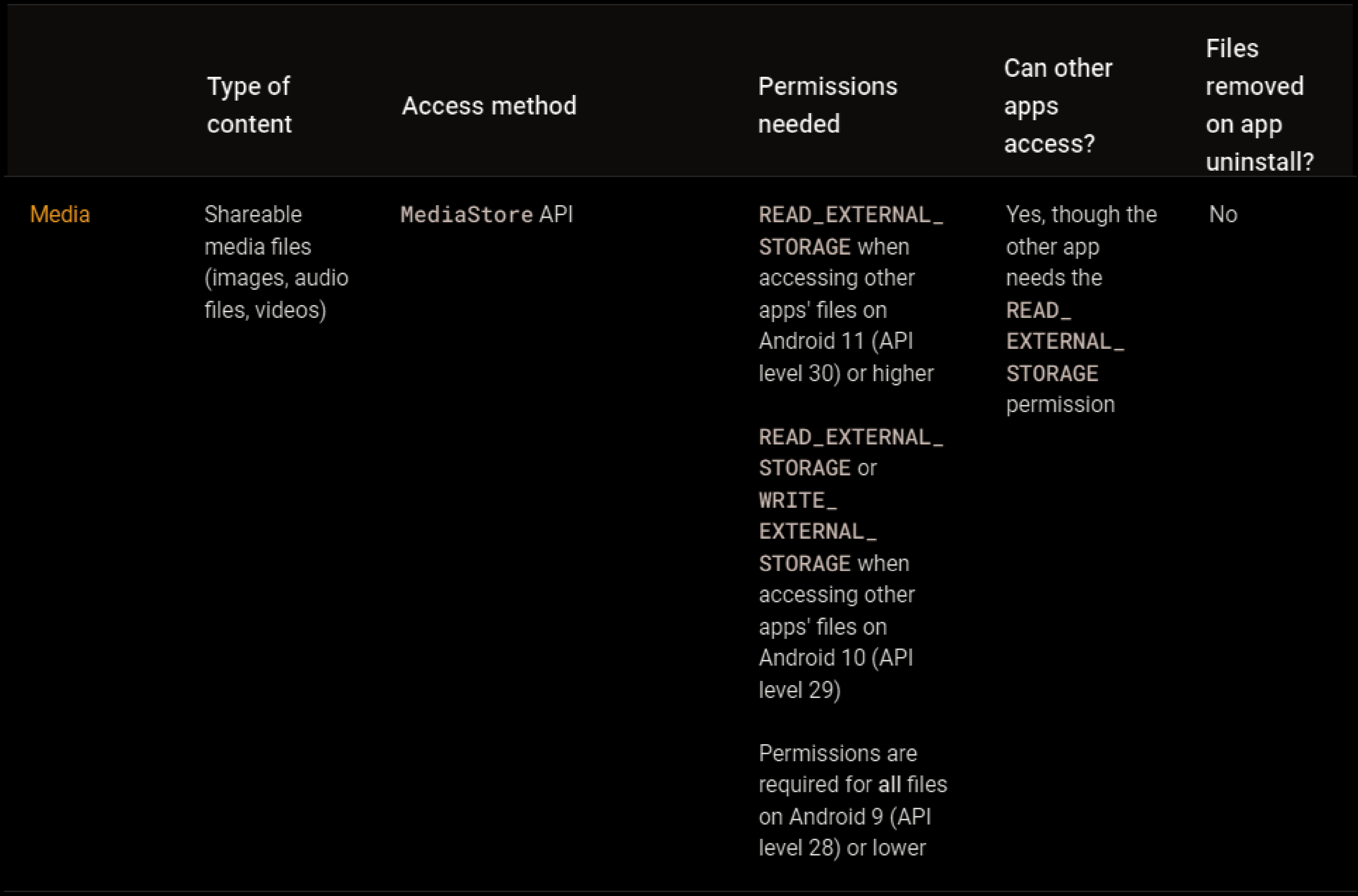
Uniwersytet Wrocławski

Przechowywanie danych



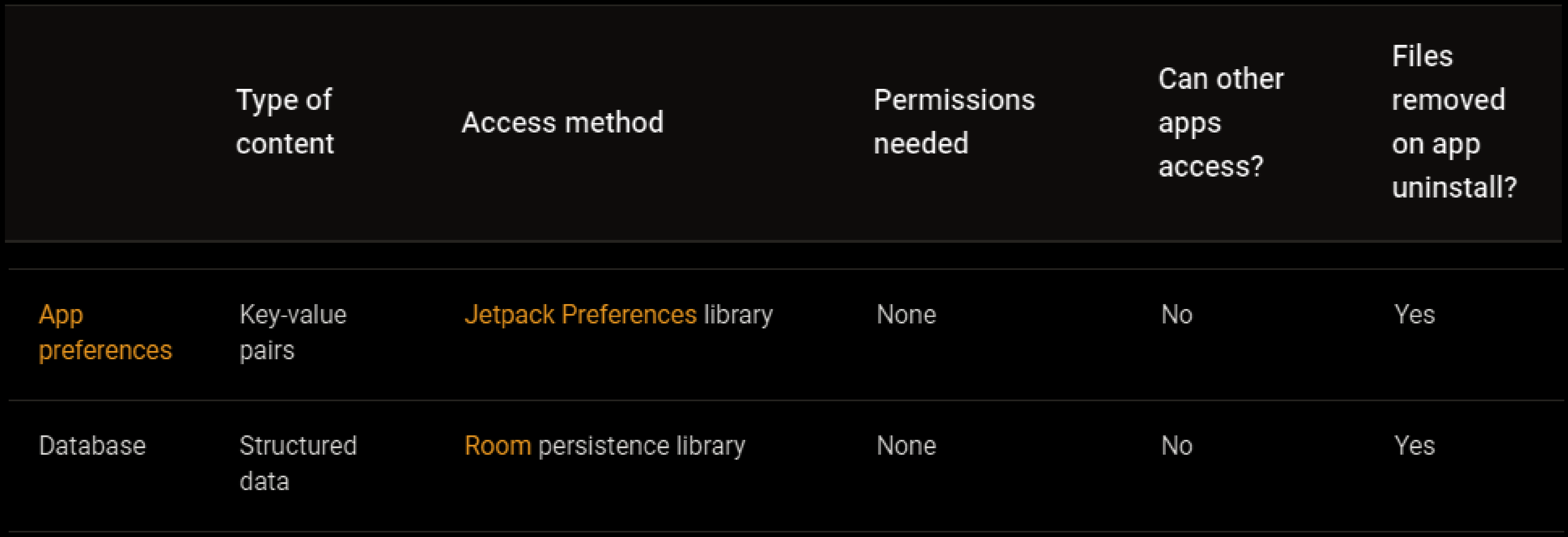
	Type of content	Access method	Permissions needed	Can other apps access?	Files removed on app uninstall?
App-specific files	Files meant for your app's use only	From internal storage, `getFilesDir()` or `getCacheDir()` From external storage, `getExternalFilesDir()` or `getExternalCacheDir()`	Never needed for internal storage Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher	No	Yes
Documents and other files	Other types of shareable content, including downloaded files	Storage Access Framework	None	Yes, through the system file picker	No

Przechowywanie danych



	Type of content	Access method	Permissions needed	Can other apps access?	Files removed on app uninstall?
Media	Shareable media files (images, audio files, videos)	`MediaStore` API	READ_EXTERNAL_STORAGE when accessing other apps' files on Android 11 (API level 30) or higher READ_EXTERNAL_STORAGE or WRITE_EXTERNAL_STORAGE when accessing other apps' files on Android 10 (API level 29) Permissions are required for **all** files on Android 9 (API level 28) or lower	Yes, though the other app needs the READ_EXTERNAL_STORAGE permission	No

Przechowywanie danych



	Type of content	Access method	Permissions needed	Can other apps access?	Files removed on app uninstall?
App preferences	Key-value pairs	Jetpack Preferences library	None	No	Yes
Database	Structured data	Room persistence library	None	No	Yes

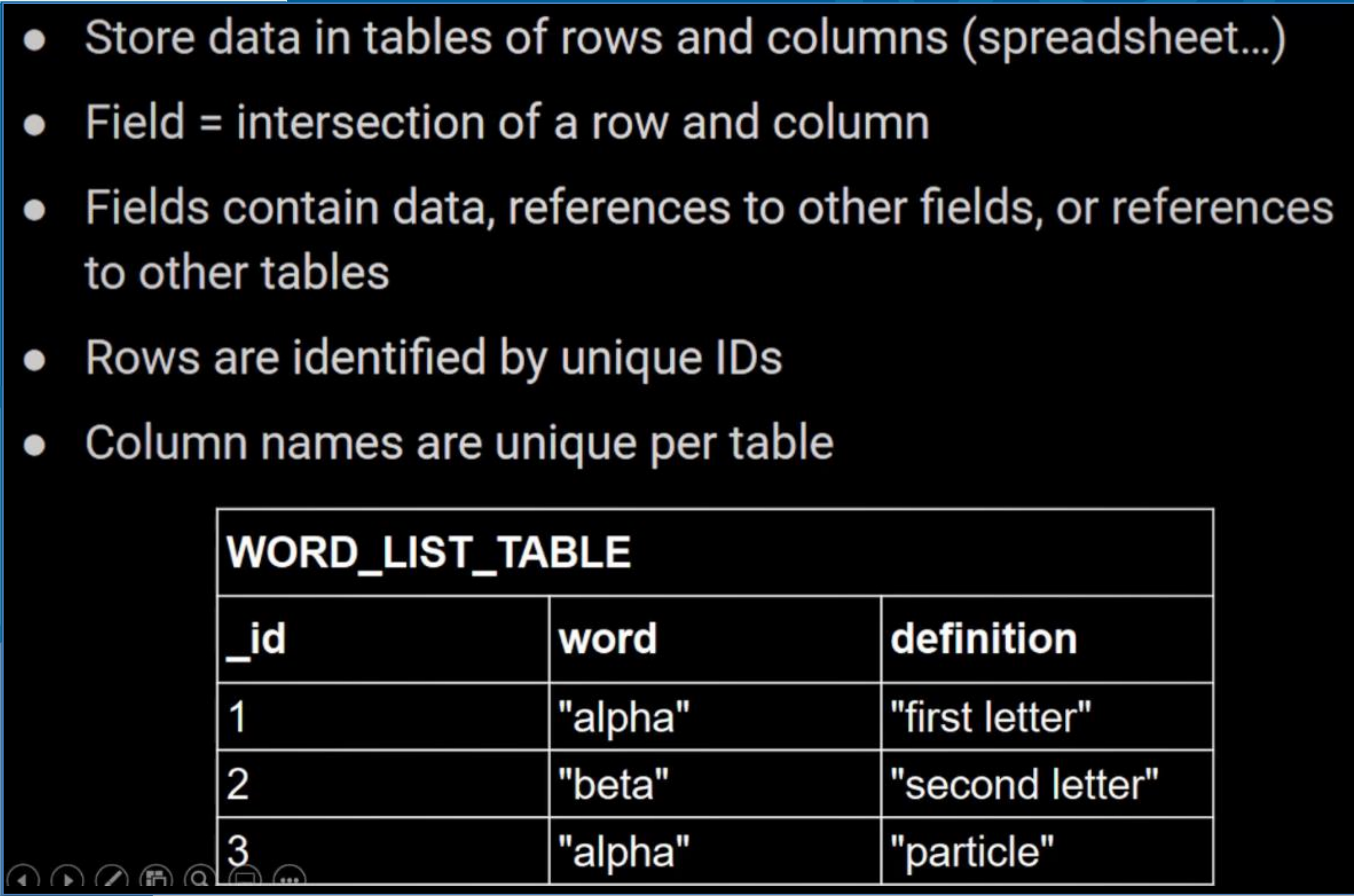
SharedPreferences

```kotlin
override fun onPause() {
    super.onPause()
    val sharedPref = getSharedPreferences("fileName", MODE_PRIVATE)
    val edit = sharedPref.edit()
    edit.apply {
        putInt("counter", binding.counter1TextView.text.toString().toInt())
        apply()
    }
}
```

```kotlin
override fun onResume() {
    super.onResume()
    val sharedPref = getSharedPreferences("fileName", MODE_PRIVATE)
    binding.counter1TextView.text = sharedPref.getInt("counter", 0).toString()
}
```

- `MODE_APPEND` - pozwala dopisywać kolejne elementy bez nadpisywania
- `MODE_PRIVATE` - najczęściej wykorzystywany, dostęp do pliku tylko z poziomu aplikacji
- `MODE_WORLD_READABLE` - zezwala innym aplikacjom na odczyt
- `MODE_WORLD_WRITABVLE` - zezwala innym aplikacjom na zapis

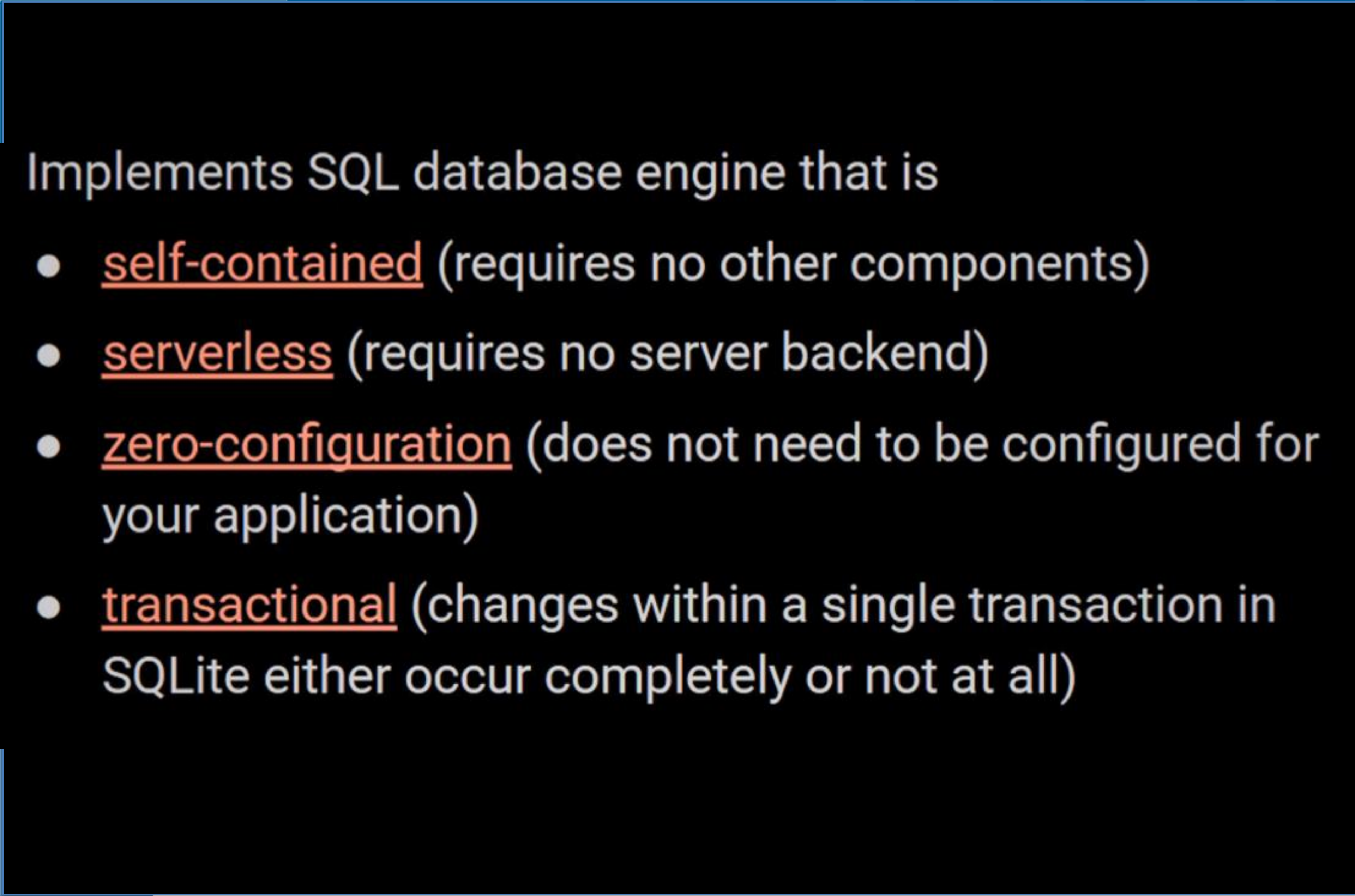
SQLite



- Store data in tables of rows and columns (spreadsheet…)
- Field = intersection of a row and column
- Fields contain data, references to other fields, or references to other tables
- Rows are identified by unique IDs
- Column names are unique per table

WORD_LIST_TABLE		
_id	**word**	**definition**
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

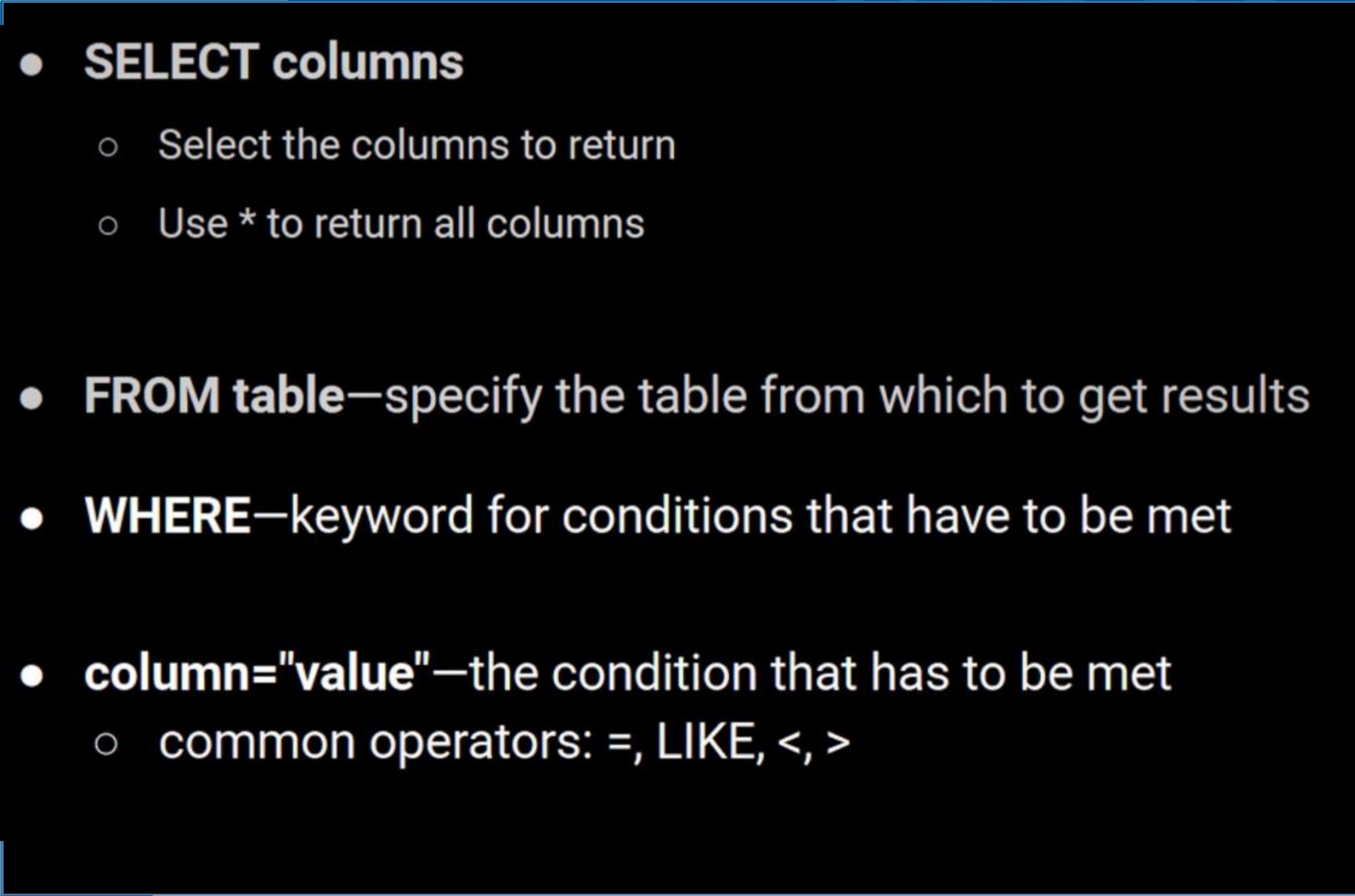
SQLite



Implements SQL database engine that is

- self-contained (requires no other components)
- serverless (requires no server backend)
- zero-configuration (does not need to be configured for your application)
- transactional (changes within a single transaction in SQLite either occur completely or not at all)

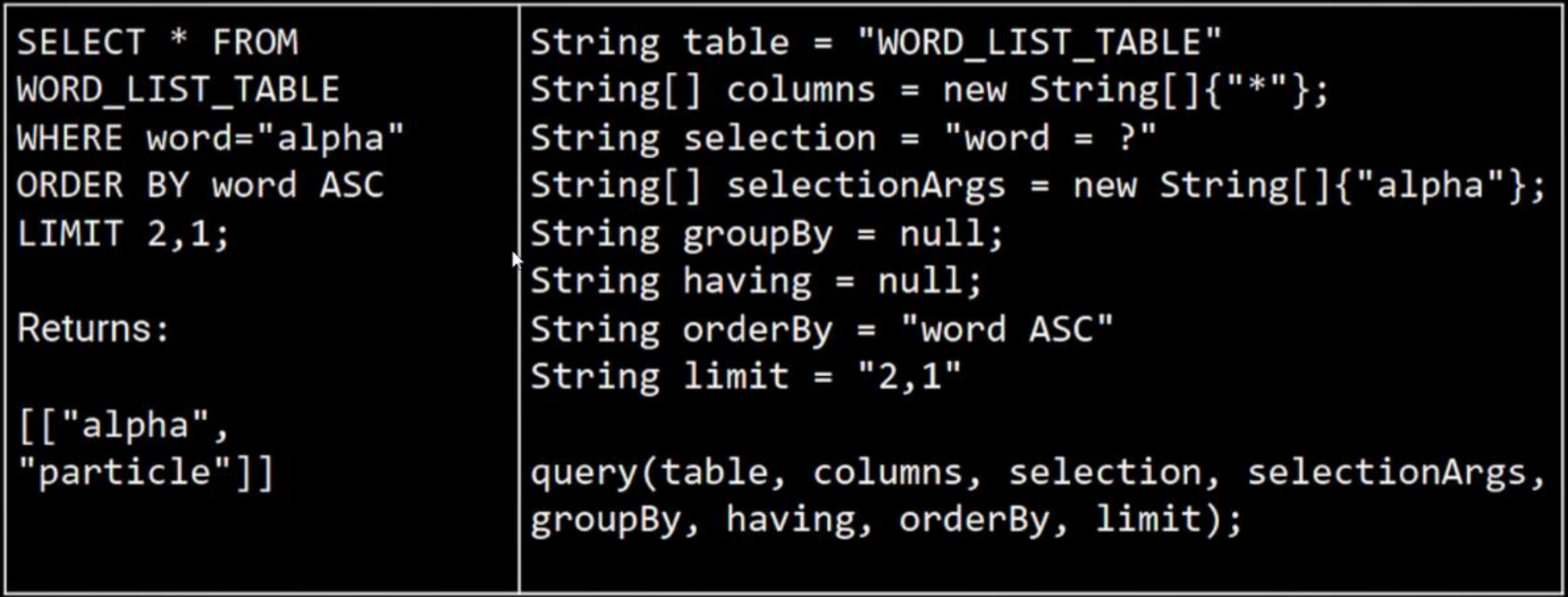
SQLite



- **SELECT columns**
  - Select the columns to return
  - Use * to return all columns
- **FROM table**—specify the table from which to get results
- **WHERE**—keyword for conditions that have to be met
- **column="value"**—the condition that has to be met
  - common operators: =, LIKE, <, >

Uniwersytet Wrocławski

SQLite



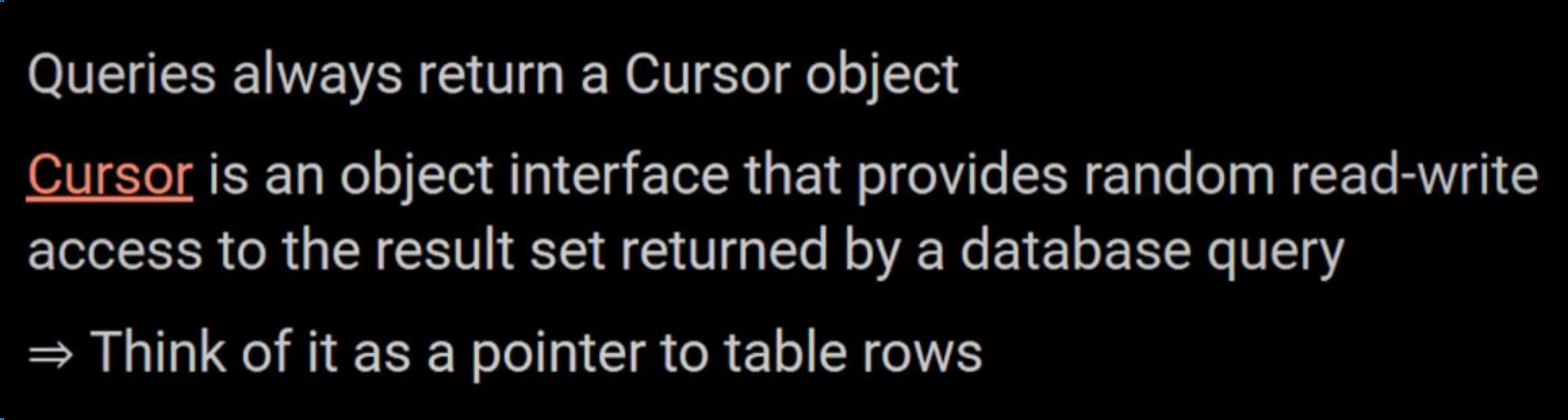
```
SELECT * FROM
WORD_LIST_TABLE
WHERE word="alpha"
ORDER BY word ASC
LIMIT 2,1;
```

Returns:

```
[["alpha",
"particle"]]
```

```
String table = "WORD_LIST_TABLE"
String[] columns = new String[]{"*"};
String selection = "word = ?"
String[] selectionArgs = new String[]{"alpha"};
String groupBy = null;
String having = null;
String orderBy = "word ASC"
String limit = "2,1"

query(table, columns, selection, selectionArgs,
groupBy, having, orderBy, limit);
```

SQLite



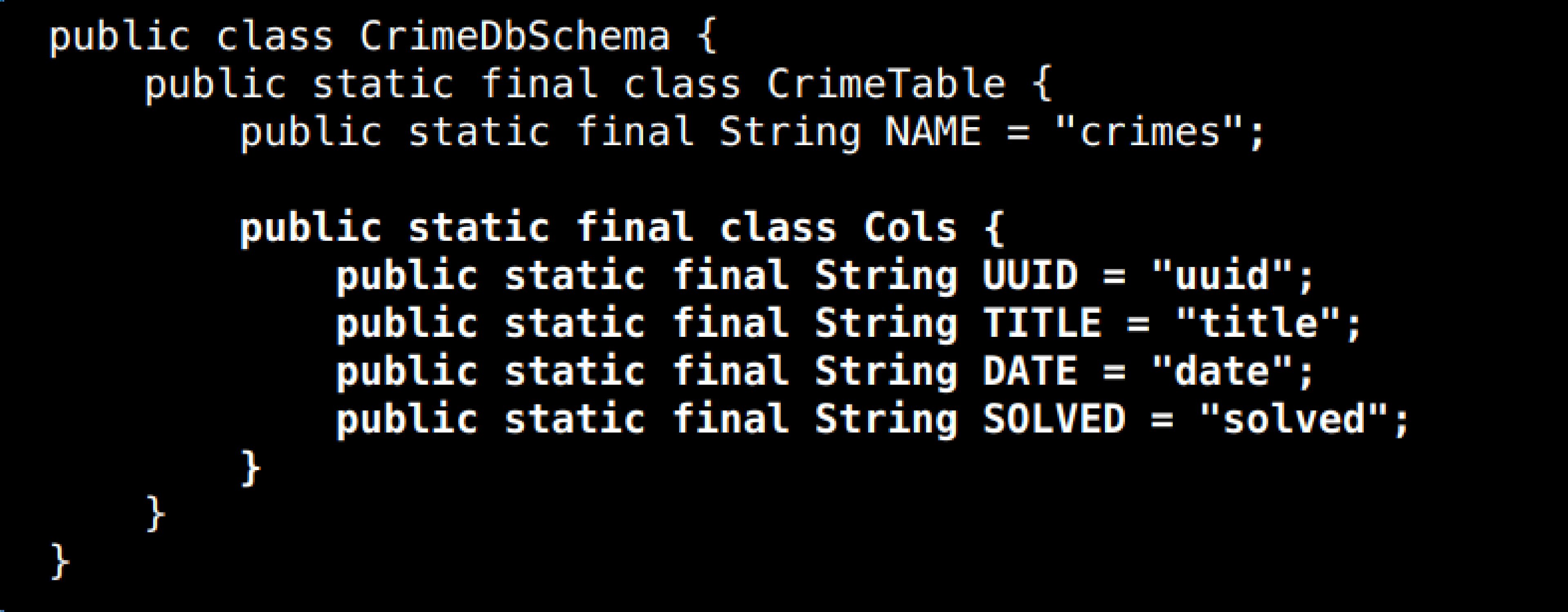
Queries always return a Cursor object

Cursor is an object interface that provides random read-write access to the result set returned by a database query

⇒ Think of it as a pointer to table rows

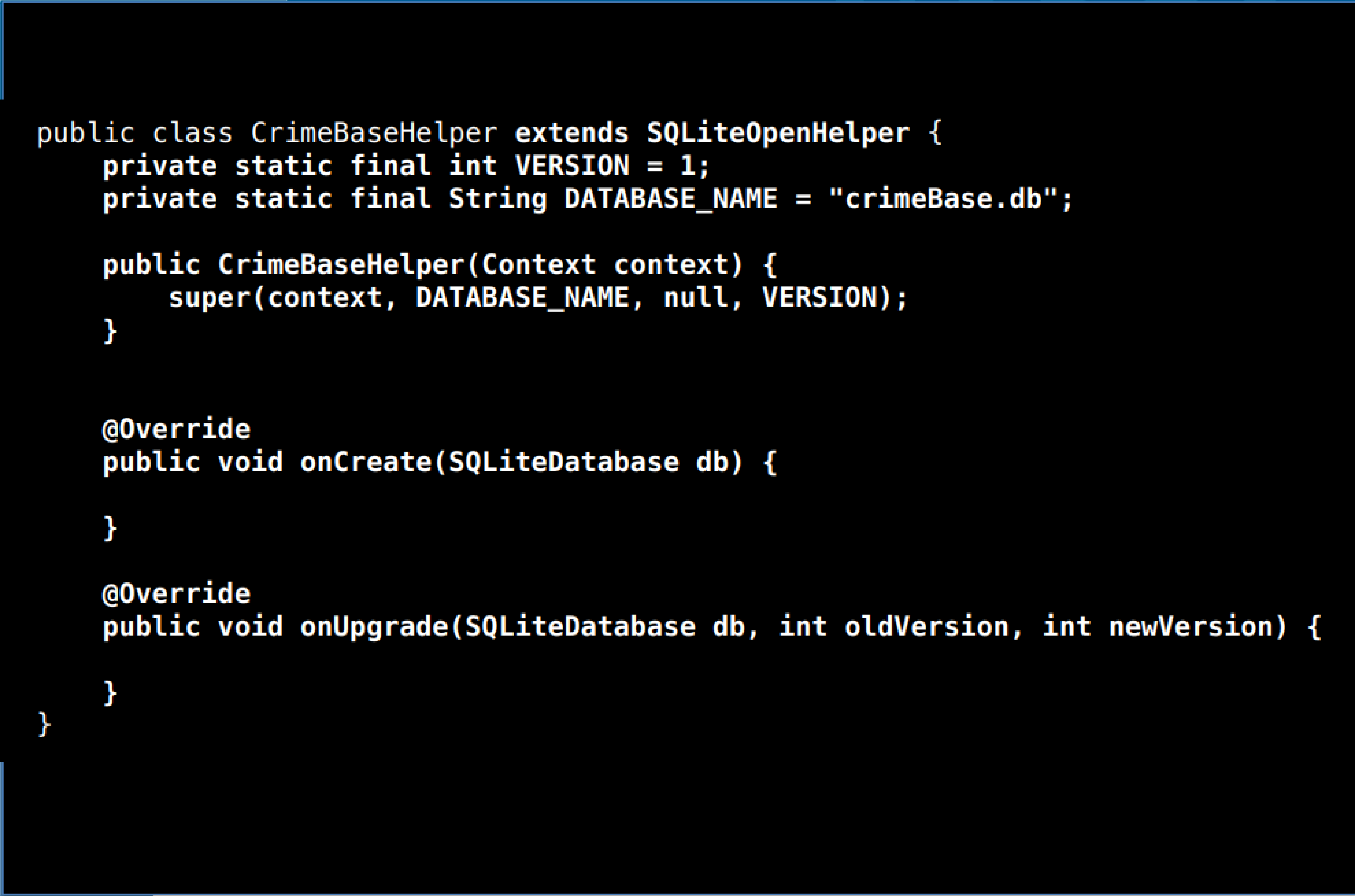
Uniwersytet Wrocławski

SQLite



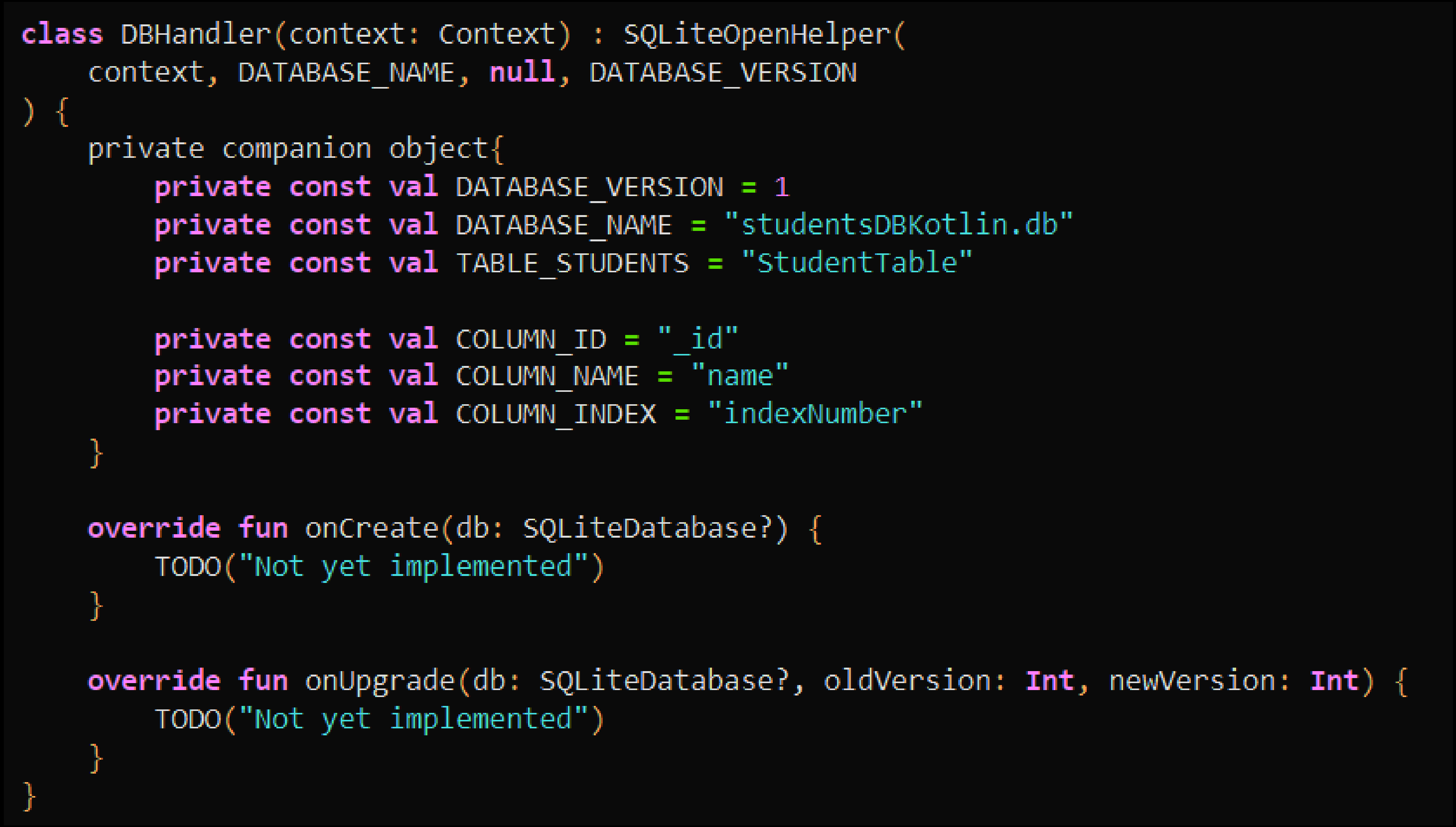
```java
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
        }
    }
}
```

SQLite



```
public class CrimeBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "crimeBase.db";

    public CrimeBaseHelper(Context context) {
        super(context, DATABASE_NAME, null, VERSION);
    }


    @Override
    public void onCreate(SQLiteDatabase db) {

    }

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

    }
}
```

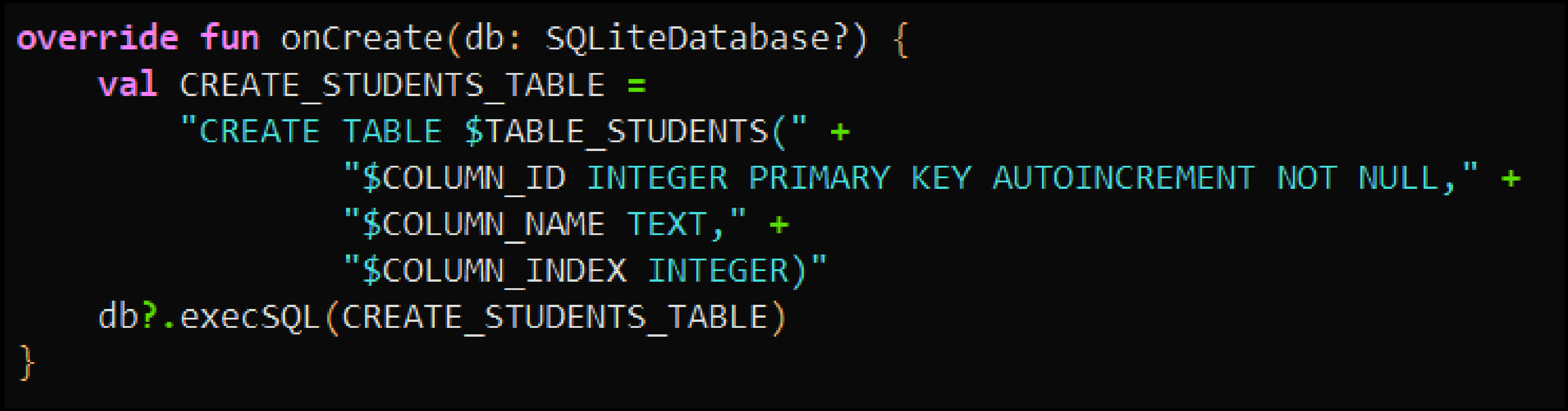
SQLite



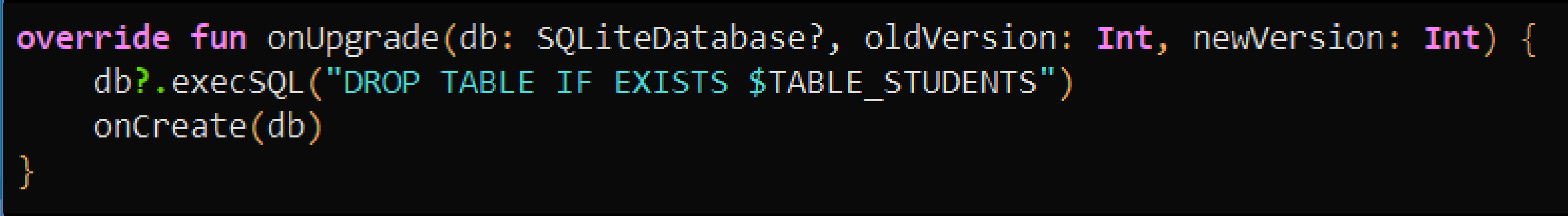
```
class DBHandler(context: Context) : SQLiteOpenHelper(
    context, DATABASE_NAME, null, DATABASE_VERSION
) {
    private companion object{
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "studentsDBKotlin.db"
        private const val TABLE_STUDENTS = "StudentTable"

        private const val COLUMN_ID = "_id"
        private const val COLUMN_NAME = "name"
        private const val COLUMN_INDEX = "indexNumber"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        TODO("Not yet implemented")
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        TODO("Not yet implemented")
    }
}
```

Uniwersytet Wrocławski

SQLite



```kotlin
override fun onCreate(db: SQLiteDatabase?) {
    val CREATE_STUDENTS_TABLE =
        "CREATE TABLE $TABLE_STUDENTS(" +
                "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL," +
                "$COLUMN_NAME TEXT," +
                "$COLUMN_INDEX INTEGER)"
    db?.execSQL(CREATE_STUDENTS_TABLE)
}
```



```kotlin
override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
    db?.execSQL("DROP TABLE IF EXISTS $TABLE_STUDENTS")
    onCreate(db)
}
```

Uniwersytet Wrocławski

SQLite

```
fun addStudent(student: Student){
    val db = this.writableDatabase

    val contentValues = ContentValues()
    contentValues.put(COLUMN_NAME, student.name)
    contentValues.put(COLUMN_INDEX, student.index)

    db.insert(TABLE_STUDENTS, null, contentValues)
    db.close()
}
```

Uniwersytet Wrocławski

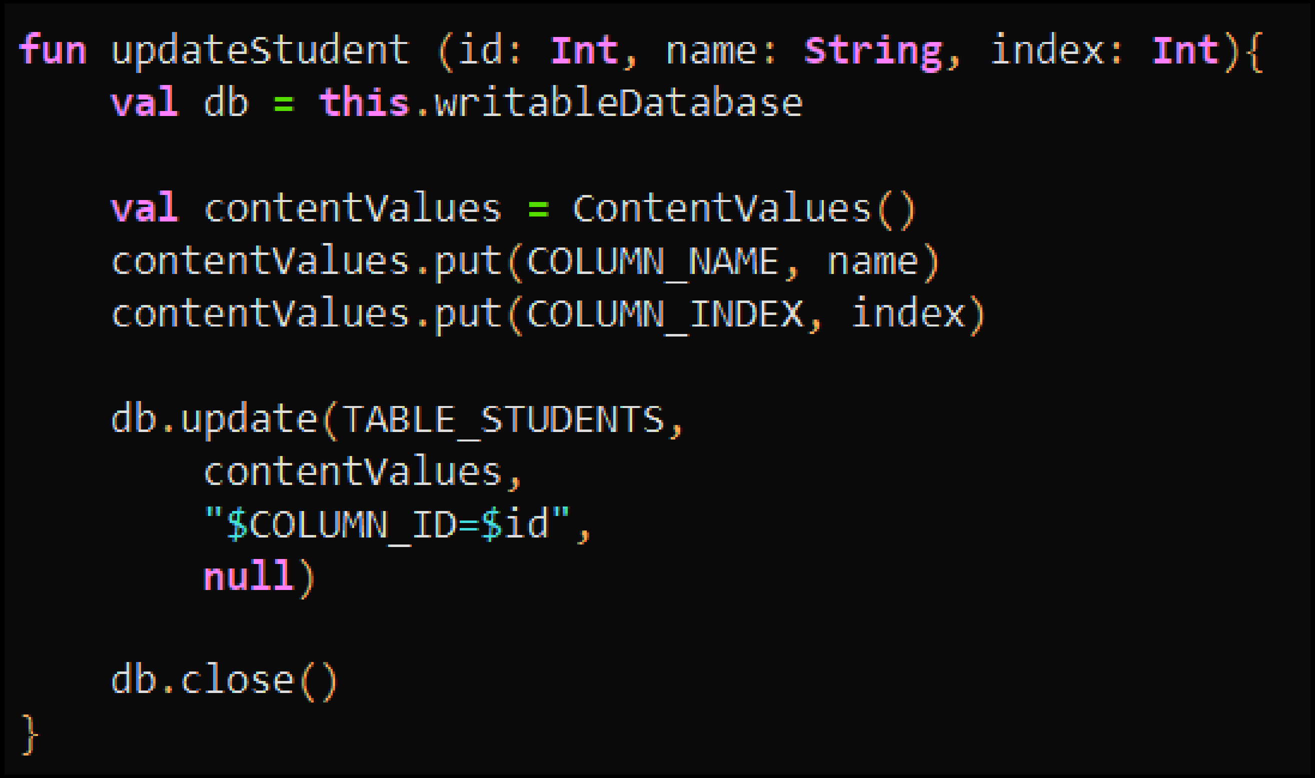
SQLite

```
fun addStudent(student: Student){
    val db = this.writableDatabase

    val contentValues = ContentValues()
    contentValues.put(COLUMN_NAME, student.name)
    contentValues.put(COLUMN_INDEX, student.index)

    db.insert(TABLE_STUDENTS, null, contentValues)
    db.close()
}
```

```
db.delete(
    TABLE_STUDENTS,
    "$COLUMN_ID=${student.id}",
    null)
```

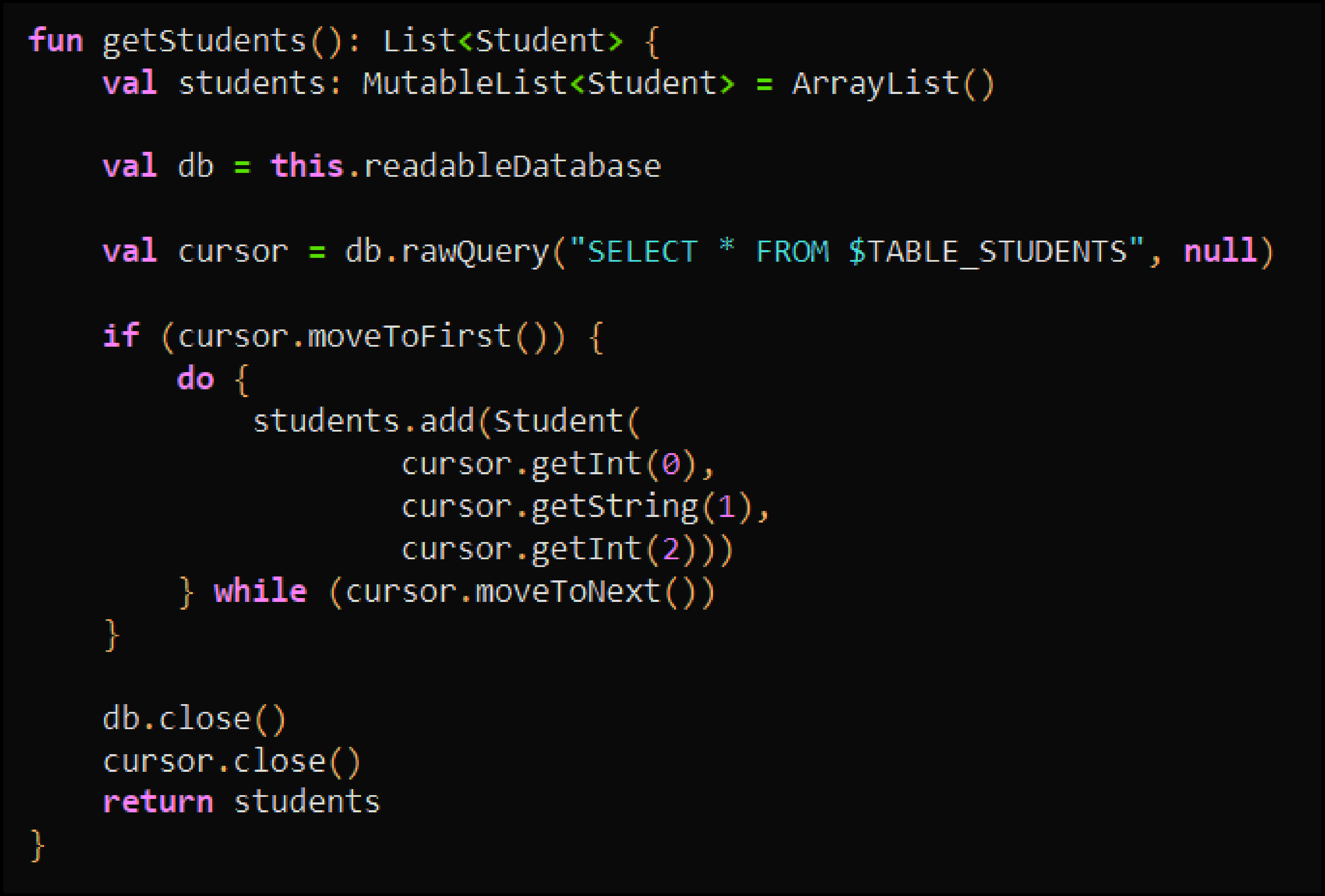
Uniwersytet Wrocławski

SQLite



```kotlin
fun updateStudent (id: Int, name: String, index: Int){
    val db = this.writableDatabase

    val contentValues = ContentValues()
    contentValues.put(COLUMN_NAME, name)
    contentValues.put(COLUMN_INDEX, index)

    db.update(TABLE_STUDENTS,
        contentValues,
        "$COLUMN_ID=$id",
        null)

    db.close()
}
```

SQLite



```kotlin
fun getStudents(): List<Student> {
    val students: MutableList<Student> = ArrayList()

    val db = this.readableDatabase

    val cursor = db.rawQuery("SELECT * FROM $TABLE_STUDENTS", null)

    if (cursor.moveToFirst()) {
        do {
            students.add(Student(
                    cursor.getInt(0),
                    cursor.getString(1),
                    cursor.getInt(2)))
        } while (cursor.moveToNext())
    }

    db.close()
    cursor.close()
    return students
}
```