Introduction to SAP HANA Cloud

DAT-161

Exercises / Solutions  
Robert Waywell / SAP  
Jason Hinsperger / SAP

Table of ContentS

[Pre-Requisites 3](#_Toc57206279)

[Exercise 1. Walk Through the Provisioning Process 5](#_Toc57206280)

[Exercise 2. Update a provisioned instance (Not Applicable for Trial Instances) 7](#_Toc57206281)

[Exercise 3: Introduction to HANA Cloud, Data Lake 8](#_Toc57206282)

[Exercise 4. Setting Up a Remote Source To Amazon Athena 14](#_Toc57206283)

[Exercise 5: Using NSE in HANA Cloud 22](#_Toc57206284)

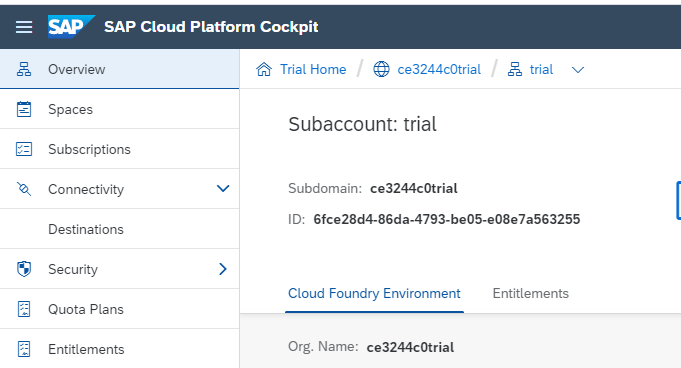
HANA Cloud is a service designed deliver a complete set of data management capabilities in the cloud and act as the gateway to all of your enterprise data. Along with the HANA database, HANA Cloud provides unique features like the ability to access remote data sources, enabling an easy transition between federated and replicated access to that data. HANA Cloud also enables you to control TCO by enabling storage of data of various temperatures using cost effective storage, including HANA Native Storage Extensions and the integrated HANA data lake.

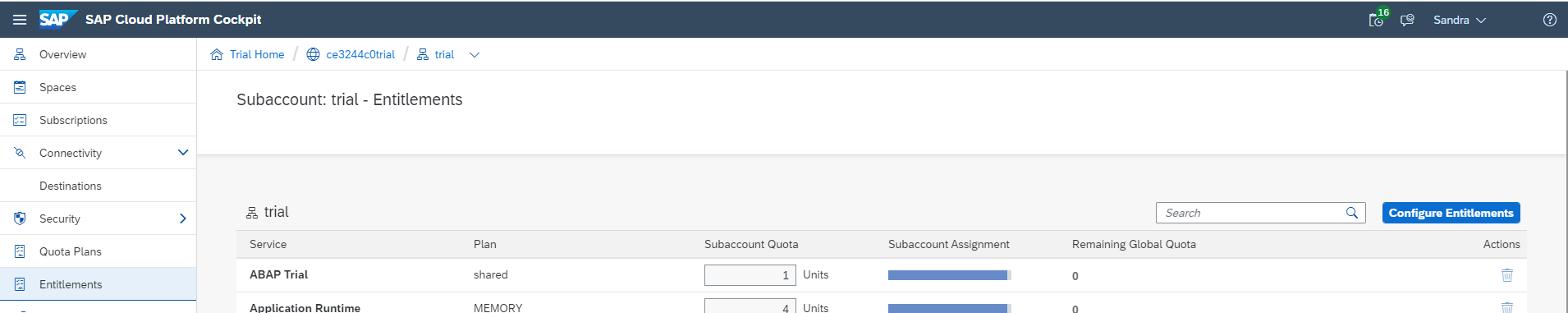
Rather than focus on the capabilities of the HANA database in HANA Cloud, for which there is already an abundance of learning materials, this set of hand-on exercises enables you to explore some of the above-mentioned features that make HANA Cloud unique.

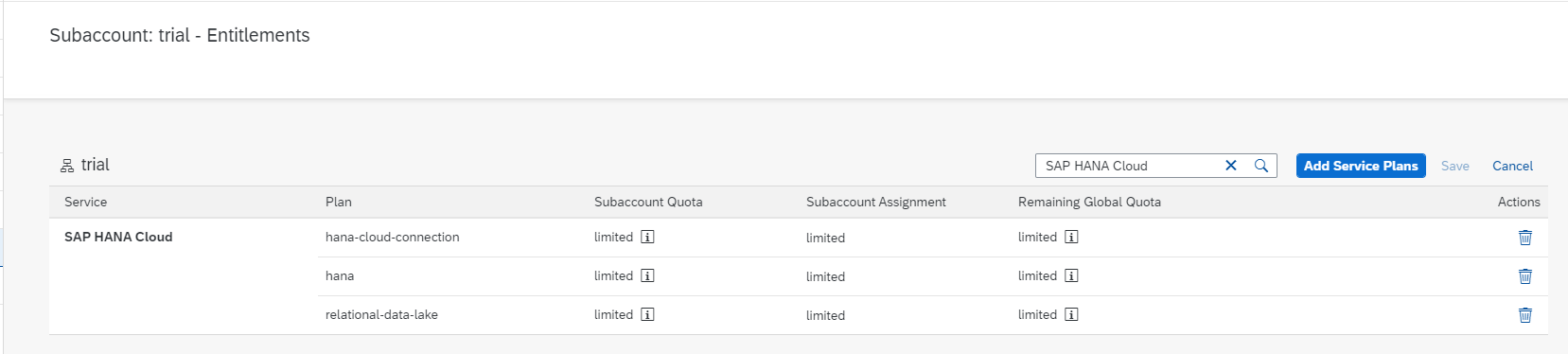
* You will go through the provisioning process (if you haven’t already),
* You will work with the HANA data lake to create tables and add data to cold storage,
* You will connect to a virtual cloud data sources (Amazon Athena), and query and replicate data
* You will explore the use of warm storage using the HANA native storage extension (NSE) feature of HANA Cloud.

## Pre-Requisites

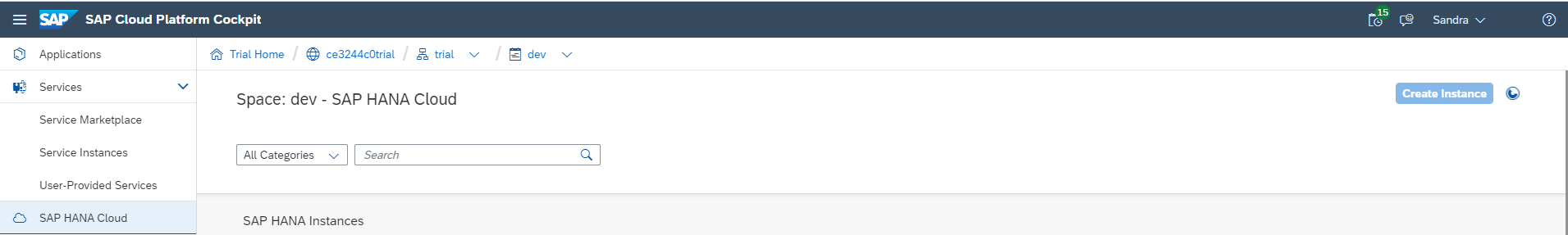
The following exercise uses HANA Cloud Trial. If you have a trial account, go to Subaccount Entitlements and add SAP HANA Cloud if you haven’t already. If you do not have an active trial account, sign-up through our HANA Cloud Trial registration form – available here: <https://www.sap.com/cmp/td/sap-hana-cloud-trial.html>







Once you log on to SAP Cloud Platform Cockpit, ensure that you have navigated to the correct global account, sub account, space, and you have selected SAP HANA Cloud from the left sidebar.



## Exercise 1. Walk Through the Provisioning Process

The goal of this exercise is to create a HANA Cloud instance and a Data Lake instance connected to it. Note, trial users may only create one instance per subaccount per geographic location. If you have already created an instance with your trial account, but have not created a data lake instance, complete steps 2 and 3. If you already have a data lake instance, skip to step 4 to ensure it is running.

| Explanation | Screenshot |
| --- | --- |
| 1. From your SAP Cloud Platform trial account, open the “SAP HANA Cloud” item and click “Create Database”  Follow through the wizard to create a HANA Cloud instance.  Be sure to create a HANA data lake instance when given the option. |  |
| 2. Complete step 2 and 3 if you already have a HANA Cloud instance, but have not created a HANA data lake instance.  From the HANA Cloud service tile inside of SCP, click ‘Actions’ and choose “Monitor landscape” |  |
|
| 3.  When the HANA Cloud manager opens, choose “Add Data Lake” from the “Actions” menu in the landscape monitor. |  |
| 4.  When the HANA Cloud instance is ready the status will change to *Running*.  If the instance was previously created and is in the “Stopped” state, you can start it from the “Actions” menu. |  |

## Exercise 2. Update a provisioned instance (Not Applicable for Trial Instances)

The goal of this exercise is to update the resources allocated to a provisioned data lake instance. HANA Cloud trial instances cannot be edited. This exercise only applies to provisioned instances of the full HANA Cloud service. You cannot update provisioning for a HANA Cloud Trial instance.

|  |  |
| --- | --- |
| 1.  Open the HANA Cloud management interface and click on the “Edit” option under the Actions extended menu. |  |
| 2.  Increase Compute to 10 vCPU and increase storage to 4 TB.  Note: we have now allocated 8 vCPUs to workers, and 2 vCPUs are coordinators.  Click on *Save.* |  |
| 3.  Wait for the updated instance to start again. Once *Starting* turns to *Running*, the changes have been reflected in the Data Lake instance. This process could take a few minutes. |  |

## Exercise 3: Introduction to HANA Cloud, Data Lake

The goal of this exercise is to create some schema and add some data to a HANA cloud, data lake instance, then query that data through the standard HANA Cloud interfaces.

The sample data we are using comes from the open-source Global Database of Events, Language, and Tone (GDELT) project. If you're unfamiliar with the information inventoried by the GDELT project, go to **www.gdeltproject.org** for more information. We use a small subset of data from GDELT that lists events and event descriptions from various locations around the globe.

| Explanation | Screenshot |
| --- | --- |
| 1.  Navigate to the HANA Cloud instance you created and open the landscape monitor. |  |
| 2.  Under the action menu, open the SAP HANA Database Explorer.  If prompted for login credentials, enter “DBADMIN” as the user id, and the password you specified when you created your HANA Cloud instance. |  |
|
| 3.  Open a SQL console for your trial instance, If there is not one already open. |  |
| 4.  In the SQL window, create a new schema for your data lake with the following command:  *Create Schema HDLExercise;*  Click the green arrow to execute the command. |  |
| 5.  Check that the Schema “HDLExercise” has been created by navigating to Catalog, Schemas, and then type “HDLExercise” in the Choose schema input form.  Ensure that HDLExercise exists. |  |
| 6.  First, we will create a HANA table which we will use later to demonstrate federated query access between the HANA Cloud, HANA database and the HANA data lake.  Copy and paste the content of DAT161Exercise\_CameoCodes.sql into the SQL console.  Execute the command by pressing the run button.  This command creates a table called CAMEOCODES and inserts some data into it. |  |
| 7.  Copy and paste the content of DAT161*Exercise\_CreateTable.sql* into SQL console. Execute the command by pressing the run button.  This command creates the following tables in data lake:  EVENT, GKG, MENTIONS.  Note the create table statements are wrapped inside the *SYSRDL#CG.remote\_execute* function. *remote\_execute()* is a stored procedure which allows you to execute SQL natively against HANA data lake. |  |
| 8.  Verify the tables have been created in the data lake.  First, navigate to *Catalog*, then *Remote Sources* from the left side bar. Click on *Remote Sources*. You should see one remote source called *SYSRDL#CG\_SOURCE*.  Double Click on *SYSRDL#CG\_SOURCE*. |  |
| 9.  A new remote source tab is open next to the „*SQL Console 1.sql“,* called „SYSRDL#CG\_SOURCE“*.*  On the remote source tab, open the schema menu and choose *SYSRDL#CG* from the drop down as the remote schema.  Click on the Search button. |  |
| 10.  You should see the list of 3 tables that you created in step 6.  Click the checkbox beside each table and then click the “Create Virtual Objects” button. |  |
| 11.  Set the „*Object Names Prefix“* to „V\_“  Then set the „Schema“ name to the name of the schema created in step 4 (*HDLEXERCISE*), and click on create.  This creates a HANA virtual table for each data lake table, and names the virtual table V\_<name of data lake table>. For instance, the virtual table referring to the data lake EVENT table will be named V\_EVENT.  The virtual tables are created in the HANA schema HDLEXERCISE. |  |
| 12.  To verify the virtual tables have been created successfully, navigate to the *Catalog* in the left hand nav., then choose *Tables*.  Find the *HDLExercise* in the list and you should see the virtual tables that you have just created. |  |
| 13.  Next we Will load some data into the data lake tables.  Copy and paste the command from DAT161Exercise\_*LoadTable.sql* into the SQL console. Execute the command by pressing the run button. Ensure the commands execute successfully.  This command loads data from a public AWS s3 bucket into the following tables in data lake:  EVENT, GKG, MENTIONS.  Note that we supply the location of the s3 bucket, its region, its access key id, as well as its secret access key directly in the LOAD statement so that the data lake can read the data directly from the S3 bucket to maximize load performance. |  |
| 14.  Now we can query the data from the data lake using the previously created virtual tables.  Copy and paste the command from *DAT161DataLake\_Query1.sql* into SQL console. Execute the command by pressing the run button. Ensure the commands execute successfully. |  |

That concludes this exercise. You have successfully configured and created schema in your HANA Cloud data lake instance and loaded data into it. You have also successfully combined HANA and HANA data lake data together in a single query.

## Exercise 4. Setting Up a Remote Source To Amazon Athena

Amazon Athena is a serverless SQL engine which allows you to directly query files that are stored in S3 object storage. The goal of this exercise is to walk through the setup of a remote data source in HANA Cloud to Amazon Athena. We will be using the native SDA (Smart Data Access) support to create a connection to Amazon Athena and then query data in an Athena table from the SAP HANA Cloud Database Explorer. You will also replicate data from Athena into HANA Cloud in order to improve the performance of the Athena queries.

A small Athena instance has been created for this hands on session which contains some schema and data from a well known dataset called TPCH.

| Explanation | Screenshot |
| --- | --- |
| 1.  In order to create a remote source to Athena, we must first create 2 certficates in HANA Cloud to enable the connection. One certificate for connections to AWS, and one certficate for connections to S3, where the data for the Athena instance is stored.  Open the file DAT161\_AthenaCertsPSE.sql and copy the first 2 statements in the file to add the certficates to your HANA Cloud instance. |  |
| 2.  Now we need to create a certificate collection (also known as a PSE – personal security environment) for these certificates that our remote source will refer to when we create the connection to Athena.  Execute the following statement to create a new PSE called HTTPS:  CREATE PSE HTTPS; |  |
| 3.  Now we must add the certificates we created in step 1 to the PSE we created in step 2. To do that we must first lookup the id assigned to the AWS certificate when it was added to HANA. Execute the following statement to get the certificate id for the certificate created with the comment ‘AWS’:  SELECT CERTIFICATE\_ID FROM CERTIFICATES WHERE COMMENT = 'AWS';  Note the value of CERTIFICATE\_ID as it will be used in the next step. It may not be the same as you see in the picture to the right. |  |
| 4.  Add the certificate id recorded in step 3 to the PSE created in step 2.  Execute the following statement:  ALTER PSE HTTPS ADD CERTIFICATE <AWS CERT ID>;  Where <AWS CERT ID> is the CERTIFICATE\_ID you wrote down from step 3. |  |
| 5.  Lookup the id assigned to the S3 certificate when it was added to HANA. Execute the following statement to get the certificate id for the certificate created with the comment ‘S3’:  SELECT CERTIFICATE\_ID FROM CERTIFICATES WHERE COMMENT = 'S3';  Note the value of CERTIFICATE\_ID as it will be used in the next step. It may not be the same as you see in the picture to the right. |  |
| 6.  Add the certificate id recorded in step 5 to the PSE created in step 2.  Execute the following statement:  ALTER PSE HTTPS ADD CERTIFICATE <S3 CERT ID>;  Where <S3 CERT ID> is the CERTIFICATE\_ID you wrote down from step 5. |  |
| 7.  Now that we have added the certificates to our certificate collection, we need to set the purpose of the collection so that HANA knows these certificates can be used to validate remote source connections.  Execute the following statement:  SET PSE HTTPS PURPOSE REMOTE SOURCE; |  |
| 8. Now we are ready to create our connection to Amazon Athena.  Copy the CREATE REMOTE SOURCE statement from DAT161Exercise\_Athena.sql and execute it. |  |
| 9.  Navigate to the remote sources folder of your instance. You should see a new remote source called “MY\_ATHENA”.  Double click to open the Athena remote connection. |  |
| 10.  Similar to the HANA data lake exercise, open the “schema” drop down list, and select “tpch\_sf1000” as your schema. Then click the “Search” button to get a list of Athena tables. |  |
| 11.  Click the checkbox next to the “nation” table, and then click the “Create Virtual Objects” button. |  |
| 12.  Set the object name to ATH\_NATION and select HDLEXERCISE as the schema name.  Then click the “Create” button. |  |
| 13.  Navigate to the list of tables for your HDLEXERCISE schema in the left hand window. You should see the new ATH\_NATION table. |  |
| 14.  Move to the SQL window, and query the table from Athena by executing the following statement:  SELECT \* FROM ATH\_NATION;  When you execute the statement, HANA Cloud connects to the Athena instance, executes the statement and retrieves the results.  Note the execution time for the statement. |  |
| 15.  If we plan to use this table extensively within HANA Cloud, and it does not change frequently, we can choose to replicate the data into HANA Cloud from Athena.  To replicate the data for the Athena “nation” table into HANA Cloud, execute the following statement in the Database Explorer SQL console:  ALTER VIRTUAL TABLE HDLEXERCISE.ATH\_NATION ADD SHARED SNAPSHOT REPLICA; |  |
| 16.  Query the remote Athena table again by executing the same statement as in step 14:  SELECT \* FROM ATH\_NATION;  Note that the execution time has improved substantially for the statement. |  |

That concludes this exercise. You have successfully created a connection from HANA Cloud to Amazon Athena, create and queried a virtual table in Athena, and replicated data from Athena into HANA Cloud in order to improve query performance.

## Exercise 5: Using NSE in HANA Cloud

In this set of exercises you will explore the warm storage component of SAP HANA Cloud. This is called HANA native storage extension and you can find out more about it here

| Explanation | Screenshot |
| --- | --- |
| 1.  If it is not already open, open the Database Explorer for your HANA Cloud Instance. See Exercise 3, steps 1-3 if you are not familiar with how to do this. |  |
| 2.  With these first set of steps, you will create a table in NSE storage.  Start by opening a SQL console by selecting the database and clicking on the SQL icon |  |
| 3.  First create a table in HANA Cloud main memory by running the following statement:  CREATE COLUMN TABLE "DBADMIN"."T1\_IN\_MEMORY"  ( COL1 INTEGER PRIMARY KEY,  COL2 VARCHAR(30)  ); |  |
| 4.  Now create an NSE table by first copying and pasting the following statement to the SQL console:  CREATE COLUMN TABLE "DBADMIN"."T2\_NSE"  ( COL1 INTEGER PRIMARY KEY,  COL2 VARCHAR(30)  ) PAGE LOADABLE;  Note: notice the only difference in syntax when creating an NSE table compared to an in-memory table is the PAGE LOADABLE clause |  |
| 5.  Insert values into both tables by running the following statements:  INSERT INTO "DBADMIN"."T1\_IN\_MEMORY" VALUES (1,'First in-memory record');  INSERT INTO "DBADMIN"."T1\_IN\_MEMORY" VALUES (2,'Second in-memory record');  INSERT INTO "DBADMIN"."T2\_NSE" VALUES (1, 'First page loadable record');  INSERT INTO "DBADMIN"."T2\_NSE" VALUES (2, 'Second page loadable record');  **Note**: there is no syntax difference between the INSERT INTO statements. Similarly, no extra syntax is required for any DML operations (UPDATE and DELETE) on page loadable data. |  |
| 6.  Query the tables by running the following SELECT statements:  SELECT \* FROM "DBADMIN"."T1\_IN\_MEMORY" WHERE COL1 = 2;  SELECT \* FROM "DBADMIN"."T2\_NSE" WHERE COL1 = 1;  **Note**: The result from the first query is returned under the tab “Result 1” and the result from the second query is returned under the tab “Result 2”. |  |
| 7.  The tables can also be viewed from the UI by navigating to Catalog and then Tables |  |
| 8.  Lets look at the load configuration for the tables we just created.  Clear the SQL console. Then copy and paste the following statement into the console. Run the statement.  SELECT SCHEMA\_NAME, TABLE\_NAME, LOAD\_UNIT FROM SYS.TABLES  WHERE SCHEMA\_NAME = 'DBADMIN';  **Note**: for this exercise, we are specifically looking at the SCHEMA\_NAME, TABLE\_NAME, and LOAD\_UNIT columns in the SYS.TABLES view. |  |
| 9.  Click on row “1”.  Since we did not explicitly assign a LOAD UNIT in the CREATE TABLE statement for the T1\_IN\_MEMORY table, it shows a LOAD UNIT of **DEFAULT**. The default LOAD UNIT for a table is **COLUMN**. If we had explicitly specified **COLUMN LOADABLE** in the CREATE TABLE statement for the T1\_IN\_MEMORY table, then the LOAD UNIT would show as **COLUMN**. |  |
| 10.  click on row “2”.  The LOAD UNIT for the **T2\_NSE** table is reported as **PAGE** indicating that this is a page loadable table. |  |
| 11.  Now lets’ create a table with a specific column that is page loadable.  Clear the SQL console and run the following statement:  CREATE COLUMN TABLE "DBADMIN"."T3\_ONE\_COLUMN\_IN\_NSE"  ( COL1 INTEGER PRIMARY KEY,  COL2 VARCHAR(30),  COL3 CLOB PAGE LOADABLE  );  Note that the first 2 column definitions are the same as we used in the T1 and T2 tables. For our T3 table, a third column has been added to the table definition and specifically designated that column to be **PAGE LOADABLE.** Note the location of the PABE LOADABLE clause as part of the column definition. There is no **PAGE LOADABLE** clause after the column definitions. This means that the table will use the DEFAULT load unit (which is COLUMN LOADABLE) for any columns that are not explicitly set to be PAGE LOADABLE. |  |
| 12.  Now run the following statement to insert values into T3\_ONE\_COLUMN\_IN\_NSE:  INSERT INTO "DBADMIN"."T3\_ONE\_COLUMN\_IN\_NSE" VALUES (1, 'Col2 is COLUMN LOADABLE', 'Col3 is a CLOB to handle really long verbose content');  INSERT INTO "DBADMIN"."T3\_ONE\_COLUMN\_IN\_NSE" VALUES (2, 'Col2 is always in memory', 'Col3 can be read into the buffer cache when needed but only when needed'); |  |
| 13.  Query the table by running the following statement:  SELECT \* FROM "DBADMIN"."T3\_ONE\_COLUMN\_IN\_NSE"; |  |
| 14.  Now lets’ look at the load configuration for the individual columns.  Query the SYS.TABLES view to see what LOAD UNIT is configured at the table level by running the following statement:  SELECT SCHEMA\_NAME, TABLE\_NAME, LOAD\_UNIT FROM SYS.TABLES  WHERE SCHEMA\_NAME = 'DBADMIN'; |  |
| 15.  Notice the LOAD UNIT for the T3\_ONE\_COLUMN\_IN\_NSE table is shown as “DEFAULT” (which is equivalent to COLUMN). If any 1 column in a table is COLUMN LOADABLE, then the LOAD UNIT for the table will be reported as DEFAULT or COLUMN |  |
| 16.  To see the LOAD UNIT configuration at a **column level**, run the following statement:  SELECT SCHEMA\_NAME, TABLE\_NAME, COLUMN\_NAME, LOAD\_UNIT FROM SYS.M\_CS\_COLUMNS  WHERE SCHEMA\_NAME = 'DBADMIN'; |  |
| 17.  Note the **LOAD UNIT** for the third column is shown as **PAGE** because that column was specifically designated as **PAGE LOADABLE** in the CREATE TABLE statement in Exercise 2 Step 1.  Also note that the **LOAD UNIT** for the first 2 columns of **T3\_ONE\_COLUMN\_IN\_NSE** (row 6 and 7) are shown as **COLUMN** because that is the default LOAD UNIT and no other LOAD UNIT was specified for these columns in the CREATE TABLE statement |  |
| 18.  Now lets create a table which has a specific partition stored in NSE.  Run the following statement to create a table:  CREATE COLUMN TABLE  "DBADMIN"."T4\_ONE\_PARTITION\_IN\_NSE"  ( COL1 INTEGER PRIMARY KEY,  COL2 VARCHAR(30),  COL3 CLOB  )  PARTITION BY RANGE ( "COL1" )  ( ( PARTITION 1 <= VALUES < 10 COLUMN LOADABLE,  PARTITION 10 <= VALUES < 20 PAGE LOADABLE,  PARTITION OTHERS  )  );  **Note**: The column definitions are the same as the columns created in T3, but since no **LOAD UNIT** is specified, all the columns will be assigned the default load unit of **COLUMN**.  The new CREATE TABLE clause that is being used for the T4 table is the **PARTITION BY RANGE** clause. This defines a single level range partitioning based on the values in COL1 |  |
| 19.  Note: In the previous SQL statement, the first partition will hold records where the VALUES of COL1range from 1 to 9 (COL1 is an INTEGER column). This partition is explicitly assigned to be stored **in-memory** as **COLUMN LOADABLE**.  The second partition will hold records where the VALUES of COL1range from 10 to 19. This partition is explicitly assigned to be stored **on disk** as **PAGE LOADABLE**.  The 3rd partition is the 'catch-all' partition OTHERS. The OTHERS partition is not assigned records in a specific range. Instead the OTHERS partition will hold any records where the value of COL1 falls outside of any of the other partitions. And since no LOAD UNIT is specified for the OTHERS partition, it will be assigned the **DEFAULT** load unit of **COLUMN**. |  |
| 20.  Insert a record into each of the 3 partitions by running the following statement:  INSERT INTO "DBADMIN"."T4\_ONE\_PARTITION\_IN\_NSE" VALUES (1, 'This is column 2', 'The COL1 value falls in the range of the 1st partition so this record will be in a COLUMN LOADABLE partition. ');  INSERT INTO "DBADMIN"."T4\_ONE\_PARTITION\_IN\_NSE" VALUES (18, 'Column 2 again', 'The COL1 value falls in the range of the 2nd partition so this record will be in a PAGE LOADABLE partition');  INSERT INTO "DBADMIN"."T4\_ONE\_PARTITION\_IN\_NSE" VALUES (99, 'Hi from COL2', 'The COL1 value for this record falls outside of the ranges of the explicitly defined partitions so this record will be in the catch-all OTHERS partition'); |  |
| 21.  Select from the table by running the following statement:  SELECT \* FROM "DBADMIN"."T4\_ONE\_PARTITION\_IN\_NSE"; |  |
| 22.  Review the table level LOAD UNIT for the T4 table by running the following statement:  SELECT SCHEMA\_NAME, TABLE\_NAME, LOAD\_UNIT FROM SYS.TABLES  WHERE SCHEMA\_NAME = 'DBADMIN'  AND TABLE\_NAME = 'T4\_ONE\_PARTITION\_IN\_NSE';  Note: Just as with the T1 and T3 tables, the LOAD UNIT for the **T4\_ONE\_PARTITION\_IN\_NSE** table is reported as **DEFAULT** because no LOAD UNIT was explicitly set for overall table. Remember that DEFAULT is equivalent to **COLUMN**. |  |
| 23.  Review the details of individual table partitions, including the LOAD UNIT configuration by running the following statement:  SELECT SCHEMA\_NAME, TABLE\_NAME, PART\_ID, LOAD\_UNIT FROM SYS.M\_TABLE\_PARTITIONS  WHERE SCHEMA\_NAME = 'DBADMIN';  Note: Since the only partitioned table in your schema is the **T4\_ONE\_PARTITION\_IN\_NSE** table, you will only see results for that table.  Partition IDs are assigned in the order that the PARTITION definitions appeared in the **CREATE TABLE** statement. |  |
| 24.  Note: In the previous result:  The LOAD UNIT of the first partition was specifically configured as **COLUMN LOADABLE** in the CREATE TABLE statement, so it is shown as fieldicon_147.  The LOAD UNIT of the second partition was specifically configured as **PAGE LOADABLE** in the CREATE TABLE statement, so it is shown as fieldicon_157.  The LOAD UNIT of the **OTHERS** partition, which was defined as the 3rd partition, was not specifically configured in the CREATE TABLE statement, so it defaults to  fieldicon_163. (Note that the OTHERS partition is required to always be COLUMN LOADABLE). |  |

That concludes this exercise. You have created tables and added data to warm storage in HANA Cloud using the Natuve Storage Extensions (NSE), and examined the meta-data for those tables that show which parts of the table are stored in hot storage (in-memory) and which tables are stored in warm storage (NSE).