MODULE 4
SQLITE DATABASE IN ANDROID

- SQLite is a open source relational database used to perform database operations on android devices like storing, manipulating or retrieving persistent data from the database.

- **Embedded in android by default. So no need an database setup or administration task.**

- SQLite is **one of the fastest-growing database engines around**, but that's growth in **terms of popularity**, not with its size. The source code for SQLite is in the public domain.
- Supports standard sql syntax, transactions and sql statements.

Why SQLite?

- SQLite does not require a separate server process or system to operate (serverless).
- SQLite comes with zero-configuration, which means no setup or administration needed.
- A complete SQLite database is stored in a single cross-platform disk file.
- SQLite is very small and light weight, less than 400KB fully configured or less than 250KB with optional features omitted.
- SQLite is self-contained, which means no external dependencies.
- SQLite transactions are fully ACID-compliant, allowing safe access from multiple processes or threads.
- SQLite supports most of the query language features found in SQL92 (SQL2) standard.
- SQLite is written in ANSI-C and provides simple and easy-to-use API.
- SQLite is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT).

- Supports 3 datatypes

Text(like string)-for storing datatype string
Integer(like int)-for storing integer primary key
Real(like double)-for storing long values

Simple steps to create a database and handle are as follows.

- **Create** "**SQLiteDatabase**" object.
- Open or **Create** a **database** and **create** a **connection**.
- Perform insert, update or delete operation.
- **Create** a Cursor to display data from the table of the **database**.
- Close the **database connectivity**.

- SQLite is a open source SQL database that stores data to a text file on a device.

Properties to set database
Some database properties must be set after connecting
to the database.
1.setVersion()-sets the database version
2.setLocale()-sets the default locale for the database
3.setLockingEnabled()-enables locking on the database

Step 1:Instantiate “ SQLITEDATABASE “object

Import **android.database.sqlite.SQLiteDatabase namespace** in your application
db=openOrCreateDatabase(String path, int mode, SQLiteDatabase.CursorFactory factory)

This method is used to create/open database. As the name suggests, it will open a database connection if it is already there, otherwise, it will create a new one.

Example,
db=openOrCreateDatabase("XYZ_Database",SQLiteDatabase.CREATE_IF_NECESSARY,null);

ARGUMENTS

String path	Name of the database
Int mode	operating mode. Use 0 or "MODE_PRIVATE" for the default operation, or "CREATE_IF_NECESSARY" if you like to give an option that "if a database is not there, create it"
CursorFactory factory	An optional factory class that is called to instantiate a cursor when a query is called

Step 2: Execute DDL command

db.execSQL(String sql) **throws** SQLException

This command is used to execute a single SQL statement that doesn't return any data means other than SELECT or any other.

```
db.execSQL("Create Table Temp (id Integer, name Text)");
```

In the above example, it takes "CREATE TABLE" statement of SQL. This will create a table of "Integer" & "Text" fields.

Try and Catch block is required while performing this operation. An exception that indicates there was an error with SQL parsing or execution.

Step 3: Create an object of "ContentValues" and Initiate it.

```
ContentValues values=new ContentValues();
```

This class is used to store a set of values. We can also say, it will map ColumnName and relevant ColumnValue.

```
1.values.put("id", eid.getText().toString());
2.values.put("name", ename.getText().toString());
```

String Key	Name of the field as in table. Ex. "id", "name"
String Value	Value to be inserted.

Step 4: Perform Insert Statement.

insert(String table, String nullColumnHack, ContentValues values)

This method returns a long. The row ID of the newly inserted row, or -1 if an error occurred.

Example,

```
db.insert("temp", null, values);
```

String table	Name of table related to the database.
String nullColumnHack	If not set to null, the nullColumnHack parameter provides the name of nullable column name to explicitly insert a NULL into in the case where your *values* is empty.
ContentValues values	This map contains the initial column values for the row.

Step 5: Create Cursor

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

```
Cursor c=db.rawQuery(String sql, String[] selectionArgs)
```

Strign sql	The SQL query
String []selectionArgs	You may include ?s in where clause in the query, which will be replaced by the values from selectionArgs. The values will be bound as Strings.

Example,

```
Cursor c=db.rawQuery("SELECT * FROM temp",null);
```

Methods

moveToFirst	Moves cursor pointer at a first position of a result set
moveToNext	Moves cursor pointer next to the current position.
isAfterLast	Returns false, if the cursor pointer is not atlast position of a result set.

Example,

```
c.moveToFirst();
while(!c.isAfterLast())
{
     //statement
c.moveToNext();
}
```

Step 6: Close Cursor and Close Database connectivity

It is very important to release our connections before closing our activity. It is advisable to release the Database connectivity in "onStop" method. And Cursor connectivity after using it.

Inserting records

We use a ContentValues instance to create a series of table fields to data matching that will be passed to insert() method.

- First we create a ContentValues object to store data to insert and use the put() method to load data.
- Then we use the insert() method to insert the records in SQLite,which contains 3 parameters
  a.Tablename
  b.Null
  c.ContentValues pairs

- This function returns long which holds the primary key of the newly inserted row.

We can create a tableor insert data into the table using execSQL method

```
db.execSQL("CREATE TABLE IF NOT EXISTS ABC Username
VARCHAR,password VARCHAR);");
db.execSQL("insert into values ABC ('admin','admin');");
```

Updating data in the database

The update() method is used to update the records in the database.
The update() supports WHERE syntax
Contains 2 parameters
1.Tablename
2.ContentValues instance

An optional WHERE syntax is allowed, add a string containing where statement as parameter 3.
Use the ? To select an argument substitute in the WHERE statement.

```
updateCountry.put(“country_name”,”united states”);
db.update(“tbl_countries”,updateCountry,”id=?”,new String[]{Long.toString(countryId)});
```

Deleting data from the database

Delete() method has 3 parameters
a.Database name
b.WHERE clause
c.An argument array for the WHERE clause

To delete all the records from a table pass null for the WHERE clause and WHERE clause argument array.

```
public void deleteItem(Item item)
{SQLiteDatabase db=getWritableDatabase();
String whereClause=“id=?”;
String whereArgs[]={item.id.toString()};
db.delete(“items”,whereClause,whereArgs);
}
```

Database – Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table.
We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatbase.rawQuery("Select * from Tablename",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

1	**getColumnCount()** This method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column
4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table

Transactions in SQLite

- A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order

- A transaction is the propagation of one or more changes to the database. For example, if you are creating, updating, or deleting a record from the table, then you are performing transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors

- SQLite is a transactional database that all changes and queries are atomic, consistent, isolated, and durable (ACID).

- SQLite guarantees all the transactions are ACID compliant even if the transaction is interrupted by a program crash, operation system dump, or power failure to the computer.

Atomic: a transaction should be atomic. It means that a change cannot be broken down into smaller ones. When you commit a transaction, either the entire transaction is applied or not.

Consistent: a transaction must ensure to change the database from one valid state to another. When a transaction starts and executes a statement to modify data, the database becomes inconsistent. However, when the transaction is committed or rolled back, it is important that the transaction must keep the database consistent.

Isolation: a pending transaction performed by a session must be isolated from other sessions. When a session starts a transaction and executes the insert or update statement to change the data, these changes are only visible to the current session, not others. On the other hand, the changes committed by other sessions after the transaction started should not be visible to the current session.

Durable: if a transaction is successfully committed, the changes must be permanent in the database regardless of the condition such as power failure or program crash. On the contrary, if the program crashes before the transaction is committed, the change should not persist.

Transaction example

```
SQLiteDatabase db=getDatabase();
db.beginTransaction();
try
{
//insert/update/delete
//insert/update/delete
db.setTransactionSuccessful();
}
finally
{db.endTransaction();
}
```

Call to setTransaction should be in a try block and call to end transaction should be in finally block,so that transaction will be ended even if an unhandled exception is thrown while modifying the database.

BOOKMARKS

The first Android database we are leaving to search is the browser bookmarks.

To access the internal android database,we use the managedQuery() method.

ANDROID includes some helper variables in Browser.Bookmark column to choose column names.

MEDIAPLAYER

ANDROID MEDIAPLAYER uses SQLITE to store media information.
Existing fields are
DATE_ADDED,DATE_MODIFIED,DISPLAY_NAME,MIME_TYPE,SIZE and TITLE.

Chapter 9

TELEPHONY AND MESSAGING

How to send and receive SMS in your Android application.

SMS Telephony

Telephony system is a software framework to provide mobile phones with telephony functionality like voice calls,video call,SMS,MMS,data service,network management and so on.

- Android phones support dialing numbers, receiving calls, sending and receiving text and multimedia messages, and other related telephony services.
- In contrast to other smartphone platforms, all these items are accessible to developers through simple-to-use APIs and built-in applications.
- You can easily leverage Android’s telephony support into your own applications.

Android Telephony architecture works in 4 layers that are :

- **Communication Processor**
- **Radio Interface Layer (RIL)**
- **Framework Services**
- **Applications**

Let us try to understand them briefly one by one :

1. Communication Processor

It is an input/output processor to distribute and collect data from a number of remote terminals. It is a specialized processor designed to communicate with the data communication network.

2. Radio Interface Layer

It is a bridge between the hardware and Android phone framework services. It is a protocol stack for Telephone. It has two main components that are:

RIL Daemon– It starts when the android system starts. It reads the system properties to find a library that is to be used for Vendor RIL.

Vendor RIL– It is also known as RIL Driver. It can be understood as a library that is specific to each modem.

3. Framework Services

The telephony Framework starts and initializes along with the system. All the queries by Application API are directed to RIL using these services.

4. Application

These are the Application UI related to telephony such as Dialer, SMS, MMS, Call tracker, etc. These applications start with the android system boot up. These are tied with framework services of telephony.

Android Telephony Framework consists of two types of packages that are:

1. Internal Telephony Packages: This is generally the used for default telephony app.apk.

2. Open Technology Packages: This is for third-party apps.

Telephony Applications

1.Dialer
2.Contacts
3.Messaging-SMS/MMS
4.Settings app-a.Airplane mode b.Network Selection(auto/manual)c.Call
5.Settings such as call waiting,call forwarding,diverting SIM application toolkit(SAT)
6.Browser
7.CellBroadcastReceiver

In Android, you can use SmsManager API or devices Built-in SMS application to send SMS's.

SmsManager API

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);
```

Built-in SMS application

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

Both need **SEND_SMS permission** in the manifest file

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

RECEIVING SMS

To receive sms,use the onReceive() method of the BroadcastReceiver class.
The android framework sends out system broadcasts of events such as receiving an sms message,containing intents that are meant to be received using a broadcast receiver.

Add a <receiver > to register a broadcast reciever to the manifest file.
We can use an intent filter to let android know that we want to launch a specific class when an sms comes in.

WIFI ACTIVITY

Android allows applications to view the state of the wireless connections at very low level. Application can access almost all the information of a wifi connection.

Android provides **WifiManager** API to manage all aspects of WIFI connectivity. We can instantiate this class by calling **getSystemService** method. Its syntax is given below −

```
WifiManager mainWifiObj;
mainWifiObj = (WifiManager) getSystemService(Context.WIFI_SERVICE);
```

In order to scan a list of wireless networks, you also need to register your BroadcastReceiver. It can be registered using **registerReceiver** method with argument of your receiver class object. Its syntax is given below −

```
class WifiScanReceiver extends BroadcastReceiver
 { public void onReceive(Context c, Intent intent) { } }
WifiScanReceiver wifiReciever = new WifiScanReceiver();
registerReceiver(wifiReciever, new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
```

The wifi scan can be start by calling the **startScan** method of the WifiManager class. This method returns a list of ScanResult objects. You can access any object by calling the **get** method of list. Its syntax is given below −

```
List<ScanResult> wifiScanList = mainWifiObj.getScanResults();
String data = wifiScanList.get(0).toString();
```

Run android app on a real device

Set up your device as follows:

Connect your device to your development machine with a USB cable. If you developed on Windows, you might need to install the appropriate device driver for your device.

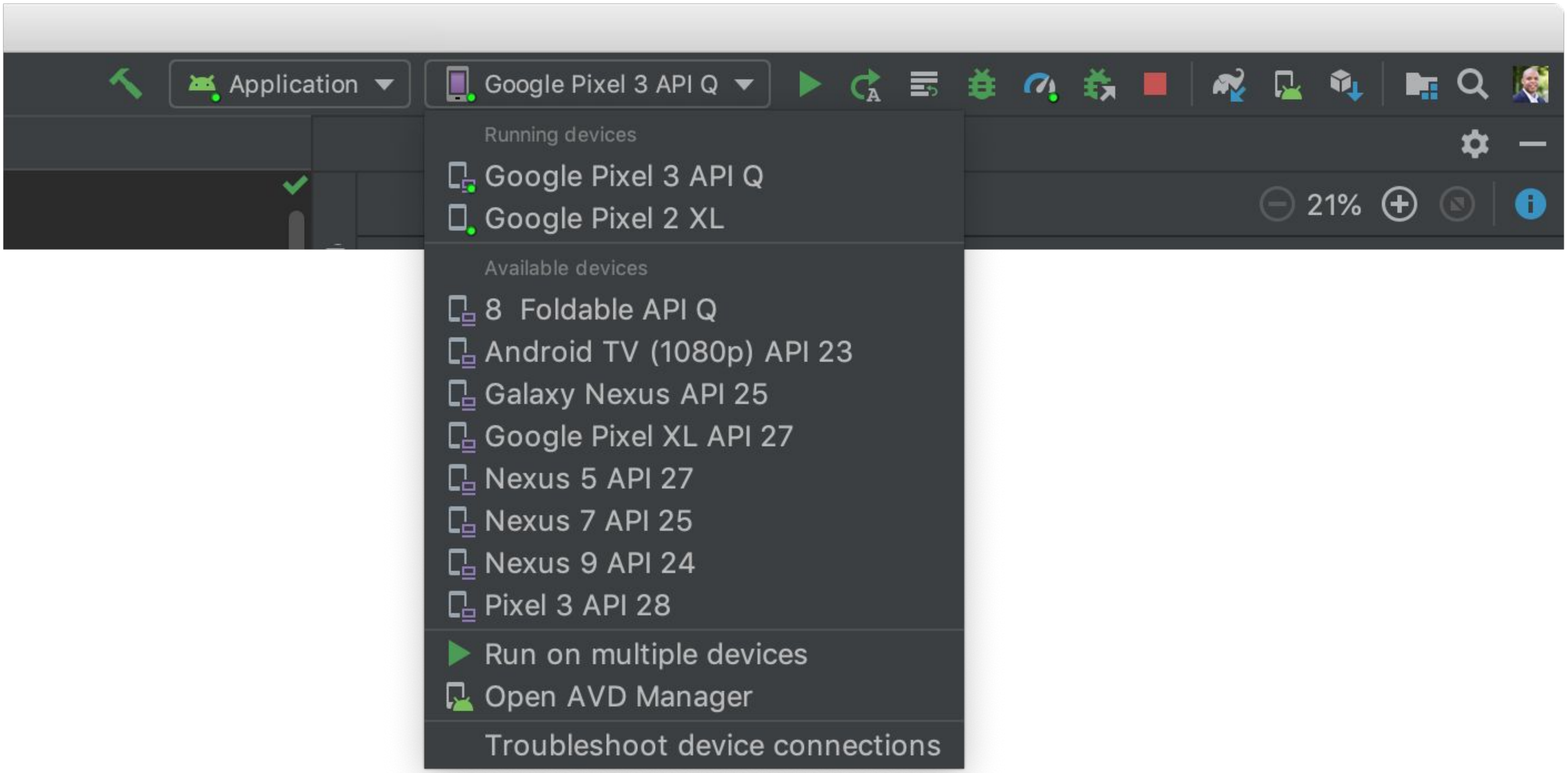
Perform the following steps to enable **USB debugging** in the **Developer options** window:

- Open the **Settings** app.
- If your device uses Android v8.0 or higher, select **System**. Otherwise, proceed to the next step.
- Scroll to the bottom and select **About phone**.
- Scroll to the bottom and tap **Build number** seven times.
- Return to the previous screen, scroll to the bottom, and tap **Developer options**.
- In the **Developer options** window, scroll down to find and enable **USB debugging**.

Run the app on your device as follows:

In Android Studio, select your app from the run/debug configurations drop-down menu in the toolbar.

In the toolbar, select the device that you want to run your app on from the target device drop-down menu.


Application
Google Pixel 3 API Q
Running devices
Google Pixel 3 API Q
Google Pixel 2 XL
Available devices
8 Foldable API Q
Android TV (1080p) API 23
Galaxy Nexus API 25
Google Pixel XL API 27
Nexus 5 API 27
Nexus 7 API 25
Nexus 9 API 24
Pixel 3 API 28
Run on multiple devices
Open AVD Manager
Troubleshoot device connections
21%