Caching Solutions

In this chapter, we will improve user experience by implementing offline support, so when we launch the application, the blog list screen must immediately show cached data.

This lesson explains the different options available on Android to cache the data for offline support.

Preferences

One of the most common ways to persist data on the Android phone is to use SharedPreferences.

The shared preference is basically a key-value store solution for a small amount of data with a built-in memory cache. We already used shared preferences to persist the login state.

Files

When the data is too big for the shared preference, we can always use plain old files.

```
String filename = "myfile";
String fileContents = "Hello world!";
try (FileOutputStream fos = context.openFileOutput(filename, Context.MODE_
PRIVATE)) {
    fos.write(fileContents.toByteArray());
}
```

The problem with files is that we need to convert our data to a format that can be stored and reconstructed later, such as a JSON string, which is not efficient in some cases.

SQLite

Fortunately, in *Android* we can create a local database - *SQLite*.

The `android.database.sqlite` package contains all sorts of API to work with databases - from simple queries to complicated joins. While the *Android SQLite* package contains everything we need, the API is pretty basic and requires us to write a lot of boilerplate code. For these reasons, Google decided to write an abstraction on top of `android.database.sqlite` package to make it easier to work with a database.

This layer is available as a separate library called *Room*. In this chapter we are going to learn how to use the *Room* database to store and access blog articles.