

# Media DB와 카메라 활용



# 목차

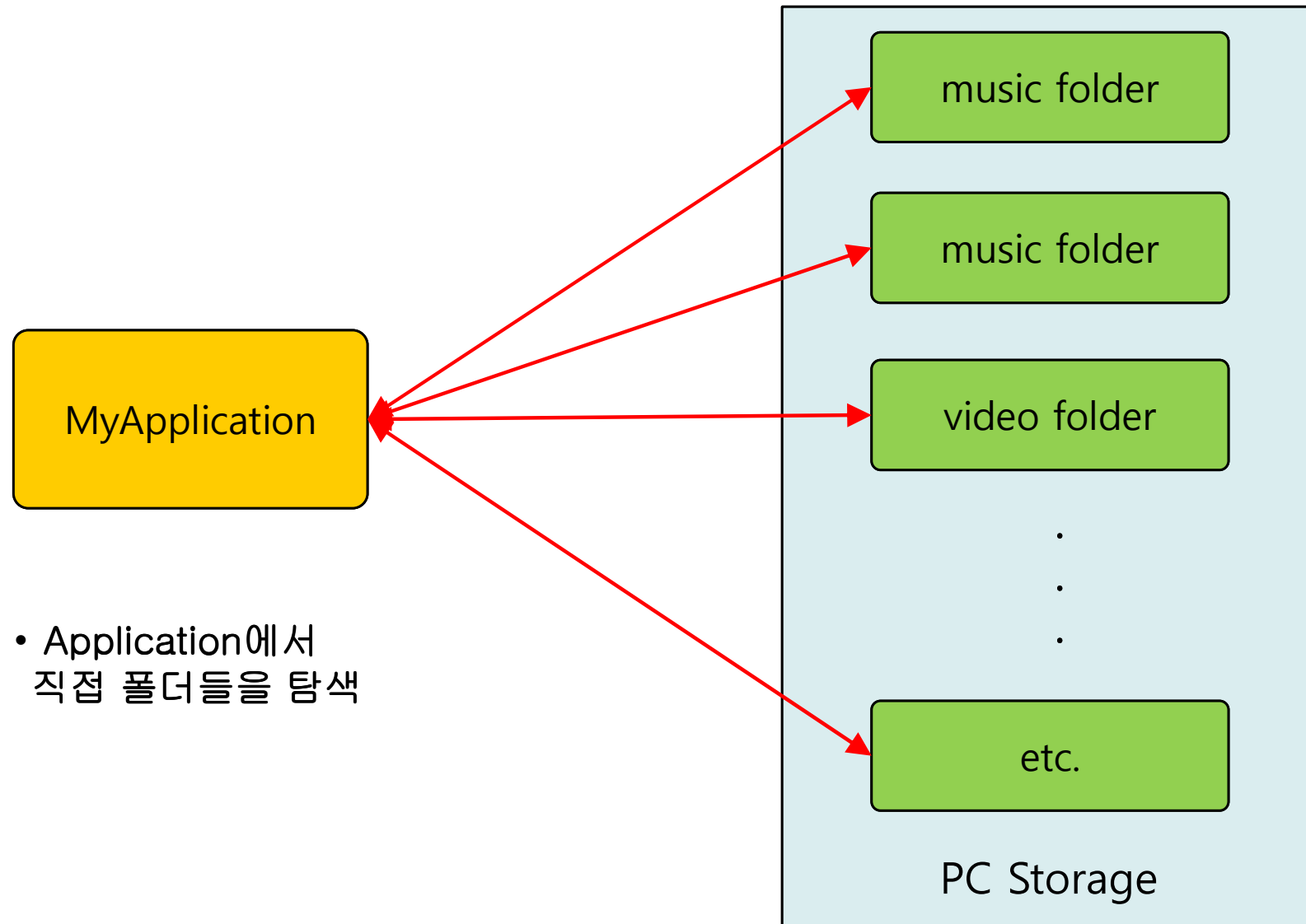
---

 미디어 데이터베이스

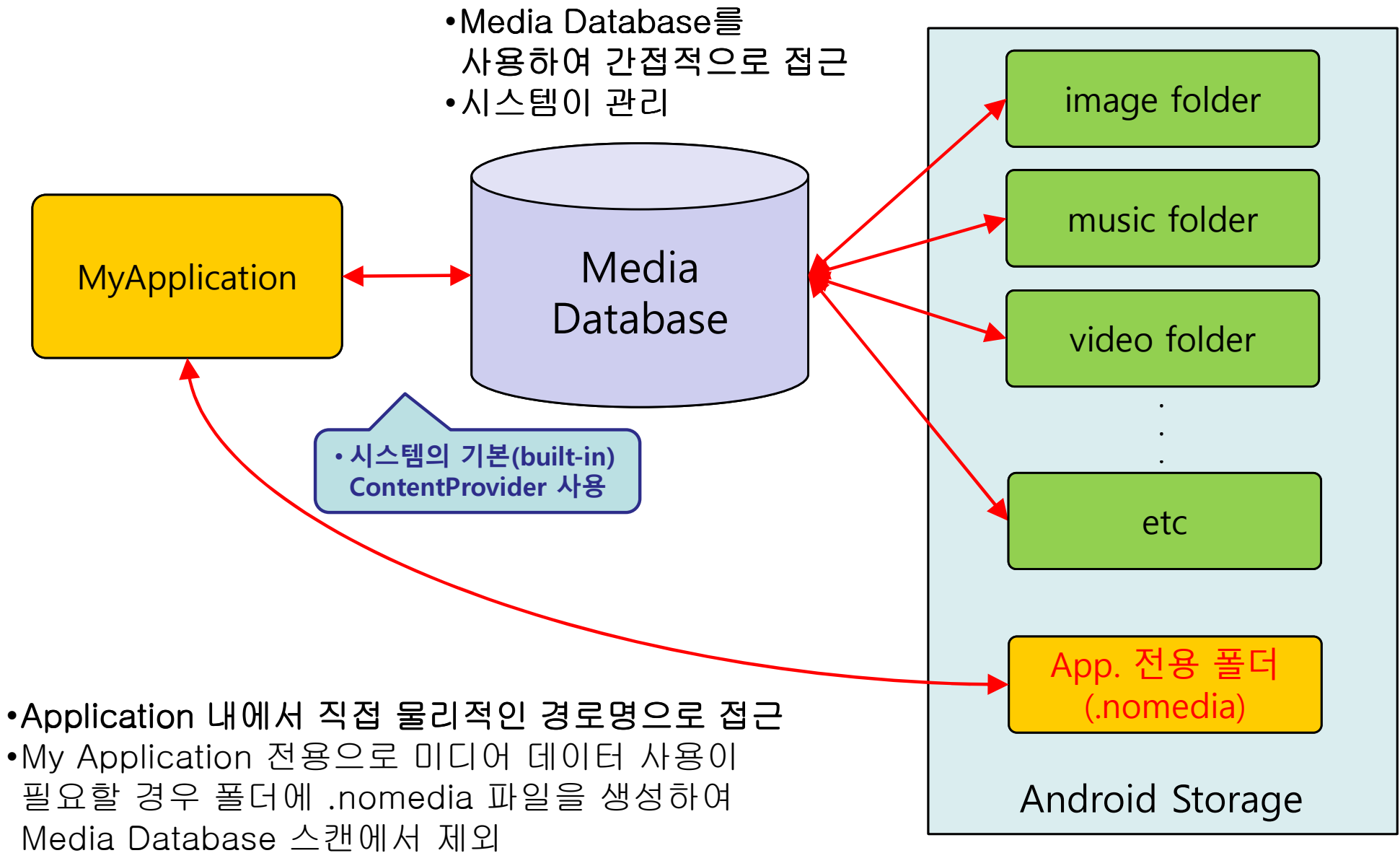
 ContentProvider 개요

 미디어 데이터베이스 검색

 카메라 활용



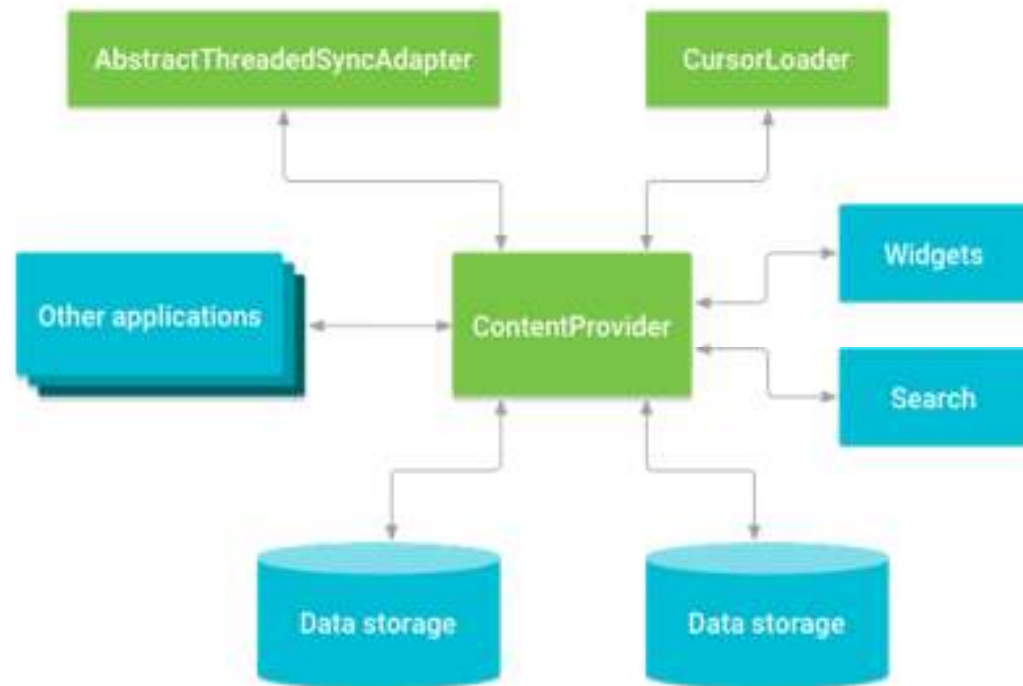
# 미디어 DB - 안드로이드 미디어 파일 접근



# ContentProvider 개요

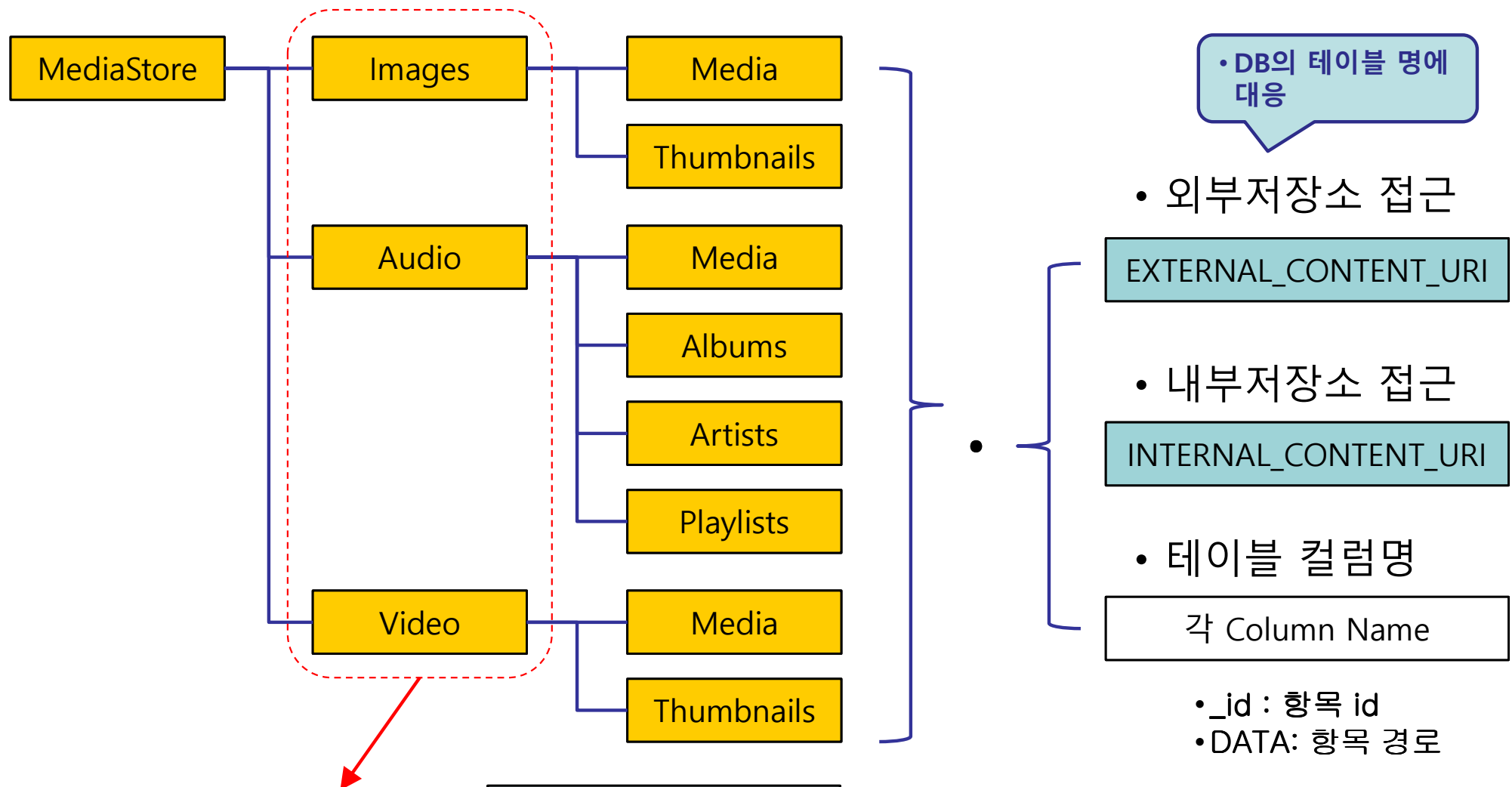
## 주요 자료 저장소의 접근을 관리

- ◆ 다른 앱에 자료를 제공하는 용도로 사용
- ◆ 일관성 있고 표준적인 데이터 접근 인터페이스 제공
- ◆ 프로세스 간 통신과 안전한 데이터 접근 제어
- ◆ ContentResolver 를 사용하여 접근



• 출처: <https://developer.android.com/guide/topics/providers/content-provider-basics>

# Media Database 접근 URI 및 컬럼명



- 각 미디어 별로

[Type]Columns.컬럼명

ImageColumns, AudioColumns, VideoColumns 존재

예) 이미지파일 전체경로명: MediaStore.Images.ImageColumns.DATA

→ /storage/emulated/0/DCIM/Camera/image\_file.jpg

# Media Database 검색

## ContentResolver

- ◆ Media의 목록을 관리하는 콘텐츠 관리 클래스
- ◆ SQLite의 Database 역할 – query 메소드 사용 동일
- ◆ 외부저장소의 이미지를 가져오고자 할 경우 URI (≡테이블명)
  - API Level 29 이하 Permission: `READ_EXTERNAL_STORAGE` `WRITE_EXTERNAL_STORAGE`
  - API Level 30 이상 추가 Permission: `READ_MEDIA_IMAGES`

```
MediaStore.Images.Media.EXTERNAL_CONTENT_URI
```

- ◆ 특정 파일형식 이미지만 가져오고자 할 경우 selection
  - Jpg: image/jpeg, png: image/png

```
MediaStore.Images.Media.MIME_TYPE + "=image/jpeg"
```

- ◆ 특정 이미지를 가져오고자 할 경우 → URI 확인
  - 예: 외부 메모리에 저장한 이미지 id 1234 항목의 URI

```
val imageUri = ContentUris.withAppendedId (MediaStore.Images.Media.EXTERNAL_CONTENT_URI, 1234)
Log.d(TAG, "${imageUri}")
```

content://media/external/images/media/1234    // 출력 결과

# Media Database 검색 1

## 외부저장소 저장 image 검색의 예

```
private fun searchImages() : List<MediaDto> {
    val imageUri = MediaStore.Images.Media.EXTERNAL_CONTENT_URI // 외부저장소 이미지 대상

    val projection = arrayOf(
        MediaStore.Images.Media._ID, // ID
        MediaStore.Images.Media.DISPLAY_NAME, // 파일명
        MediaStore.Images.Media.DATA, // 전체경로
    )

    val selection = MediaStore.Images.Media.MIME_TYPE + "=? " // 이미지 유형 지정
    val selectArgs = arrayOf("image/jpeg") // jpg 이미지 (image/png)

    var list = arrayListOf<MediaDto>()

    val cursor : Cursor = applicationContext.contentResolver.query(
        imageUri, null, null, null, null
    ) ?: return list

    with (cursor) {
        while (moveToNext()) {
            val id = getInt(getColumnIndex(MediaStore.Images.Media._ID))
            val fileName = getString(getColumnIndex(MediaStore.Images.Media.DISPLAY_NAME))
            val path = getString(getColumnIndex(MediaStore.Images.Media.DATA))
            list.add( MediaDto(id, fileName, path) )
        }
    }
    return list
}
```

• MediaDB 대상  
query 수행  
• 전체 이미지



# Media Database 검색 2

## 외부저장소의 특정 이미지 가져오기

```
val id = // 이미지의 ID (Long 타입)
```

```
if (id != null) {
```

```
    val uri = ContentUris.withAppendedId(
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
        id
    )
```

```
    mainBinding.imageView.setImageURI(uri)
```

```
}
```

• 외부 저장소의 특정 id 를  
가진 항목에 대한 URI 생성

# 카메라 활용 앱

## 📷 카메라 기능 직접 개발

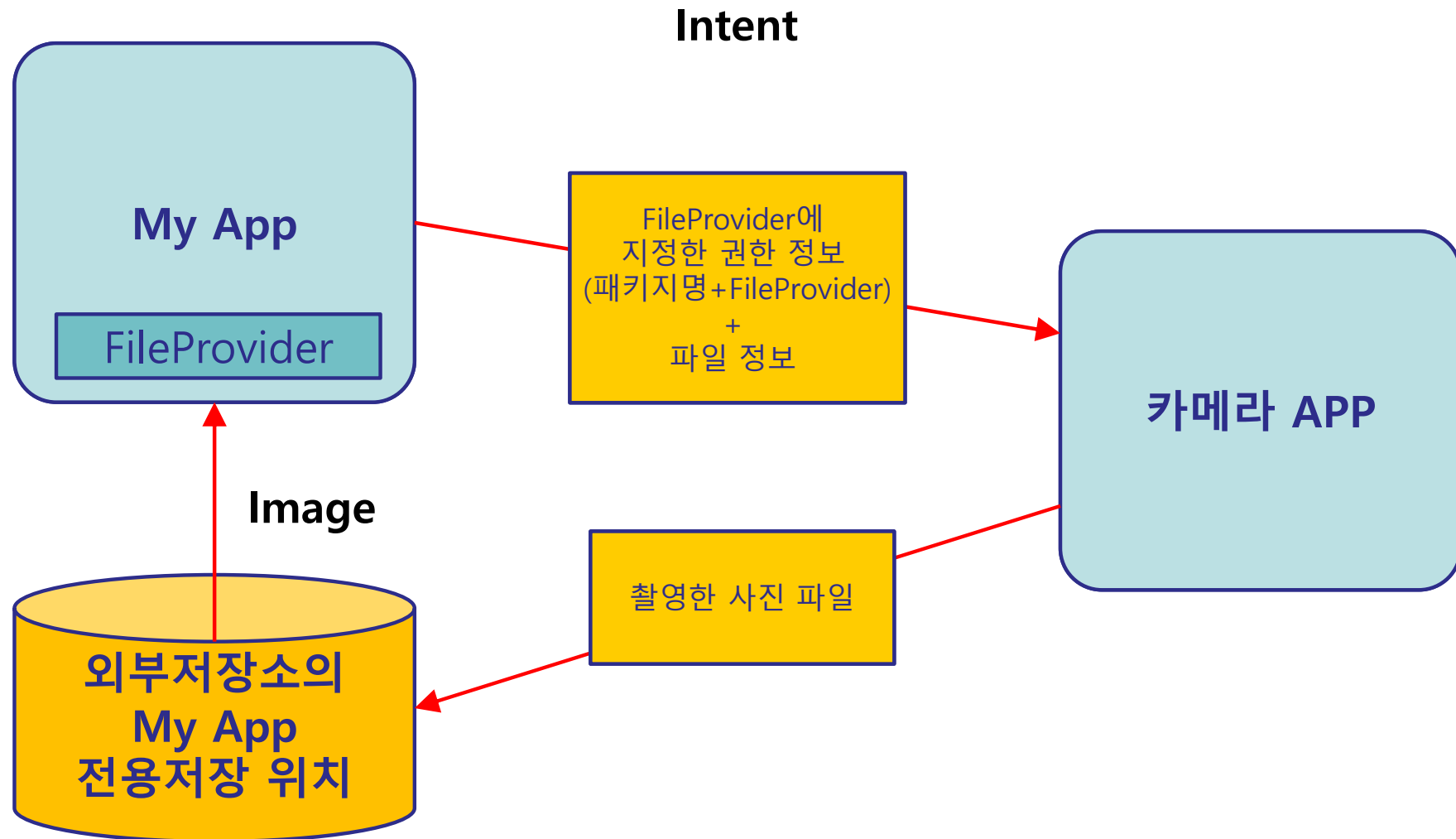
- ◆ Camera 클래스를 활용하여 카메라 기능을 앱에 포함

## 📷 카메라 앱 호출

- ◆ 기본 내장되어 있는 카메라 앱을 활용하여 필요한 이미지만 획득하는 방식
- ◆ 카메라 필요를 앱 설치 시 확인할 수 있도록 지정
- ◆ android:required="false" 로 할 경우 개발자가 앱 내부에서 카메라 유무에 따른 처리를 직접 구현

```
<uses-feature android:name="android.hardware.camera"
    android:required="true" />

<!--API Level 30 이상부터 외부의 앱 접근을 위해 지정-->
<queries>
    <intent>
        <action android:name="android.media.action.IMAGE_CAPTURE" />
    </intent>
</queries>
```



## 저화질의 이미지(썸네일) 필요 시 카메라 앱 호출

```
val REQUEST_THUMBNAIL_CAPTURE = 1
```

```
private fun dispatchTakeThumbnailIntent() {  
    val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)  
    try {  
        startActivityForResult(takePictureIntent, REQUEST_THUMBNAIL_CAPTURE)  
    } catch (e: ActivityNotFoundException) {  
        e.printStackTrace()  
    }  
}
```

• Action의 종류를 이미지 캡처로 지정

• 요구 식별을 위한 상수

## 카메라 앱에서 캡처한 썸네일을 ImageView에 출력

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
    super.onActivityResult(requestCode, resultCode, data)  
    when (requestCode) {  
        REQUEST_THUMBNAIL_CAPTURE -> {  
            if (resultCode == RESULT_OK) {  
                val imageBitmap = data?.extras?.get("data") as Bitmap  
                mainBinding.imageView.setImageBitmap(imageBitmap)  
            }  
        }  
        ...  
    }  
}
```

• 캡처 이미지를 저장하고 있는 Extra 키

## FileProvider 설정

- ◆ ContentProvider의 하위 클래스 : 파일 공유 시 보안을 강화하기 위해 적용 ( file:// → content:// )
- ◆ 외부의 앱이 내 앱 전용폴더의 파일 접근이 필요할 때 사용

```
<application>
    ...
    <provider
        android:name="androidx.core.content.FileProvider"
        [android:authorities="ddwu.com.mobile.cameratest.fileprovider"]
        android:exported="false"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/file_paths"></meta-data>
    </provider>
</application>
```

• AndroidManifest.xml

- 파일 접근 권한명 지정
- 일반적으로 패키지명.fileprovider 로 명명

### • res/xml/file\_paths.xml 추가

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="my_images" path="Android/data/ddwu.com.mobile.cameratest/files/Pictures" />
</paths>
```

- 파일의 실제 저장경로
- 외부저장소의 앱전용 폴더 사용 시의 경로
- 현재 프로젝트의 패키지명으로 변경

# 외부 카메라 앱의 호출 3 – 원본이미지 사용

## 원본 크기 이미지 필요 시 → 파일 저장 필요

### 1. 외부저장소의 공용 폴더에 저장(다른 앱과 사진 공유 시)

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" /> <!-- API Level 28 이하 -->
```

```
val path: File = Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_PICTURES
)
val file : File = File (path, "sample.jpg")
```

실제경로의 예: /storage/emulated/0/Pictures/sample.jpg

### 2. 외부저장소의 앱전용 폴더에 사진 저장

• 앱 삭제 시 해당 경로의 파일도 삭제됨

```
val path: File? = getExternalFilesDir( Environment.DIRECTORY_PICTURES )
val file : File = File (path, "sample.jpg")
```

• 필요에 따라 변경

실제 경로의 예: /storage/emulated/0/Android/data/PACKAGE\_NAME/files/Pictures/sample.jpg

- 앱 전용 경로 저장 시 불필요하나 Android 4.4 미만 버전일 경우만 permission 요청을 지정

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="18"/>
```

# 외부 카메라 앱의 호출 4 – 원본이미지 사용

## 저장할 이미지 파일 명 지정

- ◆ 일반적으로 촬영한 시간으로 파일명 지정

```
lateinit var currentPhotoPath: String // 현재 이미지 파일의 경로 저장
var currentPhotoFileName: String? = null // 현재 이미지 파일명 저장

@Throws(IOException::class)
private fun createImageFile(): File {
    val timeStamp: String = SimpleDateFormat("yyyyMMdd_HH:mm:ss").format(Date())
    val storageDir: File? = getExternalFilesDir(Environment.DIRECTORY_PICTURES)

    val file = File("${storageDir?.path}/${timeStamp}.jpg")

    /*Temp File 생성*/
    // val file = File.createTempFile(
    //     "JPEG_${timeStamp}_", /* prefix */
    //     ".jpg", /* suffix */
    //     storageDir /* directory */
    // )

    currentPhotoFileName = file.name
    currentPhotoPath = file.absolutePath
    return file
}
```

• 지정한 경로에 파일 생성  
• 예제의 경우 외부저장소  
의 앱전용 폴더 내부의  
Pictures 에 저장

• 지정 위치에 임시 파일 생성

• 필요에 따라 파일명 또는  
전체 경로를 저장

# 외부 카메라 앱의 호출 5 – 원본이미지 사용

## 카메라 앱 확인 후 호출

### 호출 시 사진 저장에 사용할 파일 정보 전달

```
private fun dispatchTakePictureIntent() { // 원본 사진 요청
    val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    if (takePictureIntent.resolveActivity(packageManager) != null) { // 카메라 앱 확인
        val photoFile: File? = try { // 고화질 사진을 저장할 파일 생성
            createImageFile()
        } catch (e: IOException) {
            e.printStackTrace()
            null
        }
        if (photoFile != null) {
            val photoURI: Uri = FileProvider.getUriForFile(
                this,
                "ddwu.com.mobile.cameratest.fileprovider",
                photoFile
            )
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI)
            startActivityResult(takePictureIntent, REQUEST_IMAGE_CAPTURE)
        }
    }
}
```

• ACTION을 처리가능한지 확인

• 앞 페이지의 함수  
• 저장할 이미지 파일 생성

• 카메라 앱이 사진 캡처 후 저장할 이미지 파일의 정보를 content:// URI 형태로 생성

• AndroidManifest 에 지정한 Authority 값 지정



# 외부 카메라 앱의 호출 6 – 원본이미지 사용

## 결과 확인

```

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    when (requestCode) {
        REQUEST_THUMBNAIL_CAPTURE -> {
            if (resultCode == RESULT_OK) {
                val imageBitmap = data?.extras?.get("data") as Bitmap
                mainBinding.imageView.setImageBitmap(imageBitmap)
            }
        }
        REQUEST_IMAGE_CAPTURE -> {
            if (resultCode == RESULT_OK) {
                setPic() // 고화질 사진을 지정할 함수 필요
            }
        }
    }
}

```

- FileProvider 에서 지정한 위치의 파일을 읽어들이어 ImageView 에 로딩하는 메소드 구현 필요

# 외부 카메라 앱의 호출 6 – 원본이미지 사용

## 이미지 크기 변경 및 확인 → Glide 로 대체 가능

- ◆ 캡처한 파일은 지정 경로에서 확인 가능
- ◆ 저장한 이미지가 고화질일 경우 ImageView에 표시할 수 있는 적절한 크기로 변환 필요

```
private fun setPic() {  
    // Glide.with(this)  
    // .load(File(currentPhotoPath))  
    // .into(mainBinding.imageView)
```

• Glide 를 사용할 경우 읽어올 파일의 전체 경로를 File 의 생성자로 지정하여 생성 후 load( File(...) ) 에 전달

```
    val targetW: Int = mainBinding.imageView.width  
    val targetH: Int = mainBinding.imageView.height
```

```
    val bmOptions = BitmapFactory.Options().apply {  
        // Get the dimensions of the bitmap  
        inJustDecodeBounds = true
```

• 저장한 파일을 로딩

```
        BitmapFactory.decodeFile(currentPhotoPath, this)
```

```
        val photoW: Int = outWidth  
        val photoH: Int = outHeight
```

```
    val scaleFactor: Int = Math.max(1, Math.min(photoW / targetW, photoH / targetH))
```


```
        inJustDecodeBounds = false  
        inSampleSize = scaleFactor  
        inPurgeable = true
```

• 지정한 크기의 Bitmap으로 decoding

```
    }  
    BitmapFactory.decodeFile(currentPhotoPath, bmOptions)?.also { bitmap ->  
        mainBinding.imageView.setImageBitmap(bitmap)
```

```
}
```

# Gallery 에 사진 추가

 외부저장소의 공용폴더에 사진을 저장하였을 경우에 사용

- ◆ `getExternalStoragePublicDirectory()` 사용 시 다른 앱 (Gallery 등)에 사진이 추가되었음을 통지하여 `MediaDatabase` 에 기록
- ◆ 갤러리 앱 등 다른 앱에서 초기에는 저장한 사진 확인 불가  
→ 해당 기능 수행 후 확인 가능

```
private fun galleryAddPic() {
    Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE).also { mediaScanIntent ->
        val f = File(currentPhotoPath)
        mediaScanIntent.data = Uri.fromFile(f)
        sendBroadcast(mediaScanIntent)
    }
}
```

# 공용 vs. 앱 전용 파일

## 📁 이미지 파일을 접근할 경우

특성 \ 구분	외부저장소 공용 폴더	외부저장소 앱 전용 폴더
저장 위치	/Pictures	예: <b>Android/data/패키지명/files/Pictures</b>
폴더 접근 메소드	getExternalStoragePublicDirectory (Environment.DIRECTORY_PICTURES)	예: getExternalFilesDir (Environment.DIRECTORY_PICTURES)
필요 권한	WRITE_EXTERNAL_STORAGE 필수	(4.4 이하를 지원할 경우에만) WRITE_EXTERNAL_STORAGE
미디어 스캔 방송	필요	불필요
앱 삭제 시	파일 유지	파일 자동 삭제

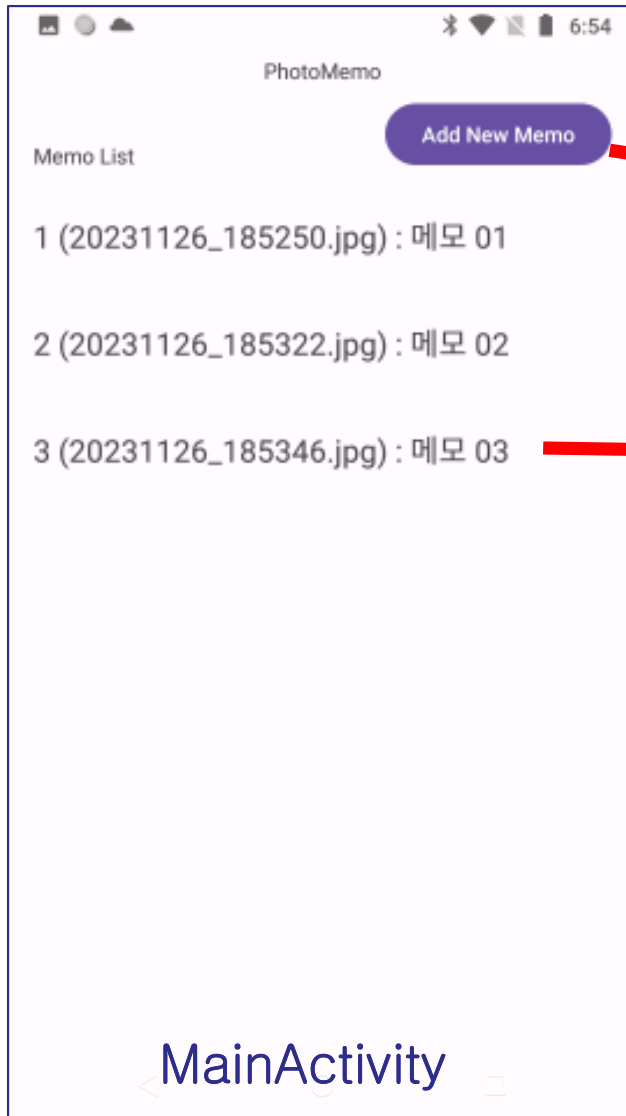
## 📁 예: Glide 로 외부저장소 앱 전용 폴더의 파일 읽기

```

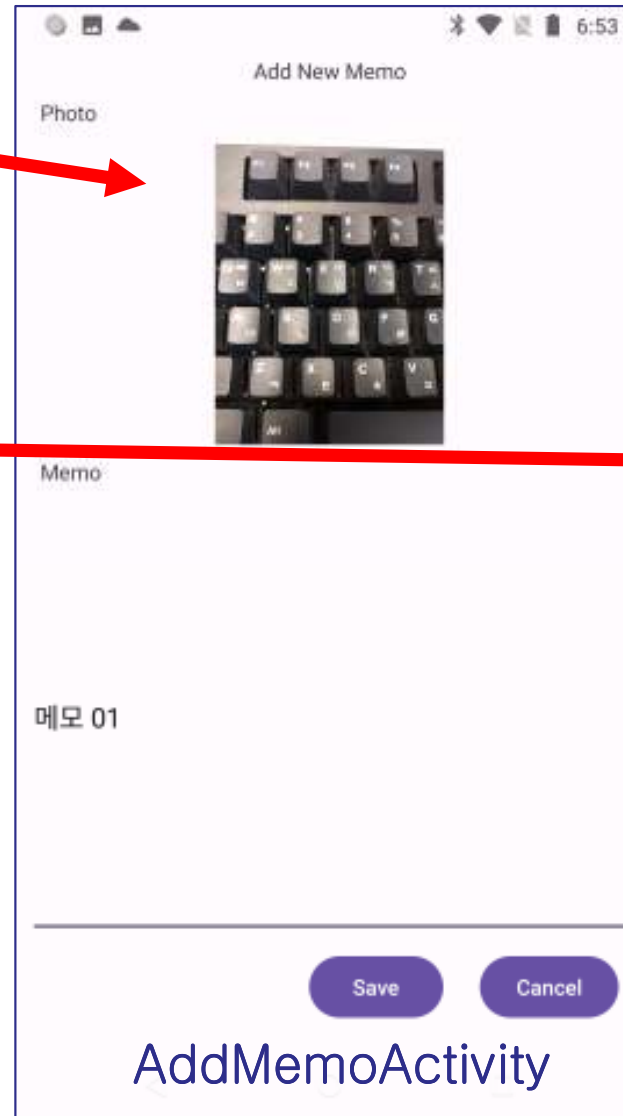
val imageFile = File (getExternalFilesDir(Environment.DIRECTORY_PICTURES), "image.jpg")
// /storage/emulated/0/Android/data/package_names/files/Pictures/image.jpg

Glide.with(this)
    .load(imageFile)
    .into(imageView)
    
```

## •초기화면



## •메모 추가 화면



## •메모 보기 화면



## MainActivity



### Add New Memo 버튼 클릭 시

- ◆ AddMemoActivity 호출

### RecyclerView 항목 클릭 시

- ◆ Intent 에 해당 항목의 DTO 객체 (MemoDto) 저장
- ◆ ShowMemoActivity 호출

## AddMemoActivity



기본 이미지 아이콘을 클릭하면 외부 카메라 앱 실행

- ◆ ImageView에 OnClickListener 연결
- ◆ 외부 카메라 호출 시 원본 사진을 외부저장소의 전용폴더에 저장하는 방법으로 호출
  - `getExternalFilesDir(Environment.DIRECTORY_PICTURES)`
- ◆ 사진을 찍은 본래의 앱으로 돌아오면 찍은 사진을 ImageView에 표시 - Glide 사용 추천

[SAVE] 클릭 시 데이터베이스에 사진의 경로와 메모 내용을 저장

- ◆ ROOM (memoDao) 사용

[Cancel] 클릭 시 파일로 저장하고 있는 사진 삭제

- ◆ `file.delete()` 메소드 사용
- ◆ 액티비티 종료

## ShowMemoActivity



### DTO 전달 받기

- ◆ MainActivity 에서 전달 받은 DTO 객체를 `getSerializableExtra(...)` 로 확인

### 각 view 에 DTO 의 내용 출력

- ◆ Memo 내용
- ◆ Image 의 경우 파일 이름만 DTO에 저장되어 있으므로 지정된 위치와 결합하여 File 생성 후 Glide 에 전달
- ◆ 강의자료 p.g.20 참조

### [Close] 시 액티비티 종료



# 참고

## 공유 저장소 정보

- ◆ <https://developer.android.com/training/data-storage/shared?hl=ko>

## 데이터 및 파일 저장소 개요

- ◆ <https://developer.android.com/training/data-storage?hl=ko>

## 런타임 권한 요청

- ◆ <https://developer.android.com/training/permissions/requesting?hl=ko>

## Android 11의 패키지 공개 상태 - 외부앱 접근

- ◆ <https://developer.android.com/about/versions/11/privacy/package-visibility?hl=ko>