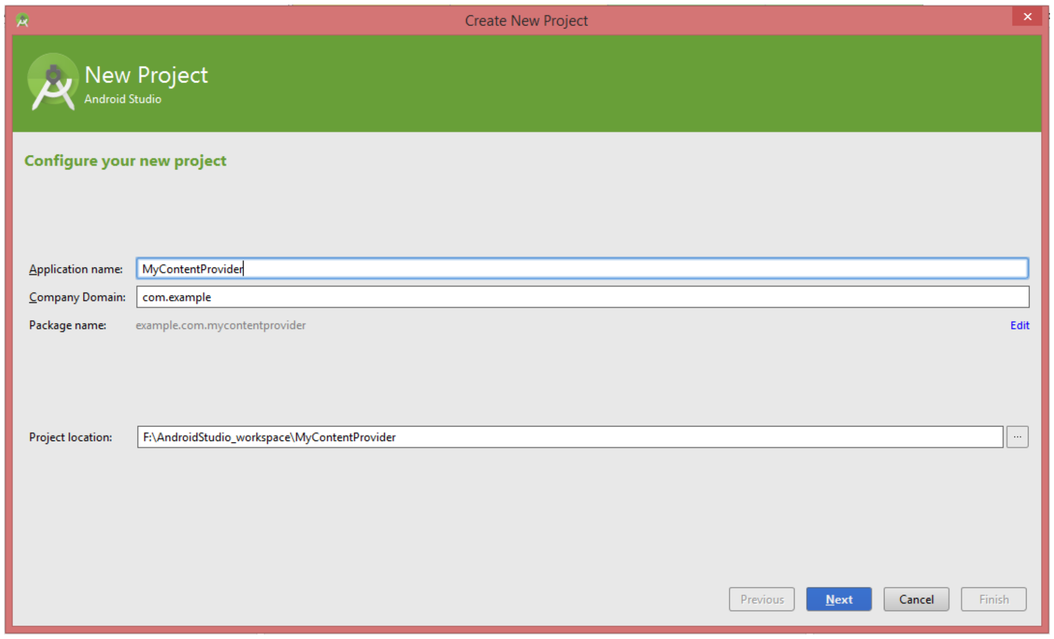
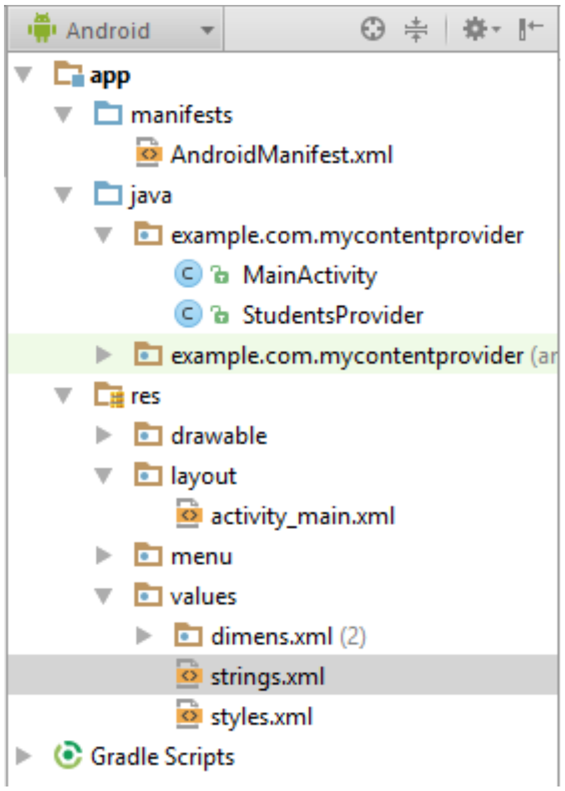
Lab: Android Content Providers (Android Studio)

Step 1: Create a simple Android Application using Android Studio. Follow the option **File -> NewProject**. Now name your application as **MyContentProvider** using the wizard window as follows:



Application name: MyContentProvider
Company Domain: com.example
Package name: example.com.mycontentprovider
Project location: F:\AndroidStudio_workspace\MyContentProvider

Then the file of this project will be display as following:



```
app
  manifests
    AndroidManifest.xml
  java
    example.com.mycontentprovider
      MainActivity
      StudentsProvider
    example.com.mycontentprovider (ar
  res
    drawable
    layout
      activity_main.xml
    menu
    values
      dimens.xml (2)
      strings.xml
      styles.xml
Gradle Scripts
```

Step 2:

Following is the content of the modified main activity file *java/com.example.mycontentprovider/ MainActivity.java*. This file can include each of the fundamental lifecycle methods. We have added two new methods *onClickAddName()* and *onClickRetrieveStudents()* to handle user interaction with the application.

```
package example.com.mycontentprovider;



import android.net.Uri;

import android.os.Bundle;

import android.app.Activity;

import android.content.ContentValues;

import android.content.CursorLoader;

import android.database.Cursor;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.EditText;

import android.widget.Toast;



public class MainActivity extends Activity {

   @Override

   protected void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

      setContentView(R.layout.activity_main);

   }

   @Override

   public boolean onCreateOptionsMenu(Menu menu) {
      // Inflate the menu; this adds items to the action bar if it is present.

      getMenuInflater().inflate(R.menu.menu_main, menu);

      return true;

   }

   public void onClickAddName(View view) {

      // Add a new student record

      ContentValues values = new ContentValues();

      values.put(StudentsProvider.NAME,

              ((EditText)findViewById(R.id.txtName)).getText().toString());

      values.put(StudentsProvider.GRADE,

              ((EditText)findViewById(R.id.txtGrade)).getText().toString());

      Uri uri = getContentResolver().insert(

              StudentsProvider.CONTENT_URI, values);

      Toast.makeText(getBaseContext(),

              uri.toString(), Toast.LENGTH_LONG).show();

   }


   public void onClickRetrieveStudents(View view) {

      // Retrieve student records

      String URL = "content://com.example.provider.College/students";

      Uri students = Uri.parse(URL);

      Cursor c = managedQuery(students, null, null, null, "name");

      if (c.moveToFirst()) {

          do{

              Toast.makeText(this,

                      c.getString(c.getColumnIndex(StudentsProvider._ID)) +
                          ", " +  c.getString(c.getColumnIndex( StudentsProvider.NAME)) +
                          ", " + c.getString(c.getColumnIndex( StudentsProvider.GRADE)),
                      Toast.LENGTH_SHORT).show();
              } while (c.moveToNext());
          }
      }
   }
```

Create new file *StudentsProvider.java* under *com.example.mycontentprovider* package and following is the content of *java/com.example.mycontentprovider/StudentsProvider.java*:

```
package example.com.mycontentprovider;
import java.util.HashMap;
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;
public class StudentsProvider extends ContentProvider {
   static final String PROVIDER_NAME = "com.example.provider.College";
   static final String URL = "content://" + PROVIDER_NAME + "/students";
static final Uri CONTENT_URI = Uri.parse(URL);

static final String _ID = "_id";

static final String NAME = "name";

static final String GRADE = "grade";

private static HashMap<String, String> STUDENTS_PROJECTION_MAP;

static final int STUDENTS = 1;

static final int STUDENT_ID = 2;

static final UriMatcher uriMatcher;

static{

   uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);

   uriMatcher.addURI(PROVIDER_NAME, "students", STUDENTS);

   uriMatcher.addURI(PROVIDER_NAME, "students/#", STUDENT_ID);

}

private SQLiteDatabase db;

static final String DATABASE_NAME = "College";

static final String STUDENTS_TABLE_NAME = "students";

static final int DATABASE_VERSION = 1;

static final String CREATE_DB_TABLE =

      " CREATE TABLE " + STUDENTS_TABLE_NAME +

            " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +

            " name TEXT NOT NULL, " +

            " grade TEXT NOT NULL);";

private static class DatabaseHelper extends SQLiteOpenHelper {

   DatabaseHelper(Context context){

      super(context, DATABASE_NAME, null, DATABASE_VERSION);

   }
   @Override

   public void onCreate(SQLiteDatabase db)

   {

      db.execSQL(CREATE_DB_TABLE);

   }

   @Override

   public void onUpgrade(SQLiteDatabase db, int oldVersion,

                    int newVersion) {

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

      onCreate(db);

   }

}

@Override

public boolean onCreate() {

   Context context = getContext();

   DatabaseHelper dbHelper = new DatabaseHelper(context);

   db = dbHelper.getWritableDatabase();

   return (db == null)? false:true;

}

@Override

public Uri insert(Uri uri, ContentValues values) {

   long rowID = db.insert(      STUDENTS_TABLE_NAME, "", values);

   if (rowID > 0)

   {

      Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);

      getContext().getContentResolver().notifyChange(_uri, null);
      return _uri;
   }
   throw new SQLException("Failed to add a record into " + uri);
}
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                String[] selectionArgs, String sortOrder) {
   SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
   qb.setTables(STUDENTS_TABLE_NAME);
   switch (uriMatcher.match(uri)) {
      case STUDENTS:
         qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
         break;
      case STUDENT_ID:
         qb.appendWhere( _ID + "=" + uri.getPathSegments().get(1));
         break;
      default:
         throw new IllegalArgumentException("Unknown URI " + uri);
   }
   if (sortOrder == null || sortOrder == ""){
      sortOrder = NAME;
   }
   Cursor c = qb.query(db,      projection,       selection, selectionArgs,
         null, null, sortOrder);
   c.setNotificationUri(getContext().getContentResolver(), uri);
   return c;
}
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
   int count = 0;
   switch (uriMatcher.match(uri)){
      case STUDENTS:
         count = db.delete(STUDENTS_TABLE_NAME, selection, selectionArgs);
         break;
      case STUDENT_ID:
         String id = uri.getPathSegments().get(1);
         count = db.delete( STUDENTS_TABLE_NAME, _ID +  " = " + id +
               (!TextUtils.isEmpty(selection) ? " AND (" +
                     selection + ')' : ""), selectionArgs);
         break;
      default:
         throw new IllegalArgumentException("Unknown URI " + uri);
   }
   getContext().getContentResolver().notifyChange(uri, null);
   return count;
}
@Override
public int update(Uri uri, ContentValues values, String selection,
              String[] selectionArgs) {
   int count = 0;
   switch (uriMatcher.match(uri)){
      case STUDENTS:
         count = db.update(STUDENTS_TABLE_NAME, values,
               selection, selectionArgs);
         break;
      case STUDENT_ID:
         count = db.update(STUDENTS_TABLE_NAME, values, _ID +
               " = " + uri.getPathSegments().get(1) +
               (!TextUtils.isEmpty(selection) ? " AND (" +
                     selection + ')' : ""), selectionArgs);
         break;
      default:
         throw new IllegalArgumentException("Unknown URI " + uri );
   }
   getContext().getContentResolver().notifyChange(uri, null);
   return count;
}
@Override
public String getType(Uri uri) {
   switch (uriMatcher.match(uri)){
      case STUDENTS:
         return "vnd.android.cursor.dir/vnd.example.students";
      case STUDENT_ID:
         return "vnd.android.cursor.item/vnd.example.students";
      default:
         throw new IllegalArgumentException("Unsupported URI: " + uri);
   }
   }

}
```

Following will the modified content of *AndroidManifest.xml* file. Here we have added <provider.../> tag to include our content provider:

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

   package="example.com.mycontentprovider" >

   <application

      android:allowBackup="true"

      android:icon="@drawable/ic_launcher"

      android:label="@string/app_name"

      android:theme="@style/AppTheme" >

      <activity

         android:name=".MainActivity"

         android:label="@string/app_name" >

         <intent-filter>

            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />

         </intent-filter>

      </activity>

      <provider android:name="StudentsProvider"

         android:authorities="com.example.provider.College">

      </provider>

   </application>

</manifest>
```

Following will be the content of *res/layout/activity_main.xml* file to include a button to broadcast your custom intent:

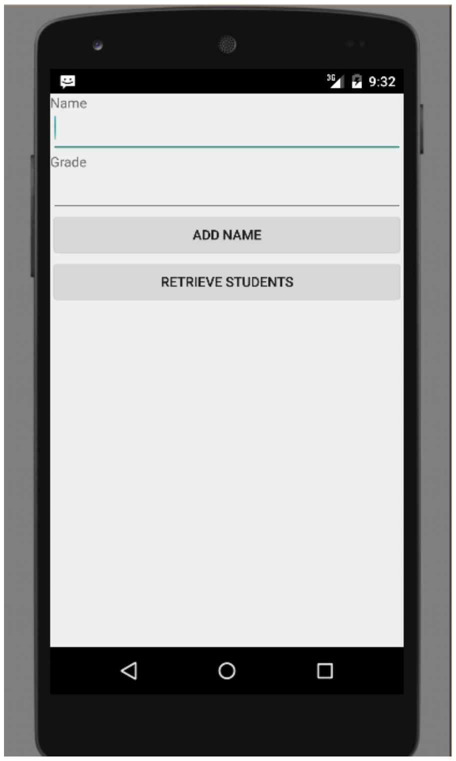
```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

  android:layout_width="fill_parent"

  android:layout_height="fill_parent"

  android:orientation="vertical" >

  <TextView

    android:layout_width="fill_parent"

    android:layout_height="wrap_content"

    android:text="Name" />

  <EditText

    android:id="@+id/txtName"

    android:layout_height="wrap_content"

    android:layout_width="fill_parent" />

  <TextView

    android:layout_width="fill_parent"

    android:layout_height="wrap_content"

    android:text="Grade" />

  <EditText

    android:id="@+id/txtGrade"

    android:layout_height="wrap_content"

    android:layout_width="fill_parent" />

  <Button

    android:text="Add Name"

    android:id="@+id/btnAdd"

    android:layout_width="fill_parent"

    android:layout_height="wrap_content"

    android:onClick="onClickAddName" />
<Button
    android:text="Retrieve Students"
    android:id="@+id/btnRetrieve"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickRetrieveStudents" />
</LinearLayout>
```

Make sure you have following content of *res/values/strings.xml* file:

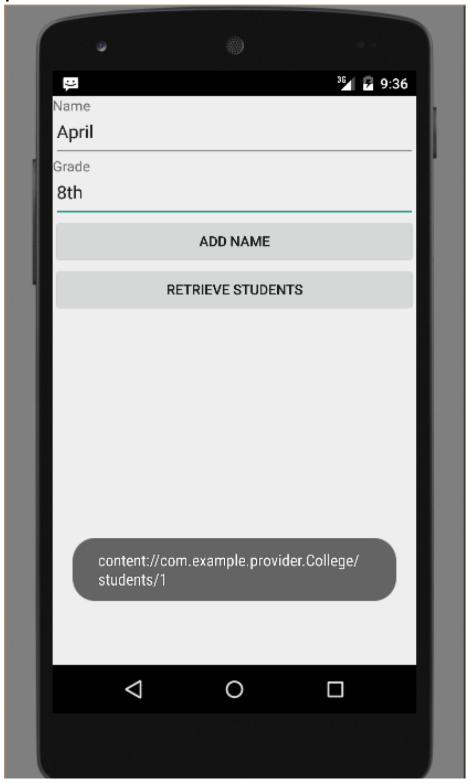
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">MyContentProvider</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_settings">Settings</string>
</resources>
```

Step 3:

Let's try to run our modified **MyContentProvider application** we just modified. I assume you had created your AVD while doing environment setup. At first, start your existed **AVD**, and then click **Run** ▶ icon from the toolbar. Choose this AVD, and click “OK”:



Now let's enter student *Name* and *Grade* and finally click on Add *Name* button, this will add student record in the database and will flash a message at the bottom showing ContentProvider URI along with record number added in the database. This operation makes use of our *insert()* method. Let's repeat this process to add few more students in the database of our content provider.

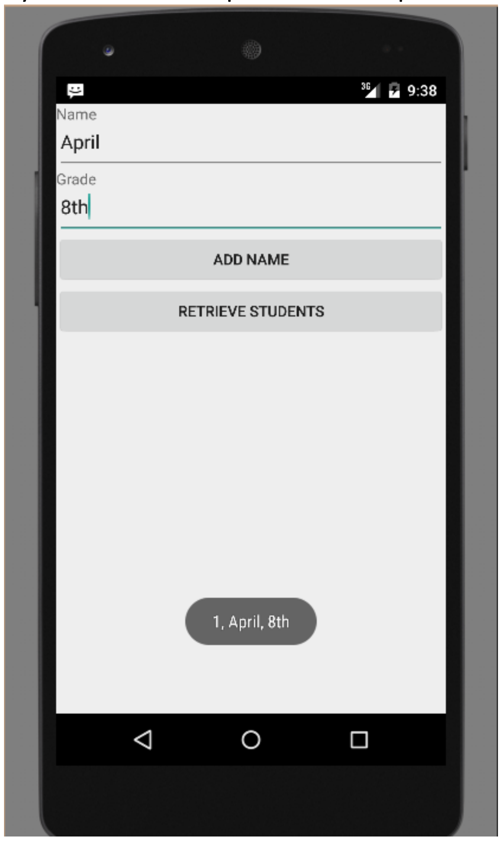


April

8th

content://com.example.provider.College/students/1

Once you are done with adding records in the database, now it`s time to ask ContentProvider to give us those records back, so let's click *Retrieve Students* button which will fetch and display all the records one by one which is as per our the implementation of our *query()* method.



You can write activities against update and delete operations by providing callback functions in *MainActivity.java* file and then modify user interface to have buttons for update and deleted operations in the same way as we have done for add and read operations.

This way you can use existing Content Provider like Address Book or you can use Content Provider concept in developing nice database oriented applications where you can perform all sort of database operations like read, write, update and delete as explained above in the example.