Mobile Application Development
Content Providers

Waterford Institute of Technology

October 26, 2016

John Fitzgerald

Content Providers

Learning Objectives

- Brief review SQLite.
- Introduce Content Provider.
- Explain how to implement in MyRentSQLite app.
- Initial brief introduction to Android services.
- Running service method on worker thread.
- Using service to exercise SQLite functionality.
- Writing a test app to access MyRent content provider.

SQLite

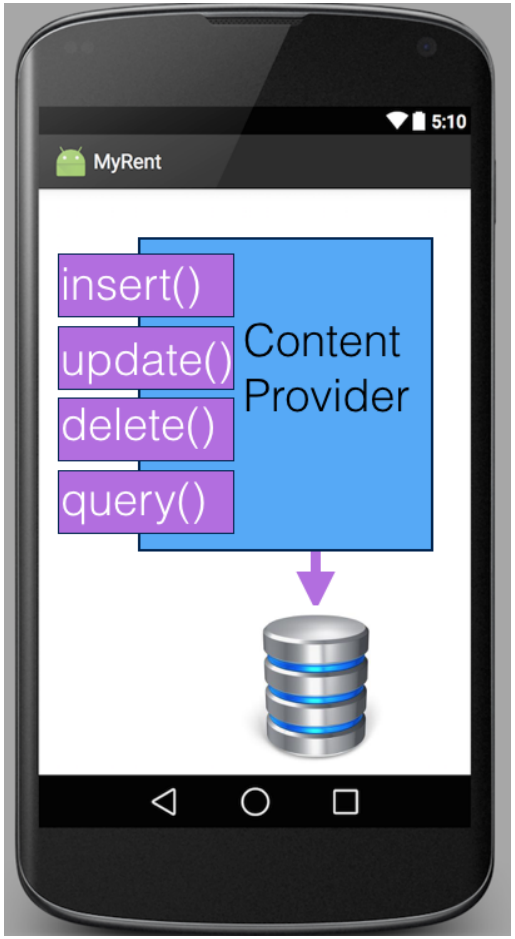
An embedded database engine

- MyRentSQLite app persists data.
- Data still persists when app shut down.
- Local copy data allows offline processing.
- SQLite database functions as cache.
- Later we put data on cloud.
- In lab we simulate cloud.

Content Providers

An interface to application data

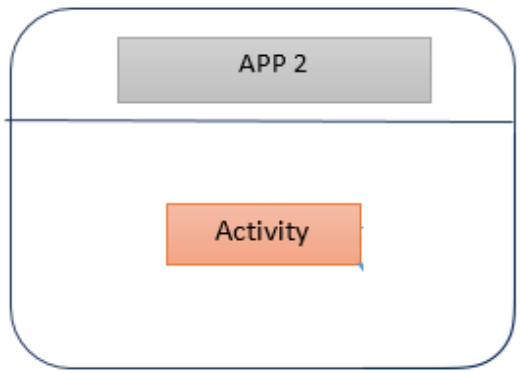
- Presently data inaccessible to other apps.
- ContentProvider: MyRent data accessible.
- Other apps on device may access data.

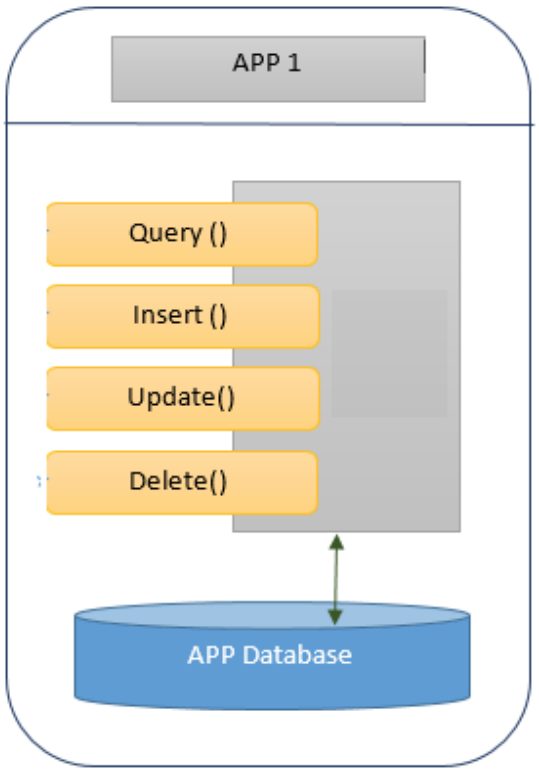


Content Providers

An interface to application data

- Presently APP1 SQLite database inaccessible to APP2

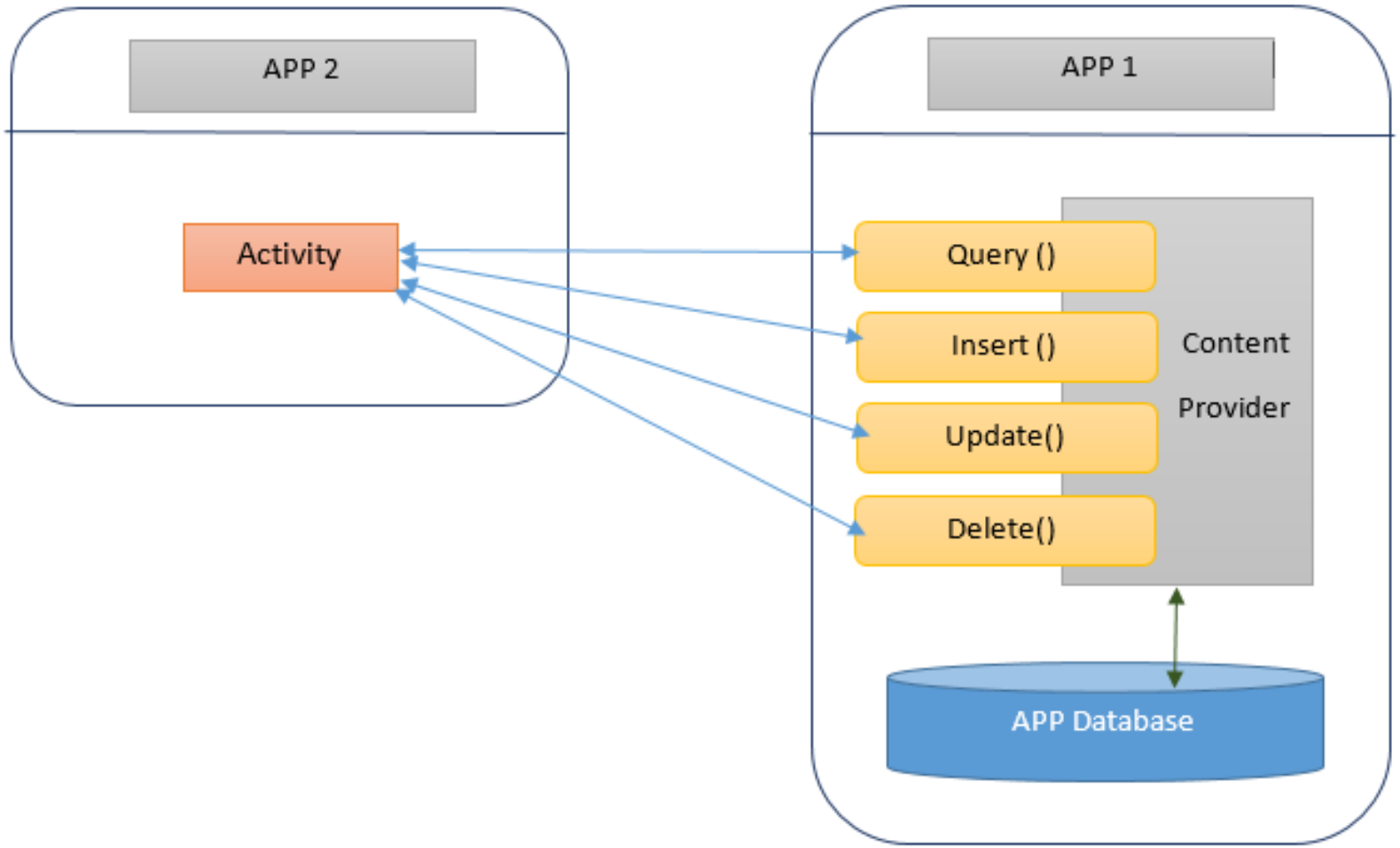




Content Providers

An interface to application data

- Content Provider facilitates APP2 access APP1 data



Content Providers

Basics

In the case of our lab, a content provider:

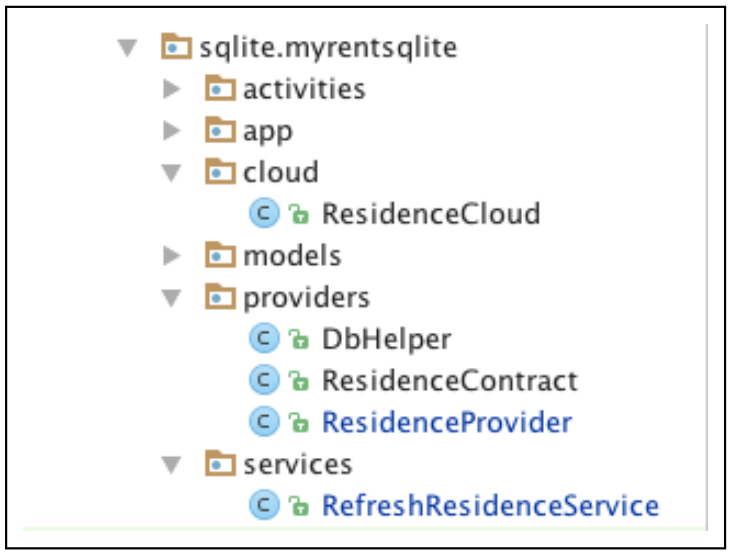
- Manages access to MyRent data.
- Intended for use by other apps.
- Presents data as tables similar to those in SQLite dbase.

Content Providers

Refactored packaging and new classes

Additional classes:

- ResidenceContract: constants
- ResidenceProvider
- ResidenceCloud: simulation
- RefreshResidenceService



Content Providers

Implementation

- Best practice: store constants in separate class

```
package sqlite.myrentsqlite.providers;

public class ResidenceContract
{
  static final String TAG = "ResidenceContract";
  static final String DATABASE_NAME = "residences.db";
  ...
  public class Column
  {
    public static final String ID = BaseColumns._ID;
    public static final String UUID = "uuid";
    public static final String GEOLOCATION = "geolocation";
    public static final String DATE = "date";
    ...
  }
}
```

Content Providers

Implementation

- Refactor DbHelper.
- CRUD methods moved to ContentProvider.

```
public class DbHelper extends SQLiteOpenHelper
{
  static final String TAG = "DbHelper";

  Context context;

  public DbHelper(Context context) {...}

  @Override
  public void onCreate(SQLiteDatabase db) {...}

  /**...*/
  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {...}
}
```

Content Providers

Implementation

- CRUD methods moved to ContentProvider.

```
public class ResidenceProvider extends ContentProvider
{
  private static final String TAG = "ResidenceProvider";
  private DbHelper dbHelper;
  private static final UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);

  static {...}

  @Override
  public boolean onCreate() {...}
  /**...*/
  @{...}
  public Cursor query(Uri uri, String[] projection, String selection,
                      String[] selectionArgs, String sortOrder) {...}
  @{...}
  public String getType(Uri uri) { return null; }
  @{...}
  public Uri insert(Uri uri, ContentValues values) {...}
  @Override
  public int delete(Uri uri, String selection, String[] selectionArgs) {...}
  @Override
  public int update(Uri uri, ContentValues values, String selection,
                    String[] selectionArgs) {...}
}
```

Content Providers

Implementation

- Create the table in *DbHelper.onCreate*

```
@Override
public void onCreate(SQLiteDatabase db) {

  db.execSQL("CREATE TABLE "
      + ResidenceContract.TABLE_RESIDENCES + " ("
      + ResidenceContract.Column.ID + " INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,"
      + ResidenceContract.Column.UUID + " TEXT,"
      + ResidenceContract.Column.GEOLOCATION + " TEXT,"
      + ResidenceContract.Column.DATE + " TEXT,"
      + ResidenceContract.Column.RENTED + " TEXT,"
      + ResidenceContract.Column.TENANT + " TEXT,"
      + ResidenceContract.Column.ZOOM + " TEXT,"
      + ResidenceContract.Column.PHOTO + " TEXT"
      + ");");
}
```

Providers

Manifest

- Requirement: declare provider in manifest

```
<provider
  android:name="sqlite.myrentsqlite.providers.ResidenceProvider"
  android:authorities="sqlite.myrentsqlite.providers.ResidenceProvider"
  android:exported="true"/>
```

```
  // This attribute necessary to expose database to other apps.
  android:exported="true"
```

Content Providers

Implementation

- Simulate the cloud.

```
public class ResidenceCloud
{
  public static Residence residence() {
    return new Residence("52.4444,-7.187162", true, "Barney Gumble", 12.0, "photo1.jpeg");
  }

  public static List<Residence> residences() {
    ArrayList<Residence> list = new ArrayList<>();

    list.add(new Residence("52.4444,-7.187162", true, "Barney Gumble", 12.0, "photo1.jpeg"));
    list.add(new Residence("52.3333,-7.187162", true, "Ned Flanders", 16.0, "photo2.jpeg"));

    return list;
  }

}
```

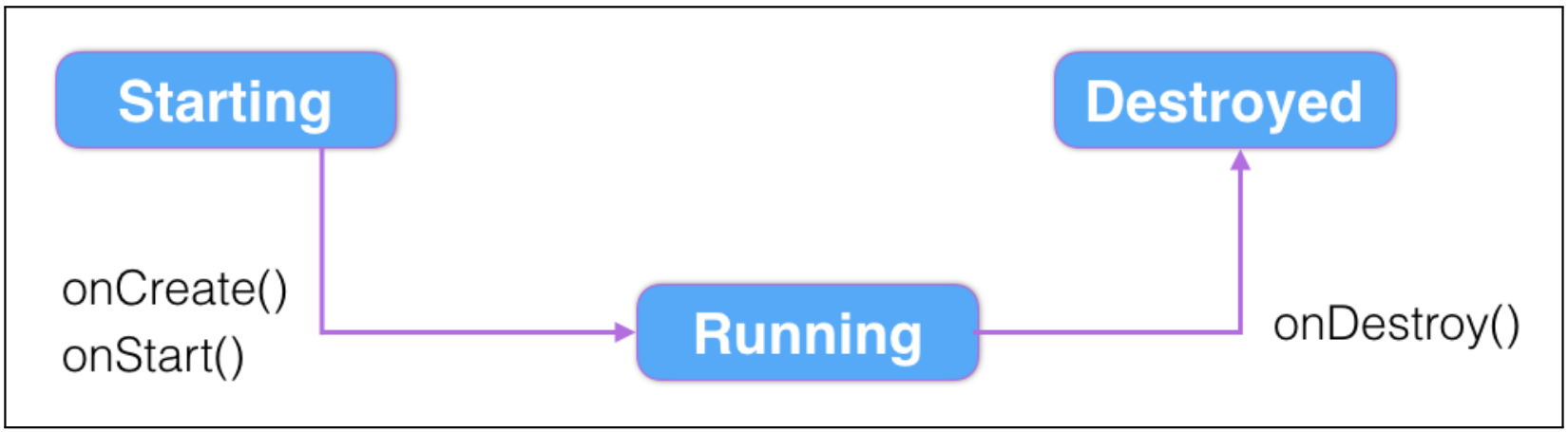
Services

An introduction - dedicated topic towards end course

- Fundamental building blocks of Android
- No user interface
- Block of code running in background
- Independent of activities
- Start & stop - come & go

Services

Have well-defined lifecycle



Services

Manifest

- Requirement: declare service in manifest
- Otherwise will not be called
- No warning (or hard to detect if exists)

```
<service android:name="sqlite.myrentsqlite.services.RefreshResidenceService"/>
```

Services

RefreshResidenceService a subclass of IntentService

```
public class RefreshResidenceService extends IntentService {

  public RefreshResidenceService() {
    super("RefreshResidenceService");
  }
  public RefreshResidenceService(String name) {
    super(name);
  }

  @Override
  protected void onHandleIntent(Intent intent) {
    String value = intent.getStringExtra(REFRESH);
    switch (value) {
      ...
  }
}
```

Services

Starting a service

- **RefreshResidenceService** subclass **IntentService**
- **startService** starts service
- Intent message determines service functionality invoked

```
/**
 * Start a RefreshResidence service, passing the intent message ADD_RESIDENCE
 * This determines what functionality is invoked in the service.
 */
private void addResidence() {
  Intent intent = new Intent(getBaseContext(), RefreshResidenceService.class);
  intent.putExtra(RefreshResidenceService.REFRESH, RefreshResidenceService.ADD_RESIDENCE);
  startService(intent);
}
```

```
putExtra(RefreshResidenceService.REFRESH,
    RefreshResidenceService.ADD_RESIDENCE);
```

Services

IntentService

- *IntentService.onHandleIntent* runs on worker thread.
- Service auto stops when work done.

```
// This method runs on worker thread
@Override
protected void onHandleIntent(Intent intent) {
  String value = intent.getStringExtra(REFRESH);
  switch (value) {
      case ADD_RESIDENCE:
        addResidence(new Residence());
        break;
        ...
    }
}
```

Services

IntentService

- *IntentService.onStartCommand* runs on UI thread.

```
// This method runs on UI thread
int onStartCommand (Intent intent, int flags, int startId)
{
    String value = intent.getStringExtra(REFRESH);
    switch (value) {
    ...
}
```

ContentProvider

How to create ResidenceProvider

- ResidenceProvider subclasses abstract class ContentProvider.
- Implement methods, example insert(), update(), delete().
- Declare Uri CONTENT_URI in ResidenceContract.
- Declare content provider in manifest.

ContentProvider

Uniform Resource Identifier (URI)

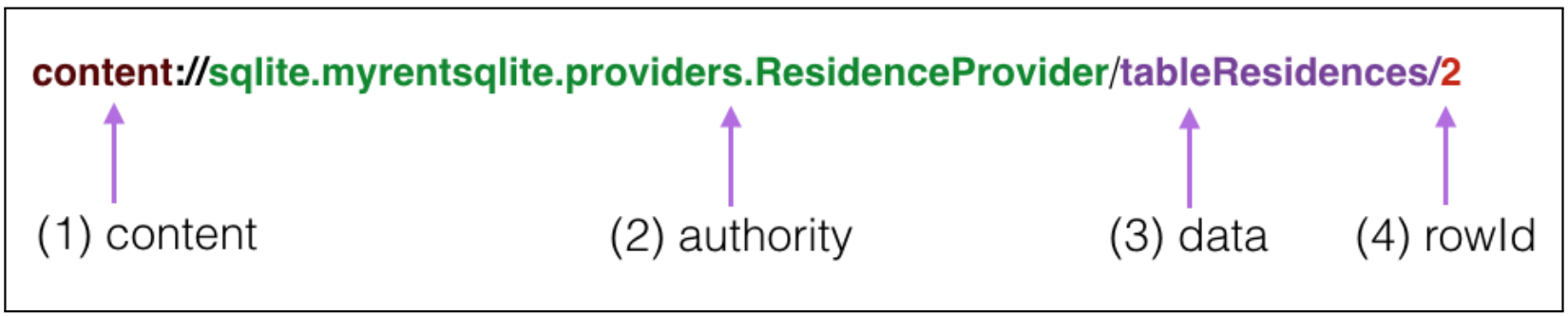
- Objects within app share same address space.
- May refer to each other using variable names.
- Other app objects do not recognize other address spaces.
- URI used to locate content provider.

ContentProvider URI

Defined in ResidenceContract

URI Components:

- (1) : Mandatory component
- (2) : Defined in ResidenceContract
  - conventionally name provider class
- (3) : The data set
- (4) : Record selected (optional)



Content Providers

ContentProviderTest application

A simple test app to read list residences

- MyRentSQLite: generate the SQLite database.
- In test app use data URI to obtain **cursor**.
  - Cursor is set rows + pointer to access these.
  - Facilitates traversal (iterating) rows data.
- Use cursor to initialize local Residence model object.

MyRentSQLite app

Insert (add) Residence record

Approach differs on integration content provider:

- Initiate add residence from gui.
- Starts refresh residence service.
- Creates ContentValues object.
- Populates object with Residence state.
- Uses URI to to identify target of database write.

MyRentSQLite app

Insert (add) Residence record - starts refresh residence service

```
/**
 * Start a RefreshResidence service, passing the intent message
       ADD_RESIDENCE
 * This determines what functionality is invoked in the service.
 */
private void addResidence() {
  Intent intent =
  new Intent(getBaseContext(), RefreshResidenceService.class);
  intent.putExtra(RefreshResidenceService.REFRESH,
                  RefreshResidenceService.ADD_RESIDENCE);
  startService(intent);
}
```

MyRentSQLite app

Insert (add) Residence record - creates ContentValues object in RefreshResidenceService

```
private void addResidence(Residence residence) {
  ContentValues values = new ContentValues();
  ..
  ..
}
```

MyRentSQLite app

Insert (add) Residence record - populates ContentValues with Residence object state

```
private void addResidence(Residence residence) {
  ContentValues values = new ContentValues();

  values.put(ResidenceContract.Column.UUID, residence.uuid.toString());
  values.put(ResidenceContract.Column.GEOLOCATION, residence.geolocation);
  ..
}
```

MyRentSQLite app

Insert (add) Residence record - uses URI to to identify target of database write

```
Uri uri = getContentResolver()
        .insert(ResidenceContract.CONTENT_URI, values);
```

Content Providers

ContentProviderTest application

```
List<Residence> residences = new ArrayList<Residence>();
try {
 Cursor cursor = getContentResolver()
                 .query(ResidenceContract.CONTENT_URI, null, null, null, null);
  if (cursor.moveToFirst()) {
      int columnIndex = 1; // skip column 0, the primary key (id)
      do {
        Residence residence = new Residence();

        residence.uuid = UUID.fromString(cursor.getString(columnIndex++));
        residence.geolocation = cursor.getString(columnIndex++);
        ..
        ..
        columnIndex = 1; // reset to 1 because skipping primary key

        residences.add(residence);
      } while (cursor.moveToNext());
    }
    cursor.close();
  } catch (Exception e){}
```

Content Providers

Summary

- Content provider exposes data to other apps.
- SQLite package and file structure best practice.
- Android service lifecycle, how to start and stop.
- Exercises service functionality on worker thread.
- Use of URI to identify centrally located data.
- Demo use test app to access MyRentSQLite generated data.

References

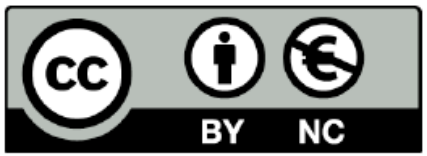
Services

1. Official documentation: Services

`https://developer.android.com/guide/components/services.html` [Accessed 2016-10-25]

2. Official documentation: Content Provider

`https://developer.android.com/guide/topics/providers/content-providers.html` [Accessed 2016-10-26]



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit