Workload testing and analysis using Swingbench with ATP :

**Objectives**

• Learn how to install Swingbench

• Using Swingbench to demo and analyse workloads with ATP

In this section you will use Swingbench, a suite of utilities developed by Dominic Giles that creates Oracle Database, objects, data, and creates workloads that can then be monitored or used for testing an ATP environment. Among the many uses for Swingbench is the ability to provided and comprehensive demo of the database and associated tools and functionality.

The objective of this lab it to learn how to install Swingbench and connect and use it with ATP, not a comprehensive Swingbench tutorial. At the end of this lab you can find links to much more information on Swingbench and its use.

1. **Installing Swingbench pre-reqs:**

To run Swingbench requires 3 components:

1. Unix BASH shell
2. Java JRE 1.8xx
3. Swingbench (and optional modules)

**Installing Unix BASH shell:**

To run Swingbench on Windows you need to simulate running a Unix shell and that can be done in many ways. This lab uses the Git BASH shell. The latest version of Windows 10 includes this functionality natively but you many need to enable/install it. The steps for getting BASH running will not be covered in this lab but for Windows 10 you can find detailed instructions here:

[**https://www.windowscentral.com/how-install-bash-shell-command-line-windows-10**](https://www.windowscentral.com/how-install-bash-shell-command-line-windows-10)

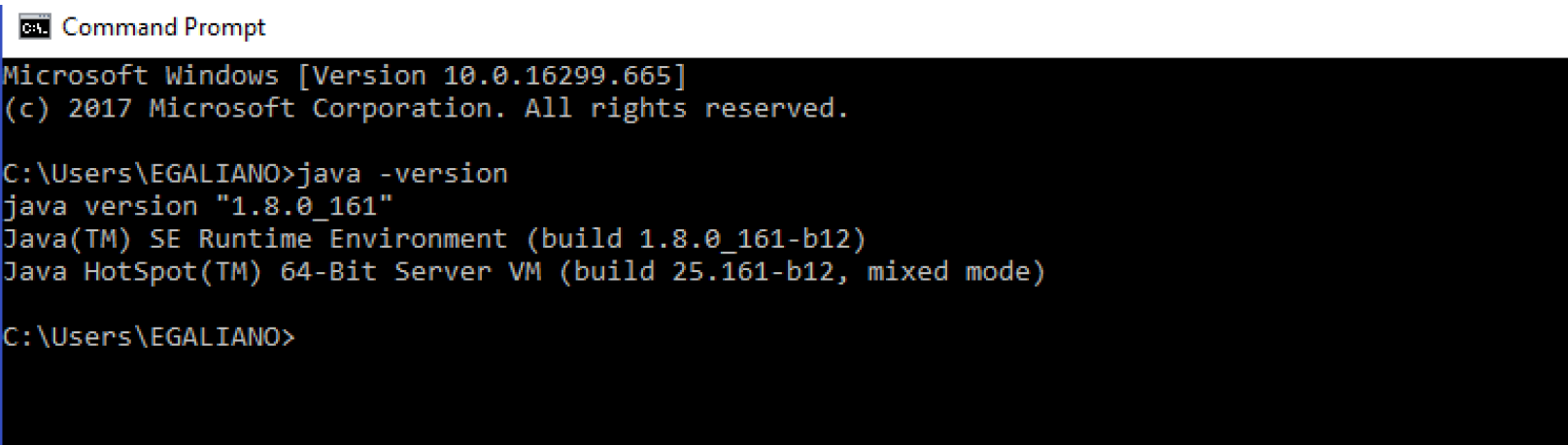
For older versions of Windows you can download and install BASH <https://gitforwindows.org/>

**Installing Java JRE 1.8xx:**

Swingbench requires Java 8 to run. You must install it if you don’t have it installed. Please notice if you already have SQL Developer running then you have the correct Java installed already. You can test to see if you have it installed by running the following command in a Command Prompt window:

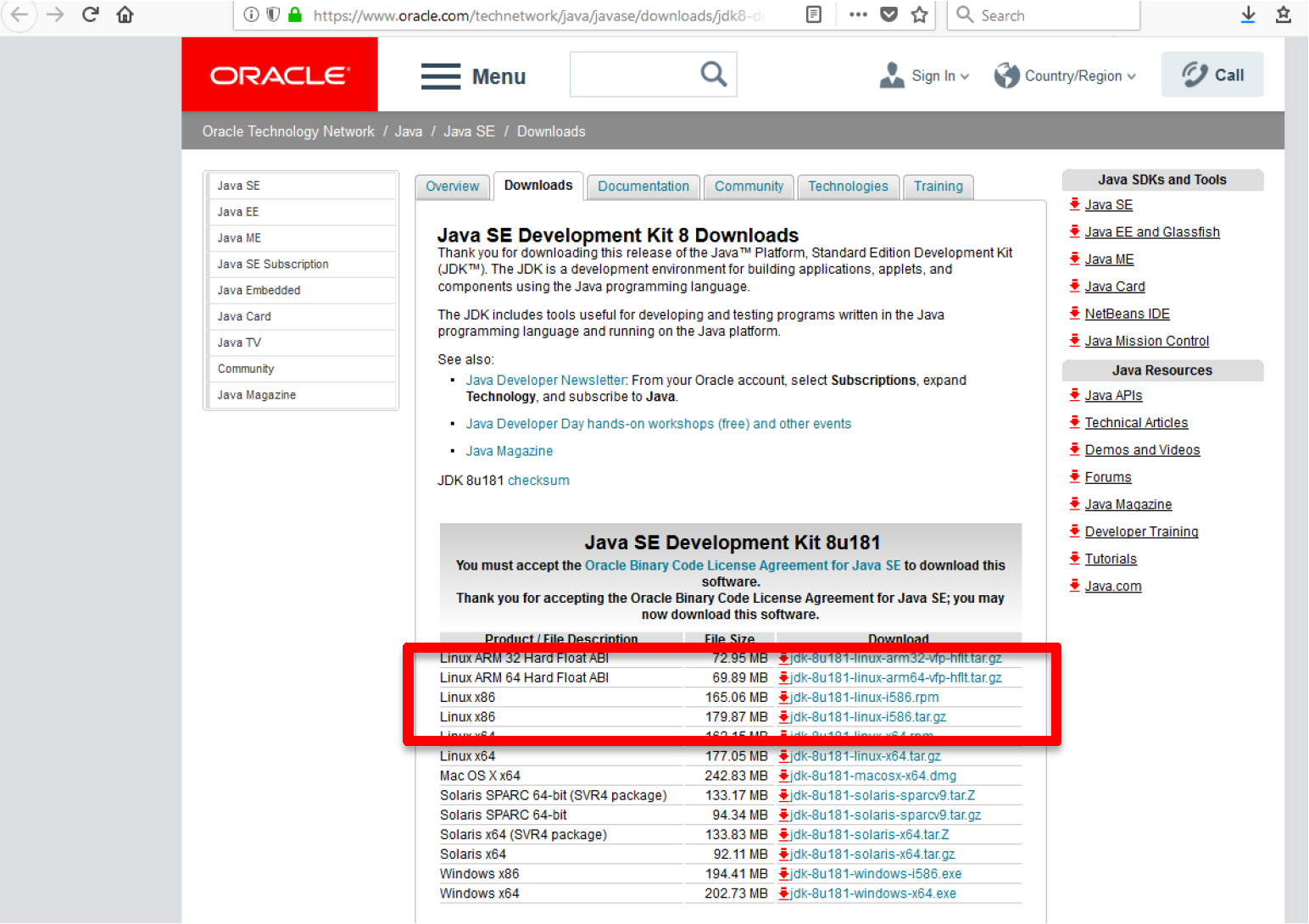
java –version

and the version should be 1.8.0\_xx

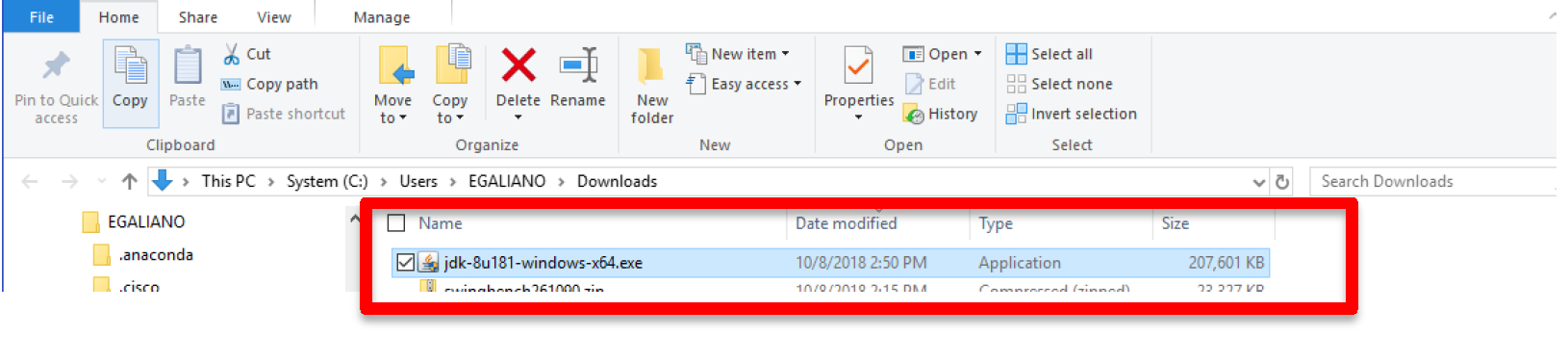


Java can be downloaded and installed from (pick the appropriate operating system and architecture):

<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>



After downloading the file, proceed to the download folder and double click to install:



Click Next:



Select Public JRE

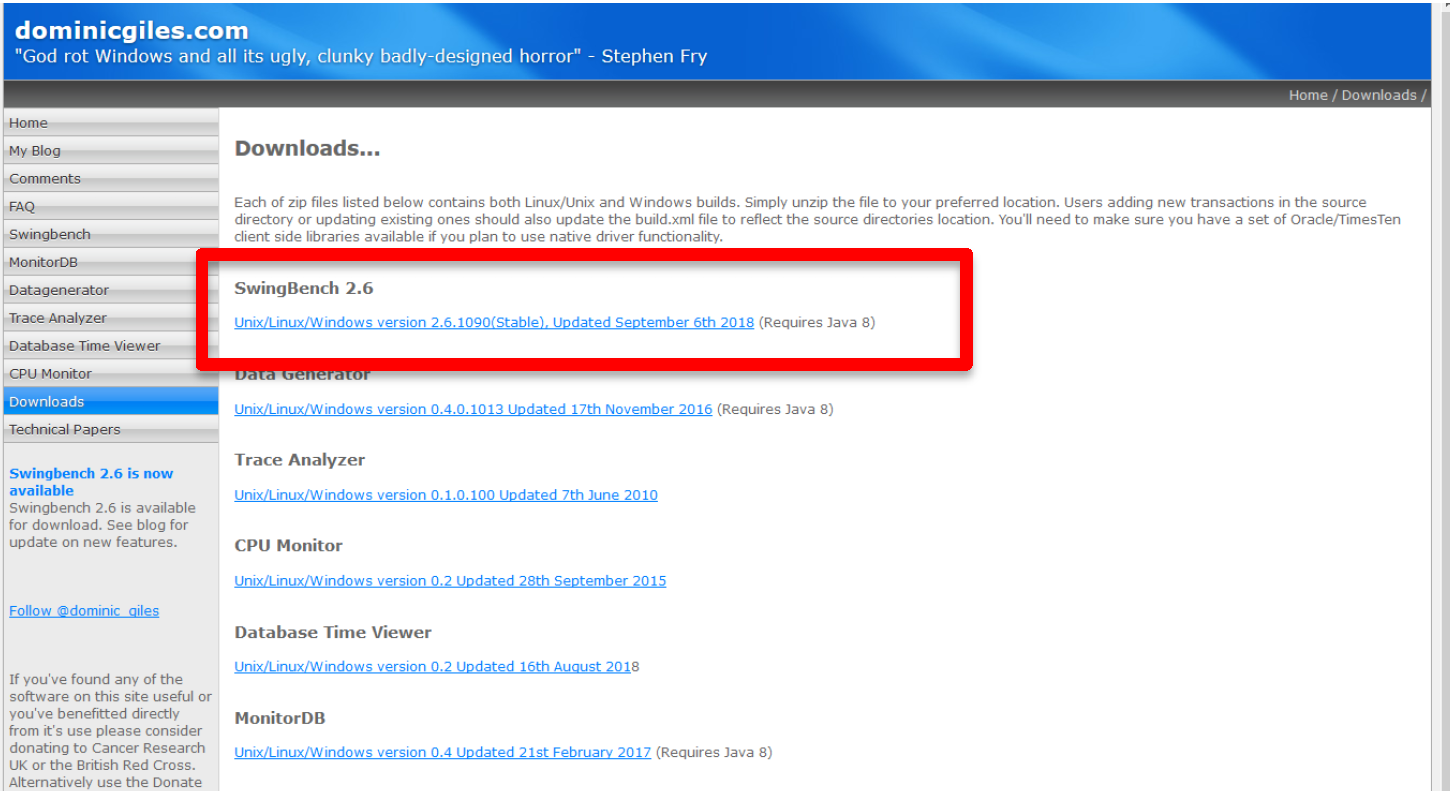


Finish the Java install process

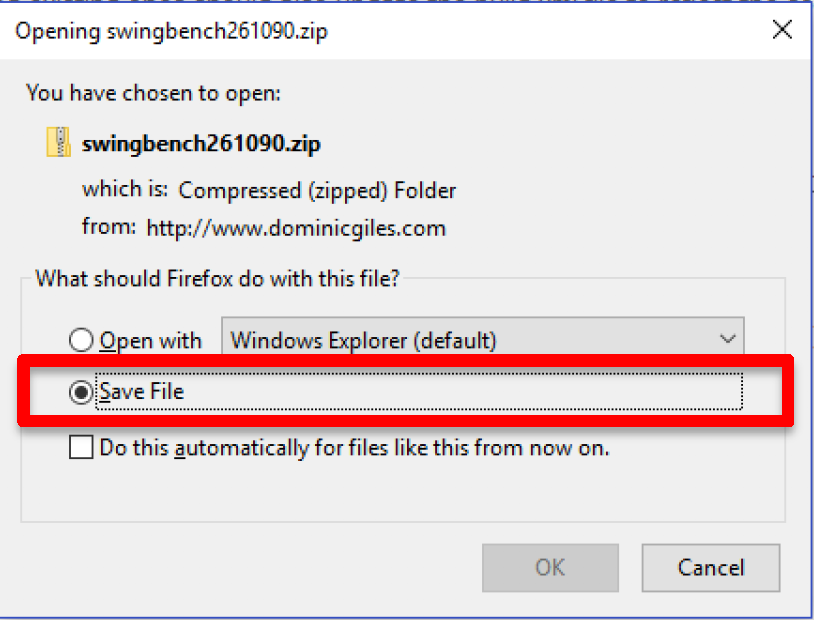
1. **Installing and running Swingbench:**

Download Swingbench from:

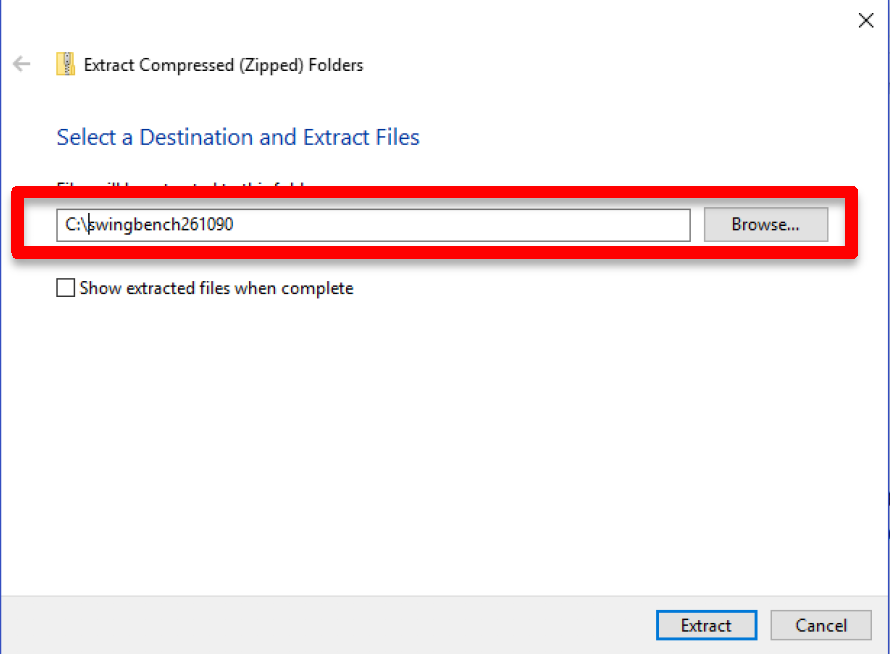
<http://www.dominicgiles.com/downloads.html>



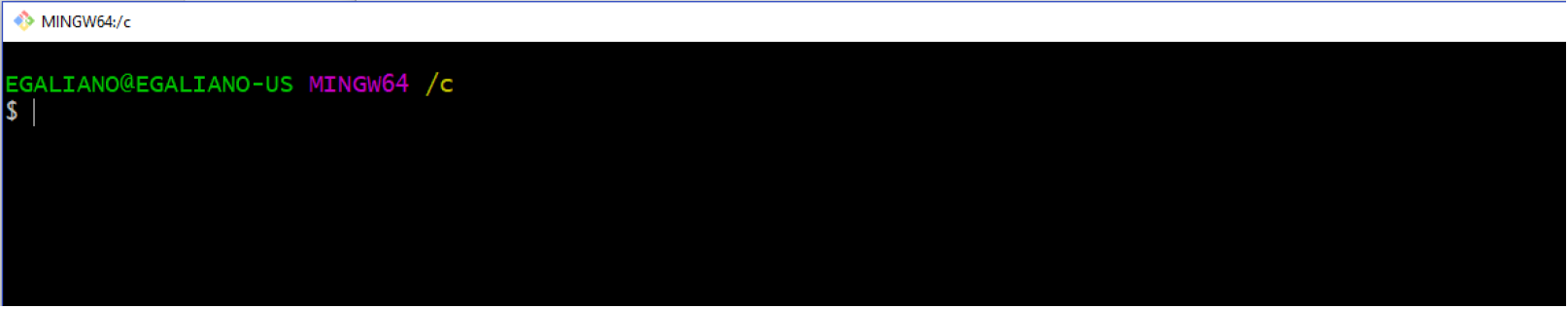
Save the file:



After the file downloads, go to the folder where it downloaded, and right click on the file, select Extract all (uncompress) to the following directory: **C:\swingbench26,** select **Extract**



Open up a BASH window (in Search Windows type BASH and select)



Change to the “bin” subdirectory where swingbench is installed. In this example: /c/swingbench261090/swingbench/bin

And start swingbench by issuing the command below (see screenshot and explanation of parameters below) (make sure to use your wallet and ATP database user/password information, fields that need to be modified to your values are in black). Notice the command below will create a database user name “soe” (-u soe with a password “Atpxweek2018” (-p Atpxweek2018) which will be used by Swingbench to load data and run workloads.

./oewizard -cf /c/wallets/wallet\_atpxweek.zip \

-cs atpxweek\_high \

-ts DATA \

-dbap Atpxweek2018 \

-dba admin \

-u soe \

-p Atpxweek2018 \

-async\_off \

-scale 1 \

-hashpart \

-create \

-cl \

-v \

-debug

Understanding the parameters above

**-cf** speficies the location of the wallet file

**-cs** specifies the ATP service to connect to

**-ts** name of the table space to install swingbench into. For ATP it is always DATA

**-dba** user **admin** created during ATP instance creation

**-dbap admin** user password

**-u** new database user created for swingbench data

**-p** new user (above) password

**-async\_off** async commits are not supported in ATP

**-scale** in GB’s, data size, plus additional 50% of data for indexes

**-debug** will display back each step of the process – leave this out if too much data is on your screen.



The creation process will take a while. Once complete verify the tables created correctly by running the following command (from the swingbench bin directory, make sure to use your wallet, change values in black):

./sbutil -soe \

-cf /c/wallets/wallet\_atpxweek.zip \

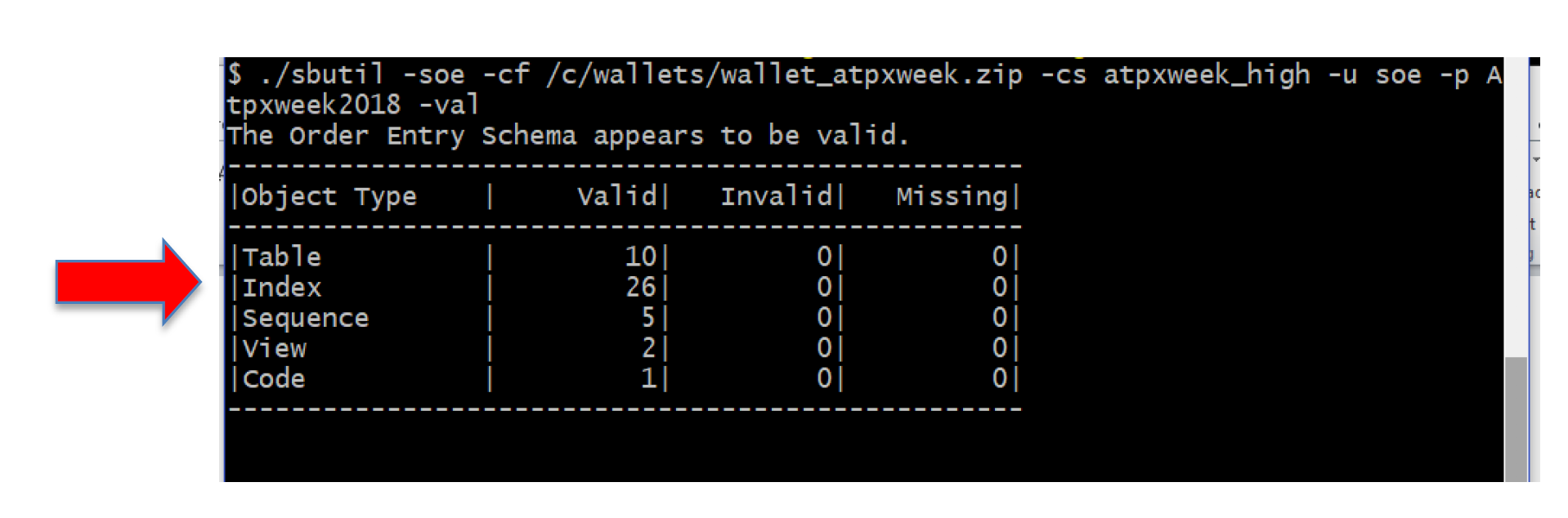
-cs atpxweek\_high \

-u soe \

-p Atpxweek2018 \

-val

Results should be similar to the screenshot below:



To see how many rows were inserted on each table run the following command (make sure to use your wallet information):

./sbutil -soe \

-cf /c/wallets/wallet\_atpxweek.zip \

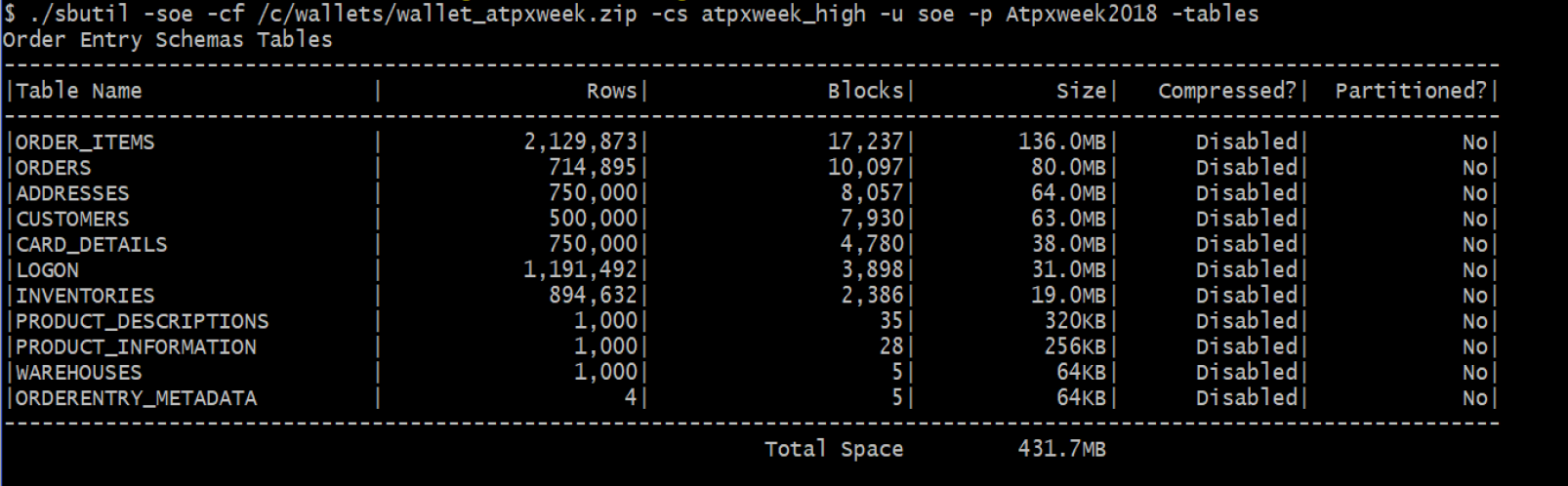
-cs atpxweek\_high \

-u soe \

-p Atpxweek2018 \

-tables

Results should be similar to the screenshot below:



To collect statistics on the tables that were created run the command below. If you will be conducting performance testing with Swingbench it is recommended that you collect statistics (make sure to use your wallet information):

./sbutil -soe \

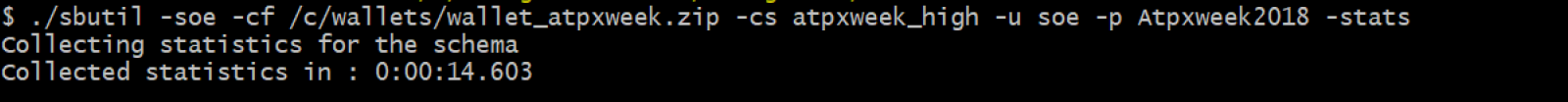
-cf /c/wallets/wallet\_atpxweek.zip \

-cs atpxweek\_high \

-u soe \

-p Atpxweek2018 \

-stats



You are ready to run Swingbench workloads on ATP. Workloads are simulated by users submitting transactions to the database. To do this, the user process must be configured. Run the following command unchanged from the same bin directory you have been running the other commands:

sed -i –e 's/<LogonGroupCount>1<\/LogonGroupCount>/<LogonGroupCount>5<\/LogonGroupCount>/' \

-e 's/<LogonDelay>0<\/LogonDelay>/<LogonDelay>300<\/LogonDelay>/' \

-e 's/<WaitTillAllLogon>true<\/WaitTillAllLogon>/<WaitTillAllLogon>false<\/WaitTillAllLogon>/' \

../configs/SOE\_Server\_Side\_V2.xml

Generate load on your database by running the charbench utility. (make sure to use your wallet information). Use the command below. There are 2 parameters you can change to modify the amount of load and users being generated. The –uc specifies the number of users that will be ramped up, in the case below 64. The –rt specificies the total running time, in the example below 30 seconds. To run continuously remove that parameter, or change the time to specify a maximum run time. Please note: charbench will place a high load on the database, so don’t run for long periods of time. You can stop running charbench at any time with Ctrl C.

./charbench -c ../configs/SOE\_Server\_Side\_V2.xml \

-cf /c/wallets/wallet\_atpxweek.zip \

-cs atpxweek\_high \

-u soe \

-p Atpxweek2018 \

-v users,tps,vresp \

-intermin 0 \

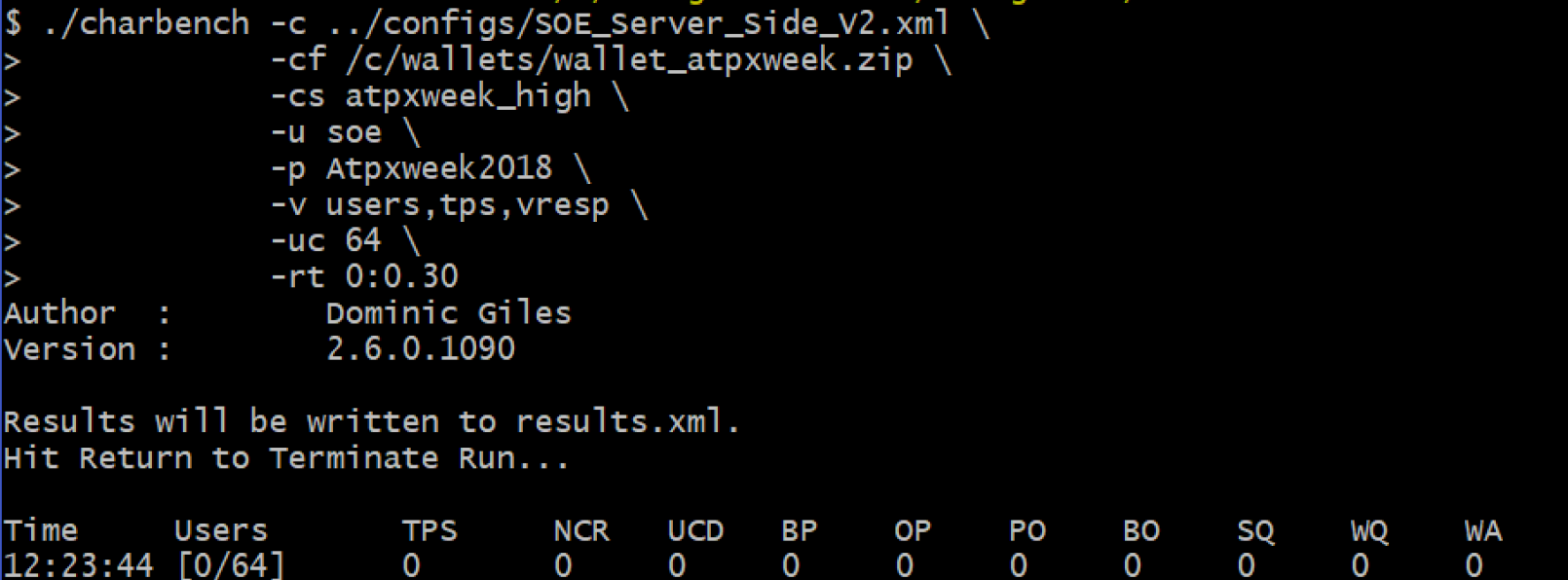
-intermax 0 \

-min 0 \

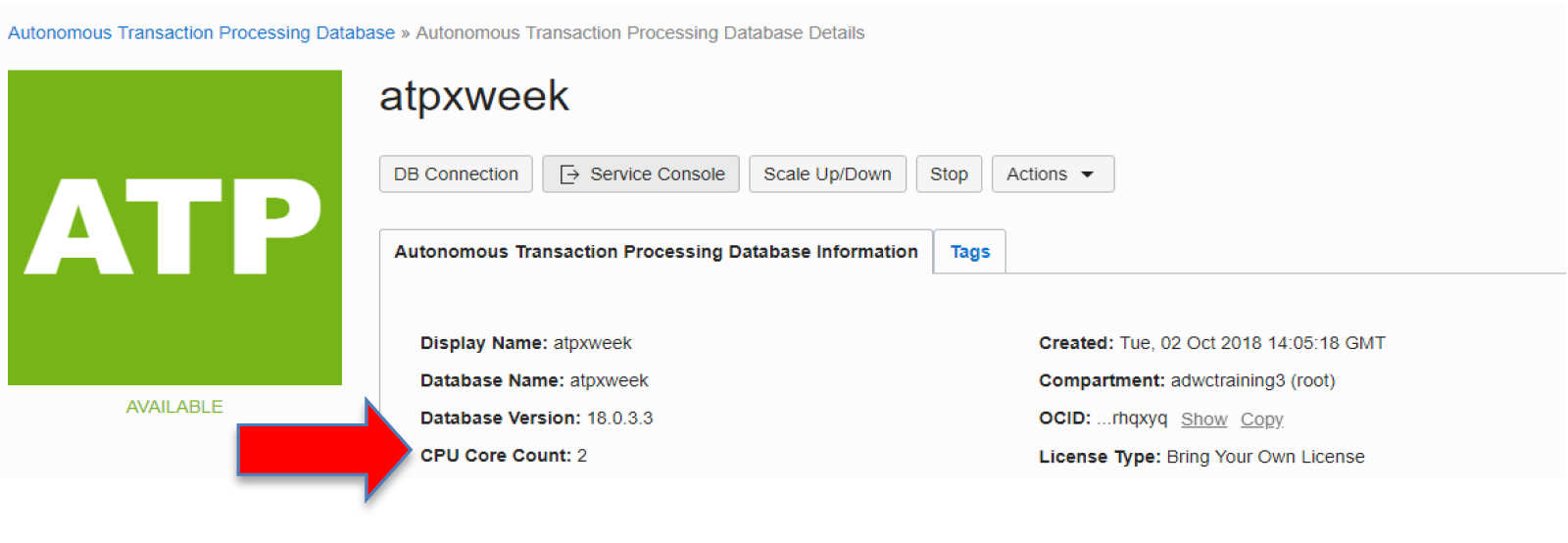
-max 0 \

-uc 64 \

Your output will look similar to the screenshot below. The columns indicate the wall clock time, the number of users connected, the transactions per second (TPS), and number of transactions per type on the following columns. There are many other parameters that can be included to show more information.



To show how scaling ATP is an online operation and how workloads are immediately affected by scaling the environment run the workload generator in continuous mode and scale the system up and down while it is running. We will monitor the TPS column to see how the transactions per second varies depending on the number of CPU’s allocated to the environment. Remember that in this case all workload is being generated through the **\_high** service which will use all resources available. Other services may display different results. Experiment with different services by changing the **–cs parameter** when launching the load generator. In this example we start with 2 CPU’s, will scale up to 4 CPU’s and then scale down to 1 CPU. For a refresher on scaling refer to Lab 4.



In your service console verify that you are starting with 2 CPU’s

Launch the workload generator with the following command on your BASH window (make sure to use your wallet information):

./charbench -c ../configs/SOE\_Server\_Side\_V2.xml \

-cf /c/wallets/wallet\_atpxweek.zip \

-cs atpxweek\_high \

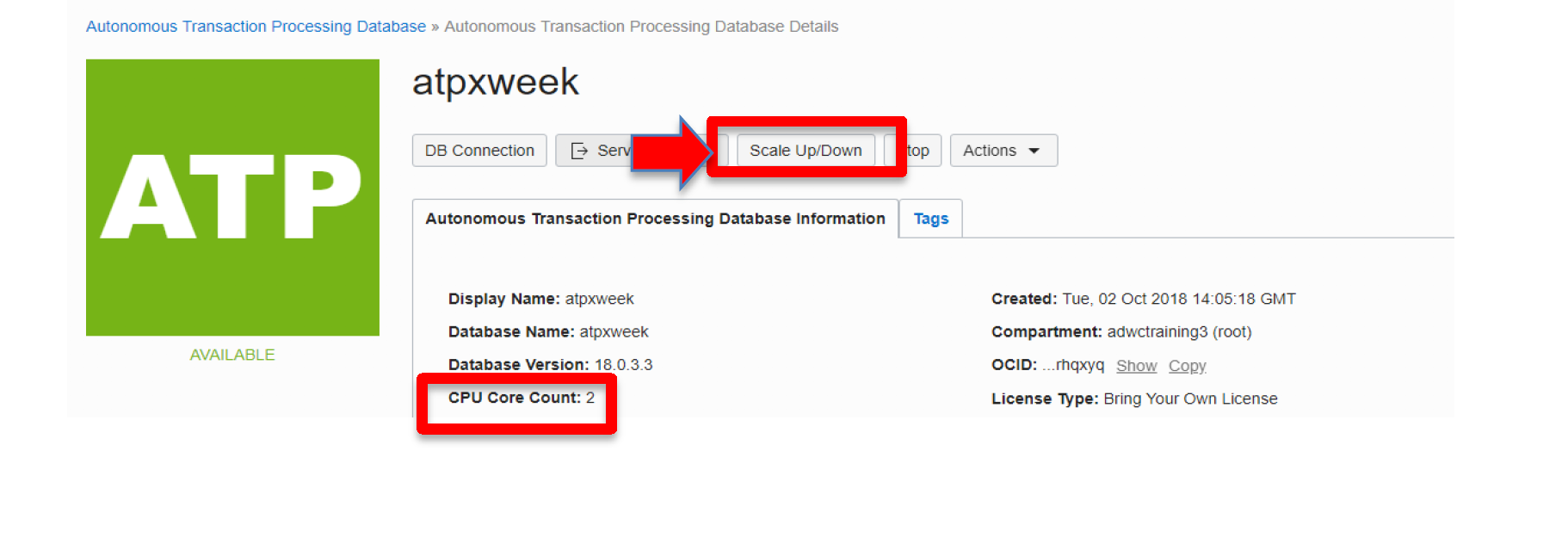
-u soe \

-p Atpxweek2018 \

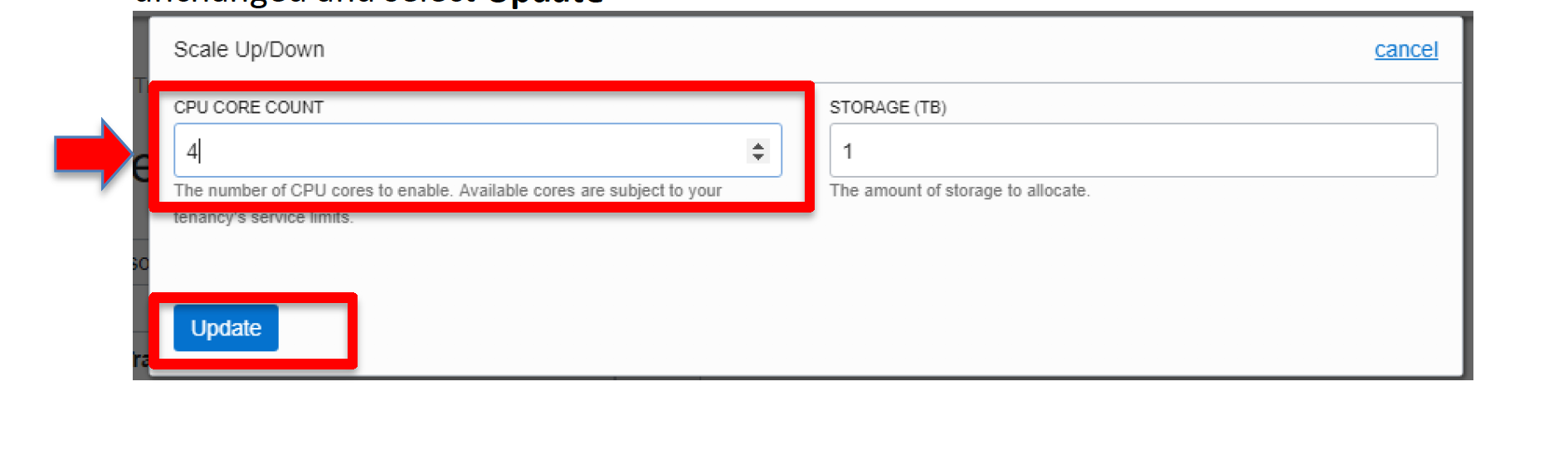
-v users,tps,vresp \

-uc 64

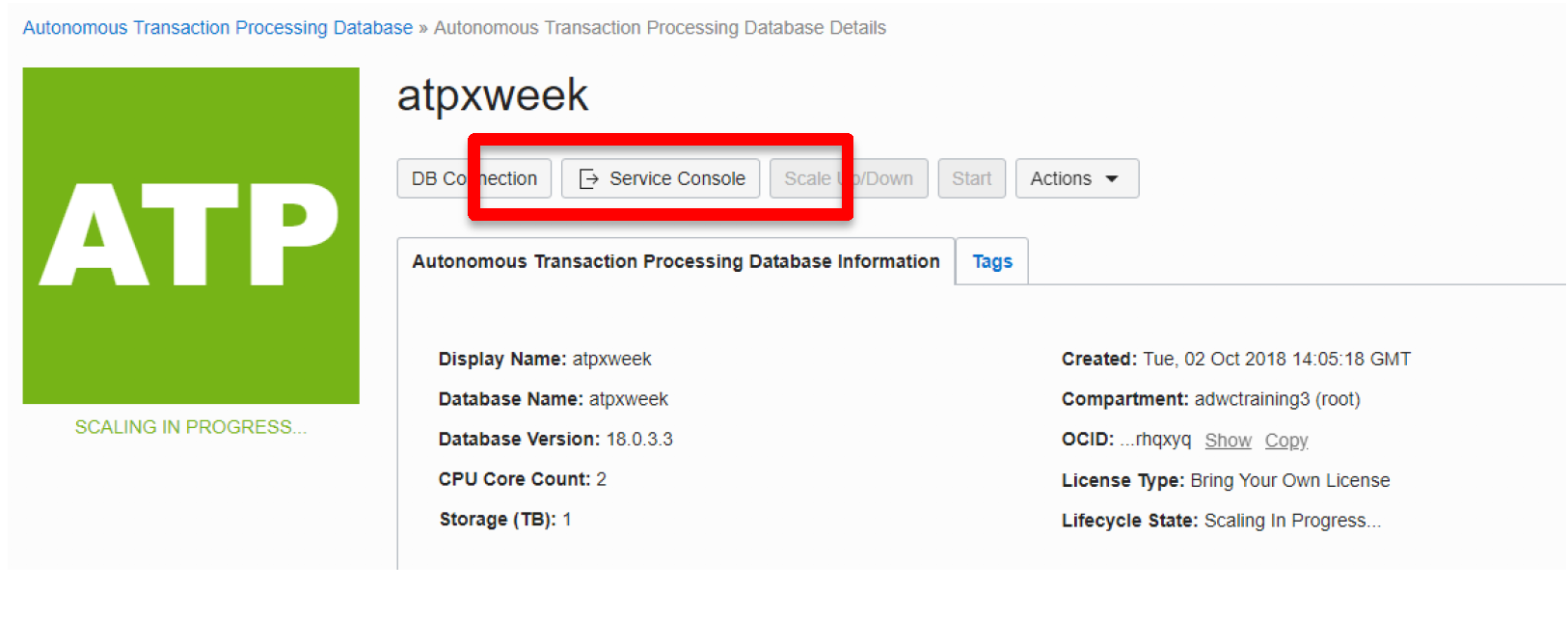
The output will start being generated in the BASH window. While the workload generator is running go to the service console and scale your environment from 2 to 4 CPU’s



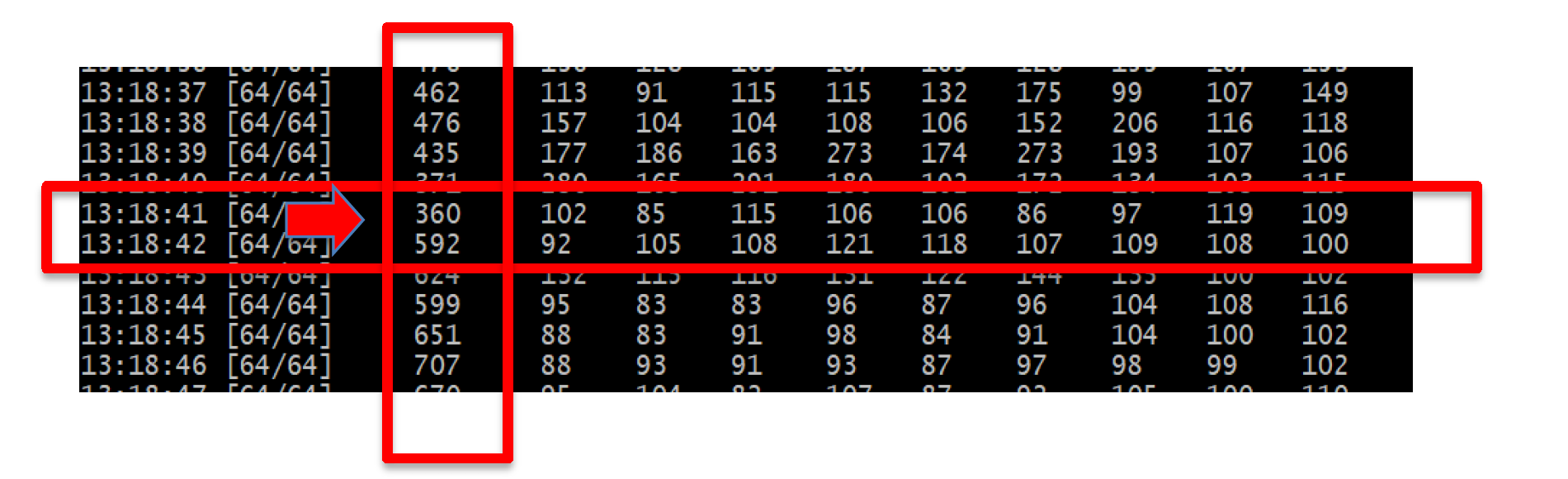
In the Scale Up/Down screen, enter 4 in the CPU CORE COUNT, leave STORAGE unchanged and select **Update**

****

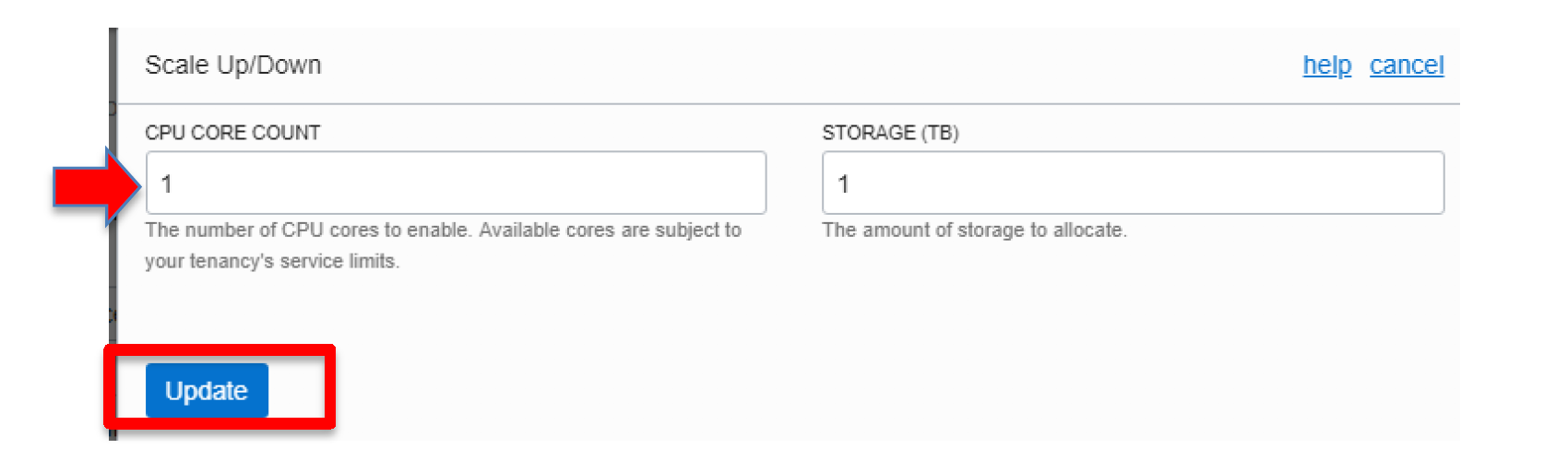
The Service Console will show scaling

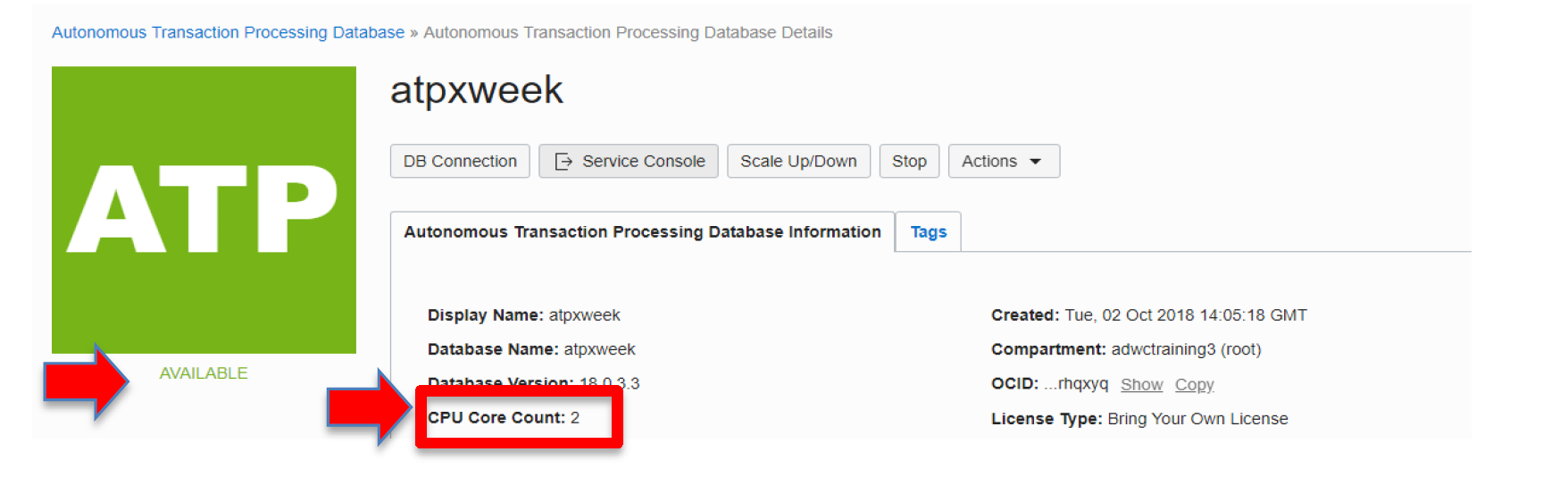


While the scaling is occurring go back to your BASH window and look at the TPS (transactions per second) during the scaling process that number will **increase** when the new CPU’s are allocated automatically. Notice the processing never ends. It will be very evident when the additional CPU’s are enabled.



Now repeat the process and scale the environment down to 1 CPU

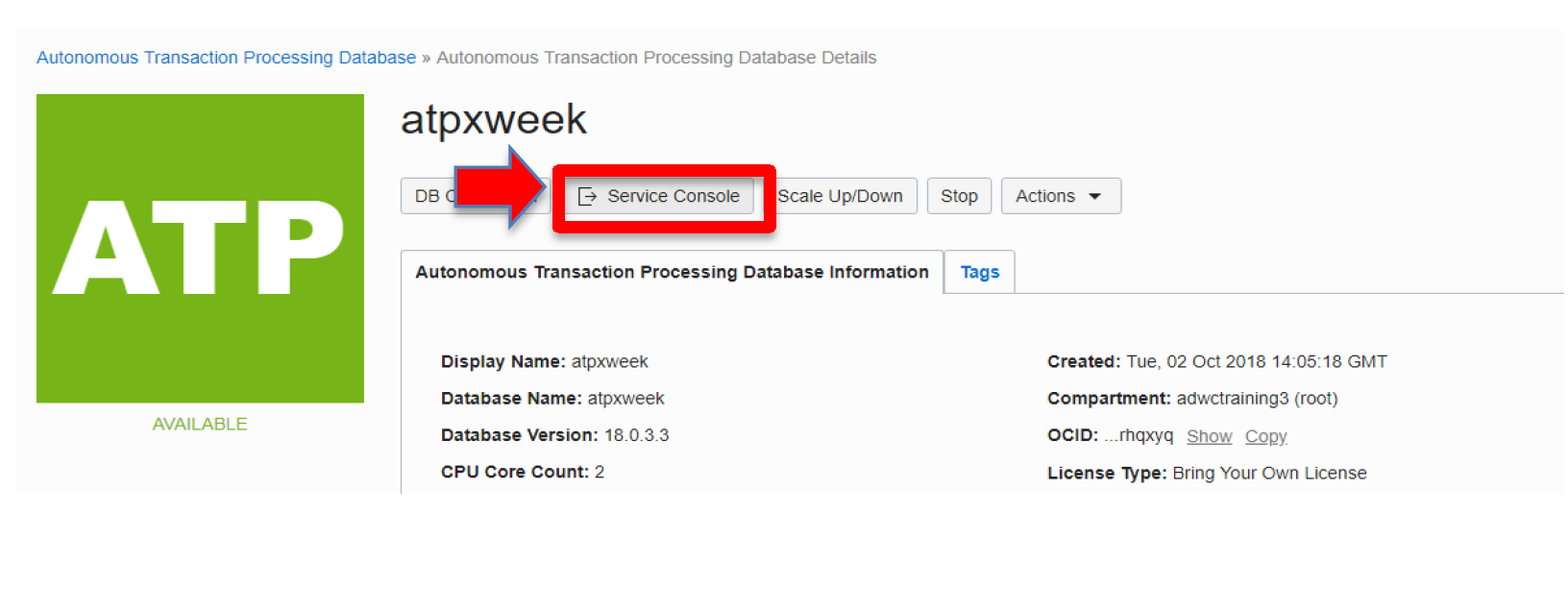


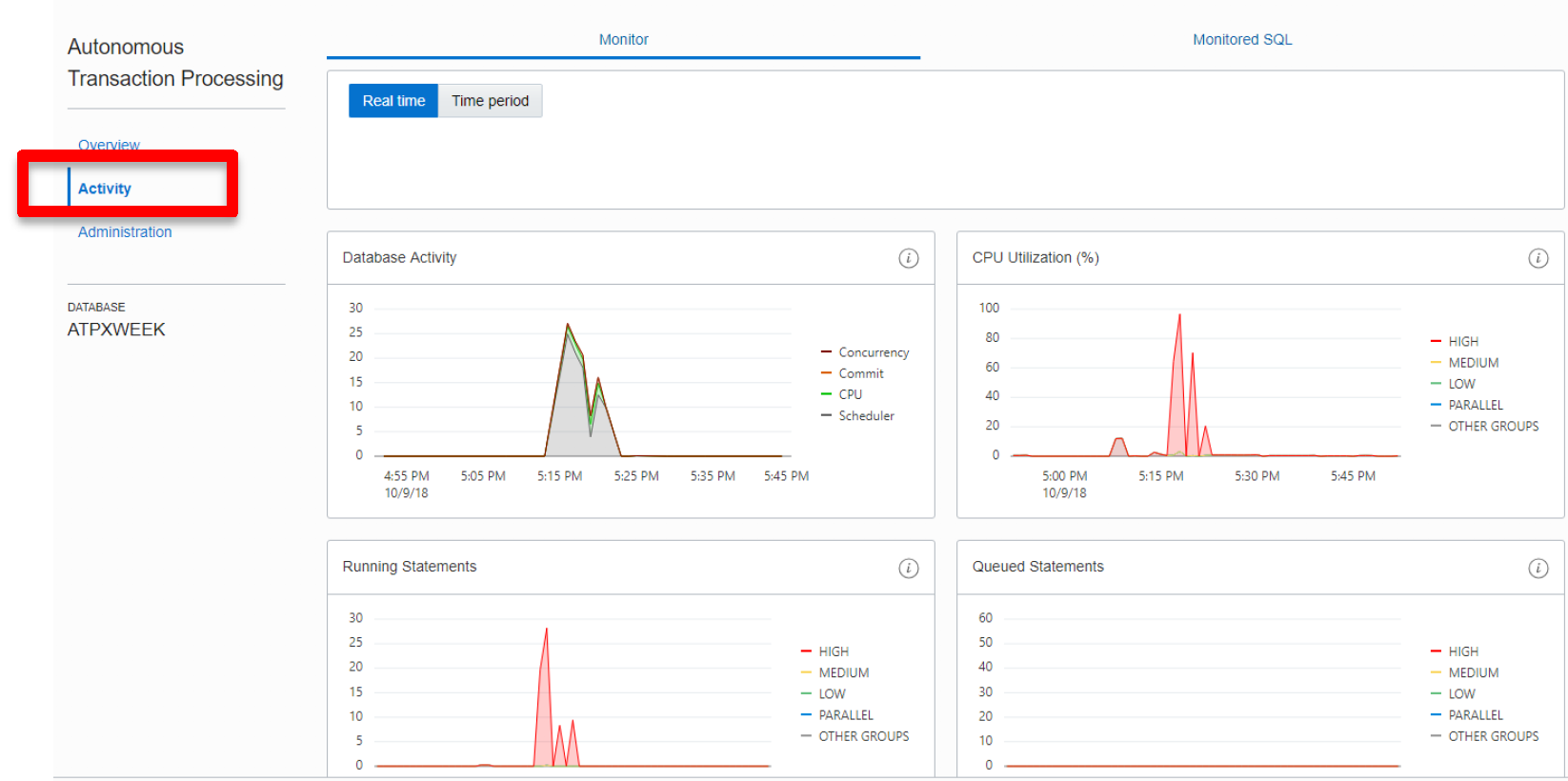


Again, while the scaling is occurring go back to your BASH window and look at the TPS (transactions per second) during the scaling process that number will **decrease** when the extra CPU’s are deallocated. Notice the processing never ends. It will be very evident when the additional CPU’s are disabled.



Also examine the database workload through the Service Console (refer to previous lab for more details how to do this)





**When you are done with this lab, it is very important that you stop the workload generator by hitting Ctrl-C on your BASH Window. Also stop your ATP instance or scale down to 1 CPU.**

**UNIFIED AUDITING**

Unified Auditing is enabled by default. You can audit the several types of activities, using unified audit policies and the AUDIT SQL statement.

All audit records are written to the unified audit trail in a uniform format and are made available through the UNIFIED\_AUDIT\_TRAIL view.

The unified audit trail captures audit information from many types of object, from SQL statements to other Oracle Database Components, such as Data Pump, SQL\*Loader etc.

**Benefits of the Unified Audit Trail**

* The audit records are placed in one location and in one format, rather than your having to look in different places to find audit trails in varying formats.
* This consolidated view enables auditors to co-relate audit information from different components.
* The management and security of the audit trail is also improved by having it in single audit trail.
* Overall auditing performance is greatly improved. By default, the audit records are automatically written to an internal relational table
* You can create named audit policies that enable you to audit the supported components. Furthermore, you can build conditions and exclusions into your policies.

The objective of this lab is to create an audit policy for the update done on ORDERS table and then to query UNIFIED\_AUDIT\_TRAIL to view the generated audit records.

We will do the following:

* Create an audit policy
* Enable the policy and apply audit settings to one or more users.
* View the generated audit records.

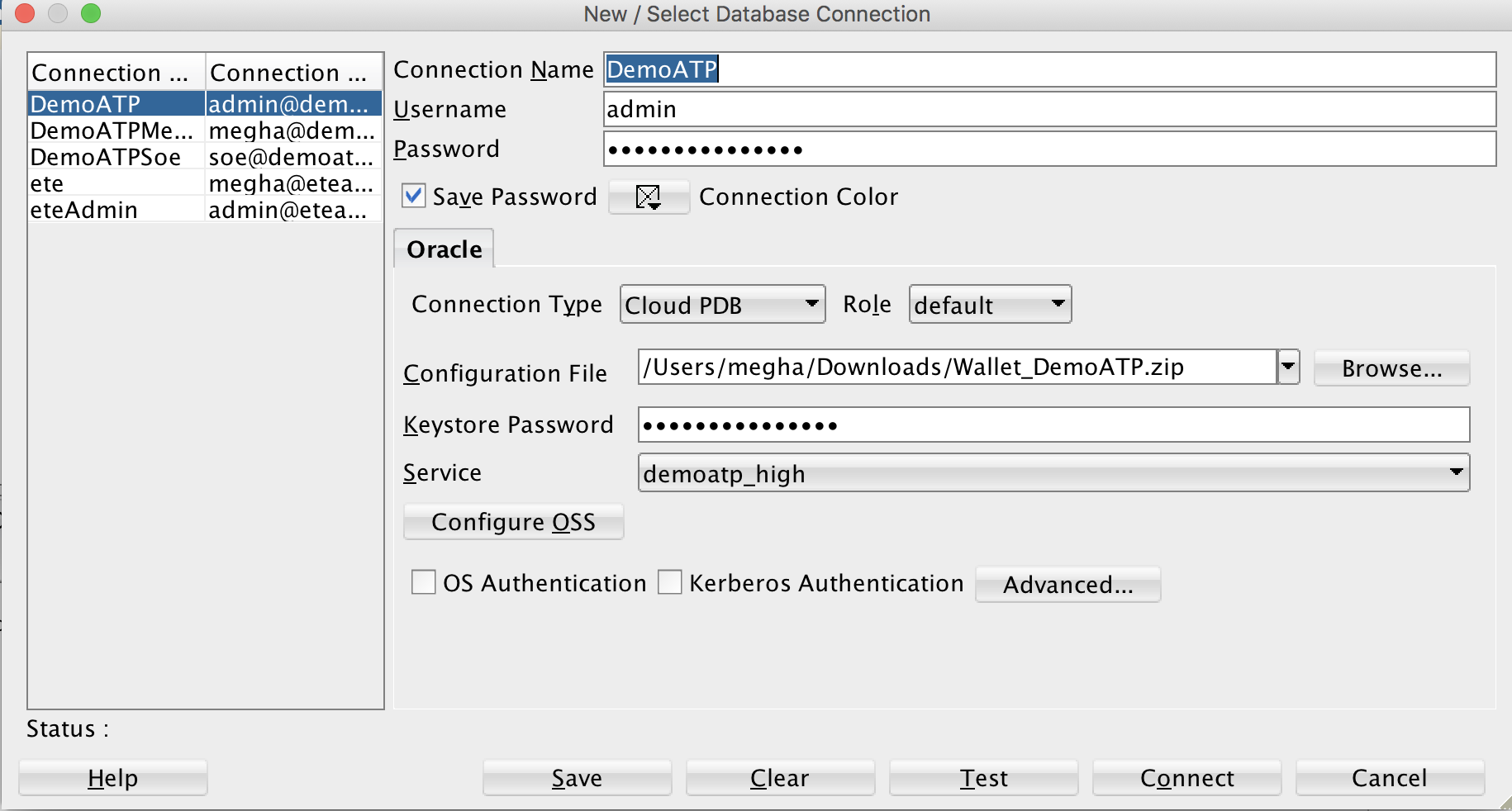
**PRE-REQUISITES**

**You need to have a connection to database through admin.**

Open up your SQL Developer and create a new connection for admin. If you already have a connection, skip this step.

Enter the following details for admin:

1. Connection Name: **DemoATP**
2. Username: **admin**
3. Password: **Password you entered while creating database on cloud.**
4. Connection Type: **Cloud PDB**
5. Configuration File: **Path to your wallet**
6. Keystore Password: **Password entered while downloading wallet.**



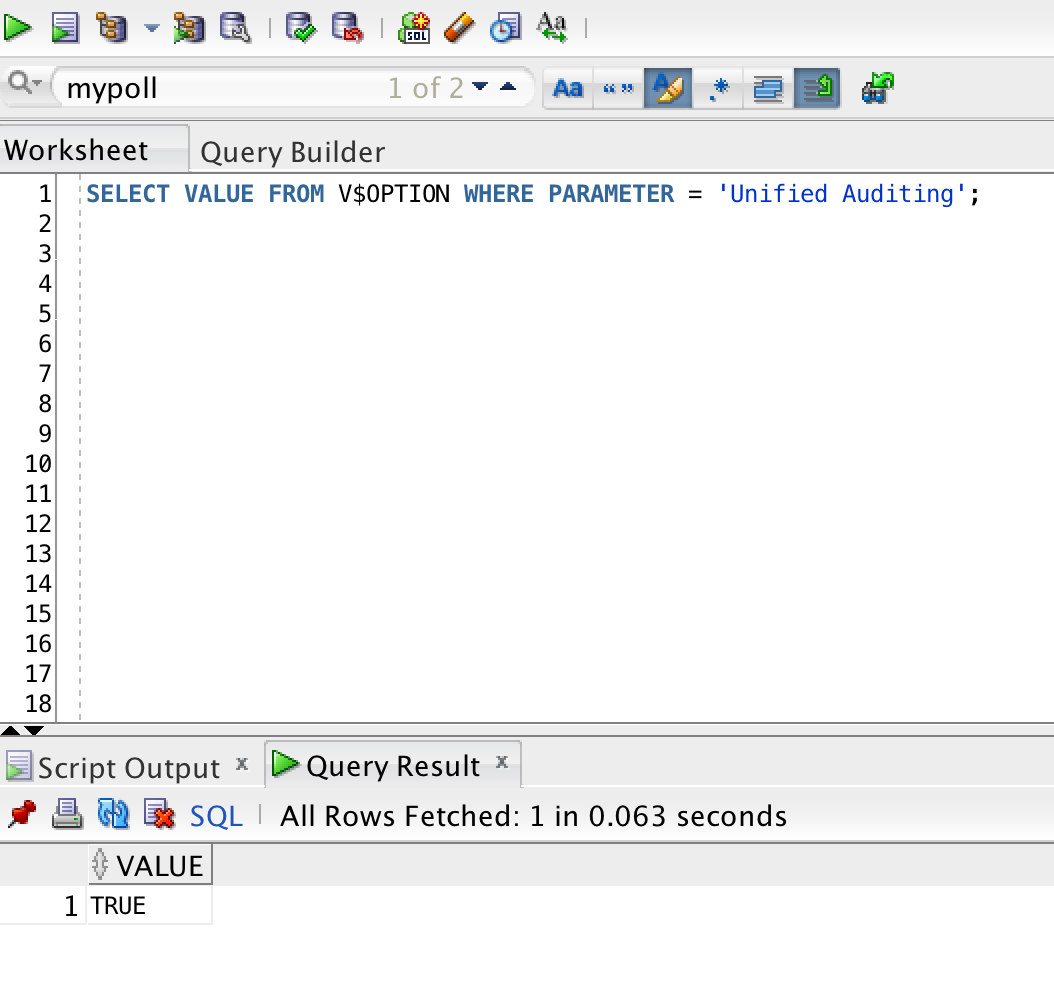
Click on Test, if it shows success, click on save and then click on connect.

**CREATE AN AUDIT POLICY**

After you are successfully connected to the database as admin, open up a SQL worksheet and type the following SQL statement:

**SELECT VALUE FROM V$OPTION WHERE PARAMETER = 'Unified Auditing';**

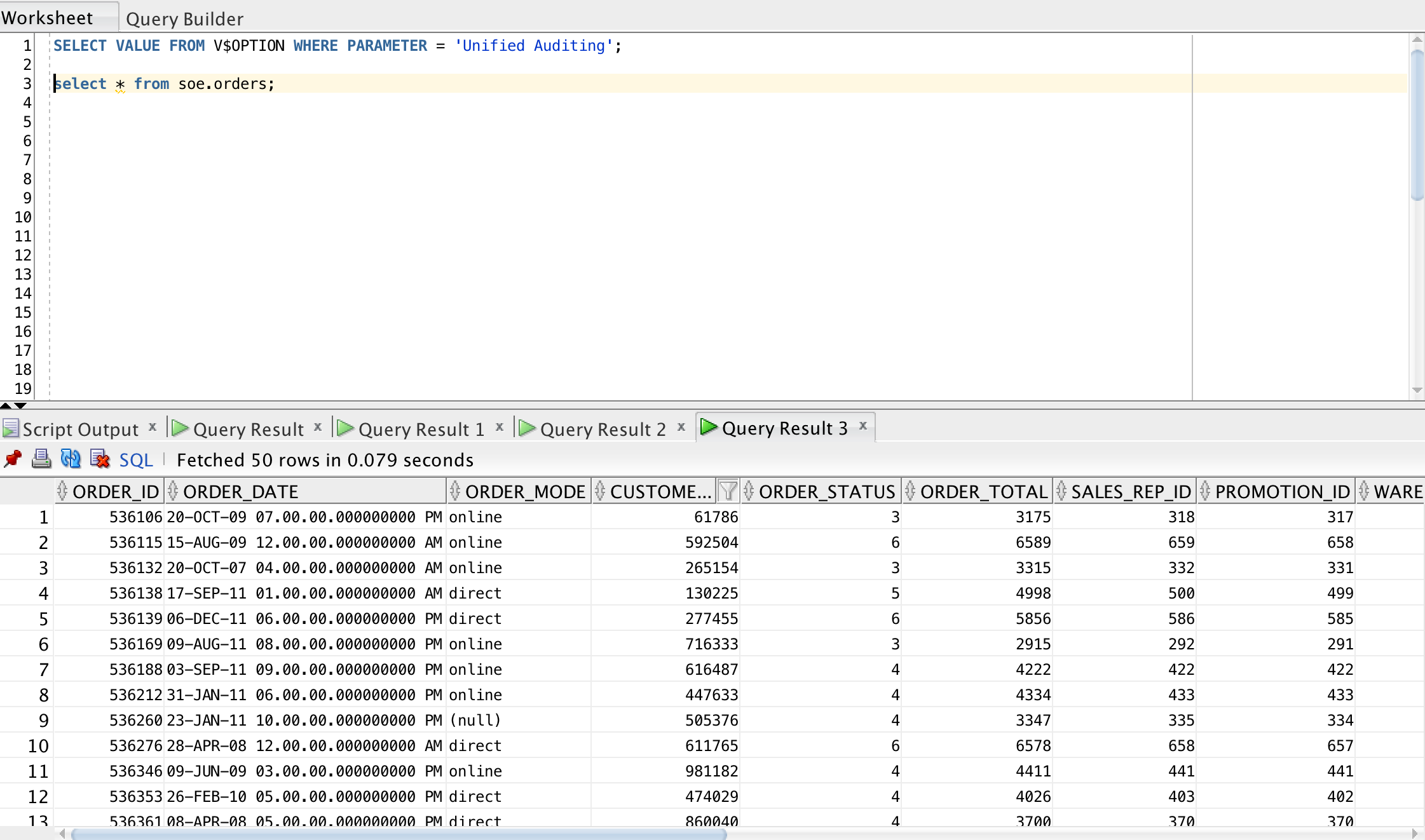
The result should be TRUE. This shows that the unified auditing is enabled by default in your database.



Lets query the ORDER table that we have created through swingbench for user soe.

Enter the following SQL sentence in SQL Worksheet:

**select \* from soe.orders;**



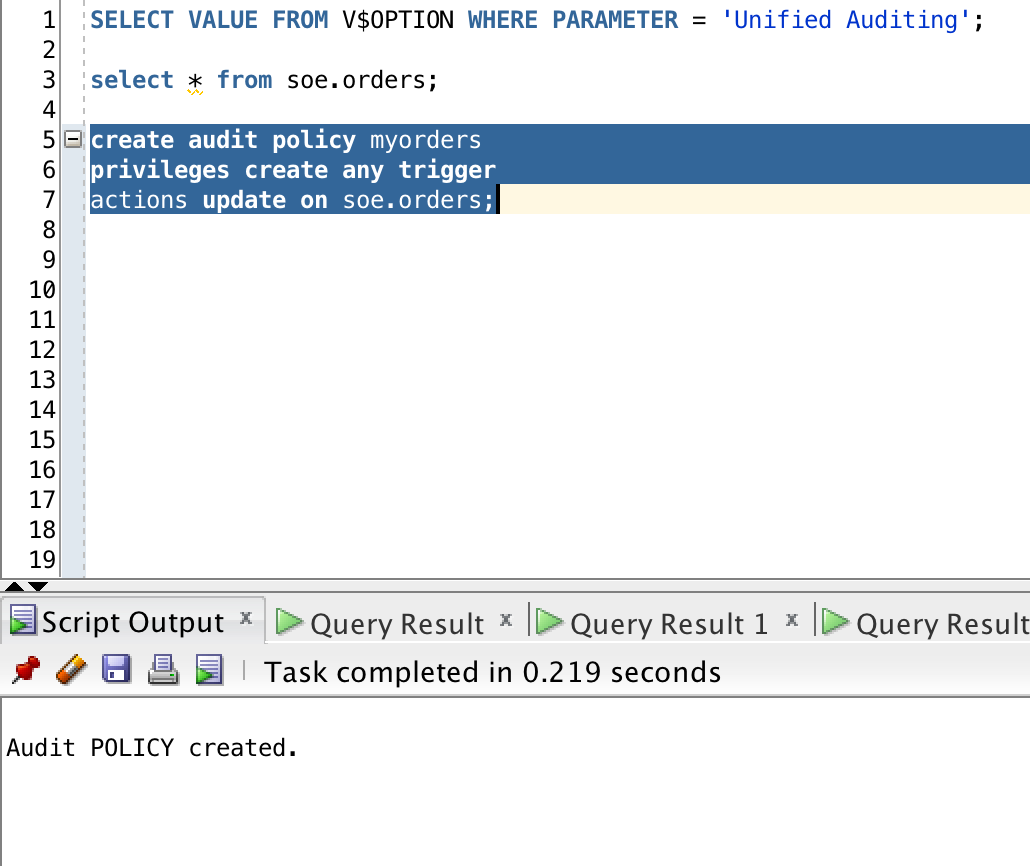
After verifying the data, lets create the audit policy named MYORDERS. This policy will audit the UPDATE activity on ORDERS table.

Copy and paste the following SQL sentence on SQL worksheet

**create audit policy myorder**

**privileges create any trigger**

**actions update on soe.orders;**

****

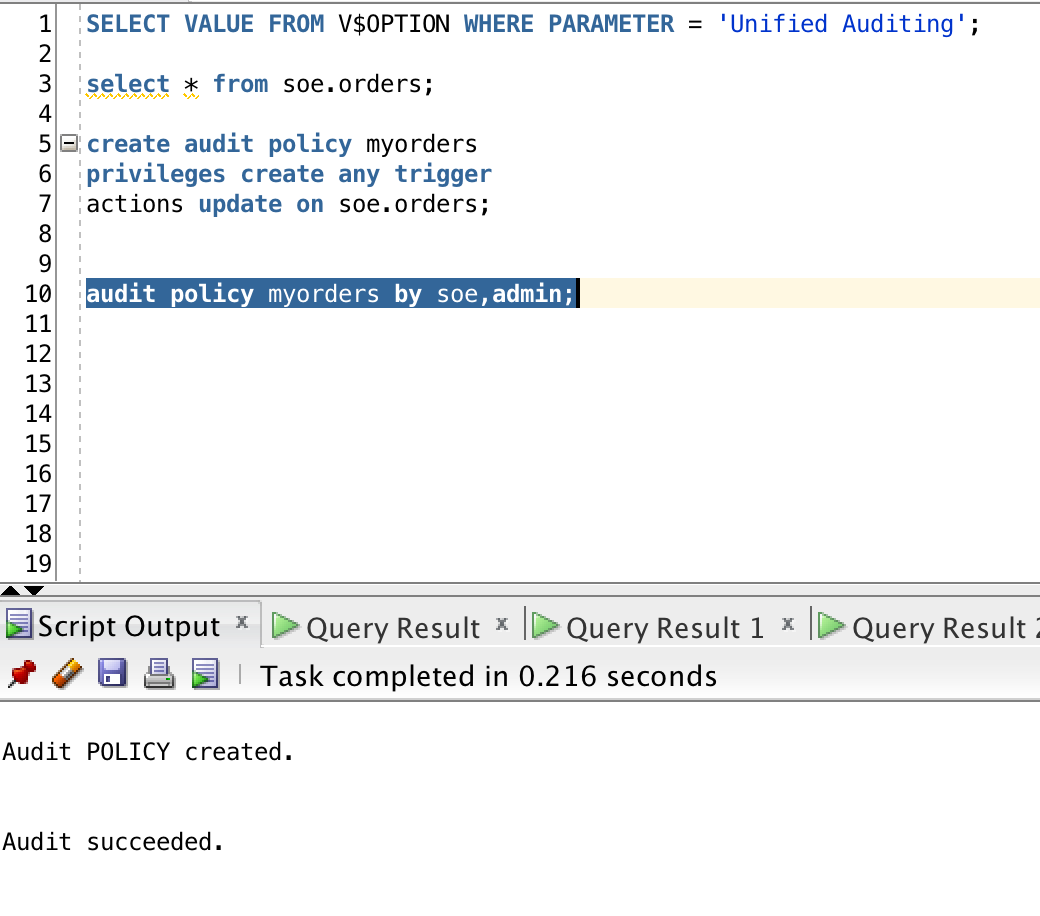
The output should be “Audit POLICY created”.

**Enable the policy and apply audit settings to one or more users.**

The AUDIT statement with the POLICY clause enables a unified audit policy. Use the BY clause to apply the policy to one or more users

Copy paste the following to your SQL Workbench:

**audit policy myorders by soe,admin;**

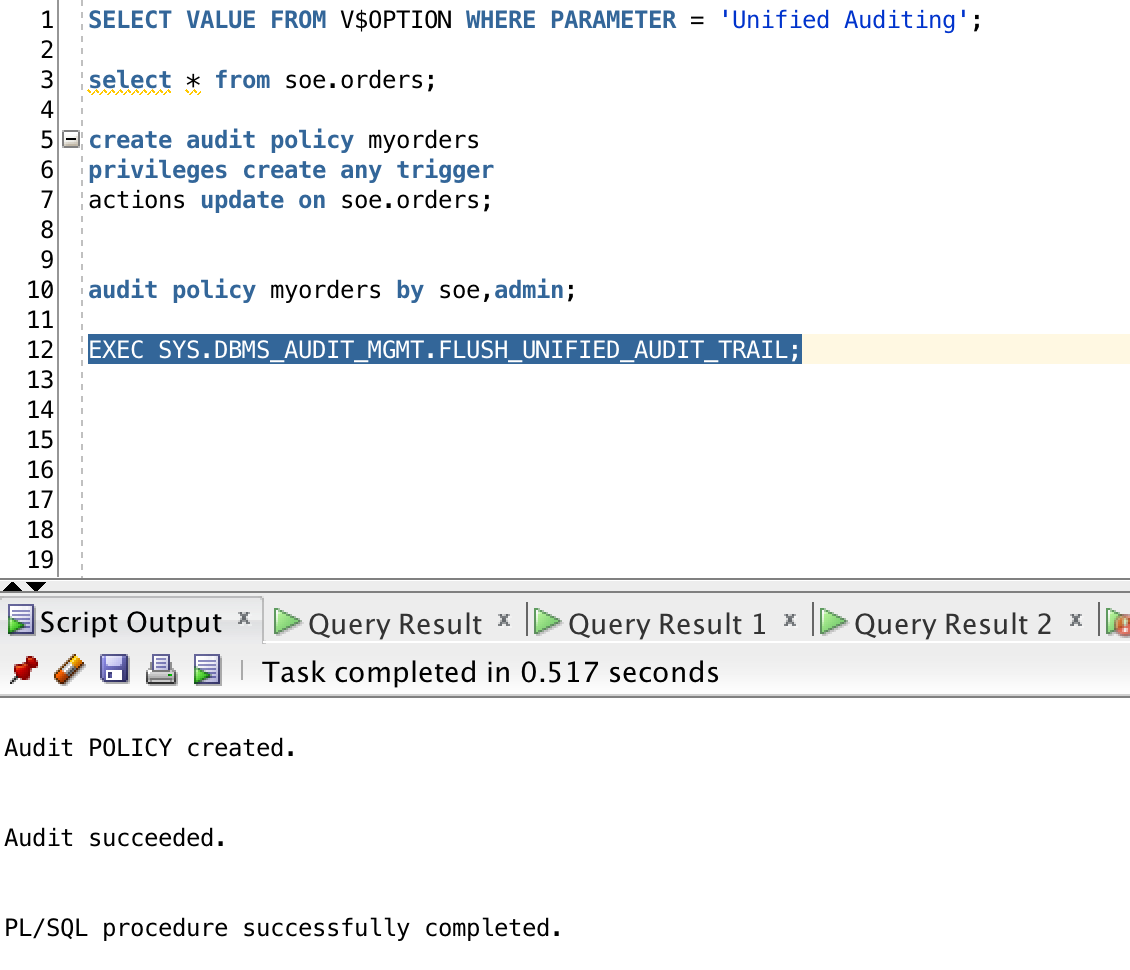
****

The output should be “Audit succeeded”.

When unified auditing is enabled in Oracle Database, the audit records are populated in this new audit trail. This view displays audit records in tabular form by retrieving the audit records from the audit trail.

Be aware that if the audit trail mode is QUEUED, then audit records are not written to disk until the in-memory queues are full. The following procedure explicitly flushes the queues to disk, so that you can see the audit trail records in the UNIFIED\_AUDIT\_TRAIL view:

**EXEC SYS.DBMS\_AUDIT\_MGMT.FLUSH\_UNIFIED\_AUDIT\_TRAIL;**

****

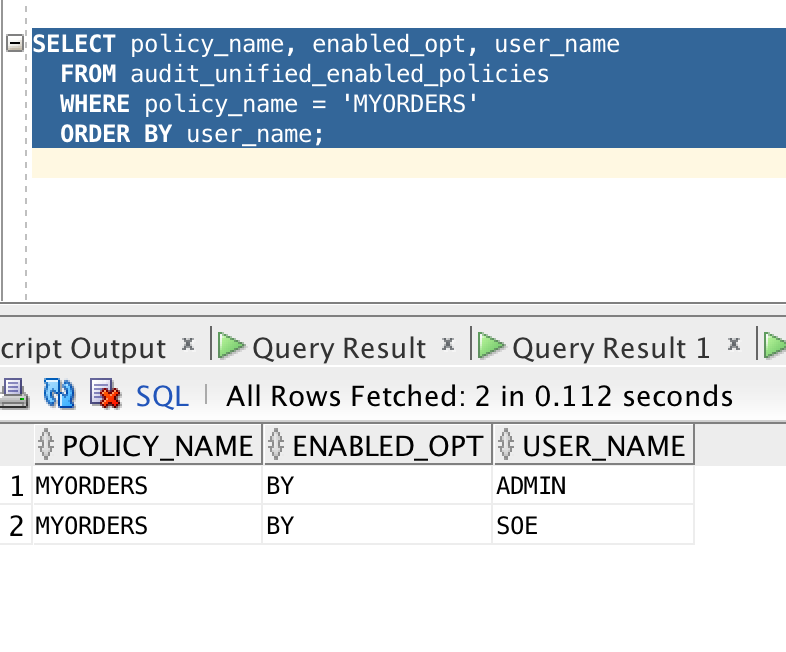
Now lets see if the policy is created. Enter the following command in SQL Workbench:

**SELECT policy\_name, enabled\_opt, user\_name**

**FROM audit\_unified\_enabled\_policies**

**WHERE policy\_name = 'MYORDERS'**

**ORDER BY user\_name;**

****

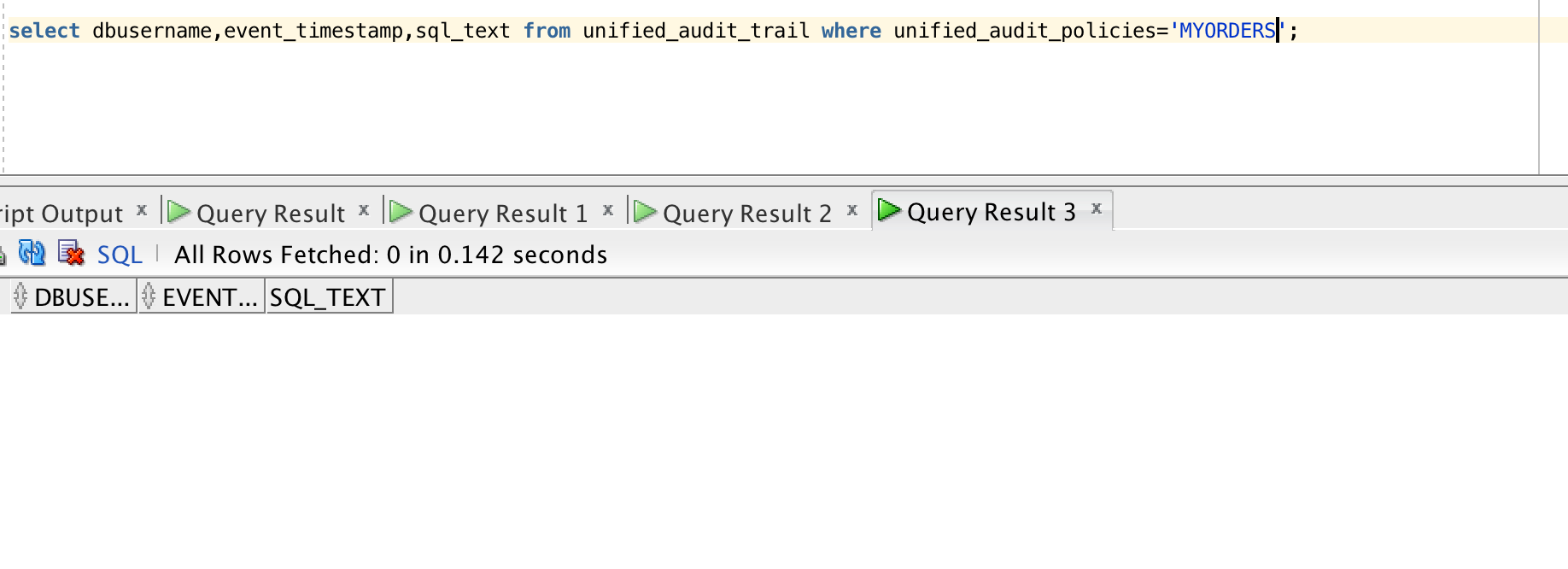
The ENABLED\_OPT shows if a user can access the policy. You can also exclude users from unified policy: AUDIT POLICY MYORDERS EXCEPT MEGHA;

**View the generated audit records.**

In order to look at all audit records, query the UNIFIED\_AUDIT\_TRAIL table.

Copy paste the following SQL statement:

**select dbusername,event\_timestamp,sql\_text from unified\_audit\_trail where unified\_audit\_policies='MYORDERS';**



Right now, there are no updates made on this table. Hence let’s go back to swingbench and generate workload again on the database.

./charbench -c ../configs/SOE\_Server\_Side\_V2.xml \

-cf /c/wallets/wallet\_atpxweek.zip \

-cs atpxweek\_high \

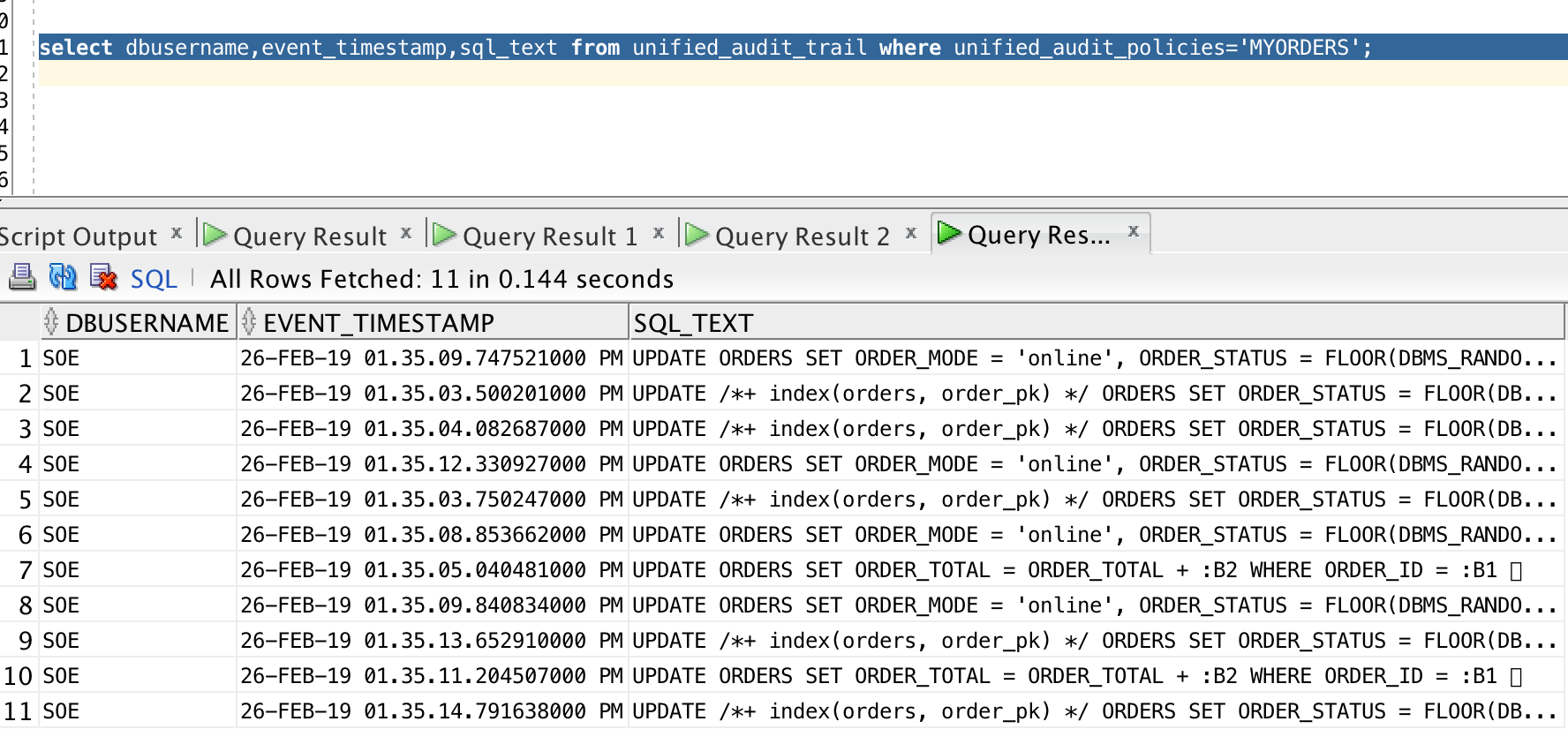
-u soe \

-p Atpxweek2018 \

-v users,tps,vresp \

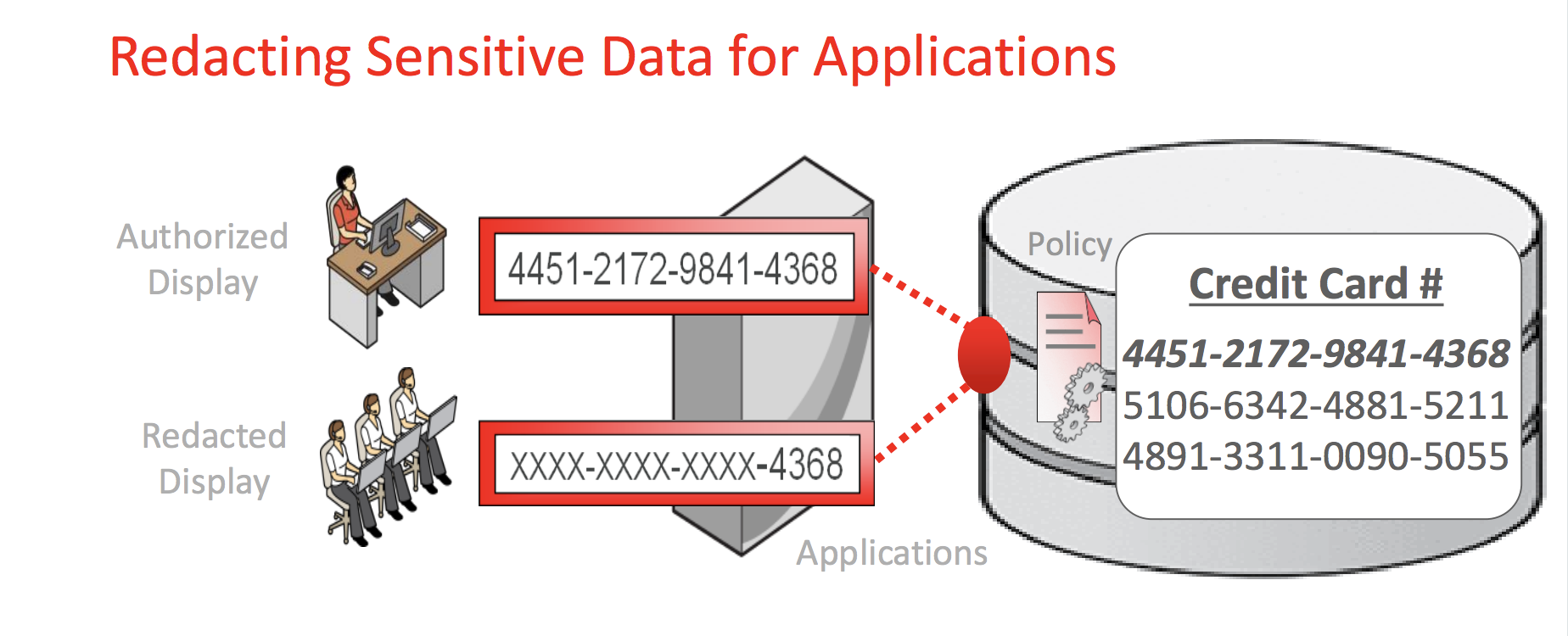
-uc 64

Wait for a while till you start seeing TPS. Let’s view the audit records again:



**REDACTION**

Redaction allows for the masking of sensitive data from the end-user layer. Until now if you wanted to mask the data on real time you needed to do it on the application layer or to use either custom made views or Virtual Private Database, all these solutions lacking functionalities that Data Redaction finally brings. With Data Redaction is now possible to easily totally or partially mask the data, randomize the data and set the masking conditions

****

The objective of this lab is to mask the CREDIT\_NUMBER column from the CREDIT\_DETAILS table that we have generated using swingbench in the previous lab for the user soe.

**PRE-REQUISITES:**

**Grant dbms\_redact access to soe**

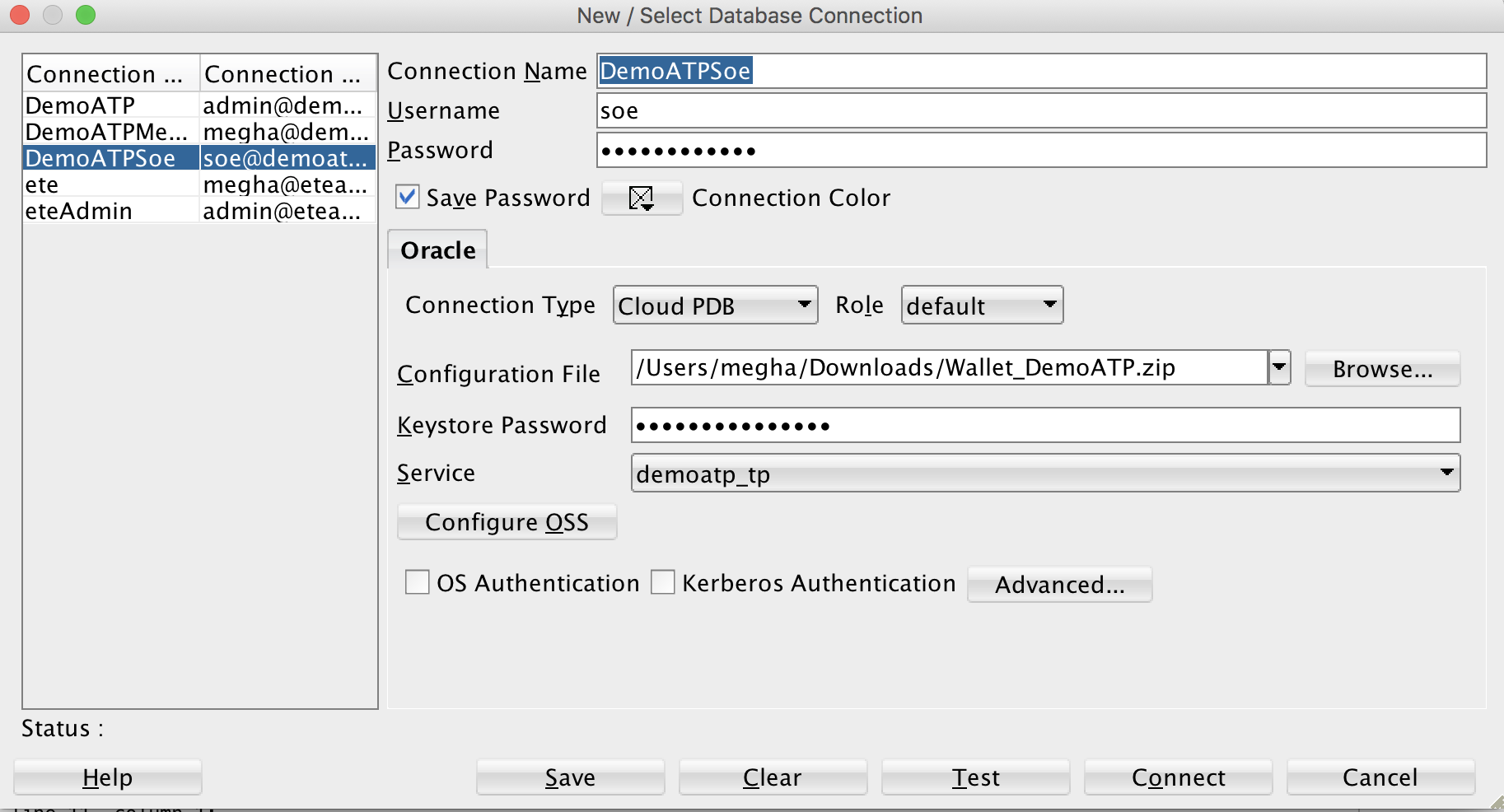
We have to grant DBMS\_REDACT access to the user soe. Type the following SQL statement in SQL Workbench while you are still connected to database as admin:

**grant execute on DBMS\_REDACT to soe;**

**CREATE REDACTION ON CARD\_NUMBER**

**You need to have a connection to database through soe.**

1. Connection Name: **DemoATPSoe**
2. Username: **soe**
3. Password: **Atpxweek2018**
4. Connection Type: **Cloud PDB**
5. Configuration File: **Path to your wallet**
6. Keystore Password: **Password entered while downloading wallet.**



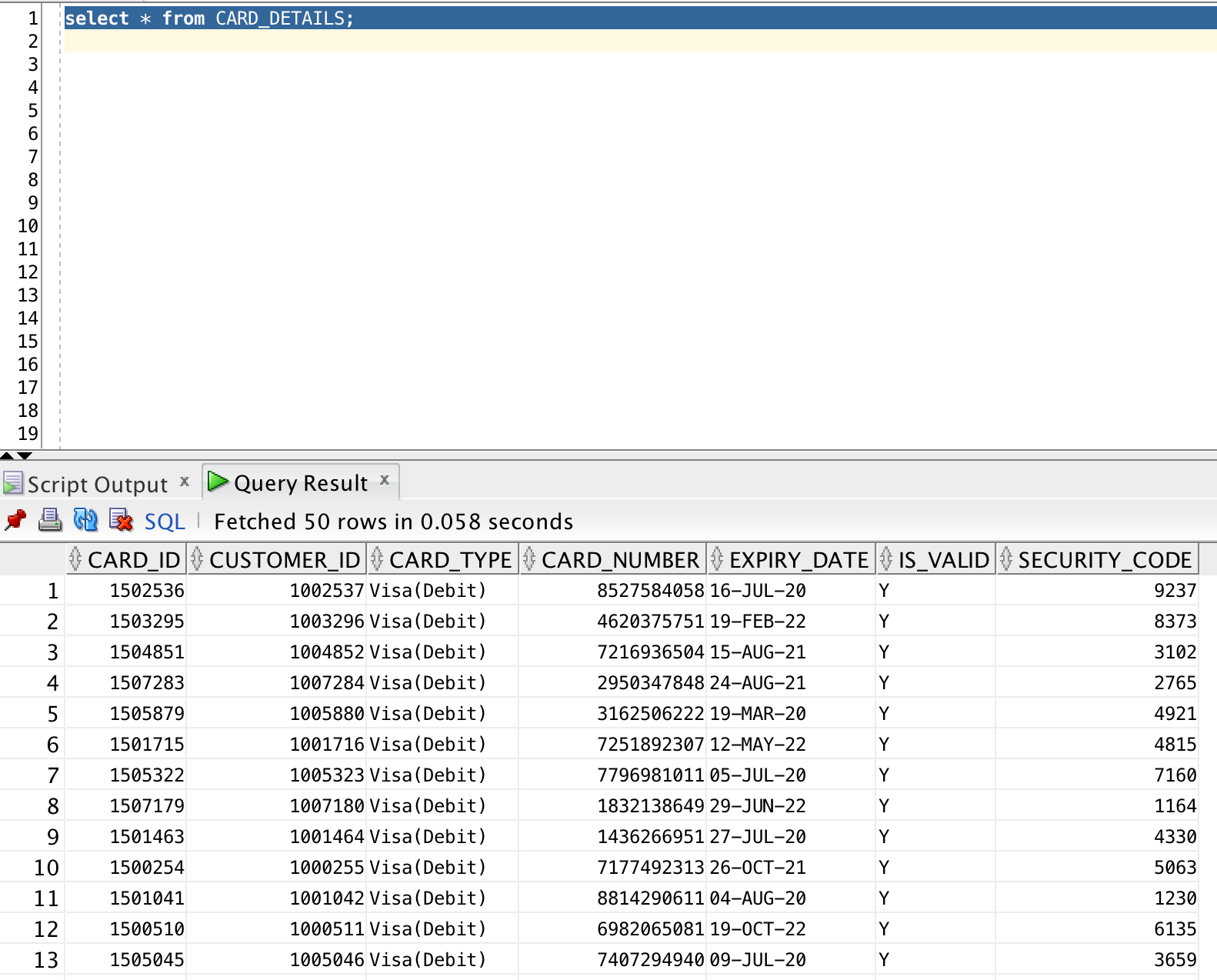
Click on Test, if it shows success, click on save and then click on connect.

We will do redaction on the card\_number column of the table CARD\_DETAILS.

Let’s look at the details of the table.

Copy paste the following command:

**Select \* from CARD\_DETAILS;**



You can see the CARD\_NUMBER column in the CARD\_DETAILS.

Creating a new redaction policy is done using the ADD\_POLICY procedure in

the DBMS\_REDACT package. A policy is made up of several distinct sections.

* Identify the object : The OBJECT\_SCHEMA, OBJECT\_NAME and COLUMN\_NAME parameters identify the column to be redacted.
* Give it a name : The POLICY\_NAME parameter assigns a name to the policy.
* What should happen? : The FUNCTION\_TYPE parameter determines the type of redaction that should take place. The allowable values are listed [here](http://docs.oracle.com/database/121/ARPLS/d_redact.htm#ARPLS73807). Depending on the type of redaction selected, you may be required to specify the FUNCTION\_PARAMETERS or various REGEXP\_\* parameters.
* When should it happen? : The EXPRESSION parameter determines when the redaction should take place. For example, an expression of "1=1" means the redaction will always take place. Alternatively, situational expressions can be defined using the SYS\_CONTEXT function.

Copy paste the following SQL statement :

BEGIN

DBMS\_REDACT.ADD\_POLICY(

object\_name => 'CARD\_DETAILS',

policy\_name => 'CD\_POLICY',

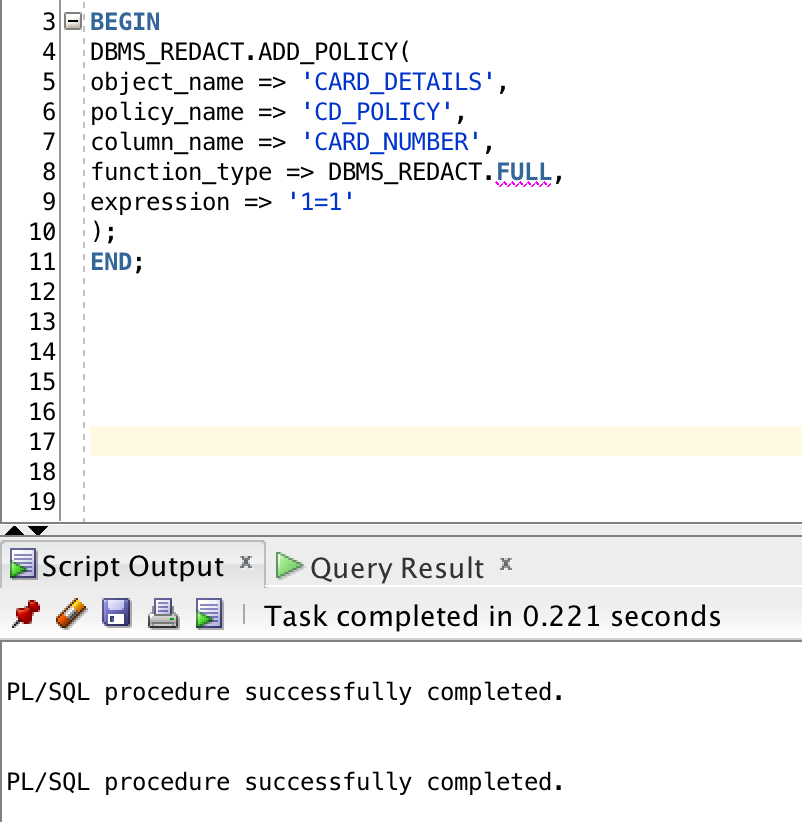
column\_name => 'CARD\_NUMBER',

function\_type => DBMS\_REDACT.FULL,

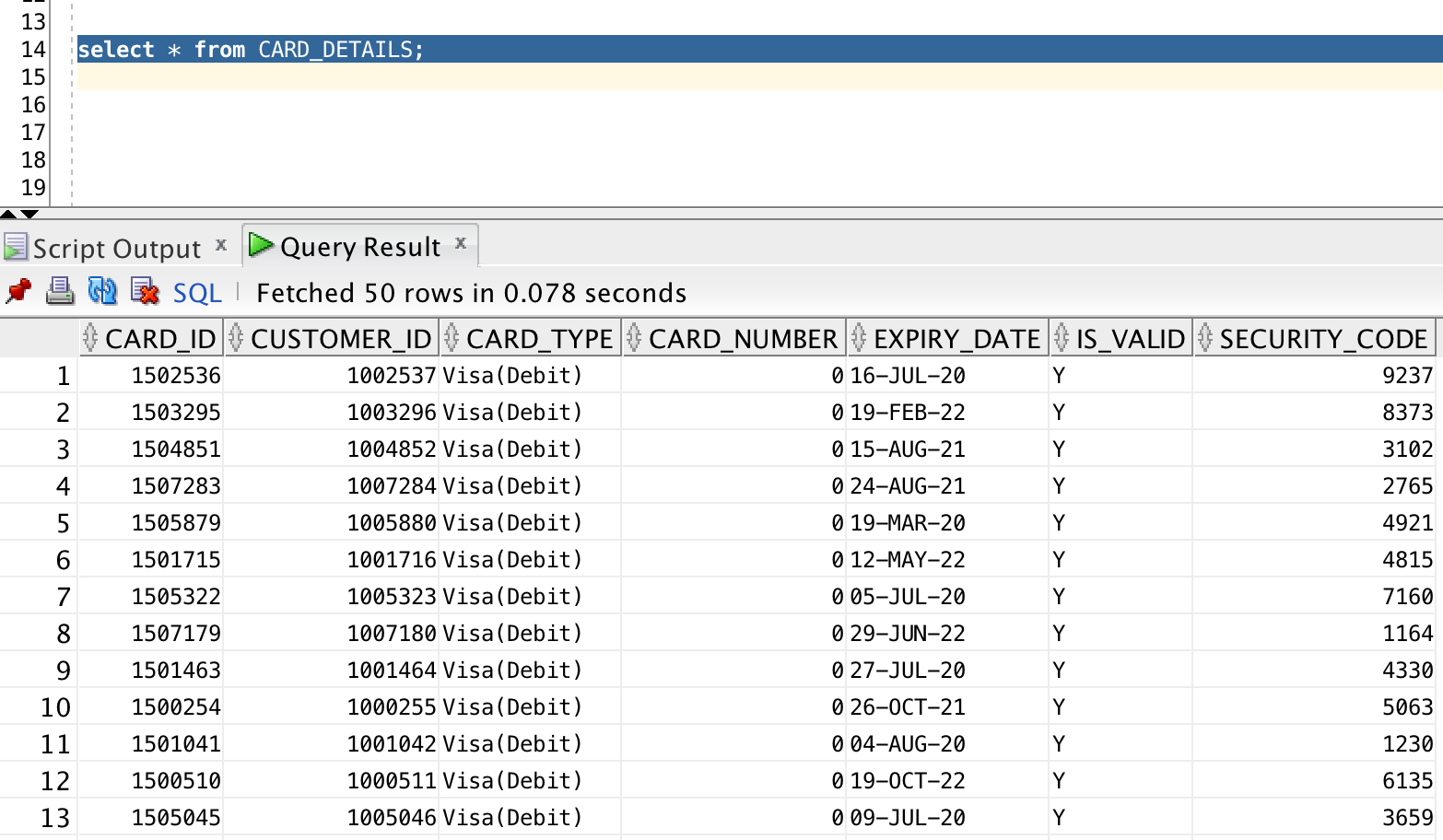
expression => '1=1'

);

END;

****

Let’s view the CARD\_DETAILS table again.



We can see the CARD\_NUMBER column is now redacted to the number "0".