20IT505

Hall Ticket Number:

III/IV B. Tech (Regular/Supplementary) DEGREE EXAMINATION

December, 2025 **Information Technology**

Fifth Semester **Mobile Application Development**

Time: Three Hours **Maximum:**70 Marks

Answer question 1 compulsorily. **(14X1 = 14Marks)**

Answer one question from each unit. **(4X14=56Marks)**

			CO	BL	M
1	a)	Define Android SDK.	CO1	L1	1M
	b)	Draw the Android platform architecture in brief (label only 4 layers).	CO1	L2	1M
	c)	What are Foreground Applications?	CO1	L1	1M
	d)	Name any four core Android libraries.	CO1	L1	1M
	e)	Name any two attributes that must be declared in the tag of the manifest file.	CO2	L1	1M
	f)	Which XML file is used to externalize string resources in Android?	CO2	L1	1M
	g)	Write the XML tag used to define a Fragment in a layout file.	CO2	L2	1M
	h)	Which method is called when an Activity becomes visible but not interactive?	CO2	L1	1M
	i)	Name the two types of Intents.	CO3	L1	1M
	j)	Which action string is used for the main entry point of an app in Intent filters?	CO3	L1	1M
	k)	Write the key used in Bundle to retrieve saved instance state in onCreate().	CO3	L2	1M
	l)	Name the main class used to perform database operations in Android.	CO4	L1	1M
	m)	Name the URI scheme used by all Content Providers.	CO4	L1	1M
	n)	Which method of ContentResolver is used to insert data via a Content Provider?	CO4	L1	1M
		<u>Unit-I</u>			
2	a)	Explain any seven key features of Android SDK in detail.	CO1	L2	7M
	b)	Describe the steps to create and run your First Android Application using Android Studio with screenshots/flow.	CO1	L2	7M
		(OR)			
3	a)	Explain the Android Development Framework with a neat diagram. List the major components.	CO1	L2	7M
	b)	Differentiate between the following types of Android applications: i) Foreground Applications ii) Background Applications iii) Intermittent Applications iv) Widgets	CO1	L2	7M
		<u>Unit-II</u>			
4	a)	Explain the structure and important elements of the **AndroidManifest.xml** file with suitable examples.	CO2	L2	7M
	b)	Describe the **complete Android Activity Lifecycle** with a neat diagram. Show all callback methods.	CO2	L2	7M
		(OR)			
5	a)	What makes an Android application different from traditional Java applications? Explain the essential components of an Android application.	CO2	L2	7M
	b)	Explain the following **Activity states** in detail: i) Running ii) Paused iii) Stopped iv) Destroyed	CO2	L2	7M
		<u>Unit-III</u>			
6	a)	Explain the concept of Intents in Android. Differentiate between Explicit and Implicit Intents with suitable examples.	CO3	L2	7M
	b)	Describe how to create and register a Broadcast Receiver both statically (in manifest) and dynamically (in code) with examples.	CO3	L2	7M
		(OR)			
7	a)	Explain the complete process of creating, saving, and retrieving SharedPreferences in Android with a coding example.	CO3	L2	7M
	b)	Write short notes on the following: i) PreferenceFragmentCompatii) getDefaultSharedPreferences() vs getPreferences() iii) Modes in SharedPreferences	CO3	L2	7M
		<u>Unit-IV</u>			
8	a)	Explain the complete structure of a Content Provider in Android. Write the mandatory methods that must be overridden and show a sample authority declaration.	CO4	L2	7M
	b)	With a neat diagram and code, explain how to create a bound service and bind it from an Activity using Binder.	CO4	L2	7M
		(OR)			
9	a)	Explain SQLiteOpenHelper class. Write a complete code to create a database with one table “Employee” having columns id, name, salary, and department.	CO4	L3	7M
	b)	Describe how to use ContentValues and Cursor to insert and retrieve data from SQLite database with example code.	CO4	L2	7M

20IT505 – Mobile Application Development
Detailed Scheme of Valuation (December 2025)

GENERAL VALUATION GUIDELINES

- This scheme is prepared strictly as per the given question paper.
- CO (Course Outcome) and BL (Bloom's Level) mapping are to be respected.
- Correct concept carries full weightage even if minor syntax/grammar errors exist.
- Diagrams: logical correctness and labeling are more important than artistic quality.
- Equivalent answers and alternative valid terminologies shall be accepted.

QUESTION 1 – VERY SHORT ANSWERS (14 × 1 = 14 Marks)

1(a) Define Android SDK (CO1 – L1 – 1M)

Expected Answer: Android SDK (Software Development Kit) is a collection of APIs, libraries, development tools, emulator, and documentation used to develop, test, and debug Android applications.

Marks Distribution:

- Expansion of SDK – 0.5M
- Purpose/tools explanation – 0.5M

1(b) Android Platform Architecture – 4 Layers (CO1 – L2 – 1M)

Expected Answer: Correctly labeled any four layers of Android architecture.

Acceptable Layers:

- Applications
- Application Framework
- Android Runtime (ART/Dalvik)
- Native Libraries
- Linux Kernel

Marks:

- Any four correct layers – 1M

1(c) Foreground Applications (CO1 – L1 – 1M)

Foreground applications are applications currently running and interacting directly with the user, having the highest priority in Android.

Marks: 1M

1(d) Core Android Libraries (CO1 – L1 – 1M)

Any four: SQLite, WebKit, OpenGL ES, Media Framework, Surface Manager, libc, SSL.

Marks: 0.25M × 4 = 1M

1(e) Attributes in tag (CO2 – L1 – 1M)

Any two: package, xmlns:android, versionCode, versionName.

Marks: 0.5M × 2 = 1M

1(f) XML file for string resources (CO2 – L1 – 1M)

Answer: strings.xml

Marks: 1M

1(g) Fragment XML tag (CO2 – L2 – 1M)

Answer:

Marks: 1M

1(h) Activity visible but not interactive (CO2 – L1 – 1M)

Answer: onStart()

Marks: 1M

1(i) Types of Intents (CO3 – L1 – 1M)

- Explicit Intent
- Implicit Intent

Marks: 0.5M + 0.5M

1(j) Main entry point action string (CO3 – L1 – 1M)

Answer: android.intent.action.MAIN

Marks: 1M

1(k) Bundle key for saved instance state (CO3 – L2 – 1M)

Answer: savedInstanceState

Marks: 1M

1(l) Database operations class (CO4 – L1 – 1M)

Answer: SQLiteDatabase

Marks: 1M

1(m) Content Provider URI scheme (CO4 – L1 – 1M)

Answer: content://

Marks: 1M

1(n) ContentResolver insert method (CO4 – L1 – 1M)

Answer: insert()

Marks: 1M

UNIT – I

Q2(a) Explain any seven key features of Android SDK (CO1 – L2 – 7M)

Detailed Expected Answer & Valuation:

1. **Application Framework** – Provides high-level services to applications such as Activity Manager, Window Manager, Content Providers, and View system. Enables component reuse and modular app design. *(1M)*
2. **Android Runtime (ART)** – Executes applications using Ahead-Of-Time (AOT) compilation, improving performance and battery efficiency. Manages core Java libraries. *(1M)*
3. **Android Emulator** – Simulates Android devices for testing applications without physical hardware. Supports different screen sizes and API levels. *(1M)*
4. **ADB (Android Debug Bridge)** – Command-line tool for debugging, installing apps, accessing logs, and interacting with devices/emulators. *(1M)*
5. **Rich APIs** – APIs for UI, multimedia, sensors, GPS, camera, Bluetooth, Wi-Fi, and telephony. *(1M)*
6. **SQLite Database Support** – Lightweight relational database embedded in Android for persistent storage. *(1M)*
7. **Gradle Build System** – Manages dependencies, build variants, APK generation, and testing automation. *(1M)*

Q2(b) Describe the steps to create and run your First Android Application using Android Studio with screenshots/flow (CO1 – L2 – 7M)

Steps to Create and Run Your First Android Application Using Android Studio (5M)

Flow diagram (2M)

Step 1: Install Android Studio

1. Download Android Studio from:
   □ https://developer.android.com/studio
2. Install with default settings.
3. Ensure **Android SDK**, **AVD (Emulator)**, and **Platform Tools** are installed.

Step 2: Launch Android Studio

1. Open **Android Studio**.
2. Click **"New Project"** on the welcome screen.

Step 3: Select a Project Template

1. Choose **Empty Activity**.
2. Click **Next**.

Step 4: Configure Project

Fill the following details:

- **Application Name:** MyFirstApp
- **Package Name:** com.example.myfirstapp
- **Save Location:** Default
- **Language:** Java or Kotlin
- **Minimum SDK:** API 21 (Android 5.0 or above)

Click **Finish**.

Step 5: Project Structure Creation

Android Studio automatically creates:

- MainActivity.java / MainActivity.kt
- activity_main.xml
- AndroidManifest.xml
- Resource folders (layout, values, drawable)

Step 6: Design the User Interface

Open:

res → layout → activity_main.xml

Example UI code:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="24sp"/>
```

Step 7: Write Activity Code

Open:

MainActivity.java

Sample code:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Step 8: Run the Application

1. Click **Run ▶ button**.
2. Choose:
   - **Emulator (AVD)** or
   - **Physical Device (USB debugging ON)**
3. App installs and runs automatically.

Step 9: Output

The app displays:

Hello World!

on the mobile screen.

Flow Diagram (Text Representation)

Start

↓

Open Android Studio

↓

Create New Project

↓

Select Empty Activity

↓

Configure Project Settings

↓

Android Studio Generates Files

↓

Design UI (XML)

↓

Write Activity Code

↓

Run App (Emulator / Device)

↓

App Output Displayed

↓

End

Important Files Used

File Name	Purpose
MainActivity.java	App logic
activity_main.xml	UI design
AndroidManifest.xml	App configuration
strings.xml	String resources

Q3(a) Explain the Android Development Framework with a neat diagram. List the major components (CO1 – L2 – 7M)

Detailed Valuation:

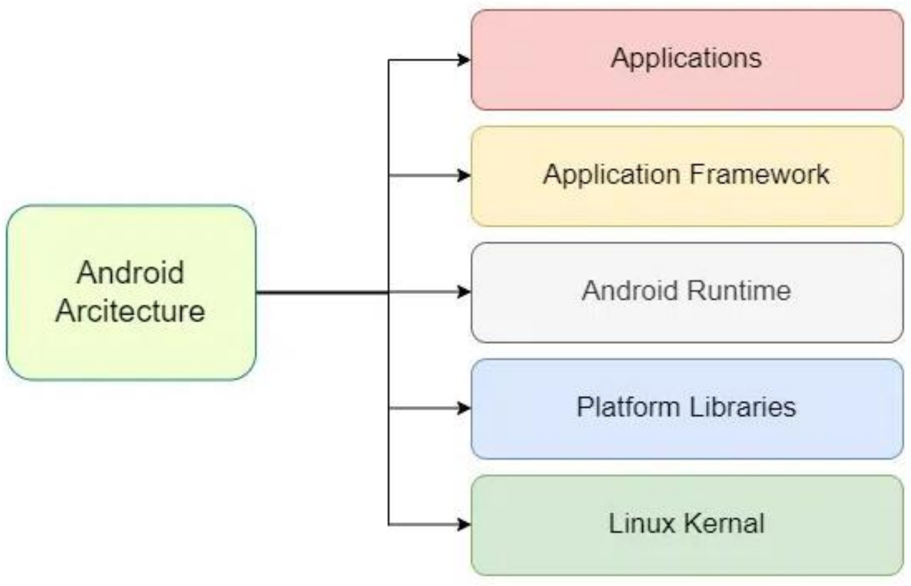
- Neat layered architecture diagram showing Applications, Framework, Runtime, Libraries, Kernel. *(3M)*
- Explanation of Application Framework services. *(2M)*
- Listing and explanation of major components (Activities, Services, Broadcast Receivers, Content Providers). *(2M)*

Android Development Framework

Definition

The **Android Development Framework** is a set of APIs, system services, and tools provided by Android that allow developers to build, run, and manage Android applications efficiently. It acts as a bridge between applications and the underlying hardware.

Android Framework Architecture (Neat Diagram)



Major Components of Android Development Framework

1. Applications Layer

- Contains built-in and user-installed applications.
- Examples: Phone, SMS, Camera, Browser, and third-party apps.
- All apps use the same framework APIs.

---

2. Application Framework

Provides high-level services to applications

Key Components:

- **Activity Manager** – Manages activity lifecycle.
- **Window Manager** – Manages windows and UI display.
- **Content Providers** – Share data between apps.
- **View System** – Builds UI components.
- **Resource Manager** – Manages app resources.
- **Notification Manager** – Displays alerts and notifications.
- **Location Manager** – Accesses location services.
- **Package Manager** – Manages app installation.

---

3. Android Runtime & Native Libraries

Android Runtime (ART):

- Executes app byte code.
- Handles memory management and garbage collection.

Native Libraries:

- **SQLite** – Database storage
- **OpenGL ES** – Graphics rendering
- **WebKit** – Web browser support
- **Media Framework** – Audio/video playback
- **SSL** – Security

---

4. Linux Kernel

- Core of the Android OS.
- Manages:
  - Device drivers
  - Process & memory management
  - Power management
  - Networking
  - Security enforcement

Q3(b) Types of Android Applications (CO1 – L2 – 7M)

- **Foreground Applications** – Active apps interacting with users. *(1.75M)*
- **Background Applications** – Apps running without UI. *(1.75M)*
- **Intermittent Applications** – Apps triggered occasionally (alarms, notifications). *(1.75M)*
- **Widgets** – Mini apps on home screen. *(1.75M)*

Difference between Types of Android Applications

Type	Description	User Interaction	Examples
Foreground Applications	Applications currently visible on the screen and actively interacting with the user.	High – user directly interacts with the app.	WhatsApp, YouTube, Camera
Background Applications	Applications running in the background without direct user interaction, performing tasks silently.	None or very minimal	Music player, file download service, location tracking
Intermittent Applications	Applications that run occasionally based on events or system triggers.	Limited or indirect	SMS receiver, alarm app, notification listener
Widgets	Small application views displayed on the home screen for quick access to information.	Limited interaction	Clock widget, Weather widget, Music control widget

1. Foreground Applications

- Actively running and visible to the user.
- Have high priority in the system.
- Directly interact with user inputs.

Example: Browsing a website using Chrome.

2. Background Applications

- Run behind the scenes without user interface.
- Used for long-running tasks such as music playback or file uploads.

Example: Google Drive sync service.

3. Intermittent Applications

- Activated occasionally by system events or user actions.
- Often triggered by broadcasts.

Example: SMS receiver responding to incoming messages.

4. Widgets

- Miniature application components displayed on the home screen.
- Provide quick access to app information or controls.

Example: Calendar or Weather widget.

Key Differences Summary

Feature	Foreground	Background	Intermittent	Widget
UI Visibility	Yes	No	Partial	Yes (limited)
Execution Time	Continuous	Long-running	Event-based	Continuous (light)
User Interaction	High	None	Minimal	Limited
Resource Usage	High	Medium	Low	Very Low

UNIT – II

Q4(a) Explain the structure and important elements of the AndroidManifest.xml file with suitable examples. ***(CO2 – L2 – 7 Marks)***

The **AndroidManifest.xml** file is a **mandatory configuration file** in every Android application.(1M)

It provides essential information about the app to the **Android operating system**, such as:

- App components (Activities, Services, Receivers, Providers)
- Permissions required
- App entry point
- Minimum SDK version
- Hardware and software features

Without this file, the Android system **cannot run the application**.

Structure of AndroidManifest.xml (1M)

Basic Structure

```
<manifest>
  <application>
    <activity>
    </activity>
  </application>
</manifest>
```

Important Elements of AndroidManifest.xml (5M)

1. <manifest> Tag

Purpose:

- Root element of the manifest file.
- Defines the package name and version information.

Example:

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapp">
```

Important Attributes:

- package – Unique app identifier.

- versionCode
- versionName

---

2. <application> Tag

Purpose:

- Contains all application components.
- Defines global app settings.

Example:

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/Theme.MyApp">
```

Common Attributes:

- android:icon
- android:label
- android:theme
- android:allowBackup

3. <activity> Tag

Purpose:

- Declares an activity (UI screen).

Example:

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Explanation:

- MAIN → Entry point of the app
- LAUNCHER → App appears in app launcher

4. <intent-filter> Tag

Purpose:

- Defines how components respond to intents.

Example:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

5. <uses-permission> Tag

Purpose:

- Declares permissions required by the app.

Example:

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

6. <uses-sdk> Tag

Purpose:

- Specifies minimum and target Android versions.

Example:

```
<uses-sdk
    android:minSdkVersion="21"
    android:targetSdkVersion="34" />
```

7. <service> Tag

Purpose:

- Declares background services.

Example:

```
<service android:name=".MyService" />
```

8. <receiver> Tag

Purpose:

- Declares broadcast receivers.

Example:

```
<receiver android:name=".MyReceiver" />
```

9. <provider> Tag

Purpose:

- Declares a content provider.

Example:

```
<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.myapp.provider" />
```

Complete Sample AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

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

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.MyApp">

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
```

</manifest>

Summary Table

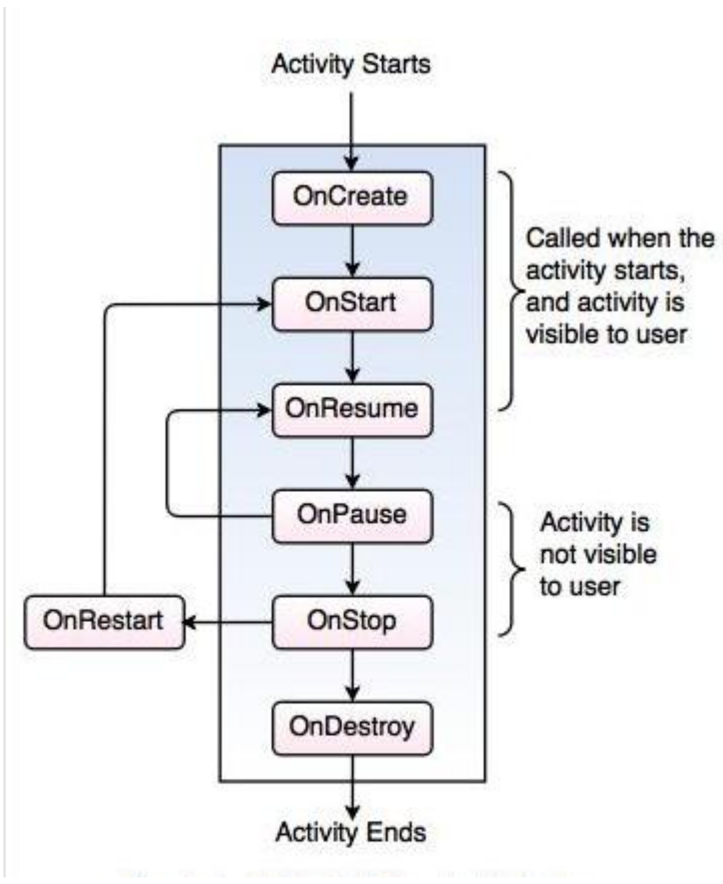
Element	Purpose
<manifest>	Root of the application
<application>	Defines app-level settings
<activity>	Represents UI screens
<intent-filter>	Handles intents
<uses-permission>	Declares permissions
<service>	Background operations
<receiver>	Listens for system events
<provider>	Manages shared app data

Q4(b) Describe the complete Android Activity Lifecycle with a neat diagram. Show all callback methods. ***(CO2 – L2 – 7 Marks)***

Introduction (1M)

An **Activity** represents a single screen with a user interface in an Android application. The **Activity Lifecycle** defines the various states an activity goes through from creation to destruction. Android manages these states using **callback methods**.

Android Activity Lifecycle Diagram (2M)


Activity Starts
OnCreate
OnStart
OnResume
OnPause
OnStop
OnDestroy
OnRestart
Activity Ends
Called when the activity starts, and activity is visible to user
Activity is not visible to user

Lifecycle Callback Methods Explanation (4M)

1. onCreate()

- Called when the activity is **first created**.
- Used to initialize UI components.
- setContentView() is called here.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

---

2. onStart()

- Called when the activity becomes **visible** to the user.
- Activity is not yet interactive.

---

3. onResume()

- Called when the activity starts **interacting with the user**.
- Activity is in the **foreground**.

---

4. onPause()

- Called when another activity partially covers the current one.
- Used to **pause ongoing tasks**, animations, or save data.

---

5. onStop()

- Called when the activity is **no longer visible**.
- Heavy operations like releasing resources are done here.

---

6. onRestart()

- Called when the activity is restarting after being stopped.

---

7. onDestroy()

- Called before the activity is destroyed.
- Used for **final cleanup**.

Lifecycle Flow Summary Table

Method	Purpose
onCreate()	Initialize activity
onStart()	Activity becomes visible
onResume()	User starts interaction
onPause()	Activity partially hidden
onStop()	Activity fully hidden
onRestart()	Restart after stop
onDestroy()	Cleanup before destruction

Simple Lifecycle Code Example

```
@Override
protected void onStart() {
super.onStart();
Log.d("Activity", "onStart called");
}

@Override
protected void onResume() {
super.onResume();
Log.d("Activity", "onResume called");
}

@Override
protected void onPause() {
super.onPause();
Log.d("Activity", "onPause called");
}

@Override
protected void onStop() {
super.onStop();
Log.d("Activity", "onStop called");
}

@Override
protected void onDestroy() {
super.onDestroy();
Log.d("Activity", "onDestroy called");
}
```

Q5(a) What makes an Android application different from traditional Java applications? Explain the essential components of an Android application. ***(CO2 – L2 – 7 Marks)***

Difference Between Android Applications and Traditional Java Applications

Android applications differ significantly from traditional Java applications in terms of **architecture, execution environment, UI design, and lifecycle management**.

Key Differences (3M)

Aspect	Android Application	Traditional Java Application
Platform	Runs on Android OS	Runs on desktop/server JVM
Execution Environment	Uses **ART (Android Runtime)**	Uses **JVM (Java Virtual Machine)**
User Interface	XML-based layouts + Activities	AWT / Swing / JavaFX
Lifecycle Management	Managed by Android OS	Controlled by developer
Entry Point	Activity declared in Manifest	main() method
Resource Management	Automatic resource handling via framework	Manual resource handling
App Structure	Component-based	Class-based
Hardware Access	Through Android APIs (camera, GPS, sensors)	Limited direct hardware access
Security	Permission-based model	No built-in permission model
Execution Style	Event-driven	Procedural / event-based

Essential Components of an Android Application (4M)

An Android application is built using **four main components**, declared in the AndroidManifest.xml.

1. Activity

➢ **Purpose:**

Represents a **single screen with a user interface**.

➢ **Example:**

```
public class MainActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

➢ **Example Use:**

Login screen, Home screen, Settings page.

2. Service

➢ **Purpose:**

Performs **background operations** without user interaction.

➢ **Example:**

- Music playback
- File download

```
public class MyService extends Service {
    public int onStartCommand(Intent intent, int flags, int startId) {
        return START_STICKY;
    }
}
```

3. Broadcast Receiver

➢ **Purpose:**

Responds to **system-wide broadcast messages**.

➢ **Example:**

- Battery low
- SMS received
- Network change

```
public class MyReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        // handle event
    }
}
```

4. Content Provider

➢ **Purpose:**

Manages and shares application data with other apps.

➢ **Example:**

- Contacts Provider
- Media Store

```
public class MyProvider extends ContentProvider {
    public boolean onCreate() {
        return true;
    }
}
```

Supporting Components

Component	Purpose
Intent	Used to communicate between components
Manifest File	Declares app components and permissions
Resources (res/)	Stores layouts, strings, images
Gradle Files	Manages dependencies and build config

Summary Diagram (Text Form)

```
Android Application
        |
------------------------------------------------------------
| Activity | Service | Broadcast Receiver | Provider |
------------------------------------------------------------
        |
     Android Framework
        |
     Linux Kernel
```

Q5(b) Explain the following Activity states in detail: i) Running ii) Paused iii) Stopped iv) Destroyed ***(CO2 – L2 – 7 Marks)***

Android Activity States (5M)

An **Activity** represents a single screen in an Android application.

During its lifetime, an activity moves through different **states** depending on user interaction and system conditions.

---

i) Running (Active) State

Description:

- The activity is in the **foreground**.
- The user is actively interacting with it.
- It has the **highest priority** in the system.

Lifecycle Method:

onResume()

Characteristics:

- Fully visible on the screen.
- Receives all user inputs.
- Consumes system resources actively.

Example:

User is typing a message in WhatsApp.

ii) Paused State

Description:

- The activity is **partially visible** but not in focus.

- Another activity partially covers it (e.g., dialog or popup).

Lifecycle Method:

onPause()

Characteristics:

- Activity is still alive but cannot receive user input.
- Should release resources like animations or sensors.
- Can return quickly to the Running state.

Example:

Incoming call popup appears over an app.

iii) Stopped State

Description:

- The activity is **completely hidden** from the user.
- It remains in memory but is not visible.

Lifecycle Method:

onStop()

Characteristics:

- Activity state is preserved.
- System may destroy it to free memory.
- Heavy resources should be released.

Example:

User presses the Home button.

iv) Destroyed State

Description:

- The activity is **removed from memory**.
- Occurs when the user closes the app or system needs resources.

Lifecycle Method:

onDestroy()

Characteristics:

- Final cleanup is performed.
- Activity instance is destroyed permanently.

Example:

User presses the Back button to exit the app.

Activity State Transition Diagram (Text Format) (2M)

```
Running
   |
   v
Paused
   |
   v
Stopped
   |
   v
Destroyed
```

Or when returning:

Stopped → onRestart() → onStart() → onResume()

Summary Table

State	**Visibility**	**User Interaction**	**Lifecycle Method**
Running	Fully visible	Yes	onResume()
Paused	Partially visible	No	onPause()
Stopped	Not visible	No	onStop()
Destroyed	Removed	No	onDestroy()

UNIT – III

Q6(a) Explain the concept of Intents in Android. Differentiate between Explicit and Implicit Intents with suitable examples ***(CO3 – L2 – 7 Marks)***

What is an Intent?(1M)

Intent is a messaging object used to request an action from another app component.
It enables communication between different components such as **Activities, Services, and Broadcast Receivers**.

Intents are mainly used to:

- Start an Activity
- Start or bind a Service
- Deliver a Broadcast

Types of Intents(2M)

There are **two main types**:

1. **Explicit Intent**
2. **Implicit Intent**

1. Explicit Intent(2M)

Definition

An **Explicit Intent** specifies the **exact component name** (class name) to be executed.

Used When:

- Starting an activity within the same application.
- Communicating with a known component.

Example:

```
Intent intent = new Intent(MainActivity.this, SecondActivity.class);
startActivity(intent);
```

Characteristics:

- Component name is specified.
- Mostly used for **intra-app communication**.
- Faster and more secure.

2. Implicit Intent(2M)

Definition

An **Implicit Intent** does **not specify the target component**.

Instead, the system finds a suitable component based on the action and data.

Used When:

- Performing general actions like opening a browser, dialing a number, or sharing content.

Example:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("https://www.google.com"));
startActivity(intent);
```

Characteristics:

- No component name is specified.
- System displays a chooser if multiple apps can handle the request.
- Used for inter-app communication.

Intent Filters

Implicit intents rely on **intent filters** defined in the AndroidManifest.xml.

Example:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

Difference Between Explicit and Implicit Intents

Feature	Explicit Intent	Implicit Intent
Target Component	Known	Unknown
Component Name	Specified	Not specified
Used For	Internal navigation	External app interaction
User Choice	No	Yes (if multiple apps)
Example	Open another activity	Open browser / dialer

Q6 (b) Describe how to create and register a Broadcast Receiver both statically (in manifest) and dynamically (in code) with examples. ***(CO3 – L2 – 7 Marks)***

What is a Broadcast Receiver?(1M)

A **Broadcast Receiver** is an Android component that **listens for system-wide or application-specific broadcast messages** and reacts to them.

Common broadcasts:

- Battery low
- Network change
- SMS received
- Device boot completed

Steps to Create a Broadcast Receiver(4M)

Step 1: Create a BroadcastReceiver class

```
public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Broadcast Received!", Toast.LENGTH_SHORT).show();
    }
}
```

1. Static Registration (Using AndroidManifest.xml)

Definition

In **static registration**, the Broadcast Receiver is declared in the **AndroidManifest.xml** file.

It works even when the app is **not running**.

Steps

Step 1: Create BroadcastReceiver class

(Already shown above)

Step 2: Register in AndroidManifest.xml

```
<receiver android:name=".MyReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

Add Permission (if required)

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

Use Case

✔ Listening for system events

✔ Auto-start on device reboot

2. Dynamic Registration (Using Java Code)

Definition

In **dynamic registration**, the Broadcast Receiver is registered **during runtime** using code and works only while the app is running.

Steps

Step 1: Create BroadcastReceiver

(Same class as before)

Step 2: Register Receiver in Activity

```
MyReceiver myReceiver = new MyReceiver();
IntentFilter filter = new IntentFilter();
filter.addAction(Intent.ACTION_BATTERY_CHANGED);

registerReceiver(myReceiver, filter);
```

Step 3: Unregister Receiver

```
@Override
protected void onDestroy() {
    super.onDestroy();
    unregisterReceiver(myReceiver);
}
```

Difference Between Static and Dynamic Registration (1M)

Feature	Static Registration	Dynamic Registration
Defined In	AndroidManifest.xml	Java code
App Running Required	No	Yes
Lifetime	Always active	Only while app is active
System Events	Supported	Limited
Power Efficiency	Less	More efficient

Example Use Cases(1M)

Scenario	Registration Type
Detect phone reboot	Static
Monitor battery level in app	Dynamic
Receive SMS	Static
Detect network change while app runs	Dynamic

Q7(a) Explain the complete process of creating, saving, and retrieving SharedPreferences in Android with a coding example. ***(CO3 – L2 – 7 Marks)***

What is Shared Preferences?(2M)

Shared Preferences is a lightweight storage mechanism in Android used to store **small amounts of primitive data** in the form of **key–value pairs**.

Used to store:

- User login status
- Username / email
- App settings (theme, language, sound settings)

Data Types Supported:

- String
- int
- float
- boolean
- long

Steps to Use Shared Preferences (5M)

1. Creating Shared Preferences

Syntax:

```
SharedPreferences sharedPreferences = getSharedPreferences("MyPrefs", MODE_PRIVATE);
```

Explanation:

- "MyPrefs" → File name
- MODE_PRIVATE → Accessible only by this app

2. Saving Data into SharedPreferences

Step 1: Get the Editor object

```
SharedPreferences.Editor editor = sharedPreferences.edit();
```

Step 2: Put values

```
editor.putString("username", "Hanuman");
editor.putInt("age", 43);
editor.putBoolean("isLoggedIn", true);
```

Step 3: Save changes

```
editor.apply();   // or editor.commit();
```

✔ apply() – asynchronous (recommended)

✔ commit() – synchronous

3. Retrieving Data from SharedPreferences

```
String username = sharedPreferences.getString("username", "Default User");
int age = sharedPreferences.getInt("age", 0);
boolean isLoggedIn = sharedPreferences.getBoolean("isLoggedIn", false);
```

4. Complete Example Code

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    SharedPreferences sharedPreferences;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Create SharedPreferences
        sharedPreferences = getSharedPreferences("MyPrefs", MODE_PRIVATE);

        // Save data
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString("username", "Hanuman");
        editor.putInt("age", 43);
        editor.putBoolean("isLoggedIn", true);
        editor.apply();

        // Retrieve data
        String name = sharedPreferences.getString("username", "N/A");
        int age = sharedPreferences.getInt("age", 0);
        boolean status = sharedPreferences.getBoolean("isLoggedIn", false);

        Toast.makeText(this,
                "Name: " + name + "\nAge: " + age + "\nLogged In: " + status,
                Toast.LENGTH_LONG).show();
    }
}
```

5. Clearing or Removing Data

Remove a specific key:

```
editor.remove("username");
editor.apply();
```

Clear all data:

```
editor.clear();
editor.apply();
```

6. Modes of Shared Preferences

Mode	Description
MODE_PRIVATE	Default, accessible only to the app
MODE_APPEND	Append data (rarely used)

Q7(b) Write short notes on the following: i) PreferenceFragmentCompatii) getDefaultSharedPreferences() vs getPreferences() iii) Modes in SharedPreferences ***(CO3 – L2 – 7 Marks)***

i) PreferenceFragmentCompat (3M)

PreferenceFragmentCompat is a class used to create a **settings screen** in Android using a **fragment-based approach**.

It is part of the **AndroidX Preference Library** and is backward compatible.

Key Points:

- Used to display app preferences as a list.
- Preferences are automatically saved using **SharedPreferences**.
- Preferred over the old PreferenceActivity.
- Supports modern UI and lifecycle handling.

Example:

```
public class SettingsFragment extends PreferenceFragmentCompat {
    @Override
    public void onCreatePreferences(Bundle savedInstanceState, String rootKey) {
        setPreferencesFromResource(R.xml.preferences, rootKey);
    }
}
```

---

ii) getDefaultSharedPreferences() vs getPreferences() (3M)

Feature	getDefaultSharedPreferences()	getPreferences()
Scope	App-wide	Activity-specific
Access	Can be accessed from any component	Only from the same Activity
File Name	Default system-defined name	Based on Activity name
Use Case	Global app settings	Temporary activity settings

Syntax Examples:

```
// Default SharedPreferences
SharedPreferences prefs =
PreferenceManager.getDefaultSharedPreferences(this);

// Activity-specific preferences
SharedPreferences prefs = getPreferences(MODE_PRIVATE);
```

---

iii) Modes in SharedPreferences(1M)

Modes define access level of preference data.

Mode	Description
MODE_PRIVATE	Data is accessible only within the same app (default).
MODE_APPEND	Appends new data to existing data (rarely used).

Example:

```
SharedPreferences prefs = getSharedPreferences("MyPrefs", MODE_PRIVATE);
```

UNIT – IV

Q8(a) Explain the complete structure of a Content Provider in Android. Write the mandatory methods that must be overridden and show a sample authority declaration. ***(CO4 – L2 – 7 Marks)***

What is a Content Provider?(1M)

A **Content Provider** is one of the four main Android application components used to **manage and share application data** between different applications in a secure way.

It provides a **standard interface** to access structured data such as databases, files, or network data using **URIs (Uniform Resource Identifiers)**.

Structure of a Content Provider(2M)

A Content Provider mainly consists of:

1. **Content URI**
2. **Authority**
3. **MIME Type**
4. **CRUD Methods**
5. **ContentResolver (Client Access)**

Basic Structure of a Content Provider Class (1M)

```
public class MyContentProvider extends ContentProvider {

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection,
                        String selection, String[] selectionArgs,
                        String sortOrder) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }

    @Override
    public int update(Uri uri, ContentValues values,
                      String selection, String[] selectionArgs) {
```

```
        return 0;
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        return 0;
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }
}
```

Mandatory Methods to Override (1M)

Method	Purpose
onCreate()	Initializes the provider
query()	Retrieves data
insert()	Inserts new data
update()	Updates existing data
delete()	Deletes data
getType()	Returns MIME type of data

Explanation of Each Method (2M)

1. onCreate()

- Called when the provider is first created.
- Used to initialize database or resources.

```
@Override
public boolean onCreate() {
    return true;
}
```

2. query()

- Used to retrieve data from the provider.

```
public Cursor query(Uri uri, String[] projection,
                    String selection, String[] selectionArgs,
                    String sortOrder)
```

3. insert()

- Inserts a new record into the data source.

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

4. update()

- Updates existing records.

```
public int update(Uri uri, ContentValues values,
                  String selection, String[] selectionArgs)
```

5. delete()

- Deletes data from the provider.

```
public int delete(Uri uri, String selection, String[] selectionArgs)
```

6. getType()

- Returns the MIME type of the data.

```
public String getType(Uri uri)
```

Content Provider Authority

What is Authority?

- A **unique identifier** for the Content Provider.
- Used in **content URIs**.
- Defined in AndroidManifest.xml.

Example Authority Declaration

In AndroidManifest.xml

```
<provider
  android:name=".MyContentProvider"
  android:authorities="com.example.myapp.provider"
  android:exported="true" />
```

Content URI Format

```
content://authority/path/id
```

Example:

```
content://com.example.myapp.provider/users
```

Accessing Content Provider Using ContentResolver

```
Cursor cursor = getContentResolver().query(
    Uri.parse("content://com.example.myapp.provider/users"),
    null, null, null, null
);
```

Diagram: Content Provider Flow (Text Form)

App → ContentResolver → ContentProvider → Database

Summary Table

Component	Description
ContentProvider	Manages shared data
Authority	Unique identifier
URI	Identifies data
ContentResolver	Access point for clients
CRUD Methods	Data operations

Q8 (b) with a neat diagram and code, explain how to create a bound service and bind it from an Activity using Binder. ***(CO4 – L2 – 7 Marks)***

What is a Bound Service?(1M)

A **Bound Service** allows components (such as Activities) to **bind to the service and interact with it directly** using a client–server interface.

The service exists **only while one or more components are bound to it**.

When to Use a Bound Service

- When the Activity needs **continuous interaction** with the service
- For **data sharing or method calls**
- Example: Music playback control, data synchronization, sensor data

---

Architecture Diagram (1M)

```
+------------------+
|     Activity     |
|------------------|
| bindService()    |
| onServiceConnected()
| call service methods
+--------|---------+
         |
         v
+----------------------+
|   Bound Service      |
|----------------------|
| onBind()             |
| LocalBinder          |
| Business Logic       |
+----------------------+
```

Steps to Create a Bound Service(4M)

Step 1: Create the Service Class

MyBoundService.java

```
public class MyBoundService extends Service {

    private final IBinder binder = new LocalBinder();

    // Binder class
    public class LocalBinder extends Binder {
        MyBoundService getService() {
            return MyBoundService.this;
        }
    }
    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    // Custom method
    public String getMessage() {
        return "Hello from Bound Service!";
    }
}
```

Step 2: Register Service in AndroidManifest.xml

```
<service
    android:name=".MyBoundService"
    android:exported="false" />
```

Step 3: Bind Service from Activity

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    MyBoundService myService;
    boolean isBound = false;

    private ServiceConnection connection = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            MyBoundService.LocalBinder binder =
                    (MyBoundService.LocalBinder) service;
            myService = binder.getService();
            isBound = true;

            // Call service method
            String msg = myService.getMessage();
            Toast.makeText(MainActivity.this, msg, Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            isBound = false;
        }
    };

    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, MyBoundService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (isBound) {
            unbindService(connection);
            isBound = false;
        }
    }
}
```

Lifecycle of a Bound Service (1M)

Method	Description
onBind()	Called when client binds
onUnbind()	Called when all clients unbind
onDestroy()	Called when service is destroyed

Key Points

- Bound services run **only while clients are bound**.
- Communication is done using **Binder**.
- Suitable for **client-server interaction inside the app**.
- Automatically destroyed when no client remains bound.

Comparison: Started vs Bound Service

Feature	Started Service	Bound Service
Lifetime	Independent	Depends on clients
Interaction	Limited	Direct method calls
Use Case	Background tasks	Client–server interaction

Q9(a) Explain SQLiteOpenHelper class. Write a complete code to create a database with one table "Employee" having columns id, name, salary, and department. ***(CO4 – L3 – 7 Marks)***

What is SQLiteOpenHelper? (1M)

SQLiteOpenHelper is an **abstract helper class** in Android used to **create, manage, and upgrade SQLite databases**.

It simplifies database management by automatically handling:

- Database creation
- Version management
- Table upgrades

Why SQLiteOpenHelper is used

- Avoids manual database handling
- Automatically calls lifecycle methods
- Efficient for structured local data storage

Important Methods of SQLiteOpenHelper (2M)

Method	Purpose
onCreate()	Called when database is created for the first time
onUpgrade()	Called when database version changes
getWritableDatabase()	Opens database for read/write
getReadableDatabase()	Opens database for read-only

Steps to Create SQLite Database (4M)

Step 1: Create Database Helper Class

EmployeeDBHelper.java

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class EmployeeDBHelper extends SQLiteOpenHelper {

    // Database details
    private static final String DATABASE_NAME = "EmployeeDB.db";
    private static final int DATABASE_VERSION = 1;

    // Table name
    public static final String TABLE_NAME = "Employee";

    // Column names
    public static final String COLUMN_ID = "id";
    public static final String COLUMN_NAME = "name";
    public static final String COLUMN_SALARY = "salary";
    public static final String COLUMN_DEPARTMENT = "department";

    // SQL query to create table
    private static final String CREATE_TABLE =
            "CREATE TABLE " + TABLE_NAME + " (" +
                    COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    COLUMN_NAME + " TEXT, " +
                    COLUMN_SALARY + " REAL, " +
                    COLUMN_DEPARTMENT + " TEXT );";

    // Constructor
    public EmployeeDBHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    // Called when DB is created for the first time
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_TABLE);
    }

    // Called when DB version changes
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

Step 2: Using the Database (Insert Example)

```
EmployeeDBHelper dbHelper = new EmployeeDBHelper(this);
SQLiteDatabase db = dbHelper.getWritableDatabase();

ContentValues values = new ContentValues();
values.put("name", "Ravi Kumar");
values.put("salary", 45000);
values.put("department", "IT");

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

Step 3: Reading Data from Database

```
SQLiteDatabase db = dbHelper.getReadableDatabase();

Cursor cursor = db.rawQuery("SELECT * FROM Employee", null);

while (cursor.moveToNext()) {
    int id = cursor.getInt(0);
    String name = cursor.getString(1);
    double salary = cursor.getDouble(2);
    String dept = cursor.getString(3);
}
cursor.close();
```

Database Table Structure

Column	Data Type
id	INTEGER (Primary Key)
name	TEXT
salary	REAL
department	TEXT

Flow Diagram

App

↓

SQLiteOpenHelper

↓

onCreate() → Create Table

↓

Insert / Read / Update / Delete

↓

SQLite Database

Advantages of SQLiteOpenHelper

✔ Handles database creation automatically

✔ Manages version upgrades

✔ Reduces boilerplate SQL code

✔ Efficient for local storage

Q9(b) Describe how to use Content Values and Cursor to insert and retrieve data from SQLite database with example code. ***(CO4 – L2 – 7 Marks)***

Introduction (2M))

In Android, **Content Values** and **Cursor** are commonly used to interact with an SQLite database.

- **Content Values** → Used to **insert or update data**
- **Cursor** → Used to **retrieve data**

They work together with **SQLiteDatabase**.

1. ContentValues

What is ContentValues?

ContentValues is a key–value pair structure used to store column values before inserting or updating records in a database.

Why use ContentValues?

- Prevents SQL injection
- Simplifies insert/update operations
- Improves code readability

2. Cursor

What is Cursor?

A **Cursor** is an interface that provides **read access** to the result set returned by a database query.

It allows moving through rows one by one.

Database Table Used (1M)

Table Name: Employee

Column	Type
id	INTEGER (Primary Key)
name	TEXT
salary	REAL
department	TEXT

Step 1: Create SQLiteOpenHelper Class (1M)

```
public class EmployeeDBHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "EmployeeDB";
    private static final int DB_VERSION = 1;
    public static final String TABLE_NAME = "Employee";
    public static final String COL_ID = "id";
    public static final String COL_NAME = "name";
    public static final String COL_SALARY = "salary";
    public static final String COL_DEPT = "department";
    public EmployeeDBHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        String query = "CREATE TABLE " + TABLE_NAME + " (" +
                COL_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                COL_NAME + " TEXT, " +
                COL_SALARY + " REAL, " +
                COL_DEPT + " TEXT)";
        db.execSQL(query);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

Step 2: Insert Data Using ContentValues (1M)

```
EmployeeDBHelper dbHelper = new EmployeeDBHelper(this);
SQLiteDatabase db = dbHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("name", "Ramesh");
values.put("salary", 55000);
values.put("department", "IT");
long result = db.insert("Employee", null, values);
if (result != -1) {
    Toast.makeText(this, "Data Inserted", Toast.LENGTH_SHORT).show();
}
```

Step 3: Retrieve Data Using Cursor(1M)

```
SQLiteDatabase db = dbHelper.getReadableDatabase();

Cursor cursor = db.rawQuery("SELECT * FROM Employee", null);

while (cursor.moveToNext()) {
    int id = cursor.getInt(cursor.getColumnIndexOrThrow("id"));
    String name = cursor.getString(cursor.getColumnIndexOrThrow("name"));
    double salary = cursor.getDouble(cursor.getColumnIndexOrThrow("salary"));
    String dept = cursor.getString(cursor.getColumnIndexOrThrow("department"));

    Log.d("EMPLOYEE", id + " " + name + " " + salary + " " + dept);
}

cursor.close();
```

Alternative: Using query() Method (1M)

```
Cursor cursor = db.query(
        "Employee",
        null,
        null,
        null,
        null,
        null,
        null
);
```

Flow Diagram

App

↓

Content Values → Insert Data

↓

SQLite Database

↓

Cursor → Retrieve Data

↓

Display Output

Advantages

✔ Safe and structured data handling

✔ Easy insertion and retrieval

✔ Reduces SQL errors

Summary Table

Component	**Purpose**
ContentValues	Insert / update data
Cursor	Read query results
SQLiteDatabase	Database access
SQLiteOpenHelper	Database management

	Name	**Signature**	**Date**
External Valuator			
Internal Valuator			

HOD – IT