ENCS5150 · EXP 4 COMPANION

Storing Data with SQLite

Where an app keeps its data, how SQLiteOpenHelper manages the file, how to read and write safely, and what Room does for you.

Five questions

01 Where can an app keep its data?

02 What does SQLiteOpenHelper do for you?

03 How do you read and write without leaks or injection?

04 What breaks once real users have data?

05 Is there something better? Room.

01 · WHERE DATA LIVES

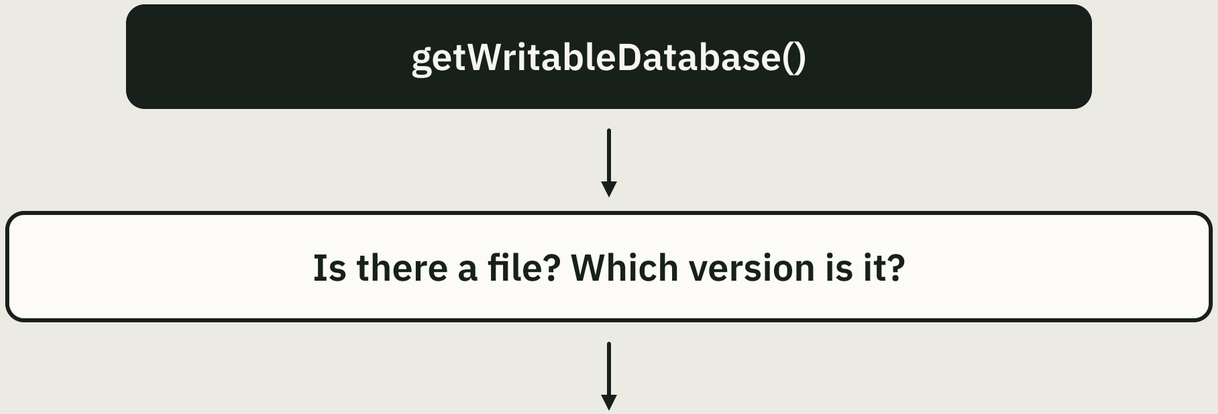
Four places an app can keep data

Where	Holds	Example
SharedPreferences (Exp 6)	A few single key–value settings	The last username
SQLite database (this lab)	Many rows of the same shape, searchable with SQL	A list of customers
Files	Bytes: images, downloads, exports	A profile photo
A server	Data shared between users and devices	Messages (Exp 8 onward)

The first three live in the app’s private folder, **/data/data/<package>/**. Other apps can’t read it, and uninstalling the app deletes it.

02 · SQLITEOPENHELPER

The helper decides what the database needs



NO FILE YET

onCreate(db)

Run CREATE TABLE. Once in the app’s life, not on every launch.

OLDER VERSION

onUpgrade(db, old, new)

Change the schema of a database that already holds data.

SAME VERSION

Just open it

Nothing to do. This is almost every call.

SQL statements and their Android methods

SQL	SQLiteDatabase method	Returns
INSERT	insert(table, null, contentValues)	The new row id, or –1 if it failed
SELECT	query(table, columns, where, args, …, orderBy)	A Cursor
UPDATE	update(table, contentValues, where, args)	How many rows changed
DELETE	delete(table, where, args)	How many rows were deleted

```
ContentValues values = new ContentValues();
values.put(COLUMN_NAME, "Sara");
long id = db.insert(TABLE_CUSTOMER, null, values);
```

insert never throws. A duplicate primary key just returns –1, so check it, as the lab does, or the customer silently goes missing.

03 · READING AND WRITING

A Cursor walks the result one row at a time

Before the first row: the cursor starts here

Row 1 · Id 1 · Sara

Row 2 · Id 2 · Ahmad

Row 3 · Id 5 · Lina

moveToNext() returns false: the loop ends

```
try (Cursor cursor = db.query(TABLE_CUSTOMER,
        null, null, null, null, null, COLUMN_ID)) {
    int nameIndex =
        cursor.getColumnIndexOrThrow(COLUMN_NAME);
    while (cursor.moveToNext()) {
        String name = cursor.getString(nameIndex);
    }
} // closed here, even after an exception
```

Close every cursor, and look columns up by name: **getString(0)** silently reads the wrong column if the table changes.

Never paste user input into SQL

DANGEROUS

```
db.execSQL("DELETE FROM CUSTOMER WHERE NAME = '" + name + "'");
```

If the user types **x' OR '1'='1** as the name, the statement becomes **… WHERE NAME = 'x' OR '1'='1'**, and every customer is deleted.

SAFE

```
db.delete(TABLE_CUSTOMER, COLUMN_NAME + " = ?", new String[]{ name });
```

The **?** is a placeholder. SQLite treats the argument as a value, never as SQL, whatever it contains.

04 · WHAT BREAKS IN REAL APPS

Disk work belongs off the main thread

The lab is fine

A few customers load in a millisecond, so the lab queries on the main thread.

Real data is not

Thousands of rows, or a slow phone, freeze the screen while the query runs.

Room refuses

Room throws an IllegalStateException for a query on the main thread, unless you opt out.

```
ExecutorService executor =
    Executors.newSingleThreadExecutor();

executor.execute(() → {
    // background thread: read the database
    List<Customer> customers = helper.getAllCustomers();
    // main thread: touch the views
    runOnUiThread(() → showCustomers(customers));
});
```

04 · WHAT BREAKS IN REAL APPS

Changing the schema without losing data

THE LAB · DROP AND RECREATE

```
db.execSQL("DROP TABLE IF EXISTS CUSTOMER");
onCreate(db);
```

Fine while you develop. In a published app, every user loses every customer on the update.

REAL APPS · MIGRATE

```
if (oldVersion < 2) {
    db.execSQL("ALTER TABLE CUSTOMER"
        + " ADD COLUMN EMAIL TEXT");
}
```

One step per version, so a user who skipped versions gets each change in order.

Bump **DATABASE_VERSION** every time the schema changes, or onUpgrade never runs and the first query on the new column crashes.

05 · IS THERE SOMETHING BETTER?

Room: SQLite with the boilerplate generated

```
@Entity
public class Customer {
    @PrimaryKey public long id;
    public String name;
}

@Dao
public interface CustomerDao {
    @Query("SELECT * FROM Customer ORDER BY id")
    List<Customer> getAll();

    @Insert
    void insert(Customer customer);
}
```

SQL checked when you build

A typo in a @Query is a compile error, not a crash.

No cursors

Room writes the code that turns rows into Customer objects.

Same SQLite underneath

The file and the SQL are the same. Google recommends Room over using SQLite directly.

TAKEAWAYS

Four things to remember

01 onCreate runs once per install; onUpgrade runs when you bump the version.

02 Check insert’s –1, close every cursor, read columns by name.

03 User input goes in ? arguments, never into the SQL string.

04 Real apps use Room, off the main thread, with migrations.