Blind SQL injection with conditional errors

1. This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics and performs a SQL query containing the value of the submitted cookie.

2. The results of the SQL query are not returned, and the application does not respond any differently based on whether the query returns any rows. If the SQL query causes an error, then the application returns a custom error message.

3. When I close the query with single quote (‘), the application responds with error message and when I close the query with double single quote (‘’) the application responds normally. I will try to inject the malicious query between single quotes using concatenation.

   cookies: TrackingId=iPMwP132b6XpbR4Q’ || <malicious query> || ‘

4. When I try to inject normal query like **SELECT ‘’**, the application responds with error, but when I inject **SELECT ‘’ FROM dual**, the application didn’t throw an error message that means that the database is Oracle database.

5. From cheat sheet I get a query for conditional errors.
   SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN TO_CHAR(1/0) ELSE NULL END FROM dual
   After some modifications the query becomes:
   ‘ || SELECT CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE NULL END FROM users WHERE username=’administrator’ and SUBSTR(password, X, 1) = ‘Y’ || ‘

6. Then I use the same python script as the previous lab but with the new query (instead of burpsuite pro) to brute force X, Y values to extract the administrator’s password.
   - What does this query do? Let’s break it down.

     I close the current string with single quote, concatenate the result of a subquery into the surrounding SQL (|| ... ||), then reopens it again with another single quote.

The subquery checks if the first character of the administrator's password = 'a'? If yes, it forces a division-by-zero inside **TO_CHAR(1/0)**, which in Oracle produces a runtime error that is usually returned.
If not, it returns an empty string. Observing whether an error occurs tells us a boolean answer (True).
CASE WHEN (1=1) THEN ... ELSE ... END: a conditional expression. **1=1** here is just an example. in a real attack the 1=1 will be replaced by the condition that we want to test (or we can leave it as 1=1 for demonstration). In our payload the real conditional of interest is **SUBSTR(password,1,1)='a'** in the **WHERE** clause, not the 1=1 inside CASE; both are used to control flow.
TO_CHAR(1/0): attempts to divide 1 by 0, then convert the result to text. In Oracle 1/0 raises a numeric exception (division by zero) at runtime. That exception causes the database to raise an error which is visible to us.
ELSE ‘’: if the **CASE** falls into **ELSE** (i.e., the row isn't found), it returns an empty string (no error).
If a row for administrator **exists** and its first password character is 'a', the **FROM ... WHERE ...** returns that row, the **CASE** is evaluated, the **TO_CHAR(1/0)** runs and throws a division-by-zero error. If the first character is **not** 'a', either the **WHERE** returns no rows (so subquery yields NULL or empty) or the CASE isn't triggered, so no division-by-zero and no error.

7. Final step is running the script and getting the password and login.