**DP 203 Manual**

**Design and implement security changes**

**Access Keys**

More any cloud platform, security is very important. For example, for adls, access should not be available to everyone. There are different ways that access can be given to adls. Go to access keys in adls.

A screenshot of a computer

Description automatically generated

Anyone that has key 1 or key 2 for adls, that person can create tables, queues, and containers in adls. If someone gets hold of key 1 and key 2, they can access all of the data in adls. The reason why there are 2 keys is a security reason. For example, if an application is using key 1 to access adls, and the key gets compromised, the application can quickly use key 2 without being impacted. The rotate key option is on the screen in case this happens, which changes key 1, stopping hackers from accessing adls. When we are mounting Databricks to adls, we use access keys as one of the parameters.

**Azure storage explorer**

Allows you to work with adls and it is a client tool. If clients only need to interacts with adls, it is best if clients or users interacts with adls through the azure storage explorer. You can use the connection string, shared access signature, or the account name and key (found in access keys in adls) to connect adls to azure storage explorer.

**Shared Access Signature**

A shared access signature can be used to give access to adls. The only difference between this and the access key is that you can fine-tune the access you are giving. For example, in the image below, you can choose to give access to the containers, files, queues, or tables in adls, and there are more options. This was not possible through the access key. You can generate the SAS and connection string which can be used by other azure resources or applications to connect to.

A screenshot of a computer

Description automatically generated

You can connect adls to storage explorer with the use of the SAS.

**Azure Active Directory (Role-based-access-control)**

You can create users and give access through role assignments for members such as users, serviceprincipals, and groups (such as security groups). This allows users/groups/service principal to read, write, and execute commands.

A screenshot of a computer

Description automatically generated

**Access Control Lists**

When you get access through role-based-access control, it will give access to every container in the storage account.Lets say you want to give the user access to only particular containers. You can use access control lists to do this. You go to a particular container in adls and you click on manage ACL.

A screenshot of a computer

Description automatically generated

Here you can add a security principal by giving a user read, write, or execute access to a container. In this manner, you can remove write, read, or execute access to specific containers.

A screenshot of a computer

Description automatically generated

So, I may give access to a user using access key to all data in the adls. However, I can then go into Manage ACL for individual containers that I don’t want the user to have access to.

**Secret Scope**

This allows you to put the azure key vault credentials into databricks so that keys are not shown in the notebook.

**Managed Identities**

We know that we can create users in Azure Entra ID, and then use Role-based-access control to give them access to resources. For example, if I am creating a linked service from azure data factory to adls, I can use the system assigned managed identity as an authentication type. This will work, as Azure Entra Id creates a managed entity id when a resource is created. This service principal then needs to have role-based access control assigned in IAM. You will then be able to use the system assigned managed identity as an authentication method.

**Network and Firewall**

When users, service principals or groups need to connect to a resource such as adls, they need access. However, there is also another mechanism known as the firewall which is in charge of giving access to certain ip addresses within a virtual network. You can allow public IP access to the adls when creating adls. However, if you have a private ip option, you can control the networks that have access to the adls. Virtual machines may be hosted in a virtual network or other resources may be hosted in a virtual network. You can make use of the virtual network service endpoint which only allows connections from the resources in the virtual network to have access to adls. This adds an extra layer of security.

A screenshot of a computer

Description automatically generated

Customer Managed Keys Data Factory Keys

So in data factory, you can go to customer managed keys in the manage tab. A customer managed key can only be configured for an empty data factory.

As of my last knowledge update in January 2022, Azure Key Vault supports customer-managed keys, also known as bring-your-own-key (BYOK) or customer-managed keys (CMK). This allows you to import or generate keys in Hardware Security Modules (HSMs) and use them to protect and manage your data in Azure Key Vault. Here's a brief overview of customer-managed keys in Azure Key Vault:

Key Import: You can import your own keys (symmetric or asymmetric) into Azure Key Vault using the Import Key operation. This enables you to bring existing keys into the Key Vault ecosystem.

A screenshot of a computer

Description automatically generated

Azure Synapse Encryption

So, data needs to be encrypted for security reasons. Although data is stored on disk at a Microsoft data centre, encrypting the data means that if a hacker gets hold of it, it is difficult to read. When data sits in resources like synapse or Azure SQL database, or a dedicated SQL pool, you can enable encryption at rest. Even though the data is in the databases, ultimately it is sitting on disk at a Microsoft data centre. By default, the data in the synapse workspace is encrypted using platform managed keys. You can enable double encryption by enabling customer managed keys which allows you to set your own keys in the key vault. However, this double encryption can only be enabled during the creation of the azure synapse workspace (if being used in synapse).

A screenshot of a computer

Description automatically generated

So, in the example above, when creating a synapse service and in the security tab, it asks if you want to use double encryption using a customer managed key. If you select enable, it will ask you for a key in the key vault.

A screenshot of a computer

Description automatically generated

In a dedicated SQL pool, there is a tab named transparent data encryption that you can turn on data encryption.

Dynamic Data Masking

If you want to limit the exposure of data, you can create rules to mask the data. You can create rules to mask columns that hold credit card data. You can encrypt email address details, and you can decide which characters to expose in a given column field. For example, in the dedicated SQL pool, there is a tab on the left named Dynamic Data Masking, it will scan the tables we have, and it will give some recommended fields to mask. You can choose a schema and table and column that you want to mask. You can then set rules and then click add and save. This will then give us a masking rule. The mask will only come into effect with non-admin users. The admin user will still be able to see the information. Only a non-admin will be able to see the encrypted version. This is why many columns at network rail had values of zero in them, the data was masked.

Implementing column level security in Dedicated SQL pool.

Azure Synapse Analytics allows you to implement column-level security using features like Dynamic Data Masking, which can be used to obfuscate sensitive data in query results for non-privileged users.

Implementing row level security in Dedicated SQL pool.

supports Row-Level Security (RLS). Row-Level Security is a feature that allows you to control access to rows in a database table based on the characteristics of the user executing a query. Security Predicate: A security predicate is a filter predicate applied to a table that defines which rows users are allowed to access. The security predicate is defined using a user-defined inline table-valued function.

**Synapse**

Types of compute

The SQL pool is used to build the data warehousing. If you want to have tables In place with data, you can use the SQL pool option. The data is held on disk in the Microsoft data centre, and compute nodes are used to process the data. You are charged by a metric known as DWU. This is a more expensive option, but here your data is already structured, and ready to be queried and visualised. The dedicated SQL pool is the data warehouse. Business users can work with the data here

Serverless SQL Pool

There are no servers involved here and all of the compute used for analysis is in the background, and you are only charged for the amount of data you analyse. The serverless SQL pool can be used to quickly perform analysis of data in the adls. You can then also transfer the data to the dedicated SQL pool. In the serverless SQL pool is that you cannot post data onto disk or storage. The serverless SQL pool is for quick analysis of data and you can use TSQL for that. It is not used for data warehousing. The dedicated SQL pool is used for data warehousing.

Working in Synapse.

Lets now open Synapse Studio. You have to create a linked service to connect synapse with adls. Once this is done, you also have to give yourself permission in adls under IAM to add your role to access the data in adls. Even though I am an admin, I still have to assign myself the role. Lets say that we have a csv file in adls, we can use the built in serverless SQL pool to structure the data and perform queries on the data using TSQL. In the develop tab, open a new query and put in the following code:

A screenshot of a computer

Description automatically generated

The OPENROWSET function is used to connect to the container, and to state the file format that is being imported into the serverless SQL pool. This function converts the file data into a structured format and presents it in a SQL table. For the example above, I am using the built in serverless SQL pool.

**Working with External Tables**

When we create the serverless SQL Pool, the default database that comes with it is the master database. The master database also contains all of the system tables. However, we can choose to create additional databases which are considered user defined databases. The three dots next to the built in (in the image above) allows you to switch from one database to another. So lets create a new database and then switch into that database (which I have named appdb).

A screenshot of a computer

Description automatically generated

Lets now add the query to create an external data source. An external data source allows you to access and query data stored in external storage locations such as Azure Data Lake Storage (ADLS), Azure Blob Storage, or Azure Synapse dedicated SQL pool (formerly SQL Data Warehouse). In Azure Synapse Analytics Serverless SQL Pool, defining external tables serves the purpose of allowing you to query and analyze data stored in external storage locations, such as Azure Data Lake Storage (ADLS) Gen2, without physically copying or moving the data into the serverless SQL pool. This enables you to perform analytics on data in its original location, providing flexibility, cost-effectiveness, and the ability to leverage existing data stored externally.This is useful when your data is distributed across different storage solutions.

The query for accessing the data without copying it from adls can be found at this link:

[CREATE EXTERNAL DATA SOURCE (Transact-SQL) - SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/t-sql/statements/create-external-data-source-transact-sql?view=azure-sqldw-latest&preserve-view=true&tabs=dedicated)

The initial part of the query is shown below, and the credential parameter is specified as SasToken, which is a made up variable. In the Microsoft documentation, it shows the syntax for the different credentials you can use. Also, the key allows access to the adls container.

A screenshot of a computer

Description automatically generated

As the Sastoken has been mentioned, we have to define what the Sastoken is. First go and get the Shared access signature from adls as shown below.

A screenshot of a computer

Description automatically generated

The generate it:

A screenshot of a computer

Description automatically generated

Copy the Sas token and then put it in the query below:

A screenshot of a computer

Description automatically generated

In the code above, we are simply saying that we want to create an external data source that we want to connect to, and we name it log data. We connect to the raw container and say that the credential is the SasToken. We then define the SasToken above and say that the identity is a shared access signature. The SAS token is then generated in adls and then pasted in the parameter named secret. The Appdb database will use the Sas token to access the container. The database is saying that lets go in and encrypt the value of the shared access signature so that nobody else can access it. In order to do this, it has to create its own encryption key and then encrypt this secret accordingly.

A screenshot of a computer

Description automatically generated

Dedicated SQL pools

Click on add dedicated sql pool.

A screenshot of a computer

Description automatically generated

Give the dedicated SQL pool a name and then select a performance level. The lower the performance, the cheaper it is. Click create

A screenshot of a computer

Description automatically generated

. In the SQL pools tab, you will see the SQL pools that exist in the workspace.

A screenshot of a computer

Description automatically generated

In Synapse studio, you can now see that the dedicated sql pool is In the workspace. You can also see that the dedicated sql pool allows you to define external tables and normal tables, whereas the serverless sql pool only allows external tables.

A screenshot of a computer

Description automatically generated

**Creating external tables in the dedicated sql pool ( reading parquet files)**

In adls, upload the log.parquet file to start with.

**A screenshot of a computer

Description automatically generated**

To define the external table, we will use most of the code that we saw earlier on when we created the external tables In the serverless sql pool. The code below allows you to read data from external parquet files.

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'W3lcome123';

CREATE DATABASE SCOPED CREDENTIAL SasToken

WITH IDENTITY = 'SHARED ACCESS SIGNATURE',

     SECRET = 'sv=2022-11-02&ss=b&srt=sco&sp=rwdlacyx&se=2023-11-24T06:35:47Z&st=2023-11-23T22:35:47Z&spr=https&sig=P65zmd%2FjGHts7McQ2ROebVjyuGVA3W3dZ6qx8nuRXf8%3D';

CREATE EXTERNAL DATA SOURCE log\_data\_parquet

WITH (

    LOCATION = 'https://adls3105.dfs.core.windows.net/raw/',

    CREDENTIAL = SasToken

);

CREATE EXTERNAL FILE FORMAT parquetfile

WITH (

    FORMAT\_TYPE = PARQUET,

    DATA\_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'

);

CREATE EXTERNAL TABLE [logdata\_parquet]

(

    [Correlationid] [varchar](200) NULL,

    [Operationname] [varchar](200) NULL,

    [Status] [varchar](100) NULL,

    [Eventcategory] [varchar](100) NULL,

    [Level] [varchar](100) NULL,

    [Time] [varchar](500) NULL,

    [Subscription] [varchar](200) NULL,

    [Eventinitiatedby] [varchar](1000) NULL,

    [Resourcetype] [varchar](1000) NULL,

    [Resourcegroup] [varchar](1000) NULL,

    [Resource] [varchar](2000) NULL)

WITH (

 LOCATION = '/log.parquet',

    DATA\_SOURCE = log\_data\_parquet,

    FILE\_FORMAT = parquetfile

)

SELECT \* FROM logdata\_parquet

An explanation of each step is given below:

This SQL script is used to set up and query an external table in a SQL Server database that references data stored in an Azure Data Lake Storage (ADLS) account. Let's break down each part of the code:

1. **CREATE MASTER KEY:**

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'W3lcome123';

This statement creates a master key for the database, and it is used for encrypting other keys in the database.

1. **CREATE DATABASE SCOPED CREDENTIAL:**

CREATE DATABASE SCOPED CREDENTIAL SasToken WITH IDENTITY = 'SHARED ACCESS SIGNATURE', SECRET = 'sv=2022-11-02&ss=b&srt=sco&sp=rwdlacyx&se=2023-11-24T06:35:47Z&st=2023-11-23T22:35:47Z&spr=https&sig=P65zmd%2FjGHts7McQ2ROebVjyuGVA3W3dZ6qx8nuRXf8%3D';

This statement creates a database-scoped credential named **SasToken** with a Shared Access Signature (SAS) as the secret. This credential is later used to access the external data source.

1. **CREATE EXTERNAL DATA SOURCE:**

CREATE EXTERNAL DATA SOURCE log\_data\_parquet WITH ( LOCATION = 'https://adls3105.dfs.core.windows.net/raw/', CREDENTIAL = SasToken );

This statement creates an external data source named **log\_data\_parquet**. It specifies the location of the data in the Azure Data Lake Storage and the credential (**SasToken**) to use for authentication.

1. **CREATE EXTERNAL FILE FORMAT:**

CREATE EXTERNAL FILE FORMAT parquetfile WITH ( FORMAT\_TYPE = PARQUET, DATA\_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec' );

This statement defines the file format for the external files. In this case, it specifies that the files are in Parquet format and are compressed using the SnappyCodec.

1. **CREATE EXTERNAL TABLE:**

CREATE EXTERNAL TABLE [logdata\_parquet] ( -- Columns definition ) WITH ( LOCATION = '/log.parquet', DATA\_SOURCE = log\_data\_parquet, FILE\_FORMAT = parquetfile );

This statement creates an external table named **logdata\_parquet** that maps to the data stored in the Azure Data Lake Storage. It defines the structure of the table based on the columns specified and sets the location, data source, and file format for the external data.

1. **SELECT statement:**

SELECT \* FROM logdata\_parquet;

This is a sample query to retrieve all records from the external table **logdata\_parquet**. It demonstrates how you can query the external data seamlessly as if it were a regular table in your SQL Server database. The actual columns and data types would be based on your specific use case and the structure of your external data.

**Pausing the dedicated SQL pool**

In the manage tab and in the dedicated sql pool, you can pause it as it costs money. You do this by pressing pause. This pauses the compute. If you delete the pool, you will lose the data in the pool. Whenever you want to resume working on the pool, select the resume button.

A screenshot of a computer

Description automatically generated

Loading data into the Dedicated SQL pool ( data warehouse). For this, you need to create tables. Initially, you would perform a one-time load to load all previous data from source systems. From that point on, you would perform a delta load to incrementally load new data into the tables.

You can use the copy statement to copy data into the synapase tables from adls external tables. More info on the query can be found here:

[COPY INTO (Transact-SQL) - Azure Synapse Analytics and Microsoft Fabric | Microsoft Learn](https://learn.microsoft.com/en-us/sql/t-sql/statements/copy-into-transact-sql?view=azure-sqldw-latest)

Create a new script file and make sure you are connected to the correct database. In our case, the DB is named dedicated\_sql\_pool. We would now create a normal table as you would in a SQL Server database. We create a table, and the table schema should be setup and we can see the columns in in the tables folder. We have not yet loaded any tables into the table.

**A screenshot of a computer

Description automatically generated**

I then copy data into the table:

**A screenshot of a computer

Description automatically generated**

We copy into the table from adls with the stated parameters. The parameters are file type and credentials (with the storage account access keys). We get the access keys from the following place. Just use key 1. Make sure to use the right adls container endpoint, container name and access key.

A screenshot of a computer

Description automatically generated

You can see that the data was copied into the table. This means that tables can be stored in the gold format and brought into the dedicated SQL pool. We have now copied the data from the log.csv file into our data warehouse. It is not simply referring to it, but it is a one time copy. Although we have copied the CSV data, we can copy the parquet data in the same way, as shown below.

A screenshot of a computer

Description automatically generated

Using PolyBase to load data

Can be used to extract data from adls, and then load it into the SQL warehouse. It is very useful when loading data into the warehouse efficiently from adls. We will cover more of this at a later date.

Loading data into Synapse dedicated SQL pool with pipelines in synapse.

A very simple way of copying data is to use the copy data tool in the integrate tab.

First lets create the destination table that we want to insert data into. We can then do the following steps to get data from adls into the table.

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

We select run once now.

A screenshot of a computer

Description automatically generated

You select the linked service and the source file that you want to copy.

A screenshot of a computer

Description automatically generated

Make the first row as the header row.

A screenshot of a computer

Description automatically generated

Select the dedicated SQL pool for the destination and use the table that was created earlier to put the data into.

A screenshot of a computer

Description automatically generated

All of the mappings can be performed to map the source columns to the destination columns.

A screenshot of a computer

Description automatically generated

In the window above, you have an option to copy data in a number of different way from source to sink. The four main options are described below:

COPY Command:

The COPY command is a part of the Azure Synapse Analytics (SQL Data Warehouse) SQL language. It's used to efficiently copy data between tables within the same or different databases. It's particularly suitable for copying large volumes of data between two tables.

PolyBase:

PolyBase is a feature in Azure Synapse Analytics that enables you to run Transact-SQL queries on external data in Azure Blob Storage, Azure Data Lake Storage, or Hadoop Distributed File System (HDFS). It provides a virtualization layer over the external data, allowing you to query and join it with your relational data.

PolyBase is a powerful tool for performing federated queries, integrating external data seamlessly with your existing data in Azure Synapse Analytics.

Bulk Insert:

Bulk Insert is a method for efficiently loading large amounts of data from external files into a table in Azure Synapse Analytics. It's useful when you have data stored in flat files (e.g., CSV or TSV) that you want to import into your Synapse Analytics environment.

This operation is optimized for high-performance data loading and is often faster than traditional row-by-row inserts.

Upsert:

Upsert stands for "update or insert," and it's an operation that either updates an existing record or inserts a new record if it doesn't exist. This operation is often used when you want to synchronize data between source and target tables.

The "Upsert" option in the Copy Data tool likely allows you to perform an upsert operation during the data movement process, ensuring that the destination table is updated with new data or modified to reflect changes in the source.

When choosing the appropriate data movement option, consider factors such as the volume of data, performance requirements, and the nature of your data integration scenario. Each option has its use cases and benefits, and the choice may depend on your specific requirements and the characteristics of the data you are working with.

In the example above, we will choose bulk insert.

In the next window, a dataset is going to be created and a pipeline, and the data will get copied over.

A screenshot of a computer

Description automatically generated

We can see that our first load of the data has taken place successfully.

A screenshot of a computer

Description automatically generated

**Design a Data Warehouse (DIM and FACT item tables).**

In your normal SQL database, you want to post your day-to-day transactional data, so this won’t be in a DIM/FACT item table format. However, in a data warehouse, the setup is in DIM/FACT item table as the data is historical. FACT tables state facts. For example, if there is a sales table, it will state the facts of how many sales have been made. This is a fact tables, as it tells you these facts. The DIM tables help to provide some context on the facts that have been presented. So, a customer table would be a dimension table, as it is telling you who has bought the products over a period of time. Even a product table would be a dimension table, as it is telling you which products were sold and when. It is giving you some contextual information about a sale. When building these tables, you may decide that you don’t want to copy all of the tables from the source system to the SQL data warehouse. Sales goes for products and customers.

A screenshot of a computer

Description automatically generated

So one major point is that you may want to bring in data into a product dimension table from multiple source systems. When you copy these rows into the product dimension table, the keys may be duplicated as the source systems will have primary keys for each row which are the same.

The way to sort this out is to create a surrogate key in the product dimension table which can be seen on the left in the image below which is named the itemKey. The column next to it which is named product ID will be the business key, as these are the product IDs that are being copied into the table. The surrogate key will now be the primary key of the table. You also don’t want NULL values in the dimension table or FACT tables.

A screenshot of a computer

Description automatically generated

We will use Adventure works database to build FACT and DIM item tables. Let’s create a new database using the sample Adventure works data so that we do not interfere with any other work that has been done previously. I created a database named AventureWorks 2 and I will connect it to SSMS so that I can create DIM and FACT item tables. There is now one SQL server with two databases.

A screenshot of a computer

Description automatically generated

I can also connect to the synapse dedicated SQL pool by finding the dedicated sql pool info from the following location:

A screenshot of a computer

Description automatically generated

You can now see that the dedicated sql pool is showing in SSMS.

A screenshot of a computer

Description automatically generated

In the adventure works 2 database, lets create a new script to create a view, selecting columns from the sales order detail table and the sales order header table, creating a join between the two. We want to join the tables as we want to use columns in both tables (after a join). We then use the columns in the view to create a new table named FactSales, and the result can be seen below. This is an approach used in a dev environment. In a production environment, you may not want to create a new table in the database, so you would need to use data factory to do the same transformation and pull the columns that you need. These columns can now be copied to a FACT table, as they show facts about the data.

A screenshot of a computer

Description automatically generated

Lets now build two dimension tables, one based on the product information, and one based on the customer information.