Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЗВІТ

з лабораторної роботи №3
з дисципліни «Сучасні мобільні операційні системи»

Тема: «Зберігання інформації в базі даних SQLite»

Варіант №4

Виконав:
студент 1 курсу, групи ІМ-51мн
Ковальов Олександр

Перевірив:
асистент, Нестерук Андрій Олександрович

Дата здачі: 25.02.2026

КИЇВ – 2026

Мета роботи. Вивчити роботу Android-програми з базою даних.

Завдання 1. Необхідно створити додаток, що взаємодіє з базою даних. Перша активність повинна містити три кнопки. При натисканні на першу кнопку повинна відкриватись нова активність, що виводить інформацію з таблиці «Одногрупники» в зручному для сприйняття форматі.

При запуску програми необхідно:

1. Створити базу даних, якщо її не існує.
2. Створити таблицю «Одногрупники», що містить поля «ID», «ПІБ» та «Час додавання запису».
3. Видалити усі записи з бази даних, а потім внести 5 записів про одногрупників. При натисканні на другу кнопку необхідно внести ще один запис в таблицю. При натисканні на третю кнопку необхідно замінити ПІБ в останньому записі на «Петренко Петро Петрович».

Завдання 2. Створити нову прикладну програму на основі додатку, створенного в завданні 1. Перевизначити функцію `onUpgrade`. При зміні версії БД необхідно вилучити таблицю «Одногрупники», створити таблицю з тією ж назвою, яка містить наступні поля «ID», «Прізвище», «Ім'я», «По-батькові» та «Час додавання запису». Змінити версію бази даних.

Хід роботи.

Спочатку, був створений клас `DatabaseHelper`, який спадкований від `SQLiteOpenHelper`. По суті, він відповідає за створення бази даних, ця логіка інкапсульована в базовому класі. Розробнику не треба думати про це, тому такий підхід є хорошою практикою.

Потрібні поля для бази даних (назва, версія, назви колонок) – статичні. Kotlin має для таких випадків дещо покращений синтаксис – `companion objects`:

```
companion object {
    1 Usage
    private const val DATABASE_NAME = "university.db"
    1 Usage
    private const val DATABASE_VERSION = 1

    // Table and columns names
    7 Usages
    private const val TABLE_NAME = "classmates"
    4 Usages
    private const val COLUMN_ID = "id"
    4 Usages
    private const val COLUMN_NAME = "name"
    3 Usages
    private const val COLUMN_TIME = "time_added"
}
```

Також, були перевантажені методи `onCreate` та `onUpgrade`. Перший застосовується, коли відбувається доступ до неіснуючої бази даних – вона автоматично створюється, і виконуються додаткові дії з цього методу. Другий метод використовується при змінах схеми бази даних. Відповідно, при створенні БД – створюється таблиця зі студентами за допомогою SQL-запиту. Якщо ж змінюється схема – таблиця перестворюється.

```
1 Usage
override fun onCreate(db: SQLiteDatabase) {
    // Create table query
    val createTableQuery = ("CREATE TABLE $TABLE_NAME ("
            + "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "$COLUMN_NAME TEXT, "
            + "$COLUMN_TIME TEXT)")
    db.execSQL( sql = createTableQuery)
}

override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    // Drop older table if exists and create fresh
    db.execSQL( sql = "DROP TABLE IF EXISTS $TABLE_NAME")
    onCreate(db)
}
```

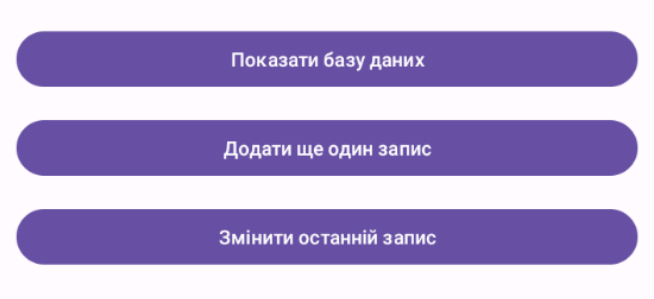
Інші методи, можна сказати, виконують функції патерну «Репозиторій» – видалення, додавання, оновлення даних. Варто зазначити, що об’єкт бази даних отримується через поле `this.writableDatabase`. Це властивість (Property) – якщо здійснити спробу отримати дані, автоматично викликається метод гетер, який створює базу даних, якщо її не існує.

```
// Delete all records from the table
1 Usage
fun clearDatabase() {
    val db = this.writableDatabase
    db.execSQL( sql = "DELETE FROM $TABLE_NAME")
    db.close()
}

// Insert a new student record
2 Usages
fun addStudent(name: String, time: String) {
    val db = this.writableDatabase
    val values = ContentValues()
    values.put(COLUMN_NAME, name)
    values.put(COLUMN_TIME, time)
    db.insert(TABLE_NAME, nullColumnHack = null, values)
    db.close()
}

// Update the last added record's name
1 Usage
fun updateLastStudent(newName: String) {
    val db = this.writableDatabase
    // Update where ID is the maximum ID in the table
    val updateQuery =
        "UPDATE $TABLE_NAME SET $COLUMN_NAME = '$newName' WHERE $COLUMN_ID = (SELECT MAX($COLUMN_ID) FROM $TABLE_NAME)"
    db.execSQL( sql = updateQuery)
    db.close()
}
```

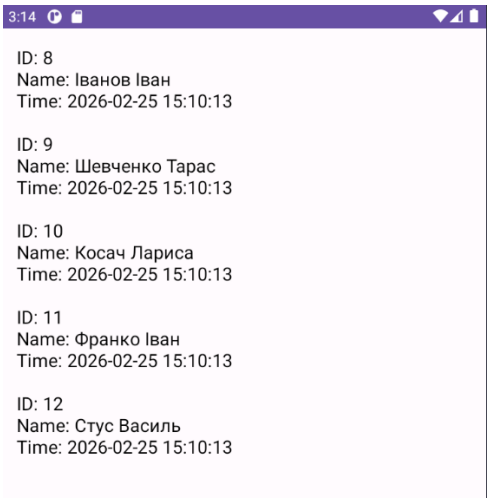
Також, присутній метод для отримання всіх записів у зручному текстовому форматі. Компонування тексту відбувається за допомогою `StringBuilder`.

```
// Retrieve all students as a formatted string
1 Usage
fun getAllStudents(): String {
    val db = this.readableDatabase
    val cursor = db.rawQuery( sql = "SELECT * FROM $TABLE_NAME", selectionArgs = null)
    val stringBuilder = StringBuilder()

    if (cursor.moveToFirst()) {
        do {
            val id = cursor.getInt( columnIndex = cursor.getColumnIndexOrThrow( columnName = COLUMN_ID))
            val name = cursor.getString( columnIndex = cursor.getColumnIndexOrThrow( columnName = COLUMN_NAME))
            val time = cursor.getString( columnIndex = cursor.getColumnIndexOrThrow( columnName = COLUMN_TIME))
            stringBuilder.append("ID: $id\nName: $name\nTime: $time\n\n")
        } while (cursor.moveToNext())
    }
    cursor.close()
    db.close()

    return stringBuilder.toString()
}
```

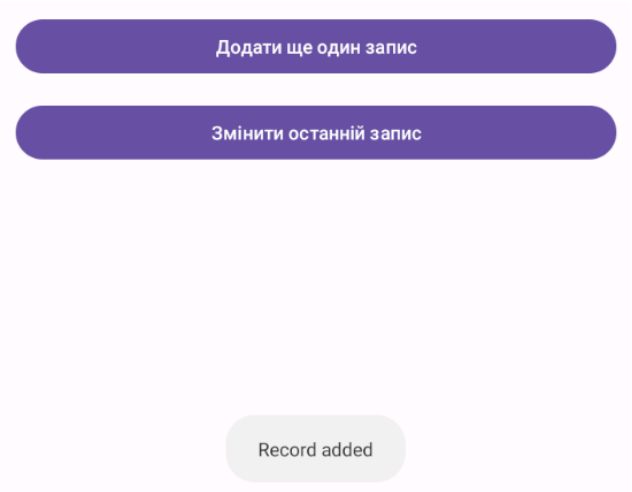
Відповідно, вигляд головної Activity містить три кнопки – показати базу даних, додати запис «Новий студент», та змінити ім’я студента з останнього запису на «Петренко Петро Петрович»:



Якщо натиснути першу кнопку, відкривається додаткова активність `DisplayActivity`, в якій виводиться список студентів. Головний контейнер – `ScrollView`:

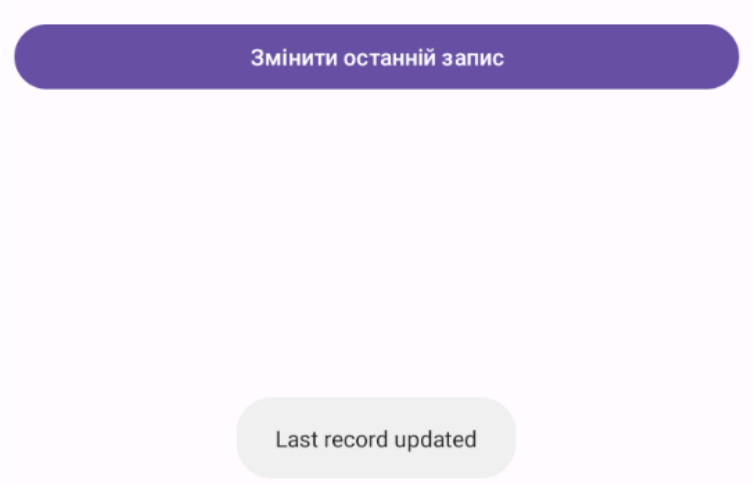


Якщо додати новий запис, отримуємо сповіщення (`Toast`):

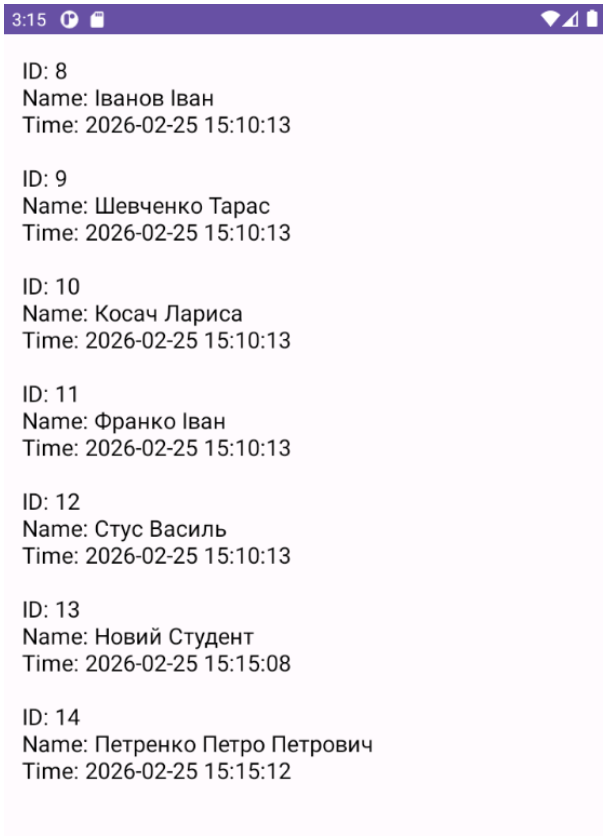


Відповідно, були додані нові записи з міткою – поточним часом:

При зміні останнього запису, користувач теж отримує сповіщення:



І, відповідно, останній запис буде зміненим.



Перше завдання виконане. Для виконання другого завдання, треба зімітувати процес «Міграції». Спочатку, була змінена версія бази даних в константі та додані нові ідентифікатори колонок:

```
companion object {
    1 Usage
    private const val DATABASE_NAME = "university.db"

    // Changed version to 2 to trigger onUpgrade method
    1 Usage
    private const val DATABASE_VERSION = 2

    // Table and new columns names
    7 Usages
    private const val TABLE_NAME = "classmates"
    4 Usages
    private const val COLUMN_ID = "id"
    4 Usages
    private const val COLUMN_LAST_NAME = "last_name"
    4 Usages
    private const val COLUMN_FIRST_NAME = "first_name"
    4 Usages
    private const val COLUMN_PATRONYMIC = "patronymic"
    3 Usages
    private const val COLUMN_TIME = "time_added"
}
```

Був перевизначений метод `onCreate` для створення розширеної таблиці:

```
1 Usage
override fun onCreate(db: SQLiteDatabase) {
    // Create table query with new separated name fields
    val createTableQuery = ("CREATE TABLE $TABLE_NAME ("
            + "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "$COLUMN_LAST_NAME TEXT, "
            + "$COLUMN_FIRST_NAME TEXT, "
            + "$COLUMN_PATRONYMIC TEXT, "
            + "$COLUMN_TIME TEXT)")
    db.execSQL( sql = createTableQuery)
}

override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    // Drop older table if exists and create fresh with new structure
    db.execSQL( sql = "DROP TABLE IF EXISTS $TABLE_NAME")
    onCreate(db)
}
```

Результат:

```
ID: 10
Прізвище: Стус
Ім'я: Василь
По-батькові: Семенович
Час: 2026-02-25 16:11:11

ID: 11
Прізвище: Петренко
Ім'я: Петро
По-батькові: Петрович
Час: 2026-02-25 16:11:23

ID: 12
Прізвище: Новенко
Ім'я: Степан
По-батькові: Іванович
Час: 2026-02-25 16:13:08
```

Висновок. Під час виконання лабораторної роботи було практично закріплено принципи роботи з локальними базами даних SQLite в операційній системі Android. За допомогою успадкування від класу `SQLiteOpenHelper` було успішно реалізовано механізм автоматичного створення та управління життєвим циклом бази даних. На першому етапі розроблено архітектуру програми для виконання базових CRUD-операцій: навмисне очищення таблиці, додавання нових студентів із використанням класу `ContentValues`, зчитування та форматування збережених записів за допомогою об’єкта `Cursor`, а також точкове оновлення конкретного рядка за заданою умовою. На другому етапі роботи було досліджено життєво важливий для реальних додатків механізм міграції баз даних. Шляхом підвищення константи версії сховища та перевизначення методу `onUpgrade` вдалося безпечно змінити структуру таблиці, розділивши загальне поле імені на атомарні складові (прізвище, ім’я, по-батькові). Виконання цих завдань дозволило глибше зрозуміти процеси персистентного зберігання даних, що є невіддільною частиною розробки надійних мобільних застосунків.

Контрольні запитання.

1. **Як називається базовий клас для роботи з базою даних SQLite в Android?**

   Базовим класом для управління локальною базою даних в операційній системі Android є абстрактний клас `SQLiteOpenHelper`. Він інкапсулює в собі логіку створення, відкриття та управління життєвим циклом бази даних, знімаючи з розробника необхідність вручну перевіряти наявність файлу БД на пристрої.

2. **Які методи обов'язкові для перевизначення при роботі з базою даних SQLite?**

   При створенні власного класу-помічника, який успадковується від `SQLiteOpenHelper`, розробник зобов'язаний перевизначити два абстрактні методи. Перший метод — це `onCreate`, який автоматично викликається системою при першому створенні бази даних і зазвичай містить SQL-запити для генерації таблиць. Другий обов'язковий метод — `onUpgrade`, який спрацьовує при збільшенні номера версії бази даних і відповідає за безпечну зміну її структури (міграцію), наприклад, при додаванні або видаленні колонок.

3. **Для чого використовується клас `ContentValues`?**

   Клас `ContentValues` використовується для зручного та безпечного формування набору даних перед їхнім записом у таблицю. Він працює за принципом асоціативного масиву (словника), зберігаючи інформацію у вигляді пар «ключ-значення», де ключем завжди виступає назва конкретного стовпця таблиці, а значенням — дані відповідного типу. Використання цього класу для операцій вставки та оновлення дозволяє уникнути ручного формування складних SQL-рядків та автоматично екранує спеціальні символи, захищаючи додаток від SQL-ін'єкцій.

4. **Для чого використовується клас `Cursor`?**

   Клас `Cursor` представляє собою системний інтерфейс, який забезпечує навігацію та доступ до набору результатів (рядків), повернутих після виконання SQL-запиту на вибірку (`SELECT`). Він діє як рухомий вказівник, який можна програмно переміщувати по отриманій таблиці результатів за допомогою методів на кшталт `moveToFirst` або `moveToNext`, та зчитувати значення з потрібних колонок поточного рядка за їхніми індексами.

5. **Як реалізуються методи `INSERT`, `QUERY`, `DELETE`, `UPDATE` для вставки, читання, видалення і додавання записів в SQLite?**

   Ці базові операції реалізуються через виклик відповідних методів об'єкта `SQLiteDatabase`. Для вставки записів використовується метод `insert`, який приймає ім'я таблиці та підготовлений об'єкт `ContentValues`. Для читання (вибірки) використовується метод `query` (для побудови запиту через параметри) або `rawQuery` (для виконання прямого SQL-коду), які повертають об'єкт `Cursor`. Для видалення застосовується метод `delete`, в який передається назва таблиці та текстова умова `WHERE` з аргументами. Оновлення існуючих записів здійснюється за допомогою методу `update`, який поєднує в собі об'єкт `ContentValues` із новими даними та умови для пошуку тих рядків, які підлягають модифікації. Також розробник може використовувати метод `execSQL` для виконання будь-яких сирих SQL-команд, які не повертають даних.

Лістинг.

task1/DatabaseHelper.kt

```
package ua.kpi.lab3_1

import android.content.ContentValues
import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

// Helper class for managing SQLite database
class DatabaseHelper(context: Context) :
SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {
    companion object {
        private const val DATABASE_NAME = "university.db"
        private const val DATABASE_VERSION = 1

        // Table and columns names
        private const val TABLE_NAME = "classmates"
        private const val COLUMN_ID = "id"
        private const val COLUMN_NAME = "name"
        private const val COLUMN_TIME = "time_added"
    }

    override fun onCreate(db: SQLiteDatabase) {
        // Create table query
        val createTableQuery = ("CREATE TABLE $TABLE_NAME ("
        + "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, "
        + "$COLUMN_NAME TEXT, "
        + "$COLUMN_TIME TEXT)")
        db.execSQL(createTableQuery)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        // Drop older table if exists and create fresh
        db.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }

    // Delete all records from the table
    fun clearDatabase() {
        val db = this.writableDatabase
        db.execSQL("DELETE FROM $TABLE_NAME")
        db.close()
    }

    // Insert a new student record
    fun addStudent(name: String, time: String) {
        val db = this.writableDatabase
        val values = ContentValues()
        values.put(COLUMN_NAME, name)
        values.put(COLUMN_TIME, time)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }

    // Update the last added record's name
    fun updateLastStudent(newName: String) {
        val db = this.writableDatabase
        // Update where ID is the maximum ID in the table
        val updateQuery =
```

```
        "UPDATE $TABLE_NAME SET $COLUMN_NAME = '$newName' WHERE $COLUMN_ID = (SELECT
        ↪  MAX($COLUMN_ID) FROM $TABLE_NAME)"
        db.execSQL(updateQuery)
        db.close()
    }

    // Retrieve all students as a formatted string
    fun getAllStudents(): String {
        val db = this.readableDatabase
        val cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        val stringBuilder = StringBuilder()

        if (cursor.moveToFirst()) {
            do {
                val id = cursor.getInt(cursor.getColumnIndexOrThrow(COLUMN_ID))
                val name = cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_NAME))
                val time = cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_TIME))
                stringBuilder.append("ID: $id\nName: $name\nTime: $time\n\n")
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()

        return stringBuilder.toString()
    }
}
```

task1/MainActivity.kt

```
package ua.kpi.lab3_1

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale

class MainActivity : AppCompatActivity() {

    private lateinit var dbHelper: DatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        dbHelper = DatabaseHelper(this)

        // Clear database and insert 5 initial records on startup
        dbHelper.clearDatabase()
        insertInitialData()

        val btnShowData = findViewById<Button>(R.id.btnShowData)
        val btnAddRecord = findViewById<Button>(R.id.btnAddRecord)
        val btnUpdateRecord = findViewById<Button>(R.id.btnUpdateRecord)

        // Button 1: Open new activity to display data
        btnShowData.setOnClickListener {
            val intent = Intent(this, DisplayActivity::class.java)
            startActivity(intent)
```

```
        }

        // Button 2: Add one more record
        btnAddRecord.setOnClickListener {
            val currentTime = getCurrentTime()
            dbHelper.addStudent("Новий Студент", currentTime)
            Toast.makeText(this, "Record added", Toast.LENGTH_SHORT).show()
        }

        // Button 3: Update the last record
        btnUpdateRecord.setOnClickListener {
            dbHelper.updateLastStudent("Петренко Петро Петрович")
            Toast.makeText(this, "Last record updated", Toast.LENGTH_SHORT).show()
        }
    }

    // Helper method to insert 5 students
    private fun insertInitialData() {
        val students =
        listOf("Іванов Іван", "Шевченко Тарас", "Косач Лариса", "Франко Іван", "Стус Василь")
        for (student in students) {
            dbHelper.addStudent(student, getCurrentTime())
        }
    }

    // Helper method to get formatted current time
    private fun getCurrentTime(): String {
        val sdf = SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault())
        return sdf.format(Date())
    }
}
```

task1/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
 ↪  xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<Button
android:id="@+id/btnShowData"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Показати базу даних"
app:layout_constraintBottom_toTopOf="@+id/btnAddRecord"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_chainStyle="packed" />

<Button
android:id="@+id/btnAddRecord"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="16dp"
android:text="Додати ще один запис"
app:layout_constraintBottom_toTopOf="@+id/btnUpdateRecord"
app:layout_constraintTop_toBottomOf="@+id/btnShowData" />

<Button
android:id="@+id/btnUpdateRecord"
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="16dp"
android:text="Змінити останній запис"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintTop_toBottomOf="@+id/btnAddRecord" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

task1/DisplayActivity.kt

```
package ua.kpi.lab3_1

import android.os.Bundle
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

class DisplayActivity : AppCompatActivity() {

    private lateinit var dbHelper: DatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_display)

        dbHelper = DatabaseHelper(this)

        val tvDatabaseContent = findViewById<TextView>(R.id.tvDatabaseContent)

        // Fetch and display all data from the database
        val allData = dbHelper.getAllStudents()
        tvDatabaseContent.text = allData
    }
}
```

task1/activity_display.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<TextView
android:id="@+id/tvDatabaseContent"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:textColor="#000000"
android:textSize="18sp" />

</ScrollView>
```

task2/DatabaseHelper.kt

```
package ua.kpi.lab3_2

import android.content.ContentValues
import android.content.Context
import android.database.sqlite.SQLiteDatabase
```

```
import android.database.sqlite.SQLiteOpenHelper

// Helper class for managing SQLite database with updated schema
class DatabaseHelper(context: Context) :
SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {
    companion object {
        private const val DATABASE_NAME = "university.db"

        // Changed version to 2 to trigger onUpgrade method
        private const val DATABASE_VERSION = 2

        // Table and new columns names
        private const val TABLE_NAME = "classmates"
        private const val COLUMN_ID = "id"
        private const val COLUMN_LAST_NAME = "last_name"
        private const val COLUMN_FIRST_NAME = "first_name"
        private const val COLUMN_PATRONYMIC = "patronymic"
        private const val COLUMN_TIME = "time_added"
    }

    override fun onCreate(db: SQLiteDatabase) {
        // Create table query with new separated name fields
        val createTableQuery = ("CREATE TABLE $TABLE_NAME ("
        + "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, "
        + "$COLUMN_LAST_NAME TEXT, "
        + "$COLUMN_FIRST_NAME TEXT, "
        + "$COLUMN_PATRONYMIC TEXT, "
        + "$COLUMN_TIME TEXT)")
        db.execSQL(createTableQuery)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        // Drop older table if exists and create fresh with new structure
        db.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }

    // Delete all records from the table
    fun clearDatabase() {
        val db = this.writableDatabase
        db.execSQL("DELETE FROM $TABLE_NAME")
        db.close()
    }

    // Insert a new student record with separated name parts
    fun addStudent(lastName: String, firstName: String, patronymic: String, time: String) {
        val db = this.writableDatabase
        val values = ContentValues()
        values.put(COLUMN_LAST_NAME, lastName)
        values.put(COLUMN_FIRST_NAME, firstName)
        values.put(COLUMN_PATRONYMIC, patronymic)
        values.put(COLUMN_TIME, time)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }

    // Update the last added record's separated name fields
    fun updateLastStudent(lastName: String, firstName: String, patronymic: String) {
        val db = this.writableDatabase
        // Update where ID is the maximum ID in the table
        val updateQuery = ("UPDATE $TABLE_NAME SET "
        + "$COLUMN_LAST_NAME = '$lastName', "
        + "$COLUMN_FIRST_NAME = '$firstName', "
```

```
        + "$COLUMN_PATRONYMIC = '$patronymic' "
        + "WHERE $COLUMN_ID = (SELECT MAX($COLUMN_ID) FROM $TABLE_NAME)")
        db.execSQL(updateQuery)
        db.close()
    }

    // Retrieve all students as a formatted string with new fields
    fun getAllStudents(): String {
        val db = this.readableDatabase
        val cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        val stringBuilder = StringBuilder()

        if (cursor.moveToFirst()) {
            do {
                val id = cursor.getInt(cursor.getColumnIndexOrThrow(COLUMN_ID))
                val lastName = cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_LAST_NAME))
                val firstName = cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_FIRST_NAME))
                val patronymic = cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_PATRONYMIC))
                val time = cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_TIME))

                stringBuilder.append("ID: $id\n")
                stringBuilder.append("Прізвище: $lastName\n")
                stringBuilder.append("Ім'я: $firstName\n")
                stringBuilder.append("По-батькові: $patronymic\n")
                stringBuilder.append("Час: $time\n\n")
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()

        return stringBuilder.toString()
    }
}
```

task2/MainActivity.kt

```
package ua.kpi.lab3_2

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale

class MainActivity : AppCompatActivity() {

    private lateinit var dbHelper: DatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        dbHelper = DatabaseHelper(this)

        // Clear database and insert 5 initial records on startup
        dbHelper.clearDatabase()
        insertInitialData()

        val btnShowData = findViewById<Button>(R.id.btnShowData)
```

```
        val btnAddRecord = findViewById<Button>(R.id.btnAddRecord)
        val btnUpdateRecord = findViewById<Button>(R.id.btnUpdateRecord)

        // Button 1: Open new activity to display data
        btnShowData.setOnClickListener {
            val intent = Intent(this, DisplayActivity::class.java)
            startActivity(intent)
        }

        // Button 2: Add one more record with separated name parts
        btnAddRecord.setOnClickListener {
            val currentTime = getCurrentTime()
            dbHelper.addStudent("Новенко", "Степан", "Іванович", currentTime)
            Toast.makeText(this, "Record added", Toast.LENGTH_SHORT).show()
        }

        // Button 3: Update the last record with specific separated names
        btnUpdateRecord.setOnClickListener {
            dbHelper.updateLastStudent("Петренко", "Петро", "Петрович")
            Toast.makeText(this, "Last record updated", Toast.LENGTH_SHORT).show()
        }
    }

    // Helper method to insert 5 students with separated names
    private fun insertInitialData() {
        dbHelper.addStudent("Іванов", "Іван", "Іванович", getCurrentTime())
        dbHelper.addStudent("Шевченко", "Тарас", "Григорович", getCurrentTime())
        dbHelper.addStudent("Косач", "Лариса", "Петрівна", getCurrentTime())
        dbHelper.addStudent("Франко", "Іван", "Якович", getCurrentTime())
        dbHelper.addStudent("Стус", "Василь", "Семенович", getCurrentTime())
    }

    // Helper method to get formatted current time
    private fun getCurrentTime(): String {
        val sdf = SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault())
        return sdf.format(Date())
    }
}
```

task2/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
 ↪  xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<Button
android:id="@+id/btnShowData"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Показати базу даних"
app:layout_constraintBottom_toTopOf="@+id/btnAddRecord"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_chainStyle="packed" />

<Button
android:id="@+id/btnAddRecord"
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

```
android:layout_marginTop="16dp"
android:text="Додати ще один запис"
app:layout_constraintBottom_toTopOf="@+id/btnUpdateRecord"
app:layout_constraintTop_toBottomOf="@+id/btnShowData" />

<Button
android:id="@+id/btnUpdateRecord"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="16dp"
android:text="Змінити останній запис"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintTop_toBottomOf="@+id/btnAddRecord" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

task2/DisplayActivity.kt

```
package ua.kpi.lab3_2

import android.os.Bundle
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

class DisplayActivity : AppCompatActivity() {

    private lateinit var dbHelper: DatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_display)

        dbHelper = DatabaseHelper(this)

        val tvDatabaseContent = findViewById<TextView>(R.id.tvDatabaseContent)

        // Fetch and display all data from the database
        val allData = dbHelper.getAllStudents()
        tvDatabaseContent.text = allData
    }
}
```

task2/activity_display.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">

<TextView
android:id="@+id/tvDatabaseContent"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:textColor="#000000"
android:textSize="18sp" />

</ScrollView>
```