SQLite Database System

Dr. Raaid Alubady - *Lecture 10*

1. Introduction

So far, all the techniques you have seen are useful for saving simple sets of data. To manage persistent data. For saving relational data, we can store the data in a ***file*** or in a ***database*** which is much more efficient. For example, if you want to store the test results of all the students in a school, *it is much more efficient to use a database to represent them because* you can use database querying to retrieve the results of specific students. Moreover, *using* databases enables you to enforce data integrity by specifying the relationships between different sets of data.

Android uses the ***SQLite database system***. The database that you create for an application ***is only accessible to itself***; other applications ***will not be able to access it***.

In this lecture, you will learn how to programmatically create a SQLite database in your Android application. For Android, the SQLite database that you create programmatically in an application is always stored in the /data/data/<package_name>/databases folder.

2. Android SQLite Database

SQLite is a popular choice for embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems (such as mobile phones), among others. SQLite has bindings to many programming languages. ***In Android, SQLite is one way of storing application data***. *It is a very lightweight database that comes with Android OS.* In Android, integrating SQLite is a tedious task as it needs writing a lot of boilerplate code to store simple data.

- **Introducing SQLite**: SQLite is a relational database management system (RDBMS). It is well regarded, being:
  - ✓ Open source
  - ✓ Standards-compliant
  - ✓ Lightweight

  It has been implemented as a compact C library that's included as part of the Android software stack. By providing functionality through a library, rather than as a separate process, each database becomes an integrated part of the application that created it. ***This reduces*** external dependencies, minimizes latency, and simplifies transaction locking and synchronization.

  SQLite has a reputation of being extremely reliable and *is the database system of choice for many consumer electronic devices, including several MP3 players, the iPhone, and the iPod Touch.* Lightweight and powerful, *SQLite differs from many conventional database engines by using a loosely typed approach to column definitions.* Rather than requiring column values to conform to a single type, the values in each row for each column are individually typed. As a result, there's no strict type checking when assigning or extracting values from each column within a row.
- **Database packets**: The main package is android.database.sqlite that contains the classes to manage your own databases
- **Database classes**: main class can be used to manage your own databases:

Class	Description
SQLiteOpenHelper	Extend this abstract class to manage a database and its version. We must override the onCreate and onUpgrade methods.
SQLiteDatabase	Includes methods to execute SQL statements
Cursor	Encapsulates a table returned by a select SQL query.

  - ✓ **SQLiteOpenHelper class**: 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.

- ✓ **SQLiteDatabase class**: *SQLiteDatabase* has methods to ***create***, ***delete***, ***execute*** SQL commands, and perform other common database management tasks.
- ✓ **Cursor and Content Values**: *ContentValues* objects are used to insert new rows into database tables (and Content Providers). Each Content Values object represents a single row, as a map of column names to values. Queries in Android are returned as Cursor objects. Rather than extracting and returning a copy of the result values, *Cursors act as pointers to a subset of the underlying data. Cursors are a managed way of controlling your position (row) in the result set of a database query.*

  The Cursor class includes several functions to navigate query results including, but not limited to, the following:

  - *moveToFirst* Moves the cursor to the first row in the query result.
  - *moveToNext* Moves the cursor to the next row.
  - *moveToPrevious* Moves the cursor to the previous row.
  - *getCount* Returns the number of rows in the result set.
  - *getColumnIndexOrThrow* Returns an index for the column with the specified name (throwing an exception if no column exists with that name).
  - *getColumnName* Returns the name of the specified column index.
  - *getColumnNames* Returns a String array of all the column names in the current cursor.
  - *moveToPosition* Moves the cursor to the specified row.
  - *getPosition* Returns the current cursor position.

3. Working with Android Databases

It's good practice to create a helper class to simplify your database interactions. Consider creating a database adapter, which adds an abstraction layer that encapsulates database interactions. It should provide intuitive, strongly typed methods for adding, removing, and updating items. A database adapter should also handle queries and wrap creating, opening, and closing the database. It's often also used as a convenient

location from which to publish static database constants, including table names, column names, and column indexes.

3.1. Opening and Creating Databases Using the SQLiteOpenHelper

SQLiteOpenHelper is an abstract class that wraps up the best practice pattern for creating, opening, and upgrading databases. By implementing and using a *SQLiteOpenHelper*, you hide the logic used to decide if a database needs to be created or upgraded before it's opened. Thus, to extend the *SQLiteOpenHelper* class by overriding the constructor, *onCreate*, and *onUpgrade* methods to handle the creation of a new database and upgrading to a new version, respectively.

To use an implementation of the helper class, create a new instance, passing in the context, database name, current version, and a *CursorFactory* (if you're using one). Call *getReadableDatabase* or *getWriteableDatabase* to open and return a readable/writable instance of the database. Behind the scenes, if the database doesn't exist, the helper executes its *onCreate* handler. If the database version has changed, the *onUpgrade* handler will fire. In both cases, the get<read/write>ableDatabase call will return the existing, newly created, or upgraded database as appropriate.

3.2. Opening and Creating Databases without the SQLiteHelper

You can create and open databases without using the *SQLiteHelper* class with the *openOrCreateDatabase* method on the application Context. Setting up a database is a two-step process. First, call *openOrCreateDatabase* to create the new database. Then, call *execSQL* on the resulting database instance to run the SQL commands that will create your tables and their relationships.

4. Android Database Design Considerations

There are several considerations specific to Android that you should consider when designing your database:

- Files (such as bitmaps or audio files) are not usually stored within database tables. Instead, use a string to store a path to the file, preferably a fully qualified Content Provider URI.

- While not strictly a requirement, it's strongly recommended that all tables include an auto-increment key field, to function as a unique index value for each row. It's worth noting that if you plan to share your table using a *ContentProvider*, this unique ID field is mandatory.

5. Querying Your Database

To execute a query on a database, use the query method on the database object, passing in:

- ✓ An optional *Boolean* that specifies if the result set should contain only unique values.
- ✓ The name of the table to query.
- ✓ A projection, as an array of Strings, that lists the columns to include in the result set.
- ✓ A "*where*" clause that defines the rows to be returned. You can include *?* wildcards that will be replaced by the values stored in the selection argument parameter.
- ✓ An array of selection argument strings that will replace the *?*'s in the "where" clause.
- ✓ A "*group by*" clause that defines how the resulting rows will be grouped.
- ✓ A "*having*" filter that defines which row groups to include if you specified a "group by" clause.
- ✓ A String that describes the order of the returned rows.
- ✓ An optional String that defines a limit to the returned rows.

Good Luck

Regards