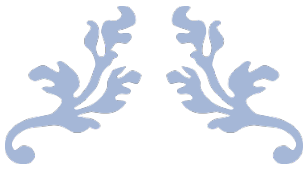


UNIT: 4 DATABASE CONNECTIVITY USING SQLITE AND CONTENT PROVIDER

CS-31 Mobile Application Development in Android using Kotlin

- **ACTIVITY LIFECYCLE IN ANDROID**
- **USING ANDROID DATA AND STORAGE APIS**
- **MANAGING DATA USING SQLITE**
- **SHARING DATA BETWEEN APPLICATIONS WITH CONTENT PROVIDERS**

-: ASSIGNMENT 4 :-

1. EXPLAIN ANDROID LIFE CYCLE/ANDROID ACTIVITY LIFE CYCLE.
2. WHAT IS DATABASE/WHAT IS USE OF DATABASE IN ANDROID.
3. WHAT IS CONTENT PROVIDER/ EXPLAIN CONTENT PROVIDER IN DETAIL
4. EXPLAIN DATABASE MANAGEMENT IN ANDROID.
5. WRITE STEPS TO CREATE AND MANAGE DATABASE IN ANDROID.
6. WRITE STEPS TO CREATE AND MANAGE CONTENT PROVIDER.

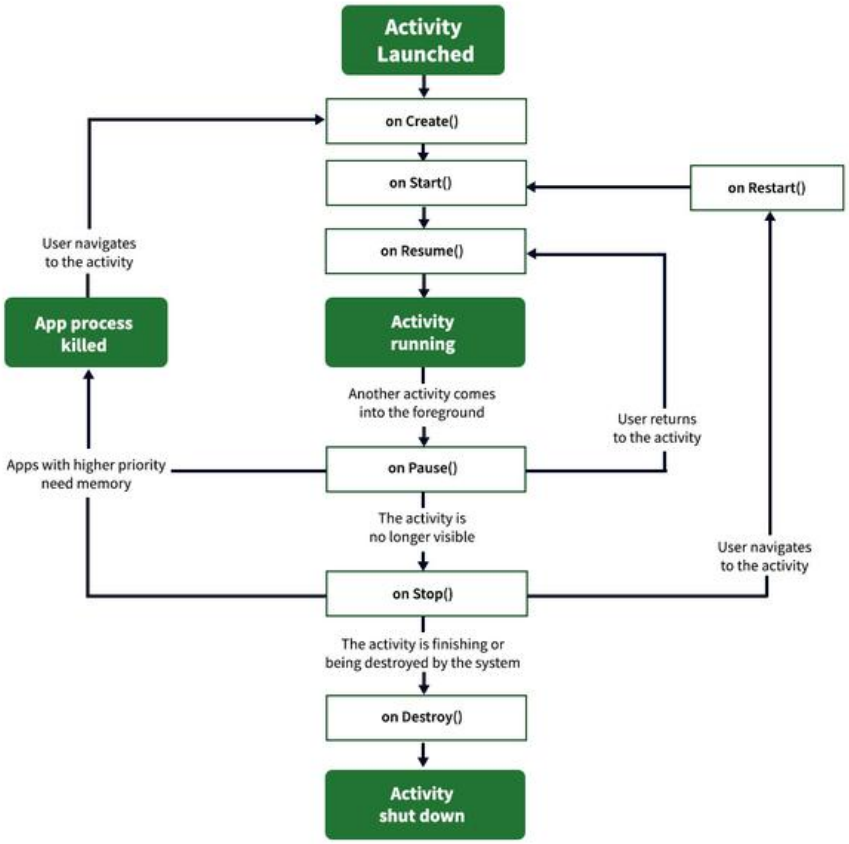
PREPARED BY: PROF. N.K.PANDYA (PHD*, MCA, BCA, BPP)
KAMANI SCIENCE COLLEGE, AMRELI(BCA/BSC)

Activity Lifecycle in Android

In Android, an activity is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities.
Each activity goes through various stages or a lifecycle and is managed by activity stacks. So when a new activity starts, the previous one always remains below it. There are four stages of an activity.

If an activity is in the foreground of the screen i.e at the top of the stack, then it is said to be active or running. This is usually the activity that the user is currently interacting with.

- If an activity has lost focus and a non-full-sized or transparent activity has focused on top of your activity. In such a case either another activity has a higher position in multi-window mode or the activity itself is not focusable in the current window mode. Such activity is completely alive.
- If an activity is completely hidden by another activity, it is stopped or hidden. It still retains all the information, and as its window is hidden thus it will often be killed by the system when memory is needed elsewhere.
- The system can destroy the activity from memory by either asking it to finish or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.
- For each stage, android provides us with a set of 7 methods that have their own significance for each stage in the life cycle. The image shows a path of migration whenever an app switches from one state to another.
- **onCreate()**:It is called when the activity is first created. This is where all the static work is done like creating views, binding data to lists, etc. This method also provides a Bundle containing its previous frozen state, if there was one.
- **onStart()**: It is invoked when the activity is visible to the user. It is followed by onResume() if the activity is invoked from the background. It is also invoked after onCreate() when the activity is first started.
- **onRestart()** :It is invoked after the activity has been stopped and prior to its starting stage and thus is always followed by onStart() when any activity is revived from background to on-screen.
- **onResume() :**It is invoked when the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, with a user interacting with it. Always followed by onPause() when the activity goes into the background or is closed by the user.
- **onPause():** It is invoked when an activity is going into the background but has not yet been killed. It is a counterpart to onResume(). When an activity is launched in front of another activity, this callback will be invoked on the top activity (currently on screen). The activity, under the active activity, will not be created until the active activity's onPause() returns, so it is recommended that heavy processing should not be done in this part.



Activity Lifecycle in Android

- **onStop():**It is invoked when the activity is not visible to the user. It is followed by onRestart() when the activity is revoked from the background, followed by onDestroy() when the activity is closed or finished, and nothing when the activity remains on the background only. Note that this method may never be called, in low memory situations where the system does not have enough memory to keep the activity's process running after its onPause() method is called.
- **onDestroy() :**The final call received before the activity is destroyed. This can happen either because the activity is finishing (when finish() is invoked) or because the system is temporarily destroying this instance of the activity to save space. To distinguish between these scenarios, check it with isFinishing() method.

Example:

```
class MainActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val toast = Toast.makeText(applicationContext,"onCreateCalled", Toast.LENGTH_LONG).show()
        }

    override fun onStart() {
        super.onStart()
        val toast = Toast.makeText(applicationContext, "onStart Called", Toast.LENGTH_LONG).show()
    }
```

```
    override fun onRestart() {
        super.onRestart()
        val toast =Toast.makeText(applicationContext,"onRestartCalled", Toast.LENGTH_LONG).show()
    }

    override fun onPause() {
        super.onPause()
        val toast =Toast.makeText(applicationContext, "onPause Called", Toast.LENGTH_LONG).show()
    }

    override fun onResume() {
        super.onResume()
        val toast=Toast.makeText(applicationContext,"onResumeCalled",Toast.LENGTH_LONG).show()
    }

    override fun onStop() {
        super.onStop()
        val toast = Toast.makeText(applicationContext, "onStop Called", Toast.LENGTH_LONG).show()
    }

    override fun onDestroy() {
        super.onDestroy()
        val toast= Toast.makeText(applicationContext,"onDestroyCalled",Toast.LENGTH_LONG).show()
    }
}
```

Using Android Data and Storage APIs

Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.

Your data storage options are the following:

- **Shared Preferences**
  - Store private primitive data in key-value pairs.
- Internal Storage
  - Store private data on the device memory.
- External Storage
  - Store public data on the shared external storage.
- **SQLite Databases**
  - Store structured data in a private database.
- **Network Connection**
  - Store data on the web with your own network server.
- **Content Providers**
  - Android provides a way for you to expose even your private data to other applications with a **content provider**. A content provider is an optional component that exposes read/write access to your application data, subject to whatever restrictions you want to impose.

Shared Preferences

- One of the most Interesting Data Storage options Android provides its users is Shared Preferences. Shared Preferences is the way in which one can store and retrieve small amounts of primitive data as key/value pairs to a file on the device storage such as String, int, float, Boolean that make up your preferences in an XML file inside the app on the device storage. Shared Preferences can be thought of as a dictionary or a key/value pair. For example, you might have a key being "username" and for the value, you might store the user's username. And then you could retrieve that by its key (here username). You can have a simple shared preference API that you can use to store preferences and pull them back as and when needed. Shared Preferences class provides APIs for reading, writing, and managing this data.
- Shared Preferences are suitable in different situations. For example, when the user's settings need to be saved or to store data that can be used in different activities within the app. As you know, onPause() will always be called before your activity is placed in the background or destroyed, So for the data to be saved persistently, it's preferred to save it in onPause(), which could be restored in onCreate() of the activity. The data stored using shared preferences are kept private within the scope of the application. However, shared preferences are different from that activity's instance state.

Steps to create/use Shared Preferences

Step 1: **Create shared Preference file.**

```
getSharedPreferences(sharedPrefFile,Context.MODE_PRIVATE)
```

- In above code, **this** is referred as context object, **sharedPrefFile** is a name of file usually we use package name "com.nkp.myapp" **Context.MODE_PRIVATE** is mode for the preference file, usually this mode is recommended to increase security.

Step 2: **Create shared Preference editor file.**

```
Val editor:SharedPreferences.Editor =  sharedPreferences.edit()
```

Step 3: Use Editor to add values to Shared Preferences.

- It uses **key and value** pair and each data type have different functions such as putInt(), putString() etc.

Step 4: Apply and commit in shared Preference file.

- Here we need to manually store data into file using editor object editor.apply() , editor.commit() If we don't the date will not be saved to the file.

Steps to retrieve/read/access Shared Preferences

Step 1: **Access/open the previous created shared Preference file**

```
val sharedPreferences: SharedPreferences
sharedPreferences = this.getSharedPreferences(sharedPrefFile,Context.MODE_PRIVATE)
```

- In above code, **this** is referred as context object, **sharedPrefFile** is a name of file usually we use package name "com.nkp.myapp" **Context.MODE_PRIVATE** is mode for the preference file, usually this mode is recommended to increase security. The same code is used to create new file as well.

Step 2: Use sharedPreferences object to access data.

- It uses **key and default value** pair and each data type have different functions such as getInt(), getString() etc.

Managing data using SQLite

- Android comes with an inbuilt implementation of a database package, which is SQLite, an open-source SQL database that stores data in form of text in devices. In this article, we will look at the implementation of Android SQLite in Kotlin.
- SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine. It is the most used database engine in the world.
- It is an in-process library and its code is publicly available.
- It is free for use for any purpose, commercial or private.
- It is basically an embedded SQL database engine.
- Ordinary disk files can be easily read and write by SQLite because it does not have any separate server like SQL.
- The SQLite database file format is cross-platform so that anyone can easily copy a database between 32-bit and 64-bit systems.
- Due to all these features, it is a popular choice as an Application File Format.

Steps to perform:

Step 1. Creating a new Android Project with Kotlin in Android Studio.
Step 2. Adding User Interface in your layout file.
Step 3. Creating a Database and tables using Kotlin.
Step 4. Implementation of CRUD Operations in Android applications.

Note: CRUD- **Create**: Operation to create data in a content provider.
Read: Used to fetch data from a content provider.
Update: To modify existing data.
Delete: To remove existing data from the storage.

listitem.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:weightSum="2">

    <TextView
        android:id="@+id/txt_item"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:padding="10dp"
        android:textColor="@color/purple_700"
        android:textSize="30dp"
        tools:text="Data" />

    <TextView
        android:id="@+id/txt_qty"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignRight="@id/txt_item"
        android:layout_weight="1"
```

```
        android:gravity="right"
        android:padding="10dp"
        android:textColor="@color/purple_700"
        android:textSize="30dp"
        tools:text="QTY" />
</LinearLayout>
```

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <Button
        android:id="@+id/btn_additem"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="Add Item" />
</RelativeLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent">
    <EditText
        android:id="@+id/edit_itemname"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:hint="Item Name"/>

    <EditText
        android:id="@+id/edit_qty"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:hint="Item Quantity"/>
    <Button
        android:id="@+id/save_btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Save Item"/>

</LinearLayout>
```

activity_edit_item.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent">
    <EditText
        android:id="@+id/edit_itemname"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:hint="Item Name"/>

    <EditText
        android:id="@+id/edit_qty"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:hint="Item Quantity"/>
    <Button
        android:id="@+id/save_btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Save Item"/>

</LinearLayout>
```

activity_add_item.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent">
    <EditText
        android:id="@+id/edit_additemname"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:hint="Item Name"/>

    <EditText
        android:id="@+id/edit_addqty"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:hint="Item Quantity"/>
    <Button
        android:id="@+id/save_btn"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Save Item"/>

</LinearLayout>
```

Adapter.kt

```
class myAdapter(
    private val context: Context,
    private val listItems: List<String>,
    private val qtyItems: List<String>
) : BaseAdapter() {
    private var layoutInflater: LayoutInflater? = null
    private lateinit var item: TextView
    private lateinit var qty: TextView

    override fun getCount(): Int {
        return listItems.size
    }

    override fun getItem(p0: Int): Any? {
        return null
    }

    override fun getItemId(p0: Int): Long {
        return 0
    }

    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        var convertView = convertView
        if (layoutInflater == null) {
            layoutInflater =
                context.getSystemService(Context.LAYOUT_INFLATER_SERVICE) as LayoutInflater
        }
        if (convertView == null) {
            convertView = layoutInflater!!.inflate(R.layout.listitem, null)
        }
        item = convertView!!.findViewById(R.id.txt_item)
        qty = convertView!!.findViewById(R.id.txt_qty)
        item.text = listItems[position]
        qty.text = qtyItems[position]
        return convertView
    }

}
```

DBHelper.kt

```
class DBHelper(context: Context, factory: SQLiteDatabase.CursorFactory?) :
    SQLiteOpenHelper(context, DATABASE_NAME, factory, DATABASE_VERSION) {
    companion object {
```

```
    private const val DATABASE_NAME = "GROCERY"
    private const val DATABASE_VERSION = 1
    const val TABLE_NAME = "grocery_list"
    const val ID_COL = "id"
    const val NAME_COl = "name"
    const val QNT_COL = "qnty"
}

override fun onCreate(db: SQLiteDatabase) {
    val query = ("CREATE TABLE " + TABLE_NAME + " ("
            + ID_COL + " INTEGER PRIMARY KEY, " +
            NAME_COl + " TEXT," +
            QNT_COL + " TEXT" + ")")
    db.execSQL(query)

}

override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME)
    onCreate(db)
}

fun setItem(name: String, qty: String) {
    val values = ContentValues()
    values.put(NAME_COl, name)
    values.put(QNT_COL, qty)
    val db = this.writableDatabase
    db.insert(TABLE_NAME, null, values)
}

fun getItems(): List<String>? {
    val db = this.readableDatabase
    var items = mutableListOf<String>()
    var cur = db.rawQuery("SELECT * FROM " + TABLE_NAME, null)
    if (cur.moveToFirst()) {
        while (cur.moveToNext()) {
            items.add(cur!!.getString(1))
        }
    }
    return items
}

fun getItem(name: String): String {
    val db = this.readableDatabase
    var cur = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE $NAME_COl = '$name'",
null)
    cur.moveToFirst()
    return cur.getString(2).toString()
}
```

```
    fun updateItem(name: String, qty: String) {
        val values = ContentValues()
        values.put(QNT_COL, qty)
        val db = this.writableDatabase
        db.update(TABLE_NAME, values, NAME_COl + "=?", arrayOf(name))
    }

    fun delItem(name: String) {
        val db = this.writableDatabase
        db.delete(TABLE_NAME, NAME_COl + "=?", arrayOf(name))
    }

    fun getQty(): List<String>? {
        val db = this.readableDatabase
        var items = mutableListOf<String>()
        var cur = db.rawQuery("SELECT * FROM " + TABLE_NAME, null)
        if (cur.moveToFirst()) {
            while (cur.moveToNext()) {
                items.add(cur!!.getString(2))
            }
        }
        return items
    }
}
```

MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    lateinit var list: ListView
    lateinit var btn_addItem: Button
    var itemArray: List<String>? = null
    var qtyArray: List<String>? = null
    val db: DBHelper = DBHelper(this, null)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        list = findViewById(R.id.list)
        btn_addItem = findViewById(R.id.btn_additem)
        reset()

        btn_addItem.setOnClickListener {
            startActivity(Intent(this, AddItemActivity::class.java))
            finish()
        }
    }

    private fun reset() {
        itemArray = db.getItems()
        qtyArray = db.getQty()
        val mainAdapter = myAdapter(this, itemArray!!,qtyArray!!)
```

```
        list.adapter = mainAdapter
        list.onItemClickListener = AdapterView.OnItemClickListener { _, _,
                                                    position, _ ->
            db.delItem(itemArray!![position])
            reset()
        }
        list.onItemLongClickListener = AdapterView.OnItemLongClickListener { _, _, position, _ ->
            startActivity(
                Intent(this, EditItemActivity::class.java).putExtra(
                    "itemName",
                    itemArray!![position]
                )
            )
            finish()
            true
        }
    }
}
```

AddItemActivity.kt

```
class AddItemActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_add_item)
        val save_btn: Button = findViewById(R.id.save_btn)
        val itemName: EditText = findViewById(R.id.edit_additemname)
        val itemQty: EditText = findViewById(R.id.edit_addqty)
        val db: DBHelper = DBHelper(this, null)
        save_btn.setOnClickListener {
            db.setItem(itemName.text.toString(), itemQty.text.toString())
            startActivity(Intent(this, MainActivity::class.java))
            Toast.makeText(this, "Item Added to the list", Toast.LENGTH_LONG).show()
            finish()
        }
    }
}
```

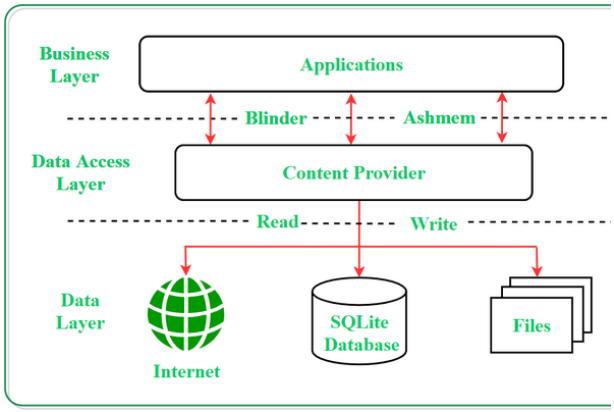
EditItemActivity.kt

```
class EditItemActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_edit_item)

        val edit_btn: Button = findViewById(R.id.save_btn)
        val itemName: EditText = findViewById(R.id.edit_itemname)
        val itemQty: EditText = findViewById(R.id.edit_qty)
        val db: DBHelper = DBHelper(this, null)

        itemName.setText(intent.getStringExtra("itemName")!!)
        itemName.isEnabled = false
```

```
        itemQty.setText(db.getItem(intent.getStringExtra("itemName")!!))

        edit_btn.setOnClickListener {
            db.updateItem(intent.getStringExtra("itemName")!!, itemQty.text.toString())
            startActivity(Intent(this, MainActivity::class.java))
            Toast.makeText(this, "Item Edited to the list", Toast.LENGTH_LONG).show()
            finish()
        }
    }
}
```

Content Provider

In Android, Content Providers are a very important component that serves the purpose of a relational database to store the data of applications. The role of the content provider in the android system is like a central repository in which data of the applications are stored, and it facilitates other applications to securely access and modifies that data based on the user requirements. Android system allows the content provider to store the application data in several ways. Users can manage to store the application data like images, audio, videos, and personal contact information by storing them in SQLite Database, in files, or even on a network. In order to share the data, content providers have certain permissions that are used to grant or restrict the rights to other applications to interfere with the data.



Content URI
Content URI(Uniform Resource Identifier) is the key concept of Content providers. To access the data from a content provider, URI is used as a query string.
Structure of a Content URI:
content://authority/optionalPath/optionalID
Details of different parts of Content URI:
content:// – Mandatory part of the URI as it represents that the given URI is a Content URI.
authority – Signifies the name of the content provider like contacts, browser, etc. This part must be unique for every content provider.
optionalPath – Specifies the type of data provided by the content provider. It is essential as this part helps content providers to support different types of data that are not related to each other like audio and video files.
optionalID – It is a numeric value that is used when there is a need to access a particular record.
If an ID is mentioned in a URI then it is an id-based URI otherwise a directory-based URI.

Operations in Content Provider
Four fundamental operations are possible in Content Provider namely Create, Read, Update, and Delete. These operations are often termed as CRUD operations.

Working of the Content Provider
UI components of android applications like Activity and Fragments use an object CursorLoader to send query requests to ContentResolver. The ContentResolver object sends requests (like create, read, update, and delete) to the ContentProvider as a client. After receiving a request, ContentProvider

process it and returns the desired result. Below is a diagram to represent these processes in pictorial form.

Creating a Content Provider

Following are the steps which are essential to follow in order to create a Content Provider:

1. Create a class in the same directory where the that MainActivity file resides and this class must extend the ContentProvider base class.
2. To access the content, define a content provider URI address.
3. Create a database to store the application data.
4. Implement the six abstract methods of ContentProvider class.
5. Register the content provider in AndroidManifest.xml file using <provider> tag.

Following are the six abstract methods and their description which are essential to override as the part of ContenProvider class:

Abstract Method	Description
query()	A method that accepts arguments and fetches the data from the desired table. Data is retired as a cursor object.
insert()	To insert a new row in the database of the content provider. It returns the content URI of the inserted row.
update()	This method is used to update the fields of an existing row. It returns the number of rows updated.
delete()	This method is used to delete the existing rows. It returns the number of rows deleted.
getType()	This method returns the Multipurpose Internet Mail Extension(MIME) type of data to the given Content URI.
onCreate()	As the content provider is created, the android system calls this method immediately to initialise the provider.

MyContentProvider.kt

```
class MyContentProvider : ContentProvider() {
   companion object {
      // defining authority so that other application can access it
      const val PROVIDER_NAME = "com.nkp.provider"

      // defining content URI
      const val URL = "content://$PROVIDER_NAME/users"

      // parsing the content URI
      val CONTENT_URI = Uri.parse(URL)
      const val id = "id"
      const val name = "name"
      const val uriCode = 1
      var uriMatcher: UriMatcher? = null
      private val values: HashMap<String, String>? = null

      // declaring name of the database
      const val DATABASE_NAME = "UserDB"

      // declaring table name of the database
```

```
    const val TABLE_NAME = "Users"

    // declaring version of the database
    const val DATABASE_VERSION = 1

    // sql query to create the table
    const val CREATE_DB_TABLE =
        (" CREATE TABLE " + TABLE_NAME
                + " (id INTEGER PRIMARY KEY AUTOINCREMENT, "
                + " name TEXT NOT NULL);")

    init {

        // to match the content URI
        // every time user access table under content provider
        uriMatcher = UriMatcher(UriMatcher.NO_MATCH)

        // to access whole table
        uriMatcher!!.addURI(
            PROVIDER_NAME,
            "users",
            uriCode
        )

        // to access a particular row
        // of the table
        uriMatcher!!.addURI(
            PROVIDER_NAME,
            "users/*",
            uriCode
        )
    }
}

override fun getType(uri: Uri): String? {
    return when (uriMatcher!!.match(uri)) {
        uriCode -> "vnd.android.cursor.dir/users"
        else -> throw IllegalArgumentException("Unsupported URI: $uri")
    }
}

// creating the database
override fun onCreate(): Boolean {
    val context = context
    val dbHelper =
        DatabaseHelper(context)
    db = dbHelper.writableDatabase
    return if (db != null) {
        true
    } else false
```

```
}

override fun query(
    uri: Uri, projection: Array<String>?, selection: String?,
    selectionArgs: Array<String>?, sortOrder: String?
): Cursor? {
    var sortOrder = sortOrder
    val qb = SQLiteQueryBuilder()
    qb.tables = TABLE_NAME
    when (uriMatcher!!.match(uri)) {
        uriCode -> qb.projectionMap = values
        else -> throw IllegalArgumentException("Unknown URI $uri")
    }
    if (sortOrder == null || sortOrder === "") {
        sortOrder = id
    }
    val c = qb.query(
        db, projection, selection, selectionArgs, null,
        null, sortOrder
    )
    c.setNotificationUri(context!!.contentResolver, uri)
    return c
}

// adding data to the database
override fun insert(uri: Uri, values: ContentValues?): Uri? {
    val rowID = db!!.insert(TABLE_NAME, "", values)
    if (rowID > 0) {
        val _uri =
            ContentUris.withAppendedId(CONTENT_URI, rowID)
        context!!.contentResolver.notifyChange(_uri, null)
        return _uri
    }
    throw SQLiteException("Failed to add a record into $uri")
}

override fun update(
    uri: Uri, values: ContentValues?, selection: String?,
    selectionArgs: Array<String>?
): Int {
    var count = 0
    count = when (uriMatcher!!.match(uri)) {
        uriCode -> db!!.update(TABLE_NAME, values, selection, selectionArgs)
        else -> throw IllegalArgumentException("Unknown URI $uri")
    }
    context!!.contentResolver.notifyChange(uri, null)
    return count
}

override fun delete(
```

```
        uri: Uri,
        selection: String?,
        selectionArgs: Array<String>?
    ): Int {
        var count = 0
        count = when (uriMatcher!!.match(uri)) {
            uriCode -> db!!.delete(TABLE_NAME, selection, selectionArgs)
            else -> throw IllegalArgumentException("Unknown URI $uri")
        }
        context!!.contentResolver.notifyChange(uri, null)
        return count
    }

    // creating object of database
    // to perform query
    private var db: SQLiteDatabase? = null

    // creating a database
    private class DatabaseHelper  // defining a constructor
    internal constructor(context: Context?) : SQLiteOpenHelper(
        context,
        DATABASE_NAME,
        null,
        DATABASE_VERSION
    ) {
        // creating a table in the database
        override fun onCreate(db: SQLiteDatabase) {
            db.execSQL(CREATE_DB_TABLE)
        }

        override fun onUpgrade(
            db: SQLiteDatabase,
            oldVersion: Int,
            newVersion: Int
        ) {

            // sql query to drop a table
            // having similar name
            db.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
            onCreate(db)
        }
    }
}
```

activity_main.xml

```
<LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
```

```
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/heading"
        android:textAlignment="center"
        android:textSize="36sp" />

    <EditText
        android:id="@+id/textName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/hintText" />

    <Button
        android:id="@+id/insertButton"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/insertButtontext" />

    <Button
        android:id="@+id/loadButton"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/loadButtonText" />

    <TextView
        android:id="@+id/res"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:clickable="false"
        android:textColor="@color/purple_700"
        android:textSize="25sp"
        android:textStyle="bold" />

</LinearLayout>
```

MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val add: Button = findViewById(R.id.insertButton)
        val show: Button = findViewById(R.id.loadButton)

        add.setOnClickListener {
            val values = ContentValues()
```

```
        // fetching text from user
        values.put(
            ContentProvider.name,
            (findViewById<View>(R.id.textName) as EditText).text.toString()
        )

        // inserting into database through content URI
        contentResolver.insert(ContentProvider.CONTENT_URI, values)

        // displaying a toast message
        Toast.makeText(baseContext, "New Record Inserted", Toast.LENGTH_LONG).show()
    }
    show.setOnClickListener {
        // inserting complete table details in this text field
        val resultView = findViewById<View>(R.id.res) as TextView

        // creating a cursor object of the
        // content URI
        val cursor = contentResolver.query(
            Uri.parse("content:// com.nkp.provider/users"),
            null,
            null,
            null,
            null
        )

        // iteration of the cursor
        // to print whole table
        if (cursor!!.moveToFirst()) {
            val strBuild = StringBuilder()
            while (!cursor.isAfterLast) {
                strBuild.append(
                    """

  ${cursor.getString(0)}-${cursor.getString(1)}
  """.trimIndent()
                )
                cursor.moveToNext()
            }
            resultView.text = strBuild
        } else {
            resultView.text = "No Records Found"
        }
    }
  }
}
```

AndroidManifest.xml

```
<provider
```

```
android:name="com.nkp.MyContentProvider"
android:authorities="com.demo.user.provider"
android:enabled="true"
android:exported="true"></provider>
```