Exercise 6: Content Provider

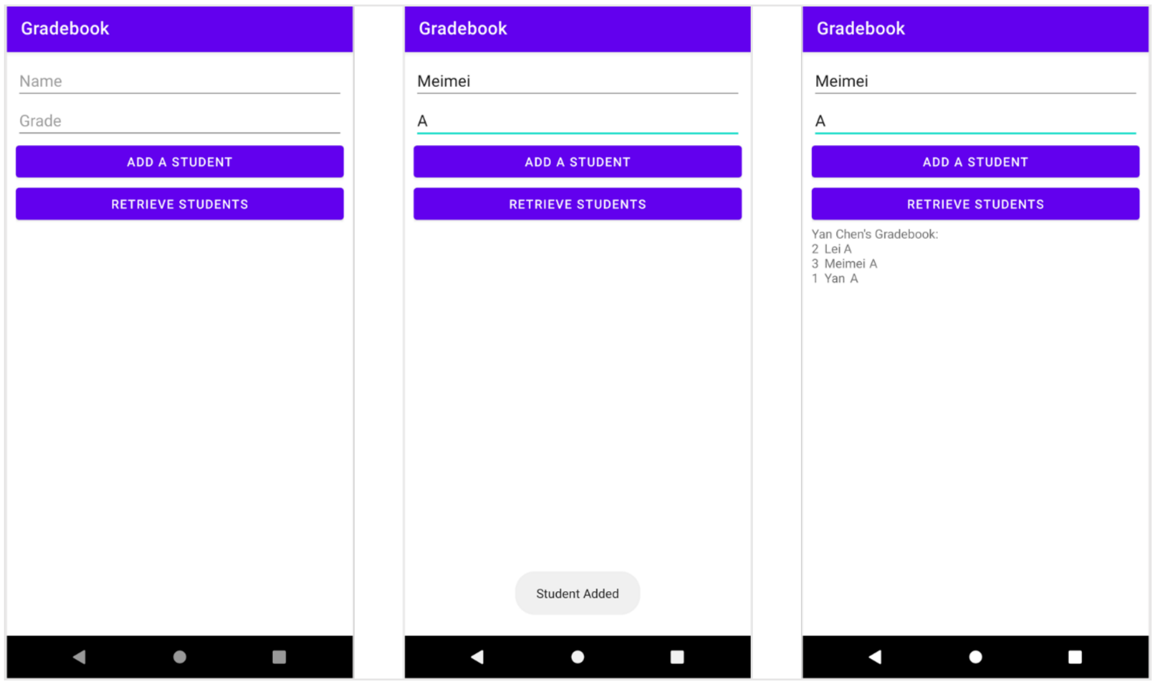
Due: Apr. 25, 23:59 PDT

Overview

This exercise provides an example for inserting and retrieving data from a database of another app through content provider. You will also learn how to check your database in Android Studio.

There will be 2 modules: DataProvider only provides a database and acontent provider without UI; And Gradebook is an app has UI (Fig. 1) and accesses DataProvider's database.

Similar to exercise 3 & 4, put both apps in one project called "Exercise6YourName".



```
Yan Chen's Gradebook:
2  Lei A
3  Meimei A
1  Yan A
```

Fig. 1 Sample run of the app. Left: launch screen; middle: a message will be toasted after adding a student; right: the data base will be displayed (ordered by name) after clicking "Retrieve Students".

Prerequisites:

- Know the process of submitting your work (exercise 0)
- Be familiar with previous topics related to project structure and UI (exercise 1 - 4, lesson 2 - 4)
- Has set up an emulator (API 29)

Procedure related to the above topics will not be provided in the instruction. Refer to corresponding exercise/lecture notes if needed.

If you set any different view id's, filenames, etc., remember to modify the corresponding part of code.

Step 0. Config the project

Add one module so that there are 2 modules. One for DataProvider and one for Gradebook app, with package name “edu.sjsu.android.dataprovider” and “edu.sjsu.android.gradebook” respectively. Check exercise 3 and 4 if you forgot how to add a new module and/or change the package name.

0.1 Create a Content Provider

Right-click the source code package of DataProvider app, choose New -> Other -> Content Provider. Set the Class Name as “StudentsProvider”, and set the URI Authorities as the package name: “edu.sjsu.android.dataprovider”. Check both Exported and Enabled. In this way, a content provider will be automatically registered in manifest. You can also add a content provider by adding a regular Java class that extending ContentProvider, and then manually register that content provider in manifest.

0.2 Add Dependency of DataProvider to Gradebook

You will need to use the URI provided by DataProvider in Gradebook app.

First, set the DataProvider module as a library, instead of an app. In DataProvider’s build.gradle, set the second line id 'com.android.application' to 'com.android.library'. Also, delete the line (~line 10) applicationId "edu.sjsu.android.dataprovider"

Then, add the following dependency in Gradebook’s build.gradle, where :dataprovider is the name of the DataProvider module.

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.2.0'
    ...
    implementation project(':dataprovider')
}
```

Sync the project. If there are any errors, check the module name in settings.gradle file, which should have two include statements for the two modules’ name and one statement that defines the root project name. Replace :dataprovider in above code to the module name of DataProvider shown in settings.gradle file, or change the module name to dataprovider (procedure provided in Exercise 4).

Note that you need to do this because they are two different modules. For mini-project 4, you don’t need to do this since there will be only one app.

Step 1. Implement database (DataProvider)

Create a Java class called “StudentsDB”, extends SQLiteOpenHelper. This class manages a SQL database, makes it easy for StudentsProvider implementations, especially for creating/upgrading the database so the application startup will not be blocked with long-running database upgrades. Implement methods to get rid of the errors. But before that, set some class variables needed.

1.1 Set Class Variables

Let’s set the database name as “studentsDatabase” with version number “1”. It has a table called “students” with 3 columns (ID, name, and grade) to store the data.

And the String variable “CREATE_TABLE” is the SQL commands for creating a table. For each column, you need to declare the data type. For example, “_ID” (must has the “_”!) is the “primary key”, which is unique and not null. And setting it as “AUTOINCREMENT” will automatically generate a unique number every time a new data is added to the table.

```
private static final String DATABASE_NAME = "studentsDatabase";
private static final int VERSION = 1;

private static final String TABLE_NAME = "students";
private static final String ID = "_id";
private static final String NAME = "name";
private static final String GRADE = "grade";

static final String CREATE_TABLE =
        " CREATE TABLE " + TABLE_NAME +
                " ("+ ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
                + NAME + " TEXT NOT NULL, "
                + GRADE + " TEXT NOT NULL);";
```

1.2 Implement Constructor

You must have a constructor that calls any one of the parent’s constructors.

```
public StudentsDB(@Nullable Context context) {
    super(context, DATABASE_NAME, null, VERSION);
}
```

1.3 Implement onCreate

You must implement the onCreate method, which is called when the database is first created. Here we just create an empty table by calling the method to executing the create table command.

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    sqLiteDatabase.execSQL(CREATE_TABLE);
}
```

1.4 Implement onUpgrade

You must implement the onUpgrade method, which is called when the database needs to be upgraded (version changed). You can leave it blank for this exercise since we won’t upgrade this database; or add the following code, which deletes the old table and create a new one.

```
@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldV, int newV) {
    sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
    onCreate(sqLiteDatabase);
}
```

1.5 Implement Custom Methods (for Insert and Retrieve data)

You’ve implemented all required constructor/methods in the previous sub-steps so all errors should be gone. In this sub-step, we are going to implement two methods as helper methods that will be called in StudentsProvider class. We implement them in this class instead of in StudentsProvider since we will need to use the table name, and the method “getWritableDatabase” to create and/or open a database is provided in the parent class (SQLiteOpenHelper).

Note that, when getWritableDatabase is first called, the database will be opened and cached, so we can call this method every time we want to write to the database. According to the official documentation, “...you should not call this method from the application main thread...”, therefore, instead of setting a class variable for the database and initializing it in the instructor, we call getWritableDatabase every time we need it.

```
public long insert(ContentValues contentValues) {
    SQLiteDatabase database = getWritableDatabase();
    return database.insert(TABLE_NAME, null, contentValues);
}

public Cursor getAllStudents(String orderBy) {
    SQLiteDatabase database = getWritableDatabase();
    return database.query(TABLE_NAME,
            new String[]{ID, NAME, GRADE},
            null, null, null, null, orderBy);
}
```

Step 2. Implement content provider (DataProvider)

Open the StudentsProvider class. The starter code should be generated for you. You can delete the constructor since we don’t need it. We will implement onCreate, insert, and query. You can keep all other methods as unimplemented since we won’t use them in this exercise.

2.1 Provide URI

Add a static public variable for the URI that will be used by activities/fragments/other apps when they need to access data through this provider.

Note that the string AUTHORITY must be the same as the what you set in step 0.1.

```
private final static String AUTHORITY = "edu.sjsu.android.dataprovider";
public final static Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY);
```

2.2 Implement onCreate

This class has a StudentsDB object as a class attribute. Construct the object in onCreate.

```
private StudentsDB database;

@Override
public boolean onCreate() {
    database = new StudentsDB(getContext());
    return true;
}
```

2.3 Implement insert

As explained in class, the insert method is used to insert a new row of data. Implement it by calling the insert method in StudentsDB to insert data. The URI returned is for the newly inserted row, and we need to call notifyChange using the new URI.

```
@Override
public Uri insert(Uri uri, ContentValues values) {
    long rowID = database.insert(values);
    //If record is added successfully
    if (rowID > 0) {
        Uri _uri = ContentUris.withAppendedId(uri, rowID);
        getContext().getContentResolver().notifyChange(_uri, null);
        return _uri;
    }
    throw new SQLException("Failed to add a record into " + uri);
}
```

2.4 Implement query

Call the getAllStudents method in StudentsDB to retrieve all data in the database.

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {
    // If not specified, sort by ID
    sortOrder = sortOrder == null ? "_id" : sortOrder;
    return database.getAllStudents(sortOrder);
}
```

Step 3. Setup UI (Gradebook)

The Gradebook app has a UI. So, set up the activity_main.xml under Gradebook app.

Similar to what you did in previous exercises, set the following widgets.

Widget	id	text/hint	onClick
EditText	studentName	android:hint="Name"	n/a
EditText	studentGrade	android:hint="Grade"	n/a
Button	n/a	android:text="Add a Student"	addStudent
Button	n/a	android:text="Retrieve Students"	getAllStudents
TextView	result	set later	n/a

Step 4 Implement MainActivity (Gradebook)

The activity adds/gets data from DataProvider's database (studentsDatabase) based on user input.

4.1 Get URI from StudentsProvider

Add the following class variable. Import StudentsProvider class by ALT (OPTION) + ENTER.

```
private Uri uri = StudentsProvider.CONTENT_URI;
```

4.2 Implement onCreate

If you are using binding, remember to enable it in Gradebook's module build.gradle file, and then inflate binding and set content view to binding.getRoot(). Refer to lesson 3 page 12 or previous exercises/projects if needed. If you don't use binding, you don't need to change onCreate.

4.3 Implement addStudent

AddStudent is the method we set in Step 3 to respond to a click on “Add a Student” button. In this method, we use a ContentValues object to store the data to the database. In put method, the key is the column name, the value is the String you get from the user input. If you are using findViewById instead of binding, but without having EditText objects, you need to explicitly cast them to EditText. For example: ((EditText) findViewById(R.id.studentName)).getText().toString().

Then use getContentResolver method provided in Activity class to get a ContentResolver object for communicating with an accessible content provider, which is StudentsProvider in this exercise. The Uri is used to identify the provider.

```
public void addStudent(View view) {
    ContentValues values = new ContentValues();
    values.put("name", binding.studentName.getText().toString());
    values.put("grade", binding.studentGrade.getText().toString());
    // Toast message if successfully inserted
    if (getContentResolver().insert(uri, values) != null)
        Toast.makeText(this, "Student Added", Toast.LENGTH_SHORT).show();
}
```

4.4 Implement getAllStudents

When a user clicks the “Retrieve Students” button, getAllStudents method will be called, as you set in Step 3. Similarly, use getContentResolver() to communicate with StudentsProvider identified by Uri. The query method in StudentsProvider returns a Cursor object, which is used to get data from a database row by row. So call moveToFirst to read from the first row (the method returns true if moves successfully), and display each row until there is no next row to move to. Remember to change the first line of the result to your own name.

```
public void getAllStudents(View view) {
    // Sort by student name
    try (Cursor c = getContentResolver().
            query(uri, null, null, null, "name")) {

        if (c.moveToFirst()) {
            String result = "Yan Chen's Gradebook: \n";
            do {
                for (int i = 0; i < c.getColumnCount(); i++) {
                    result = result.concat
                            (c.getString(i) + "\t");
                }
                result = result.concat("\n");
            } while (c.moveToNext());
            binding.result.setText(result);
        }
    }
}
```

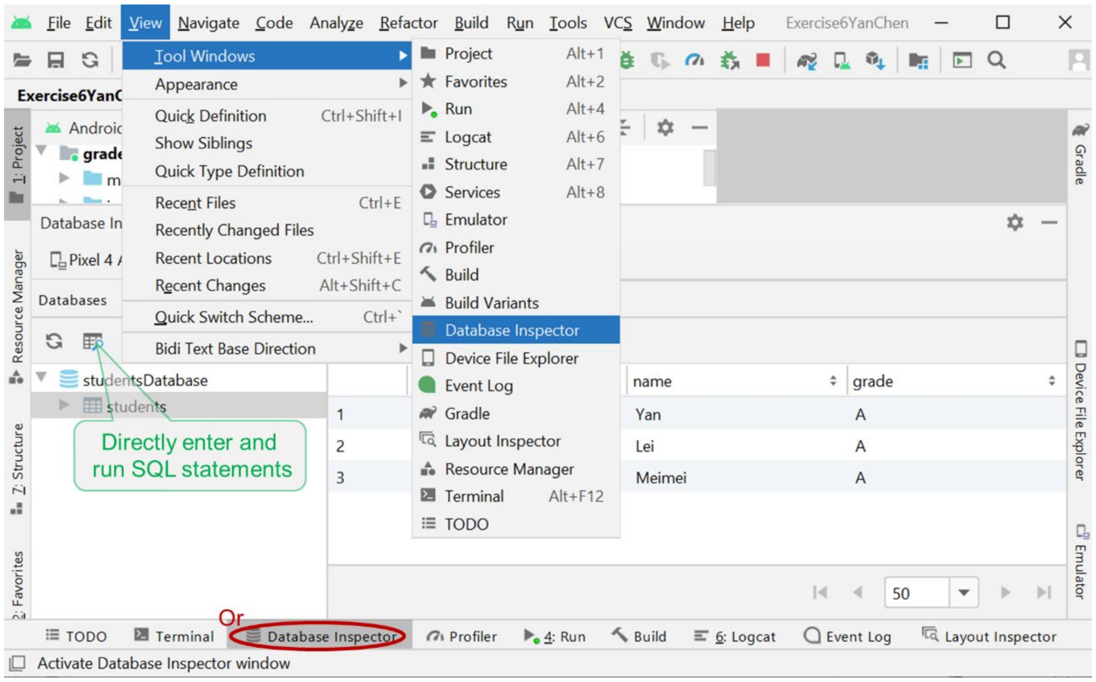
Step 5. Test Gradebook and check the database

Run the Gradebook app to test. You don’t need to run (and actually you can’t) Data Provider.

If the Gradebook app crashes, check if “android:exported” is set to true. If still crashes, try to add permission in DataProvider’s manifest, and request permission in Gradebook’s manifest.

App Name	Code
DataProvider	<permission android:name="edu.sjsu.android.dataprovider.PERMISSION"/>
Gradebook	<uses-permission android:name="edu.sjsu.android.dataprovider.PERMISSION "/>

To check the database, while the app is running, open “Database Inspector” (View -> Tool Windows -> Database Inspector or click the tab down at the bottom, as shown in Fig. 2). Since the database will be created only after getWritableDatabase is called, so you need to add a student first in order to see the database in Database Inspector. Double click the tablename to open the table.



	name	grade
1	Yan	A
2	Lei	A
3	Meimei	A

Fig. 2 Database Inspector

Submission

- Push your project to a Bitbucket repository (name it “exercise6”) by the due date.
- Invite and share your Bitbucket repository the grader (edmond.lin@sjsu.edu) and the instructor (yan.chen01@sjsu.edu).
- Submit repository links, etc. by answering all the questions in the “Exercise 6 - Content Provider” quiz on Canvas.
- Only your last submission before deadline will be graded based on the following criteria:
  2 pts if meets all requirements;
  1 pt if app failed/missing any requirement.