PERTEMUAN 11
DATABASE SQLite

Menggunakan database adalah cara yang tepat untuk menyimpan data terstruktur atau data berulang, seperti informasi kontak. Halaman ini berasumsi bahwa Anda sudah familier dengan database SQL secara umum dan akan membantu Anda memulai database SQLite di Android. API yang nanti Anda perlukan untuk menggunakan database di Android tersedia dalam paket `android.database.sqlite`.

Menentukan Skema dan Kontrak

Salah satu prinsip utama database SQL adalah skemanya: deklarasi formal tentang cara database diatur. Skema ini tercermin dalam Pernyataan SQL yang Anda gunakan untuk membuat database. Ada baiknya Anda membuat kelas pendamping yang disebut dengan kelas *kontrak*, yang secara eksplisit menetapkan tata letak skema Anda dalam cara yang sistematis dan terdokumentasi sendiri.

Kelas kontrak adalah penampung untuk konstanta yang menentukan nama URI, tabel, dan kolom. Kelas kontrak memungkinkan Anda menggunakan konstanta yang sama pada semua kelas lain dalam paket yang sama. Hal ini memungkinkan Anda mengubah nama kolom di satu tempat, kemudian mengatur agar perubahan tersebut disebarkan ke seluruh kode.

Cara yang tepat untuk mengatur kelas kontrak adalah dengan memberikan definisi yang bersifat global pada seluruh database Anda di tingkat root kelas tersebut. Kemudian, buat kelas dalam untuk setiap tabel. Setiap kelas dalam akan menghitung kolom tabel yang terkait.

Contohnya, kontrak berikut menentukan nama tabel dan nama kolom untuk satu tabel yang merepresentasikan feed RSS:

```
object FeedReaderContract {
    // Table contents are grouped together in an anonymous object.
    object FeedEntry : BaseColumns {
        const val TABLE_NAME = "entry"
        const val COLUMN_NAME_TITLE = "title"
        const val COLUMN_NAME_SUBTITLE = "subtitle"
    }
}
```

Membuat database menggunakan SQL helper

Setelah menentukan tampilan database, Anda harus menerapkan metode yang akan membuat serta mengelola database dan tabel. Berikut adalah beberapa pernyataan umum untuk membuat dan menghapus tabel:

```
private const val SQL_CREATE_ENTRIES =
        "CREATE TABLE ${FeedEntry.TABLE_NAME} (" +
                "${BaseColumns._ID} INTEGER PRIMARY KEY," +
                "${FeedEntry.COLUMN_NAME_TITLE} TEXT," +
                "${FeedEntry.COLUMN_NAME_SUBTITLE} TEXT)"

private const val SQL_DELETE_ENTRIES = "DROP TABLE IF EXISTS
${FeedEntry.TABLE_NAME}"
```

Sama seperti file yang disimpan di penyimpanan internal perangkat, Android menyimpan database Anda dalam folder pribadi aplikasi. Data Anda akan selalu aman karena secara default area ini tidak dapat diakses oleh aplikasi lain atau oleh pengguna.

Kelas `SQLiteOpenHelper` berisi kumpulan API yang berguna untuk mengelola database Anda. Saat kelas ini digunakan untuk memperoleh referensi ke database, sistem hanya akan melakukan operasi pembuatan dan update database, yang mungkin memerlukan banyak waktu, hanya ketika diperlukan; *bukan pada saat aplikasi dimulai*. Yang perlu Anda lakukan hanyalah memanggil `getWritableDatabase()` atau `getReadableDatabase()`.

Untuk menggunakan `SQLiteOpenHelper`, buat subclass yang mengganti metode callback `onCreate()` dan `onUpgrade()`. Anda mungkin juga perlu menerapkan metode `onDowngrade()` atau `onOpen()`, tetapi keduanya tidak diperlukan.

Misalnya, berikut adalah penerapan `SQLiteOpenHelper` yang menggunakan beberapa perintah yang ditampilkan di atas:

```
class FeedReaderDbHelper(context: Context) :
SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {
    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL(SQL_CREATE_ENTRIES)
    }
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int,
newVersion: Int) {
        // This database is only a cache for online data, so its
upgrade policy is
        // to simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES)
        onCreate(db)
    }
    override fun onDowngrade(db: SQLiteDatabase, oldVersion: Int,
newVersion: Int) {
```

```
        onUpgrade(db, oldVersion, newVersion)
    }
    companion object {
        // If you change the database schema, you must increment the
database version.
        const val DATABASE_VERSION = 1
        const val DATABASE_NAME = "FeedReader.db"
    }
}
```

Untuk mengakses database, buat instance subclass `SQLiteOpenHelper` Anda:

```
val dbHelper = FeedReaderDbHelper(context)
```

Memasukkan informasi ke dalam database

Sisipkan data ke dalam database dengan meneruskan objek `ContentValues` ke metode `insert()`:

```
// Gets the data repository in write mode
val db = dbHelper.writableDatabase

// Create a new map of values, where column names are the keys
val values = ContentValues().apply {
    put(FeedEntry.COLUMN_NAME_TITLE, title)
    put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle)
}

// Insert the new row, returning the primary key value of the new row
val newRowId = db?.insert(FeedEntry.TABLE_NAME, null, values)
```

Argumen pertama untuk `insert()` cukup berisi nama tabel.

Argumen kedua memberi tahu framework tentang apa yang harus dilakukan jika `ContentValues` kosong (yaitu, Anda tidak `put` (memasukkan) nilai apa pun). Jika Anda menetapkan nama sebuah kolom, framework akan menyisipkan baris dan menetapkan nilai kolom tersebut ke null. Jika Anda menetapkan `null`, seperti dalam contoh kode ini, framework tidak akan menyisipkan baris ketika nilai tidak ada.

Metode `insert()` mengembalikan ID untuk baris yang baru dibuat, atau akan menampilkan -1 jika terjadi error saat memasukkan data. Hal ini dapat terjadi jika ada konflik dengan data yang sudah ada dalam database.

Membaca informasi dari database

Untuk membaca dari database, gunakan metode `query()`, dengan meneruskan kriteria pemilihan dan kolom yang diinginkan. Metode ini menggabungkan elemen `insert()` dan `update()`, tetapi daftar kolomnya menentukan data yang ingin diambil ("proyeksi"), bukan data yang akan dimasukkan. Hasil kueri ditampilkan kepada Anda dalam objek `Cursor`.

```
val db = dbHelper.readableDatabase

// Define a projection that specifies which columns from the database
// you will actually use after this query.
val projection = arrayOf(BaseColumns._ID,
FeedEntry.COLUMN_NAME_TITLE, FeedEntry.COLUMN_NAME_SUBTITLE)

// Filter results WHERE "title" = 'My Title'
val selection = "${FeedEntry.COLUMN_NAME_TITLE} = ?"
val selectionArgs = arrayOf("My Title")

// How you want the results sorted in the resulting Cursor
val sortOrder = "${FeedEntry.COLUMN_NAME_SUBTITLE} DESC"

val cursor = db.query(
        FeedEntry.TABLE_NAME,   // The table to query
        projection,             // The array of columns to return
(pass null to get all)
        selection,              // The columns for the WHERE clause
        selectionArgs,          // The values for the WHERE clause
        null,                   // don't group the rows
        null,                   // don't filter by row groups
        sortOrder               // The sort order
)
```

Argumen ketiga dan keempat (`selection` and `selectionArgs`) digabungkan untuk membuat klausa WHERE. Argumen ini diberikan secara terpisah dari kueri pemilihan sehingga akan dikecualikan sebelum digabungkan. Oleh karena itu, pernyataan pemilihan Anda tidak akan terpengaruh oleh penambahan SQL. Untuk mengetahui detail selengkapnya tentang semua argumen, lihat referensi `query()`.

Untuk melihat baris dalam kursor, gunakan salah satu metode pemindahan `Cursor`, yang harus selalu Anda panggil sebelum mulai membaca nilai. Kursor dimulai pada posisi -1 sehingga memanggil `moveToNext()` akan menempatkan "posisi baca" pada entri pertama dalam hasil dan mengembalikan apakah kursor sudah melewati entri terakhir dalam kumpulan hasil atau belum. Untuk setiap baris, Anda dapat membaca nilai kolom dengan memanggil salah satu metode get `Cursor`, seperti `getString()` atau `getLong()`. Untuk setiap metode get, Anda harus meneruskan posisi indeks kolom yang Anda inginkan, yang bisa Anda

dapatkan dengan memanggil getColumnIndex() atau getColumnIndexOrThrow(). Setelah selesai mengiterasi hasilnya, panggil close() pada kursor untuk melepaskan resource-nya. Misalnya, berikut adalah cara mendapatkan semua ID item yang disimpan dalam kursor dan menambahkannya ke daftar:

```
val itemIds = mutableListOf<Long>()
with(cursor) {
    while (moveToNext()) {
        val itemId = getLong(getColumnIndexOrThrow(BaseColumns._ID))
        itemIds.add(itemId)
    }
}
```

Menghapus informasi dari database

Untuk menghapus baris dari tabel, Anda harus memberikan kriteria pemilihan yang mengidentifikasi baris ke metode delete(). Mekanismenya bekerja dalam cara yang sama seperti argumen pemilihan untuk metode query(). Proses ini membagi spesifikasi pemilihan menjadi klausa pemilihan dan argumen pemilihan. Klausa menentukan kolom yang harus dilihat, juga memungkinkan Anda untuk menggabungkan proses pengujian kolom. Argumen adalah nilai yang akan digunakan untuk pengujian, yang terikat dengan klausa tersebut. Karena hasilnya tidak ditangani seperti Pernyataan SQL biasa, penambahan SQL pun tidak akan berpengaruh.

```
// Define 'where' part of query.
val selection = "${FeedEntry.COLUMN_NAME_TITLE} LIKE ?"
// Specify arguments in placeholder order.
val selectionArgs = arrayOf("MyTitle")
// Issue SQL statement.
val deletedRows = db.delete(FeedEntry.TABLE_NAME, selection,
selectionArgs)
```

Nilai hasil untuk metode delete() menunjukkan jumlah baris yang telah dihapus dari database.

Mengupdate database

Bila Anda perlu memodifikasi subset nilai database, gunakan metode update().

Memperbarui tabel akan menggabungkan sintaks ContentValues dari insert() dengan sintaks WHERE dari delete()

```
val db = dbHelper.writableDatabase

// New value for one column
val title = "MyNewTitle"
```

```
val values = ContentValues().apply {
    put(FeedEntry.COLUMN_NAME_TITLE, title)
}

// Which row to update, based on the title
val selection = "${FeedEntry.COLUMN_NAME_TITLE} LIKE ?"
val selectionArgs = arrayOf("MyOldTitle")
val count = db.update(
        FeedEntry.TABLE_NAME,
        values,
        selection,
        selectionArgs)
```

Nilai hasil dari metode `update()` adalah jumlah baris yang terpengaruh dalam database.

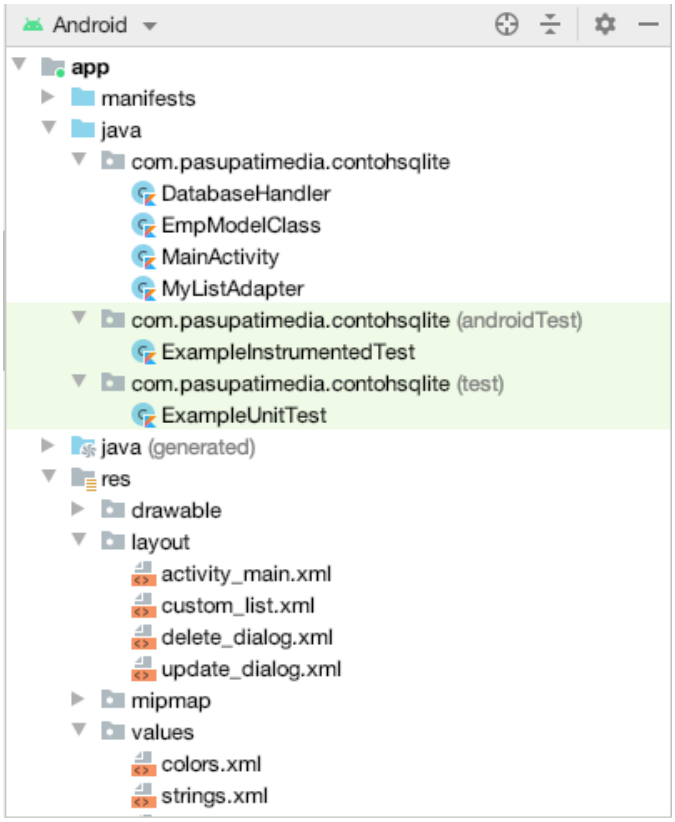
Mempertahankan koneksi database

Karena `getWritableDatabase()` dan `getReadableDatabase()` sulit dipanggil jika database ditutup, koneksi database harus tetap terbuka selama Anda perlu mengaksesnya. Biasanya, akan lebih optimal untuk menutup database dalam `onDestroy()` Aktivitas pemanggilan

```
override fun onDestroy() {
    dbHelper.close()
    super.onDestroy()
}
```

Buatlah Project Baru dengan nama ContohSQLite

Struktur File Project



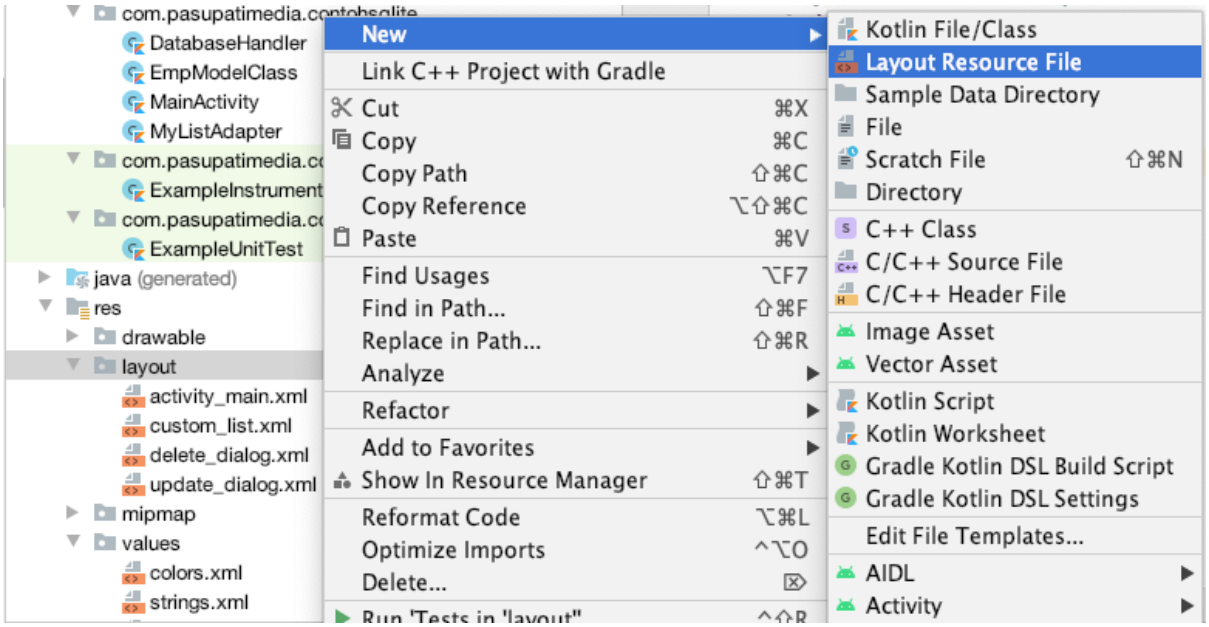
Ketik koding berikut pada lembar kerja activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <TableRow>

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="1"
                android:text="User Id" />

            <EditText

                android:id="@+id/u_id"
```

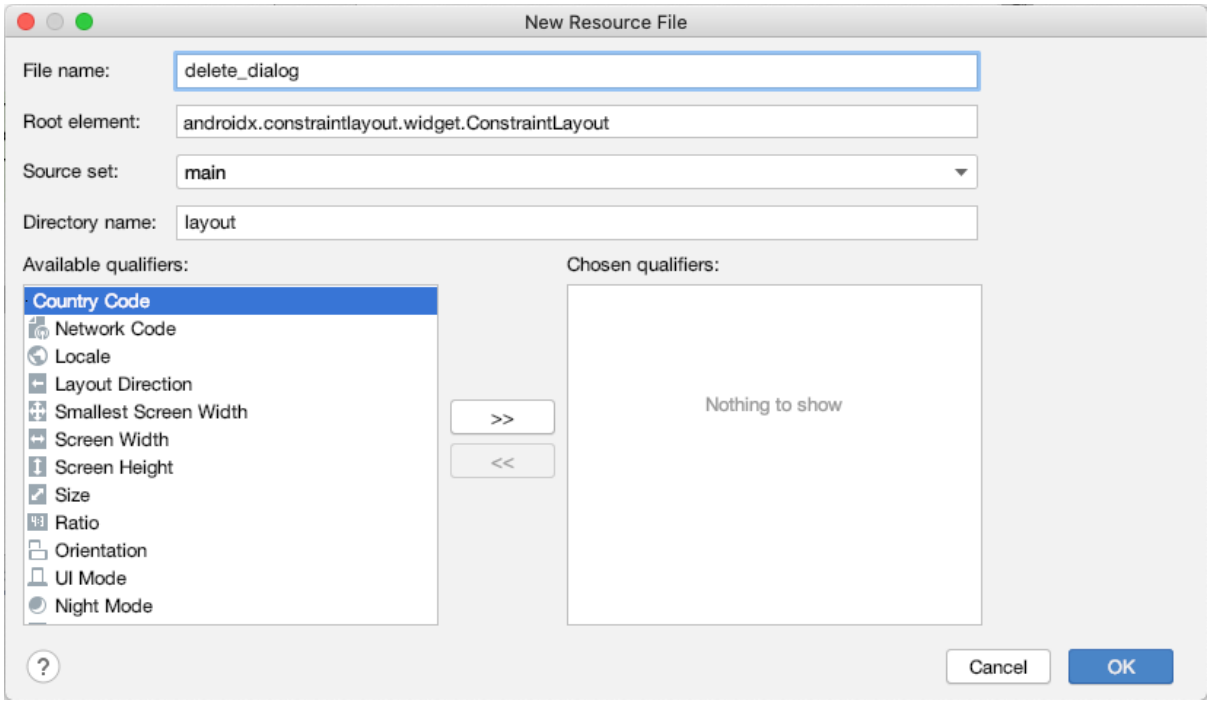
```
                android:layout_width="200dp"
                android:layout_height="wrap_content"
                android:layout_marginStart="20sp"
                android:layout_marginLeft="20sp"
                android:width="150px" />
        </TableRow>

        <TableRow>

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="1"
                android:text="User Name" />

            <EditText
                android:id="@+id/u_name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="2"
                android:layout_marginStart="20sp"
                android:layout_marginLeft="20sp"
                android:width="200dp" />
        </TableRow>

        <TableRow>

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="1"
                android:text="User Email" />

            <EditText
                android:id="@+id/u_email"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="2"
                android:layout_marginStart="20sp"
                android:layout_marginLeft="20sp"
                android:width="200dp" />
        </TableRow>

    </TableLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="350sp"
        android:layout_marginTop="20sp">

        <ListView
            android:id="@+id/listView"
            android:layout_width="wrap_content"
            android:layout_height="350sp" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="40sp"
        android:orientation="horizontal">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:onClick="saveRecord"
            android:text="Save" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="viewRecord"
            android:text="View" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="updateRecord"
            android:text="Update" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="deleteRecord"
            android:text="Delete" />
    </LinearLayout>
</LinearLayout>
```

Buatlah file delete_dialog.xml dengan cara sebagai berikut:



Kemudian berinama sebagai delete_dialog dan klik OK



Ketik koding pada delete_dialog.xml seperti berikut:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="10dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/deleteId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="enter id" />
</LinearLayout>
```

Kemudian buat file custom_list.xml dengan cara yang sama seperti di atas, kemudian ketik koding berikut:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/linearLayout">

    <TextView
        android:id="@+id/textViewId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Id"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"/>

    <TextView
        android:id="@+id/textViewName"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Name"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"/>

    <TextView
        android:id="@+id/textViewEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Email"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"/>
</LinearLayout>
```

Selanjutnya buat file update_dialog.xml dengan cara yang sama seperti di atas, kemudian ketik koding sebagai berikut:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp">

    <EditText
        android:id="@+id/updateId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="enter id" />

    <EditText
        android:id="@+id/updateName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="enter name" />

    <EditText
        android:id="@+id/updateEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="enter email" />
</LinearLayout>
```

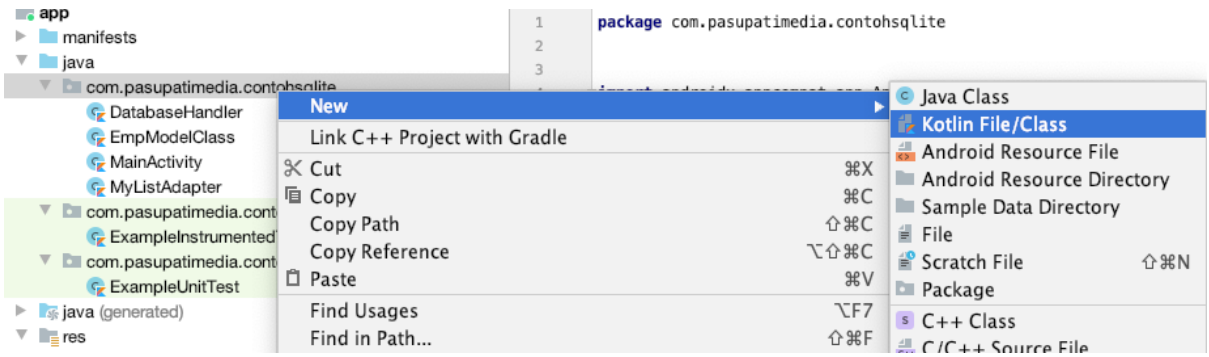
Selanjutnya pada file MainActivity.kt silakan ketik koding sebagai berikut:

```
package com.pasupatimedia.contohsqlite


import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.EditText
import android.widget.Toast
import kotlinx.android.synthetic.main.activity_main.*
import android.content.DialogInterface
import androidx.appcompat.app.AlertDialog

//import android.support.v7.app.AlertDialog
//import android.support.v7.appcompat.R;


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

    //method for saving records in database
    fun saveRecord(view: View) {
        val id = u_id.text.toString()
        val name = u_name.text.toString()
        val email = u_email.text.toString()
        val databaseHandler: DatabaseHandler = DatabaseHandler(this)
        if (id.trim() != "" && name.trim() != "" && email.trim() != "") {
            val status =
                databaseHandler.addEmployee(EmpModelClass(Integer.parseInt(id),
name, email))
            if (status > -1) {
                Toast.makeText(applicationContext, "record save",
Toast.LENGTH_LONG).show()
                u_id.text.clear()
                u_name.text.clear()
                u_email.text.clear()
            }
        } else {
            Toast.makeText(
                applicationContext,
                "id or name or email cannot be blank",
                Toast.LENGTH_LONG
            ).show()
        }

    }

    //method for read records from database in ListView
    fun viewRecord(view: View) {
        //creating the instance of DatabaseHandler class
        val databaseHandler: DatabaseHandler = DatabaseHandler(this)
        //calling the viewEmployee method of DatabaseHandler class to read the
records
        val emp: List<EmpModelClass> = databaseHandler.viewEmployee()
        val empArrayId = Array<String>(emp.size) { "0" }
        val empArrayName = Array<String>(emp.size) { "null" }
        val empArrayEmail = Array<String>(emp.size) { "null" }
        var index = 0
        for (e in emp) {
            empArrayId[index] = e.userId.toString()
            empArrayName[index] = e.userName
```

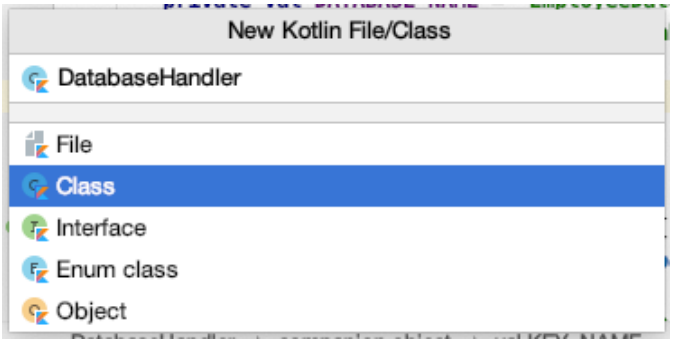
```
            empArrayEmail[index] = e.userEmail
            index++
        }
        //creating custom ArrayAdapter
        val myListAdapter = MyListAdapter(this, empArrayId, empArrayName,
empArrayEmail)
        listView.adapter = myListAdapter
    }

    //method for updating records based on user id
    fun updateRecord(view: View) {
        val dialogBuilder = AlertDialog.Builder(this)
        val inflater = this.layoutInflater
        val dialogView = inflater.inflate(R.layout.update_dialog, null)
        dialogBuilder.setView(dialogView)

        val edtId = dialogView.findViewById(R.id.updateId) as EditText
        val edtName = dialogView.findViewById(R.id.updateName) as EditText
        val edtEmail = dialogView.findViewById(R.id.updateEmail) as EditText

        dialogBuilder.setTitle("Update Record")
        dialogBuilder.setMessage("Enter data below")
        dialogBuilder.setPositiveButton("Update", DialogInterface.OnClickListener
{ _, _ ->

            val updateId = edtId.text.toString()
            val updateName = edtName.text.toString()
            val updateEmail = edtEmail.text.toString()
            //creating the instance of DatabaseHandler class
            val databaseHandler: DatabaseHandler = DatabaseHandler(this)
            if (updateId.trim() != "" && updateName.trim() != "" &&
updateEmail.trim() != "") {
                //calling the updateEmployee method of DatabaseHandler class to
update record
                val status = databaseHandler.updateEmployee(
                    EmpModelClass(
                        Integer.parseInt(updateId),
                        updateName,
                        updateEmail
                    )
                )
                if (status > -1) {
                    Toast.makeText(applicationContext, "record update",
Toast.LENGTH_LONG).show()
                }
            } else {
                Toast.makeText(
                    applicationContext,
                    "id or name or email cannot be blank",
                    Toast.LENGTH_LONG
                ).show()
            }

        })
        dialogBuilder.setNegativeButton("Cancel", DialogInterface.OnClickListener
{ dialog, which ->
            //pass
        })
        val b = dialogBuilder.create()
        b.show()
    }

    //method for deleting records based on id
    fun deleteRecord(view: View) {
        //creating AlertDialog for taking user id
        val dialogBuilder = AlertDialog.Builder(this)
```

```
        val inflater = this.layoutInflater
        val dialogView = inflater.inflate(R.layout.delete_dialog, null)
        dialogBuilder.setView(dialogView)

        val dltId = dialogView.findViewById(R.id.deleteId) as EditText
        dialogBuilder.setTitle("Delete Record")
        dialogBuilder.setMessage("Enter id below")
        dialogBuilder.setPositiveButton("Delete", DialogInterface.OnClickListener
{ _, _ ->

            val deleteId = dltId.text.toString()
            //creating the instance of DatabaseHandler class
            val databaseHandler: DatabaseHandler = DatabaseHandler(this)
            if (deleteId.trim() != "") {
                //calling the deleteEmployee method of DatabaseHandler class to
delete record
                val status = databaseHandler.deleteEmployee(
                    EmpModelClass(
                        Integer.parseInt(deleteId),
                        "",
                        ""
                    )
                )
                if (status > -1) {
                    Toast.makeText(applicationContext, "record deleted",
Toast.LENGTH_LONG).show()
                }
            } else {
                Toast.makeText(
                    applicationContext,
                    "id or name or email cannot be blank",
                    Toast.LENGTH_LONG
                ).show()
            }

        })
        dialogBuilder.setNegativeButton("Cancel", DialogInterface.OnClickListener
{ _, _ ->
            //pass
        })
        val b = dialogBuilder.create()
        b.show()
    }
}
```

Kemudian buat class baru dengan nama DatabaseHandler.kt dengan cara sebagai berikut:



Kemudian buat file DatabaseHandler.kt seperti berikut dan klik Enter atau bisa menggunakan cursor.



Pada file DatabaseHandler.kt silakan ketik koding berikut:

```
package com.pasupatimedia.contohsqlite

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import android.content.ContentValues
import android.database.Cursor
import android.database.sqlite.SQLiteException

//creating the database logic, extending the SQLiteOpenHelper base class
class DatabaseHandler(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {
    companion object {
        private val DATABASE_VERSION = 1
        private val DATABASE_NAME = "EmployeeDatabase"
        private val TABLE_CONTACTS = "EmployeeTable"
        private val KEY_ID = "id"
        private val KEY_NAME = "name"
        private val KEY_EMAIL = "email"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        // TODO("not implemented") //To change body of created functions use File
| Settings | File Templates.
        //creating table with fields
        val CREATE_CONTACTS_TABLE = ("CREATE TABLE " + TABLE_CONTACTS + "("
                + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
                + KEY_EMAIL + " TEXT" + ")")
        db?.execSQL(CREATE_CONTACTS_TABLE)
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int)
{
        //  TODO("not implemented") //To change body of created functions use
File | Settings | File Templates.
        db!!.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS)
        onCreate(db)
    }


    //method to insert data
    fun addEmployee(emp: EmpModelClass): Long {
        val db = this.writableDatabase
        val contentValues = ContentValues()
        contentValues.put(KEY_ID, emp.userId)
        contentValues.put(KEY_NAME, emp.userName) // EmpModelClass Name
```

```
        contentValues.put(KEY_EMAIL, emp.userEmail) // EmpModelClass Phone
        // Inserting Row
        val success = db.insert(TABLE_CONTACTS, null, contentValues)
        //2nd argument is String containing nullColumnHack
        db.close() // Closing database connection
        return success
    }

    //method to read data
    fun viewEmployee(): List<EmpModelClass> {
        val empList: ArrayList<EmpModelClass> = ArrayList<EmpModelClass>()
        val selectQuery = "SELECT  * FROM $TABLE_CONTACTS"
        val db = this.readableDatabase
        var cursor: Cursor? = null
        try {
            cursor = db.rawQuery(selectQuery, null)
        } catch (e: SQLiteException) {
            db.execSQL(selectQuery)
            return ArrayList()
        }
        var userId: Int
        var userName: String
        var userEmail: String
        if (cursor.moveToFirst()) {
            do {
                userId = cursor.getInt(cursor.getColumnIndex("id"))
                userName = cursor.getString(cursor.getColumnIndex("name"))
                userEmail = cursor.getString(cursor.getColumnIndex("email"))
                val emp = EmpModelClass(userId = userId, userName = userName,
userEmail = userEmail)
                empList.add(emp)
            } while (cursor.moveToNext())
        }
        return empList
    }

    //method to update data
    fun updateEmployee(emp: EmpModelClass): Int {
        val db = this.writableDatabase
        val contentValues = ContentValues()
        contentValues.put(KEY_ID, emp.userId)
        contentValues.put(KEY_NAME, emp.userName) // EmpModelClass Name
        contentValues.put(KEY_EMAIL, emp.userEmail) // EmpModelClass Email

        // Updating Row
        val success = db.update(TABLE_CONTACTS, contentValues, "id=" +
emp.userId, null)
        //2nd argument is String containing nullColumnHack
        db.close() // Closing database connection
        return success
    }

    //method to delete data
    fun deleteEmployee(emp: EmpModelClass): Int {
        val db = this.writableDatabase
        val contentValues = ContentValues()
        contentValues.put(KEY_ID, emp.userId) // EmpModelClass UserId
        // Deleting Row
        val success = db.delete(TABLE_CONTACTS, "id=" + emp.userId, null)
        //2nd argument is String containing nullColumnHack
        db.close() // Closing database connection
        return success
    }
}
```

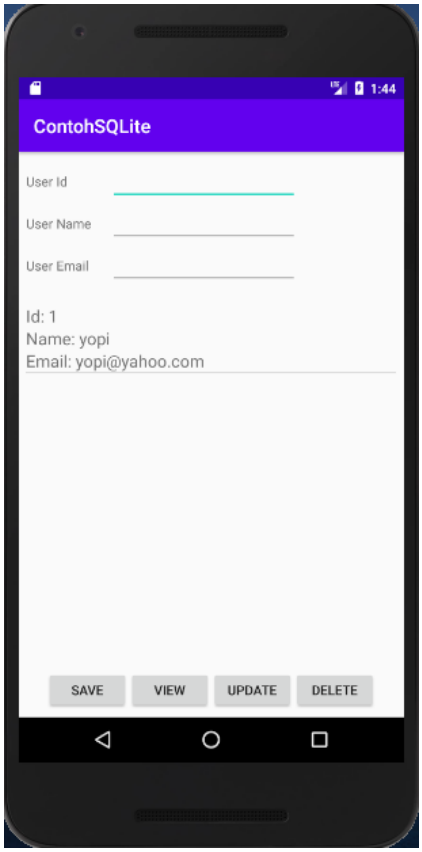
Kemudian buat file EmpModelClass.kt dengan cara yang sama seperti diatas, dan ketik kode program berikut:

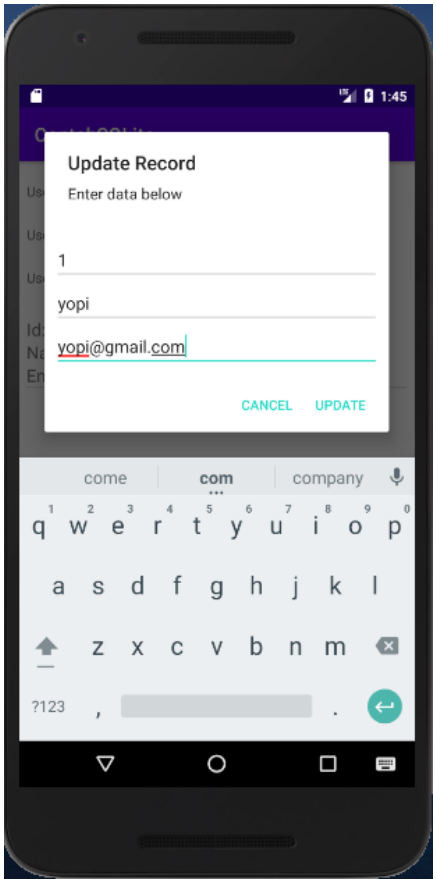
```
package com.pasupatimedia.contohsqlite

class EmpModelClass (var userId: Int, val userName:String , val userEmail:
String)
```

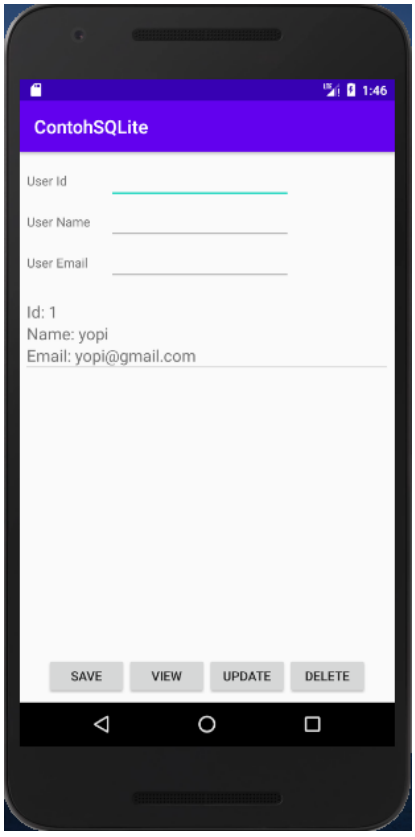
Selanjutnya buat file MyListAdapter.kt dengan cara yang sama seperti di atas, dan ketik kode program sebagai berikut:

```
package com.pasupatimedia.contohsqlite

import android.app.Activity
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.TextView

class MyListAdapter(private val context: Activity, private val id: Array<String>,
private val name: Array<String>, private val email: Array<String>)
    : ArrayAdapter<String>(context, R.layout.custom_list, name) {

    override fun getView(position: Int, view: View?, parent: ViewGroup): View {
        val inflater = context.layoutInflater
        val rowView = inflater.inflate(R.layout.custom_list, null, true)

        val idText = rowView.findViewById(R.id.textViewId) as TextView
        val nameText = rowView.findViewById(R.id.textViewName) as TextView
        val emailText = rowView.findViewById(R.id.textViewEmail) as TextView

        idText.text = "Id: ${id[position]}"
        nameText.text = "Name: ${name[position]}"
        emailText.text = "Email: ${email[position]}"
        return rowView
    }
}
```

Silakan jalankan programnya, jika sukses maka akan tampil seperti gambar dibawah ini:


1:44
ContohSQLite
User Id
User Name
User Email
Id: 1
Name: yopi
Email: yopi@yahoo.com
SAVE
VIEW
UPDATE
DELETE


1:45
Update Record
Enter data below
1
yopi
yopi@gmail.com
CANCEL
UPDATE
come
com
company


1:46
ContohSQLite
User Id
User Name
User Email
Id: 1
Name: yopi
Email: yopi@gmail.com
SAVE
VIEW
UPDATE
DELETE