Good move 👍 — this is **WSTG-INPV-08: Testing for Client-Side SQL Injection**.  
This is less common than server-side SQLi, but still dangerous.

**🧪 Testing for Client-Side SQL Injection**

**🔎 What is it?**

* **Client-side SQL Injection** happens when the application uses **client-side databases** (like **SQLite**, **WebSQL**, **IndexedDB**) and inserts **unsanitized user input** into queries executed directly in the browser or client app.
* Example places:
  + **Web apps** with WebSQL (deprecated but still found in old code).
  + **Mobile apps** (Android/iOS) using SQLite.
  + **Desktop apps** that bundle SQLite for storage.

**⚡ Example Scenarios**

**1. WebSQL in JavaScript**

// vulnerable code

db.transaction(function(tx) {

tx.executeSql("SELECT \* FROM users WHERE name = '" + userInput + "'");

});

Payload:

' OR '1'='1

**2. Android App with SQLite**

// vulnerable

Cursor c = db.rawQuery("SELECT \* FROM users WHERE id='" + id + "'", null);

Payload:

1; DROP TABLE users--

**🔧 Testing Approach**

**With Burp Suite**

1. Look for requests where the server just reflects data but the logic happens **in the client**.
2. Modify parameters with SQL payloads (' OR '1'='1--, ') OR ('a'='a).
3. Use **Burp Repeater** → check how the app behaves (extra data returned, bypassed login, crashes).

**Manual Testing in Browser Console**

If WebSQL is used:

// attacker injects payload in input field

// then inside devtools console

db.transaction(function(tx){ tx.executeSql("SELECT \* FROM users"); });

**Using sqlmap**

Normally **sqlmap** targets server DBs.  
But if the client app sends SQL-like input (e.g., SQLite queries in Android apps), you can extract the APK, analyze queries, and fuzz them manually.

**🔍 Signs of Vulnerability**

* App crashes with DB errors (sqlite3.OperationalError, WebSQL syntax error).
* Data returned beyond what should be accessible.
* Authentication bypass on client side (e.g., offline mobile app).

**✅ Mitigation**

* Never concatenate user input in client-side SQL queries.
* Use **parameterized queries** (e.g., db.rawQuery("SELECT \* FROM users WHERE id=?", new String[]{id});).
* Validate and sanitize input before it ever reaches the query.
* Prefer **IndexedDB** over WebSQL (more structured API).

⚡ **Summary:**

* Client-side SQLi targets **SQLite/WebSQL/embedded DBs** in browsers, mobile apps, or desktop apps.
* Test using **Burp**, browser console, or mobile app decompilation.
* Errors or authentication bypass show the weakness.
* Mitigate with parameterized queries and input validation.

👉 Do you want me to also write **step-by-step Burp Suite test workflow** specifically for catching client-side SQLi in web apps (with WebSQL), since that’s common in WSTG labs?