Android Persistent Data Storage:
Introduction

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Data Storage Options on Android

- Android offers several ways to store data
  - SQLite database
  - Files
  - SharedPreferences
Android Persistent Data Storage: Overview of SQLite

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

• Understand what SQLite is & how to use it in Android
Android SQLite

- Android supports SQLite, which provides a relational database for a mobile device
  - i.e., it contains tables (consisting of rows & columns), indexes, etc. that form a “schema”
Android SQLite

- Android supports SQLite, which provides a relational database for a mobile device.
- It’s designed to operate within a small footprint (~350kB) within a single cross-platform disk file.

en.wikipedia.org/wiki/SQLite
Android SQLite

- Android supports SQLite, which provides a relational database for a mobile device.
- It’s designed to operate within a small footprint (<300kB) within a single cross-platform disk file.
- Implements most of SQL92 & supports so-called “ACID” transactions.
  - Atomic, Consistent, Isolated, & Durable.
Android SQLite

- Android supports SQLite, which provides a relational database for a mobile device.
- It’s designed to operate within a small footprint (<300kB) within a single cross-platform disk file.
- Implements most of SQL92 & supports so-called “ACID” transactions.
- Access to an SQLite database typically involves accessing the Android filesystem.
  - Database operations are typically asynchronous since filesystem access can be slow.
    - e.g., access is often made via AsyncTask, AsyncQueryHandler, CursorLoader, etc.

[Link: www.vogella.com/articles/AndroidSQLite/article.html]
SQLiteDatabase

• SQLiteDatabase is the base class for working with a SQLite database in Android
  • It provides the insert(), update(), & delete() methods
(SQLiteDatabase is the base class for working with a SQLite database in Android

- It provides the insert(), update(), & delete() methods
- It also provides the execSQL() method that can execute an SQL statement directly

**SQLiteDatabase**

**Class Overview**

Exposes methods to manage a SQLite database.

SQLiteDatabase has methods to create, delete, execute SQL commands, and perform other common database management tasks.

See the Notepad sample application in the SDK for an example of creating and managing a database.

Database names must be unique within an application, not across all applications.

**Localized Collation - ORDER BY**

In addition to SQLite's default BINARY collator, Android supplies two more, LOCALIZED, which changes with the system's current locale, and UNICODE, which is the Unicode Collation Algorithm and not tailored to the current locale.

SQLiteDatabase

- SQLiteDatabase is the base class for working with a SQLite database in Android
- Queries can be created via rawQuery() & query() methods or via the SQLiteQueryBuilder class

SQLiteQueryBuilder

This is a convenience class that helps build SQL queries to be sent to SQLiteDatabase objects.

dev.android.com/reference/android/database/sqlite/SQLiteQueryBuilder.html
**ContentValues**

- The **ContentValues** object is used by SQLiteDatabase to define key/values.
  - The "key" represents the table column identifier & the "value" represents the content for the table record in this column.

### Class Overview

This class is used to store a set of values that the ContentResolver can process.

```java
ContentValues
extends Object
implements Parcelable
```

- `java.lang.Object`
- `android.content.ContentValues`

---

**developer.android.com/reference/android/content/ContentValues.html**
ContentValues

- The ContentValues object is used by SQLiteDatabase to define key/values.
- ContentValues can be used for inserts & updates of database entries.

```
Class Overview

This class is used to store a set of values that the ContentResolver can process.
```

developer.android.com/reference/android/content/ContentValues.html
**SQLiteOpenHelper**

- Recommended means of using SQLiteDatabase is to subclass the SQLiteOpenHelper class
- In constructor call the super() method of SQLiteOpenHelper, specifying database name & current database version

---

`SQLiteOpenHelper` extends `Object`

```java
java.lang.Object
    android.database.sqlite.SQLiteOpenHelper
```

**Class Overview**

A helper class to manage database creation and version management.

You create a subclass implementing `onCreate(SQLiteDatabase)`, `onUpgrade(SQLiteDatabase, int, int)` and optionally `onOpen(SQLiteDatabase)`, and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state.

This class makes it easy for `ContentProvider` implementations to defer opening and upgrading the database until first use, to avoid blocking application startup with long-running database upgrades.

For an example, see the NotePadProvider class in the NotePad sample application, in the `samples/` directory of the SDK.

**Note:** this class assumes monotonically increasing version numbers for upgrades.
**SQLiteOpenHelper**

- Recommended means of using SQLiteDatabase is to subclass the SQLiteOpenHelper class.
  - In constructor call the super() method of SQLiteOpenHelper, specifying database name & current database version.
  - Override onCreate(), which is called by SQLite if the database does not yet exist.
    - e.g., execute CREATE TABLE command.

---

**Class Overview**

A helper class to manage database creation and version management. You create a subclass implementing `onCreate(SQLiteDatabase)`, `onUpgrade(SQLiteDatabase, int, int)` and optionally `onOpen(SQLiteDatabase)`, and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state.

This class makes it easy for ContentProvider implementations to defer opening and upgrading the database until first use, to avoid blocking application startup with long-running database upgrades.

For an example, see the NotePadProvider class in the NotePad sample application, in the samples/ directory of the SDK.

**Note:** this class assumes monotonically increasing version numbers for upgrades.

Recommended means of using SQLiteDatabase is to subclass the SQLiteOpenHelper class

- In constructor call the super() method of SQLiteOpenHelper, specifying database name & current database version
- Override onCreate(), which is called by SQLite if the database does not yet exist
- Override onUpgrade(), which is called if the database version increases in App code to allow database schema updates

---

**SQLiteOpenHelper**

extends Object

java.lang.Object

*android.database.sqlite.SQLiteOpenHelper*

**Class Overview**

A helper class to manage database creation and version management.

You create a subclass implementing `onCreate(SQLiteDatabase)`, `onUpgrade(SQLiteDatabase, int, int)` and optionally `onOpen(SQLiteDatabase)`, and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state.

This class makes it easy for `ContentProvider` implementations to defer opening and upgrading the database until first use, to avoid blocking application startup with long-running database upgrades.

For an example, see the NotePadProvider class in the NotePad sample application, in the `samples/` directory of the SDK.

**Note:** this class assumes monotonically increasing version numbers for upgrades.

---

developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html
Recommended means of using SQLiteDatabase is to subclass the SQLiteOpenHelper class.

Use SQLiteOpenHelper methods to open & return underlying database

- e.g., getReadableDatabase() & getWritableDatabase() to access an SQLiteDatabase object either in read or write mode, respectively.

---

**SQLiteOpenHelper**

extends Object

java.lang.Object

android.database.sqlite.SQLiteOpenHelper

**Class Overview**

A helper class to manage database creation and version management.

You create a subclass implementing onCreate(SQLiteDatabase), onUpgrade(SQLiteDatabase, int, int) and optionally onOpen(SQLiteDatabase), and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state.

This class makes it easy for ContentProvider implementations to defer opening and upgrading the database until first use, to avoid blocking application startup with long-running database upgrades.

For an example, see the NotePadProvider class in the NotePad sample application, in the samples/ directory of the SDK.

**Note:** this class assumes monotonically increasing version numbers for upgrades.

public class ArtistDatabaseHelper extends SQLiteOpenHelper {
    final private static String CREATE_CMD =
        "CREATE TABLE artists ("
        + "_id" + " INTEGER PRIMARY KEY AUTOINCREMENT, "
        + "name" + " TEXT NOT NULL);"

    public ArtistDatabaseHelper(Context context) {
        super(context, "artists_db", null, 1);
    }

    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_CMD);
    }

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

    ... Support schema evolution

It's common to create an SQLiteOpenHelper subclass for each SQL table

SQL commands to create a table of artists

Give a name to the table

Create the SQL table
Using an SQLite Database Via an Activity

```java
public class DatabaseExampleActivity extends ListActivity {
    final static String[] columns = {"_id", "name"};
    static SQLiteDatabase db = null;

    public void onCreate(Bundle savedInstanceState) {
        ... 
        ArtistDatabaseHelper dbHelper =
            new ArtistDatabaseHelper
                (getActivity().getApplicationContext());
        Make the SQLiteOpenHelper subclass instance
        db = dbHelper.getWritableDatabase();
        insertArtists();
        deleteLadyGaga();
        Cursor c = readArtists();
        displayArtists(c);
    }
    ... 
}
```

SQLiteOpenHelper subclass instance
Create a read/ write database
Perform various operations

SQLiteDatabase is often put in an App singleton to simplify access
Inserting Values Into an SQLite Database

- Method for inserting a row into the database
  
  ```java
  public long insert (String table, String nullColHack, ContentValues values)
  ```

  **Parameters**
  
  - `table` The table to insert the row into
  - `nullColHack` Optional (often null)
  - `values` Map containing initial col values for row; keys are col names

Inserting Values Into an SQLite Database

- Method for inserting a row into the database

```java
public long insert (String table, String nullColHack, ContentValues values)
```

[Diagram of app interfaces and database insertion process]

Inserting Values Into an SQLite Database

```java
private void insertArtists() {
    ContentValues values = new ContentValues();
    "key" represents the table column identifier & the "value"
    represents the content for the table record in this column
    values.put("name", "Lady Gaga");
    db.insert("artists", null, values);
    values.clear();
    values.put("name", "Johnny Cash");
    db.insert("artists", null, values);
    values.clear();
    values.put("name", "Sting");
    db.insert("artists", null, values);
    ...
}
```
Deleting a Row From an SQLite Database

- Method for deleting row(s) from the database

```java
public int delete(String table, String whereClause, String[] whereArgs)
```

**Parameters**

- `table`: the table to delete from
- `whereClause`: optional WHERE clause to apply when deleting
- `whereArgs`: Passing null deletes all rows

Deleting a Row From an SQLite Database

- Method for deleting row(s) from the database

```java
public int delete(String table, String whereClause, String[] whereArgs)
```
Deleting a Row From an SQLite Database

```java
private int deleteLadyGaga() {
    return db.delete("artists",
                    "name" + "=?",
                    new String []
                    {
                        "Lady Gaga"
                    });
}
```

Remove Lady Gaga from the database of artists

Note the use of the “whereArgs”
Querying an SQLite Database

- You can use rawQuery() or a query() on an SQLiteDatabase

```java
public Cursor rawQuery(String sql, String[] selectionArgs)
```
Runs the provided SQL and returns a Cursor over the result set

**Parameters**

- `sql` the SQL query. The SQL string must not be ; terminated
- `selectionArgs` You may include ?’s in where clause in the query, which are replaced by the values from selectionArgs (the values will be bound as Strings)

**Returns**

- A Cursor object, which is positioned before the first entry

Cursors aren’t synchronized, see documentation for details
Querying an SQLite Database

- You can use rawQuery() or a query() on an SQLiteDatabase

    public Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)

query() builds up a SQL SELECT statement from its component parts

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String table</td>
<td>The table name to compile the query against</td>
</tr>
<tr>
<td>int[] columns</td>
<td>A list of which table columns to return (&quot;null&quot; returns all columns)</td>
</tr>
<tr>
<td>String selection</td>
<td>Where-clause filters for the selection of data (null selects all data)</td>
</tr>
<tr>
<td>String[] selectionArgs</td>
<td>You may include ?s in the “selection” where-clause that get replaced by the values from the selectionArgs array</td>
</tr>
<tr>
<td>String[] groupBy</td>
<td>A filter declaring how to group rows (null means rows not grouped)</td>
</tr>
<tr>
<td>String[] having</td>
<td>Filter for the groups (null means no filter)</td>
</tr>
<tr>
<td>String[] orderBy</td>
<td>Table columns used to order the data (null means no ordering)</td>
</tr>
</tbody>
</table>

Returns

A Cursor object, which is positioned before the first entry
Using Query() vs. rawQuery()

• Using rawQuery() on an SQLiteDatabase

```java
private Cursor readArtists() {
    // returns all rows
    return db.rawQuery("SELECT _id, name FROM artists", null);
}
```

developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html
#rawQuery(java.lang.String, java.lang.String[])

Using Query() vs. rawQuery()

- Using rawQuery() on an SQLiteDatabase

```java
private Cursor readArtists() {
    // returns all rows
    return db.rawQuery("SELECT _id, name FROM artists", null);
}
```

- Using query() on an SQLiteDatabase

```java
private Cursor readArtists() {
    // returns all rows
    return db.query("artists", new String [] {"_id", "name"},
    null, null, null, null, null);
}
```

developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html
#query(java.lang.String, java.lang.String[], java.lang.String, java.lang.String[],
java.lang.String, java.lang.String, java.lang.String)
Cursor Iterators

- Query() returns a Cursor Iterator that represents result of a query & points to one row of query result
- This allows buffering of query results efficiently since all data needn’t be loaded into memory

**Cursor**

implements
Closeable

android.database.Cursor

- Known Indirect Subclasses
  AbstractCursor, AbstractWindowedCursor, CrossProcessCursor,
  CrossProcessCursorWrapper, CursorWrapper, MatrixCursor, MergeCursor,
  MockCursor, SQLiteCursor

**Class Overview**

This interface provides random read-write access to the result set returned by a database query.

Cursor implementations are not required to be synchronized so code using a Cursor from multiple threads should perform its own synchronization when using the Cursor.

Implementations should subclass *AbstractCursor*.

Cursor Iterators

- Query() returns a Cursor Iterator that represents result of a query & points to one row of query result
- getCount() returns # of elements of the resulting query
Cursor Iterators

- Query() returns a Cursor Iterator that represents result of a query & points to one row of query result
- getCount() returns # of elements of the resulting query
- moveToFirst() & moveToNext() move between individual data rows

**Cursor**

implements Closeable

android.database.Cursor

- Known Indirect Subclasses
  - AbstractCursor, AbstractWindowedCursor, CrossProcessCursor,
  - CrossProcessCursorWrapper, CursorWrapper, MatrixCursor, MergeCursor,
  - MockCursor, SQLiteDatabase

**Class Overview**

This interface provides random read-write access to the result set returned by a database query.

Cursor implementations are not required to be synchronized so code using a Cursor from multiple threads should perform its own synchronization when using the Cursor.

Implementations should subclass AbstractCursor.
Cursor Iterators

- Query() returns a Cursor Iterator that represents result of a query & points to one row of query result
- getCount() returns # of elements of the resulting query
- moveToFirst() & moveToNext() move between individual data rows
- isAfterLast() checks if the end of the query result has been reached

```
Cursor
implements
Closeable

android.database.Cursor

Known Indirect Subclasses
AbstractCursor, AbstractWindowedCursor, CrossProcessCursor,
CrossProcessCursorWrapper, CursorWrapper, MatrixCursor, MergeCursor,
MockCursor, SQLiteDatabase

Class Overview

This interface provides random read-write access to the result set returned by a database query.

Cursor implementations are not required to be synchronized so code using a Cursor from multiple threads should perform its own synchronization when using the Cursor.

Implementations should subclass AbstractCursor.
```

developer.android.com/reference/android/database/Cursor.html
Cursor Iterators

- Query() returns a Cursor Iterator that represents result of a query & points to one row of query result
- getCount() returns # of elements of the resulting query
- moveToFirst() & moveToNext() move between individual data rows
- isAfterLast() checks if the end of the query result has been reached
- Provides typed get*() methods
  - e.g., getLong(columnIndex) & getString(columnIndex) to access column data for current position of result

Cursor

```
Cursor
implements Closeable

android.database.Cursor

> Known Indirect Subclasses
  AbstractCursor, AbstractWindowedCursor, CrossProcessCursor,
  CrossProcessCursorWrapper, CursorWrapper, MatrixCursor, MergeCursor,
  MockCursor, SQLiteCursor
```

Class Overview

This interface provides random read-write access to the result set returned by a database query.

Cursor implementations are not required to be synchronized so code using a Cursor from multiple threads should perform its own synchronization when using the Cursor.

Implementations should subclass AbstractCursor.
Cursor Iterators

- Query() returns a Cursor Iterator that represents result of a query & points to one row of query result
- getCount() returns # of elements of the resulting query
- moveToFirst() & moveToNext() move between individual data rows
- isAfterLast() checks if the end of the query result has been reached
- Provides typed get*() methods
- Provides getColumnIndexOrThrow(String) to get column index for a column name of table

Cursor

<table>
<thead>
<tr>
<th>implements Closeable</th>
</tr>
</thead>
<tbody>
<tr>
<td>android.database.Cursor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Known Indirect Subclasses</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractCursor, AbstractWindowedCursor, CrossProcessCursor, CrossProcessCursorWrapper, CursorWrapper, MatrixCursor, MergeCursor, MockCursor, SQLiteCursor</td>
</tr>
</tbody>
</table>

Class Overview

This interface provides random read-write access to the result set returned by a database query.

Cursor implementations are not required to be synchronized so code using a Cursor from multiple threads should perform its own synchronization when using the Cursor.

Implementations should subclass AbstractCursor.
Cursor Iterators

- Query() returns a Cursor Iterator that represents result of a query & points to one row of query result
- getCount() returns # of elements of the resulting query
- moveToFirst() & moveToNext() move between individual data rows
- isAfterLast() checks if the end of the query result has been reached
- Provides typed get*() methods
- Provides getColumnIndexOrThrow(String) to get column index for a column name of table
- Must be closed via close()
Displaying an SQLite Database

- The SimpleCursorAdapter class maps the columns to the Views based on the Cursor passed to it.

```java
private void displayArtists (Cursor c) {
    setListAdapter(
        new SimpleCursorAdapter
            (this, R.layout.list_layout, c,
             new String [] {"_id", "name"},
             new int[] { R.id._id, R.id.name }));
}
```

[Developer documentation link](developer.android.com/reference/android/widget/SimpleCursorAdapter.html)
Examining an SQLite Database

- If your App creates a database it is saved by default in a directory file:
  
  ```
  /DATA/data/APP_NAME/databases/FILENAME
  ```

- DATA is the path that the Environment. getDataDirectory() method returns.
- APP_NAME is your app name.
- FILENAME is the name you specify in your application code for the database.
Examining an SQLite Database

- If your App creates a database it is saved by default in a directory file
- You can examine this database with sqlite3
  
  ```
  # adb shell
  # sqlite3 /data/data/com.android.launcher/databases/launcher.db
  ```

```
PS C:\Users\kz> adb shell
# sqlite3 /data/data/com.android.launcher/databases/launcher.db
sqlite3 /data/data/com.android.launcher/databases/launcher.db
SQLite version 3.7.4
Enter "help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
.tables
android_metadata favorites
sqlite> .schema favorites
.schema favorites
CREATE TABLE favorites (_id INTEGER PRIMARY KEY,title TEXT,intent TEXT,container INTEGER,screen INTEGER,cellx INTEGER,celly INTEGER,spanx INTEGER,spany INTEGER,itemType INTEGER,appWidgetId INTEGER NOT NULL DEFAULT -1,isShortcut INTEGER,iconType INTEGER,iconPackage TEXT,iconResource TEXT,icon BLOB,uri TEXT,displayMode INTEGER);
sqlite> .schema android_metadata
.schema android_metadata
CREATE TABLE android_metadata (locale TEXT);
sqlite>```

39
Summary

- SQLite is embedded into every Android device
- Using an SQLite database in Android does not require a setup procedure or administration of the database
Summary

- SQLite is embedded into every Android device
- You only have to define the SQL statements for creating & updating the database
- Afterwards the database is automatically managed for you by the Android platform
Android Content Providers: Introduction

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Overview of Content Providers

- ContentProviders manage access to a central repository of structured data & can make an App’s data available to other Apps

developer.android.com/guide/topics/providers/content-providers.html
Overview of Content Providers

- ContentProviders manage access to a central repository of structured data & can make an App’s data available to other Apps.
- They encapsulate the data & provide mechanisms for defining data security.

developer.android.com/guide/topics/providers/content-providers.html
Overview of Content Providers

- Content Providers manage access to a central repository of structured data & can make an App’s data available to other Apps.
- They encapsulate the data & provide mechanisms for defining data security.
- Content providers are the standard interface that connects data in one process with code running in another process.

[developer.android.com/guide/topics/providers/content-providers.html](https://developer.android.com/guide/topics/providers/content-providers.html)
Overview of Content Providers

- ContentProviders manage access to a central repository of structured data & can make an App’s data available to other Apps
- They encapsulate the data & provide mechanisms for defining data security
- Content providers are the standard interface that connects data in one process with code running in another process
- Content providers support database “CRUD” operations (Create, Read, Update, Delete), where “read” is implemented as “query”

developer.android.com/guide/topics/providers/content-providers.html
Overview of Content Providers

- Content Providers manage access to a central repository of structured data & can make an App’s data available to other Apps.
- They encapsulate the data & provide mechanisms for defining data security.
- Content providers are the standard interface that connects data in one process with code running in another process.
- Content providers support database “CRUD” operations (Create, Read, Update, Delete), where “read” is implemented as “query”.
- Apps can provide Activities that allow users to query & modify the data managed by a provider.

[developer.android.com/guide/topics/providers/content-providers.html]
Example Android ContentProviders

- Android itself includes many Content Providers that manage data for:
  - Browser – bookmarks, history
  - Call log – telephone usage
  - Contacts – contact data
  - MMS/SMS – Stores messages sent & received
  - Media – media database
  - UserDictionary – database for predictive spelling
  - Maps – previous searches
  - YouTube – previous searches
  - Many more

developer.android.com/reference/android/provider/package-summary.html
ContentProvider Data Model

• A content provider typically presents data to external Apps as one or more tables
  • e.g., the tables found in a relational SQL database
ContentProvider Data Model

- A content provider typically presents data to external Apps as one or more tables.
- A row represents an instance of some type of data the provider collects.
- Each column in a row represents an individual piece of data collected for an instance.

One provider in Android is the user dictionary, which stores the spellings of non-standard words that the user wants to keep.

<table>
<thead>
<tr>
<th>word</th>
<th>app id</th>
<th>freq</th>
<th>locale</th>
<th>_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce</td>
<td>user1</td>
<td>100</td>
<td>en_US</td>
<td>1</td>
</tr>
<tr>
<td>precompiler</td>
<td>user14</td>
<td>200</td>
<td>fr_FR</td>
<td>2</td>
</tr>
<tr>
<td>applet</td>
<td>user2</td>
<td>225</td>
<td>fr_CA</td>
<td>3</td>
</tr>
<tr>
<td>const</td>
<td>user1</td>
<td>255</td>
<td>pt_BR</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>user5</td>
<td>100</td>
<td>en_UK</td>
<td>5</td>
</tr>
</tbody>
</table>
Overview of ContentResolver

- ContentProvider never accessed directly, but accessed indirectly via a ContentResolver
  - ContentProvider not created until a ContentResolver tries to access it

developer.android.com/reference/android/content/ContentResolver.html
Overview of ContentResolver

- ContentProvider never accessed directly, but accessed indirectly via a ContentResolver.
- ContentResolvers manage & support ContentProviders.
  - Enables use of ContentProviders across multiple Apps.

developer.android.com/reference/android/content/ContentResolver.html
Overview of ContentResolver

- ContentProvider never accessed directly, but accessed indirectly via a ContentResolver
- ContentResolvers manage & support ContentProviders
  - Enables use of ContentProviders across multiple Apps
- Provides additional services, such as change notification & IPC

[developer.android.com/reference/android/content/ContentResolver.html]
Overview of ContentResolver

- ContentProvider never accessed directly, but accessed indirectly via a ContentResolver
- ContentResolvers manage & support ContentProviders
- Context.getContentResolver() accesses default ContentResolver

```
ContentResolver cr = getContentResolver();
```

[developer.android.com/reference/android/content/ContentResolver.html](http://developer.android.com/reference/android/content/ContentResolver.html)
When you query data via a ContentProvider, you don't communicate with the provider directly.

Instead, you use a ContentResolver object to communicate with the provider.
ContentResolver vs. ContentProvider

- When you query data via a ContentProvider, you don't communicate with the provider directly.
- A call to `getContentResolver().query()` is made.
- This method call invokes `ContentResolver.query()`, not `ContentProvider.query()`.

www.androiddesignpatterns.com/2012/06/content-resolvers-and-content-providers.html
ContentResolver vs. ContentProvider

- When you query data via a ContentProvider, you don't communicate with the provider directly.
- A call to getContentResolver().query() is made.
- When this query method is invoked, the Content Resolver parses the uri argument & extracts its authority.
- The Content Resolver directs the request to the content provider registered with the (unique) authority by calling the Content Provider's query() method.

www.androiddesignpatterns.com/2012/06/content-resolvers-and-content-providers.html
ContentResolver vs. ContentProvider

- When you query data via a ContentProvider, you don't communicate with the provider directly.
- A call to getContentResolver().query() is made.
- When this query method is invoked, the Content Resolver parses the uri argument & extracts its authority.
- When the Content Provider's query() method is invoked, the query is performed & a Cursor is returned (or an exception is thrown).
- The resulting behavior depends on Content Provider's implementation.

www.androiddesignpatterns.com/2012/06/content-resolvers-and-content-providers.html
Content URIs

- Any URI that begins with the `content://` scheme represents a resource served up by a Content Provider.
Content URIs

- Any URI that begins with the content:// scheme represents a resource served up by a Content Provider
- e.g., `content://authority/path/id`
- content - data is managed by a ContentProvider
Content URIs

- Any URI that begins with the content:// scheme represents a resource served up by a Content Provider.
- e.g., content://authority/path/id
  - content - data is managed by a ContentProvider
  - authority - id for the content provider
Content URIs

- Any URI that begins with the `content://` scheme represents a resource served up by a Content Provider
  - e.g., `content://authority/path/id`
  - content - data is managed by a ContentProvider
  - authority - id for the content provider
  - path - 0 or more segments indicating the type of data to access
Content URIs

- Any URI that begins with the content:// scheme represents a resource served up by a Content Provider
  - e.g., content://authority/path/id
    - content - data is managed by a ContentProvider
    - authority – id for the content provider
    - path – 0 or more segments indicating the type of data to access
    - id – specific record being requested
Content URIs

• Any URI that begins with the content:// scheme represents a resource served up by a Content Provider
  
  e.g., content://authority/path/id
  
• ContentProviders are a façade that offers data encapsulation via Content Uri objects used as handles
  
  • The data could be stored in a SQLite database, in flat files, retrieved off a device, be stored on some server accessed over the Internet, etc.
Inserting Data Via ContentResolver

• Use `ContentResolver.insert()` to insert data into a ContentProvider

```java
public final Uri insert(Uri uri, ContentValues values)
```

• Inserts a row into a table at the given URI
• If the content provider supports transactions the insertion will be atomic

**Parameters**

• `uri` – The uri of the table to insert into
• `values` – The initial values for the newly inserted row, where the key is the column name for the field (passing an empty `ContentValues` will create an empty row)

**Returns**

• the URI of the newly created row

[developer.android.com/reference/android/content/ContentProvider.html#insert(android.net.Uri, android.content.ContentValues)](https://developer.android.com/reference/android/content/ContentProvider.html#insert(android.net.Uri, android.content.ContentValues))
Deleting Data Via ContentResolver

Use ContentResolver.delete() to delete data from a ContentProvider

```java
public final int delete(Uri uri, String where, String[] selectionArgs)
```

• Deletes row(s) specified by a content URI. If the content provider supports transactions, the deletion will be atomic

**Parameters**

• `uri` – The uri of the row to delete
• `where` – A filter to apply to rows before deleting, formatted as an SQL WHERE clause (excluding the WHERE itself)
• `selectionArgs` – SQL pattern args

**Returns**

• The number of rows deleted

[developer.android.com/reference/android/content/ContentProvider.html](http://developer.android.com/reference/android/content/ContentProvider.html#delete(android.net.Uri, java.lang.String, java.lang.String[]))
Inserting/Deleting via applyBatch()

- ContentResolver.applyBatch() can insert (& delete) groups of data

```java
public ContentProviderResult[] applyBatch (String authority,
    ArrayList<ContentProviderOperation> operations)
```

- Applies each ContentProviderOperation object & returns array of results
- If all the applications succeed then a ContentProviderResult array with the same number of elements as the operations will be returned

**Parameters**

- `authority` - authority of the ContentProvider to apply this batch
- `operations` - the operations to apply

**Returns**

- the results of the applications

[developer.android.com/reference/android/content/ContentProvider.html#applyBatch(java.util.ArrayList<android.content.ContentProviderOperation>)]
Querying a ContentResolver

- Use ContentResolver. query() to retrieve data
- Returns a Cursor instance for accessing results
- A Cursor is an iterator over a result set

developer.android.com/reference/android/content/ContentProvider.html#query(Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String)
## query() Parameters Compared to SQL Query

<table>
<thead>
<tr>
<th>query() argument</th>
<th>SELECT keyword/parameter</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uri</td>
<td>FROM <em>table_name</em></td>
<td>Uri maps to the table in the provider named <em>table_name</em></td>
</tr>
<tr>
<td>projection</td>
<td><em>col</em>, <em>col</em>, <em>col</em>,…</td>
<td>projection is an array of columns that should be included for each row retrieved</td>
</tr>
<tr>
<td>selection</td>
<td>WHERE <em>col</em> = <em>value</em></td>
<td>selection specifies the criteria for selecting rows</td>
</tr>
<tr>
<td>selectionArgs</td>
<td>No exact equivalent in SQL</td>
<td>Selection arguments replace the ? placeholders in the selection clause</td>
</tr>
<tr>
<td>sortOrder</td>
<td>ORDER BY <em>col</em>, <em>col</em>,…</td>
<td>sortOrder specifies the order in which rows appear in the returned Cursor</td>
</tr>
</tbody>
</table>
Summary

• A SQLite database is private to the App which creates it

• If you want to share data with other App you can use a content provider
Summary

- A SQLite database is private to the App which creates it
- A content provider allows App to access data
  - In most cases this data is stored in an SQLite database
Summary

- A SQLite database is private to the App which creates it.
- A content provider allows App to access data.
- While a content provider can be used within an App to access data, it is typically used to share data with other App.
Summary

- A SQLite database is private to the App which creates it
- A content provider allows App to access data
- While a content provider can be used within an App to access data, it is typically used to share data with other App
- App data is by default private, so a content provider is a convenient to share your data with other application based on a structured interface
Summary

• A SQLite database is private to the App which creates it

• A content provider allows App to access data

• While a content provider can be used within an App to access data, it is typically used to share data with other App

• App data is by default private, so a content provider is a convenient to share your data with other application based on a structured interface

• A content provider must be declared in the AndroidManifest.xml file