

TECNOLOGICO JOSE MARIO MOLINA
PASQUEL Y ENRRIQUEZ UNIDAD
ACADEMICA ZAPOTLANEJO

MANUAL DE PROGRAMADOR
ADMINISTRACION Y ORGANIZCION DE
DATOS

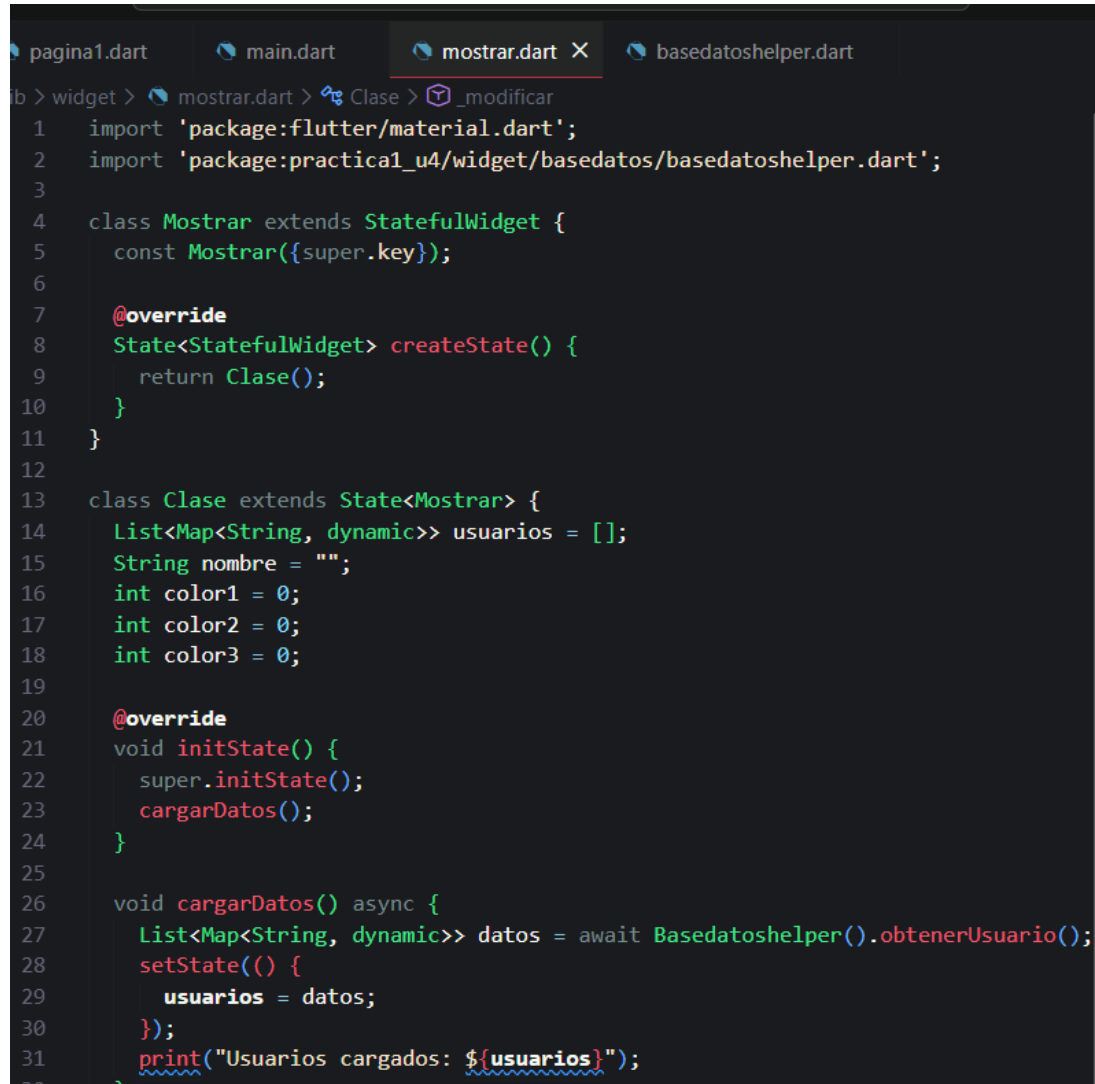
MARCOS ALFREDO RAZO MARTINEZ

Contenido

Programa 1	3
Mostrar.dart.....	3
Basedatoshealper.dart.....	8
Resultado	10

Programa 1

Mostrar.dart

The image is a screenshot of an IDE window showing the code for a file named 'Mostrar.dart'. The window has several tabs at the top: 'pagina1.dart', 'main.dart', 'mostrar.dart' (which is the active tab), and 'basedatoshelper.dart'. The code is written in Dart and defines two classes: 'Mostrar' and 'Clase'. The 'Mostrar' class extends 'StatefulWidget' and has a 'createState()' method that returns a 'Clase' object. The 'Clase' class extends 'State<Mostrar>' and contains several variables: 'usuarios' (a list of maps), 'nombre' (a string), and three integers 'color1', 'color2', and 'color3'. It also has an 'initState()' method that calls 'super.initState()' and 'cargarDatos()', and a 'cargarDatos()' method that is asynchronous and calls 'Basedatoshelper().obtenerUsuario()' to fetch data, then updates the 'usuarios' state and prints a message. The code is line-numbered from 1 to 31.

```
1 import 'package:flutter/material.dart';
2 import 'package:practica1_u4/widget/basedatos/basedatoshelper.dart';
3
4 class Mostrar extends StatefulWidget {
5   const Mostrar({super.key});
6
7   @override
8   State<StatefulWidget> createState() {
9     return Clase();
10  }
11 }
12
13 class Clase extends State<Mostrar> {
14   List<Map<String, dynamic>> usuarios = [];
15   String nombre = "";
16   int color1 = 0;
17   int color2 = 0;
18   int color3 = 0;
19
20   @override
21   void initState() {
22     super.initState();
23     cargarDatos();
24   }
25
26   void cargarDatos() async {
27     List<Map<String, dynamic>> datos = await Basedatoshelper().obtenerUsuario();
28     setState(() {
29       usuarios = datos;
30     });
31     print("Usuarios cargados: ${usuarios}");
32   }
33 }
```

```

void cargarDatos() async {
  usuarios = datos;
});
print("Usuarios cargados: {usuarios}");
}

void eliminar(int id) async {
  await Basedatoshelper().eliminar(id);
  cargarDatos();
}

void _eliminar(int id) {
  eliminar(id);
}

void _modificar(int id, String usuario, String password) {
  TextEditingController usuarionuevo = TextEditingController();
  TextEditingController passwordnuevo = TextEditingController();

  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text('Editar Usuario'),
        content: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.min,
            children: [
              TextField(
                controller: usuarionuevo,
                decoration: InputDecoration(labelText: '{usuario}'),

```

```

        textField(
            controller: passwordnuevo,
            decoration: InputDecoration(labelText: 'Escribe el password'),
            obscureText: true,
        ), // TextField
    ],
), // Column
), // SingleChildScrollView
actions: [
    TextButton(
        onPressed: () async {
            await Basedatoshelper().modificar(
                id,
                usuarionuevo.text,
                passwordnuevo.text,
            );
            cargarDatos();
            Navigator.pop(context);
        },
        child: Text('Guardar'),
    ), // TextButton
    TextButton(
        onPressed: () => Navigator.pop(context),
        child: Text('Cancelar'),
    ), // TextButton
]

```

```

        child: Text( 'Cancelar' ),
      ), // TextButton
    ],
  ); // AlertDialog
},
);
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Mostrar datos'),
      backgroundColor: Colors.limeAccent,
    ), // AppBar
    body: usuarios.isEmpty
      ? Center(child: Text('Base de datos vacía'))
      : ListView.builder(
          itemCount: usuarios.length,
          itemBuilder: (context, index) {
            nombre = usuarios[index]['usuario'][0];
            return ListTile(
              leading: CircleAvatar(
                //backgroundColor: Color.fromARGB(0, 0, 0, 0),
                child: Text(
                  nombre.toUpperCase(),
                  style: TextStyle(color: Colors.black, fontSize: 24),
                ), // Text
              ),
            );
          },
        );
}

```

```

        style: TextStyle(color: Colors.black, fontSize: 24),
      ), // Text
    ), // CircleAvatar
    title: Text(usuarios[index]['usuario']),
    subtitle: Text(usuarios[index]['password']),
    trailing: Wrap(
      direction: Axis.horizontal,
      children: [
        IconButton(
          onPressed: () {
            _eliminar(usuarios[index]['id']);
          },
          icon: Icon(Icons.delete),
        ), // IconButton
        IconButton(
          onPressed: () {
            _modificar(
              usuarios[index]['id'],
              usuarios[index]['usuario'],
              usuarios[index]['password'],
            );
          },
          icon: Icon(Icons.update),
        ), // IconButton
      ],
    ), // Wrap
  ); // ListTile
}, // ListView.builder
); // Scaffold
}

```

```

        ), // IconButton
      ],
    ), // Wrap
  ); // ListTile
}, // ListView.builder
); // Scaffold
}
}

```

Basedatoshealper.dart

```
pagina1.dart  main.dart  mostrar.dart  basedatoshelper.dart X
> widget > basedatos > basedatoshelper.dart > Basedatoshelper
1  import 'package:path/path.dart';
2  import 'package:sqflite/sqflite.dart';
3
4  class Basedatoshelper {
5    static final Basedatoshelper _instance = Basedatoshelper._internal();
6    factory Basedatoshelper() => _instance;
7
8    Basedatoshelper._internal();
9
10   Database? _database;
11
12   Future<Database> get database async {
13     if (_database != null) return _database!;
14     _database = await _initDB();
15     return _database!;
16   }
17
18   Future<Database> _initDB() async {
19     String path = join(await getDatabasesPath(), 'usuarios.db');
20     return await openDatabase(
21       path,
22       version: 1,
23       onCreate: (db, version) async {
24         await db.execute('''
25           CREATE TABLE usuarios(
26             id INTEGER PRIMARY KEY AUTOINCREMENT,
27             usuario TEXT NOT NULL,
28             password TEXT NOT NULL
29           )
30         ''');
31       },
32     );
33   }
34 }
```



```

        password TEXT NOT NULL
    )
    ''');
},
);
}

Future<int> insertar(String usuario, String password) async {
    final db = await database;
    return await db.insert('usuarios', {
        'usuario': usuario,
        'password': password,
    });
}

Future<List<Map<String, dynamic>>> obtenerUsuario() async {
    final db = await database;
    return await db.query('usuarios');
}

Future eliminar(int id) async {
    final db = await database;
    return await db.delete('usuarios', where: 'id = ?', whereArgs: [id]);
}

Future<int> modificar(int nid, String nusuario, String npassword) async {
    final db = await database;
    return await db.update(

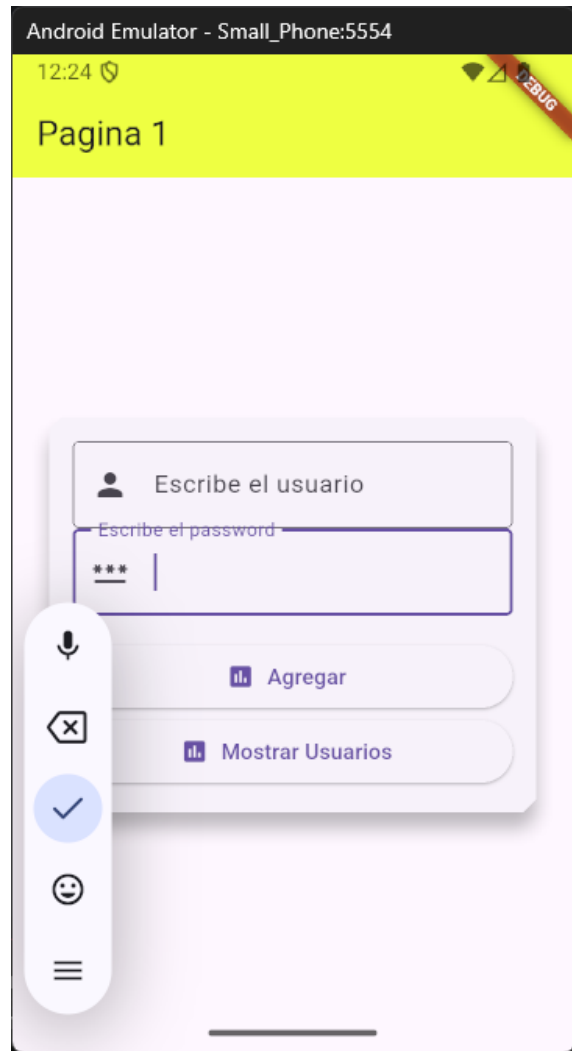
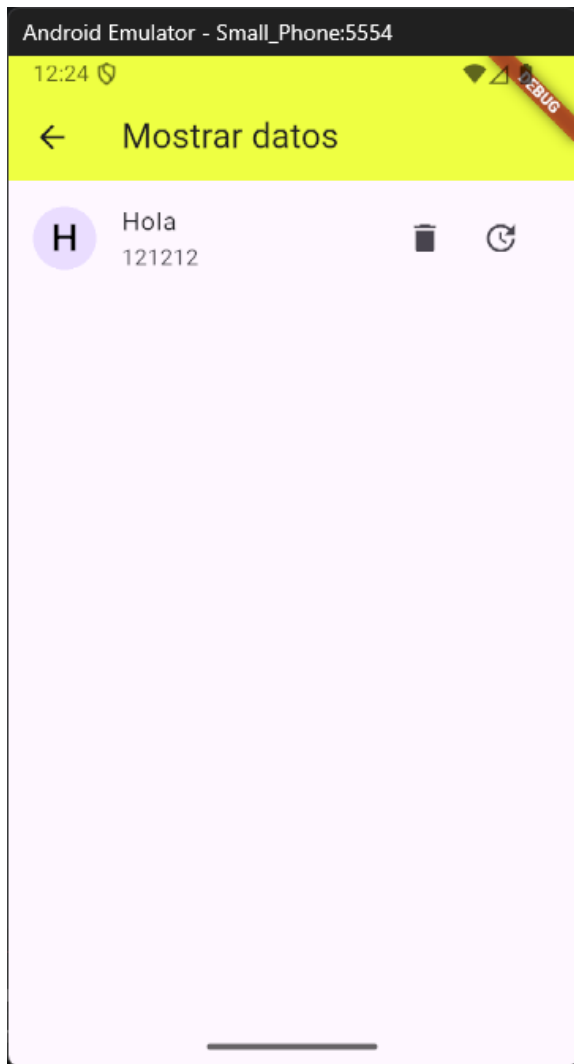
```

```

        final db = await database;
        return await db.update(
            'usuarios',
            {'usuario': nusuario, 'password': npassword},
            where: 'id = ?',
            whereArgs: [nid],
        );
    }
}

```

Resultado



QrsqLite

Pagina1.dart

```
pagina1.dart X
lib > widget > pagina1.dart > Clase
1 import 'package:flutter/material.dart';
2 import 'package:mobile_scanner/mobile_scanner.dart';
3 import 'package:qrsqLite/basedatos.dart/basedatoshelper.dart';
4
5 class Pagina1 extends StatefulWidget {
6   const Pagina1({super.key});
7
8   @override
9   State<StatefulWidget> createState() {
10     return Clase();
11   }
12 }
13
14 class Clase extends State<Pagina1> {
15   // 1) Bandera para evitar abrir múltiples diálogos al detectar varios frames del QR.
16   bool ventana = false;
17
18   // 2) Controladores creados una sola vez y limpiados en dispose para evitar fugas de memoria.
19   final TextEditingController nombre = TextEditingController();
20   final TextEditingController precio = TextEditingController();
21
22   @override
23   void dispose() {
24     // 3) Liberar recursos al cerrar la pantalla.
25     nombre.dispose();
26     precio.dispose();
27     super.dispose();
28   }
29
30   // 4) Utilidad: Limpiar campos antes de mostrar el diálogo.
31   void _limpiarCampos() {
32     nombre.clear();
33   }
34 }
```

Flutter: Building Windows application...

```

class Clase extends State<Pagina1> {
  void _limpiarCampos() {
    nombre.clear();
    precio.clear();
  }

  // 5) Diálogo para capturar datos; recibe el código QR como 'numeros'.
  Future<void> _mostrarDatos(String numeros) async {
    _limpiarCampos(); // 5.1) Asegurar campos en blanco al abrir.

    // 5.2) showDialog devuelve un Future; esperamos (await) para saber cuándo cerrar y poder resetear 'v
    await showDialog(
      context: context,
      barrierDismissible:
        false, // 5.3) Evitar cierres accidentales al tocar fuera.
      builder: (context) {
        return AlertDialog(
          title: const Text('Agregar producto'),
          content: SingleChildScrollView(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.min,
              children: [
                // 5.4) Mostrar el código leído.
                Text(
                  'El código: $numeros',
                  style: const TextStyle(fontSize: 15),
                ), // Text
                const SizedBox(height: 12),
                // 5.5) Campo nombre (texto normal).
                TextField(
                  controller: nombre,

```

Flutter: Running Gradle task 'assembleDebug'...

```

Future<void> _mostrarDatos(String numeros) async {
  // 5.5) Campo nombre (texto normal).
  TextField(
    controller: nombre,
    decoration: const InputDecoration(
      labelText: 'Escribe el nombre del producto',
      border: OutlineInputBorder(),
    ), // InputDecoration
    textInputAction: TextInputAction.next,
  ), // TextField
  const SizedBox(height: 12),
  // 5.6) Campo precio (numérico). Quitar obscureText: no es contraseña.
  TextField(
    controller: precio,
    decoration: const InputDecoration(
      labelText: 'Escribe el precio',
      border: OutlineInputBorder(),
    ), // InputDecoration
    keyboardType: TextInputType.numberWithOptions(decimal: true),
  ), // TextField
],
), // Column
), // SingleChildScrollView
actions: [
  // 5.7) Guardar: validar mínimo y luego insertar.
  TextButton(
    onPressed: () async {
      final codigo = numeros.trim();
      final nom = nombre.text.trim();
      final pre = precio.text.trim();

```

Flutter: Running Gradle task

```

void _mostrarDatos(String numeros) async {
  TextButton(
    onPressed: () async {
      final codigo = numeros.trim();
      final nom = nombre.text.trim();
      final pre = precio.text.trim();

      // 5.7.1 Validaciones básicas para UX clara.
      if (nom.isEmpty) {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text('El nombre no puede estar vacío'),
          ), // SnackBar
        );
        return;
      }
      if (pre.isEmpty) {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text('El precio no puede estar vacío'),
          ), // SnackBar
        );
        return;
      }
      // 5.7.2 Validar que el precio sea número válido (decimal).
      final precioNum = double.tryParse(pre);
      if (precioNum == null || precioNum <= 0) {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text('Ingresa un precio válido')
          ), // SnackBar
        );
        return;
      }
    },
  );
}

```

Flutter: Running

```


class Clase extends State<Pagina1> {
  Future<void> _mostrarDatos(String numeros) async {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Ingresa un precio válido mayor a 0'),
      ), // SnackBar
    );
    return;
  }

  try {
    // 5.7.3) Insertar en BD; guardar precio como texto (consistente
    await Basedatoshelper().insertar(codigo, nom, pre);

    // 5.7.4) Feedback de éxito opcional.
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Producto guardado')),
    );

    // 5.7.5) Cerrar el diálogo tras guardar.
    Navigator.pop(context);
  } catch (e) {
    // 5.7.6) Manejo de error en inserción.
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Error al guardar: $e')),
    );
  }
},
child: const Text('Guardar'),
), // TextButton
// 5.8) Cancelar: cerrar sin acciones.

```

 Flutter: Running Gradle

```

class Clase extends State<Pagina1> {
  Future<void> _mostrarDatos(String numeros) async {
    // 5.8) Cancelar: cerrar sin acciones.
    TextButton(
      onPressed: () => Navigator.pop(context),
      child: const Text('Cancelar'),
    ), // TextButton
  ],
); // AlertDialog
},
);

// 5.9) Al cerrar el diálogo (éxito o cancelación), permitir otro escaneo.
setState(() {
  ventana = false;
});
}

```

```

class Clase extends State<Pagina1> {
  @override
  Widget build(BuildContext context) {
    // 6) Estructura principal de la pantalla.
    return Scaffold(
      appBar: AppBar(
        title: const Text('QR con SQLite'),
        backgroundColor: const Color.fromARGB(255, 5, 255, 17),
      ), // AppBar
      // 7) Lector de QR: 'onDetect' puede dispararse muchas veces por segundo.
      body: MobileScanner(
        onDetect: (capture) {
          // 7.1) Evitar múltiples diálogos simultáneos.
          if (ventana) return;

          // 7.2) Tomar el primer código detectado.
          final barcode = capture.barcodes.first;
          final numeros = barcode.rawValue ?? 'Sin codigo';

          // 7.3) Si hay código válido, abrir diálogo y bloquear nueva ventana.
          if (numeros != 'Sin codigo' && numeros.trim().isNotEmpty) {
            setState(() {
              ventana = true;
            });
            _mostrarDatos(numeros);
          }
        },
      ), // MobileScanner
    ); // Scaffold
  }
}

```

Basedatoshelper.dart

```
basedatoshelper.dart X
lib > basedatos.dart > basedatoshelper.dart > Basedatoshelper > _initDB
1  import 'package:path/path.dart';
2  import 'package:sqflite/sqflite.dart';
3
4  class Basedatoshelper {
5    // 1) Singleton: misma instancia en toda la app.
6    static final Basedatoshelper _instance = Basedatoshelper._internal();
7    factory Basedatoshelper() => _instance;
8    Basedatoshelper._internal();
9
10   // 2) Referencia a la BD (perezosa).
11   Database? _database;
12
13   // 3) Getter: abre o devuelve la BD existente.
14   Future<Database> get database async {
15     if (_database != null) return _database!;
16     _database = await _initDB();
17     return _database!;
18   }
19
20   // 4) Inicialización: crear/abrir archivo 'abarrotes.db'.
21   Future<Database> _initDB() async {
22     final dbPath = await getDatabasesPath();
23     final path = join(dbPath, 'abarrotes.db');
24
25     // 4.1) Abrir BD, crear tabla si versión 1.
26     return await openDatabase(
27       path,
28       version: 1,
29       onCreate: (db, version) async {
30         // 4.2) Esquema corregido: clave primaria autoincremental en 'id' y 'cantidad'.
31         await db.execute('''
32           CREATE TABLE productos(

```

Flutter: Launching

basedatos.dart / basedatoshelper.dart / basedatoshelper / _initDB

```
class Basedatoshelper {
  Future<Database> _initDB() async {
    CREATE TABLE productos(
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      codigo TEXT NOT NULL UNIQUE,
      nombre TEXT NOT NULL,
      precio REAL NOT NULL
    )
    '';
  },
);
}

// 5) Inserción: devuelve el id del nuevo registro.
Future<int> insertar(String codigo, String nombre, String precio) async {
  final db = await database;

  // 5.1) Convertir precio a número para esquema REAL.
  final precioNum = double.tryParse(precio) ?? 0.0;

  return await db.insert(
    'productos',
    {'codigo': codigo, 'nombre': nombre, 'precio': precioNum},
    // 5.2) En caso de 'codigo' duplicado, fallará con excepción (capturada en UI).
    conflictAlgorithm: ConflictAlgorithm.abort,
  );
}

// 6) (Opcional) Obtener todos los productos.
Future<List<Map<String, dynamic>>> obtenerProductos() async {
  final db = await database;
  return await db.query('productos', orderBy: 'id DESC');
```

Flutter: Running Gradle ta

```

// 6) (Opcional) Obtener todos los productos.
Future<List<Map<String, dynamic>>> obtenerProductos() async {
  final db = await database;
  return await db.query('productos', orderBy: 'id DESC');
}

// 7) (Opcional) Eliminar por id.
Future<int> eliminar(int id) async {
  final db = await database;
  return await db.delete('productos', where: 'id = ?', whereArgs: [id]);
}

// 8) (Opcional) Actualizar nombre y precio por código.
Future<int> modificarPorCodigo(
  String codigo,
  String nombre,
  double precio,
) async {
  final db = await database;
  return await db.update(
    'productos',
    {'nombre': nombre, 'precio': precio},
    where: 'codigo = ?',
    whereArgs: [codigo],
    conflictAlgorithm: ConflictAlgorithm.abort,
  );
}

```

Resultado