**ЛЕКЦІЯ 17. РОБОТА З БАЗАМИ ДАНИХ SQLITE**

**12.1 Підключення до бази даних SQLite**

**12.1.1 Загальні основи**

В Android є вбудована підтримка однієї з поширених систем управління базами даних – SQLite. Для цього в пакеті *android.database.sqlite* визначено набір класів, які дозволяють працювати з базами даних SQLite. Кожен додаток може створити свою базу

даних.

Щоб використовувати SQLite в Android, треба створити базу даних за допомогою відповідного виразу на мові SQL. Після цього база даних буде зберігатися в катало зі застосунку *DATA/data/[назва\_застосунку]/databases/[назва\_файла\_бази\_ даних]*.

ОС Android по замовчуванню вже містить ряд вбудованих баз SQLite, які використовуються стандартними програмами – для списку контактів, для зберігання фотографій з камери, музичних альбомів і т. д.

Основну функціональність по роботі з базами даних надає пакет *android.database*. Функціональність безпосередньо для роботи з SQLite знаходиться в пакеті *android.database.sqlite*.

База даних в SQLite представлена класом *android.database.sqlite.SQLiteDatabase*.

Він дозволяє виконувати запити до БД, виконувати з нею різні маніпуляції.

Клас *android.database.sqlite.SQLiteCursor* надає запит і дозволяє повертати набір рядків, які відповідають цьому запиту.

Клас *android.database.sqlite.SQLiteQueryBuilder* дозволяє створювати SQL- запити. Самі sql-вирази представлені класом *android.database.sqlite.SQLiteStatement*, які дозволяють за допомогою плейсхолдерів вставляти в вирази динамічні дані.

Клас *android.database.sqlite.SQLiteOpenHelper* дозволяє створити базу даних з усіма таблицями, якщо їх ще не існує.

У SQLite застосовується наступна система типів даних:

● **INTEGER**: представляє ціле число, аналог типу *int* в Java;

● **REAL**: представляє число з плаваючою точкою, аналог *float* і *double* в Java;

● **TEXT**: представляє набір символів, аналог *String* і *char* в Java;

● **BLOB**: представляє масив бінарних даних, наприклад, зображення, аналог типу *int* в Java.

Дані, що зберігаються в базі, повинні відповідати типам даних в мові програмуваня Java.

**12.1.2 Створення та відкриття бази даних**

Для створення або відкриття нової бази даних з коду *Activity* в Android ми можемо викликати метод *openOrCreateDatabase*(). Цей метод може приймати три параметри:

● назва для бази даних;

● числове значення, яке визначає режим роботи (як правило, у вигляді константи **MODE\_PRIVATE**);

● необов’язковий параметр у вигляді об’єкта *SQLiteDatabase.CursorFactory*,

який представляє фабрику створення курсора для роботи з БД.

Наприклад, створення бази даних *app.db*:

SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);

Для виконання запиту до бази даних можна використовувати метод *execSQL* класу *SQLiteDatabase*. У цей метод передається SQL-вираз. Наприклад, створення в базі даних таблиці *users*:

SQLiteDatabase db=getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);

db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)");

Якщо нам треба не просто виконати вираз, але і отримати з БД будь-які дані, то використовується метод *rawQuery*(). Цей метод в якості параметра приймає SQL-вираз, а також набір значень для виразу SQL. Наприклад, отримання всіх об’єктів з бази даних:

SQLiteDatabase db=getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);

db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)"); Cursor query=db.rawQuery("SELECT \* FROM users;", null);

if(query.moveToFirst()){

String name=query.getString(0);

int age=query.getInt(1);

}

Метод *db.rawQuery()* повертає об’єкт *Cursor*, за допомогою якого ми можемо витягти отримані дані.

Можлива ситуація, коли в базі даних не буде об’єктів, і для цього методом *query.moveToFirst()* намагаємося переміститися до першого об’єкту, отриманого з БД. Якщо цей метод поверне значення *false*, значить запит не отримав ніяких даних з БД.

Тепер для роботи з базою даних розробимо простий додаток. Для цього створимо новий проект. У файлі *activity\_main.xml* визначимо найпростіший графічний інтерфейс:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android) android:id=["@+id](mailto:@+id)/activity\_main"

android:layout\_width="match\_parent" android:layout\_height="match\_parent" android:padding="16dp" android:orientation="vertical">

<Button android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" android:text="Click" android:onClick="onClick"/>

<TextView android:id=["@+id](mailto:@+id)/textView" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" android:textSize="18sp" />

</LinearLayout>

А в класі *MainActivity* визначимо взаємодію з базою даних:

public class MainActivity extends AppCompatActivity {

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

public void onClick(View view){

SQLiteDatabase db=getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);

db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)"); db.execSQL("INSERT INTO users VALUES ('Tom Smith', 23);"); db.execSQL("INSERT INTO users VALUES ('John Dow', 31);");

Cursor query = db.rawQuery("SELECT \* FROM users;", null); TextView textView = (TextView) findViewById(R.id.textView); if(query.moveToFirst()){

do{

String name=query.getString(0);

int age=query.getInt(1);

textView.append("Name: "+name+" Age: "+age+"\n");

}

while(query.moveToNext());

}

query.close();

db.close();

}

}

При натисканні на кнопку тут спочатку створюється в базі даних *app.db* нова

таблиця *users*, а потім в неї додаються два об’єкти в базу даних за допомогою SQL-

виразу **INSERT**.

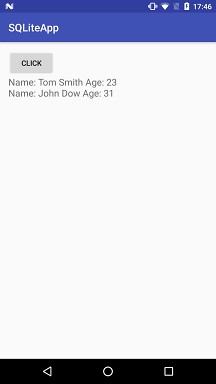
Далі за допомогою виразу **SELECT** отримуємо всіх доданих користувачів з бази даних у вигляді курсору *Cursor*.

Викликом *query.moveToFirst()* переміщаємося в курсорі до першого об’єкту, і так як у нас може бути більше одного об’єкта, то проходимо по всьому курсору в циклі *do ... while*.

Для отримання даних з курсору застосовуються методи *query.getString(0)* і *query.getInt(1)*. У дужках в методи передається номер стовпчика, з якого ми отримуємо дані. Наприклад, вище ми додали спочатку ім’я користувача у вигляді рядка, а потім вік у вигляді числа. Значить, нульовим стовпцем буде йти рядку значення, яке отримуємо за допомогою методу *getString*(), а першим стовпцем йде числове значення, для якого застосовується метод *getInt*().

Після завершення роботи з курсором і базою даних ми закриваємо всі пов’язані об’єкти. Якщо ми не закриємо курсор, то можемо зіткнутися з проблемою витоку пам’яті: query.close(); db.close();

І якщо ми тепер звернемося до додатка, то після натискання на кнопку в текстове поле будуть виведені дані:



**12.2 SimpleCursorAdapter і отримання даних**

**12.2.1 Використання об’єкта SQLiteOpenHelper**

Підемо далі і створимо повністю інтерфейс для роботи з базою даних. Отже, створимо новий проект. Для спрощення роботи з базами даних SQLite в Android нерідко застосовується клас *SQLiteOpenHelper*. Для використання необхідно створити клас-спадкоємець від *SQLiteOpenHelper*, перевизначивши як мінімум два його методи:

● *onCreate()*: викликається при спробі доступу до бази даних, але коли ще ця база даних не створена;

● *onUpgrade()*: викликається, коли потрібно оновити схеми бази даних. Тут можна перебудувати раніше створену базу даних в onCreate(), встановивши відповідні правила перетворення від старої БД до нової.

Тому додамо в проект в ту ж папку, де знаходиться клас *MainActivity*, новий клас

*DatabaseHelper*:

public class DatabaseHelper extends SQLiteOpenHelper {

private static final String DATABASE\_NAME="userstore.db"; // назва бд

private static final int SCHEMA=1; // версія

static final String TABLE="users"; // назва таблиць в бд

// назви стовпців

public static final String COLUMN\_ID="\_id";

public static final String COLUMN\_NAME="name";

public static final String COLUMN\_YEAR="year";

public DatabaseHelper(Context context) {

super(context, DATABASE\_NAME, null, SCHEMA);

}

[@Override](mailto:@Override)

public void onCreate(SQLiteDatabase db) {

db.execSQL("CREATE TABLE users ("+COLUMN\_ID

+" INTEGER PRIMARY KEY AUTOINCREMENT,"+COLUMN\_NAME

+" TEXT, "+COLUMN\_YEAR+" INTEGER);");

// початкові дані

db.execSQL("INSERT INTO "+TABLE+" ("+COLUMN\_NAME

+", "+COLUMN\_YEAR+") VALUES ('Том Сміт', 1981);");

}

[@Override](mailto:@Override)

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

db.execSQL("DROP TABLE IF EXISTS "+TABLE);

onCreate(db);

}

}

Якщо база даних відсутня або її версія (яка задається у змінній **SCHEMA**) вище

поточної, то спрацьовує метод *onCreate*().

Для виконання запитів до бази даних нам потрібно об’єкт *SQLiteDatabase*, який представляє базу даних. Метод *onCreate*() отримує в якості параметра базу даних програми.

Для виконання запитів до SQLite використовується метод *execSQL*(). Він приймає SQL-вираз **CREATE TABLE**, який створює таблицю. Тут також при необхідності ми можемо виконати й інші запити, наприклад, додати будь-які

початкові дані. Так, в даному випадку за допомогою того ж методу і виразу

SQL **INSERT**, додається один об’єкт в таблицю.

У методі *onUpgrade*() відбувається оновлення схеми БД. В даному випадку для прикладу використаний примітивний похід з видаленням попередньої бази даних за допомогою SQL-виразу **DROP** і подальшим її створенням. Але в реальності якщо вам буде необхідно зберегти дані, цей метод може включати більш складну логіку – додавання нових стовпців, видалення непотрібних, додавання додаткових даних і т. д.

Далі визначимо в файлі *activity\_main.xml* наступну розмітку:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android) android:orientation="vertical"

android:layout\_width="match\_parent" android:layout\_height="match\_parent">

<TextView android:id=["@+id](mailto:@+id)/header" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:textSize="18dp"/>

<ListView android:id=["@+id](mailto:@+id)/list" android:layout\_width="match\_parent" android:layout\_height="match\_parent" />

</LinearLayout>

Тут визначено список *ListView*, для відображення отриманих даних, з заголовком, який буде виводити число отриманих об’єктів. Змінимо код класу *MainActivity* наступним чином:

public class MainActivity extends AppCompatActivity { ListView userList;

TextView header; DatabaseHelper databaseHelper; SQLiteDatabase db;

Cursor userCursor; SimpleCursorAdapter userAdapter;

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main);

header = (TextView)findViewById(R.id.header);

userList = (ListView)findViewById(R.id.list);

databaseHelper = new DatabaseHelper(getApplicationContext());

}

[@Override](mailto:@Override)

public void onResume() {

super.onResume();

// відкриваємо підключення

db = databaseHelper.getReadableDatabase();

//отримуємо дані з бд у вигляді курсора

userCursor = db.rawQuery("select \* from "+ DatabaseHelper.TABLE, null);

// визначаємо стовпці, які будуть виводитися в ListView

String[] headers = new String[] {DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};

// створюємо адаптер

userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item, userCursor, headers,

new int[]{android.R.id.text1, android.R.id.text2}, 0);

header.setText("Знайдено елементів: "+String.valueOf(userCursor.getCount()));

userList.setAdapter(userAdapter);

}

[@Override](mailto:@Override)

public void onDestroy(){

super.onDestroy();

// Закриваємо підключення і курсор

db.close();

userCursor.close();

}

}

У методі *onCreate*() відбувається створення об’єкта *SQLiteOpenHelper*. Сама ініціалізація об’єктів для роботи з базою даних відбувається в методі *onResume*(), який спрацьовує після методу *onCreate*().

Щоб отримати об’єкт бази даних, слід використовувати метод *getReadableDatabase*() (отримання бази даних для читання) або *getWritableDatabase*(). Так як в даному випадку ми будемо лише зчитувати дані з бд, то скористаємося першим методом: db=sqlHelper.getReadableDatabase().

**12.2.2 Отримання даних і Cursor**

Android надає різні способи для здійснення запитів до об’єкта SQLiteDatabase. У більшості випадків ми можемо застосовувати метод *rawQuery*(), який приймає два параметри: SQL-вираз **SELECT** і додатковий параметр, що задає параметри запиту.

Після виконання запиту *rawQuery*() повертає об’єкт *Cursor*, який зберігає результат виконання SQL-запиту:

userCursor = db.rawQuery("select\*from"+DatabaseHelper.TABLE, null);

Клас *Cursor* пропонує ряд методів для управління вибіркою, зокрема:

● *getCount()*: Отримує кількість витягнутих з бази даних об’єктів.

● методи *moveToFirst()* і *moveToNext*() дозволяють переходити до першого і до наступного елементів вибірки. Метод *isAfterLast*() дозволяє перевірити, чи досягнуто кінець вибірки;

● Методи *get...(columnIndex*) (наприклад, *getLong*(), *getString*()) дозволяють за індексом стовпчика звернутися до даного стовпця поточного рядка.

**12.2.3 CursorAdapter**

Додатково, для управління курсором в Android є клас *CursorAdapter*. Він дозволяє адаптувати отриманий за допомогою курсора набір до відображення в спискових елементах на кшталт *ListView*. Як правило, при роботі з курсором використовується підклас *CursorAdapter* – *SimpleCursorAdapter*. Хоча можна використовувати і інші адаптери, типу *ArrayAdapter*.

userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item, userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);

userList.setAdapter(userAdapter);

Конструктор класу *SimpleCursorAdapter* приймає шість параметрів:

● **Першим параметром** виступає контекст, з яким асоціюється адаптер, наприклад, поточна *activity*;

● **Другий параметр** – ресурс розмітки інтерфейсу, який буде використовуватися для відображення результатів вибірки;

● **Третій параметр** – курсор;

● **Четвертий параметр** – список стовпців з вибірки, які будуть відображатися в розмітці інтерфейсу;

● **П’ятий параметр** – елементи всередині ресурсу розмітки, які будуть відображати значення стовпців з четвертого параметра;

● **Шостий параметр** – прапори, що задають поведінки адаптера.

При використанні *CursorAdapter* і його підкласів слід враховувати, що вибірка курсора повинна включати цілочисельний стовпець з назвою *\_id*, який повинен бути унікальним для кожного елемента вибірки. Значення цього стовпця при натисканні на елемент списку потім передається в метод обробки *onListItemClick*(), завдяки чому ми можемо по *id* ідентифікувати натиснутий елемент.

В даному випадку у нас перший стовпець як раз називається *"\_id"*.

Після завершення роботи курсор повинен бути закритий методом *close*().

І також треба враховувати, що якщо ми використовуємо курсор в *SimpleCursorAdapter*, то ми не можемо використовувати метод *close*(), поки не завершимо використання *SimpleCursorAdapter*. Тому метод *cursor* краще викликати в методі onDestroy() фрагмента або *activity*.

І якщо ми запустимо додаток то побачимо список з одного доданого елемента:



**12.3 Додавання, видалення та оновлення даних в SQLite**

**12.3.1 ContentValues**

Продовжимо роботу з проектом. Тепер додамо в нього стандартну CRUD-логіку (створення, оновлення, видалення). Щоб не нагромаджувати форму з головною *activity*, всі інші дії по роботі з даними будуть відбуватися на іншому екрані. Додамо в проект новий клас *activity*, який назвемо *UserActivity*.

У файлі *activity\_user.xml* визначимо форму для додавання/оновлення/видалення

даних:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android) android:orientation="vertical"

android:layout\_width="match\_parent" android:layout\_height="match\_parent" android:padding="16dp">

<EditText android:id=["@+id](mailto:@+id)/name" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:hint="Введіть ім’я"/>

<EditText android:id=["@+id](mailto:@+id)/year" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:hint="Введіть рік нарождения"/>

<LinearLayout android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:orientation="horizontal">

<Button android:id=["@+id/sav](mailto:@+id)eButton" android:layout\_width="0dp" android:layout\_weight="1" android:layout\_height="wrap\_content"

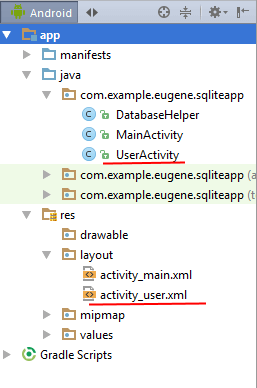
android:text="Зберегти"

android:onClick="save"/>

<Button android:id=["@+id/del](mailto:@+id)eteButton" android:layout\_width="0dp" android:layout\_weight="1" android:layout\_height="wrap\_content" android:text="Видалити" android:onClick="delete"/>

</LinearLayout>

</LinearLayout>



Також змінимо код *UserActivity*:

public class UserActivity extends AppCompatActivity { EditText nameBox;

EditText yearBox; Button delButton; Button saveButton;

DatabaseHelper sqlHelper; SQLiteDatabase db;

Cursor userCursor;

long userId=0;

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_user); nameBox=(EditText) findViewById(R.id.name); yearBox=(EditText) findViewById(R.id.year); delButton=(Button) findViewById(R.id.deleteButton); saveButton= (Button) findViewById(R.id.saveButton); sqlHelper=new DatabaseHelper(this); db=sqlHelper.getWritableDatabase();

Bundle extras=getIntent().getExtras();

if (extras !=null) {

userId=extras.getLong("id");

}

// якщо 0, то додаємо

if (userId > 0) {

// отримуємо елемент по id з бд

userCursor=db.rawQuery("select\*from "+DatabaseHelper.TABLE +" where " + DatabaseHelper.COLUMN\_ID+"=?", new String[]{String.valueOf(userId)});

userCursor.moveToFirst(); nameBox.setText(userCursor.getString(1)); yearBox.setText(String.valueOf(userCursor.getInt(2))); userCursor.close();

} else {

// ховаємо кнопку видалення

delButton.setVisibility(View.GONE);

}

}

public void save(View view){ ContentValues cv=new ContentValues(); cv.put(DatabaseHelper.COLUMN\_NAME, nameBox.getText().toString()); cv.put(DatabaseHelper.COLUMN\_YEAR,

Integer.parseInt(yearBox.getText().toString()));

if (userId > 0) {

db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN\_ID + "=" + String.valueOf(userId), null);

} else {

db.insert(DatabaseHelper.TABLE, null, cv);

}

goHome();

}

public void delete(View view){

db.delete(DatabaseHelper.TABLE, "\_id=?", new String[]{String.valueOf(userId)});

goHome();

}

private void goHome(){

// закриваємо підключення

db.close();

// перехід до головної activity

Intent intent=new Intent(this, MainActivity.class);

intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);

startActivity(intent);

}

}

При оновленні або видаленні об’єкта зі списку з головною activity в *UserActivity*

передаватиметься *id* об’єкта:

long userId=0;

//.................

Bundle extras=getIntent().getExtras();

if (extras !=null) {

userId=extras.getLong("id");

}

Якщо з *MainActivity* не було передано *id*, то встановлюємо його значення 0, отже,

у нас буде додавання, а не редагування/знищення. Якщо *id* визначено, то отримуємо по ньому з бази даних об’єкт для редагування/видалення:

if (id < 0) {

userCursor=db.rawQuery("select\*from "+DatabaseHelper.TABLE+" where "+ DatabaseHelper.COLUMN\_ID+"=?", new String[]{String.valueOf(id)});

userCursor.moveToFirst(); nameBox.setText(userCursor.getString(1)); yearBox.setText(String.valueOf(userCursor.getInt(2))); userCursor.close();

}

Інакше просто приховуємо кнопку видалення. Для виконання операцій по

вставці, оновленню і видаленню даних *SQLiteDatabase* має методи *insert*(), *update*() і

*delete*(). Ці методи викликаються в обробниках кнопок:

db.delete(DatabaseHelper.TABLE, "\_id=?", new String[]{String.valueOf(id)});

У метод *delete*() передається назва таблиці, а також стовпець, по якому відбувається видалення, і його значення. В якості критерію можна вибрати кілька стовпців, тому третім параметром йде масив. Знак питання *?* позначає параметр, замість якого підставляється значення з третього параметра.

**12.3.2 ContentValues**

Для додавання або оновлення нам треба створити об’єкт *ContentValues*. Даний об’єкт являє собою словник, який містить набір пар «ключ-значення». Щоб додати до цього словника новий об’єкт застосовується метод *put()*. Перший параметр методу – це ключ, а другий – значення, наприклад:

ContentValues cv=new ContentValues(); cv.put("NAME", "Tom"); cv.put("YEAR", 30);

В якості значення в метод *put*() можна передавати рядки, цілі числа, числа з плаваючою точкою. В даному ж випадку додаються введені в текстове поля значення: ContentValues cv =new ContentValues();

cv.put(DatabaseHelper.COLUMN\_NAME, nameBox.getText().toString());

cv.put(DatabaseHelper.COLUMN\_YEAR, Integer.parseInt(yearBox.getText().toString()));

При оновленні в метод *update*() передається назва таблиці, об’єкт *ContentValues* і критерій, за яким відбувається оновлення (в даному випадку стовпець *id*):

db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN\_ID+"="+ String.valueOf(id), null);

Метод *insert*() приймає назву таблиці, об’єкт *ContentValues* з додаваними значеннями. Другий параметр є необов’язковим: він передає стовпець, в який треба додати значення *NULL*: db.insert(DatabaseHelper.TABLE, null, cv).

Замість цих методів можна використовувати метод *execSQL*() з точним зазначенням виконуваного SQL-виразу. У той же час методи *delete / insert / update* мають перевагу – вони повертають *id* зміненого запису, за яким ми можемо дізнатися про успішність операції, або -1 у випадку невдалої операції:

long result=db.insert(DatabaseHelper.TABLE, null, cv);

if(result > 0){

// дії

}

Після кожної операції виконується метод *goHome*(), який повертає на головну

*activity*.

Після цього нам треба виправити код *MainActivity*, щоб вона ініціювала виконання коду в *UserActivity*. Для цього змінимо код *activity\_main.xml*:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android) android:orientation="vertical"

android:layout\_width="match\_parent" android:layout\_height="match\_parent">

<Button android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:text="Добавити" android:onClick="add" android:textSize="18dp"/>

<ListView android:id=["@+id](mailto:@+id)/list" android:layout\_width="match\_parent" android:layout\_height="match\_parent" />

</LinearLayout>

В даному випадку була додана кнопка для виклику *UserActivity*. Змінимо відповідним чином код класу *MainActivity*:

public class MainActivity extends AppCompatActivity { ListView userList;

DatabaseHelper databaseHelper; SQLiteDatabase db;

Cursor userCursor; SimpleCursorAdapter userAdapter;

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main); userList=(ListView)findViewById(R.id.list); userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {

[@Override](mailto:@Override)

public void onItemClick(AdapterView<?> parent, View view, int position, long id) { Intent intent=new Intent(getApplicationContext(), UserActivity.class); intent.putExtra("id", id);

startActivity(intent);

}

});

databaseHelper = new DatabaseHelper(getApplicationContext());

}

[@Override](mailto:@Override)

public void onResume() { super.onResume(); db=databaseHelper.getReadableDatabase();

userCursor=db.rawQuery("select\*from "+DatabaseHelper.TABLE, null); String[] headers=new String[] {DatabaseHelper.COLUMN\_NAME,

DatabaseHelper.COLUMN\_YEAR};

userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item, userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);

userList.setAdapter(userAdapter);

}

// при натисканні на кнопку запускаємо UserActivity для добавлення даних

public void add(View view){

Intent intent=new Intent(this, UserActivity.class);

startActivity(intent);

}

[@Override](mailto:@Override)

public void onDestroy(){ super.onDestroy(); db.close();

userCursor.close();

}

}

При натисканні на кнопку запускається *UserActivity*, при цьому не передається

ніякого *id*, тобто в *UserActivity id* буде дорівнює нулю, значить буде відбуватися додавання даних:

public void add(View view){

Intent intent=new Intent(this, UserActivity.class);

startActivity(intent);

}

Іншу ситуацію представляє обробник натискання на елемент списку – при

натисканні також буде запускатися *UserActivity*, але тепер буде передаватися *id*

обраного запису:

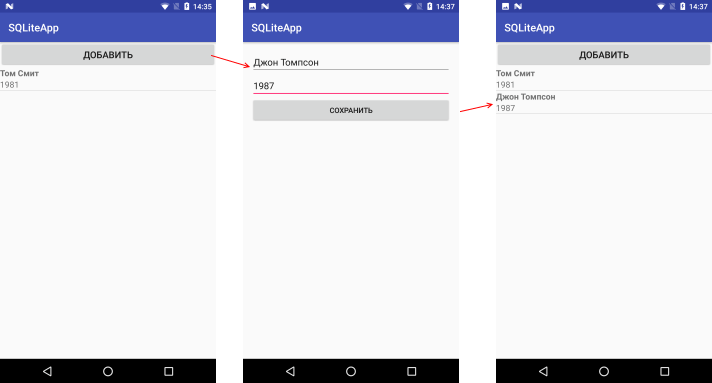
public void onItemClick(AdapterView<?> parent, View view, int position, long id) { Intent intent=new Intent(getApplicationContext(), UserActivity.class); intent.putExtra("id", id);

startActivity(intent);

}

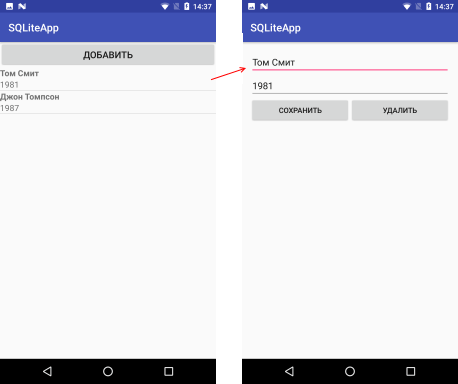
Запустимо програму і натиснемо на кнопку, яка має перенаправляти на

*UserActivity*:



При натисканні в *MainActivity* на елемент списку цей елемент потрапить на

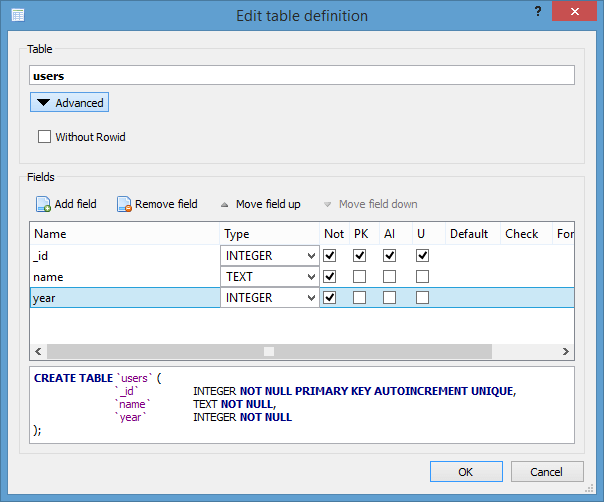
*UserActivity*, де його можна буде видалити або відредагувати:



**12.4 Використання існуючої БД SQLite**

Крім створення нової бази даних ми також можемо використовувати вже існуючу. Візьмемо проект, створений в попередніх темах. Для початку створимо базу даних SQLite. У цьому нам може допомогти такий інструмент як **Sqlitebrowser**. Він безкоштовний і доступний для різних операційних систем за адресою <http://sqlitebrowser.org/>. Хоча можна використовувати і інші способи для створення початкової БД.

Sqlitebrowser представляє графічний інтерфейс для створення бази даних і визначення в ній всіх необхідних таблиць:



Як видно з рисунку, визначаємо таблицю *users* з трьома полями: *\_id*, *name*, *age*.

Команда на створення таблиці буде наступною:

CREATE TABLE `users` (

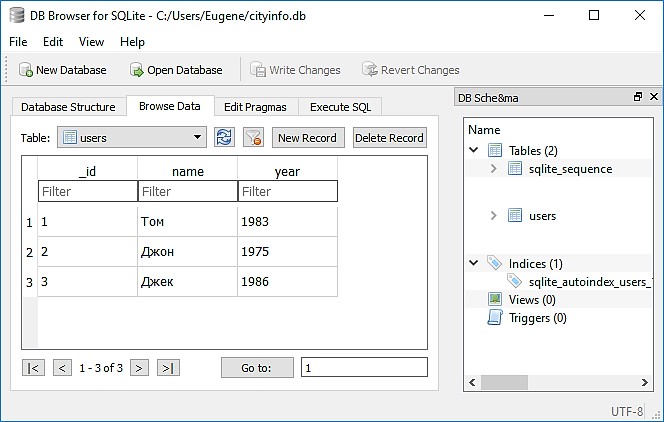
`\_id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,

`name` TEXT NOT NULL,

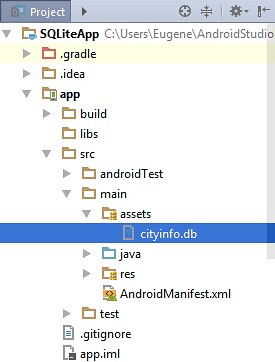
`year` INTEGER NOT NULL

);

Там же в програмі додамо кілька елементів в створену таблицю:



Після створення таблиці додамо в проект в Android Studio папку *assets*, а в папку *assets* – щойно створену базу даних. Для цього перейдемо до повного визначення проекту, натиснемо на папку *main* правою кнопкою миші і в меню виберемо *New->Directory*. Назвемо додавану папку *assets* і потім скопіюємо в неї нашу базу даних:



В нашому випадку база даних називається *"cityinfo.db"*. Змінимо код

*DatabaseHelper* наступним чином:

class DatabaseHelper extends SQLiteOpenHelper {

private static String DB\_PATH;

private static String DB\_NAME="cityinfo.db";

private static final int SCHEMA=1; static final String TABLE="users"; static final String COLUMN\_ID="\_id";

static final String COLUMN\_NAME="name"; static final String COLUMN\_YEAR="year"; private Context myContext;

DatabaseHelper(Context context) { super(context, DB\_NAME, null, SCHEMA); this.myContext=context;

DB\_PATH=context.getFilesDir().getPath()+DB\_NAME;

}

[@Override](mailto:@Override)

public void onCreate(SQLiteDatabase db) {

}

[@Override](mailto:@Override)

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

}

void create\_db(){

InputStream myInput=null; OutputStream myOutput=null; try {

File file=new File(DB\_PATH);

if (!file.exists()) {

this.getReadableDatabase();

// отримуємо локальну бд як потік

myInput = myContext.getAssets().open(DB\_NAME);

// шлях до нової бд

String outFileName=DB\_PATH;

// відкриваємо пусту бд

myOutput=new FileOutputStream(outFileName);

// побайтово копіюємо дані byte[] buffer=new byte[1024]; int length;

while ((length=myInput.read(buffer)) > 0) {

myOutput.write(buffer, 0, length);

} myOutput.flush(); myOutput.close(); myInput.close();

}

}

catch(IOException ex){ Log.d("DatabaseHelper", ex.getMessage());

}

}

public SQLiteDatabase open()throws SQLException {

return SQLiteDatabase.openDatabase(DB\_PATH, null, SQLiteDatabase.OPEN\_READWRITE);

}

}

По замовчуванню база даних буде розміщуватися в зовнішньому сховищі, що

виділяється для програми у папці */data/data/[назва\_пакета]/databases/*, і щоб отримати повний шлях до бази даних в конструкторі використовується вираз: DB\_PATH=context.getFilesDir().getPath()+DB\_NAME;

Метод *onCreate*() нам не потрібен, так як нам не потрібне створення вбудованої бази даних. Зате тут визначено додатковий метод *create*\_*db()*, мета якого копіювання бази даних з папки *assets* в те місце, яке зазначено в змінній **DB\_PATH**.

Крім цього тут також визначено метод відкриття бази даних *open*() за допомогою методу *SQLiteDatabase.openDatabase*().

Новий спосіб організації підключення змінить використання *DatabaseHelper* в

*activity*. Тому, слід оновити клас *MainActivity*:

public class MainActivity extends AppCompatActivity { ListView userList;

DatabaseHelper databaseHelper; SQLiteDatabase db;

Cursor userCursor; SimpleCursorAdapter userAdapter;

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main); userList=(ListView)findViewById(R.id.list); userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {

[@Override](mailto:@Override)

public void onItemClick(AdapterView<?> parent, View view, int position, long id) { Intent intent=new Intent(getApplicationContext(), UserActivity.class); intent.putExtra("id", id);

}

});

startActivity(intent);

databaseHelper=new DatabaseHelper(getApplicationContext());

databaseHelper.create\_db();

}

[@Override](mailto:@Override)

public void onResume() { super.onResume(); db=databaseHelper.open();

userCursor=db.rawQuery("select\*from "+DatabaseHelper.TABLE, null); String[] headers=new String[] {DatabaseHelper.COLUMN\_NAME,

DatabaseHelper.COLUMN\_YEAR};

userAdapter=new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item, userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);

userList.setAdapter(userAdapter);

}

// при натисканні на кнопку запускаємо UserActivity для добавлення даних

public void add(View view){

Intent intent=new Intent(this, UserActivity.class);

startActivity(intent);

}

[@Override](mailto:@Override)

public void onDestroy(){ super.onDestroy(); db.close();

userCursor.close();

}

}

Змінимо клас *UserActivity* таким чином:

public class UserActivity extends AppCompatActivity { EditText nameBox;

EditText yearBox;

Button delButton; Button saveButton; DatabaseHelper sqlHelper; SQLiteDatabase db;

Cursor userCursor;

long userId=0;

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_user); nameBox=(EditText) findViewById(R.id.name); yearBox=(EditText) findViewById(R.id.year); delButton=(Button) findViewById(R.id.deleteButton); saveButton=(Button) findViewById(R.id.saveButton); sqlHelper=new DatabaseHelper(this); db=sqlHelper.open();

Bundle extras=getIntent().getExtras();

if (extras !=null) {

userId=extras.getLong("id");

}

// якщо 0, то додаємо

if (userId > 0) {

// отримуємо елемент по id із бд

userCursor=db.rawQuery("select\*from "+DatabaseHelper.TABLE+"where"+ DatabaseHelper.COLUMN\_ID+"=?", new String[]{String.valueOf(userId)});

userCursor.moveToFirst(); nameBox.setText(userCursor.getString(1)); yearBox.setText(String.valueOf(userCursor.getInt(2))); userCursor.close();

} else {

// ховаємо кнопку видалення

delButton.setVisibility(View.GONE);

}

}

public void save(View view){ ContentValues cv=new ContentValues(); cv.put(DatabaseHelper.COLUMN\_NAME, nameBox.getText().toString()); cv.put(DatabaseHelper.COLUMN\_YEAR,

Integer.parseInt(yearBox.getText().toString()));

if (userId > 0) {

db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN\_ID+"="+ String.valueOf(userId), null);

} else {

db.insert(DatabaseHelper.TABLE, null, cv);

}

goHome();

}

public void delete(View view){

db.delete(DatabaseHelper.TABLE, "\_id=?", new String[]{String.valueOf(userId)});

goHome();

}

private void goHome(){

// закриваємо підключення

db.close();

// перехід до головної activity

Intent intent = new Intent(this, MainActivity.class);

intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);

startActivity(intent);

}

}

Вся інша робота з даними буде тією ж, щоб і в минулих темах. В результаті

отримаємо на екрані:



**12.5 Динамічний пошук по базі даних SQLite**

Розглянемо, як ми можемо створити в додатку на Android динамічний пошук по базі даних SQLite. Створимо новий проект з порожньою *MainActivity*. Для цього проекту візьмемо базу даних з минулої теми. Дана база даних називається *cityinfo* і має одну таблицю *users* з трьома полями *\_id*, *name*, *age*:

CREATE TABLE `users` (

`\_id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,

`name` TEXT NOT NULL,

`year` INTEGER NOT NULL

);

Додамо в проект в Android Studio папку *assets*, а в папку *assets* – щойно створену

базу даних. У нашому випадку база даних називається *"cityinfo.db"*. Додамо в проект в одну папку з *MainActivity* новий клас *DatabaseHelper*. Перейдемо до файлу *activity\_main.xml*, який визначає візуальний інтерфейс, і змінимо його наступним чином:

<LinearLayout xmlns:android=["http://schemas.android.com/apk/res/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/userFilter"](mailto:@+id)

android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:hint="Пошук" />

<ListView android:id=["@+id](mailto:@+id)/userList" android:layout\_width="match\_parent" android:layout\_height="wrap\_content">

</ListView>

</LinearLayout>

Отже, у нас буде елемент *ListView* для відображення списку і текстове поле для фільтрації. Тепер змінимо код *MainActivity*:

public class MainActivity extends AppCompatActivity { DatabaseHelper sqlHelper;

SQLiteDatabase db; Cursor userCursor; SimpleCursorAdapter userAdapter; ListView userList;

EditText userFilter;

[@Override](mailto:@Override)

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main);

userList = (ListView)findViewById(R.id.userList); userFilter = (EditText)findViewById(R.id.userFilter); sqlHelper = new DatabaseHelper(getApplicationContext());

// створюємо бд

sqlHelper.create\_db();

}

[@Override](mailto:@Override)

public void onResume() {

super.onResume();

try {

db=sqlHelper.open();

userCursor=db.rawQuery("select \* from "+DatabaseHelper.TABLE, null); String[] headers=new String[]{DatabaseHelper.COLUMN\_NAME,

DatabaseHelper.COLUMN\_YEAR};

userAdapter=new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item, userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);

// якщо в текстовому полі є текст, то здійснюємо фільтрацію

// дана перевірка необхідна при переході з однієї орієнтації екрану до іншої

if(!userFilter.getText().toString().isEmpty())

userAdapter.getFilter().filter(userFilter.getText().toString());

// встановлення слухача змін тексту

userFilter.addTextChangedListener(new TextWatcher() {

public void afterTextChanged(Editable s) { }

public void beforeTextChanged(CharSequence s, int start, int count, int after) { }

// при зміні тексту виконуємо фільтрацію

public void onTextChanged(CharSequence s, int start, int before, int count) {

userAdapter.getFilter().filter(s.toString());

}

});

// встановлюємо провайдер фільтрації

userAdapter.setFilterQueryProvider(new FilterQueryProvider() {

[@Override](mailto:@Override)

public Cursor runQuery(CharSequence constraint) {

if (constraint == null || constraint.length() == 0) {

return db.rawQuery("select\*from" DatabaseHelper.TABLE, null);

}

else {

return db.rawQuery("select \* from "+DatabaseHelper.TABLE+" where"+ DatabaseHelper.COLUMN\_NAME+" like ?", new String[]{"%"+constraint.toString()+

"%"});

}

}

});

userList.setAdapter(userAdapter);

}

catch (SQLException ex){}

}

[@Override](mailto:@Override)

public void onDestroy(){ super.onDestroy(); db.close();

userCursor.close();

}

}

Перш за все треба відзначити, що для фільтрації даних в адаптері, нам треба

отримати фільтр адаптера, а у цього фільтра виконати метод *filter*():

userAdapter.getFilter().filter(s.toString());

У цей метод *filter*() передається ключ пошуку. Для текстового поля ми можемо відстежувати зміни вмісту за допомогою слухача:

userFilter.addTextChangedListener(new TextWatcher() {

public void afterTextChanged(Editable s) {

}

public void beforeTextChanged(CharSequence s, int start, int count, int after) {

}

// здійснюємо фільтрацію при зміні тексту

public void onTextChanged(CharSequence s, int start, int before, int count) {

userAdapter.getFilter().filter(s.toString());

}

});

У слухачі *TextWatcher* в методі *onTextChanged* якраз і викликається метод *filter*(),

в який передається введена користувачем в текстове поле послідовність символів.

Сам виклик методу *filter*() мало на що впливає. Нам потрібно ще визначити провайдер фільтрації адаптера, які і буде інкапсулювати реальну логіку фільтрації: userAdapter.setFilterQueryProvider(new FilterQueryProvider() {

[@Override](mailto:@Override)

public Cursor runQuery(CharSequence constraint) {

if (constraint == null || constraint.length() == 0) {

return db.rawQuery("select\*from"+DatabaseHelper.TABLE, null);

}

else {

return db.rawQuery("select\*from"+DatabaseHelper.TABLE+"where" + DatabaseHelper.COLUMN\_NAME+"like ?", new String[]{"%"+

constraint.toString()+"%"});

}

}

});

Сутність цього провайдера полягає у виконанні SQL-виразів до БД, а саме

конструкцій *"select from"* і *"select from where like"*. Дані найпростіші вирази виконують чутливі до регістру фільтрацію. В результаті адаптар отримує відфільтровані дані.

Слід також відзначити наступний код:

if(!userFilter.getText().toString().isEmpty())

userAdapter.getFilter().filter(userFilter.getText().toString());

Даний код нам потрібен при зміні орієнтації (наприклад, з портретної на альбомну). І якщо орієнтація пристрою змінена, але в текстовому полі все ж є деякі текст-фільтри, то виконується фільтрація. Інакше вона б не виконувалася.

Після запуску додатку ми зможемо користуватися фільтрацією даних:

