7. Data Virtualization and Object Storage

B. WardSQL Server 2022 Revealedhttps://doi.org/10.1007/978-1-4842-8894-8_7

Bob Ward[1]
(1)
North Richland Hills, TX, USA

As far back as SQL Server 7.0, we provided a way to query data *outside* of SQL Server, even for data sources that were not SQL Server. We called this capability linked servers. We used OLE-DB as a mechanism to allow the query processor to push a query to another data source and bring the results back into SQL Server. There were even OLE-DB drivers to query "file" data such as Excel spreadsheet files. In a way, the concept of a linked server query is *data virtualization*. Data virtualization means accessing data "where it lives" vs. copying data into SQL Server. Linked server queries were baked into normal SQL statements like SELECT but also through a new T-SQL function called **OPENROWSET()**. You will see later in this chapter that OPENROWSET() is still alive and well.

In SQL Server 2016, we introduced a concept that was first implemented with Parallel Data Warehouse called **Polybase**. Polybase was brought to Parallel Data Warehouse by David Dewitt and the Microsoft research team. The concept was to use the T-SQL language to execute queries against a Hadoop file system, all from the database engine. In SQL Server 2016, we implemented the Polybase concept with Hadoop calling it Polybase services. Polybase services included separate Windows services that were integrated with the SQL Server engine. Polybase services would take T-SQL requests from the engine and translate them into Hadoop MapReduce jobs using Java. Users accessed files from Hadoop systems through T-SQL statements like CREATE EXTERNAL TABLE and OPENROWSET(). This feature has been removed in SQL Server 2022.

In SQL Server 2019, we expanded the concept of data virtualization and Polybase by adding the ability to use ODBC data sources for accessing data through external tables and OPENROWSET(). We even included built-in *connectors* with drivers like Oracle, MongoDB, Teradata, and SQL Server and *generic* ODBC drivers (in other words, any ODBC driver). SQL Server had truly become a **data hub**. You could use the power of T-SQL and query to just about any data source, without using ETL or copying the data. This functionality continues to exist in SQL Server 2022.

Both of these capabilities were transforming. We realized customers didn't always have their data in SQL Server and didn't want to have to build expensive ETL applications to copy data, often out of date. However, the Polybase design using services outside the engine had its downsides. We started to look for another method to continue the vision of data virtualization but implemented in a different way. We also recognized a trend in the industry for data storage. The trend was the expansion of companies offering object storage using the Simple Storage Service **(S3)** protocol.

Data Virtualization in SQL Server 2022

Back in the early days of project Dallas, I had a meeting with James Rowland-Jones (we all called him JRJ) about what he was working on. He brought up the name *project Gravity*. Project Gravity was effectively *Polybase v3,* as JRJ called it. Version 1 was our original support for Hadoop using Java.

Version 2 was the inclusion of ODBC drivers in SQL Server 2019. JRJ told me project Gravity, version 3, would use REST APIs to access data. One of the biggest targets for this project was object storage providers using S3.

REST, Azure Storage, and S3

Representational state transfer (REST) is an interface for software components to communicate, typically over a remote or Internet connection. REST has been around a long time and uses typical HTTP methods to send and receive data between software components. I think Wikipedia has a good description of REST at https://en.wikipedia.org/wiki/Representational_state_transfer, but also there is a good summary in Azure documentation at https://docs.microsoft.com/rest/api/azure/devops/?view=azure-devops-rest-7.1. In the Azure docs I really like this description of REST: "Representational State Transfer (REST) APIs are service endpoints that support sets of HTTP operations (methods), which provide create, retrieve, update, or delete access to the service's resources."

REST has many benefits. It is lightweight and uniform, uses HTTP(S), and is portable. It pretty much works across all operating systems, platforms, and clouds. I don't always find it intuitive, but once you get used to the use of HTTP *verbs*, it becomes more natural.

One of the benefits of REST is that systems like Azure storage (including Azure Data Lake) and Amazon Simple Storage Service (S3) can be accessed via a known set of REST commands. Instead of requiring an ODBC driver, any client that can submit HTTP(S) to these providers can send and receive data.

Given that we already know in the engine how to send HTTP(S) requests (that is how we back up to URL), we decided to implement the next set of Polybase innovation using REST inside the engine, instead of through Polybase services.

I will admit while Azure Blob Storage and Azure Data Lake Storage support REST, a big motivator to innovate with REST was S3. Simple Storage Service (S3) from Amazon originates as far back as 2006 (https://en.wikipedia.org/wiki/Amazon_S3). One of the things that Amazon did was "open up" the protocol to access S3 *object storage*. S3 is called object storage because objects are files and metadata that describes the files. Objects are collected in *buckets* or containers for files. The core concepts of how object storage, files, and buckets work can be found at https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html#CoreConcepts. Amazon S3 object storage is accessed using a specific set of REST API as documented at https://docs.aws.amazon.com/AmazonS3/latest/userguide/RESTAPI.html. Since Amazon launched S3, several companies have come forward to offer *S3-compatible object storage services*. Provided you follow the S3 REST API protocol, you can set up an object storage endpoint and allow clients to use it. We have seen a proliferation in providers offering S3 object storage. Therefore, we decided to add to our REST API story for Polybase by implementing the ability to query files from S3 object endpoints.

Project Gravity Becomes Polybase v3

With this context in mind, SQL Server 2022 changes the Polybase model entirely. The SQL engine now includes the ability to leverage external data source *connectors* for the following storage providers:

- Azure Blob Storage (**abs**)
- Azure Data Lake Storage Gen2 (**adls**)

- S3 compatible object storage (**s3**)

Note

Azure SQL Managed Instance also supports **abs** and **adls**.

The monikers for each of these connectors are used as part of the syntax for LOCATION when creating an external data source. ODBC driver–based connectors use monikers like **oracle**, **teradata**, **mongodb**, **sqlserver**, and **odbc**.

ODBC driver–based connectors run in the context of Polybase services. REST API connectors run inside the SQL Server engine.

Note

Even though REST API connectors run inside the engine and don't technically require Polybase services, in SQL Server 2022, you still must enable the Polybase feature, which installs these services.

Hugo Queiroz, Senior Program Manager at Microsoft over Polybase, helped me build the following visual to show the connector architecture as seen in Figure 7-1.

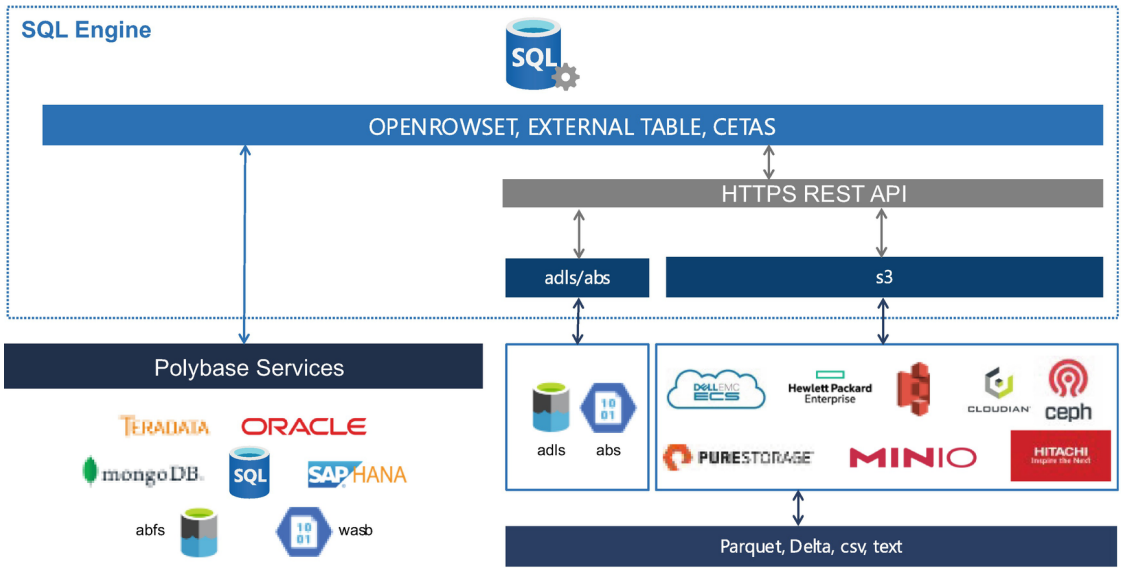

Figure 7-1
Data virtualization in SQL Server 2022

In this figure, T-SQL operations like OPENROWSET and CREATE EXTERNAL TABLE are used to access external data sources from connectors. On the left-hand side of this figure, ODBC connectors are accessed by the engine connecting with Polybase services (which are actual Windows services). On the right-hand side of the figure, the SQL Server engine communicates with **abs**, **adls**, and **s3** connectors through REST APIs to access files that are **text/csv** (you provide the format) or **parquet/delta** (the format is baked into the file). I'll describe more about the types of file formats like parquet and delta in the next section of this chapter.

You might be wondering what is the difference between using abs and adls. Azure Blob Storage (abs) is a general-purpose storage system where Azure Data Lake Storage (adls) is specifically built for data analytic workloads. You can read further a detailed comparison at https://docs.microsoft.com/azure/data-lake-store/data-lake-store-comparison-with-blob-storage. Both Azure Blob Storage and Azure Data Lake Storage are considered object storage systems like S3.

I've already mentioned the T-SQL statements OPENROWSET and CREATE EXTERNAL TABLE. CETAS stands for CREATE EXTERNAL TABLE AS SELECT. The concept comes from Polybase via Azure Synapse and allows you to create an external table as the result of a SELECT statement in SQL Server. While OPENROWSET and EXTERNAL TABLE are "read" operations, CETAS allows you to "export" data from SQL Server into files and record the metadata of an external table.

Polybase v3 File Formats

You probably know what a CSV (comma-delimited) or text (any format you want) file format is, but you may not be familiar with parquet or delta. Let's review these file formats and discuss why they have become popular to use.

Parquet

Parquet files are a formatted file founded by an Apache project that was started in 2013. The Apache project website at https://parquet.apache.org has a simple definition: *"*Apache Parquet is an open source, column-oriented data file format designed for efficient data storage and retrieval." Column-oriented makes parquet an efficient format for analytic workloads, like columnstore indexes in SQL Server. Parquet is a binary formatted file as opposed to CSV or text. It contains metadata inside the file about the columns and data types. Even though parquet uses a binary format, there is documentation on the details, so developers know how to read and write the format. You can read the format details at https://parquet.apache.org/docs/file-format/.

Parquet has become one of the most popular file formats to use for analytics from ground to cloud. Its popularity is one of the reasons we chose to support parquet as a native format for data virtualization in SQL Server 2022. SQL Server 2022 natively supports reading parquet files and writing out SQL data as formatted parquet files.

Delta

Parquet files are static files to read and write as an entire file unit. An open source project was created by several companies called **Delta Lake** to support a *lakehouse* of data based on **delta tables**. Delta tables are effectively a collection of parquet formatted files with *transaction log type* capabilities through JSON files. I found this interesting article by Databricks on the internals of the "transaction log" for delta at https://databricks.com/blog/2019/08/21/diving-into-delta-lake-unpacking-the-transaction-log.html.

You can read about the Delta Lake project and all resources at https://delta.io. Since Delta Lake is a complete open source project (the GitHub project is at https://github.com/delta-io/delta), you can read all the details of how delta tables are formatted, but most users create and use delta tables through interfaces such as Spark. For example, you can see at https://docs.delta.io/latest/delta-batch.html#-ddlcreatetable how to use T-SQL with Spark to create a new delta table.

SQL Server 2022 supports natively reading delta tables, but you cannot export data in SQL Server to delta tables. You can export SQL data to parquet files. Spark offers functionality to convert parquet to delta should you need to do that.

Using parquet files offers no capabilities to use predicate pushdown or push your "filter" to a file to get only the data you need. Delta tables do offer this type of capability through a concept called a *partition column*. You will see in this chapter examples of how to use both parquet and delta with SQL Server 2022.

Using the New Polybase v3

To use Polybase v3 or REST API–based data virtualization, you will use a series of T-SQL statements to create objects as seen in Figure 7-2.

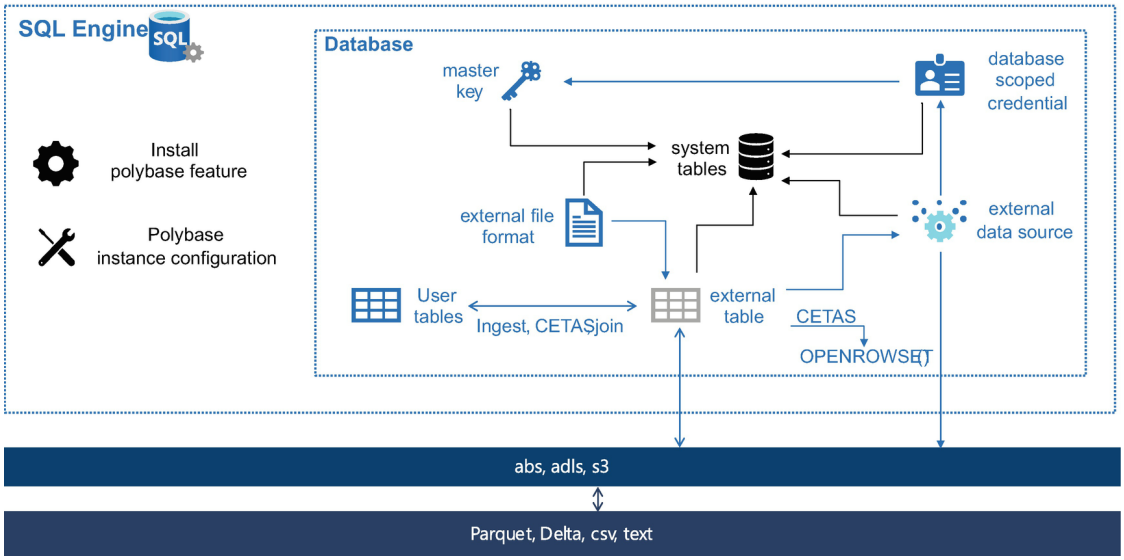

Figure 7-2
Using REST API data virtualization in SQL Server 2022

Let's go through this figure to see how to use REST API data virtualization in SQL Server 2022. You will see an example of each of these steps in the section titled "**Try Out Polybase v3**."

Install and Configure Polybase

Even though REST APIs do not use Polybase services in SQL Server 2022, you must install the Polybase Query Service for External Data feature (or add the feature). You must also enable Polybase setting the **sp_configure** 'polybase enabled' option to 1. If you plan to use CETAS, you will also need to set 'allow polybase export' to 1.

Set Up Credentials and Data Sources

The next step, within the *context of a user database,* is to create a database-scoped credential using the CREATE DATABASE SCOPED CREDENTIAL statement and external data source using the CREATE EXTERNAL DATA SOURCE statement (which requires the credential). You will need to do this for each connector you plan to use. You must first create a master key before you can create a database-scoped credential using the CREATE MASTER KEY statement, but you only need to do this once no matter how many credentials and data sources you want to create in the database. These objects are stored as metadata in system tables in the user database.

Query Files Directly with OPENROWSET( )

With an external data source created, you can directly query files using the OPENROWSET() T-SQL function with the BULK option. File formats such as PARQUET and DELTA don't require any options for column names or types; CSV and text files will require this.

Create an External Table

You can also create an external table using the CREATE EXTERNAL TABLE statement. The external table requires an external data source and can be mapped directly to files for connectors using LOCATION. You must first create a file format, using the CREATE EXTERNAL FILE FORMAT statement, for use with the external table. The file format can be for a "known" format like

ParquetFileFormat or for a CSV file, which will require you to provide more details. The external table can provide all the column names from the source file or a subset. No data is stored for an external table, only metadata in system tables, which is the same for file formats.

You can also use the CREATE EXTERNAL TABLE AS SELECT (CETAS) syntax to create an external table and store the results from a user table in the LOCATION target. It is a method to "export" data to REST API connectors. You can even use CETAS with the source of a query to OPENROWSET().

Once you create an external table, you can query the external table like any other SQL table (and even assign permissions to it). You can join with other user tables or other external tables and execute OPENROWSET() queries. You can use standard T-SQL like INSERT..SELECT or SELECT INTO to take data from external tables and populate user tables (i.e., ingestion).

Try Out Polybase v3

Let's go through some examples to see REST API–based data virtualization using an S3 object provider.

Note

If you do not want to install non-Microsoft software required for this exercise, you can review the results of the exercise in the SQL notebook files **queryparquet.ipynb** and **querydelta.ipynb**, which can be found with the sample files and scripts in the **ch07_datavirt_objectstorage\datavirt\parquet** and **ch07_datavirt_objectstorage\datavirt\delta** folders. You will need to install Azure Data Studio to use these notebooks, or you can view the notebooks in the GitHub repo with a browser.

Prerequisites

To go through this exercise, you will need the following prerequisites:

SQL Server 2022 Evaluation Edition with the database engine and the Polybase Query Service for External Data feature installed.

Note

For Linux, install the Polybase packages at https://docs.microsoft.com/sql/relational-databases/polybase/polybase-linux-setup.

- Virtual machine or computer with minimum of two CPUs with 8Gb RAM.
- SQL Server Management Studio (SSMS). The latest 18.x or 19.x build will work.
- The latest build of Azure Data Studio (ADS). **This is optional**. I have included a SQL notebook with the results of this exercise, so if you do not want to install non-MSFT software, you can review the results with ADS.

Note

The following prerequisites are for non-Microsoft software. The use of this software does not represent any official endorsement from Microsoft. This software is not supported by Microsoft, so any issues using this software are up to the user to resolve.

- The **minio** server for Windows, which you can download at https://min.io/download#/windows. For the demo I assume you have created a directory called **c:\minio** and have downloaded the minio.exe for Windows into that directory.

- **openssl** for Windows, which you can download at https://slproweb.com/products/Win32OpenSSL.html. I chose the Win64 OpenSSL v3.0.5 MSI option. Set the system environment variable OPENSSL_CONF=C:\Program Files\OpenSSL-Win64\bin\openssl.cfg and put c:\Program Files\OpenSSL-Win64\bin in the system path.

Note

This exercise can also work with Linux and containers. You will need to use the minio server for Linux. openssl comes with Linux or is available by installing an optional package. Check your Linux documentation.

A copy of the scripts and files from the GitHub repo of this book, from the **ch07_datavirt_objectstorage\datavirt\parquet** directory.

Set Up minio for the Exercise

To use minio.exe with SQL Server, TLS is required. In order to use TLS, you must have a valid certificate. For testing purposes, we will generate a self-signed certificate.

Follow these steps to set up minio for the exercise:

1. 1.

   Generate a private key by running the following command from the c:\minio directory at the command prompt:

openssl genrsa -out private.key 2048

2. 2.

   Copy the supplied **openssl.conf** file to c:\minio. Edit this file by changing the **IP.2** to your local IP address and **DNS.2** to your local computer name.

3. 3.

   Generate a self-signed certificate by running the following command from the c:\minio directory at the command prompt:
   openssl req -new -x509 -nodes -days 730 -key private.key -out public.crt -config openssl.conf

4. 4.

   For Windows users, double-click the public.crt file and select Install Certificate. Choose Local Machine and then Place all certificates in the following store. Browse and select Trusted Root Certification Authorities.

   Note Self-signed certificates are great for testing but *not* secure. I recommend you delete the certificate via Manage User Certificates in Control Panel when you are finished with this exercise.

5. 5.

   Copy the files **private.key** and **public.crt** from the c:\minio directory into the %%USERPROFILE%%\mino\certs directory.

5. 6.

   From a command prompt, navigate to the c:\minio directory. Then start the minio program by running the following command (cmd.exe users don't need the .\):

.\minio.exe server c:\minio --console-address ":9001"

This program starts and runs until you quit with Ctrl+C. Your output in the command prompt should look something like this:
MinIO Object Storage Server

Copyright: 2015-2022 MinIO, Inc.

License: GNU AGPLv3 <https://www.gnu.org/licenses/agpl-3.0.html>

Version: RELEASE.2022-07-30T05-21-40Z (go1.18.4 windows/amd64)

Status: 1 Online, 0 Offline.

API: https://<local IP>:9000 https://127.0.0.1:9000

RootUser: <user>

RootPass: <password>

Console: https://<local IP>:9001 https://127.0.0.1:9001

RootUser: <user>

RootPass: <password>

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide

  $ mc.exe alias set myminio https://<local IP>:9000 <user> <password>

Documentation: https://docs.min.io

1. 7.

   Test your connection to minio by using a web browser on the local computer for https://127.0.0.1:9001. You should be presented with a login screen like Figure 7-3. Use the RootUser and RootPass from the preceding minio server output.

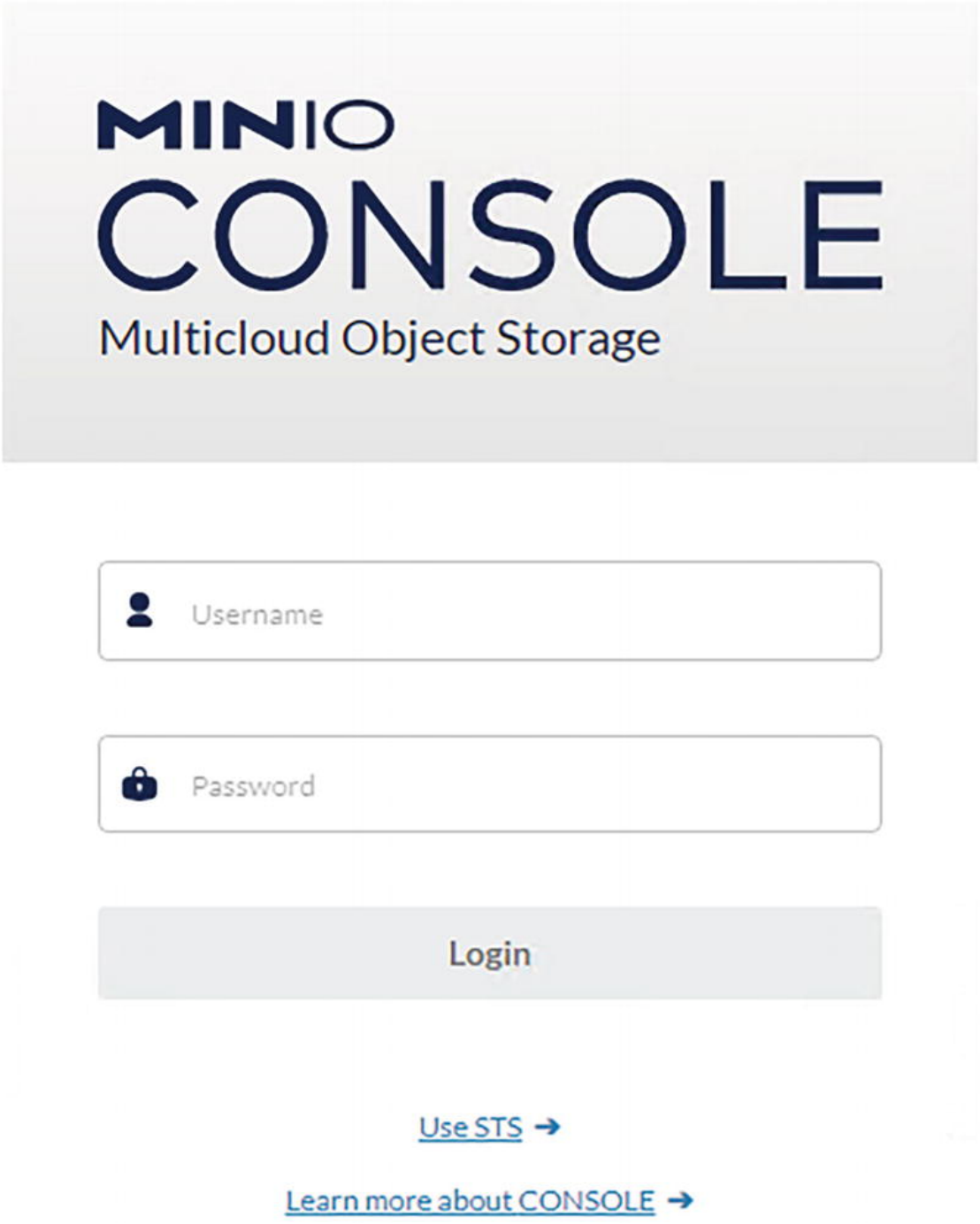

Figure 7-3
The minio console login screen

1. 8.

   On the left-hand side menu, click Identity and Users. Select Create User. Create a user and password. Select the readwrite policy for the user. Note down this user and password. This is the user and password that you will use for the **SECRET** value in the **creates3creds.sql** script later in this exercise.

2. 9.

   Select the menu option for Buckets. Select Create Bucket. Use a bucket name of **wwi**. Leave all the defaults and click Create Bucket. Your screen should now look like Figure 7-4.

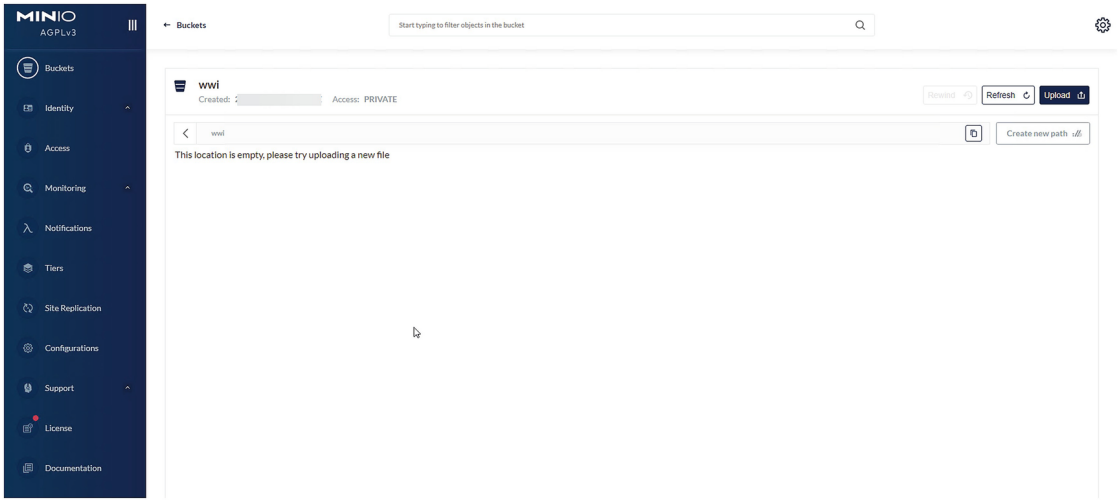

Figure 7-4
A new bucket in S3 storage

Learn to Use REST API to Access Parquet Files on S3

With an S3 provider ready to use, we can now access S3 compatible storage with SQL Server. Follow these steps for the exercise for basics of REST API data virtualization to access parquet files on an S3 object provider. The next section has an exercise to access delta files.

1. 1.

   Copy the **WideWorldImporters** sample database from https://aka.ms/WideWorldImporters to a local directory (the restore script assumes c:\sql_sample_databases).

2. 2.

   Edit the **restorewwi.sql** script to specify the correct path for the backup and where data and log files should go.

3. 3.

   Execute the script **restorewwi.sql.** This script executes the following T-SQL statements:
   USE master;

   GO

   DROP DATABASE IF EXISTS WideWorldImporters;

   GO

   RESTORE DATABASE WideWorldImporters FROM DISK =
   'c:\sql_sample_databases\WideWorldImporters-Full.bak' with

   MOVE 'WWI_Primary' TO 'c:\sql_sample_databases\WideWorldImporters.mdf',

   MOVE 'WWI_UserData' TO 'c:\sql_sample_databases\WideWorldImporters_UserData.ndf',

   MOVE 'WWI_Log' TO 'c:\sql_sample_databases\WideWorldImporters.ldf',

   MOVE 'WWI_InMemory_Data_1' TO
   'c:\sql_sample_databases\WideWorldImporters_InMemory_Data_1',

   stats=5;

   GO

   ALTER DATABASE WideWorldImporters SET QUERY_STORE CLEAR ALL;

   GO

4. 4.

   Execute the script **enablepolybase.sql** to configure instance-level settings to allow Polybase features to execute and export data from Polybase to S3. This script executes the following T-SQL statements:
   EXEC sp_configure 'polybase enabled', 1;

   GO

   RECONFIGURE;

   GO

   EXEC sp_configure 'allow polybase export', 1;

   GO

   RECONFIGURE;

   GO

5. 5.

   Edit the script **createmasterkey.sql** to put in a password. Execute the script to create a master key to protect the database-scoped credential. This script executes the following T-SQL statements:
   USE [WideWorldImporters]

   GO

   IF NOT EXISTS (SELECT * FROM sys.symmetric_keys WHERE name = "##MS_DatabaseMasterKey##")

   CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<password>';

   GO

6. 6.

   Edit the script **creates3creds.sql** to put in the user and password you created in the minio console for SECRET. Execute the script to create a database-scoped credential. This script executes the following T-SQL statements:
   IF EXISTS (SELECT * FROM sys.database_scoped_credentials WHERE name = 's3_wwi_cred')

   DROP DATABASE SCOPED CREDENTIAL s3_wwi_cred;

   GO

   CREATE DATABASE SCOPED CREDENTIAL s3_wwi_cred

   WITH IDENTITY = 'S3 Access Key',

   SECRET = '<user>:<password>';

   GO

7. 7.

   Edit the script **creates3datasource.sql** to substitute in your local IP address for the minio server. Execute the script to create an external data source. This script executes the following T-SQL statements:
   IF EXISTS (SELECT * FROM sys.external_data_sources WHERE name = 's3_wwi')

   DROP EXTERNAL DATA SOURCE s3_wwi;

   GO

   CREATE EXTERNAL DATA SOURCE s3_wwi

   WITH

   (

   LOCATION = 's3://<your local IP>:9000'

   ,CREDENTIAL = s3_wwi_cred

   );

   GO

   Tip You could put in the bucket name here after the port number so that the data source is focused only on a specific bucket. We will instead use the bucket when specifying location for queries.

8. 8.

   Create a file format for parquet by executing the script **createparquetfileformat.sql**. This script executes the following T-SQL statements:
   USE [WideWorldImporters];

   GO

   IF EXISTS (SELECT * FROM sys.external_file_formats WHERE name = 'ParquetFileFormat')

   DROP EXTERNAL FILE FORMAT ParquetFileFormat;

   CREATE EXTERNAL FILE FORMAT ParquetFileFormat WITH(FORMAT_TYPE = PARQUET);

   GO

9. 9.

Let’s first learn how to export data in parquet format to the wwi bucket by executing the script **wwi_cetas.sql**. In SSMS, select the Include Actual Execution Plan option. This script executes the following T-SQL statements:

```
USE [WideWorldImporters];

GO

IF OBJECT_ID('wwi_customer_transactions', 'U') IS NOT NULL

    DROP EXTERNAL TABLE wwi_customer_transactions;

GO

CREATE EXTERNAL TABLE wwi_customer_transactions

WITH (

   LOCATION = '/wwi/',

   DATA_SOURCE = s3_wwi,

   FILE_FORMAT = ParquetFileFormat

)

AS

SELECT * FROM Sales.CustomerTransactions;

GO
```

Let’s examine this script. The LOCATION is the bucket name. The LOCATION value can be a bucket name or even a series of folders and even a specific filename. In this case, because the external data source didn’t specify a specific bucket, I need to specify that here. Since I only specified the bucket, the query will place all files in the parent folder. There is no predicate pushdown with parquet files and REST API connectors. Any filtering is done inside SQL Server (even if you specify a WHERE clause).

The DATA_SOURCE is the external data source we created to connect to the minio server. The file format matches up to parquet. Notice we do not have to specify anything about the format other than just parquet. We can also include column names with this definition, but we don’t need to. Using the AS SELECT part of the statement will create a single parquet file with column names, types, and data.

In the Messages tab, you should see 97147 rows affected. In the Execution Plan tab you should see a plan like Figure 7-5.

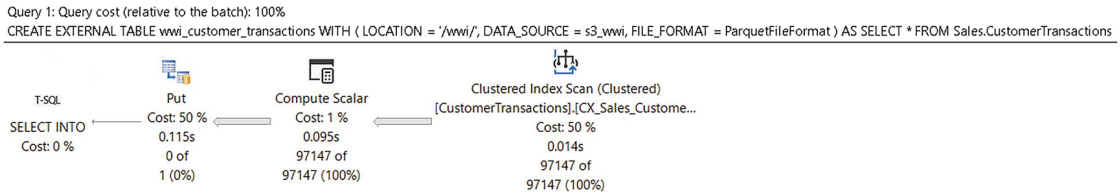

Figure 7-5
Query execution plan for CETAS

Notice the **Put** operator, which is used to export data to the s3 connector as a parquet file.

Tip

One observation I have had using CETAS is there is no “duplicate” detection. If you run the preceding command twice, two parquet files will be created.

1. 10.

   Use the minio console to browse the wwi bucket and confirm the parquet file has been added. Your results should look like Figure 7-6.

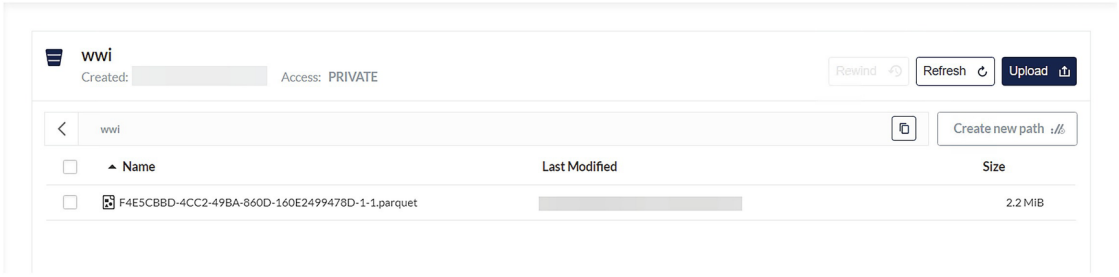

Figure 7-6
Parquet file added to S3 bucket through CETAS

1. 11.

   Query the new external table by executing the script **querywwiexternaldata.sql**. This script executes the following T-SQL statements:
   USE [WideWorldImporters];

   GO

   SELECT c.CustomerName, SUM(wct.OutstandingBalance) as total_balance

   FROM wwi_customer_transactions wct

   JOIN Sales.Customers c

   ON wct.CustomerID = c.CustomerID

   GROUP BY c.CustomerName

   ORDER BY total_balance DESC;

   GO

   Notice in this example the script joins the external table (stored in parquet in S3) with a local table in the user database. Your results should display 263 rows. One common use case for an example like this is to build a VIEW on top of this query and only give users access to the view. They will be *abstracted* from the source of the data, whether it be in SQL Server or a parquet file.

2. 12.

   Use SSMS with Object Explorer to see the column definitions for the table created through CETAS. Your results should look like Figure 7-7.

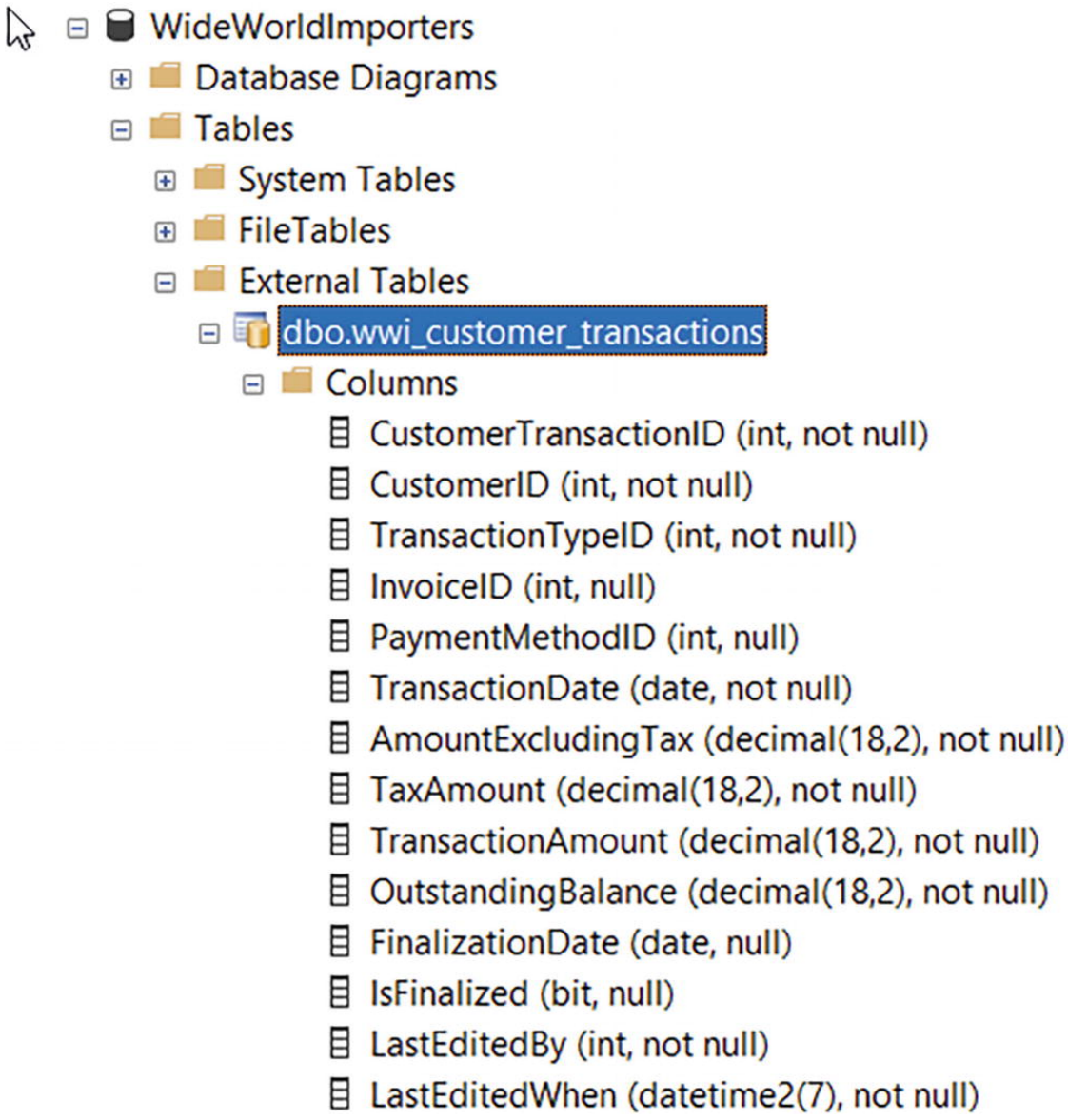

Figure 7-7
Columns created automatically with parquet through CETAS

Notice the column names and types match the original table in the database; this demonstrates the power of parquet including metadata.

1. 13.

   Let’s use OPENROWSET() to run an “ad hoc” query from the generated parquet file stored in S3 by executing the script **querybyopenrowset.sql**. In SSMS, select the Include Actual Execution Plan option. This script executes the following T-SQL statements:
   USE [WideWorldImporters];

   GO

   SELECT *

   FROM OPENROWSET

   (BULK '/wwi/'

   , FORMAT = 'PARQUET'

   , DATA_SOURCE = 's3_wwi')

   as [wwi_customer_transactions_file];

   GO

   It turns out the BULK option is not just about the BULK insert of data; it can also be used to “open a rowset” on a file.

   In the Messages tab you should see 97147 rows affected, which is the same number of rows in the original table, Sales.CustomerTransactions. If you look at the Execution Plan tab, you should see a query execution plan like Figure 7-8.

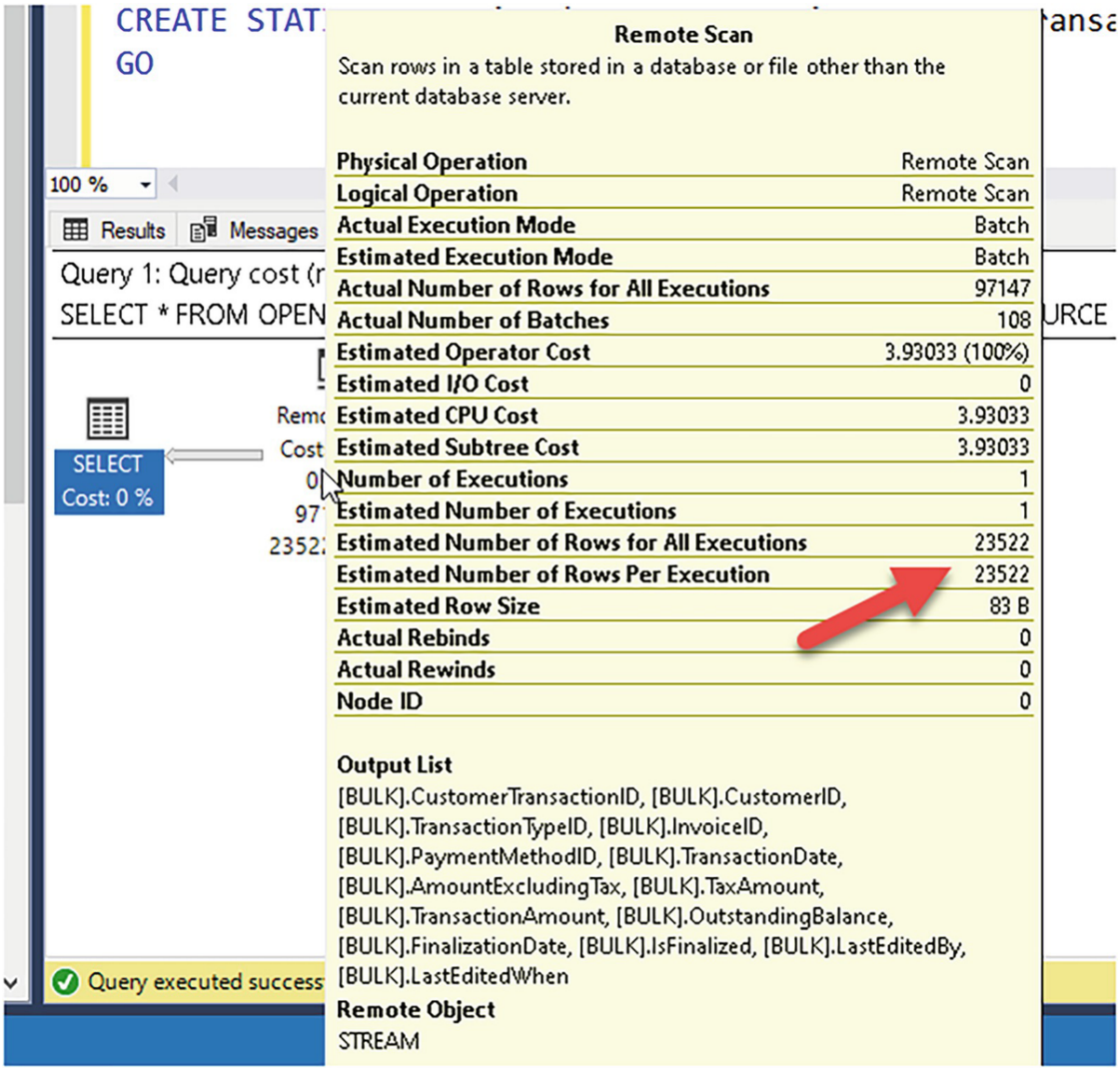

Figure 7-8
Query execution plan for OPENROWSET()

Notice that the estimated number of rows is roughly 30% of the total actual rows. This is because the optimizer doesn't have any statistics from the parquet file itself. This is opposite from **wwi_customer_transactions**, which was created as an external table. You can manually create statistics on an external table, or if you have the database option AUTO_CREATE_STATISTICS set to ON (the default), statistics will be automatically created when you query the external table. The estimates will be accurate if these statistics are available. OPENROWSET() query performance may or may not be adversely affected without statistics.

As I've mentioned earlier in the chapter, there is no predicate pushdown for parquet files. If you add a WHERE clause to the preceding SQL statement

```
USE [WideWorldImporters];

GO

SELECT *

FROM OPENROWSET

    (BULK '/wwi/'

    , FORMAT = 'PARQUET'

    , DATA_SOURCE = 's3_wwi')
```

as [wwi_customer_transactions_file]

WHERE wwi_customer_transactions_file.CustomerTransactionID = 2;

the query processor in the engine must bring the results of the parquet file(s) in this bucket parent folder and then filter within the engine to produce the final result.

You can help reduce the amount of data that SQL Server has to read and filter by partitioning your parquet files with a folder structure within the bucket and then specify only the folder you need with the BULK syntax or LOCATION for external tables.

1. 14.

   Another nice feature for external tables is the ability to create one based on a subset of columns in the parquet file. Execute the script **querybyexternaltable.sql** to see an example of this. This script executes the following T-SQL statements:

```
IF OBJECT_ID('wwi_customer_transactions_base', 'U') IS NOT NULL
    DROP EXTERNAL TABLE wwi_customer_transactions_base;
GO
CREATE EXTERNAL TABLE wwi_customer_transactions_base
(
    CustomerTransactionID int,
    CustomerID int,
    TransactionTypeID int,
    TransactionDate date,
    TransactionAmount decimal(18,2)
)
WITH
(
    LOCATION = '/wwi/'
  , FILE_FORMAT = ParquetFileFormat
  , DATA_SOURCE = s3_wwi
);
GO
SELECT * FROM wwi_customer_transactions_base;
GO
```

   You will see the same number of rows from the wwi_customer_transactions external table, but only with the columns you specified. The columns don’t have to be in any specific order.

2. 15.

   As I mentioned earlier, you can manually create statistics on any number of columns in an external table. Execute the script **creatstats.sql** for an example. This script executes the following T-SQL statements:
   USE WideWorldImporters;

   GO

   CREATE STATISTICS wwi_ctb_stats ON wwi_customer_transactions_base (CustomerID)
   WITH FULLSCAN;

   GO

   The statistics are stored in user database system tables.

3. 16.

   I previously mentioned in this chapter that many of the objects for data virtualization are stored in system tables. Execute the script **exploremetadata.sql** to see what data is stored for external data sources, file formats, and external tables. This script executes the following T-SQL statements:
   USE [WideWorldImporters];

   GO

   SELECT * FROM sys.external_data_sources;

   GO

   SELECT * FROM sys.external_file_formats;

   GO

   SELECT * FROM sys.external_tables;

   GO

4. 17.

   SQL Server also provides system procedures to explore metadata in files like parquet. Execute the script **getparquetmetadata.sql** to see an example. This script executes the following T-SQL statements:
   EXEC sp_describe_first_result_set N'

   SELECT *

   FROM OPENROWSET

   (BULK "/wwi/"

   , FORMAT = "PARQUET"

   , DATA_SOURCE = "s3_wwi")

   as [wwi_customer_transactions_file];';

   GO

5. 18.

   OPENROWSET also provides a nice feature to gather metadata about the source parquet files. Execute the script **getfilemetadata.sql** to see an example. The script executes the following T-SQL statements:
   USE [WideWorldImporters];

   GO

   SELECT TOP 1 wwi_customer_transactions_file.filepath(),

   wwi_customer_transactions_file.filename()

   FROM OPENROWSET

   (BULK '/wwi/'

   , FORMAT = 'PARQUET'

   , DATA_SOURCE = 's3_wwi')

   as [wwi_customer_transactions_file];

   GO

Learn How to Use Delta Files

This next exercise assumes you have completed all the prerequisites for the previous exercise with parquet. Furthermore, it assumes you have completed all the steps in the previous exercise to restore the WideWorldImporters database, configure Polybase, and create a master key, database-scoped credential, external data source, and external file format. You can leave the previous wwi bucket. You will create a new bucket as part of this exercise.

All *new* scripts and files specifically for this exercise can be found in the **ch07_datavirt_objectstorage\datavirt\delta** folder**.** If you don't want to go through the exercise to use minio for delta, you can view the results in the notebook **querydelta.ipynb**.

Note

The people-10m delta table is a sample delta table from a sample dataset from Databricks as found at https://docs.microsoft.com/azure/databricks/data/databricks-datasets#sql. This dataset contains names, birthdates, and SSN, which are all fictional and don't represent actual people. This dataset falls under the Creative Commons license at http://creativecommons.org/licenses/by/4.0/legalcode and can be shared and provided in this book. My colleague Hugo Queiroz wrote a Spark job to grab the delta table from this dataset and downloaded it into all the files that make up the delta table. Hugo also provided many of the queries and ideas for this exercise. I could not have written this chapter without his help!

1. 1.

   Use the minio console to create a new bucket called **delta**. Use the Upload folder option in the console to upload the folder from the example files called **people-10m**. This will create a folder under the delta bucket called people-10m with several parquet files and a folder called **_delta_log**.

2. 2.

   Query the delta table by executing the script **querydeltatable.sql**. This script executes the following T-SQL statements:
   USE [WideWorldImporters];

   GO

   SELECT * FROM OPENROWSET

   (BULK '/delta/people-10m',

   FORMAT = 'DELTA', DATA_SOURCE = 's3_wwi') as [people];

   GO

   There are 10 million rows in the delta table spread across all the parquet formatted files. It will take around 1 minute to bring back all the results. Pay attention to the various columns in the table. Most of the query execution time is spent displaying all the rows in SSMS, not actually reading the data.

3. 3.

   By default, the delta table was partitioned by the **id** column. Let's run a query to filter rows on a column that is not partitioned. Execute the script **querybyssn.sql**. This script executes the following T-SQL statements:
   USE [WideWorldImporters];

   GO

   SELECT * FROM OPENROWSET

   (BULK '/delta/people-10m',

   FORMAT = 'DELTA', DATA_SOURCE = 's3_wwi') as [people]

   WHERE [people].ssn = '992-28-8780';

   GO

   This query only returns one row but takes around 4 seconds. SQL Server had to read all 10 million rows from the delta table, but it used a Filter operator to only return the one row based on **ssn**.

4. 4.

   Now let's run a query and filter by the id column. Execute the script **querybyid.sql**. This script executes the following T-SQL statements:
   USE [WideWorldImporters];

   GO

   SELECT * FROM OPENROWSET

   (BULK '/delta/people-10m',

   FORMAT = 'DELTA', DATA_SOURCE = 's3_wwi') as [people]

   WHERE [people].id = 10000000;

   GO

   I specifically chose an id value that is "at the end" of one of the files. Notice the query comes back in ~1 second. That is because the delta table is partitioned by the id column; it is a form of predicate pushdown that can make queries faster.

   Note When you look at a query execution plan, the optimizer does not know the partition column for delta. The power is within the REST API calls to get the data from the delta table. Therefore, you may see a Filter operator after a remote scan. But only one row is returned from the REST API calls, so the Filter operator is not indicative of any performance issues. In the future, we plan to investigate how to make the optimizer more efficient in conjunction with delta-partitioned columns.

5. 5.

   One of the concepts I mentioned with CETAS is the ability to create an external table based on an OPENROWSET() query. Let's look at an example and at the same time use a folder within S3 to narrow what data to query.

6. 6.

   Create a new set of parquet files as an external table in a new folder for a subset of the delta table by executing the script **createparquetfromdelta.sql**. This script executes the following T-SQL statements:

```
USE [WideworldImporters];

GO

IF EXISTS (SELECT * FROM sys.objects WHERE NAME = 'PEOPLE10M_60s')

    DROP EXTERNAL TABLE PEOPLE10M_60s;

GO

CREATE EXTERNAL TABLE PEOPLE10M_60s

WITH

(   LOCATION = '/delta/1960s',

   DATA_SOURCE = s3_wwi,

   FILE_FORMAT = ParquetFileFormat)

AS

SELECT * FROM OPENROWSET

(BULK '/delta/people-10m', FORMAT = 'DELTA', DATA_SOURCE = 's3_wwi') as [people]

WHERE YEAR(people.birthDate) > 1959 AND YEAR(people.birthDate) < 1970;

GO
```

   This script will create a new folder called 1960s under the delta bucket. It will generate parquet files only for people in the delta table who were born in the 1960s (chosen because that is my birth decade).

   To be clear about the flow here, SQL Server executes the OPENROWSET() query against the delta table to bring back all 10 million rows, but then filters out rows based on the WHERE clause. It takes those results and creates a set of parquet files in the /delta/1960s folder and stores the metadata about the table and columns in system tables.

7. 7.

   Use the minio console to browse the **1960s** folder under the delta bucket to see the new parquet files created.

8. 8.

   Query the new external table by executing the script **query1960speople.sql**. This script executes the following T-SQL statements:
   USE [WideWorldImporters];

   GO

   SELECT * FROM PEOPLE10M_60s

   ORDER BY birthDate;

   GO

You have now seen several examples of how to query parquet files and delta tables using the new REST API interfaces for S3. You have also seen some interesting ways to export SQL data or results of OPENROWSET queries into parquet formatted files.

What Else Do You Need to Know

Going through these exercises has allowed you to see various aspects of using Polybase v3 with new REST API connectors.

There are a few details that are worth noting as you consider using this innovative technology:

- There are other options I didn't show you for CREATE EXTERNAL TABLE like "reject options" to handle data that doesn't match columns and data types. You can get a complete set of options and limitations when using external tables at https://docs.microsoft.com/sql/t-sql/statements/create-external-table-transact-sql.
- There are some limits when using S3 compatible storage such as the number of files. You can read the complete set of limits and requirements for using S3 at https://docs.microsoft.com/sql/relational-databases/polybase/polybase-configure-s3-compatible.
- The focus of the exercises was how to use S3. You can see an example of how to use delta tables on **abs** and **adls** at https://docs.microsoft.com/sql/relational-databases/polybase/virtualize-delta.

Backup and Restore with S3 Compatible Object Storage

A long-standing best practice for SQL Server is to separate the storage of backups from your data. A very long time ago, customers would put backups in separate drives, network file shares, or tapes (yikes!). Then SANs came along, and separate SAN storage could be used.

With the innovation of Azure, SQL Server embraced the ability to store backups in the cloud by writing directly to Azure storage starting in SQL Server 2012. We enhanced the BACKUP and RESTORE T-SQL syntax to support the URL option, and we still support this today. you can see all the details about how to use Azure storage as offsite storage at https://docs.microsoft.com/sql/relational-databases/backup-restore/sql-server-backup-to-url. Inside the SQL engine, we use REST API calls through HTTPS to send and receive data for backups to Azure.

And since we have figured out how to use REST APIs to support S3 object storage for data virtualization, why not just enhance the storage engine to support a database backup and restore to any S3-compatible storage?

How Does It Work?

When you back up or restore a database, you have the option of using the URL syntax for a backup:
BACKUP DATABASE WideWorldImporters

TO URL =
'https://<mystorageaccountname>.blob.core.windows.net/<mycontainername>/WideWorldImporters.bak';

SQL Server will take the data from the data file, and instead of streaming this to a local disk, it will stream the data over REST API calls to a .bak file in Azure Blob Storage. Notice the **https** keyword.

This is all completely supported in SQL Server 2022. For S3, we simply extended the URL syntax to support S3:
BACKUP DATABASE WideWorldImporters

TO URL = 's3://<endpoint>:<port>/<bucket>/wwi.bak'

If the engine detects the keyword s3 as part of the URL syntax, it will use code built to support the S3 protocol to send data to an S3-compatible object storage provider.

What is great about this feature is that just about anything you can do with BACKUP and RESTORE works with S3, just like backup and restore to Azure storage. Full database, differential, log, and file backups are all supported. And most of the RESTORE options work as well including VERIFYONLY, HEADERONLY, and FILELISTONLY.

There are some limits and caveats to consider, such as maximum size, and some backup options are not supported. One of the biggest factors to consider is that the S3 object provider will store the backup file internally into multiple parts, which then affects the maximum backup size. But there are options like MAXTRANSFERSIZE to help. Read more about parts, file sizes, and limitations at https://docs.microsoft.com/sql/relational-databases/backup-restore/sql-server-backup-to-url-s3-compatible-object-storage.

Let’s Look at an Example

Using the same minio server setup (including the restore of WideWorldImporters) with the data virtualization example in this chapter, use scripts from **ch07_datavirt_objectstorage\s3objectstorage** to see a basic backup and restore with S3. You do not have to configure Polybase to use backup and restore with S3.

If you have not set up minio and just want to see the results of this exercise, you can use the Azure Data Studio notebook **backuprestores3.ipynb**.

1. 1.

    Use the minio console to create a new bucket called **backups**.

2. 2.

Edit the script **creates3creds.sql** to enter your local IP, user, and password (without the <>) from the minio setup earlier in this chapter. Create a credential for the backup by executing the script **creates3creds.sql.** This script executes the following T-SQL statements:
USE MASTER

GO

CREATE CREDENTIAL [s3://<local IP>:9000/backups]

WITH IDENTITY = 'S3 Access Key',

SECRET = '<user>:<password>';

GO

You cannot use the same database-scoped credential that was used in the data virtualization example. This credential is specifically created against the S3 storage and is created in the context of the master database.

3. 3.

   Edit the script **backupdbtos3.sql** and enter in your local IP in all instances. You can review basic BACKUP commands to S3 by executing the script **backupdbtos3.sql**. This script executes the following T-SQL statements:
   USE MASTER;

   GO

   ALTER DATABASE WideWorldImporters SET RECOVERY FULL;

   GO

   BACKUP DATABASE WideWorldImporters

   TO URL = 's3://<local IP>:9000/backups/wwi.bak'

   WITH CHECKSUM, INIT, FORMAT;

   GO

   BACKUP DATABASE WideWorldImporters

   TO URL = 's3://<local IP>:9000/backups/wwidiff.bak'

   WITH CHECKSUM, INIT, FORMAT, DIFFERENTIAL

   GO

   BACKUP LOG WideWorldImporters

   TO URL = 's3://<local IP>:9000/backups/wwilog.bak'

   WITH CHECKSUM, INIT, FORMAT

   GO

   BACKUP DATABASE WideWorldImporters

   FILE = 'WWI_UserData'

   TO URL = 's3://<local IP>:9000/backups/wwiuserdatafile.bak'

   WITH CHECKSUM, INIT, FORMAT;

   GO

4. 4.

   Use the minio console to browse the **backups** bucket to see the new backup files stored in S3.

   Tip Another way to look at files for any S3-compatible storage is to use the free tool S3 Browser at https://s3browser.com.

5. 5.

   Edit the script **restoredbfroms3.sql** to enter your local IP in all instances. Let's restore the full database backup to another database name by executing the script **restoredbfroms3.sql**. This script executes the following T-SQL statements:
   USE MASTER;

   GO

   RESTORE VERIFYONLY FROM URL = 's3://<local IP>:9000/backups/wwi.bak';

   GO

   RESTORE HEADERONLY FROM URL = 's3://<local IP>:9000/backups/wwi.bak';

   GO

   RESTORE FILELISTONLY FROM URL = 's3://<local IP>:9000/backups/wwi.bak';

   GO

   DROP DATABASE IF EXISTS WideWorldImporters2;

   GO

   RESTORE DATABASE WideWorldImporters2

   FROM URL = 's3://<local IP>:9000/backups/wwi.bak'

   WITH MOVE 'WWI_Primary' TO 'c:\sql_sample_databases\WideWorldImporters2.mdf',

   MOVE 'WWI_UserData' TO 'c:\sql_sample_databases\WideWorldImporters2_UserData.ndf',

   MOVE 'WWI_Log' TO 'c:\sql_sample_databases\WideWorldImporters2.ldf',

   MOVE 'WWI_InMemory_Data_1' TO
   'c:\sql_sample_databases\WideWorldImporters2_InMemory_Data_1';

   GO

6. 6.

   Since this is implemented in the engine, all the standard backup and restore history exists in the **msdb** database such as **backupmediafamily**, **backupset**, and **restorehistory**. You can browse these system tables to see the results from this exercise.

Migration from AWS

In the spring of 2022, as I was studying the concept of backup and restore with S3 object storage, something dawned on me on an afternoon walk (I am one of those strange people who comes up with ideas about SQL Server while I walk and exercise).

If AWS invented S3 and SQL Server 2022 supports restoring a database from S3, then (wait for it)…I should be able to migrate a SQL Server database from AWS into SQL Server using the new RESTORE from S3, right?

Right. I did it. I don't have a specific exercise for you, but I'll share the steps I took:

- I deployed an AWS RDS SQL Server database using a version of SQL Server 2019.
- I created an S3 bucket in AWS for storage. I used the following documentation from AWS to do this: https://docs.aws.amazon.com/AmazonS3/latest/userguide/setting-up-s3.html.
- I enabled on-demand backup/restore for AWS RDS using the following instructions: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.SQLServer.Options.BackupRestore.html.
- I created a new database in AWS RDS and backed it up to the S3 bucket.
- I obtained the AWS Access Key ID and Secret Key for the S3 storage.
- I created a credential against the S3 bucket using a T-SQL statement against SQL Server 2022 (my S3 URL is based on the region where my S3 bucket was created):

```
IF EXISTS (SELECT * FROM sys.credentials WHERE [name] = 's3://s3.us-east-1.amazonaws.com/bwsqlbackups')

BEGIN

    DROP CREDENTIAL [s3://s3.us-east-1.amazonaws.com/<bucket>];

END

GO

CREATE CREDENTIAL   [s3://s3.us-east-1.amazonaws.com/<bucket>]

WITH

    IDENTITY    = 'S3 Access Key'

,   SECRET      = '<Access Key ID>:<Secret Key>;

GO
```

I then ran the following T-SQL statement against SQL Server 2022 to restore the database:

```
RESTORE DATABASE <db> FROM URL = 's3://s3.us-east-1.amazonaws.com/<bucket>/<backupfile>.bak'
```

That was it. I completed a very simple offline migration by restoring a full backup from AWS RDS into SQL Server 2022. Pretty cool, right?

SQL Server Is a Data Hub

I made the claim with SQL Server 2019 that SQL Server had become a data hub. I have to admit I borrowed that term from my Vice President Rohan Kumar. I remember him using this term as we talked about SQL Server and its future capability to connect to just about any data source *without moving the data*. That is, in a nutshell, the concept of data virtualization. Point your application to SQL Server using T-SQL, and the power of the SQL engine will access just about any data source for any type of data. Our innovations in SQL Server 2022 expand this capability by connecting with REST API connectors including S3 object storage. We also added innovations to allow SQL backups to be stored and restored with S3 object storage. I asked Hugo Queiroz to sum up his thoughts about

data virtualization and SQL Server 2022. Hugo said, *“One of our major goals for SQL Server 2022 is to make Data Virtualization more flexible and* easier for everyone to use. Less dependencies, less complexity, less code, and a broader range of supported file types and connectors. To make this happen changing to a Rest-API implementation was fundamental, it allows SQL Server 2022 to be used as a data hub and leverage in-database analytics*.”*