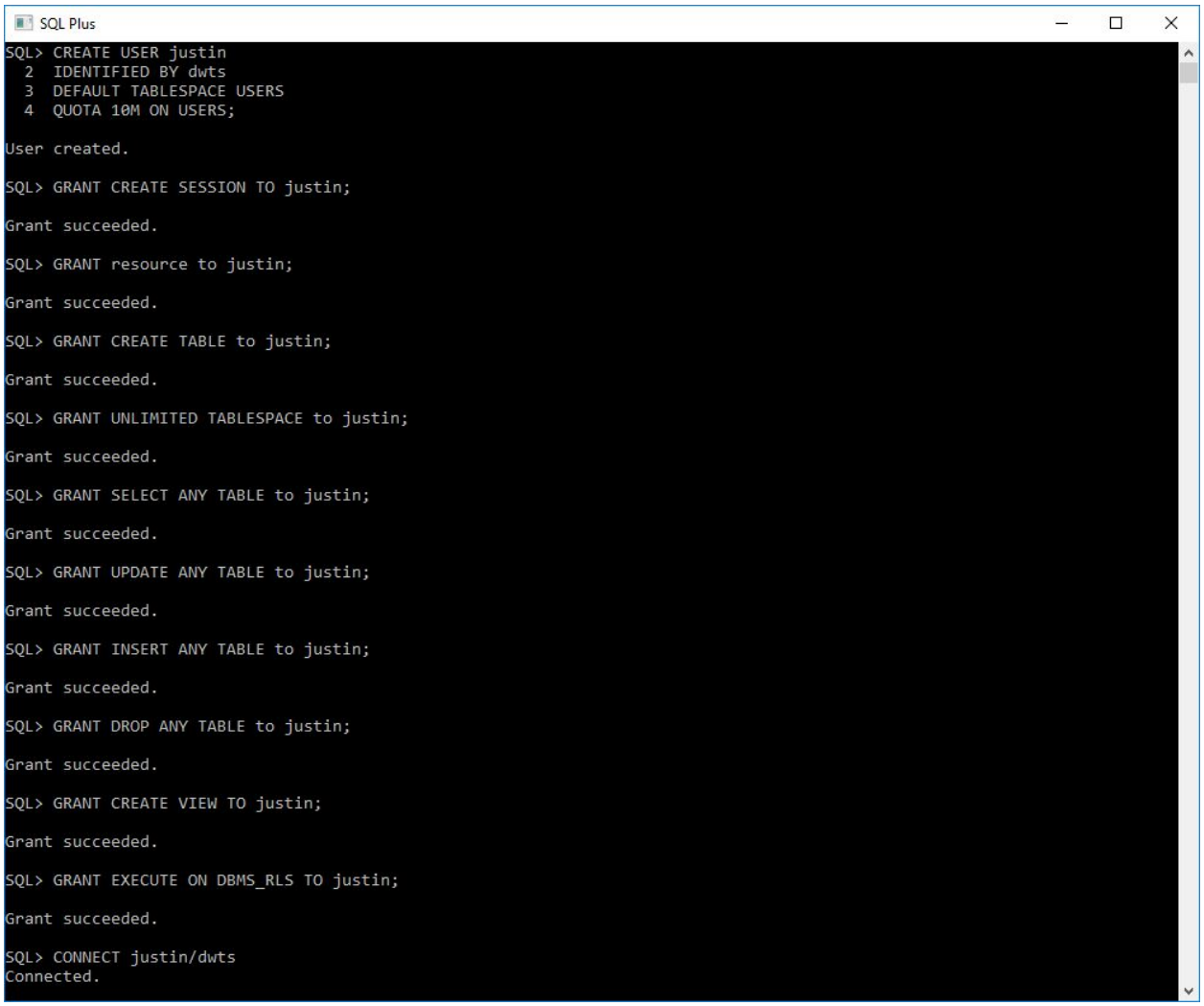
First we use the “SYS” user to create a new user named “justin”. We do this because running commands under the SYS user is bad practice and you should always run your commands under a created user in almost all cases. We grant this user the ability to create, edit, and drop tables as well as giving access to the DBMS_RLS Oracle package to create a virtual private database.



```
SQL Plus
SQL> CREATE USER justin
  2  IDENTIFIED BY dwts
  3  DEFAULT TABLESPACE USERS
  4  QUOTA 10M ON USERS;

User created.

SQL> GRANT CREATE SESSION TO justin;

Grant succeeded.

SQL> GRANT resource to justin;

Grant succeeded.

SQL> GRANT CREATE TABLE to justin;

Grant succeeded.

SQL> GRANT UNLIMITED TABLESPACE to justin;

Grant succeeded.

SQL> GRANT SELECT ANY TABLE to justin;

Grant succeeded.

SQL> GRANT UPDATE ANY TABLE to justin;

Grant succeeded.

SQL> GRANT INSERT ANY TABLE to justin;

Grant succeeded.

SQL> GRANT DROP ANY TABLE to justin;

Grant succeeded.

SQL> GRANT CREATE VIEW TO justin;

Grant succeeded.

SQL> GRANT EXECUTE ON DBMS_RLS TO justin;

Grant succeeded.

SQL> CONNECT justin/dwts
Connected.
```

We create two tables, one named “customers” and the other named “products” which will store the information about the customers and products. We then insert 5 rows of information into the customers table and commit our entries.

```
SQL Plus
SQL> CREATE TABLE customers(
  2     id    NUMBER(6),
  3     first_name    VARCHAR2(20) NOT NULL,
  4     last_name     VARCHAR2(20) NOT NULL,
  5     address       VARCHAR2(60) NOT NULL,
  6     city          VARCHAR2(20) NOT NULL,
  7     zipcode NUMBER(6)    NOT NULL,
  8     phone VARCHAR2(20),
  9     CONSTRAINT customer_id_pk PRIMARY KEY(id));

Table created.

SQL> commit;

Commit complete.

SQL> CREATE TABLE products(
  2     product_id    NUMBER(6),
  3     name          VARCHAR2(20) NOT NULL,
  4     price         VARCHAR2(20) NOT NULL,
  5     description   VARCHAR2(60),
  6     CONSTRAINT product_id_pk PRIMARY KEY(product_id));

Table created.

SQL> commit;

Commit complete.

SQL> INSERT INTO customers VALUES ('000001', 'Justin', 'Casteel', '1 Main Street', 'Centreville', '21617', '4104651457' )
;

1 row created.

SQL> INSERT INTO customers VALUES ('000002', 'Jerreak', 'Purnell', '54 Romancoke Drive', 'Grasonville', '21638', ' ' );

1 row created.

SQL> INSERT INTO customers VALUES ('000003', 'Augustus', 'Gloop', '100 Elm Street', 'Centreville', '21617', '' );

1 row created.

SQL> INSERT INTO customers VALUES ('000004', 'Orlando', 'Hogan', '87 Obon', 'Centreville', '21617', '4105874525' );

1 row created.

SQL> INSERT INTO customers VALUES ('000005', 'Julia', 'Hogan', '16 Poplar Drive', 'Chester', '21619', '4102148965' );

1 row created.

SQL> commit;

Commit complete.
```

We insert product information into the products table as well and commit our entries. We now have two tables that contain 5 rows of information each. I used the customers and products tables specifically because I thought that this would be a great example on a real website. The customers table holds all customer info such as name, address, zipcode and the products table lists all of the different products available to purchase from the store each having its own id number. If you did not want to show a specific product to a customer you could simply create a function and policy and then apply it to a specific user and they would not be able to view that information. This could be used for example when you wanted to not show something because the customer isn't a certain age, or the product may be out of stock at that time.

```
SQL Plus
SQL> INSERT INTO products VALUES ('000001', 'Football', '10.99', 'NFL certified football.');

1 row created.

SQL> INSERT INTO products VALUES ('000002', 'Baseball', '8.99', 'MLB certified baseball.');

1 row created.

SQL> INSERT INTO products VALUES ('000003', 'Mousepad', '15.99', 'SteelSeries XL Mousepad');

1 row created.

SQL> INSERT INTO products VALUES ('000004', 'Mouse', '59.99', 'Logitech G500s Mouse');

1 row created.

SQL> INSERT INTO products VALUES ('000005', 'HyperX Headphones', '8.99', 'HyperX Cloud II Headphones');

1 row created.

SQL> commit;

Commit complete.
```

We create two functions, one called “no_customers_num1” and another called “no_product_num5”. What each of these functions do is quite simple. On line 7 of both queries we see:

Return ‘id !=1’;

From the no_customers_num1 function, and:

Return ‘product_id != 5’;

From the no_product_num5 function.

What they mean is that when we try to do a query such as “Select id FROM customers;” it will not show the customer with the id of 1 but will show the rest of the data where the id is not 1. This is also the same for the products table, so when we query the products table it will no longer show any product where the id is equal to 5. For these to work properly we will need to create a policy for these functions.

SQL Plus

```
SQL> create or replace function no_customers_num1 (
  2  p_schema in varchar2,
  3  p_object in varchar2)
  4  return varchar2
  5  as
  6  begin
  7  return 'id != 1';
  8  end;
  9  /

Function created.

SQL> create or replace function no_product_num5 (
  2  p_schema in varchar2,
  3  p_object in varchar2)
  4  return varchar2
  5  as
  6  begin
  7  return 'product_id != 5';
  8  end;
  9  /

Function created.
```

We create two policies so that our above functions work properly when we try to retrieve information from the customers and products tables.

```
SQL Plus
SQL> begin
  2  dbms_rls.add_policy
  3  (object_schema => 'JUSTIN',
  4  object_name => 'CUSTOMERS',
  5  policy_name => 'customers_vpd1',
  6  policy_function => 'no_customers_num1');
  7  end;
  8  /

PL/SQL procedure successfully completed.

SQL> begin
  2  dbms_rls.add_policy
  3  (object_schema => 'JUSTIN',
  4  object_name => 'PRODUCTS',
  5  policy_name => 'products_vpd1',
  6  policy_function => 'no_product_num5');
  7  end;
  8  /

PL/SQL procedure successfully completed.
```

To test our functions and policies that we have just created, let's execute a select statement that retrieves all of the customers data from the table. Below we see that when we use the select statement to retrieve all information from the table, it does not show the user with the id number of 1 because we have made a function and policy above that makes it so this user cannot see that specific id number.

```
SQL Plus
SQL> select * from customers;

        ID FIRST_NAME           LAST_NAME
---------- -------------------- --------------------
ADDRESS
------------------------------------------------------------
CITY                    ZIPCODE PHONE
-------------------- ---------- --------------------
         2 Jerreak              Purnell
54 Romancoke Drive
Grasonville               21638

         3 Augustus             Gloop
100 Elm Street
Centreville               21617

        ID FIRST_NAME           LAST_NAME
---------- -------------------- --------------------
ADDRESS
------------------------------------------------------------
CITY                    ZIPCODE PHONE
-------------------- ---------- --------------------

         4 Orlando              Hogan
87 Obon
Centreville               21617 4105874525

         5 Julia                Hogan
16 Poplar Drive

        ID FIRST_NAME           LAST_NAME
---------- -------------------- --------------------
ADDRESS
------------------------------------------------------------
CITY                    ZIPCODE PHONE
-------------------- ---------- --------------------
Chester                   21619 4102148965
```

This is only showing the id and first_name columns for a better idea of what is happening in this query. We see id numbers 2 through 5 but our function and policy do not allow us to see the id number 1.

```
SQL> select id, first_name FROM customers;

        ID FIRST_NAME
---------- --------------------
         2 Jerreak
         3 Augustus
         4 Orlando
         5 Julia
```

This select statement shows how the product id 5 is not shown as per the function and policy we have made above. This policy and function says that if there is any id that has a value of 5, then it should not be shown to the user.

```
SQL> select * from products;

PRODUCT_ID NAME                 PRICE
---------- -------------------- --------------------
DESCRIPTION
------------------------------------------------------------
         1 Football             10.99
NFL certified football.

         2 Baseball             8.99
MLB certified baseball.

         3 Mousepad             15.99
SteelSeries XL Mousepad


PRODUCT_ID NAME                 PRICE
---------- -------------------- --------------------
DESCRIPTION
------------------------------------------------------------
         4 Mouse                59.99
Logitech G500s Mouse


SQL>
```

This is not in my script file but I would like to show you how the SYS account is not affected by the function and policy we have created above. We use a select statement to retrieve all of the data that we have just created as the justin account. As you can see, because we are on the SYS account we are able to override the function and policy that we have created above. It shows us all of the entries, including id 1 even though we have a function and policy against this.

```
SQL Plus
Enter user-name: sqlplus / as sysdba
Enter password:

Connected to:
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

SQL> select * from justin.customers;

        ID FIRST_NAME           LAST_NAME
---------- -------------------- --------------------
ADDRESS
------------------------------------------------------------
CITY                    ZIPCODE PHONE
-------------------- ---------- --------------------
         1 Justin               Casteel
1 Main Street
Centreville               21617 4104651457

         2 Jerreak              Purnell
54 Romancoke Drive
Grasonville               21638

        ID FIRST_NAME           LAST_NAME
---------- -------------------- --------------------
ADDRESS
------------------------------------------------------------
CITY                    ZIPCODE PHONE
-------------------- ---------- --------------------

         3 Augustus             Gloop
100 Elm Street
Centreville               21617

         4 Orlando              Hogan
87 Obon

        ID FIRST_NAME           LAST_NAME
---------- -------------------- --------------------
ADDRESS
------------------------------------------------------------
CITY                    ZIPCODE PHONE
-------------------- ---------- --------------------
Centreville               21617 4105874525

         5 Julia                Hogan
16 Poplar Drive
Chester                   21619 4102148965


SQL>
```

Here we do the same example as above but with the products table. As the SYS account we are able to see all of the entries including the ones that the function and policy have been told to not display to the user.

```
SQL Plus
SQL> select * from justin.products;

PRODUCT_ID NAME                 PRICE
---------- -------------------- --------------------
DESCRIPTION
------------------------------------------------------------
         1 Football             10.99
NFL certified football.

         2 Baseball             8.99
MLB certified baseball.

         3 Mousepad             15.99
SteelSeries XL Mousepad


PRODUCT_ID NAME                 PRICE
---------- -------------------- --------------------
DESCRIPTION
------------------------------------------------------------
         4 Mouse                59.99
Logitech G500s Mouse

         5 HyperX Headphones    8.99
HyperX Cloud II Headphones


SQL>
SQL>
```

The SYS account will override any function and policy created and will always be able to see all information unless you tell it to otherwise.

What I mean is you could use a WHERE clause in your statement such as:

select * from justin.products where product_id != 3;

Which would return the following output:

SQL Plus

```
SQL> select * from justin.products where product_id != 3;

PRODUCT_ID NAME                 PRICE
---------- -------------------- --------------------
DESCRIPTION
------------------------------------------------------------
         1 Football             10.99
NFL certified football.

         2 Baseball             8.99
MLB certified baseball.

         4 Mouse                59.99
Logitech G500s Mouse


PRODUCT_ID NAME                 PRICE
---------- -------------------- --------------------
DESCRIPTION
------------------------------------------------------------
         5 HyperX Headphones    8.99
HyperX Cloud II Headphones


SQL>
```