Android SQLite – Database

Poradnik o przechowywaniu danych w systemie operacyjnym Android

Opracował: Arkadiusz Krupiński

Spis treści

Android SQLite – Database 1
Poradnik o przechowywaniu danych w systemie operacyjnym Android 1
Czym jest SQLite? 3
Dlaczego SQLite? 3
Zapisywanie danych za pomocą SQLite na systemie operacyjnym Android 3
Implementacja klasy DatabaseHelper 3
Implementacja klasy DBManager 5
MainActivity 7
Przykładowa realizacja 8
Literatura 8

Czym jest SQLite?

SQLite [1] to biblioteka języka C, która implementuje mały, szybki, samodzielny, wysoce niezawodny, w pełni funkcjonalny silnik bazy danych SQL. W przeciwieństwie do większości innych baz danych SQL, SQLite nie ma osobnego procesu serwera. SQLite czyta i zapisuje bezpośrednio do zwykłych plików na dysku.

Dlaczego SQLite?

SQLite [3] jest popularnym wyborem jako wbudowane oprogramowanie bazy danych do przechowywania danych lokalnych w oprogramowaniu aplikacyjnym, takim jak przeglądarki internetowe. Jest to prawdopodobnie najbardziej rozpowszechniony silnik bazy danych [1].
SQLite to kompaktowa biblioteka. Po włączeniu wszystkich funkcji [1] rozmiar biblioteki może być mniejszy niż 600 kB. Wydajność jest zwykle całkiem dobra, nawet w środowiskach o niskiej pamięci. W zależności od tego, jak jest używany, SQLite może być szybszy niż bezpośrednie wejście / wyjście systemu plików.

Zapisywanie danych za pomocą SQLite na systemie operacyjnym Android

Zakładam, że znasz podstawy z baz danych SQL.
Interfejs API potrzebny do korzystania z bazy danych SQLite na systemie operacyjnym Andorid dostępne są w pakiecie ***andorid.database.sqlite***.

Zrealizowana zostanie baza danych przechowywująca podstawowe informacje o studencie, m.in.: indeks, imię, nazwisko.

Klasa pomocnicza

Jedną z głównych zasad - których się nauczyłem podczas opracowania tematu -
baz danych SQL jest schemat: formalna deklaracja organizacji bazy danych.
Pomocne może być utworzenie klasy, która wyraźnie określa układ schematu w sposób systematyczny. Klasa pomocnicza jest kontenerem dla stałych, które definiują nazwy dla identyfikatorów tabel i kolumn. Klasa kontraktu pozwala używać tych samych stałych we wszystkich innych klasach w tym samym pakiecie. Umożliwia to zmianę nazwy kolumny w jednym miejscu i propagowanie jej w całym kodzie.

Implementacja klasy DatabaseHelper

Na samym początku warto utworzyć klasę pomocniczą – *DatabaseHelper* i zamieścić w niej podstawowe dane tworzące bazę danych. Przykład poniżej:

```
//  https://developer.android.com/training/data-storage/sqlite

public class DatabaseHelper {

//      Table Name
    public static final String TABLE_NAME = "STUDENTS";

//      Table columns
    public static final String _ID = "_id";
    public static final String INDEX = "INDEX_NR";
    public static final String NAME = "NAME";
    public static final String SURNAME = "SURNAME";

//      Database Information
    private static final String DB_NAME = "STUDENT.DB";
```

```
//     Database version
    private static final int DB_VERSION = 1;
//     Creating table query
private static final String CREATE_TABLE =
        "create table" + " " + TABLE_NAME + "("
                + _ID + " " + "INTEGER PRIMARY KEY AUTOINCREMENT,"
                + INDEX + " " + "TEXT NOT NULL,"
                + NAME + " " + "TEXT,"
                + SURNAME + " " + "TEXT" + ");";
...
```

Podobnie jak pliki zapisywane w pamięci wewnętrznej urządzenia, system Android przechowuje bazę danych w prywatnym folderze aplikacji [2]. Dane są bezpieczne, ponieważ domyślnie ten obszar nie jest dostępny dla innych aplikacji ani użytkownika.

Klasa *SQLiteOpenHelper* zawiera przydatny zestaw interfejsów API do zarządzania bazą danych [2]. Gdy używasz tej klasy do uzyskiwania referencji do bazy danych, system wykonuje potencjalnie długotrwałe operacje tworzenia i aktualizowania bazy danych tylko w razie potrzeby, a nie podczas uruchamiania aplikacji.
Właściwie wszystko, co programista musi zrobić to wywołać funkcję getWritableDatabase() lub getReadableDatabase().

Należy zaimportować następujące zależności, aby wszystko prawidłowo funkcjonowało.

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
```

Aby użyć SQLiteOpenHelper, utwórz podklasę, która zastąpi metody wywołania zwrotnego onCreate () i onUpgrade ().
Możesz także zaimplementować metody onDowngrade () lub onOpen (), ale nie są one wymagane.

Definiujemy dalej naszą klasę:

```
public class DatabaseHelper extends SQLiteOpenHelper {
//    This takes the Context (e.g., an Activity)
    public DatabaseHelper(@Nullable Context context) {
//        When the application runs the first time – At this point, we do not yet
          have a database.
//        So we will have to create the tables, indexes, starter data, and so on.
        super(context, DB_NAME, null, DB_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
//        Execute a single SQL statement that is NOT a SELECT or any other SQL
          statement that returns data.
        db.execSQL(CREATE_TABLE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
//        Execute a single SQL statement that is NOT a SELECT or any other SQL
          statement that returns data.
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
```

```
} // end of class
```

Implementacja klasy DBManager

Warto w tym momencie utworzyć kolejną klasę zarządzającą naszą bazą danych, pozwalającą na użycie podstawowych poleceń: select, insert, update, delete rekordów w tabeli.
Pierw utworzymy podstawowe pola przechowujące dane potrzebne do pracy z bazą danych SQLite oraz podstawowe metody, które pozwolą zarządzać nią zanim wydamy polecenia, m.in. select etc.

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

public class DBManager {
    private DatabaseHelper databaseHelper;
    private Context context;
    private SQLiteDatabase database;

    public DBManager(Context c) {
        context = c;
    }

//    Before performing any database operations like insert, update, delete
      records in a table, first open the database connection
    public DBManager open() {
        databaseHelper = new DatabaseHelper(context);
//        Create and/or open a database that will be used for reading and writing.
        database = databaseHelper.getWritableDatabase();
        return this;
    }
//    Close any open database object.
    public void close() {
        databaseHelper.close();
    }
}
```

Gdy mamy to już za sobą teraz wystarczy dodać metody, które pozwalają pracę z bazą danych.

Wprowadzanie danych

```
/* Inserting new Record into Android SQLite database table
   Returning the primary key value of the new row
   or it will return -1 if there was an error inserting the data.
   This can happen if you have a conflict with pre-existing data in the database.
*/
    public long insert(String index, String name, String surname) {
// Content Values creates an empty set of values using the given initial size
// ContentValues class is used to store a set of values
        ContentValues contentValue = new ContentValues();

        contentValue.put(DatabaseHelper.INDEX, index);
        contentValue.put(DatabaseHelper.NAME, name);
        contentValue.put(DatabaseHelper.SURNAME, surname);

// Insert the new row, returning the primary key value of the new row
        long newRowId = database.insert(DatabaseHelper.TABLE_NAME, null,
                                                                contentValue);
        return newRowId;
    }
```

Pobieranie danych

```
//    Read information from a database
    public Cursor fetch() {
//     Define a projection that specifies which columns from the database
//     you will actually use after this query.
        String[] projection = {
                DatabaseHelper._ID,
                DatabaseHelper.INDEX,
                DatabaseHelper.NAME,
                DatabaseHelper.SURNAME
        };

//     How you want the results sorted in the resulting Cursor
    String sortOrder =
            databaseHelper.INDEX + " DESC";

    Cursor cursor = database.query(
            DatabaseHelper.TABLE_NAME,      // The table to query
            projection,   // The array of columns to return (pass null to get all)
            null,                // The columns for the WHERE clause
            null,               // The values for the WHERE clause
            null,               // don't group the rows
            null,               // don't filter by row groups
            sortOrder           // The sort order
    );

//    Once the query is fetched a call to cursor.moveToFirst() is made.
//    Calling moveToFirst() it moves the cursor to the first result
//    (when the set is not empty)
        if (cursor != null) {
            cursor.moveToFirst();
        }

    return cursor;
    }
```

Usuwanie danych

```
//    Deleting a Record in Android SQLite database table
public void delete(long _id) {
    database.delete(DatabaseHelper.TABLE_NAME, DatabaseHelper._ID + "=" + _id,
                                                                    null);

}
```

Aktualizacja danych

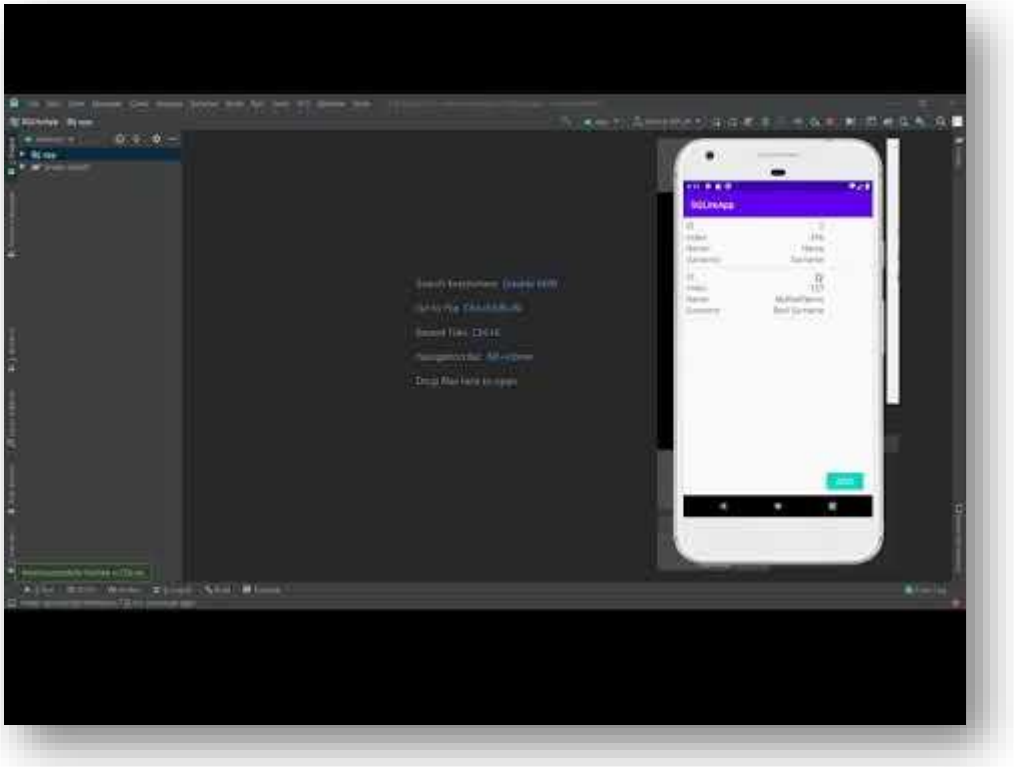
```
//    Updating Record in Android SQLite database table
    public int update(long _id, String index, String name, String surname) {
//    Content Values creates an empty set of values using the given initial size
//    ContentValues class is used to store a set of values
        ContentValues contentValues = new ContentValues();

        contentValues.put(DatabaseHelper.INDEX, index);
        contentValues.put(DatabaseHelper.NAME, name);
        contentValues.put(DatabaseHelper.SURNAME, surname);
        return database.update(DatabaseHelper.TABLE_NAME, contentValues,
                                        DatabaseHelper._ID + " = " + _id, null);
    }
}
```

MainActivity

Tak przygotowane klasy wystarczy użyć w naszym projekcie.

```
public class MainActivity extends AppCompatActivity {

    private DBManager dbManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // instance of database
        dbManager = new DBManager(getApplicationContext());

        // make this database writable
        dbManager.open();

        // insert; params: Index, Name, Surname
        database.insert("123456", "Name", "Surname");

        // get data from database
        Cursor cursor = dbManager.fetch();

        // about display data
        // display(cursor);

        // update; params: ID, Index, Name, Surname
        dbManager.update(1, "123456", "Name", "Surname");

        // about display data
        // display(cursor);

        // delete; params: ID
        dbManager.delete(1);

        // about display data
        // display(cursor); <EMPTY>
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // always remember to close database
        dbManager.close();
    }
}
```

Przykładowa realizacja



Odnośnik: https://www.youtube.com/watch?v=MS_xKWiBCpA

Literatura

[1] Most Widely Deployed and Used Database Engine; sqlite.org; dostęp 05-05-2020r.
[2] Save data using SQLite; Android Developers; developer.android.com; dostęp 05-05-2020r.
[3] SQLite w Androidzie – kompletny poradnik dla początkujących; lipiec 27, 2011; Mirosław Stanek; android4devs.pl; dostęp 05-05-2020r.