Android SQLite Database Tutorial - CRUD Operations

by Kapil - Monday, March 21, 2016

http://www.androidtutorialpoint.com/storage/android-sqlite-database-tutorial/

YouTube Video

Android SQLite Introduction

In Android SDK there are three main ways to store data.

1. Shared Preferences
2. File Storage
3. SQLiteDatabase

(adsbygoogle = window.adsbygoogle || []).push({});
DataBase is one of the ways of storing data. However we should prefer using a DB only if the data we are dealing with is structured data like Employee details of an organization.

Android SDK comes with built in support for SQLite Database. Android SQLite Database is a light weight database mainly useful for embedded applications, it reads and writes directly to disk files.

In this tutorial we will be creating Employees Management System needed for every company. We will create a simple employees database which will be used for storing employee information. We will perform the SQLite CRUD(Create, Read, Update and Delete) operation.

All the classes and interfaces that are required to work with Android SQLite Database are available in the package,**android.database.sqlite**. The SQLite Database class represents a file on the Android device. You can control the name of the file which contains the DB and it will be a single file rather than multiple scattered files.

Let's review some of the important classes that we'll use when working with an SQLite database.

Class	Description
SQLiteDatabase	represents the Android SQLite database
SQLiteOpenHelper	a helper class that manages the Android SQLite DB
SQLiteQuery	represents an Android SQLite DB query
SQLite Statement	represents an Android SQLite statement
SQLiteCursor	exposes the results from a query, use to iterate through the results from the query.
SQLiteQueryBuilder	a helper class to build and manage queries
SQLiteException	represents an Android SQLite Exceptions

We will use the following structure to create the **SQLite** database in android. For now, we will be creating a single table named the employee.

Table Name : employee

Field	Type	Key
empId	INT	PRIMARY
firstname	TEXT	
lastname	TEXT	
gender	TEXT	
hiredate	TEXT	
dept	TEXT	

The primary key column will be an integer and it will auto-increment and then there are five text columns.

Pre-requisites for Creating Android SQLite Database App

1) Android Studio installed on your PC (Unix or Windows). You can learn how to install it here .
2) A real time android device (Smartphone or Tablet) configured with Android Studio. .
3) A basic knowledge of Android lifecycle and different classes & functions used in Android Studio.

Create new Android Project and name it EmployeeManagement System. Next follow the below steps.

Creating a Model Class

Since we are dealing with employee information let's create Employee. class which will act as model for our application. Create a package named Model, create a class Employee. and put the following code.

Employee.

```
[]
package com.androidtutorialpoint.employeemanagementsystem.Model;

public class Employee {
private long empId;
private String firstname;
private String lastname;
private String gender;
private String hiredate;
private String dept;

  public Employee(long empId, String firstname, String lastname, String gender, String hiredate, String
dept){
this.empId = empId;
this.firstname = firstname;
this.lastname = lastname;
this.gender = gender;
this.hiredate = hiredate;
this.dept = dept;

  }

  public Employee(){

  }

  public long getEmpId() {
return empId;
}

  public void setEmpId(long empId) {
this.empId = empId;
}

  public String getFirstname() {
return firstname;
}

  public void setFirstname(String firstname) {
this.firstname = firstname;
}

  public String getLastname() {
return lastname;
}

  public void setLastname(String lastname) {
this.lastname = lastname;
}

  public String getGender() {
return gender;
}

  public void setGender(String gender) {
this.gender = gender;
}

  public String getHiredate() {
return hiredate;
}
```

```
  public void setHiredate(String hiredate) {
this.hiredate = hiredate;
}

  public String getDept() {
return dept;
}

  public void setDept(String dept) {
this.dept = dept;
}

  public String toString(){
return "Emp id: "+getEmpId()+ "\n" +
"Name: "+getFirstname() + " " + getLastname() + "\n" +
"Hire Date: "+getHiredate() + "\n" +
"Department : "+getDept();

  }
}
[/]
```

We are Overriding the `toString()` method in order to get a proper output when we print an Employee.

Creating Employee DB

We will create a class **EmployeeDBHandler**, a helper class that manages the Android SQLite Database EmployeeDB. It will extend `SQLiteOpenHelper` class. Create a new package **DB**. In that package create a new class, name it **EmployeeDBHandler**
and put the following code. We will explain everything in moment.

EmployerDBHandler.

```
[]
package com.androidtutorialpoint.employeemanagementsystem.DB;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class EmployeeDBHandler extends SQLiteOpenHelper{

  private static final String DATABASE_NAME = "employees.db";
private static final int DATABASE_VERSION = 1;
```

```
  public static final String TABLE_EMPLOYEES = "employees";
public static final String COLUMN_ID = "empId";
public static final String COLUMN_FIRST_NAME = "firstname";
public static final String COLUMN_LAST_NAME = "lastname";
public static final String COLUMN_GENDER = "gender";
public static final String COLUMN_HIRE_DATE= "hiredata";
public static final String COLUMN_DEPT= "dept";

  private static final String TABLE_CREATE =
"CREATE TABLE " + TABLE_EMPLOYEES + " (" +
COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
COLUMN_FIRST_NAME + " TEXT, " +
COLUMN_LAST_NAME + " TEXT, " +
COLUMN_GENDER + " TEXT, " +
COLUMN_HIRE_DATE + " NUMERIC, " +
COLUMN_DEPT + " TEXT " +
")";

  public EmployeeDBHandler(Context context){
super(context,DATABASE_NAME,null,DATABASE_VERSION);
}

  @Override
public void onCreate(SQLiteDatabase db) {
db.execSQL(TABLE_CREATE);

  }

  @Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

  db.execSQL("DROP TABLE IF EXISTS "+TABLE_EMPLOYEES);
db.execSQL(TABLE_CREATE);
}
}
[/]
```

We have defined some constants. First there's constant for the name of the database, then there are constants for the version and columns we will be creating and finally, there's a constant named Table Create that defines an SQL statement to create **employee** table. The version for the Android SQLite Database is required, it starts from 1 and every time DB structure is changed, we increment it by 1. There is constructor that receives the context and calls the super class constructor along with the db version.

The `onCreate()` and `onUpgrade()` methods are be called by the Android OS. `onCreate()` method is called in case Android SQLite Database doesn't exist and we want to get a connection to the database. However if the database already exists, but the database version is changed the onUpgrade method is invoked.

In the `onCreate()`method, we add the code to create the employee table. The **execSQL()** method will execute the sqlite create table query.

In the `onUpgrade()` method, we simply drop the existing table, and create the new with updated version number. After dropping we recreate it using the same query as in **onCreate()**

Create, Read, Update and Delete operation for Employees

Create a class EmployeeOperations. under DB package and put the following code.

EmployeeOperations.

```
[]
package com.androidtutorialpoint.employeemanagementsystem.DB;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

import com.androidtutorialpoint.employeemanagementsystem.Model.Employee;
import .util.ArrayList;
import .util.List;

public class EmployeeOperations {
public static final String LOGTAG = "EMP_MNGMNT_SYS";

  SQLiteOpenHelper dbhandler;
SQLiteDatabase database;

  private static final String[] allColumns = {
EmployeeDBHandler.COLUMN_ID,
EmployeeDBHandler.COLUMN_FIRST_NAME,
EmployeeDBHandler.COLUMN_LAST_NAME,
EmployeeDBHandler.COLUMN_GENDER,
EmployeeDBHandler.COLUMN_HIRE_DATE,
EmployeeDBHandler.COLUMN_DEPT

  };

  public EmployeeOperations(Context context){
dbhandler = new EmployeeDBHandler(context);
```

```
}

  public void open(){
Log.i(LOGTAG,"Database Opened");
database = dbhandler.getWritableDatabase();

  }
public void close(){
Log.i(LOGTAG, "Database Closed");
dbhandler.close();

  }
public Employee addEmployee(Employee Employee){
ContentValues values = new ContentValues();
values.put(EmployeeDBHandler.COLUMN_FIRST_NAME,Employee.getFirstname());
values.put(EmployeeDBHandler.COLUMN_LAST_NAME,Employee.getLastname());
values.put(EmployeeDBHandler.COLUMN_GENDER, Employee.getGender());
values.put(EmployeeDBHandler.COLUMN_HIRE_DATE, Employee.getHiredate());
values.put(EmployeeDBHandler.COLUMN_DEPT, Employee.getDept());
long insertid = database.insert(EmployeeDBHandler.TABLE_EMPLOYEES,null,values);
Employee.setEmpId(insertid);
return Employee;

  }

  // Getting single Employee
public Employee getEmployee(long id) {

  Cursor cursor = database.query(EmployeeDBHandler.TABLE_EMPLOYEES,allColumns,EmployeeDB
Handler.COLUMN_ID + "=?",new String[]{String.valueOf(id)},null,null, null, null);
if (cursor != null)
cursor.moveToFirst();

  Employee e = new Employee(Long.parseLong(cursor.getString(0)),cursor.getString(1),cursor.getString(
2),cursor.getString(3),cursor.getString(4),cursor.getString(5));
// return Employee
return e;
}

  public List<Employee> getAllEmployees() {

  Cursor cursor = database.query(EmployeeDBHandler.TABLE_EMPLOYEES,allColumns,null,null,null,
null, null);

  List<Employee> employees = new ArrayList<>();
if(cursor.getCount() > 0){
```

```
while(cursor.moveToNext()){
Employee employee = new Employee();
employee.setEmpId(cursor.getLong(cursor.getColumnIndex(EmployeeDBHandler.COLUMN_ID)));
employee.setFirstname(cursor.getString(cursor.getColumnIndex(EmployeeDBHandler.COLUMN_FIRS
T_NAME)));
employee.setLastname(cursor.getString(cursor.getColumnIndex(EmployeeDBHandler.COLUMN_LAST
_NAME)));
employee.setGender(cursor.getString(cursor.getColumnIndex(EmployeeDBHandler.COLUMN_GENDE
R)));
employee.setHiredate(cursor.getString(cursor.getColumnIndex(EmployeeDBHandler.COLUMN_HIRE_
DATE)));
employee.setDept(cursor.getString(cursor.getColumnIndex(EmployeeDBHandler.COLUMN_DEPT)));
employees.add(employee);
}
}
// return All Employees
return employees;
}

  // Updating Employee
public int updateEmployee(Employee employee) {

  ContentValues values = new ContentValues();
values.put(EmployeeDBHandler.COLUMN_FIRST_NAME, employee.getFirstname());
values.put(EmployeeDBHandler.COLUMN_LAST_NAME, employee.getLastname());
values.put(EmployeeDBHandler.COLUMN_GENDER, employee.getGender());
values.put(EmployeeDBHandler.COLUMN_HIRE_DATE, employee.getHiredate());
values.put(EmployeeDBHandler.COLUMN_DEPT, employee.getDept());

  // updating row
return database.update(EmployeeDBHandler.TABLE_EMPLOYEES, values,
EmployeeDBHandler.COLUMN_ID + "=?",new String[] { String.valueOf(employee.getEmpId())});
}

  // Deleting Employee
public void removeEmployee(Employee employee) {

  database.delete(EmployeeDBHandler.TABLE_EMPLOYEES, EmployeeDBHandler.COLUMN_ID +
"=" + employee.getEmpId(), null);
}

}
[/]
```

This class consists of methods to perform all the four operations(CRUD) on the employee table of our Android SQLite Database. We have a constuctor that will instantiate the database handler. The `open()`

method is used to get a writeable database object and `close()` do disconnect it. Using these two methods we can open or close the database connection anytime from anywhere.

Then we have the method `addEmployee()` for android sqlite insert. It will add an employee to the Android SQLite database. We put the values of all the required fields into **ContentValues** object and then insert into DB. The insert operation will return the autogenerated **empId**.

Next we have, `getEmployee()` function that takes in empId and returns an **Employee** object. This is sqlite SELECT query, here we get the result of query into a cursor and then return the first row.

The `getAllEmployees()` function is similar to `getEmployee()` however it returns a list of **Employees**.

The **updateEmployee** function is similar to **addEmployee** here we put new values in the **ContentValues** object and then update it in the database. This represents sqlite update query.

At last we have **removeEmployee()** function for sqlite delete query. It is used to remove an employee based upon the given employee object.

All the queries are really simple, so no neet to explain them in detail. Please leave comments below if you don't understand anything.

Creating a User Interace

We will be creating a user interface for our **Employee Management System**. On completion it will look like as shown below.

Open the layout for the **MainActivity** and add the following code.

activity_main.

[]

```
<? version="1.0" encoding="utf-8"?>
<LinearLayout ns:android="http://schemas.android.com/apk/res/android"
ns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:orientation="vertical"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity"
android:gravity="center_horizontal">

  <TextView android:text="@string/choose_menu" android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:paddingTop="@dimen/header_margin"
android:paddingRight="@dimen/header_margin"
android:paddingLeft="@dimen/header_margin"
android:paddingBottom="@dimen/header_bottom_margin"
android:textSize="@dimen/text_size"
android:fontFamily="sans-serif"
android:textStyle="bold"
android:gravity="clip_horizontal" />
<Button
android:id="@+id/button_add_employee"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/add_employee"/>

  <Button
android:id="@+id/button_edit_employee"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/update_employee"/>

  <Button
android:id="@+id/button_delete_employee"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/remove_employee"/>

  <Button
android:id="@+id/button_view_employees"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/view_all_employees"/>
</LinearLayout>

[/]
```

The layout is really simple we have a header **TextView** and four buttons, each for adding,updating,removing a single employee and one listing all employees.

Next, open **MainActivity.** and add the following code. We will explain everything in a moment.

MainActivity.

```
[]
package com.androidtutorialpoint.employeemanagementsystem;
```

```
import android.content.DialogInterface;
import android.content.Intent;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.androidtutorialpoint.employeemanagementsystem.DB.EmployeeOperations;

public class MainActivity extends AppCompatActivity{

  private Button addEmployeeButton;
private Button editEmployeeButton;
private Button deleteEmployeeButton;
private Button viewAllEmployeeButton;
private EmployeeOperations employeeOps;
private static final String EXTRA_EMP_ID = "com.androidtutorialpoint.empId";
private static final String EXTRA_ADD_UPDATE = "com.androidtutorialpoint.add_update";
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

  setContentView(R.layout.activity_main);
addEmployeeButton = (Button) findViewById(R.id.button_add_employee);
editEmployeeButton = (Button) findViewById(R.id.button_edit_employee);
deleteEmployeeButton = (Button) findViewById(R.id.button_delete_employee);
viewAllEmployeeButton = (Button)findViewById(R.id.button_view_employees);

  addEmployeeButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent i = new Intent(MainActivity.this,AddUpdateEmployee.class);
i.putExtra(EXTRA_ADD_UPDATE, "Add");
startActivity(i);
}
});
editEmployeeButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
getEmpIdAndUpdateEmp();
}
});
deleteEmployeeButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
getEmpIdAndRemoveEmp();
}
});
viewAllEmployeeButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent i = new Intent(MainActivity.this, ViewAllEmployees.class);
startActivity(i);
}
});

  }

  @Override
public boolean onCreateOptionsMenu(Menu menu) {
getMenuInflater().inflate(R.menu.employee_menu, menu);
return true;
}

  @Override
public boolean onOptionsItemSelected(MenuItem item) {
int id = item.getItemId();
if (id == R.id.menu_item_settings) {
return true;
}
return super.onOptionsItemSelected(item);
}

  public void getEmpIdAndUpdateEmp(){

  LayoutInflater li = LayoutInflater.from(this);
View getEmpIdView = li.inflate(R.layout.dialog_get_emp_id, null);

  AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
// set dialog_get_emp_id. to alertdialog builder
alertDialogBuilder.setView(getEmpIdView);

  final EditText userInput = (EditText) getEmpIdView.findViewById(R.id.editTextDialogUserInput);

  // set dialog message
alertDialogBuilder
```

```
.setCancelable(false)
.setPositiveButton("OK",new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog,int id) {
// get user input and set it to result
// edit text
Intent i = new Intent(MainActivity.this,AddUpdateEmployee.class);
i.putExtra(EXTRA_ADD_UPDATE, "Update");
i.putExtra(EXTRA_EMP_ID, Long.parseLong(userInput.getText().toString()));
startActivity(i);
}
}).create()
.show();

  }

  public void getEmpIdAndRemoveEmp(){

  LayoutInflater li = LayoutInflater.from(this);
View getEmpIdView = li.inflate(R.layout.dialog_get_emp_id, null);

  AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
// set dialog_get_emp_id. to alertdialog builder
alertDialogBuilder.setView(getEmpIdView);

  final EditText userInput = (EditText) getEmpIdView.findViewById(R.id.editTextDialogUserInput);

  // set dialog message
alertDialogBuilder
.setCancelable(false)
.setPositiveButton("OK",new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog,int id) {
// get user input and set it to result
// edit text
employeeOps = new EmployeeOperations(MainActivity.this);
employeeOps.removeEmployee(employeeOps.getEmployee(Long.parseLong(userInput.getText().toStrin
g())));
Toast t = Toast.makeText(MainActivity.this,"Employee removed
successfully!",Toast.LENGTH_SHORT);
t.show();
}
}).create()
.show();

  }

  @Override
```

```
protected void onResume() {
super.onResume();
employeeOps.open();
}

  @Override
protected void onPause() {
super.onPause();
employeeOps.close();

  }
}
[/]
```

- First, we define two constant we will be using to as EXTRA while starting the **addUpdateActivity**

  ```
  []
  private static final String EXTRA_EMP_ID = "com.androidtutorialpoint.empId";

  private static final String EXTRA_ADD_UPDATE = "com.androidtutorialpoint.add_update";
  [/]
  ```

- Next, we `setOnClickListener()` for the addEmployeeButton. Here we invoke **AddUpdateEmployee** activity, another activity where we are getting employee information from the user, we will talk about this activity in a moment. Please Note that we are using a common activity to Add as well as update employee information, so we are passing an extra **EXTRA_ADD_UPDATE**. This extra will have a value "Add" if we want to add a new employee, and a value "Update" when we click on Update employee button.
- Similarly, we `setOnClickListener()` for the editEmployeeButton, Here we want the user to input his employee Id. So we are calling `getEmpIdAndUpdateEmp()` function. Let's review the function code. We will using an AlertDialog to get user input, so we have to create a layout for aour alertDialog. Create a new layout file and name it **dialog_get_emp_id.** and put the folowing code.The alert Dialog will look like this.

  Please refer to the following tutorial to read more about Android Alert Dialogs Tutorial

  ```
  []
  <? version="1.0" encoding="utf-8"?>
  <LinearLayout ns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/layout_root"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical"
  android:padding="@dimen/view_padding" >
  ```

```
  <TextView
android:id="@+id/text_view_emp_id"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/enter_employee_id"
android:layout_gravity="center"
android:textSize="@dimen/text_size"
android:textStyle="bold" />

  <EditText
android:id="@+id/editTextDialogUserInput"
android:layout_width="match_parent"
android:layout_height="wrap_content"/>

</LinearLayout>
[/]
```

We are using a LinearLayout with a TextView and an EditText to take employee Id from user. Then we are inflating this layout and creating an `AlertDialog.Builder` object and when user presses "OK", we call the **AddUpdateEmployee** class and pass in the EXTRA_ADD_UPDATE as "Update" so that it works in Update Mode and then pass empId in the EXTRA_EMP_ID.

- For the removeEmployeeButton we have created a similar function **getEmpIdAndRemoveEmp()**, however instead of calling any activity we are simply creating a employeeOperations object and then calling **removeEmployee** and pass in the **Employee** object.
- Then we set **setOnClickListener** for the viewAllEmployeeButton and call another activity **ViewAllEmployees** which will list all the employees. We will discuss about this activity in a while.
- At last we are overriding the `onResume()` and `onPause()` method for the MainActivity.

```
[]

@Override
protected void onResume() {
super.onResume();
employeeOps.open();
}

  @Override
protected void onPause() {
super.onPause();
employeeOps.close();

  }
[/]
```

These are Activity lifecycle methods which are called by Android SDK automatically.When the activity appears on the screen, it's `onResume()` method is called. So we open a connection to database in this method so that we always have a writable database handy. Similarly, we will close the database connection explicitly in case the activity pauses, which is when the `onPause()` is called.

Add or Update Employe

As discussed we have created a separate Activity to add or update an employee, create a new layout file named **activity_add_update_employee.** and put the following code. The layout will look like this on the device.

```
[]
<? version="1.0" encoding="utf-8"?>
<LinearLayout ns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical" android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="30dp">
<EditText
android:id="@+id/edit_text_first_name"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="@string/add_first_name"
android:padding="@dimen/view_padding"
android:textSize="@dimen/add_edit_textsize"/>
<EditText
android:id="@+id/edit_text_last_name"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="@string/add_last_name"
android:padding="@dimen/view_padding"
android:textSize="@dimen/add_edit_textsize"/>
<LinearLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:gravity="center_vertical"
android:orientation="horizontal"
android:padding="@dimen/view_padding">

  <TextView
android:id="@+id/text_view_gender"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
android:text="@string/gender"
android:textSize="@dimen/add_edit_textsize"
android:paddingRight="25dp"/>
<RadioGroup
android:id="@+id/radio_gender"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:orientation="horizontal"
android:paddingRight="25dp"
>

  <RadioButton
android:id="@+id/radio_male"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/string_male"
android:textSize="@dimen/add_edit_textsize"
android:checked="true"/>

  <RadioButton
android:id="@+id/radio_female"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/string_female"
android:textSize="@dimen/add_edit_textsize"/>

  </RadioGroup>

  </LinearLayout>
<LinearLayout
android:orientation="horizontal"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:paddingBottom="@dimen/view_padding">

  <EditText
android:id="@+id/edit_text_hire_date"
android:hint="@string/add_date"
android:layout_width="wrap_content"
android:layout_weight="8"
android:layout_height="wrap_content"
android:gravity="left" />
<ImageView
android:id="@+id/image_view_hire_date"
android:src="@android:drawable/ic_menu_my_calendar"
android:layout_weight="5"
```

```
android:layout_width="wrap_content"
android:layout_height="match_parent" />
</LinearLayout>
<EditText
android:id="@+id/edit_text_dept"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="@string/add_dept"
/>
<Button
android:id="@+id/button_add_update_employee"
android:layout_width="@dimen/button_width"
android:layout_height="@dimen/button_height"
android:layout_marginTop="@dimen/button_padding"
android:text="@string/add_employee"
android:layout_gravity="center"/>
</LinearLayout>
[/]
```

In the layout we have **EditText** for FirstName, LastName, HireDate and Department. For Gender we are using a RadioGroup and for the HireDate the user can even click on the calender icon to select from a date picker.

To add the functionality, create a new class and name it **AddUpdateEmployee** and put the following code in it.

AddUpdateEmployee.

```
[]
package com.androidtutorialpoint.employeemanagementsystem;

import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.FragmentManager;
import android.support.v7.app.AppCompatActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

import com.androidtutorialpoint.employeemanagementsystem.Model.Employee;
import com.androidtutorialpoint.employeemanagementsystem.DB.EmployeeOperations;

import .text.SimpleDateFormat;
import .util.Date;

public class AddUpdateEmployee extends AppCompatActivity implements
DatePickerFragment.DateDialogListener{

  private static final String EXTRA_EMP_ID = "com.androidtutorialpoint.empId";
private static final String EXTRA_ADD_UPDATE = "com.androidtutorialpoint.add_update";
private static final String DIALOG_DATE = "DialogDate";
private ImageView calendarImage;
private RadioGroup radioGroup;
private RadioButton maleRadioButton,femaleRadioButton;
private EditText firstNameEditText;
private EditText lastNameEditText;
private EditText deptEditText;
private EditText hireDateEditText;
private Button addUpdateButton;
private Employee newEmployee;
private Employee oldEmployee;
private String mode;
private long empId;
private EmployeeOperations employeeData;

  @Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_add_update_employee);
newEmployee = new Employee();
oldEmployee = new Employee();
firstNameEditText = (EditText)findViewById(R.id.edit_text_first_name);
lastNameEditText = (EditText)findViewById(R.id.edit_text_last_name);
hireDateEditText = (EditText) findViewById(R.id.edit_text_hire_date);
radioGroup = (RadioGroup) findViewById(R.id.radio_gender);
maleRadioButton = (RadioButton) findViewById(R.id.radio_male);
femaleRadioButton = (RadioButton) findViewById(R.id.radio_female);
calendarImage = (ImageView)findViewById(R.id.image_view_hire_date);
deptEditText = (EditText)findViewById(R.id.edit_text_dept);
addUpdateButton = (Button)findViewById(R.id.button_add_update_employee);
employeeData = new EmployeeOperations(this);
employeeData.open();

  mode = getIntent().getStringExtra(EXTRA_ADD_UPDATE);
if(mode.equals("Update")){
 addUpdateButton.setText("Update Employee");
empId = getIntent().getLongExtra(EXTRA_EMP_ID,0);

 initializeEmployee(empId);

 }

 radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {

 @Override
public void onCheckedChanged(RadioGroup group, int checkedId) {
// find which radio button is selected
if (checkedId == R.id.radio_male) {
newEmployee.setGender("M");
if(mode.equals("Update")){
oldEmployee.setGender("M");
}
} else if (checkedId == R.id.radio_female) {
newEmployee.setGender("F");
if(mode.equals("Update")){
oldEmployee.setGender("F");
}

 }
}

 });

 calendarImage.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {

 FragmentManager manager = getSupportFragmentManager();
DatePickerFragment dialog = new DatePickerFragment();
dialog.show(manager, DIALOG_DATE);
}
});

 addUpdateButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {

 if(mode.equals("Add")) {
newEmployee.setFirstname(firstNameEditText.getText().toString());
newEmployee.setLastname(lastNameEditText.getText().toString());
newEmployee.setHiredate(hireDateEditText.getText().toString());
newEmployee.setDept(deptEditText.getText().toString());
employeeData.addEmployee(newEmployee);
Toast t = Toast.makeText(AddUpdateEmployee.this, "Employee "+ newEmployee.getFirstname() + "has
been added successfully !", Toast.LENGTH_SHORT);
t.show();
Intent i = new Intent(AddUpdateEmployee.this,MainActivity.class);
startActivity(i);
}else {
oldEmployee.setFirstname(firstNameEditText.getText().toString());
oldEmployee.setLastname(lastNameEditText.getText().toString());
oldEmployee.setHiredate(hireDateEditText.getText().toString());
oldEmployee.setDept(deptEditText.getText().toString());
employeeData.updateEmployee(oldEmployee);
Toast t = Toast.makeText(AddUpdateEmployee.this, "Employee "+ oldEmployee.getFirstname() + " has
been updated successfully !", Toast.LENGTH_SHORT);
t.show();
Intent i = new Intent(AddUpdateEmployee.this,MainActivity.class);
startActivity(i);

  }

  }
});

  }

  private void initializeEmployee(long empId) {
oldEmployee = employeeData.getEmployee(empId);
firstNameEditText.setText(oldEmployee.getFirstname());
lastNameEditText.setText(oldEmployee.getLastname());
hireDateEditText.setText(oldEmployee.getHiredate());
radioGroup.check(oldEmployee.getGender().equals("M") ? R.id.radio_male : R.id.radio_female);
deptEditText.setText(oldEmployee.getDept());
}

  @Override
public void onFinishDialog(Date date) {
hireDateEditText.setText(formatDate(date));

  }

  public String formatDate(Date date) {
SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
String hireDate = sdf.format(date);
return hireDate;
}
```

}

[/]

Let's quickly review the activity code.

- Here first we declared all the EXTRA's which we will be needed to get info from the calling activity.
- In the `onCreate` method we reference the **EditText**'s **RadioGroup**,`RadioButton` and all other views.

  Then we get the mode information from the calling activity, as we discussed earlier mode will be "Add" if we want to add a new employee and "Edit" if want to update an existing employee.

  In case the mode is Update we have to initialize the form with the existing data of the employee and get the empId from the calling Activity.
- Then we set the **onCheckedChangeListener** on our radioGroup and set the employee's gender accordingly.
- The user can enter the hireDate or select by clicking on the caledar image. We have created a `DatePickerFragment` for this purpose. Create a new class named **DatePickerFragment** and put the following code.

  []
  ```
  package com.androidtutorialpoint.employeemanagementsystem;

  import android.app.Activity;
  import android.app.AlertDialog;
  import android.app.Dialog;
  import android.content.DialogInterface;
  import android.content.Intent;
  import android.os.Bundle;
  import android.support.v4.app.DialogFragment;
  import android.view.LayoutInflater;
  import android.view.View;
  import android.widget.DatePicker;

  import .lang.reflect.AccessibleObject;
  import .util.Date;
  import .util.GregorianCalendar;

  public class DatePickerFragment extends DialogFragment {

    private DatePicker datePicker;
  private static final int REQUEST_DATE = 1;

    public static final String EXTRA_DATE =
  ```

```
"com.androidtutorialpoint.date";

  public interface DateDialogListener {
void onFinishDialog(Date date);
}
@Override
public Dialog onCreateDialog(Bundle savedInstanceState){

  View v = LayoutInflater.from(getActivity())
.inflate(R.layout.dialog_date,null);

  datePicker = (DatePicker) v.findViewById(R.id.dialog_date_date_picker);

  return new android.support.v7.app.AlertDialog.Builder(getActivity())
.setView(v)
.setTitle(R.string.date_picker_title)
.setPositiveButton(android.R.string.ok,
new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
int year = datePicker.getYear();
int mon = datePicker.getMonth();
int day = datePicker.getDayOfMonth();
Date date = new GregorianCalendar(year,mon,day).getTime();
DateDialogListener activity = (DateDialogListener) getActivity();
activity.onFinishDialog(date);
dismiss();
}
})
.create();
}

}

[/]
```

Please note that the **AddUpdateEmployee** Activity implements `DatePickerFragment.DateDialogListener` which is used to pass the selected date to the **hireDateEditText**

- Returning back to discussion of **AddUpdateEmployee** Activity, we finally `setOnClickListener()` for the addUpdateButton. In case the mode equals "Add" we are setting attributes for the newEmployee, and then adding the employee to the DB. In case of "Update" mode we are setting the attributes to the oldEmployee object and then updating the employee. In both the cases we are finally displaying a success toast and then going back to our **MainActivity**.

View All Employees

Finally we come to the ViewAllEmployees Activity. For that create a new layout file named **activity_view_all_employees.** and put the following code.

activity_view_all_employee.

```
[]
<? version="1.0" encoding="utf-8"?>
<LinearLayout ns:android="http://schemas.android.com/apk/res/android"
ns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".ViewAllEmployees"
android:gravity="center_horizontal">
<ListView
android:id="@android:id/list"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_above="@+id/textView1"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"/>

</LinearLayout>
[/]
```

Here we are using a simple **ListView** inside a **LinearLayout**. Next create a class, name it **ViewAllEmployees.** and put the following code.

ViewAllEmployees

```
[]
package com.androidtutorialpoint.employeemanagementsystem;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;

import com.androidtutorialpoint.employeemanagementsystem.Model.Employee;
import com.androidtutorialpoint.employeemanagementsystem.DB.EmployeeOperations;

import .util.List;

public class ViewAllEmployees extends ListActivity{
  private EmployeeOperations employeeOps;
List<Employee> employees;

  @Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_view_all_employees);
employeeOps = new EmployeeOperations(this);
employeeOps.open();
employees = employeeOps.getAllEmployees();
employeeOps.close();
ArrayAdapter<Employee> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, employees);
setListAdapter(adapter);
}
}

[/]
```

The code for this activity is really simple, our activity extends ListActivity to show a list of all the employees. We create an instance of EmployeeOperations class, then query the DB for all employees and store those employees in the **employees** list.

At last we create an **ArrayAdapter** and then use it throught `setListAdapter`.

(adsbygoogle = window.adsbygoogle || []).push({});

Manifest & Resource Files

Since we are using two additional activities, do remember to add these in the manifest file. Here is the our **AndroidManifest.** file.

AndroidManifest.

```
[]
<? version="1.0" encoding="utf-8"?>
<manifest ns:android="http://schemas.android.com/apk/res/android"
package="com.androidtutorialpoint.employeemanagementsystem" >

  <application
android:allowBackup="true"
android:icon="@mipmap/ic_icon"
android:label="@string/app_name"
android:supportsRtl="true"
```

```
android:theme="@style/AppTheme" >
<activity android:name=".MainActivity" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />

  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity android:name=".AddUpdateEmployee"></activity>
<activity android:name=".ViewAllEmployees"></activity>

  </application>

</manifest>
[/]
```

You can also copy the App icon from the drawable folder from the Download Code link at the top of the post.

We have used the following **Strings** and **dimensions**. Copy the below resource files in the respective files.

strings.

```
[]

<resources>
<string name="app_name">EmployeeManagementSystem</string>
<string name="date_picker_title">Hire Date</string>
<string name="new_employee">New Employee</string>
<string name="show_subtitle">Show Subtitle</string>
<string name="hide_subtitle">Hide Subtitle</string>
<string name="subtitle_format">%1$s crimes</string>
<string name="choose_menu">Choose a Menu Option</string>
<string name="view_all_employees">View All Employees</string>
<string name="remove_employee">Remove Employee</string>
<string name="update_employee">Update Employee</string>
<string name="add_employee">Add Employee</string>
<string name="gender">Gender</string>
<string name="add_last_name">Last Name</string>
<string name="add_first_name">First Name</string>
<string name="string_male">Male</string>
<string name="string_female">Female</string>
<string name="add_date">Hire Date (DD/MM/YYYY)</string>
<string name="add_dept">Department</string>
<string name="enter_employee_id">Enter Your Employee Id</string>
```

```
<string name="settings">Settings</string>
</resources>
[/]
```

dimens.

```
[]
<resources>
<!-- Default screen margins, per the Android Design guidelines. -->
<dimen name="activity_horizontal_margin">16dp</dimen>
<dimen name="activity_vertical_margin">16dp</dimen>
<dimen name="header_margin">10dp</dimen>
<dimen name="text_size">20sp</dimen>
<dimen name="header_bottom_margin">30dp</dimen>
<dimen name="view_padding">10dp</dimen>
<dimen name="add_edit_textsize">18dp</dimen>
<dimen name="button_width">200dp</dimen>
<dimen name="button_padding">20dp</dimen>
<dimen name="button_height">50dp</dimen>
</resources>
[/]
```

Now save and run the app, try adding a few employees and then try to update an employee and then view them all using **View All Employees** Button. Please leave comments in case you like this tutorial and also post your queries :-).

Click on the Download Now button to download the full code.