MODUL PEMROGRAMAN MOBILE

CIM 430

MODUL PERTEMUAN 11

FLUTTER - DATABASE CONNECTION

DISUSUN OLEH :

7174 – SAWALI WAHYU, S.KOM, M.KOM

8126 – JEFRY SUNUPURWA ASRI, S.KOM, M.KOM

UNIVERSITAS ESA UNGGUL

FAKULTAS ILMU KOMPUTER

TAHUN 2021

DATABASE CONNECTION

A. Kemampuan Akhir Yang Diharapkan

Setelah mempelajari modul ini, diharapkan mahasiswa mampu :

1. Mahasiswa mampu membuat koneksi database dengan tools yang tersedia seperti SQLite, MySQL, PosgreSQL, Atau NoSQL (Firebase, MongoDB dll).
2. Mahasiswa mampu membuat arsitektur model aplikasi untuk memodelkan simulasi pada aplikasi mobile yang dibuat.
3. Mahasiswa mampu membuat aplikasi dengan kompleksitas dan relasi basis data yang dapat memiliki output yang relevan.

Flutter - Konsep Database

Flutter menyediakan banyak paket lanjutan untuk bekerja dengan database. Paket terpenting yaitu :

1) **sqflite** - Digunakan untuk mengakses dan memanipulasi database SQLite, dan
2) **firebase_database** - Digunakan untuk mengakses dan memanipulasi database NoSQL yang dihosting di cloud dari Google.

Dalam bab ini, mari kita bahas masing-masing secara mendetail.

SQLite

Database SQLite adalah de-facto dan mesin database tertanam berbasis SQL standar. Ini adalah mesin database kecil dan teruji waktu. paket sqflite menyediakan banyak fungsi untuk bekerja secara efisien dengan database SQLite. Ini menyediakan metode standar untuk memanipulasi mesin database SQLite. Fungsionalitas inti yang disediakan oleh paket sqflite adalah sebagai berikut -

a) Buat / Buka (metode openDatabase) database SQLite.
b) Jalankan pernyataan SQL (metode eksekusi) terhadap database SQLite.
c) Metode kueri tingkat lanjut (metode kueri) untuk mengurangi menjadi kode yang diperlukan untuk kueri dan mendapatkan informasi dari database SQLite.

Mari kita buat aplikasi produk untuk menyimpan dan mengambil informasi produk dari mesin database SQLite standar menggunakan paket sqflite dan memahami konsep di balik database SQLite dan paket sqflite.

a) Buat aplikasi Flutter baru di studio Android, product_sqlite_app.
b) Ganti kode startup default (main.dart) dengan kode product_rest_app kami .

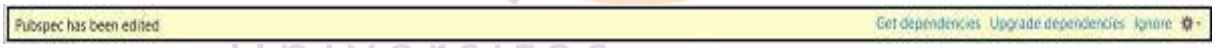
c) Salin folder aset dari product_nav_app ke product_rest_app dan tambahkan aset di dalam file * pubspec.yaml`.

```
flutter:
   assets:
      - assets/appimages/floppy.png
      - assets/appimages/iphone.png
      - assets/appimages/laptop.png
      - assets/appimages/pendrive.png
      - assets/appimages/pixel.png
      - assets/appimages/tablet.png
```

Konfigurasikan paket sqflite di file pubspec.yaml seperti yang ditunjukkan di bawah ini - dependencies: sqflite: any

Gunakan nomor versi terbaru dari sqflite sebagai pengganti

1) Konfigurasikan paket path_provider di file pubspec.yaml seperti yang ditunjukkan di bawah ini - dependencies: path_provider: any
2) Di sini, paket path_provider digunakan untuk mendapatkan jalur folder sementara dari sistem dan jalur aplikasi. Menggunakan nomor versi terbaru dari *sqflite* di tempat *manapun*.
3) Android studio akan memberi tahu bahwa pubspec.yaml telah diperbarui.



4) Klik Dapatkan opsi ketergantungan. Android studio akan mendapatkan paket dari Internet dan mengkonfigurasinya dengan benar untuk aplikasi.

5) Dalam database kita membutuhkan primary key, id sebagai field tambahan beserta properti Product seperti name, price, dll. Jadi, tambahkan properti id pada class Product. Juga, tambahkan metode baru, toMap untuk mengubah objek produk menjadi objek Map. fromMap dan toMap digunakan untuk membuat serial dan deserialisasi objek Produk dan digunakan dalam metode manipulasi database.

```
class Product
   { final int
   id;
   final String name;
   final String description;
   final int price;
   final String image;
   static final columns = ["id", "name", "description", "price",
"image"];
   Product(this.id, this.name, this.description, this.price,
this.image);
   factory Product.fromMap(Map<String, dynamic> data)
      { return Product(
         data['id'],
         data['name'],
         data['description'],
         data['price'],
         data['image'],
      );
   }
   Map<String, dynamic> toMap() =>
      { "id": id,
      "name": name,
      "description": description,
      "price": price,
      "image": image
   };
```

Tata Cara :

1) Buat file baru, Database.dart di folder lib untuk menulis fungsionalitas terkait SQLite.
2) Impor pernyataan impor yang diperlukan di Database.dart.


```
import 'dart:async';
import 'dart:io';
import 'package:path/path.dart';
import 'package:path_provider/path_provider.dart';
import 'package:sqflite/sqflite.dart';
import 'Product.dart';
```

1) Perhatikan poin-poin berikut di sini -

a) **async** digunakan untuk menulis metode asynchronous.

b) **io** digunakan untuk mengakses file dan direktori.

c) **path** digunakan untuk mengakses fungsi utilitas inti panah yang terkait dengan jalur file.

d) **path_provider** digunakan untuk mendapatkan jalur sementara dan aplikasi.

e) **sqflite** digunakan untuk memanipulasi database SQLite.

2) Buat kelas baru **SQLiteDbProvider**

3) Deklarasikan objek SQLiteDbProvider statis berbasis tunggal seperti yang ditentukan di bawah ini –

```
class SQLiteDbProvider
   { SQLiteDbProvider._();
   static final SQLiteDbProvider db
   = SQLiteDbProvider._();
   static Database _database;
}
```

3) Objek SQLiteDBProvoider dan metodenya dapat diakses melalui variabel db statis.

SQLiteDBProvoider.db.<emthod>

4) Buat metode untuk mendapatkan database (opsi Future) berjenis Future <Database>. Buat tabel produk dan muat data awal selama pembuatan database itu sendiri.

```
Future<Database> get database
   async { if (_database != null)
   return _database;
   _database = await
   initDB(); return
   _database;
}
initDB() async {
   Directory documentsDirectory =
await
getApplicationDocumentsDirectory();
   String path = join(documentsDirectory.path,
   "ProductDB.db"); return await openDatabase(
      path,
      version:
      1,
      onOpen: (db) {},
      onCreate: (Database db, int version) async {
```

```
await db.execute(
              "CREATE TABLE Product
              (" "id INTEGER PRIMARY
              KEY,"
              "name TEXT,"
              "description
              TEXT," "price
              INTEGER," "image
              TEXT" ")"
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name',
  'description', 'price', 'image')
               values (?, ?, ?, ?, ?)",
               [1, "iPhone", "iPhone is the stylist phone
  ever", 1000, "iphone.png"]
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name',
  'description', 'price', 'image')
               values (?, ?, ?, ?, ?)",
               [2, "Pixel", "Pixel is the most feature phone
  ever", 800, "pixel.png"]
```

```
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name',
'description', 'price', 'image')
               values (?, ?, ?, ?, ?)",
               [3, "Laptop", "Laptop is most productive
development tool", 2000, "laptop.png"]\
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name',
'description', 'price', 'image')
               values (?, ?, ?, ?, ?)",
               [4, "Tablet", "Laptop is most productive
development tool", 1500, "tablet.png"]
            );
            await
               db.execute( "INSE
               RT INTO Product
               ('id', 'name', 'description', 'price', 'image')
               values (?, ?, ?, ?, ?)",
               [5, "Pendrive", "Pendrive is useful storage
medium", 100, "pendrive.png"]
            );
            await
               db.execute( "INSE
               RT INTO Product
               ('id', 'name', 'description', 'price', 'image')
               values (?, ?, ?, ?, ?)",
               [6, "Floppy Drive", "Floppy drive is useful
rescue storage medium", 20, "floppy.png"]
            );
         }
      );
}
```

Di sini, kami telah menggunakan metode berikut -

1) getApplicationDocumentsDirectory - Mengembalikan jalur direktori aplikasi
2) join - Digunakan untuk membuat jalur khusus sistem. Kami telah menggunakannya untuk membuat jalur database.
3) openDatabase - Digunakan untuk membuka database SQLite

4) onOpen - Digunakan untuk menulis kode saat membuka database
5) onCreate - Digunakan untuk menulis kode saat database dibuat untuk pertama kalinya
6) db.execute - Digunakan untuk mengeksekusi query SQL. Ini menerima kueri. Jika kueri memiliki tempat penampung (?), Maka itu menerima nilai sebagai daftar di argumen kedua.

Tulis metode untuk mendapatkan semua produk di database

```
Future<List<Product>> getAllProducts() async
   { final db = await database;
   List<Map>
   results = await db.query("Product", columns: Product.columns,
orderBy: "id ASC");

   List<Product> products = new List();
   results.forEach((result) {
      Product product = Product.fromMap(result);
      products.add(product);
   });
   return products;
}
```

Keterangan :

1) Metode kueri digunakan untuk mengambil semua informasi produk. query menyediakan jalan pintas untuk membuat kueri informasi tabel tanpa menulis seluruh kueri. metode kueri akan menghasilkan kueri yang tepat itu sendiri dengan menggunakan masukan kami seperti kolom, orderBy, dll,

2) Menggunakan metode fromMap Produk untuk mendapatkan detail produk dengan mengulang objek hasil, yang menampung semua baris dalam tabel.

Tulis metode untuk mendapatkan produk khusus untuk **id**

```
Future<Product> getProductById(int id) async
    { final db = await database;
    var result = await db.query("Product", where: "id = ",
whereArgs: [id]);
    return result.isNotEmpty ? Product.fromMap(result.first) :
Null;
}
```

1) telah menggunakan where dan whereArgs untuk menerapkan filter.
2) Buat tiga metode - menyisipkan, memperbarui, dan menghapus metode untuk memasukkan, memperbarui, dan menghapus produk dari database

```
insert(Product product) async
    { final db = await database;
    var maxIdResult = await db.rawQuery(
        "SELECT MAX(id)+1 as last_inserted_id FROM Product");


    var id = maxIdResult.first["last_inserted_id"]; var
    result = await db.rawInsert(
        "INSERT Into Product (id, name, description, price, image)" " VALUES
        (?, ?, ?, ?, ?)",
        [id, product.name, product.description, product.price,
product.image]
    );
    return result;
}
update(Product product) async
    { final db = await database;
    var result = await db.update("Product", product.toMap(), where:
    "id = ?", whereArgs: [product.id]); return result;
}
delete(int id) async {
    final db = await database;
    db.delete("Product", where: "id = ?", whereArgs: [id]);
}
```

Kode terakhir dari Database.dart adalah sebagai berikut :

```
import 'dart:async';
import 'dart:io';
import 'package:path/path.dart';
import 'package:path_provider/path_provider.dart'; import
'package:sqflite/sqflite.dart';
import 'Product.dart';


class SQLiteDbProvider
    { SQLiteDbProvider._();
    static final SQLiteDbProvider db = SQLiteDbProvider._(); static
    Database _database;


    Future<Database> get database async { if
        (_database != null)
        return _database;
        _database = await initDB(); return
        _database;
    }
    initDB() async {
        Directory documentsDirectory = await
        getApplicationDocumentsDirectory();
```

```
      String path = join(documentsDirectory.path,
"ProductDB.db");
      return await openDatabase(
         path, version: 1,
         onOpen: (db) {},
         onCreate: (Database db, int version) async {
            await db.execute(
               "CREATE TABLE Product ("
               "id INTEGER PRIMARY KEY,"
               "name TEXT,"
               "description TEXT,"
               "price INTEGER,"
               "image TEXT"")"
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name', 'description',
'price', 'image')
               values (?, ?, ?, ?, ?)",
               [1, "iPhone", "iPhone is the stylist phone ever",
1000, "iphone.png"]
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name', 'description',
'price', 'image')
               values (?, ?, ?, ?, ?)",
               [2, "Pixel", "Pixel is the most feature phone
ever", 800, "pixel.png"]
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name', 'description',
'price', 'image')
               values (?, ?, ?, ?, ?)",
               [3, "Laptop", "Laptop is most productive
development tool", 2000, "laptop.png"]
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name', 'description',
'price', 'image')
               values (?, ?, ?, ?, ?)",
               [4, "Tablet", "Laptop is most productive
development tool", 1500, "tablet.png"]
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name', 'description',
'price', 'image')
               values (?, ?, ?, ?, ?)",
               [5, "Pendrive", "Pendrive is useful storage
medium", 100, "pendrive.png"]
            );
            await db.execute(
               "INSERT INTO Product ('id', 'name', 'description',
'price', 'image')
               values (?, ?, ?, ?, ?)",
```

```
                [6, "Floppy Drive", "Floppy drive is useful rescue
storage medium", 20, "floppy.png"]
              );
          }
      );
   }
   Future<List<Product>> getAllProducts() async {
      final db = await database;
      List<Map> results = await db.query(
         "Product", columns: Product.columns, orderBy: "id ASC"
      );
      List<Product> products = new List();
      results.forEach((result) {
         Product product = Product.fromMap(result);
         products.add(product);
      });
      return products;
   }
   Future<Product> getProductById(int id) async {
      final db = await database;
      var result = await db.query("Product", where: "id = ",
whereArgs: [id]);
      return result.isNotEmpty ? Product.fromMap(result.first) :
Null;
   }
   insert(Product product) async {
      final db = await database;
      var maxIdResult = await db.rawQuery("SELECT MAX(id)+1 as
last_inserted_id FROM Product");
      var id = maxIdResult.first["last_inserted_id"];
      var result = await db.rawInsert(
         "INSERT Into Product (id, name, description, price,
image)"
         " VALUES (?, ?, ?, ?, ?)",
         [id, product.name, product.description, product.price,
product.image]
      );
      return result;
   }
   update(Product product) async {
      final db = await database;
      var result = await db.update(
         "Product", product.toMap(), where: "id = ?", whereArgs:
[product.id]
      );
      return result;
   }
   delete(int id) async {
      final db = await database;
      db.delete("Product", where: "id = ?", whereArgs: [id]);
   }
}
```

1) Ubah metode utama untuk mendapatkan informasi produk.

```
void main() {
   runApp(MyApp(products: SQLiteDbProvider.db.getAllProducts()));
}
```

2) Di sini, kami telah menggunakan metode getAllProducts untuk mengambil semua produk dari database.
3) Jalankan aplikasinya dan lihat hasilnya. Ini akan serupa dengan contoh sebelumnya, *Mengakses API layanan Produk* , kecuali informasi produk disimpan dan diambil dari database SQLite lokal.

Cloud Firestore

Firebase adalah platform pengembangan aplikasi BaaS. Ini menyediakan banyak fitur untuk mempercepat pengembangan aplikasi seluler seperti layanan otentikasi, penyimpanan cloud, dll., Salah satu fitur utama Firebase adalah Cloud Firestore, database NoSQL waktu nyata berbasis cloud.

Flutter menyediakan paket khusus, cloud_firestore untuk diprogram dengan Cloud Firestore. Mari kita buat toko produk online di Cloud Firestore dan buat aplikasi untuk mengakses toko produk.

1) Buat aplikasi Flutter baru di studio Android, product_firebase_app.
2) Ganti kode startup default (main.dart) dengan kode *product_rest_app* kami .

```
class Product {
   final String name;
   final String description;
   final int price;
   final String image;


   Product(this.name, this.description, this.price, this.image);
   factory Product.fromMap(Map<String, dynamic> json) {
      return
         Product( json['name']
         ,
         json['description'],
         json['price'],
         json['image'],
      );
   }
```

3) Salin file Product.dart dari product_rest_app ke dalam folder lib.

Salin folder aset dari product_rest_app ke product_firebase_app dan tambahkan aset di dalam file pubspec.yaml.

```
flutter:
   assets:
   - assets/appimages/floppy.png
   - assets/appimages/iphone.png
   - assets/appimages/laptop.png
   - assets/appimages/pendrive.png
```

```
   - assets/appimages/pixel.png
   - assets/appimages/tablet.png
```

Konfigurasikan paket cloud_firestore di file pubspec.yaml seperti yang ditunjukkan di bawah ini -

1) dependencies: cloud_firestore: ^0.9.13+1
2) Di sini, gunakan versi terbaru paket cloud_firestore.
3) Studio Android akan memberi tahu bahwa pubspec.yaml diperbarui seperti yang ditunjukkan di sini –
4) Klik Dapatkan opsi ketergantungan. Android studio akan mendapatkan paket dari Internet dan mengkonfigurasinya dengan benar untuk aplikasi.
5) Buat proyek di Firebase menggunakan langkah-langkah berikut -
   a) Buat akun Firebase dengan memilih Paket gratis di https://firebase.google.com/pricing/.
   b) Setelah akun Firebase dibuat, ini akan dialihkan ke halaman ringkasan proyek. Ini mencantumkan semua proyek berbasis Firebase dan memberikan opsi untuk membuat proyek baru.
   c) Klik Tambahkan proyek dan itu akan membuka halaman pembuatan proyek.
   d) Masukkan db aplikasi produk sebagai nama proyek dan klik opsi Buat proyek.
   e) Buka * Firebase console.
   f) Klik Ringkasan proyek. Ini membuka halaman tinjauan proyek.

g) Klik ikon android. Ini akan membuka pengaturan proyek khusus untuk pengembangan Android.
h) Masukkan nama Paket Android, com.tutorialspoint.flutterapp.product_firebase_app.
i) Klik Daftarkan Aplikasi. Ini menghasilkan file konfigurasi proyek, google_service.json.
j) Unduh google_service.json lalu pindahkan ke direktori android / app proyek. File ini adalah koneksi antara aplikasi kita dan Firebase.
k) Buka android / app / build.gradle dan sertakan kode berikut -

```
apply plugin: 'com.google.gms.google-services'
```

Buka android / build.gradle dan sertakan konfigurasi berikut -

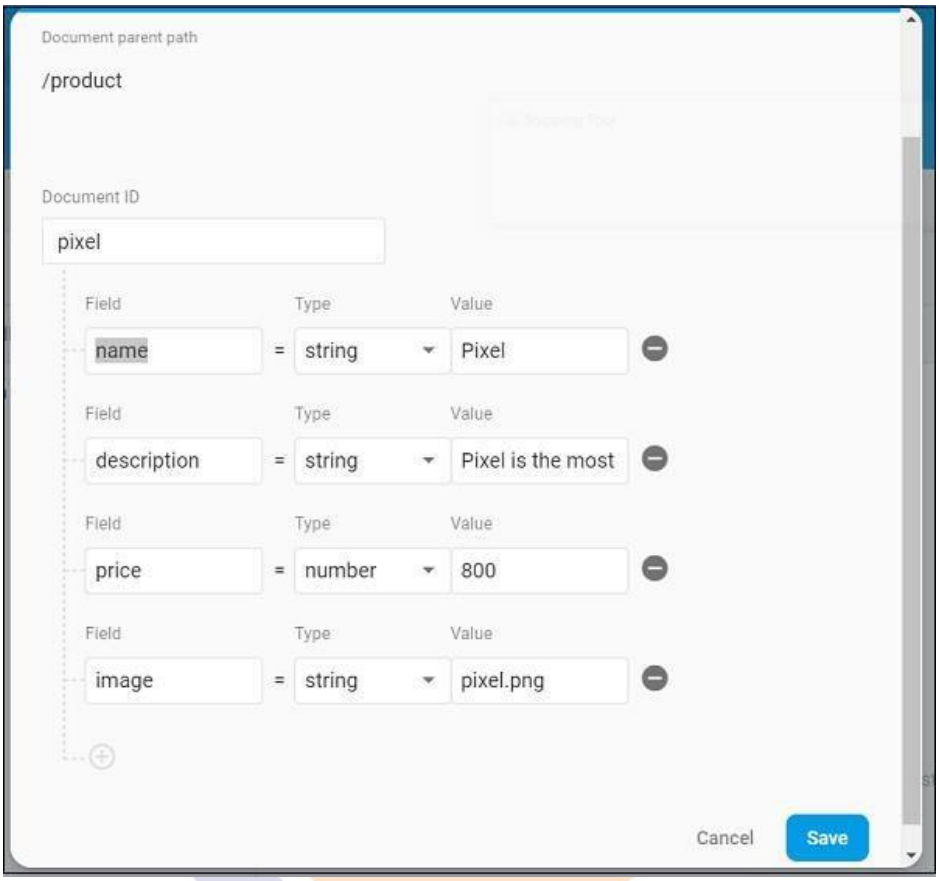
```
buildscript { repositories {
// ...
}
dependencies {
// ...

classpath 'com.google.gms:google-services:3.2.1' // new
}
}
```

plugin dan jalur kelas digunakan untuk tujuan membaca file google_service.json.
Buka android / app / build.gradle dan sertakan juga kode berikut.

```
android {
   defaultConfig {
      ...
      multiDexEnabled true
   }
   ...
}
dependencies {
   ...
   compile 'com.android.support: multidex:1.0.3'
}
```

Ketergantungan ini memungkinkan aplikasi android menggunakan beberapa fungsionalitas dex.

a) Ikuti langkah-langkah selanjutnya di Firebase Console atau lewati saja.
b) Buat toko produk dalam proyek yang baru dibuat menggunakan langkah-langkah berikut –
    a. Buka konsol Firebase.
    b. Buka proyek yang baru dibuat.
    c. Klik opsi Database di menu sebelah kiri.
    d. Klik Buat opsi database.
    e. Klik Mulai dalam mode uji, lalu Aktifkan.
    f. Klik Tambahkan koleksi. Masukkan produk sebagai nama koleksi, lalu klik Berikutnya.
    g. Masukkan informasi produk sampel seperti yang ditunjukkan pada gambar di sini -



a) Tambahkan informasi produk tambahan menggunakan Tambahkan opsi dokumen .
b) Buka file main.dart dan impor file plugin Cloud Firestore dan hapus paket http. import 'package:cloud_firestore/cloud_firestore.dart';
c) Hapus parseProducts dan perbarui fetchProducts untuk mengambil produk dari Cloud Firestore, bukan API layanan Produk. Stream<QuerySnapshot> fetchProducts() { return Firestore.instance.collection('product').snapshots(); }
d) Di sini, metode Firestore.instance.collection digunakan untuk mengakses koleksi produk yang tersedia di cloud store. Firestore.instance.collection menyediakan banyak opsi untuk memfilter koleksi untuk mendapatkan dokumen yang diperlukan. Namun, kami belum menerapkan filter apa pun untuk mendapatkan semua informasi produk.

e) Cloud Firestore menyediakan koleksi melalui konsep Dart Stream sehingga memodifikasi jenis produk di widget MyApp dan MyHomePage dari Future <list <Product>> ke Stream <QuerySnapshot>.

f) Ubah metode build widget MyHomePage untuk menggunakan StreamBuilder, bukan FutureBuilder.

```
@override
Widget build(BuildContext context)
   { return Scaffold(
      appBar: AppBar(title: Text("Product Navigation")),
      body: Center(
         child: StreamBuilder<QuerySnapshot>(
            stream: products, builder: (context, snapshot)
               {          if          (snapshot.hasError)
               print(snapshot.error); if(snapshot.hasData) {
                  List<DocumentSnapshot>
                  documents = snapshot.data.documents;

                  List<Product>
                  items = List<Product>();

                  for(var i = 0; i < documents.length; i++)
                     { DocumentSnapshot document =
                     documents[i];
                     items.add(Product.fromMap(document.data));
                  }
                  return ProductBoxList(items: items);
               } else {
                  return Center(child:
CircularProgressIndicator());
               }
            },
         ),
      )
```

a) Di sini, kami telah mengambil informasi produk sebagai tipe List <DocumentSnapshot>. Karena, widget kami, ProductBoxList tidak kompatibel dengan dokumen, kami telah mengonversikan dokumen ke dalam tipe Daftar <Produk> dan selanjutnya menggunakannya.

b) Terakhir, jalankan aplikasinya dan lihat hasilnya. Karena, kami telah menggunakan informasi produk yang sama seperti yang ada pada aplikasi SQLite dan hanya mengubah media penyimpanan, aplikasi yang dihasilkan terlihat identik dengan aplikasi aplikasi SQLite