GUIDE. 2

Capaian

1. Mahasiswa mampu menambahkan dependensi yang diperlukan serta membuat kelas model untuk mendefinisikan data yang akan disimpan dalam tabel
2. Mahasiswa mampu membuat schema tabel yang dibutuhkan
3. Mahasiswa mampu membuat fungsi fungsi CRUD dengan dilengkapi error handling

Dokumen ini bertujuan untuk menjadi dokumen pengantar materi pembelajaran Pemrograman Flutter dengan Database SQLite. Mahasiswa diharapkan untuk menyiapkan environment yang dibutuhkan untuk mempelajari cara membuat proyek Flutter baru. Tahapan persiapan environment dapat dilihat pada Guide A01.

Menambahkan Dependensi atau *Package*

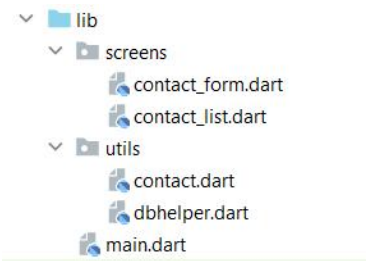
1. Tambahkan package sqflite, path, dan path_provider di puspec.yaml. Ketiga package ini adalah package umum yang harus ditambahkan pada saat membuat sebuah project dengan database sqlite.
2. Lakukan unduh package, dengan mengklik tombol 'Pub Get' di navbar Android Studio sebelah pojok kanan atas
3. Pastikan jika pengunduhan berhasil, dengan ditandai exit status code 0, seperti dibawah ini

```
Messages:  [simple_database] Flutter
D:\src\flutter\bin\flutter.bat --no-color pub get
Running "flutter pub get" in simple_database...                 2,383ms
Process finished with exit code 0
```

4. Jika exit code menunjukkan angka 1, maka terjadi kegagalan. Pesan kegagalan akan muncul untuk memberitahu apa yang terjadi.

Membuat Struktur Direktori

1. Buatlah 2 package/direktori/folder baru di lib dengan nama screens dan utils.
2. Pada folder utils, buat 2 file baru dengan nama dbhelper.dart dan contact.dart. Sedangkan di folder screens buatlah 2 file baru dengan nama contact_list.dart dan form_list.dart
3. Susunan direktori pada folder lib project simple_database menjadi seperti ini:



Membuat Model pada File Contact.dart

1. Model digunakan untuk menyimpan informasi apa saja yang dibutuhkan.
2. Pada class contact.dart di folder utils. Buatlah class Contact dan memiliki variabel berikut ini:

Tipe data	Nama variabel
int?	id
String?	name
	number
	email
	company

Contoh deklarasi variabel:

```
int? id;
```

3. Kemudian tambahkan konstruktor Contact yang memiliki parameter id, required name, required number, required email, dan required company. Required artinya tidak boleh null. Contoh kode untuk konstruktor

```
Contact({
    this.id,
    required this.name,
    …
    …
    …
  });
```

4. Buatlah fungsi toMap dengan tipe data Map<String, dynamic>. Fungsi ini digunakan untuk merubah isian data Contact dalam bentuk map.

```
Map<String, dynamic> toMap() {
  var map = <String, dynamic>{};
  if (id != null) {
    map['id'] = id;
  }
  map['name'] = name;
  map['number'] = number;
  map['email'] = email;
  map['company'] = company;
  return map;
}
```

5. Buatlah fungsi fromMap. Fungsi ini digunakan untuk memetakan data Contact dari map menjadi list atau string.

```
Contact.fromMap(Map<String, dynamic> map) {
  id = map['id'];
  name = map['name'];
  number = map['number'];
  email = map['email'];
  company = map['company'];
}
```

Membuat Class DbHelper pada File DbHelper.dart

1. Pada file dbhelper.dart di folder utils, buatlah class DbHelper.
2. Import 3 package yang dibutuhkan yaitu sqflite, path_provider, dan path. Lalu import model Contact yang telah dibuat (contact.dart) dan import material.dart.
3. Tambahkan deklarasi variabel instance yang bertiper DbHelper sebagai sebuah object dengan kata kunci static dan final. Nilai instance diinisialisasi dengan memanggil kontruktor _internal(). Kata kunci static digunakan untuk membuat variabel dan metode terkait dengan kelas itu sendiri, bukan dengan instance objeknya. Kata kunci final menunjukkan bahwa nilai dari variabel tersebut hanya dapat ditetapkan sekali.

```
static final DbHelper instance = DbHelper._internal();
```

4. Tambahkan deklarasi factory konstruktor. Factory constructor digunakan untuk mengembalikan instance kelas tanpa harus membuat instance baru setiap kali dipanggil. Pada kode ini, factory constructor mengembalikan instance DbHelper yang sama yang telah dibuat sebelumnya melalui variabel instance.

```
factory DbHelper() => instance;
```

5. Tambahkan konstruktor dengan nama _internal() yag bersifat private.

```
DbHelper._internal();
```

6. Buatlah variabel _database dari Database dengan kata kunci static.

```
static Database? _database;
```

7. Buat function initDatabase yang mengembalikan Future<Database?>, menunjukkan bahwa pada akhirnya akan menyediakan objek Database atau null dan bersifat *asynchronous*.

```
Future<Database?> initDatabase() async{};
```

8. Tambahkan code dibawah ini pada fungsi initDatabase(), untuk menginisialisasi komponen Flutter terkait.

```
WidgetsFlutterBinding.ensureInitialized();
```

9. Dibawah code tersebut, tambahkan kode error handling. Kode ini digunakan untuk mengatasi eror yang kemungkinan muncul apabila proses pembuatan database terjadi. Kode error handling seperti dibawah ini

```
try{
…
} catch(e){
   print('Error opening database: $e');
}
```

10. Tambahkan kode berikutnya dibawah syntax try. Buatlah variabel 'databasesPath' yang memiliki value getApplicationDocumentsDirectory(). ode ini digunakan untuk mengambil

jalur ke direktori dokumen aplikasi menggunakan dari paket path_provider. Tambahkan kata kunci final pada variabel, dan await pada value.

```
final databasePath = await getApplicationDocumentsDirectory();
```

11. Kemudian buat variabel 'path' yang menggabungkan antara databasePath dan nama file database 'contactList.db' menggunakan syntax join

```
final path = join(databasesPath.path, 'contactList.db');
```

12. Buatlah fungsi openDatabase() sebagai nilai kembalian dibawah variabel path, Fungsi ini memiliki beberapa parameter yaitu version dengan nilai 1, dimana ini menunjukkan versi awal database, path, dan onCreate adalah fungsi callback yang dijalankan saat database pertama kali dibuat. Di dalam callback ini, tabel dan skema yang diperlukan dapat didefinisikan menggunakan pernyataan SQL.

```
return openDatabase(
       path,
       version: 1,
       onCreate: (db, version) async {
         await db.execute('''
         CREATE TABLE contact(
           …
         )
       ''');
       },
     );
```

13. Untuk membuat tabel contact maka gunakan query sql create table. Atribut yang ada pada tabel contact sebagai berikut:

atribut	Tipe data	Primary key	Auto increment	Not Null
id	INT	YES	YES	YES
name	TEXT	NO	NO	YES
number	TEXT	NO	NO	YES
email	TEXT	NO	NO	YES
company	TEXT	NO	NO	YES

```
CREATE TABLE contact(
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          name TEXT NOT NULL,
          …,
          …,
          …
        )
```

14. Setalah itu buat fungsi getter ‘database’ yang menyediakan akses ke objek database.. Fungsi ini untuk pengecekan apakah database sudah terdefinisikan. Di dalam fungsi ini memanggil fungsi initDatabase(). Tambahkan kodisi if. Ketika variabel _database yang telah dideklarasikan pada nomor 6 tidak sama dengan null maka mengembalikan nilai _database. Masih pada fungsi getter ini, setelah kondisi if inisialisasi variabel _database dengan set value await initDatabase(). Fungsi getter mengembalikan nilai _database!.

```
Future<Database?> get database async {
    if (_database != null) {
      return _database;
    }

    _database = await initDatabase();
    return _database!;
  }
```

15. Buat fungsi ‘saveContact()’ yanng memiliki parameter object Contact dan nilai kembalian berupa int. Pada fungsi ini juga diterapkan error handling. Dalam fungsi ini menggunakan salah satu metode yang ada pada package sqflite yaitu rawInsert(). Nilai yang dimasukkan yaitu name, number, email, dan company

```
Future<int?> saveContact(Contact contact) async {
    var dbClient = await database;
    try {
     int result = await dbClient!.rawInsert(
          "INSERT INTO … (…, …, …, …) VALUES (?, ?, ?, ?)",
          [contact.name, …, …, …]);
    } catch(e){
      print('Error inserting data: $e');
      return 0;
    }
  }
```

16. Buat fungsi ‘getAllContact’ yang digunakan untuk mengambil semua rekaman dari tabel database dan mengembalikannya sebagai list. Pada fungsi ini juga diterapkan error handling. Dalam fungsi ini menggunakan salah satu metode yang ada pada package sqflite yaitu rawQuery()

```
Future<List?> getAllContact() async {
    var db = await database;
    try {
      var result = await db!.rawQuery("SELECT * FROM …");
      return result.toList();
    } catch(e){
      print('Error retrieve all the data: $e');
      return null;
    }
  }
```

17. Lalu buat fungsi ‘updateContact()’ yang memiliki parameter objek Contact dan mengembalikan nilai int. Pada fungsi ini juga diterapkan error handling. Dalam fungsi ini menggunakan salah satu metode yang ada pada package sqflite yaitu rawUpdate()

```
Future<int?> updateContact(Contact contact) async {
    var dbClient = await database;
    try{
    var result = await dbClient!.rawUpdate("UPDATE … SET
name=?, …, …, … WHERE id=?",
        [contact.name, …, …, …, contact.id]);
    return result;
    }catch(e){
      print('update data error: $e');
      return 0;
    }
  }
```

18. Terakhir tambahkan fungsi ‘deleteContact()’ yang memiliki parameter int id dan mengembalikan nilai int. Pada fungsi ini juga diterapkan error handling. Dalam fungsi ini menggunakan salah satu metode yang ada pada package sqflite yaitu rawDelete()

```
Future<int?> deleteContact(int id) async {
    final dbClient = await database;
    try {
      int result = await dbClient!.rawDelete("DELETE FROM …
WHERE id=?", [id]);
      return result;
    } catch(e){
      print('deleting data error: $e');
      return 0;
    }
  }
```

Testing Hasil Pengerjaan

1. Sebelum mengunduh test files, tambahkan dependensi sqflite_common_ffi pada file puspec.yaml dibagian dev_dependencies. Lalu klik Pub Get

2. Unduh test file guide2_test kemudian copy paste kan ke dalam direktori test project anda.

3. Jalankan program dan jika berhasil akan muncul keterangan ‘Test Success’ dan jika gagal akan muncul keterangann ‘Test Failed’ disertai keterangan kegagalannya.