

Teradata Vantage™ - Native Object Store

Getting Started Guide

Release 17.10




July 2021

Copyright and Trademarks

Copyright © 2020 - 2021 by Teradata. All Rights Reserved.

All copyrights and trademarks used in Teradata documentation are the property of their respective owners. For more information, see [Trademark Information](#).

Product Safety

Safety type	Description
 NOTICE	Indicates a situation which, if not avoided, could result in damage to property, such as to equipment or data, but not related to personal injury.
 CAUTION	Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury.
 WARNING	Indicates a hazardous situation which, if not avoided, could result in death or serious personal injury.

Third-Party Materials

Non-Teradata (i.e., third-party) sites, documents or communications ("Third-party Materials") may be accessed or accessible (e.g., linked or posted) in or in connection with a Teradata site, document or communication. Such Third-party Materials are provided for your convenience only and do not imply any endorsement of any third party by Teradata or any endorsement of Teradata by such third party. Teradata is not responsible for the accuracy of any content contained within such Third-party Materials, which are provided on an "AS IS" basis by Teradata. Such third party is solely and directly responsible for its sites, documents and communications and any harm they may cause you or others.

Warranty Disclaimer

Except as may be provided in a separate written agreement with Teradata or required by applicable laws, all designs, specifications, statements, information, recommendations and content (collectively, "content") available from the Teradata Documentation website or contained in Teradata information products is presented "as is" and without any express or implied warranties, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or noninfringement, which are hereby disclaimed. In no event shall Teradata corporation, its suppliers or partners be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of content, even if advised of the possibility of such damage.

The Content available from the Teradata Documentation website or contained in Teradata information products may contain references or cross-references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that Teradata Corporation intends to announce such features, functions, products, or services in your country. Please consult your local Teradata Corporation representative for those features, functions, products, or services available in your country.

The Content available from the Teradata Documentation website or contained in Teradata information products may be changed or updated by Teradata at any time without notice. Teradata may also make changes in the products or services described in the Content at any time without notice.

The Content is subject to change without notice. Users are solely responsible for their application of the Content. The Content does not constitute the technical or other professional advice of Teradata, its suppliers or partners. Users should consult their own technical advisors before implementing any Content. Results may vary depending on factors not tested by Teradata.

Machine-Assisted Translation

Certain materials on this website have been translated using machine-assisted translation software/tools. Machine-assisted translations of any materials into languages other than English are intended solely as a convenience to the non-English-reading users and are not legally binding. Anybody relying on such information does so at his or her own risk. No automated translation is perfect nor is it intended to replace human translators. Teradata does not make any promises, assurances, or guarantees as to the accuracy of the machine-assisted translations provided. Teradata accepts no responsibility and shall not be liable for any damage or issues that may result from using such translations. Users are reminded to use the English contents.

Feedback

To maintain the quality of our products and services, e-mail your comments on the accuracy, clarity, organization, and value of this document to: docs@teradata.com.

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed nonconfidential. Without any payment or other obligation of any kind and without any restriction of any kind, Teradata and its affiliates are hereby free to (1) reproduce, distribute, provide access to, publish, transmit, publicly display, publicly perform, and create derivative works of, the Feedback, (2) use any ideas, concepts, know-how, and techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, and marketing products and services incorporating the Feedback, and (3) authorize others to do any or all of the above.

Confidential Information

Confidential Information means any and all confidential knowledge, data or information of Teradata, including, but not limited to, copyrights, patent rights, trade secret rights, trademark rights and all other intellectual property rights of any sort.

The Content available from the Teradata Documentation website or contained in Teradata information products may include Confidential Information and as such, the use of such Content is subject to the non-use and confidentiality obligations and protections of a non-disclosure agreement or other such agreements to protect Confidential Information that you have executed with Teradata.

Contents

Chapter 1: Welcome to Native Object Store	6
NOS Functionality	8
Command Syntax	10
Changes and Additions	10
Chapter 2: Setting Up Access	11
Variable Substitutions for Examples	11
Setting Up a Test User	13
Setting Access Privileges	13
Controlling Foreign Table Access with an AUTHORIZATION Object	14
Running Examples Without an Authorization Object	15
Chapter 3: Reading Data	17
Sampling Data	17
CSV Format	17
JSON Format	17
Parquet Format	18
Setting Up to Run Examples	18
Examples: For Business Analysts	19
Examples: For DBAs and Advanced Users	24
Chapter 4: Writing Data to External Object Store	41
WRITE_NOS	41
Setting Up WRITE_NOS	41
Writing All Vantage Data to External Object Storage	42
Working with Manifest Files	44
Chapter 5: Using Delta Lake Manifest Files	51
Setting up Vantage to Delta Lake Integration	51
Examples: CREATE FOREIGN TABLE and READ_NOS Queries	51
Delta Lake Manifest Files Limitations	57
Chapter 6: Data Cleanup	59
Appendix A: External File Formats	60
Appendix B: Authentication for External Object Stores	65
Appendix C: Setting Up an Object Store for River Flow Data	69

Appendix D: Best Practices for Using NOS 71

Appendix E: Native Object Store Limitations 72

Appendix F: Terminology 75

Appendix G: Additional Information 76

Welcome to Native Object Store

Teradata Vantage™ is our flagship analytic platform offering, which evolved from our industry-leading Teradata® Database. Until references in content are updated to reflect this change, the term Teradata Database is synonymous with Teradata Vantage.

Using Native Object Store

Native Object Store (NOS) is a Vantage capability that enables [Business Analysts, System Administrators, Data Scientists, and Database Administrators](#) to use standard Teradata SQL and APIs to:

- Search and query CSV, JSON, and Parquet format datasets located on S3-compatible object store platforms
- Write data stored on Vantage to S3-compatible object store platforms

For more details, see [NOS Functionality](#).

Why Would I Use this Content?

This document shows you how to access your S3-compatible object storage from Advanced SQL Engine, so that you can analyze data stored in external object storage.

Examples are provided showing how to perform various tasks, such as sampling the externally stored data, loading external data into the database, joining external data to data stored in the database, filtering data for efficiency, and so on. You can query external object store by creating a foreign table in the database or by reading directly from external object store.

How Do I Use this Content?

These instructions are for Database Administrators:

Before using NOS, verify that the following steps are completed. The following table provides links to detailed instructions.

Operation	Examples
Create a user or modify existing users to have the required access privileges.	Set up test user Set access privileges
Create an authorization object.	Create authorization object for foreign table credentials
To run the Teradata-supplied examples to read from external object store, create a foreign table for your data format.	Setting Up to Run Examples
To run the Teradata-supplied examples to write to external object store: <ul style="list-style-type: none"> • Create a user with the required privileges • Create an authorization object 	Setting Up WRITE_NOS

These instructions are for Business Analysts and Data Scientists:

Follow the instructions in How Do I Get Started.

Focus your attention on the documentation for your data format: CSV, JSON, or Parquet.

How Do I Get Started?

You can run all the examples in the order given or use the examples specific to the task you want to perform.

You can test reading CSV, JSON, and Parquet examples by using the Teradata-supplied public object stores and sample data. You do not need to create your own object store for the sample data. The examples use a sample river flow data set. USGS Surface-Water Data Sets are provided courtesy of the U.S. Geological Survey.

Prerequisites

- A Vantage-supported client, such as Teradata Studio™ or BTEQ to run the examples and query the external object store.
- For WRITE_NOS, you need your own external object store on a Teradata-supported external storage platform. Data is written to your external storage by the WRITE_NOS examples.

Note:

Before proceeding, you need a user with the required access privileges. Depending on which examples you run, you may need a foreign table and an authorization object.

Operation	Examples
Read and load data from external object store	Examples: For Business Analysts Examples: For DBAs and Advanced Users
Write data to external object store	Writing Data to External Object Store

Additional Resources

Advanced SQL Engine	<ul style="list-style-type: none"> • <i>Native Object Store: Teradata Vantage™ Advanced SQL Engine</i>, TDN0009800 (Orange Book) <p>Note: Log in to https://docs.teradata.com/ to access it.</p> <ul style="list-style-type: none"> • <i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i>, B035-1144 • <i>Teradata Vantage™ - SQL Data Manipulation Language</i>, B035-1146
AWS	<ul style="list-style-type: none"> • <i>Teradata Vantage™ on AWS Getting Started Guide</i>, B700-4015 • <i>Teradata Vantage™ on AWS Marketplace Subscription Guide</i>, B035-2806
Azure	<ul style="list-style-type: none"> • <i>Teradata Vantage™ on Azure Getting Started Guide</i>, B700-4016

Google

• *Teradata Vantage™ on Google Cloud Getting Started Guide, B700-4020*

NOS Functionality

NOS enables you to do the following

- Analyze data stored on an external object storage
- Read data in CSV, JSON, or Parquet format from an external object storage
- Join or aggregate external data to relational data stored in Advanced SQL Engine
- Query cold data offloaded to an external object storage
- Load data from an external object storage into the database using one SQL request
- Write Advanced SQL Engine data to an external object storage. The data to be written can come from a table, derived results, another object store, QueryGrid federated query, and so on.

Reading data from an external object storage:

Foreign Tables

Users with CREATE TABLE privilege can create a foreign table inside the database, point this virtual table to an external storage location, and use SQL to translate the external data into a form useful for business.

Using a foreign table in Advanced SQL Engine gives you the ability to:

- Load external data to the database
- Join external data to data stored in the database
- Filter the data
- Use views to simplify how the data appears to your users
- Use Delta Lake manifest files

Data read through a foreign table is not automatically stored on disk and the data can only be seen by that query. Data can be loaded into the database by accessing a foreign table using these commands: CREATE TABLE AS ... WITH DATA, CREATE TABLE AS ... FROM READ_NOS, and INSERT ... SELECT.

READ_NOS

READ_NOS allows you to do the following:

- Perform an ad hoc query on all data formats with the data in-place on an external object storage
- List all the objects and path structure of an object storage
- List the object store
- Discover the schema of the data
- Read CSV, JSON, and Parquet data
- Bypass creating a foreign table in the Advanced SQL Engine

- Load data into the database with INSERT ... SELECT where the select references READ_NOS
- Use a foreign table to query data stored by READ_NOS
- Use Delta Lake manifest files

Writing data to an external object storage:

WRITE_NOS

WRITE_NOS allows you to write data from database tables to external object storage and store it in Parquet format. Data stored by WRITE_NOS can be queried using a foreign table and READ_NOS.

WRITE_NOS allows you to do the following:

- Extract selected or all columns from an Advanced SQL Engine table or from derived results and write to an external object storage in Parquet data format.
- Write to Teradata-supported external object storage, such as Amazon S3.
- Load data into the database with INSERT ... SELECT where the select references WRITE_NOS
- Use a foreign table to query data stored by WRITE_NOS

Supported External Object Storage Platforms

At the time of printing of this guide, the following external object storage platforms are supported:

- Amazon S3
- Microsoft Azure Blob storage
- Azure Data Lake Storage Gen2
- Google Cloud Storage
- Hitachi Content Platform
- MinIO
- Dell EMC/ECS
- NetApp StorageGRID
- IBM Cloud Object Store (IBM COS)
- Scality

Supported Compression Formats

External data may arrive from an object in a compressed format. If that is the case, the data will be decompressed inside the Advanced SQL Engine, but only after decryption has been completed on the object store before being transmitted. GZIP is the only compression format supported for both JSON and CSV. Snappy is supported for Parquet. The database recognizes the ".gz" suffix on the incoming files and performs the decompression automatically. Note, compression may bring some trade-offs, such as CPU overhead versus reduced needed Bandwidth amongst others.

Encryption

To encrypt files written to object store, configure the destination bucket to encrypt all objects using server-side encryption. Server-side encryption at the bucket level is supported by WRITE_NOS, READ_NOS, and foreign tables.

Note, all data is transmitted between the Vantage platform and the external object store using TLS encryption, independent of whether the data is encrypted at rest in the object store.

Command Syntax

The syntax and examples provided use a subset of the full syntax. To see the full syntax:

Command	See
CREATE FOREIGN TABLE	<i>Teradata Vantage™ - SQL Data Definition Language Syntax and Examples</i> , B035-1144
READ_NOS and WRITE_NOS	<i>Teradata Vantage™ - SQL Data Manipulation Language</i> , B035-1146

Changes and Additions

Date	Description
June 2022	<ul style="list-style-type: none"> Added support for Delta Lake. See Using Delta Lake Manifest Files. Reorganized content and fixed minor errors.
July 2021	<ul style="list-style-type: none"> NOS is enabled by default. NOS supports both Google S3-compatible and native connectors. See Authentication for External Object Stores. READ_NOS now supports Parquet data format. SQL syntax has been simplified for NOS and all examples have been updated accordingly: <ul style="list-style-type: none"> CREATE FOREIGN TABLE syntax is updated to include columns for each field in the CSV data instead of using a complex data type in the payload column. This eliminates the need to use dot notation and it is more efficient. Authorization examples are updated to use simplified system-wide authorization model new in this release. If you need information about the schema of your CSV, JSON, and Parquet data, you can use the NOSREAD_SCHEMA return type in your READ_NOS statement and the schema is detected automatically. The syntax is RETURNTYPE('NOSREAD_SCHEMA'). READ_NOS syntax has been simplified so that the SELECT statement does not require the READ_NOS table operator in the statement.
January 2021	Added WRITE_NOS feature.
September 2020	Google Cloud Storage is a supported external object store.

Setting Up Access

Variable Substitutions for Examples

The variables in the examples must be replaced with values that allow access to your external object store.

authorization_object

Replace *authorization_object* with the name of the authorization object. You may use any name for the authorization object that conforms to object naming rules, see *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

table_name

Replace *table_name* with a name for your foreign table that conforms to object naming rules, see *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

YOUR-OBJECT-STORE-URI

When you run the examples, replace *YOUR-OBJECT-STORE-URI* with the location of your object store

If you run the examples that use the Teradata-supplied public buckets or containers, the following table lists the paths to use to access the sample USGS river flow data. Replace *YOUR-OBJECT-STORE-URI* in the examples to read the data from the public buckets and containers:

Platform	Public Storage Location
Amazon S3	<ul style="list-style-type: none"> • CSV: /s3/td-usgs-public.s3.amazonaws.com/CSVDATA/ • JSON: /s3/td-usgs-public.s3.amazonaws.com/JSONDATA/ • Parquet: /s3/td-usgs-public.s3.amazonaws.com/PARQUETDATA/
Azure Blob storage	<ul style="list-style-type: none"> • CSV: /az/akiaox5jikeotfww4ul.blob.core.windows.net/td-usgs/CSVDATA/ • JSON: /az/akiaox5jikeotfww4ul.blob.core.windows.net/td-usgs/JSONDATA/ • Parquet: /az/akiaox5jikeotfww4ul.blob.core.windows.net/td-usgs/PARQUETDATA/
Google Cloud Storage	<ul style="list-style-type: none"> • CSV: /gs/storage.googleapis.com/td-usgs/CSVDATA/ • JSON: /gs/storage.googleapis.com/td-usgs/JSONDATA/ • Parquet: /gs/storage.googleapis.com/td-usgs/PARQUETDATA/

If you downloaded the USGS river flow data to your own external object store (see [Setting Up an Object Store for River Flow Data](#)), replace *YOUR-STORAGE-ACCOUNT* with the path to your data, which will be unique, but similar to the following:

Object Store	External Storage Location Values
Amazon S3 Note: The examples in this document use "td-usgs" for the bucket name. Your bucket name must be unique.	CSV: /s3/YOUR-BUCKET.s3.amazonaws.com/td-usgs/CSVDATA/ For example: /s3/td-usgs.s3.amazonaws.com/td-usgs/CSVDATA/
	JSON: /S3/YOUR-BUCKET.s3.amazonaws.com/td-usgs/JSONDATA/ For example: /s3/td-usgs.s3.amazonaws.com/td-usgs/JSONDATA/
	Parquet: /S3/YOUR-BUCKET.s3.amazonaws.com/td-usgs/PARQUETDATA/ For example: /s3/td-usgs.s3.amazonaws.com/td-usgs/PARQUETDATA/
Azure storage Note: Replace <i>YOUR-STORAGE-ACCOUNT</i> with the value for your account.	CSV: /az/YOUR-STORAGE-ACCOUNT.blob.core.windows.net/YOUR-CONTAINER/td-usgs/CSVDATA/ For example: /az/akiaxox5jikeotfww4ul.blob.core.windows.net/td-usgs/CSVDATA/
	JSON: /az/YOUR-STORAGE-ACCOUNT.blob.core.windows.net/YOUR-CONTAINER/td-usgs/JSONDATA/ For example: /az/akiaxox5jikeotfww4ul.blob.core.windows.net/YOUR-CONTAINER/td-usgs/JSONDATA/
	Parquet: /az/YOUR-STORAGE-ACCOUNT.blob.core.windows.net/YOUR-CONTAINER/td-usgs/PARQUETDATA/ For example: /az/akiaxox5jikeotfww4ul.blob.core.windows.net/YOUR-CONTAINER/td-usgs/PARQUETDATA/
Google Cloud Storage	CSV: /gs/storage.googleapis.com/YOUR-BUCKET/CSVDATA/ For example: /gs/storage.googleapis.com/td-usgs/CSVDATA/
	JSON: /gs/storage.googleapis.com/YOUR-BUCKET/JSONDATA/ For example: /gs/storage.googleapis.com/td-usgs/JSONDATA/
	Parquet: /gs/storage.googleapis.com/YOUR-BUCKET/PARQUETDATA/ For example: /gs/storage.googleapis.com/td-usgs/PARQUETDATA/

Note:

Teradata requires the storage location to start with the following:

- Amazon S3 bucket location must begin with /S3 or /s3
- Google Cloud Storage bucket location must begin with /GS or /gs
- Azure Blob storage location (including Azure Data Lake Storage Gen2 in Blob Interop Mode) must begin with /AZ or /az

YOUR-ACCESS-KEY-ID and YOUR-SECRET-ACCESS-KEY

Your external object store must be configured to allow Advanced SQL Engine to access it.

The USER and PASSWORD keywords (used in the CREATE AUTHORIZATION command) and ACCESS_ID and ACCESS_KEY (used by READ_NOS and WRITE_NOS) correspond to the values shown in the following table:

Keyword	Access Variables Values
USER or ACCESS_ID	Replace <i>YOUR-ACCESS-KEY-ID</i> with the access ID to your external storage.
PASSWORD or ACCESS_KEY	Replace <i>YOUR-SECRET-ACCESS-KEY</i> with the secret key value to your external storage.

Public buckets or containers in external object stores do not require credentials for access. To access a public bucket or container, put an empty string between the straight quotes for USER and PASSWORD.

Setting Up a Test User

You can run the Teradata-supplied examples running as your own database user or as a test user.

1. Create the test user called nos_usr:

```
CREATE USER nos_usr FROM dbc AS PERMANENT=30e8 PASSWORD=user_password;
```

Replace *user_password* with the password for the user.

Setting Access Privileges

To access data in the Advanced SQL Engine, the user must have certain privileges.

1. Log in as an administrative user and assign privileges depending on the operations the user is performing.

Privilege	Description	Command
CREATE TABLE	Allows user to create tables and foreign tables in the database	GRANT CREATE TABLE on <i>user</i> to <i>user</i> ;
EXECUTE FUNCTION on READ_NOS	Provides access to READ_NOS	GRANT EXECUTE FUNCTION on TD_SYSFNLIB.READ_NOS to <i>user</i> ;
EXECUTE FUNCTION on WRITE_NOS	Provides access to WRITE_NOS	GRANT EXECUTE FUNCTION on TD_SYSFNLIB.WRITE_NOS to <i>user</i> ;
CREATE AUTHORIZATION	Allows user to create an authorization object to make an association between the database user and operating system user. See Controlling Foreign Table Access with an AUTHORIZATION Object .	GRANT CREATE AUTHORIZATION on <i>user</i> to <i>user</i> ;

For example, issue the grants your user needs. Your users may only need a subset of the following grants, depending on the access they require:

```
GRANT CREATE TABLE on nos_usr to nos_usr;
GRANT EXECUTE FUNCTION on TD_SYSFNLIB.READ_NOS to nos_usr;
GRANT EXECUTE FUNCTION on TD_SYSFNLIB.WRITE_NOS to nos_usr;
GRANT CREATE AUTHORIZATION on nos_usr to nos_usr;
```

Controlling Foreign Table Access with an AUTHORIZATION Object

An authorization object is used to control who can access an external object store.

Before creating the authorization object Advanced SQL Engine must have permission from the external object store to access the data. The credentials are configured on the object store that you want to access. For example, to access an Amazon S3 bucket an Access Key ID or an AWS Identity and Access Management (IAM) user credential is required, depending on your external object store. See [Authentication for External Object Stores](#) for required credentials for the external object stores.

Once your external storage allows Advanced SQL Engine to access it, set up an authorization object with the credentials for your external object store.

Note:

Public buckets or containers in external object stores do not require credentials for access. To access a public bucket or container, put an empty string between the straight quotes for USER and PASSWORD.

Prerequisites

1. If not already done, log on to Advanced SQL Engine as an administrative user who can grant others privileges.
2. [Grant the appropriate privileges to the user.](#)
To create an authorization object, the user needs the following privileges:
 - CREATE AUTHORIZATION
3. Log off as the administrative user.

Create the Authorization Object

4. To run NOS-related commands, log on to the database as a user with the required privileges.
5. Create an authorization object in Advanced SQL Engine with the credentials to your external object store.

Note:

Create the authorization object in the same database as the foreign table that uses it.

```
CREATE AUTHORIZATION authorization_object
USER 'YOUR-ACCESS-KEY-ID'
PASSWORD 'YOUR-SECRET-ACCESS-KEY';
```

See [Variable Substitutions for Examples](#).

For example, the Teradata-supplied public object store has an empty string for USER and PASSWORD. To create an authorization object for the public object store, run:

```
CREATE AUTHORIZATION MyAuthObj
USER ''
PASSWORD '';
```

6. View the definition of the authorization object. For example:

```
SHOW AUTHORIZATION NOS_USR.MyAuthObj;
```

Note, the password is not returned in the result.

Postrequisite:

In later examples, the authorization object is linked to a foreign table and in other examples used in READ_NOS statements.

Running Examples Without an Authorization Object

If you want to test examples or test that your credentials work without creating an authorization object, you can use your access id and key in plain text in the command. Include your credentials in the USING clause of the READ_NOS statement in the ACCESS_ID and ACCESS_KEY keywords.

For example:

```
SELECT TOP 2 location(CHAR(100)), ObjectLength FROM READ_NOS (
USING
LOCATION ('YOUR-OBJECT-STORE-URI')
ACCESS_ID ('YOUR-ACCESS-KEY-ID')
ACCESS_KEY ('YOUR-SECRET-ACCESS-KEY')
RETURNTYPE ('NOSREAD_KEYS')
```

```
)
AS d;
```

Replace *YOUR-OBJECT-STORE-URI*, *YOUR-ACCESS-KEY-ID*, and *YOUR-SECRET-ACCESS-KEY* to access your own external storage and test that your credentials work. For information, see [Variable Substitutions for Examples](#).

For public buckets, *ACCESS_ID* and *ACCESS_KEY* are empty strings.

ObjectLength is the size of the external files in the specified *LOCATION* of the object store.

By default, the *RETURNTYPE* is *NOSREAD_KEYS*, which retrieves a list of files from the path specified by *LOCATION*. You do not need to specify the *RETURNTYPE* parameter if you want the default behavior.

Example: Running Examples Without an Authorization Object

The following example shows the *READ_NOS* statement without an authorization object. It accesses a Teradata-supplied public bucket.

For example:

```
SELECT TOP 2 * FROM (
  LOCATION='/s3/td-usgs-public.s3.amazonaws.com/CSVDATA/'
  ACCESS_ID=''
  ACCESS_KEY=''
  RETURNTYPE='NOSREAD_KEYS'
) AS d;
```

This example returns one row for each external file. The result lists the object length and location of each file.

For public buckets, *ACCESS_ID* and *ACCESS_KEY* are empty strings. If your bucket or container is not public, include the credentials inside the single quotes for each variable.

ObjectLength is the size of the external files in the specified *LOCATION* of the object store.

Reading Data

Sampling Data

Using NOS and SQL, you can sample data located on an external cloud server using either of the following methods:

- Performing ad hoc sampling of data
- Creating a foreign table structure inside the database that points to the external data, then querying the table

CSV Format

Historically, a CSV file is stored in a database as one row, requiring a LOB data type. With NOS, each CSV file record is a single row in the foreign table.

Using NOS, the maximum size of a record is 16,776,192 bytes.

- For UNICODE, that is equivalent to 8,388,096 characters.
- For LATIN, that is equivalent to 16,776,192 characters.

The actual record size is equivalent to the dataset individual record size.

NOS automatically discovers the schema of the data, including the data format, field delimiter, column names, column data types, and object store path definition.

If there are more columns than the number of fields in some of the records, then NULLs are returned for those columns. Similarly if there are less columns than the fields in some of the records, the extra fields are ignored.

For supported CSV formats, see [CSV External Files](#).

For instructions on working with CSV files without headers, see the Orange Book, *Native Object Store: Teradata Vantage™ Advanced SQL Engine*, TDN0009800.

JSON Format

Using NOS, the maximum payload size of a record is 16,776,192 bytes.

- For UNICODE, that is equivalent to 8,388,096 characters.
- For LATIN, that is equivalent to 16,776,192 characters.

The actual payload size is equivalent to the dataset individual record size.

Payload attributes for JSON objects are the names in the name:value pairs. In the example river flow data, some of the payload attributes include: datetime, Flow, Velocity, Precipitation, Temp, GageHeight.

When referring to a column name in JSON, use `payload.column_name`. For example, `payload.datetime`, `payload.Flow`, and so on.

Attributes may differ from one JSON object to another. JSON objects do not have to include all attributes.

For supported JSON formats, see [JSON External Files](#).

For more details, see the Orange Book, *Native Object Store: Teradata Vantage™ Advanced SQL Engine*, TDN0009800.

Parquet Format

When reading external Parquet data, the maximum Parquet page size supported is 16 MB.

Using NOS, the maximum record size is 16,776,192 bytes.

- If your record consists of all character data, these are the limitations for each character set:
 - For UNICODE, 16,776,192 bytes is equivalent to 8,388,096 characters.
 - For LATIN, 16,776,192 bytes is equivalent to 16,776,192 characters.

If some of data in the record is binary, the maximum number of characters is proportionately reduced.

Parquet tables don't have a payload column. The user creates a foreign table and maps the Parquet logical data type to the corresponding Teradata data type.

Redshift is the supported format for the manifest files.

Parquet format limitations:

- `READ_NOS` can be used to view the Parquet schema, using `RETURNTYPE('NOSREAD_SCHEMA')`. This is helpful in creating the foreign table when you do not know the schema of your Parquet data beforehand.
- Certain complex data types are not supported, including `STRUCT`, `MAP`, `LIST`, and `ENUM`.
- Because support for the `STRUCT` data type is not available, nested Parquet object stores cannot be processed by NOS.

The following examples use external Parquet data in this format:

```
message schema {
  optional double GageHeight2;
  optional double Flow;
  optional int64 site_no;
  optional binary datetime (UTF8);
  optional double Precipitation;
  optional double GageHeight;
}
```

For supported Parquet formats, see [Parquet External Files](#).

Setting Up to Run Examples

To use NOS, create a foreign table definition inside Advanced SQL Engine, point it at any external object store you are authorized to access, and then you can explore the data using all the analytics ability of Teradata Vantage. The NOS table definition is called a *foreign table*. A foreign table makes the external data available in a structured relational format, allowing it to be aggregated or joined to other relational tables inside the database.

Data read through a foreign server is not automatically stored in the database and the data can only be seen by that query. Data can be loaded into the database using the techniques shown in [Loading External Data into the Database](#).

The examples and results show a sample river flow data set. To use your own data, replace the table and column names, and authorization object.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. Create the foreign table or ask your database administrator to create the foreign table that is used by the Teradata-supplied examples. You can create it without specifying the data columns of the data (and let NOS discover the schema) or you can specify the data columns:

Create the foreign table and let NOS figure out the schema of the data:

```
CREATE FOREIGN TABLE table_name
, EXTERNAL SECURITY authorization_object
USING (LOCATION('YOUR-OBJECT-STORE-URI'));
```

See [Variable Substitutions for Examples](#) for the credentials and location values for the sample river flow data set.

Example: Setting Up to Run Examples

If the authorization object does not exist, create it or ask your database administrator to create it. See [Variable Substitutions for Examples](#).

```
CREATE FOREIGN TABLE riverflow
, EXTERNAL SECURITY MyAuthObj
USING (LOCATION('/s3/td-usgs-public.s3.amazonaws.com/CSVDATA/'));
```

For other data formats, the sample URIs are shown in the "YOUR-OBJECT-STORE-URI" section of [Variable Substitutions for Examples](#).

Examples: For Business Analysts

SELECT TOP *N* is used in many examples:

- TOP does not return the same results each time it is run.
- TOP may return records with only a subset of attributes because not all records in different objects contain all of the same attributes.

Sampling External Data In Place Using READ_NOS

Preview the contents of an external object store by using the READ_NOS table operator. In this example, the payload column contains the data.

The examples and results show a sample river flow data set. To use your own data, replace the table and column names, and authorization object.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. To see sample rows without column names:

```
SELECT TOP 2 * FROM (
  LOCATION='YOUR-OBJECT-STORE-URI'
  AUTHORIZATION=authorization_object
) AS D;
```

See [Variable Substitutions for Examples](#) for the credentials and location values for the sample river flow data set.

Example: Sampling External Data In Place Using READ_NOS

```
SELECT TOP 2 * FROM (
  LOCATION='/s3/td-usgs-public.s3.amazonaws.com/CSVDATA/'
  AUTHORIZATION=MyAuthObj
) AS D;
```

Your result will be similar to the following:

```
Location /S3/s3.amazonaws.com/td-usgs-public/CSVDATA/09429070/2018/07/
GageHeight2 ?
Flow ?
site_no 9429070
datetime 2018-07-02 14:35
Precipitation 1.16
GageHeight ?

Location /S3/s3.amazonaws.com/td-usgs-public/CSVDATA/09400815/2018/07/
```

```
GageHeight2      ?
      Flow      .00
      site_no    9400815
      datetime   2018-07-10 00:00
Precipitation     .00
      GageHeight -.01
```

The output is displayed vertically for readability.

Sampling External Data In Place Using a Foreign Table

Preview random rows of data in place from external object store using a foreign table.

The examples use a sample river flow data set. To use your own data, replace the table and column names, and authorization object. See [Variable Substitutions for Examples](#) for the credentials and location values for the sample data set.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. If it does not exist, create the foreign table or ask your database administrator to create the foreign table. See [Setting Up to Run Examples](#).
3. To see sample records with the attribute names included before each string of values, issue the following command against the foreign table:

```
SELECT TOP 2 *
FROM table_name;
```

Note:

If you run this on JSON data, the results look different because you get a payload. Inside the payload field, you get a comma delimited list of values.

On CSV and Parquet, you get a list of columns with data.

Example: Sampling External Data In Place Using a Foreign Table

If not already done, create the foreign table. See [Setting Up to Run Examples](#).

```
SELECT TOP 2 *
FROM riverflow;
```

Your results will be similar to the following:

```
Location /S3/s3.amazonaws.com/td-usgs-public/CSVDATA/09400815/2018/07/10.csv
GageHeight2      ?
```

```

      Flow      .00
      site_no    9400815
      datetime  2018-07-10 00:00
Precipitation    .00
GageHeight      -.01

      Location  /S3/s3.amazonaws.com/td-usgs-public/CSVDATA/09400815/2018/07/12.csv
GageHeight2     ?
      Flow      .00
      site_no    9400815
      datetime  2018-07-12 00:00
Precipitation    .00
GageHeight      -.01

```

The output is displayed vertically for readability.

Finding Maximum and Minimum Values

The examples use a sample river flow data set. To use your own data, replace the table and column names, and authorization object. See [Variable Substitutions for Examples](#) for the credentials and location values for the sample data set.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. If it does not exist, create the foreign table or ask your database administrator to create the foreign table. See [Setting Up to Run Examples](#).
3. Run the following command to get the maximum and minimum values for a specific payload attribute, Flow in this example. It executes against the foreign table and considers all records contained within the object as specified by the LOCATION clause in the foreign table.

```

SELECT MAX(columnN), MIN(columnN)
FROM table_name;

```

Example: Finding Maximum and Minimum Values

If not already done, create the foreign table. See [Setting Up to Run Examples](#).

```

SELECT MAX(Flow), MIN(Flow)
FROM riverflow;

```

Your result will be similar to the following:

```

Maximum(Flow)  Minimum(Flow)
-----
          9.91          -9.72

```

Querying for a Specific Set of Rows

The examples use a sample river flow data set. To use your own data, replace the table and column names, and authorization object. See [Variable Substitutions for Examples](#) for the credentials and location values for the sample data set.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. If it does not exist, create the foreign table or ask your database administrator to create the foreign table. See [Setting Up to Run Examples](#).
3. Query for a set of rows by specifying a specific value in the WHERE clause. This requests that only records with that value be returned for viewing.

```
SELECT MAX(columnX)
FROM table_name
WHERE columnN = 'value';
```

Example: Querying for a Specific Set of Rows

Specify a specific value for the site_no attribute in the WHERE clause. In this example, the max value for the Flow attribute for a single site_no is returned.

If not already done, create the foreign table. See [Setting Up to Run Examples](#).

```
SELECT MAX(Flow)
FROM riverflow
WHERE site_no = '09394500';
```

Your result will be similar to the following:

```
Maximum(Flow)
-----
          9.91
```

Using NOSREAD_KEYS to List the Files in the Specified LOCATION

NOSREAD_KEYS retrieves a list of files from the path specified by LOCATION.

```
SELECT TOP 2 location(CHAR(200)), ObjectLength FROM (
LOCATION='YOUR-OBJECT-STORE-URI'
```

```
AUTHORIZATION=authorization_object
RETURNRTYPE='NOSREAD_KEYS'
) AS d;
```

ObjectLength is the size of the external files in the specified LOCATION of the object store.

Example: Using NOSREAD_KEYS to List the Files in the Specified LOCATION

```
SELECT TOP 2 location(CHAR(200)), ObjectLength FROM (
LOCATION='/s3/td-usgs-public.s3.amazonaws.com/CSVDATA/'
AUTHORIZATION=MyAuthObj
RETURNRTYPE='NOSREAD_KEYS'
) AS d;
```

The example accesses the Teradata-supplied public bucket.

Your result will be similar to the following:

```
Location  /S3/s3.amazonaws.com/td-usgs-public/CSVDATA/09380000/2018/06/30.csv
ObjectLength  5009

Location  /S3/s3.amazonaws.com/td-usgs-public/CSVDATA/09380000/2018/06/29.csv
ObjectLength  5088
```

Examples: For DBAs and Advanced Users

SELECT TOP *N* is used in many examples:

- TOP does not return the same results each time it is run.
- TOP may return records with only a subset of attributes because not all records in different objects contain all of the same attributes.

Loading External Data into the Database

The methods for loading external data into the database are as follows:

Method	Description
CREATE TABLE AS ... WITH DATA	Accesses table definitions and data from an existing foreign table and creates a new permanent table inside the database
CREATE TABLE AS ... FROM READ_NOS	Accesses data directly from the object store and creates a permanent table inside the database
INSERT ... SELECT	Stores values from external data in a persistent database table using an INSERT ... SELECT statement

Loading External Data into the Database Using CREATE TABLE AS ... WITH DATA

If you access the same external data often, copy the data inside the database. To do this, copy the table definitions and data from the existing foreign table to a new target table inside the database using a CREATE TABLE AS ... WITH DATA command.

The examples use a sample river flow data set. To use your own data, replace the table and column names, and authorization object. See [Variable Substitutions for Examples](#) for the credentials and location values for the sample data set.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. If the foreign table does not exist, create it or ask your database administrator to create it. See [Setting Up to Run Examples](#).
3. Create a table in the database to load a subset of the external data that you are interested in analyzing:

```
CREATE MULTISET TABLE multiset_table_name AS (
  SELECT columnX, columnY, columnN
  FROM table_name
  WHERE columnN > value )
WITH DATA
NO PRIMARY INDEX;
```

Example: Loading External Data into the Database Using CREATE TABLE AS ... WITH DATA

If not already done, create the foreign table that points to external data to be loaded. See [Setting Up to Run Examples](#).

Create a table in the database to load a subset of external data to analyze.

```
CREATE MULTISET TABLE riverflowprecip AS (
  SELECT site_no, Flow, GageHeight
  FROM riverflow
  WHERE Precipitation > 0 )
WITH DATA
NO PRIMARY INDEX;
```

Show the number of rows in the table:

```
SELECT COUNT(*) FROM riverflowprecip;
```

Your result will be similar to the following:

```
Count(*)
-----
      291
```

Loading External Data into the Database Using READ_NOS and CREATE TABLE AS

READ_NOS combined with a CREATE TABLE AS clause accesses data from the object store and creates a permanent table for the data inside the database, without needing to create a foreign table first.

Note:

READ_NOS does not need to be specified in the command. The SELECT statement invokes it.

In this example, all the data in the object store identified by the LOCATION is loaded into the database. The examples and results show a sample river flow data set. To use your own data, replace the table and column names, and authorization object.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. Create and load the table:

```
CREATE MULTISET TABLE multiset_table_name AS (
  SELECT columnX, columnY, ...columnN
  FROM (
    LOCATION='YOUR-OBJECT-STORE-URI'
    AUTHORIZATION=authorization_object
  ) AS d
  ) WITH DATA;
```

See [Variable Substitutions for Examples](#) for the credentials and location values for the sample river flow data set.

Example: Loading External Data into the Database Using READ_NOS and CREATE TABLE AS

Create and load the table:

```
CREATE MULTISET TABLE riverflowperm AS (
  SELECT site_no, Flow, GageHeight, Precipitation, datetime, GageHeight2
  FROM (
    LOCATION='/s3/td-usgs-public.s3.amazonaws.com/CSVDATA/'
    AUTHORIZATION=MyAuthObj
  ) AS d
) WITH DATA;
```

Show the number of rows in the table:

```
SELECT COUNT (*) FROM riverflowperm;
```

Your result will be similar to the following:

```
Count(*)
-----
17058
```

Loading External Data into the Database Using INSERT ... SELECT

This example illustrates how a single siteno value can be stored in a persistent database table using an INSERT ... SELECT statement.

The examples use a sample river flow data set. To use your own data, replace the table and column names, and authorization object. See [Variable Substitutions for Examples](#) for the credentials and location values for the sample data set.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. If it does not exist, create the foreign table or ask your database administrator to create the foreign table. See [Setting Up to Run Examples](#).
3. In this example, data from a single site number is loaded into the database. Create the permanent table to store the external data:

```
CREATE TABLE perm_table_name (
  columnX data_type
  ,columnY data_type
  ,columnZ data_type
  ,columnN data_type )
PRIMARY INDEX (columnN);
```

4. Insert the external data into the database table:

```
INSERT INTO perm_table_name
SELECT columnX, columnY, ...columnN
WHERE columnN = value
FROM table_name;
```

Example: Loading External Data into the Database Using INSERT ... SELECT

Create the permanent table to store external data:

```
CREATE TABLE RiverFlowPermInsert (
SiteNo INTEGER
,Flow DECIMAL(3,2)
,GageHeight DECIMAL(3,2)
,Precipitation DECIMAL(3,2)
,datetime TIMESTAMP(0) FORMAT'Y4-MM-DDBHH:MI'
,GageHeight2 DECIMAL(3,2) )
PRIMARY INDEX (SiteNo);
```

Insert the external data FROM the foreign table into the permanent database table. If the foreign table does not exist, create it first. See [Setting Up to Run Examples](#).

```
INSERT INTO RiverFlowPermInsert
SELECT site_no, Flow, GageHeight, Precipitation, datetime, GageHeight2
WHERE site_no = 9474000
FROM riverflow;
```

Query the data from the database table:

```
SELECT TOP 2 * FROM RiverFlowPermInsert;
```

Your result will be similar to the following:

```
SiteNo      9474000
Flow        .80
GageHeight   3.51
Precipitation .00
datetime    2018-07-26 01:00
GageHeight2  6.66

SiteNo      9474000
Flow        .80
GageHeight   3.51
Precipitation .00
datetime    2018-07-26 00:30
GageHeight2  6.66
```

The output is displayed vertically for readability.

Joining External Data and Database Tables

In the following example, create and populate a database table, create a foreign table, and join the two tables.

The examples use a sample river flow data set. To use your own data, replace the table and column names, and authorization object. See [Variable Substitutions for Examples](#) for the credentials and location values for the sample data set.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

Prerequisite

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. If the rivernames table already exists, skip the rest of the steps in the Prerequisite section.
3. Create the database table:

Note:

In the absence of a local database table to represent the database side of the object store-to-database join, a simulation is required. Typically, you would join to an existing table that is already in your environment and these simulation steps would not be needed.

As a solution a small dimension table has been placed in an external object store to:

- Make the join practical
 - Ensure that all users trying this example have the same version of the table data
-

```
CREATE MULTISET TABLE rivernames (
  site_no INT,
  name CHAR(90) CHARACTER SET LATIN NOT CASESPECIFIC )
UNIQUE PRIMARY INDEX ( site_no );
```

4. Create the foreign table or ask your database administrator to create the foreign table that is used to populate the dimension table:

```
CREATE FOREIGN TABLE nos_rivernames
, EXTERNAL SECURITY MyAuthObj
USING ( LOCATION('/s3/td-usgs-public.s3.amazonaws.com/RIVERS/rivers.csv') );
```

Replace LOCATION with your path to RIVERS/rivers.csv.

5. Populate the dimension table:

```
INSERT INTO rivernames
SELECT site_no, name
FROM nos_rivernames;
```

Join the Dimension Table and External Data

6. If it does not exist, create the foreign table or ask your database administrator to create the foreign table. See [Setting Up to Run Examples](#).
7. Join the dimension and foreign tables:

```
SELECT DISTINCT name(CHAR(100))
FROM riverflow rf, rivernames rn
WHERE rf.site_no = rn.site_no
AND rf.Precipitation > 0.1
ORDER BY 1;
```

Your result will be similar to the following:

```
name
-----
CIBECUE CREEK NEAR OVERGAARD
CRIR LWR MAIN DRAIN BLW TYSON WW
GILA RIVER AT KELVIN
LITTLE COLORADO RIVER AT WOODRUFF
NEW RIVER NEAR ROCK SPRINGS
NEWMAN CANYON ABOVE UPPER LAKE MARY
POLACCA WASH NEAR SECOND MESA
```

Using Filtering to Retrieve Data More Efficiently

Reading all the objects in a bucket can be time-consuming and expensive. An alternative solution is to filter the object store using one or both of the following filtering methods:

Path Filtering

- A path is a series of values that constitute a multi-level key (for example, site_no, year, month, day) which identifies one or more objects within an object store.
- Path filtering lets you filter and reduce the amount of objects you bring back to the database by providing specified values for different levels in a path key (for example, WHERE \$path.\$site_no='09380000')
- Path filtering is the most efficient way to read a selected portion on object store.

In path filtering, by default the folders are referred to as \$var1, \$var2, and so on.

Note:

For CSV data, we attempt to use the column name, if it is the same as the folder name, instead of the generic \$varX variable, for example, WHERE \$path.\$site_no='09380000'.

In the example we show the generic syntax with the generic PATHPATTERN.

td-usgs123	-- Unknown
CSVDATA	-- Unknown
09380000	-- Unknown
2018	-- Unknown
06	-- Unknown
27.csv	5.0 KiB 6/24/2020 9:
28.csv	5.0 KiB 6/24/2020 9:
29.csv	5.0 KiB 6/24/2020 9:
30.csv	4.9 KiB 6/24/2020 9:
> 07	-- Unknown
> 09394500	-- Unknown
> 09396100	-- Unknown

You create a foreign table against this structure like this:

```
CREATE FOREIGN TABLE table_name
, EXTERNAL SECURITY authorization_object
USING (
LOCATION(' YOUR-OBJECT-STORE-URI ')
PATHPATTERN(' $var1/$var2/$var3/$var4/$var5 ' )
);
```

Note:

PATHPATTERN is optional. As an example, the PATHPATTERN in the CREATE FOREIGN statement matches the structure of the object store shown in the diagram.

You can look at the table definition with SHOW TABLE:

```
SHOW TABLE table_name;
```

The table definition for the object store folder in the figure shows that the variables point to specific folders and files in the folder:

- \$var1 points to the CSVDATA folder
- \$var2 points to the 09380000 folder
- \$var3 points to the 2018 folder
- \$var4 points to the 06 folder

- \$var5 points to the files, such as 27.csv, 28.csv, and so on

You can change \$var names in the foreign table definition with PATHPATTERN, if you know the structure of your data.

For example:

```
CREATE FOREIGN TABLE table_name
, EXTERNAL SECURITY authorization_object
USING (
LOCATION(' YOUR-OBJECT-STORE-URI ')
PATHPATTERN(' $datatype/$site_no/$year/$month/$filename ')
);
```

If you defined the table with meaningful names, you use those names instead of \$varX when referring to a \$path. for example, see [Using Path Variables as Columns in a View](#).

Payload Filtering

- The payload column within a foreign table contains all of the values within an object.
- In payload filtering you apply a filter to the data itself and not to the partitioning (path key) of the object store.
- When filtering is applied to a column in the data (such as WHERE Flow > '15'), all objects in the object store have to be brought inside the database and be transformed prior to applying filtering.
- When given a choice, this is usually less efficient because it will incur greater effort at both the object store level and the database level than does path filtering.

Filtering Using a Path Filter in the CREATE FOREIGN TABLE Definition

When a subset of objects in an object store have been identified for future attention, Teradata recommends using a path filtering value based on the folder values in the definition of the foreign table. The LOCATION clause in the foreign table definition can specify specific values at one or more key structure levels to limit the objects that are returned. However, each time a different filter is required, either an additional foreign table must be created, or the previous table must be dropped and recreated. To avoid that issue, you can use the PATHPATTERN clause as part of the CREATE FOREIGN TABLE.

In the example, filtering is expressed in the LOCATION clause by *value*.

The examples and results show a sample river flow data set. To use your own data, replace the table and column names, and authorization object.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. Create a foreign table or ask your database administrator to create the foreign table specifying a LOCATION that filters on one of the PATHPATTERN variables:

```
CREATE FOREIGN TABLE table_name
, EXTERNAL SECURITY authorization_object
USING (
LOCATION ('YOUR-OBJECT-STORE-URI/')
PATHPATTERN('$var1, $var2, $var3, $var4, $var5') );
```

See [Variable Substitutions for Examples](#) for the credentials and location values for the sample river flow data set.

You can use SHOW TABLE *table_name* to see the table definition.

Example: Filtering Using a Path Filter in the CREATE FOREIGN TABLE Definition

In the example, filtering is expressed in the LOCATION clause. Only data from river sites that match the site_no (for example, that begin with "0938") will be returned through this foreign table definition.

If not already created, create the authorization object or ask your database administrator to create it. Use the authorization object created in [Controlling Foreign Table Access with an AUTHORIZATION Object](#).

```
CREATE FOREIGN TABLE riverflow_pathfilter
, EXTERNAL SECURITY MyAuthObj
USING (
LOCATION ('/s3/td-usgs-public.s3.amazonaws.com/CSVDATA/')
PATHPATTERN('$data/$site_no/$year/$month/$day') );
```

The site_no is used to filter the data.

Show the table definition:

```
SHOW TABLE riverflow_pathfilter;
```

Your result will be similar to the following:

```
CREATE MULTISET FOREIGN TABLE NOS_USR.riverflow_pathfilter ,FALLBACK ,
EXTERNAL SECURITY NOS_USR.MYAUTHOBJ ,
MAP = TD_MAP1
(
Location VARCHAR(2048) CHARACTER SET UNICODE CASESPECIFIC,
Temp DECIMAL(3,1),
```

```

Flow SMALLINT,
site_no INTEGER,
datetime TIMESTAMP(0) FORMAT 'Y4-MM-DDBHH:MI',
Conductance SMALLINT,
Precipitation DECIMAL(3,2),
GageHeight DECIMAL(4,2))
USING
(
  LOCATION ('/s3/td-usgs-public.s3.amazonaws.com/CSVDATA/')
  PATHPATTERN ('$data/$site_no/$year/$month/$day')
  MANIFEST ('FALSE')
  ROWFORMAT
  ('{"field_delimiter":",","record_delimiter":"\n","character_set":"LATIN"}')
  STOREDAS ('TEXTFILE')
  HEADER ('TRUE')
  STRIP_EXTERIOR_SPACES ('FALSE')
  STRIP_ENCLOSING_CHAR ('NONE')
)
NO PRIMARY INDEX ;

```

Filtering Using a Column Within the Data Set

If a query filters on a column value, the database reads all the objects in the foreign table, transforms them, and examines the individual rows in order to apply the WHERE clause criteria.

Path filtering reduces the number of external objects read. You can view the query log, if it is enabled, to see the amount of external objects that are accessed. There is less data transferred with the path filtering query.

For details on the NOS-related DBQL fields, see the Orange Book, *Native Object Store: Teradata Vantage™ Advanced SQL Engine*, TDN0009800 and *Teradata Vantage™ - Data Dictionary*, B035-1092.

The examples and results show a sample river flow data set. To use your own data, replace the table and column names, and authorization object.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. Use the WHERE clause to filter on site_no:

```

SELECT TOP 2 columnX, columnY
FROM table_name
WHERE columnN = value;

```

See [Variable Substitutions for Examples](#) for the credentials and location values for the sample river flow data set.

Example: Filtering Using a Column Within the Data Set

If not already done, create the foreign table. See [Filtering Using a Path Filter in the CREATE FOREIGN TABLE Definition](#).

```
SELECT TOP 2 GageHeight, Flow
FROM riverflow_pathfilter
WHERE site_no = 09396100;
```

Your result will be similar to the following:

GageHeight	Flow
1.12	9.54
0.49	0

Path Filtering Using a Key in the Path of the Data Set

Key (or path) filtering reduces the number of external objects that are transferred to provide the result. This reduces the I/O performed on the external object store time and results in less work for the SQL Engine.

The examples and results show a sample river flow data set. To use your own data, replace the table and column names, and authorization object.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. Create a foreign table or ask your database administrator to create the foreign table specifying a LOCATION that filters on site_no. Use the authorization object created in [Controlling Foreign Table Access with an AUTHORIZATION Object](#):

```
SELECT TOP 2 columnX, columnY
FROM table_name
WHERE $path.columnN = value;
```

Example: Path Filtering Using a Key in the Path of the Data Set

If not already done, create the foreign table. See [Filtering Using a Path Filter in the CREATE FOREIGN TABLE Definition](#).

```
SELECT TOP 2 GageHeight, Flow
FROM riverflow_pathfilter
WHERE $path.$site_no = 09396100;
```

Your result will be similar to the following:

GageHeight	Flow
1.06	6.82
0.49	0

Using Views and Path Filtering

There are specific benefits to creating foreign table views, in addition to the usual reasons of security and hiding SQL complexity. These benefits include:

- Makes it easier to query object store data with inconsistent uppercase and lowercase field names. CSV field names are case sensitive, but Teradata SQL is case insensitive.
- You can hide the \$path references in a foreign table and make path names look like standard column names.

Follow these recommendations when creating a view on a foreign table:

- Include both path names and payload attributes as columns in the view.
- CAST all values in the view to an appropriate data type.
- It is often the case that the path segment name (such as site_no in data / site_no / year / month / day) is also represented by an attribute in your data. When this is true, Teradata recommends you equate them in the WHERE expression in the view and only specify one of them in the column list.

Creating a Basic View

Creating a view allows you to:

- CAST the payload fields to an appropriate data type.
- Rename the header fields using case insensitive naming. Although the example shows the headers in mixed case, you can use whatever case you want in queries.

The examples use a sample river flow data set. To use your own data, replace the table and column names, and authorization object. See [Variable Substitutions for Examples](#) for the credentials and location values for the sample data set.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. If the foreign table that the new view will represent does not exist, create it or ask your database administrator to create it. See [Filtering Using a Path Filter in the CREATE FOREIGN TABLE Definition](#).

3. Create the view on the foreign table:

```
CREATE VIEW view_name AS (
SELECT
  columnX,
  columnY,
  columnZ,
  columnN
FROM table_name );
```

Example: Creating a Basic View

Create the view of the foreign table.

If not already done, create the foreign table that the new view will represent. See [Filtering Using a Path Filter in the CREATE FOREIGN TABLE Definition](#).

```
CREATE VIEW riverflowview AS (
SELECT
  Flow,
  GageHeight,
  Precipitation,
  Site_no
FROM riverflow_pathfilter );
```

Query the view:

```
SELECT TOP 2 * FROM riverflowview;
```

Your result will be similar to the following:

Flow	GageHeight	Precipitation	site_no
17500	10.21	.00	09396100
17700	10.25	.00	09396100

Columns with missing attributes return NULLs when the record does not contain the attribute.

Using Path Variables as Columns in a View

By creating a view, path expressions (such as \$path.\$site_no) become columns in the view allowing the user to specify WHERE clauses on those columns without being aware that the underlying table contains complex path expressions.

The names given to the columns in the view are case insensitive. Although shown in mixed case, they can be referenced in SQL queries using any mix of case desired.

The examples use a sample river flow data set. To use your own data, replace the table and column names, and authorization object. See [Variable Substitutions for Examples](#) for the credentials and location values for the sample data set.

Note:

[Before proceeding, verify with your database administrator that you have the correct privileges and an authorization object.](#)

1. To run NOS-related commands, log on to the database as a user with the required privileges.
2. If the foreign table that the new view will represent does not exist, create it or ask your database administrator to create it. See [Filtering Using a Path Filter in the CREATE FOREIGN TABLE Definition](#).
3. Create the view of the foreign table:

```
REPLACE VIEW view_name AS (
SELECT
  CAST($path.$columnX AS data_type) columnX_alias,
  CAST($path.$columnY AS data_type) columnY_alias,
  CAST(SUBSTR($path.$columnZ, X, Y) AS data_type) columnZ_alias,
  columnN
FROM table_name;
```

Example: Using Path Variables as Columns in a View

If not already done, create the view that is getting replaced. See [Creating a Basic View](#).

If not already done, create the foreign table that the new view will represent. See [Filtering Using a Path Filter in the CREATE FOREIGN TABLE Definition](#).

```
REPLACE VIEW riverflowview AS (
SELECT
  CAST($path.$site_no AS CHAR(10)) TheSite,
  CAST($path.$year AS CHAR(4)) TheYear,
  CAST($path.$month AS CHAR(2)) TheMonth,
  CAST(SUBSTR($path.$day, 1, 2) AS CHAR(2)) TheDay,
  Flow,
  GageHeight,
  Precipitation
FROM riverflow_pathfilter);
```

Query the view:

```
SELECT TOP 2 * FROM riverflowview;
```

Your result will be similar to the following:

TheSite	TheYear	TheMonth	TheDay	Flow	GageHeight	Precipitation
09396100	2018	07	18	17500	10.21	.00
09396100	2018	07	24	17700	10.25	.00

Columns with missing attributes return NULLs when the record does not contain the attribute.

Example: Run EXPLAIN on the Query to See How the Filtering is Done

```
EXPLAIN
SELECT thesite,COUNT(*)
FROM riverflowview WHERE thesite=09396100'
GROUP BY 1;
```

Your result will be similar to the following:

```
Explanation
-----
[...]
```

- 2) Next, we do a single-AMP RETRIEVE step from NOS_USR.riverflow_pathfilter in view riverflowview metadata by way of an all-rows scan with a condition of ("(TD_SYSFNLIB.NosExtractVarFromPath (NOS_USR.riverflow_pathfilter in view riverflowview.Location, '/s3/td-usgs-public.s3.amazonaws.com', 2)(CHAR(10), CHARACTER SET UNICODE, NOT CASESPECIFIC))= '09396100 '") into Spool 2 (one-amp), which is built locally on that AMP. Then we do a SORT to order Spool 2 by the sort key. The size of Spool 2 is estimated with no confidence to be 4 rows (2,820 bytes). The estimated time for this step is 0.55 seconds.
- 3) We do a single-AMP RETRIEVE step from Spool 2 (Last Use) by way of an all-rows scan into Spool 5 (all_amps), which is binpacked and redistributed by size to all AMPs in TD_Map1. The size of Spool 5 is estimated with no confidence to be 4 rows (2,852 bytes). The estimated time for this step is 0.06 seconds.
- 4) We do an all-AMPs SUM step in TD_MAP1 to aggregate from NOS_USR.riverflow_pathfilter in view riverflowview by way of an object-store scan using Spool 5 (Last Use) with a condition of ("(TD_SYSFNLIB.NosExtractVarFromPath (NOS_USR.riverflow_pathfilter in view riverflowview.Location, '/s3/td-usgs-public.s3.amazonaws.com', 2)(CHAR(10), CHARACTER SET UNICODE, NOT CASESPECIFIC)(FLOAT, FORMAT '-9.999999999999999E-999'))= (NOS_USR.riverflow_pathfilter in view riverflowview.site_no)"), and the grouping identifier in field 1. Aggregate intermediate results are computed globally, then placed in Spool 4 in TD_Map1. The size of Spool 4 is estimated with no confidence to be 222 rows (311,244 bytes). The estimated time for this step is 2.18 seconds.

```
[...]
```

(2) is doing the path filtering. It is using the constant 09396100 as a path filtering expression to build the metadata spool. The metadata spool is the spool table that identifies the list of objects the query will actually process.

(4) is doing traditional row filtering. It compares the site number extracted from the location string to the value in the actual data in the object store.

Writing Data to External Object Store

WRITE_NOS

WRITE_NOS allows you to extract selected or all columns from a database table or from derived results and write to external object storage, such as Amazon S3, Azure Blob storage, Azure Data Lake Storage Gen2, and Google Cloud Storage.

WRITE_NOS stores data in Parquet format.

Data written to external object storage can be queried using a foreign table and READ_NOS.

You can use WRITE_NOS to transform CSV and JSON data without having to write it locally to a relational table. Read the CSV or JSON data using CREATE FOREIGN TABLE or READ_NOS, then use WRITE_NOS to write the data in Parquet format.

WRITE_NOS can write relational data to external object store, which you can partition in an optimal way for reading back based on expected query usage.

The maximum input data for both READ_NOS and WRITE_NOS is 16,776,192 bytes (including LOB and LOB-based UDT columns).

It is the responsibility of the user to clean up object store files on interrupted write operations. Write operations can be interrupted on transaction aborts or system resets, among other reasons.

Concurrent WRITE_NOS requests to the same location are likely to cause an error to be returned. If a request gets an error, any objects written by the request are not deleted from external storage and must be manually cleaned up using tools provided by the external object store vendor, such as s3cmd. To avoid this, Teradata recommends specifying a unique location for concurrent requests.

An error is reported if you attempt to write an object with same path name in the same location.

Setting Up WRITE_NOS

Example: Using WRITE_NOS with AUTHORIZATION

The example uses the AUTHORIZATION keyword with an authorization object to provide credentials to external object store.

Prerequisites

1. Set up or obtain access to the external object store where you want to write data. Follow the instructions from your external storage vendor.
2. If not already done, have an administrative user create the authorization object that has the credential to the bucket. See [Controlling Foreign Table Access with an AUTHORIZATION Object](#).

Use AUTHORIZATION Keyword

3. Use WRITE_NOS with the AUTHORIZATION keyword:

```
SELECT * FROM WRITE_NOS (
ON ( SELECT * FROM RiverFlowPerm WHERE DateTime = '2018-06-27
00:00:00' ) USING
LOCATION('/S3/s3.amazonaws.com/iewritenostest/RiverFlowPerm2/')
AUTHORIZATION(MyAuthObj)
STOREDAS('PARQUET')
) AS d;
```

Replace LOCATION with the URI to the external object store where you are writing the data.

Your result will be similar to the following:

NodeId	AmpId	Sequence	ObjectName
33	0	1	/S3/s3.amazonaws.com/ie-writenos-bucket/20180701_auth_example/object_33_0_1.parquet
33	1	1	/S3/s3.amazonaws.com/ie-writenos-bucket/20180701_auth_example/object_33_1_1.parquet
33	2	1	/S3/s3.amazonaws.com/ie-writenos-bucket/20180701_auth_example/object_33_2_1.parquet

Writing All Vantage Data to External Object Storage

The following examples show how to use WRITE_NOS to write all data from an Advanced SQL Engine table to Teradata-supported external object storage.

Examples: Write All Rows from a Table to External Object Storage Using Partitioning

The following examples write all the data stored in a database table to external object storage. The data is partitioned (sorted) by Advanced SQL Engine before being written. The objects containing the data in external storage are named to reflect the sorted data they contain.

Prerequisites

1. If not already done, set up or obtain access to the external object storage where you want to write data. Follow the instructions from your external storage vendor.

Example: Write All Rows to External Object Storage, Use PARTITION BY and NAMING('DISCRETE')

2. Write all the rows from a database table to external object storage:

```
SELECT * FROM WRITE_NOS (
ON ( SELECT * FROM RiverFlowPerm )
PARTITION BY site_no ORDER BY site_no
USING
LOCATION('YOUR-OBJECT-STORE-URI/RiverFlowPerm_Discrete/')
AUTHORIZATION(MyAuthObj)
STOREDAS('PARQUET')
```

```

COMPRESSION('SNAPPY')
NAMING('DISCRETE')
INCLUDE_ORDERING('TRUE')
) AS d;

```

Replace LOCATION with the URI to the external object storage where you want to write the data.

Things to note in this example:

- Each row of the result set describes one object written to external storage that contains data from Vantage.
- Each object contains data from one partition created according to the PARTITION BY clause. In this case there is one partition for each data collection site, that is, for each SiteNo column value.
- NAMING('DISCRETE') causes the object names to include the partition name, in this case the object names include the values for SiteNo. In the object name, /S3/s3.amazonaws.com/ie-writenos-bucket/RiverFlowPerm_Discrete/**09497500**/object_33_0_1.parquet, "09497500" is the SiteNo of the data contained by this object.
- Because the query included INCLUDE_ORDERING('TRUE'), the objects include data from the ORDER BY column(s), in this case the object data includes the site number from which the data in that row was collected.

Your result will be similar to the following:

```

      NodeId      33
      AmpId       0
      Sequence    1
      ObjectName  /S3/s3.amazonaws.com/ie-writenos-bucket/RiverFlowPerm_Discrete/
09497500/object_33_0_1.parquet
      ObjectSize      27682
      RecordCount    2944

      NodeId      33
      AmpId       0
      Sequence    1
      ObjectName  /S3/s3.amazonaws.com/ie-writenos-bucket/RiverFlowPerm_Discrete/
09513780/object_33_0_1.parquet
      ObjectSize      19136
      RecordCount    2941

      NodeId      33
      AmpId       0
      Sequence    1
      ObjectName  /S3/s3.amazonaws.com/ie-writenos-bucket/RiverFlowPerm_Discrete/
09424900/object_33_0_1.parquet
      ObjectSize      23174

```

```
RecordCount          2946
[...]
```

The output is displayed vertically for readability.

Working with Manifest Files

A manifest file is created when you write objects to external object store. The manifest file lists the paths of the objects stored on external object store.

- The files that are specified in the manifest can be in different buckets, but all the buckets must be in the same region.
- Manifest files are in JSON format.
- The storage location for manifest files must start with the following:
 - Amazon S3 location must begin with /S3 or /s3, for example `/S3/YOUR-BUCKET.s3.amazonaws.com/20180701/ManifestFile2/manifest1.json`
 - Google Cloud Storage location must begin with /GS or /gs, for example `/gs/storage.googleapis.com/YOUR-BUCKET/JSONDATA/manifest1.json`
 - Azure Blob location (including Azure Data Lake Storage Gen2 in Blob Interop Mode) must begin with /AZ or /az, for example `/az/YOUR-STORAGE-ACCOUNT.blob.core.windows.net/td-usgs/JSONDATA/manifest1.json`
- Manifest files are not cumulative. If you want to add entries to a manifest, you must create a new manifest that includes the original entries plus the ones you want to add.
- An error is reported if you attempt to write another manifest file in the same location. Use `OVERWRITE('TRUE')` with `MANIFESTONLY('TRUE')` keywords to replace a manifest in the same location.

Creating a Manifest File of the Actual Objects Written by WRITE_NOS

When Write_NOS is done creating objects on external object storage it can create a manifest file with the names of the objects written by WRITE_NOS. This example shows how to use the MANIFESTFILE option of WRITE_NOS to write data from an Advanced SQL Engine table to a manifest file on external object storage.

Prerequisites

1. If not already done, set up an authorization to access external object storage that has the credential to the bucket. See [Controlling Foreign Table Access with an AUTHORIZATION Object](#).
2. If not already done, set up or obtain access to the external object storage where you want to write data. Follow the instructions from your external storage vendor.

Example: Using the MANIFESTFILE Option

Create a manifest file listing the paths of the objects stored on external storage using the MANIFESTFILE option. MANIFESTFILE creates a manifest file on external object store at the location specified.

1. Write all rows matching the criteria to external object store and create a manifest file listing the paths of the objects stored on external storage:

```
SELECT * FROM WRITE_NOS (
ON ( SELECT * FROM RiverFlowPerm WHERE DateTime = '2018-06-27
00:00:00' ) USING
LOCATION(' YOUR-OBJECT-STORE-URI/80627/ManifestFile/')
AUTHORIZATION(MyAuthObj)
STOREDAS('PARQUET')
MANIFESTFILE(' YOUR-OBJECT-STORE-URI/20180627/ManifestFile/manifest1.json')
) AS d;
```

Replace the LOCATION of *YOUR-OBJECT-STORE-URI/20180701/ManifestFile/* with the URI to the external object store where you want to write the data.

Replace the MANIFESTFILE location of *YOUR-OBJECT-STORE-URI/20180701/ManifestFile/manifest1.json* with the URI to the manifest file on external object store.

Your result will be similar to the following:

```

      NodeId      33
      AmpId       0
      Sequence    1
      ObjectName  /S3/s3.amazonaws.com/iewritenostest/20180627/
ManifestFile/object_33_0_1.parquet
      ObjectSize      1327
      RecordCount    3

      NodeId      33
      AmpId       3
      Sequence    1
      ObjectName  /S3/s3.amazonaws.com/iewritenostest/20180627/
ManifestFile/object_33_3_1.parquet
      ObjectSize      1346
      RecordCount    3
      [...]

```

2. You can view the manifest file on external storage using the command line of your external object store.

The entries stored in the manifest file are similar to this:

```
{
  "entries": [
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_0_1.parquet",
      "meta": {
        "content_length": 2803
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_6_1.parquet",
      "meta": {
        "content_length": 2733
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_1_1.parquet",
      "meta": {
        "content_length": 3009
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_7_1.parquet",
      "meta": {
        "content_length": 2591
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_2_1.parquet",
      "meta": {
        "content_length": 2725
      }
    }
  ]
}
```

Creating a Manifest File Without Writing Separate Objects to the Object Storage

The following examples show how to use WRITE_NOS to create a manifest file.

When Write_NOS is done creating objects on external object storage it can create a manifest file with the names of the objects created. If there is a crash, abort, or error while writing objects, the manifest file is not created.

- The first example shows how to create the manifest file if a WRITE_NOS operation aborts before all the objects have been written out.
- The second example shows how to overwrite a manifest file.
- The third example shows how to create a manifest file using READ_NOS with RETURNTYPE('NOSREAD_KEYS') as input to writing the manifest file.

Note:

In these examples only the manifest file is written, not the actual objects the manifest file identifies.

Prerequisites

1. If not already done, set up or obtain access to the external object storage where you want to write data. Follow the instructions from your external storage vendor.
2. If not already done, set up an authorization to access external object storage that has the credential to the bucket. See [Controlling Foreign Table Access with an AUTHORIZATION Object](#).

Create A Manifest Table in the Database

3. Create a database table to store the manifest file entries:

```
CREATE TABLE ManifestTbl
(
  ObjectName VARCHAR(1024) CHARACTER SET UNICODE NOT CASESPECIFIC,
  ObjectSize BIGINT
);
```

Example: Using MANIFESTFILE('file_name') and MANIFESTONLY('TRUE')

The following creates a manifest file on external object store.

First, create a database table containing manifest file values, then writes the rows in the table to a manifest file on external object store.

1. If not already done, create a manifest table called ManifestTbl in the database to store the manifest values. See [Creating a Manifest File Without Writing Separate Objects to the Object Storage](#).
2. Insert data into ManifestTbl:

```
INSERT INTO ManifestTbl VALUES ('YOUR-OBJECT-STORE-URI/20180627/
ManifestFile/object_33_0_1.parquet', 2803);
INSERT INTO ManifestTbl VALUES ('YOUR-OBJECT-STORE-URI/20180627/
ManifestFile/object_33_1_1.parquet', 3009);
INSERT INTO ManifestTbl VALUES ('YOUR-OBJECT-STORE-URI/20180627/
ManifestFile/object_33_2_1.parquet', 2733);
INSERT INTO ManifestTbl VALUES ('YOUR-OBJECT-STORE-URI/20180627/
ManifestFile/object_33_3_1.parquet', 2591);
```

3. Write the data in ManifestTbl to the manifest file on external object store:

```
SELECT *
FROM WRITE_NOS (
  ON ( SELECT * FROM ManifestTbl )
  USING
    LOCATION(' YOUR-OBJECT-STORE-URI/20180627/ManifestFile2/')
    AUTHORIZATION(MyAuthObj)
    STOREDAS('PARQUET')
    MANIFESTFILE('/YOUR-OBJECT-STORE-URI/20180627/
ManifestFile2/manifest2.json')
    MANIFESTONLY('TRUE')
) AS d;
```

Replace the LOCATION of *YOUR-OBJECT-STORE-URI/20180701/ManifestFile2/* with the URI to the external object store location where you want to write the manifest file.

Replace the MANIFESTFILE location of *YOUR-OBJECT-STORE-URI/20180701/ManifestFile2/manifest2.json* with the URI to the manifest file on external object store.

MANIFESTONLY('TRUE') writes only a manifest file to external storage; no data objects are written:

- Use this option to create a new manifest file in the event that a WRITE_NOS operation fails due to a database abort or restart, or when network connectivity issues interrupt and stop a WRITE_NOS operation before all data has been written to external storage.

- The manifest is created from the table or query result set that is input to WRITE_NOS. The input must be a list of storage object names and sizes, with one row per object.

Your result will be similar to the following:

ObjectName	ObjectSize
/S3/s3.amazonaws.com/iewritenostest/20180627/ManifestFile2/manifest2.json	433

- You can view the manifest file on external storage using commands from the command line of your external object store. The manifest file contains information similar to the following:

```
{
  "entries": [
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_2_1.parquet",
      "meta": {
        "content_length": 2733
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_7_1.parquet",
      "meta": {
        "content_length": 2591
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_0_1.parquet",
      "meta": {
        "content_length": 2803
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_1_1.parquet",
      "meta": {
        "content_length": 3009
      }
    }
  ]
}
```

Example: Using MANIFESTFILE('file_name'), MANIFESTONLY('TRUE'), and OVERWRITE('TRUE') to Overwrite the Manifest File

The following overwrites a manifest file on external object store.

Create a database table containing manifest file values, then write the rows in the table to the manifest file on external object store. Because the manifest file already exists on external object store, it is overwritten, not appended to.

- If not already done, create a manifest table called ManifestTbl in the database to store the manifest values. See [Creating a Manifest File Without Writing Separate Objects to the Object Storage](#).
- Overwrite the manifest file with data from the table called ManifestTbl:

```
SELECT *
FROM WRITE_NOS (
  ON ( SELECT * FROM ManifestTbl )
  USING
    LOCATION(' YOUR-OBJECT-STORE-URI/20180627/ManifestFile2/')
    AUTHORIZATION(MyAuthObj)
    STOREDAS('PARQUET')
    MANIFESTFILE(' YOUR-OBJECT-STORE-URI/20180627/
ManifestFile2/manifest2.json')
    MANIFESTONLY('TRUE')
    OVERWRITE('TRUE')
) AS d ;
```

Replace the LOCATION of *YOUR-OBJECT-STORE-URI/20180701/ManifestFile2/* with the URI to the external object store location where you want to write the manifest file.

Replace the MANIFESTFILE location of *YOUR-OBJECT-STORE-URI/20180701/ManifestFile2/manifest2.json* with the URI to the manifest file on external object store.

MANIFESTONLY('TRUE') writes only a manifest file to external storage; no data objects are written:

- Use this option to create a new manifest file in the event that a WRITE_NOS operation fails due to a database abort or restart, or when network connectivity issues interrupt and stop a WRITE_NOS operation before all data has been written to external storage.
- The manifest is created from the table or query result set that is input to WRITE_NOS. The input must be a list of storage object names and sizes, with one row per object.

OVERWRITE('TRUE') overwrites an existing manifest file. Note, OVERWRITE('TRUE') must be used with MANIFESTONLY('TRUE').

Your result will be similar to the following:

ObjectName	ObjectSize
/S3/s3.amazonaws.com/iewritenostest/20180627/ManifestFile2/manifest2.json	433

3. You can view the manifest file on external storage using commands from the command line of your external object store. The manifest file contains information similar to the following:

```
{
  "entries": [
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_0_1.parquet",
      "meta": {
        "content_length": 2803
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_2_1.parquet",
      "meta": {
        "content_length": 2725
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_6_1.parquet",
      "meta": {
        "content_length": 2733
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_7_1.parquet",
      "meta": {
        "content_length": 2591
      }
    },
    {
      "url": "s3://ie-writenos-bucket/20180701/ManifestFile/object_33_1_1.parquet",
      "meta": {
        "content_length": 3009
      }
    }
  ]
}
```

Example: Create a Manifest File Using READ_NOS

If a WRITE_NOS operation fails before it can create a manifest file you can use READ_NOS with RETURNTYPE('NOSREAD_KEYS') to read the keys of the existing objects in the external storage. This is used as input to create a new manifest file. The new manifest file created this way reflects all data objects currently in the external storage location, and can aid in determining which data objects resulted from an incomplete WRITE_NOS operation.

1. Create a manifest file using READ_NOS as input:

```
SELECT * FROM WRITE_NOS
( ON
  (SELECT Location, ObjectLength FROM (
    LOCATION= 'YOUR-OBJECT-STORE-URI/20180627/'
    AUTHORIZATION=MyAuthObj
    RETURNTYPE= 'NOSREAD_KEYS'
  ) as d1 )
USING
```

```

    AUTHORIZATION(MyAuthObj)
    MANIFESTFILE('YOUR-OBJECT-STORE-URI/20180627/
ManifestFile3/manifest3.json')
    MANIFESTONLY('TRUE')
    OVERWRITE('TRUE')
) AS d;

```

Replace the LOCATION of *YOUR-OBJECT-STORE-URI/20180701/* with the URI to the external object store location where you want to write the manifest file.

Replace the MANIFESTFILE location of *YOUR-OBJECT-STORE-URI/20180701/ManifestFile3/manifest3.json* with the URI to the manifest file on external object store.

Your result will be similar to the following:

ObjectName	ObjectSize
/S3/s3.amazonaws.com/iewritenostest/20180627/ManifestFile3/manifest3.json	996

Using Delta Lake Manifest Files

Teradata Vantage supports reading from object store tables using a manifest file, which is a text file that lists the paths of the objects stored on external object store to read for querying a table. When a foreign table is defined in the object store using manifest files, Vantage can use the list of files in the manifest rather than finding the files by directory listing.

Setting up Vantage to Delta Lake Integration

To set up a Vantage to Delta Lake integration using manifest files and query Delta tables:

Generate Manifests of a Delta Table

1. See <https://docs.delta.io/latest/integrations.html> for instructions on how to generate and configure manifests of a Delta table.

Configure Vantage to Read the Generated Manifests

2. Define a new foreign table in Vantage using the format `SymlinkTextInputFormat` and the manifest location `path-to-delta-table/_symlink_format_manifest/`:

```
CREATE FOREIGN TABLE mytable
  , EXTERNAL SECURITY myauthobj
  USING (
    LOCATION(' /s3/mybucket.s3.amazonaws.com/path-to-
delta=table/_symlink_format_manifest/')
    MANIFEST('TRUE')
    TABLE_FORMAT('DELTALAKE') );
```

The presence of the `_symlink_format_manifest` kestring confirms that it is a Delta Lake table and is validated only if the `MANIFEST` parameter is set to `TRUE` and the `TABLE_FORMAT` parameter is set as `DELTALAKE`.

`SymlinkTextInputFormat` instructs Vantage to read the data for *mytable* by reading the manifest file instead of using a directory listing to find data files. Replace *mytable* with the name of the external table and *path-to-delta-table* with the absolute path to the Delta table.

Note:

Even though Delta Lake supports schema evolution and queries on a Delta table automatically use the latest schema regardless of the schema defined in the table, Vantage uses the schema defined in its foreign table definition and does not query with the updated schema until the table definition is updated to the new schema.

Examples: CREATE FOREIGN TABLE and READ_NOS Queries

CREATE FOREIGN TABLE and READ_NOS Queries

The following examples show CREATE FOREIGN TABLE and READ_NOS queries with TABLE_FORMAT NVP as DELTALAKE, MANIFEST NVP set to TRUE and the location of the manifest file inside the _symlink_format_manifest folder.

CREATE FOREIGN TABLE is used to create a manifest table. The table is created using the data files from the manifest files in the given location.

```
create foreign table dl_table1
,EXTERNAL SECURITY INVOKER TRUSTED test0
USING
( LOCATION ('/s3/s3.amazonaws.com/testdeltalake/
deltalakewp/_symlink_format_manifest')
MANIFEST('TRUE')
TABLE_FORMAT('DELTALAKE')
);

*** Table has been created.
*** Total elapsed time was 4 seconds.
+-----+-----+-----+-----+-----+-----+-----+-----+

show table dl_table1;

*** Text of DDL statement returned.
*** Total elapsed time was 1 second.
-----

CREATE MULTISET FOREIGN TABLE dl_table1 ,FALLBACK ,
EXTERNAL SECURITY INVOKER TRUSTED TEST0 ,
MAP = TD_MAP1
(
  Location VARCHAR(2048) CHARACTER SET UNICODE CASESPECIFIC,
  street VARCHAR(22) CHARACTER SET UNICODE NOT CASESPECIFIC,
  state VARCHAR(2) CHARACTER SET UNICODE NOT CASESPECIFIC,
  bedrooms INTEGER,
  bathrooms INTEGER,
  sqft INTEGER,
  types VARCHAR(11) CHARACTER SET UNICODE NOT CASESPECIFIC,
  price INTEGER)
USING
(
```

```

        LOCATION ('/s3/s3.amazonaws.com/testdeltalake/
deltalakewp/_symlink_format_manifest')
        MANIFEST ('TRUE')
        TABLE_FORMAT ('DELTALAKE')
        PATHPATTERN ('$var1/$zip/$var3')
        STOREDAS ('PARQUET')
    )
    NO PRIMARY INDEX
    PARTITION BY COLUMN ADD 65524;

```

```
SELECT * FROM dl_table1;
```

```

*** Query completed. 61 rows found. 8 columns returned.
*** Total elapsed time was 6 seconds.

```

Location

```

-----
-----
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00009-70377ec9-23db-4c9b-9c1f-50bf90a72d9a.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00007-1ca7a971-5117-46b4-85b2-c7752bbdc2cb.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-
a4ea-0ba96b39a598.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-
b754-b6d47dab8901.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00004-66a61240-33fd-4fba-8bd1-5e53653984f5.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-
f6f1e0f6-5ccd-48ff-97f0-b2d427b5cccec.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00003-
e4c7c3cb-7aa0-4814-85e2-4683c1cec53e.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00002-83b8aad2-9d4a-4cef-
ac9a-acb07340b7d4.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00009-70377ec9-23db-4c9b-9c1f-50bf90a72d9a.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00007-1ca7a971-5117-46b4-85b2-c7752bbdc2cb.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-
a4ea-0ba96b39a598.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-
b754-b6d47dab8901.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00004-66a61240-33fd-4fba-8bd1-5e53653984f5.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-
f6f1e0f6-5ccd-48ff-97f0-b2d427b5cccec.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00003-
e4c7c3cb-7aa0-4814-85e2-4683c1cec53e.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00002-83b8aad2-9d4a-4cef-
ac9a-acb07340b7d4.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00009-70377ec9-23db-4c9b-9c1f-50bf90a72d9a.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00007-1ca7a971-5117-46b4-85b2-c7752bbdc2cb.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-
a4ea-0ba96b39a598.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-
b754-b6d47dab8901.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00004-66a61240-33fd-4fba-8bd1-5e53653984f5.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-
f6f1e0f6-5ccd-48ff-97f0-b2d427b5cccec.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00002-83b8aad2-9d4a-4cef-

```

```

ac9a-acb07340b7d4.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00009-70377ec9-23db-4c9b-9c1f-50bf90a72d9a.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00007-1ca7a971-5117-46b4-85b2-c7752bbdc2cb.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-
a4ea-0ba96b39a598.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-
b754-b6d47dab8901.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00004-66a61240-33fd-4fba-8bd1-5e53653984f5.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-
f6f1e0f6-5ccd-48ff-97f0-b2d427b5cccec.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00009-70377ec9-23db-4c9b-9c1f-50bf90a72d9a.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00007-1ca7a971-5117-46b4-85b2-c7752bbdc2cb.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-
a4ea-0ba96b39a598.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-
b754-b6d47dab8901.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00004-66a61240-33fd-4fba-8bd1-5e53653984f5.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-
f6f1e0f6-5ccd-48ff-97f0-b2d427b5cccec.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00001-79ff7696-ba19-48fc-
bbb4-842fbe04baac.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00007-1ca7a971-5117-46b4-85b2-c7752bbdc2cb.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-
a4ea-0ba96b39a598.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-
b754-b6d47dab8901.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00004-66a61240-33fd-4fba-8bd1-5e53653984f5.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-
f6f1e0f6-5ccd-48ff-97f0-b2d427b5cccec.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00001-79ff7696-ba19-48fc-
bbb4-842fbe04baac.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00000-a1f8f541-c0b2-44ad-
bfef-447a383400d1.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00002-00c3c9b4-
fc91-4762-8724-d1f7cdcd8a72.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-
b754-b6d47dab8901.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00004-66a61240-33fd-4fba-8bd1-5e53653984f5.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00001-79ff7696-ba19-48fc-
bbb4-842fbe04baac.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00000-a1f8f541-c0b2-44ad-
bfef-447a383400d1.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00002-00c3c9b4-
fc91-4762-8724-d1f7cdcd8a72.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-
b754-b6d47dab8901.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/
part-00004-66a61240-33fd-4fba-8bd1-5e53653984f5.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00001-79ff7696-ba19-48fc-
bbb4-842fbe04baac.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00000-a1f8f541-c0b2-44ad-
bfef-447a383400d1.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00002-00c3c9b4-
fc91-4762-8724-d1f7cdcd8a72.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00001-09b48cd3-5c6a-467a-
a63a-134c752cdede.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00000-0ab15a43-
e3b4-4832-8775-d3db4b960bcc.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00000-a1f8f541-c0b2-44ad-
bfef-447a383400d1.c000.snappy.par

```

```
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00001-09b48cd3-5c6a-467a-
a63a-134c752cdede.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00000-0ab15a43-
e3b4-4832-8775-d3db4b960bcc.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00000-a1f8f541-c0b2-44ad-
bfef-447a383400d1.c000.snappy.par
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00000-0ab15a43-
e3b4-4832-8775-d3db4b960bcc.c000.snappy.par
```

```
SELECT street FROM dl_table1;
```

```
*** Query completed. 61 rows found. One column returned.
```

```
*** Total elapsed time was 3 seconds.
```

```
street
```

```
-----
3536 SUN MAIDEN WAY
3828 BLACKFOOT WAY
3361 ALDER CANYON WAY
4437 MITCHUM CT
8716 LONGSPUR WAY
1207 CRESCENDO DR
1675 VERNON ST Unit 8
318 ANACAPA DR
5712 MELBURY CIR
3318 DAVIDSON DR
7921 DOE TRAIL WAY
8020 WALERGA RD
7837 ABBINGTON WAY
140 VENTO CT
7641 ROSEHALL DR
448 ELMWOOD CT
4085 COUNTRY DR
8721 SPRUCE RIDGE WAY
4636 TEAL BAY CT
4508 OLD DAIRY DR
4008 GREY LIVERY WAY
633 HANISCH DR
9197 CORTINA CIR
7863 CRESTLEIGH CT
4741 PACIFIC PARK DR
5312 MARBURY WAY
3305 RIO ROCA CT
7895 CABER WAY
652 FIFTEEN MILE DR
5708 RIDGEPOINT DR
5308 MARBURY WAY
```


8108 FILIFERA WAY
 8316 NORTHAM DR
 4712 PISMO BEACH DR
 308 ATKINSON ST
 6709 ROSE BRIDGE DR
 4509 WINJE DR
 7901 GAZELLE TRAIL WAY
 4240 WINJE DR
 3228 BAGGAN CT
 227 MAHAN CT Unit 1
 7697 ROSEHALL DR
 201 FIRESTONE DR
 2981 WRINGER DR
 3604 KODIAK WAY
 8428 MISTY PASS WAY
 37 WHITE BIRCH CT
 167 VALLEY OAK DR
 3930 ANNABELLE AVE
 8636 LONGSPUR WAY
 4844 CLYDEBANK WAY
 1675 VERNON ST Unit 24
 281 SPYGLASS HL
 1975 SIDESADDLE WAY
 3212 CORNICHE LN
 506 BEDFORD CT
 419 DAWN RIDGE RD
 1407 TIFFANY CIR
 1786 PIEDMONT WAY
 624 HOVEY WAY
 500 WINCHESTER CT

```

SELECT * FROM ( LOCATION='/s3/s3.amazonaws.com/testdeltalake/deltalakewp/_symlink_format_manifest'
AUTHORIZATION=test1 MANIFEST='TRUE' TABLE_FORMAT='DELTA_LAKE') as dt;
  
```

```

*** Query completed. 61 rows found. 8 columns returned.
*** Total elapsed time was 44 seconds.
  
```

Location

```

-----
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00009-70377ec9-23db-4c9b-9c1f-50bf90a7
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00007-1ca7a971-5117-46b4-85b2-c7752bbd
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-a4ea-0ba96b39
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-b754-b6d47dab
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00009-70377ec9-23db-4c9b-9c1f-50bf90a7
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00007-1ca7a971-5117-46b4-85b2-c7752bbd
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-a4ea-0ba96b39
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-b754-b6d47dab
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00009-70377ec9-23db-4c9b-9c1f-50bf90a7
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00007-1ca7a971-5117-46b4-85b2-c7752bbd
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-a4ea-0ba96b39
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-b754-b6d47dab
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00009-70377ec9-23db-4c9b-9c1f-50bf90a7
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00007-1ca7a971-5117-46b4-85b2-c7752bbd
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-a4ea-0ba96b39
  
```



```

/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-b754-b6d47dab
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00004-66a61240-33fd-4fba-8bd1-5e536539
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00007-1ca7a971-5117-46b4-85b2-c7752bbd
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00006-917150c3-054a-4ac7-a4ea-0ba96b39
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-b754-b6d47dab
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00004-66a61240-33fd-4fba-8bd1-5e536539
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-f6f1e0f6-5ccd-48ff-97f0-b2d427b5
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00003-e4c7c3cb-7aa0-4814-85e2-4683c1ce
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00005-0d48136f-25b1-4ee7-b754-b6d47dab
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00004-66a61240-33fd-4fba-8bd1-5e536539
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-f6f1e0f6-5ccd-48ff-97f0-b2d427b5
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-f6f1e0f6-5ccd-48ff-97f0-b2d427b5
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00002-00c3c9b4-fc91-4762-8724-d1f7cdcd
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00002-83b8aad2-9d4a-4cef-ac9a-acb07340
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00004-66a61240-33fd-4fba-8bd1-5e536539
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-f6f1e0f6-5ccd-48ff-97f0-b2d427b5
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00002-00c3c9b4-fc91-4762-8724-d1f7cdcd
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00002-83b8aad2-9d4a-4cef-ac9a-acb07340
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00004-66a61240-33fd-4fba-8bd1-5e536539
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00008-f6f1e0f6-5ccd-48ff-97f0-b2d427b5
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00001-09b48cd3-5c6a-467a-a63a-134c752c
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95843/part-00004-66a61240-33fd-4fba-8bd1-5e536539
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00000-a1f8f541-c0b2-44ad-bfef-447a3834
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00001-09b48cd3-5c6a-467a-a63a-134c752c
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00001-79ff7696-ba19-48fc-bbb4-842f8e04
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00000-a1f8f541-c0b2-44ad-bfef-447a3834
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00001-79ff7696-ba19-48fc-bbb4-842f8e04
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00000-a1f8f541-c0b2-44ad-bfef-447a3834
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00001-79ff7696-ba19-48fc-bbb4-842f8e04
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95678/part-00000-a1f8f541-c0b2-44ad-bfef-447a3834
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00000-0ab15a43-e3b4-4832-8775-d3db4b96
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00000-0ab15a43-e3b4-4832-8775-d3db4b96
/S3/s3.amazonaws.com/testdeltalake/deltalakewp/zip=95661/part-00000-0ab15a43-e3b4-4832-8775-d3db4b96

```

Delta Lake Manifest Files Limitations

The Vantage integration has known limitations in its behavior.

Data Consistency

Whenever Delta Lake generates updated manifests, it atomically overwrites existing manifest files.

Therefore, Vantage will always see a consistent view of the data files. However, the granularity of the consistency guarantees depends on whether the table is partitioned or not.

- **Unpartitioned tables:** All the file names are written in one manifest file, which is updated atomically. Vantage sees full table snapshot consistency.
- **Partitioned tables:** A manifest file is partitioned in the same Hive-partitioning-style directory structure as the original Delta table. This means that each partition is updated atomically, and Vantage sees a consistent view of each partition, but not a consistent view across partitions. Furthermore, since all manifests of all partitions cannot be updated together, concurrent attempts to generate manifests can lead to different partitions having manifests of different versions. While this consistency guarantee under data change is weaker than that of reading Delta tables with Spark, it is still stronger than formats like Parquet as they do not provide partition-level consistency.

Depending on what storage system you are using for Delta tables, it is possible to get incorrect results when Presto or Athena concurrently queries the manifest while the manifest files are being rewritten. In file system implementations that lack atomic file overwrites, a manifest file may be momentarily unavailable. Therefore, use manifests with caution if their updates are likely to coincide with queries from Presto or Athena.

Performance

Very large numbers of files can hurt Vantage performance. Databricks recommends that you compact the files of the table before generating the manifests. The number of files should not exceed 1000 (for the entire unpartitioned table or for each partition in a partitioned table).

Schema Evolution

Delta Lake supports schema evolution and queries on a Delta table automatically using the latest schema regardless of the schema defined in the table in the Hive metastore. However, Vantage uses the schema defined in its table definition, and will not query with the updated schema until the table definition is updated to the new schema.

Data Cleanup

After you finish testing, clean up test users, tables, and authorization objects.

1. Log on as an administrative user and issue the following commands:

```
DELETE USER nos_usr;  
DROP USER nos_usr;
```

External File Formats

External File Object Rules

External files must be in one of the following formats:

- JSON
- CSV
- Parquet

Field names in CSV and JSON are case sensitive.

Unsupported files and compression types are skipped.

Records that contain errors are skipped. Errors can include mismatched quotation marks (" "), embedded line feeds, and so forth.

External files with UTF-8 BOMs (Byte Order Markers) are supported, but UTF-16LE, UTF-16BE, UTF-32LE, UTF-32BE BOMs are not supported.

Queries return results only for file formats where the format matches the format of the first file specified by the LOCATION option, including:

- File type (JSON, CSV, or Parquet)
- File compression (GZIP for JSON or CSV, SNAPPY for Parquet, or uncompressed)
- Character encoding (LATIN or UTF-8)
- Field delimiter (comma, tab, and so on)
- Record delimiter (line feed)

JSON External Files

Each record in the JSON external data must be formatted on a single line, terminated by a new line symbol (\n).

External JSON files can contain individual records or a single JSON array that uses a comma (,) as the record delimiter. For single-array JSON files, the record can span across multiple lines. Optionally, the JSON array can be named.

Vantage can read the following JSON document in external storage:

```
{"field1": true,"field2": "somestring","field3": {"field4": 1}}
```

Vantage cannot read the external JSON data if it is stored in this format, due to the line breaks:

```
{
  "field1" : true,
```

```

    "field2":
        "somestring",
    "field3":
    {
        "field4":1
    }
}

```

Because the following external JSON data is in a single array (delimited by the opening and closing square bracket characters), using commas to delimit individual records in the array, Vantage can read the data, even with the embedded line breaks:

```

[ {
    "field1": "string1", "field2": "string2"
}, {
    "field1": "string3", "field2": "string4"
}]

```

The field names are case sensitive. So, a reference to Field1 will not match the "field1" in the examples above.

Spaces inside quotes for the field names are significant. So, if the record contained "field1 ", then a reference to `payload.field1` does not match.

Here is an example of a named JSON array:

```

{ "Fruits":    [
    \{ "fruit": "Apple", "size": "Large", "color": "Red" }
    ,
    { "fruit": "Banana", "size": "Medium", "color": "Yellow" },
    { "fruit": "Orange", "size": "Medium", "color": "Orange" },
    { "fruit": "Guava", "size": "Small", "color": "Green" },
    { "fruit": "Grapes", "size": "VerySmall", "color": "Black" }
  ]
}

```

External JSON files can be compressed in GZIP format. Other forms of file compression are not supported.

CSV External Files

Do not enclose a CSV value in quotation marks if it is to be used or cast as a numeric value. If you do, Vantage returns a null value or skipped record error. To strip quotation marks, use `STRIP_ENCLOSING_CHAR`.

You can compress an external CSV file only in GZIP format.

Parquet External Files

To access external files containing Parquet-formatted data, you must specify `PARQUET` for the `USING STOREDAS` option in the `CREATE FOREIGN TABLE` statement.

Column names for Parquet data can contain a maximum of 128 characters and are case-sensitive.

Foreign tables for Parquet data are created without a primary index (NOPI) and are column-partitioned.

You cannot specify the following table-level options for foreign tables that contain Parquet formatted data:

- Row partitioning
- Subrow partitioning (partial row partitioning for a column-partitioned table)
- Multicolumn partitions
- Autocompression
- `ROWFORMAT` option of the `USING` clause
- A primary index

External Parquet files can be compressed in Snappy format.

You must explicitly define data columns for foreign tables that contain Parquet data. Define the columns in the same order as the columns in the Parquet files, and use Vantage data types that correspond to the Parquet data types:

Parquet Data Type	Corresponding Vantage Data Type	Description
UINT_8	SMALLINT	Unsigned integer with range 0 to 255.
UINT_16	INTEGER	Unsigned integer with range 0 to 65535.
UINT_32	BIGINT	Unsigned integer with range 0 to 4,294,967,295.
UINT_64	DECIMAL(20,0)	Unsigned integer with range 0 to 18,446,744,073,709,551,615.
INT_8	BYTEINT	Signed integer with range -128 to 127.
INT_16	SMALLINT	Signed integer with range -32768 to 32767.
INT_32	INTEGER	Signed integer with range -2,147,483,648 to 2,147,483,647.
INT_64	BIGINT	Signed integer with range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

Parquet Data Type	Corresponding Vantage Data Type	Description
INT_96	TIMESTAMP(6)	Used to represent timestamp. First 8 bytes has number of nanoseconds and next 4 bytes has number of days since Julian day.
DECIMAL(p,s)	DECIMAL(p,s)	Decimal where precision is between 1 and 38 (inclusive) and scale is between 0 and 38 (inclusive).
	VARBYTE(p+1), where p is the precision of the DECIMAL.	Precision and scale is greater than 38.
FLOAT	REAL	IEEE 32-bit floating point.
DOUBLE	REAL	IEEE 64-bit floating point.
DATE	DATE	Number of days from the Unix epoch, 1 January 1970
TIME_MILLIS	TIME(3) or TIME(6)	Stores time in milliseconds.
TIME_MICROS	TIME(6)	Stores time in microseconds.
TIMESTAMP_MILLIS	TIMESTAMP(3) or TIMESTAMP(6)	Stores timestamp in milliseconds.
TIMESTAMP_MICROS	TIMESTAMP(6)	Stores timestamp in microseconds.
INTERVAL	VARCHAR CHARACTER SET UNICODE	Stores interval in months, days and milliseconds.
BOOL	BYTEINT	Boolean (True or False).
STRING	VARCHAR or CLOB CHARACTER SET UNICODE	Encoded as a UTF8 byte array.
BSON	JSON STORAGE FORMAT BSON	BSON data.
JSON	JSON CHARACTER SET UNICODE	JSON data.
STRUCT	VARCHAR or CLOB CHARACTER SET UNICODE	Group of fixed members.
MAP	VARCHAR or CLOB CHARACTER SET UNICODE	Maps keys to values.
LIST	VARCHAR or CLOB CHARACTER SET UNICODE	Contains data that is stored in array.

Parquet Data Type	Corresponding Vantage Data Type	Description
ENUM	VARCHAR or CLOB CHARACTER SET UNICODE	Store enumerated values encoded as a UTF8 string.
ARRAY	VARCHAR or CLOB CHARACTER SET UNICODE	Stored as repeated fields. Can be an array of a single value or multiple values.

The Parquet files do not need the same column layout and do not need the same data type (if the data types are compatible) as the foreign table definition. For example:

File	Column 1	Column 2	Column 3	Column 4	Allowed?
Foreign table is defined with 3 columns with the data types specified to the right	INT	VARCHAR	DATE		N/A
File 1 - has 3 columns of the same data types as the foreign table	INT	VARCHAR	DATE		YES
File 2 - added columns at the end	INT	VARCHAR	DATE	TIMESTAMP	YES
File 3 - removed columns from the end	INT	VARCHAR			YES, if column 3 is nullable
File 4 - different data types	VARCHAR	INTERVAL	BIGINT		NO, the file is skipped
File 5 - different, but compatible data types	BIGINT	CLOB	DATE		YES

Authentication for External Object Stores

Your external object store must be configured to allow Advanced SQL Engine access.

When you configure external storage, you set the credentials to your external object store. Those credentials are used in SQL statements by Advanced SQL Engine. The supported credentials correspond to the values shown in the following table. These credentials are used for USER and PASSWORD by the CREATE AUTHORIZATION command and for ACCESS_ID and ACCESS_KEY by READ_NOS and WRITE_NOS.

System/Scheme	USER	PASSWORD
AWS	Access Key ID	Access Key Secret
Azure / Shared Key	Storage Account Name	Storage Account Key
Azure Shared Access Signature (SAS)	Storage Account Name	Account SAS Token
Google Cloud (S3 interop mode)	Access Key ID	Access Key Secret
Google Cloud (native)	Client Email	Private Key
On-premises object stores	Access Key ID	Access Key Secret
Public access object stores	<empty string> Enclose the empty string in single straight quotes: USER ''	<empty string> Enclose the empty string in single straight quotes: PASSWORD ''

To see examples of supported credentials, see [Variable Substitutions for Examples](#).

Using IAM Credentials with Amazon S3 Buckets

IAM is an alternative to using an access key and password to secure S3 buckets. To allow Advanced SQL Engine access to S3 buckets that use IAM, your S3 bucket policy must be configured with the following Actions for the role that allows access to the bucket.

For READ_NOS:

- S3:GetObject
- S3:ListBucket
- S3:GetBucketLocation

For WRITE_NOS:

- S3:PutObject

Note:

Other Actions are also allowed, such as S3:HeadBucket, S3:HeadObject, S3:ListBucket, and so on.

The following shows an example security policy. You need your EC2 role name and EC2 instance account ID, which are provided to you by Teradata. Once you have those, add an inline policy to your Amazon S3 bucket to grant access to the Teradata EC2 instance.

For example, assuming 's3-cross-access-role' denotes the name of the role, '142600571999' denotes the Teradata EC2 instance account ID, and 'bucketname' denotes the name of your Amazon S3 bucket, an example of the policy to apply to your bucket is as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "s3acl",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam:: 142600571999:role/s3-cross-access-role"
      },
      "Action": [
        "s3:GetObject",
        "s3:ListBucket",
        "s3:GetBucketLocation",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucketname/*",
        "arn:aws:s3:::bucketname"
      ]
    }
  ]
}
```

Related Information

For more information about the security policy, see the Orange Book: *Native Object Store: Teradata Vantage™ Advanced SQL Engine*, TDN0009800.

Accessing Azure Blob Storage Containers

Access to Azure containers must be defined using either Access Keys or Account Shared Access Signatures (SAS). A user with access key information has full control over the entire account. Alternatively, SAS can be defined on Containers, or on objects within containers, so it provides a more fine-grained

authentication approach. NOS uses either type of authentication and does not need to know what type of secret is being supplied.

After the authentication is configured on Azure, the storage account name and key or SAS token can be used in CREATE AUTHORIZATION and READ_NOS statements.

Accessing Google Cloud Storage

When accessing Google Cloud Storage, SQL Engine automatically uses either the S3-compatible connector or the native Google connector, depending on the user credentials.

Example Values for S3-Compatible Google Connector

```
USER: "GOOGDMA4C5FPZ4BHZ7SJGUG0"
PASSWORD: "BVa6b/0Sp2a1syN0G0gCeYWPVEoN5ZEbdZgLPbyT"
```

Example Values for Native Google Connector

```
USER: "teradata-nos-service-account@tde-database-
engineering.iam.gserviceaccount.com"
PASSWORD: "-----BEGIN PRIVATE KEY-----
\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwrgSjAgEAAoIBAQCyfg3398i0xjte\nPQ640BP9Mj12M
H8a8PD9QCilWPXrMKISPM6wy2Dl19bRnRrKTih8QWzBEiPuVnPl\n+1EstIJH7AV0zn+p+dELaN3XCArI
MlRji04cBAXniWZKrkjGaoVG3NJVPEoxjaQQF\na0KtTg4URDTEWx9MpsEvKYS3B7VroSABTp4+n+xyJF6
n8atfNsJ746LSvQAEebB7\nnoOCrSkShjcKci3afBkuFBcMG4XnmJCwK40fmaf15krzN05fQmo6Tuh2JGg
+iLI9E\nnRVTIrc4VDLOmWDDjz/
kC1XqOL19RHxG2lRjYrQUI9X5retXGP85u7i7bViHztyZ\na4Jzoa4HAgMBAAECggEAWJIdi10c2+A5K
H/C10mH6b3q6sGQ6QUDiJ+0HsWjrjPB\nnUKx04hzhtdv/
YJ+FYjsbcuEiPsIqK0QiGc62Tqk59seDMMJebjG1TnSYlHIMq5/
x\n6TxS73r9lRHEvS0oNzNysab5qjWpjVDYqnm5qfbQHLVlu8P9yZL7zNPc49gC6edD\nnpwPf5emTNXF/
G/sJy2IP1claKe3b2RZLSriekmfk3HznfANDJohYc8T+5zA/
a4Ng\n5FzfPWUd6JAJ2PnbpLv7apwj89mo31Ng8W6zcF9P0rDFwWlWzyNSjoF9fWbGiBvq\nnFyHEcy+Xf
p6ZhMQ1tKjKMh5YlVKHHHnC+/
C8T4xqOQKBgQDZCOht40ar6Zn8Lm4e\nf37M82Hy2HLXLNOyrp0UngGFCHz1Sw1Rf0pbsG7sbTPA3UgdG
C6cga+rF0G7L2I9\nns+W0mK8CSJ7PKI8qzJM61eC1Kl10H9Ym6XmqoTp+6A51nYHYzFQBq/
c5wrzNr3Z8\nnFyMgr89aqR7rviP/
VwQEYAp5iQKBgQDSibUaICU9F9oUd7nMn1qzRbCd0Z13Xphv\nn2JXJog57Zm9f08yATj/i/
4wBTuPPZpWwEcQzN9LEEoTQ+lys5JY8MUjfxQAYrif\nn1WE1y3+ZracLrIfbv1J3jjY5ykwuDigJSQ2
xBXiDFzk13EMta+CylCmMm5R1xJJ\nnUPAnaz9XDwKBgChMoCPu88Y0HTqXQHkz8aaKtrBKAQDzwOdXxS1
+XVffaJx/
cBym\nn3x7BVwdti40PUxmb60PErOK+8cSZGsvMQJ7GuRBafaqHpHXVAK2XEx9wm205maEm\nnDyJgx6TS4
AF8Sqt/48Vfsgs2nqoun3iRL00dgYobBxPM/
ZAnDyb4IgHJAoGBAImq\nnW84ieNcS0pmRk981z9PrXMPFNybhPvtzbrYyji+oUmi+LEE1aVbf7Ecqs5F5
LaW8\nnaENpXBqzVT9khdBtFV0Mi08NmNKG9jVj4PidaQiRVk/
znIk13BGq3tA49Ek1Ho6\nn7wwUeY7irV8w9I3v+zyUR9XUoggdTwpVT0MnQHqHAoGAVKiDUTvwQOBDHPR
```

```
1f0zu\ntoor3YKmoymRItpwoiemLKhKsonE2h0zxYTdrR43GmaripoplXgz2kz61TTR0m4T\nF8Ucw//  
ocYIsXxZeGrzfJR80FIPE4ki1fA08i0KGYoGY0Ke6vQIj699bLdK9pYJZ\n3ZTalmSAvk9SqPoyZR7JJF  
s=\n-----END PRIVATE KEY-----\n”
```

Setting Up an Object Store for River Flow Data

Many of the examples use a sample river flow data set. USGS Surface-Water Data Sets are provided courtesy of the U.S. Geological Survey.

To run the examples, you can use the river flow data from Teradata-supplied public buckets. Or you can set up a small object store for the data set.

The following instructions explain how to set up the river flow data on your own external object storage.

Your external object storage must be configured to allow Advanced SQL Engine access.

When you configure external storage, you set the credentials to your external object storage. Those credentials are used in SQL commands. The supported credentials for USER and PASSWORD (used in the CREATE AUTHORIZATION command) and for ACCESS_ID and ACCESS_KEY (used by READ_NOS and WRITE_NOS) correspond to the values shown in the following table:

System/Scheme	USER	PASSWORD
AWS	Access Key ID	Access Key Secret
Azure / Shared Key	Storage Account Name	Storage Account Key
Azure Shared Access Signature (SAS)	Storage Account Name	Account SAS Token
Google Cloud (S3 interop mode)	Access Key ID	Access Key Secret
Google Cloud (native)	Client Email	Private Key
On-premises object stores	Access Key ID	Access Key Secret
Public access object stores	<empty string> Enclose the empty string in single straight quotes: USER ''	<empty string> Enclose the empty string in single straight quotes: PASSWORD ''

The following provides further details for setting up credentials on your external object storage:

Platform	Notes
Amazon S3 IAM	<p>IAM is an alternative to using an access key and password to secure S3 buckets. To allow Advanced SQL Engine access to S3 buckets that use IAM, your S3 bucket policy must be configured with the following Actions for the role that allows access to the bucket:</p> <ul style="list-style-type: none"> • S3:GetObject • S3:ListBucket • S3:GetBucketLocation <p>For WRITE_NOS:</p> <ul style="list-style-type: none"> • S3:PutObject

Platform	Notes
	Note: Other Actions are also allowed, such as S3:HeadBucket, S3:HeadObject, S3:ListBucket, and so on.
Azure Blob storage and Azure Data Lake Storage Gen2	A user with access key information has full control over the entire account. Alternatively, SAS can be defined on Containers, or on objects within containers, so it provides a more fine-grained authentication approach. NOS uses either type of authentication and does not need to know what type of secret is being supplied. Note: Only Account SAS tokens are supported. Service SAS tokens generate errors and are rejected.
Google Cloud Storage	To allow Advanced SQL Engine access, the following permissions are needed: <ul style="list-style-type: none"> • storage.objects.get • storage.objects.list

See your cloud vendor documentation for instructions on creating an external object storage account.

The following steps may require the assistance of your public cloud administrator.

1. Create an external object storage on a Teradata-supported external object storage platform. Give your external object storage a unique name. In the Teradata-supplied examples, the bucket/container is called td-usgs. Because the bucket/container name must be unique, choose a name other than td-usgs.
2. On Amazon, generate an access ID and matching secret key for your bucket or generate an Identity and Access Management (IAM) user credential. On Azure, generate Account SAS tokens (not Service SAS tokens) for your td-usgs container. On Google Cloud Storage, generate an access ID and matching secret key for your bucket.
3. Download the sample data from <https://downloads.teradata.com/> (look for NOS Download Data) to your client/laptop. The ZIP file contains sample river flow data in CSV, JSON, and Parquet data formats.
4. Copy the sample data to your bucket or container, being careful to preserve the data directory structure. For example, use a location similar to the following:
 - Amazon S3: /S3/YOUR-BUCKET.s3.amazonaws.com/JSONDATA
 - Azure Blob storage and Azure Data Lake Storage Gen2: /az/YOUR-STORAGE-ACCOUNT.blob.core.windows.net/td-usgs/CSVDATA/
 - Google Cloud Storage: /gs/storage.googleapis.com/YOUR-BUCKET/CSVDATA/

Note, you can use the Amazon S3 or Azure management consoles or a utility like AWS CLI to copy the data to your external object storage. For Google Cloud Storage, you can use the gsutil tool to copy the data to your external object storage.

5. In the example code replace td-usgs, YOUR-BUCKET, and YOUR-STORAGE-ACCOUNT with the location of your object storage.
6. Replace YOUR-ACCESS-KEY-ID and YOUR-SECRET-ACCESS-KEY with the access values for your external object storage.

Best Practices for Using NOS

- To reduce the volume of data processed in the Advanced SQL Engine, either specify a more specific LOCATION string in the USING clause of the foreign table definition or use PATHPATTERN variables in the SQL statement.
- To ensure that path filtering is applied consistently, Teradata recommends that the database administrator capture path filtering (with appropriate casting of column data types) in a view of the foreign table and make the view available to end users.
- To improve performance of SQL statements with GROUP BY or ORDER BY clauses that include JSON files, cast values from their default VARCHAR data type to narrower data types.
- When joining a foreign table to either a relational table or another foreign table, collect statistics on the payload attribute that is used as a join constraint.
- If you generate spool tables, the spool is smaller if you cast values to narrower data types.

Native Object Store Limitations

These Native Object Store limitations are relevant to the CREATE FOREIGN TABLE statement.

CSV and JSON Payload Size

- The maximum payload size is 16,776,192 bytes for READ_NOS for CSV and JSON data formats.
 - For UNICODE, that is equivalent to 8,388,096 characters.
 - For LATIN, that is equivalent to 16,776,192 characters.

The actual payload size is equivalent to the dataset individual record size.

Parquet Page Size

The maximum Parquet page size supported is for both READ_NOS and WRITE_NOS is 16,776,192 bytes (including LOB and LOB-based UDT columns).

If some of the data in the Parquet record is binary, the maximum number of characters is proportionately reduced.

Foreign Table Limitations

- The following are not supported on foreign tables:
 - Primary indexes
 - Secondary indexes
 - Join indexes
 - Hash indexes
 - Row or column partitioning (except for Parquet foreign tables, which require column partitioning)
 - DML operations that change table contents (INSERT, UPDATE, DELETE)

Parquet Format Limitations

- Certain complex data types are not supported, including STRUCT, MAP, LIST, and ENUM.
- Because support for the STRUCT data type is not available, nested Parquet object stores cannot be processed by Native Object Store.

Bad Values in a Numeric Field

As with any CSV or JSON datasets, a NOS table can become unusable if a newly uploaded Amazon S3 file contains a single bad value in a numeric field. If this happens, you can identify the bad record/field by using TRYCAST. You can first upload new Amazon S3 files to a temporary location in the object storage and use TRYCAST to make sure the numeric data is not corrupted. Then, you can move the data to its permanent location in the object store. See *Teradata Vantage™ - SQL Functions, Expressions, and Predicates*, B035-1145.

For example:

Create an authorization object, if not already done:

```
CREATE AUTHORIZATION MyAuthObj
USER 'YOUR-ACCESS-KEY-ID'
PASSWORD 'YOUR-SECRET-ACCESS-KEY';
```

Create an example table.

```
CREATE MULTISET FOREIGN TABLE bad_numeric_fields
, EXTERNAL SECURITY MyAuthObj
USING (
    LOCATION ('/s3/td-usgs-public.s3.amazonaws.com/DATA-BAD/bad_numeric_data')
);
```

Select:

```
SELECT payload FROM bad_numeric_fields;
```

Result:

Payload

```
-----
{ "site_no":"09396100", "datetime":"2018-07-16 00:00", "Flow":"44.7",
"GageHeight":"1.50", "Precipitation":"0.00", "GageHeight2":"1.50"}
{ "site_no":"09396100", "datetime":"2018-07-14 00:00", "Flow":"232",
"GageHeight":"2.16", "Precipitation":"0.00", "GageHeight2":"2.16"}
{ "site_no":"09396100", "datetime":"2018-07-14 00:14", "Flow":"186",
"GageHeight":"2.05", "Precipitation":"", "GageHeight2":"2.05"}
{ "site_no":"09400812", "datetime":"2018-07-12 00:09", "Flow":"", "GageHeight":"jjjj",
"Precipitation":"bcde", "BatteryVoltage":"12.5"}
{ "site_no":"09400815", "datetime":"2018-07-12 00:00", "Flow":"0.00",
"GageHeight":"-0.01", "Precipitation":"0.00", "BatteryVoltage":""}
{ "site_no":"09400815", "datetime":"2018-07-12 00:07", "Flow":"", "GageHeight":"",
"Precipitation":"", "BatteryVoltage":"12.5"}
```

Select from the table:

```
SELECT payload.site_no, payload.GageHeight(INT) FROM bad_numeric_fields;
```

Result: The query returns an error.

Use TRYCAST to figure out which data is bad:

```
SELECT payload.site_no (CHAR(20)), TRYCAST(payload.GageHeight AS INT)
FROM bad_numeric_fields;
```

Sample result:

Payload.site_no	TRYCAST(Payload.GageHeight)
-----	-----
09396100	1
09396100	2
09396100	2
09400812	?
09400815	0
09400815	0

The results show there is bad data in the record with site_no 09400812.

WRITE_NOS Limitations

- The maximum Parquet page size supported for both READ_NOS and WRITE_NOS is 16,776,192 bytes (including LOB and LOB-based UDT columns). WRITE_NOS can create an unlimited number of these files.
- It is the responsibility of the user to clean up object store files on interrupted write operations. Write operations can be interrupted on transaction aborts or system resets, among other reasons.
- Concurrent WRITE_NOS requests to the same location are likely to cause an error to be returned. If a request gets an error, any objects written by the request are not deleted from external storage and must be manually cleaned up using tools provided by the external object store vendor, such as s3cmd. To avoid this, Teradata recommends specifying a unique location for concurrent requests.
- An error is reported if you attempt to write an object with same path name in the same location.
- Manifest files are not cumulative. If you want to add entries to a manifest, you must create a new manifest that includes the original entries plus the ones you want to add.
- An error is reported if you attempt to write another manifest file in the same location. Use OVERWRITE('TRUE') with MANIFESTONLY('TRUE') keywords to replace a manifest in the same location.

Terminology

Files and Objects

Files and objects can be used interchangeably to describe the components of an object store. Each file or object contains records in which the data itself is held.

Foreign Table

Foreign tables carry information about the location of the external object store, as well as other definitional information, and are the main vehicle within Advanced SQL Engine for reading external data. A foreign table may support access to the entire object store, or a subset of an object store.

Key and Path

A key in object store data is the unique identifier for an object and may contain several logical levels.

Path names are identifiers or pointers to an object; a path name is the entire key.

A path prefix is a subset of the path. For example, if the path of an object is `/td-usgs/bucket/a/b/c/d/object1`, then a path prefix can be any of the following, amongst others:

- `a/b/c`
- `a/b`
- `a/b/c/d/object1`

Object and Objects stores

Objects are the discrete units that compose an object store. Objects can be organized with shared names called prefixes.

Every object has a unique key or path. However, many objects might be identified by or share a common path prefix. For example, `/a/b/c/d` can contain hundreds of objects.

An object store is a collection of related objects, with all participating objects located in the same bucket or container and organized in a hierarchy.

Records and Rows

Row is a relational concept that refers to part of a table.

Record is a logical grouping of values within an object.

For CSV and JSON formatted data, the database will either convert each record into a row, or will convert a record with an array at the top level into a set of rows.

For Parquet formatted data, fields are grouped together (similar to columnar). Then individual field values are converted to a Teradata column value.

Additional Information

Users

Teradata Vantage documentation refers to *users* to indicate who might perform a task we describe. Your organization may use different names for these users.

Use this table to clarify the skill set Teradata assigns to each user.

User	Typical Responsibilities
System Administrator/IT Specialist	<ul style="list-style-type: none"> • Monitors and manages systems and applications • Configures and integrates new technologies • Configures and monitors system security • Performs installation activities
Database Administrator (DBA)	<ul style="list-style-type: none"> • Administers and maintains the Vantage system and connections • Controls optimization, workload, and security • Plans and supports installation activities
Data Scientist	<ul style="list-style-type: none"> • Builds analytic models and solutions • Delivers insights for customer journey • Prepares, moves, and samples data in a secure environment
Business Analyst	<ul style="list-style-type: none"> • Solves critical business issues and measures business value • Studies business requirements and use cases • Converts business needs into IT plans

Teradata Links

Link	Description
https://docs.teradata.com/	Search Teradata Documentation, customize content to your needs, and download PDFs. Customers: Log in to access Orange Books.
https://support.teradata.com	Helpful resources in one place: <ul style="list-style-type: none"> • Support requests • Account management and software downloads • Knowledge base, community, and support policies • Product documentation • Learning resources, including Teradata University
https://www.teradata.com/University/Overview	Teradata education network
https://support.teradata.com/community	Link to Teradata community