**DP-203 Notes**

**2. Azure Data Lake Storage Gen2**

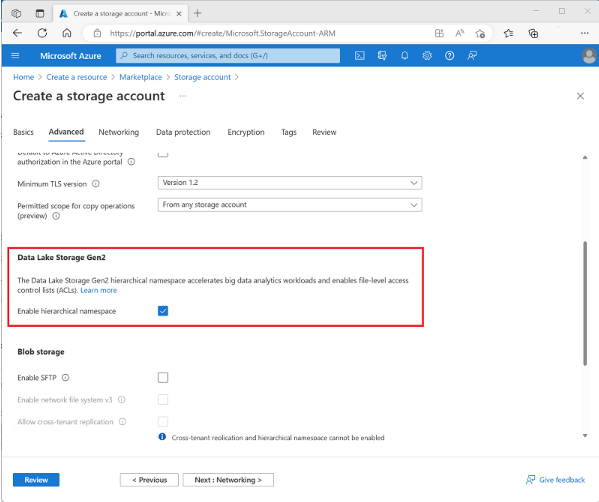
2.1 Understand Azure Data Lake Storage Gen2

* A data lake is a repository of data that is stored in its natural format, usually blobs or files.
* Azure Data Lake Storage is a comprehensive, massively scalable, secure, and cost-effective data lake solution for high-performance analytics built into Azure.
* Azure Data Lake storage builds on Azure Blob storage capabilities to optimize specifically for analytics workloads.
* Benefits
  + Batch and Real-time solutions
    - Designed to deal with hundreds of gigabytes of throughput.
    - It can be used for both batch and real-time solutions.
  + Hadoop compatible access
    - You can treat data as if they are stored in a Hadoop Distributed File System (HDFS):
      * Data are stored in one place and can be accessed through compute technologies (incl. Azure Databricks, Azure HDInsight, and Azure Synapse Analytics), without moving the data between environments.
  + Security
    - Data lake storage supports access control lists (ACLs) and Portable Operating System Interface (POSIX) permissions that do not inherit the permissions of the parent directory.
    - The user can set permissions at a directory level or file level for the data stored within the data lake, providing a much more secure storage system.
    - The security is configurable through technologies such as Hive/Spark or utilities such as Azure Storage Explorer.
    - All data that is stored is encrypted at rest by using Microsoft or customer-managed keys.
  + Performance
    - Data Lake Storage organizes the stored data into hierarchy of directories and sub-directories, much like a file system for easy navigation.
    - As a result, data processing requires less computational resources, reducing both time and costs.
  + Data redundancy
    - Data Lake storage takes advantage of the Azure Blob replication models:
      * LRS: redundancy in a single data centre with locally redundant storage.
      * GRS: redundancy in a secondary region

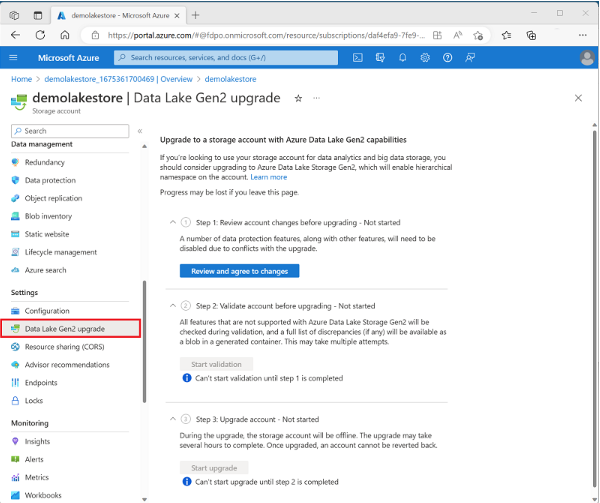
2.2 Enable Azure Data Lake Storage Gen2 in Azure Storage

* Azure Data Lake Storage Gen2 is not a stand-alone service, but rather a configurable capability of a Storage V2 (General Purpose V2) Azure Storage.
* To enable it, you can select the option Enable hierarchical namespace in the advice page, when you are creating the storage account in the Azure portal (Fig A).
* Alternatively, if you already have an Azure Storage account and want to enable the Azure Data Lake Storage Gen2 capability, you can use the **Data Lake Gen2 upgrade** wizard in the Azure portal page for your storage account resource (Fig B).

**Fig A**



**Fig B**



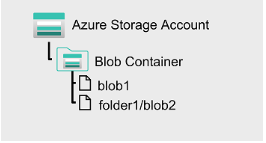
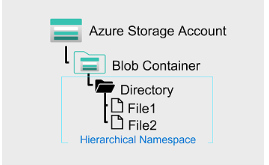
2.3 Compare Azure Data Lake Store to Azure Blob Storage

*Azure Blob Storage*

* In Azure Blob Storage, you can store large amounts of unstructured (“object”) data in a flat namespace within a blob container.
* Blob names can include “/” characters to organize blobs into virtual “folders”, but in terms of blob manageability, the blobs are stored as a single-level hierarchy in a flat namespace.
* You can access this data by using HTTP or HTTPs.

*Azure Data Lake Storage Gen2*

* Azure Data Lake Storage Gen2 builds on blob storage and optimize I/O of high-volume data by using a hierarchical namespace that organize blobs into *directories*, and stores metadata about each directory and files within it.
* This structure allows operations, such as directory renames and deletes, to be performed in a single atomic operation (which is something that is not possible with flat namespaces).
* Hierarchical namespaces keep the data organized, which yields better storage and retrieval performance for analytical use case and lowers the cost of analysis.

2.4 Understand the stages for processing big data

* Data Lakes have a fundamental role in a wide range of big data architectures, for instance:
  + An enterprise data warehouse
  + Advanced analytics against big data
  + A real-time analytical solution
* There are 4 stages for processing big data solutions that are common to all architectures:
  + Ingest
    - Phase that identifies the tech and processes that are used to acquire the source data.
    - This data can come from files, logs, and other types of unstructured data that must be put into the data lake.
  + Store
    - Phase identifies where the ingested data should be placed. Azure Data Lake Storage Gen2 provides a secure and scalable storage solution that is compatible with commonly used technologies.
  + Perp and Train
    - Phase that identifies the technologies that are used to perform data prep/model training and scoring for machine learning solutions.
  + Model and Serve
    - Phase that involves the technologies that will present the data to the users, (incl. visualization tools).

2.5 Use Azure Data Lake Storage Gen2 in data analytics workloads

* Azure Data Lake Storage Gen2 enables technology for multiple data analytics use cases:
  + Big data processing and analytics
    - Analytical workloads that involve massive *volumes* of data in a *variety* of formats that needs to be processed at a fast *velocity*.
    - Azure Data Lake Storage Gen2 provides a scalable and secure distributed data store on which big data services (Azure Synapse Analytics, Azure Databricks, Azure HDInsight) can apply data processing frameworks (Apache Spark, Hive, Hadoop).
    - The distributed nature of the storage and the processing compute enables tasks to be performed in parallel, resulting in high-performance and scalability even when processing huge amounts of data.
  + Data warehousing
    - Data warehousing has evolved in recent years to integrate large volumes of data stored as files in a data lake with relational tables in a data warehouse.
    - In a typical example of a data warehouse solution:
      * Data is extracted from operational data stores (Azure SQL DBs, Azure Cosmos DB).
      * Data is transformed into structures more suitable for analytical workloads.
      * Often the data is staged in a data lake to facilitate distributed processing before being loaded into a relational data warehouse.
      * In some cases, the data warehouse uses *external* tables to define a relational metadata layer over files in the data lake and create a hybrid “data lakehouse” or “lake database” architecture.
      * The data warehouse can then support analytical queries for reporting and visualization.
  + Real-time analytics
    - Increasingly, businesses and other organizations need to capture and analyse perpetual streams of data and analyse it in real-time.
    - Unlike traditional batch processing, streaming data requires a solution that can be captured and process as data events occur.
    - Streaming events are often captured in a queue for processing.
    - Azure Stream Analytics enables to create jobs that query and aggregate data event as they arrive and write the results in an output sink.
      * For instance Azure Data Lake Storage Gen2, from where the captured real-time data can be analysed and visualized.
  + Data science and machine learning

**3. Introduction to Azure Synapse Analytics**

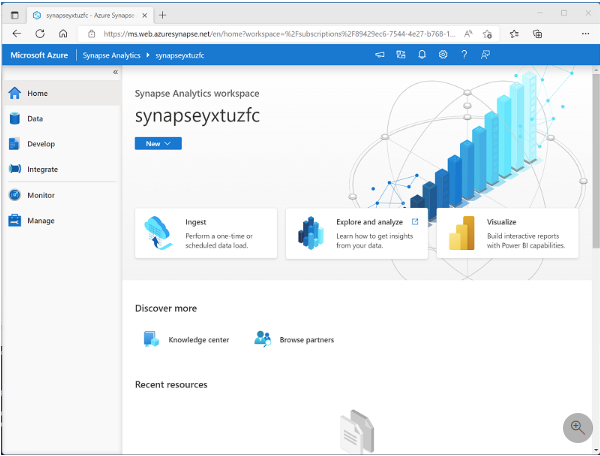
3.1 How Azure Synapse Analytics works

* Azure Synapse Analytics provides a cloud platform for analytical workflows (e.g., descriptive, diagnostic, predictive, prescriptive).
* To support analytics needs, Azure Synapse Analytics combines

1. A centralized service for data storage and processing with
2. an extensible architecture through which *linked services* enable you to integrate commonly used data stores, processing platforms, and visualization tools.

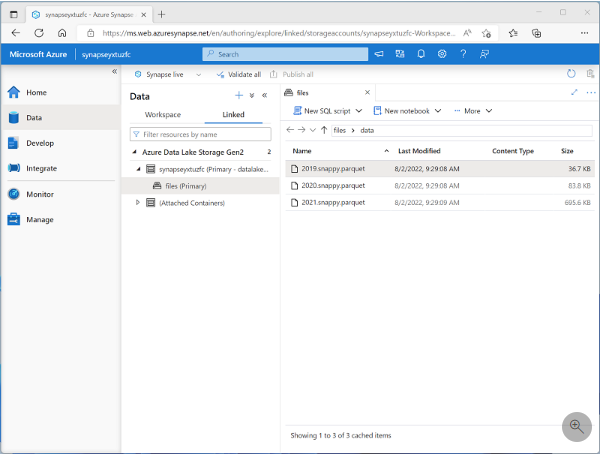
3.1.1 Creating and using an Azure Synapse Analytics workspace

* A Synapse Analytics *workspace* defines an instance of the Synapse Analytics service in which you can manage the services and data resources needed.
* You can create a Synapse Analytics workspace in an Azure subscription interactively by using the Azure Portal or you can automate the deployment by using (i) Azure PowerShell (Azure command-line interface), (ii) Azure Resource Manager, (iii) Bicep template.
* After creating a Synapse workspace, you can manage the service in it and perform data analytics tasks with them by using Synapse Studio, a web-based portal for Azure Synapse Analytics.



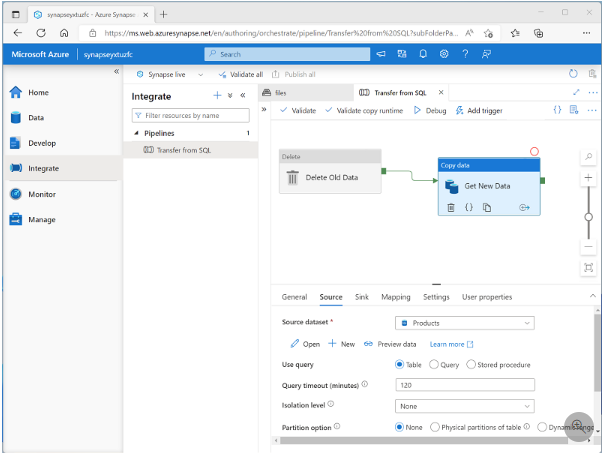
3.1.2 Working with files in a data lake

* One of the core resources in a Synapse Analytics workplace is a *data lake*, in which data files can be stored and processed at scale.
* A workspace typically has a default data lake, which is implemented as linked service to an Azure Data Lake Storage Gen2 container.
* You can add linked services for multiple data lakes that are based on different storage platforms as required.



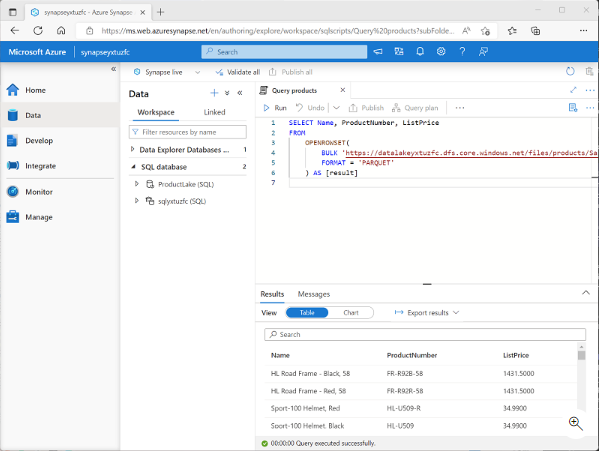
3.1.3 Ingesting and transforming data with pipelines

* In most enterprise data analytics solutions, data is extracted from multiple operational sources and transferred to a central data lake or data warehouse.
* Azure Synapse Analytics includes built-in support for creating, running, and managing pipelines that orchestrate the activities necessary to retrieve data from a range of sources, transform the data as required, and load the resulting transformed data into an analytical store.



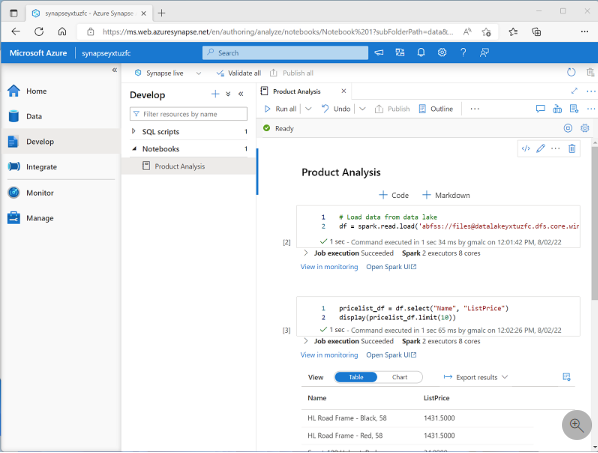
3.1.4 Querying and manipulating data with SQL

* Azure Synapse Analytics supports SQL-based data querying and manipulation through two kinds of SQL pool that are based on the SQL Server relational database engine:
* A built-in serverless pool that is optimized for using relational SQL semantics to query file-based data in a data lake.
* Custom dedicated SQL pools that host relational data warehouses.
* The Azure Synapse SQL system uses a distributed querying processing model to parallelize SQL operations, resulting in a highly scalable solution for relational data processing.
* You can use the built-in serverless pool for cost-effective analysis and processing of file data in the data lake and use dedicated SQL pools to create relational data warehouses for enterprise data modelling and reporting.



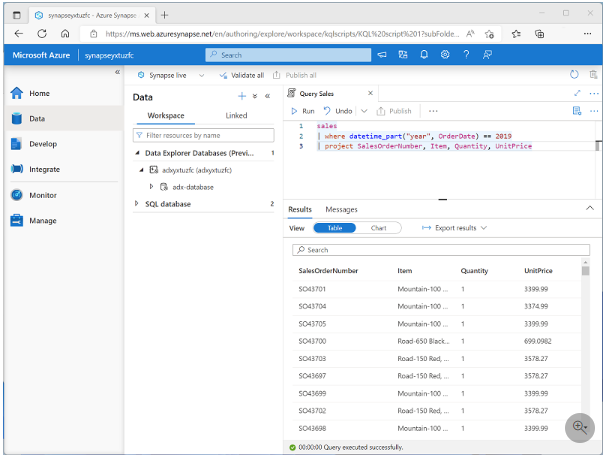
3.1.5 Processing and analysing data with Apache Spark

* Apache Spark is an opensource platform for big data analytics. Spark performs distributed processing of files in a data lake by running jobs that can be implemented using any of a range of supported programming languages (Python, Scala, Java, SQL, and #C).
* In Azure Synapse Analytics, you can create one or more Spark Pools and use interactive *notebooks* to combine code and notes as you build solutions for data analytics, ML, and data visualization.



3.1.6 Exploring data with Data Explorer

* Azure Synapse Data Explorer is a data processing engine in Azure Synapse Analytics that is based on the Azure Data Explorer service.
* Data Explorer uses an intuitive query syntax named Kusto Query Language (KQL) to enable high performance, low-latency analysis of batch and streaming data.



3.1.7 Integrating with other Azure data services

* Azure Synapse Analytics can be integrated with other Azure data services for end-to end analytics solutions.
  + Azure Synapse Link
    - It enables near-realtime sync between operational data in Azure Cosmos DB, Azure SQL Database, SQL Server, and Microsoft Power Platform Dataverse and analytical data storage that can be queried in Azure Synapse Analytics.
  + Microsoft Power BI
    - Integration enables data analysts to integrate a Power BI workspace into a Synapse workspace, and perform interactive data visualization in Azure Synapse Studio
  + Microsoft Purview
    - Integration enables organizations to:
      * catalog data assets in Azure Synapse Analytics;
      * make it easier for data engineers to find data assets and track data lineage when implementing data pipelines that ingest data into Azure Synapse Analytics
  + Azure ML
    - Integration enables data analysts and data scientists to integrate predictive model training and consumption into analytical solutions.

3.2 When to use Azure Synapse Analytics

* Large-scale data warehousing
* Advanced analytics (through the integration with Azure Machine Learning)
* Data exploration and discovery
* Real-time analytics
* Data integration
* Integrated analytics

**4. Use Azure Synapse serverless SQL pool to query files in a data lake**

* Azure Synapse Analytics includes serverless SQL pools, which are tailored for querying data in a Data Lake.
  + SQL code to query the data in files of various common formats without the needing to load the file data into / maintain a database storage.

4.1 Azure Synapse serverless SQL pool capabilities and use cases

* *Azure Synapse SQL* is a distributed query system in Azure Synapse Analytics that offers two kinds of runtime environments:
  + Serverless SQL pool
    - On-demand SQL query processing, primarily used to work with data in a data lake.
  + Dedicated SQL pool
    - Enterprise-scale relational database instance used to host data warehouses in which data is stored in relational tables.
* Serverless SQL pools
  + They provide a pay-per-query endpoint to query the data in your data lake*.*
  + Benefits
    - T-SQL syntax + no need to copy or load data into a specialized stored.
    - Integrated connectivity from a wide range of BI and ad-hoc querying tools.
    - Distributed query processing built for large-scale data, and computational functions (fast query performances)
    - Built-in query execution fault-tolerance
    - No infra to set up / to maintain + Elimination of management burden re ingesting the data into the system
      * A built-in endpoint for this service is provided within every Azure Synapse workspace, so you can start querying data as soon as the workspace is created.
    - No charge for resources reserved, you are only charged for the data processed by queries you run.
      * Using the serverless pool helps when you need to know exact costs for each query executed to monitor and attribute costs.
  + Common uses cases When to use serverless SQL pools,
    - Data exploration
      * Data exploration involves browsing the data lake to get initial insights about the data and it is achievable with Azure Synapse Studio.
      * You can browse through the files in your linked data storage and use the built-in serverless SQL pool to automatically generate a SQL scripts.
    - Data transformation
      * Serverless SQL pool enables the user to perform SQL-based data transformations, either interactively or as part of an automated data pipeline.
    - Logical data warehouse
      * After an initial exploration of the data in the data lake, the user can define external objects such as tables and views in a serverless SQL databases.
        + The data remains stored in the data lake files, but are abstracted by a relational schema that can be used by client applications and analytical tools to query the data as they would in a relational DB hosted in SQL Server.

4.2 Query files using a serverless SQL pool

* You can use a serverless SQL pool to query data files in various common file formats, incl.:
  + Delimited text, such as comma-separated values (CSV) files
  + JSON files
  + Parquet files
* The basic syntax for querying is the same for all of these types of files and it is built on OPENROWSET SQL function:

|  |
| --- |
| SELECT TOP 100 \*  FROM OPENROWSET(  BULK 'https://mydatalake.blob.core.windows.net/data/files/\*.csv',  FORMAT = 'csv') AS rows |
| 1. BULK    1. The parameter includes the full URL to the location in the data lake containing the data files.    2. This can be an individual file, or a folder with a wildcard expression to filter the file types that should be included.    3. Also, multiple file paths can be specified, separating each path with a comma.      1. FORMAT:    1. The parameter specifies the type of data being queried. The example above reads delimited text from all .csv files in the files folder. 2. The output from OPENROWSET is a rowset to which an alias must be assigned. In the previous example, the alias **rows** is used to name the resulting rowset. |

* Querying delimited text files
  + The specific formatting used in delimited files can vary, for example
    - * With/without a header row
      * Comma and tab-delimited values
      * Windows and Unix line endings
      * Non-quoted and quoted values and escaping charters.

|  |
| --- |
| SELECT TOP 100 \*  FROM OPENROWSET(  BULK 'https://mydatalake.blob.core.windows.net/data/files/\*.csv',  FORMAT = 'csv',  PARSER\_VERSION = '2.0',  FIRSTROW = 2) AS rows |
| * Regardless of the type of delimited file, the FORMAT parameter can always be set as ‘csv’. * PARSER\_VRSION:   + Used to determine how the query interprets the text encoding used in the files. * FIRSTROW   + Skip the first two rows   Other commonly used paras:   * HEADER\_ROW   + Query engine to use the first row of data in each file as column names * WITH Clause to specify column names and data types, for instance:   SELECT TOP 100 \*  FROM OPENROWSET(  BULK 'https://mydatalake.blob.core.windows.net/data/files/\*.csv',  FORMAT = 'csv',  PARSER\_VERSION = '2.0')  WITH (  product\_id INT,  product\_name VARCHAR(20) COLLATE Latin1\_General\_100\_BIN2\_UTF8,  list\_price DECIMAL(5,2)  ) AS rows |

* Querying JSON files
  + JSON is a popular format for web applications that exchange data through REST interfaces or use NoSQL data stores such as Azure Cosmos DB

|  |
| --- |
| SELECT JSON\_VALUE(doc, '$.product\_name') AS product,  JSON\_VALUE(doc, '$.list\_price') AS price  FROM  OPENROWSET(  BULK 'https://mydatalake.blob.core.windows.net/data/files/\*.json',  FORMAT = 'csv',  FIELDTERMINATOR ='0x0b',  FIELDQUOTE = '0x0b',  ROWTERMINATOR = '0x0b'  ) WITH (doc NVARCHAR(MAX)) as rows |
| * OPENROWSET has no specific format for JSON files, so you must use csv format with FIELDTERMINATOR, FIELDQUOTE, and ROWTERMINATOR set to 0x0b. * The schema needs to includes a single NVARCHAR(MAX) column. * In addition, to avoid a rowset containing a single column of JSON documents and extract individual values, use of JSON\_VALUE function in the SELECT statement. |

* Querying Parquet files
  + Parquet is a commonly used format for big data processing on distributed file storage.
  + It is an efficient data format that it is optimized for compression and analytical querying.
  + In most cases, the schema of data is embedded within the Parquet file, so you only need to specify the BULK and FORMAT parameters (FORMAT = ‘parquet’).
* Query partitioned data
  + It is common in a data lake to partition data by splitting across multiple files in subfolders that reflect partitioning criteria.
  + This enables distributed processing systems to work in parallel systems to work in parallel on multiple partitions of the data, or to easily eliminate data reads from specific folders based on filtering criteria.

|  |  |
| --- | --- |
|  | SELECT \*  FROM OPENROWSET(  BULK 'https://mydatalake.blob.core.windows.net/data/orders/year=\*/month=\*/\*.\*',  FORMAT = 'parquet') AS orders  WHERE orders.filepath(1) = '2020'  AND orders.filepath(2) IN ('1','2'); |

4.3 Create external database objects

* You can use OPENROWSET function in SQL queries that run in the default **master** database of the built-in serverless pool to explore data in the data lake.
* However, sometimes you may want to create a custom DB that contains some objects that make it easier to work with external data in the data lake you need to query frequently.
* Creating a database
  + Creating a database in a serverless SQL pool is like what you would do in a SQL Server instance.
  + You can sue the graphical interface in Synapse Studio, or a CREATE DATABASE statement.
    - Set collation that it supports conversion of text data in files to appropriate Transact-SQL data types.

|  |
| --- |
| CREATE DATABASE SalesDB  COLLATE Latin1\_General\_100\_BIN2\_UTF8 |

* Creating an external data source
  + It is efficient to define an external data source that reference to a location, if you plan to quey data in the same location frequently – simplifying the OPENROWSET function call

|  |
| --- |
| CREATE EXTERNAL DATA SOURCE files  WITH (  LOCATION = 'https://mydatalake.blob.core.windows.net/data/files/'  )  SELECT \*  FROM  OPENROWSET(  BULK 'orders/\*.csv',  DATA\_SOURCE = 'files',  FORMAT = 'csv',          PARSER\_VERSION = '2.0'  ) AS orders |

* + Other benefit – Access to the underlying storage through SQL, without permitting user to access the data directly in the storage account.

|  |
| --- |
| CREATE DATABASE SCOPED CREDENTIAL sqlcred  WITH  IDENTITY='SHARED ACCESS SIGNATURE',  SECRET = 'sv=xxx...';  CREATE EXTERNAL DATA SOURCE secureFiles  WITH (  LOCATION = 'https://mydatalake.blob.core.windows.net/data/secureFiles/'  CREDENTIAL = sqlcred  ); |
| * The code creates a credential that uses a shared access signature (SAS) to authenticate against the underlying Azure storage account hosting the data lake. * In addition to SAS authentication, you can define credentials that use managed identity (the Micorsoft Entra identity used by your Azure Synapse workspace). |

* Creating an external file format
  + While external data source simplifies the code needed to access files with the OPENROWSET function, you still need to provide formal details for the file being access.
  + This may include multiple settings for delimited text files.
  + Settings can be encapsulated as follows

|  |
| --- |
| CREATE EXTERNAL FILE FORMAT CsvFormat  WITH (  FORMAT\_TYPE = DELIMITEDTEXT,  FORMAT\_OPTIONS(  FIELD\_TERMINATOR = ',',  STRING\_DELIMITER = '"'  )  ); |

* Creating an external table
  + When you need to perform a lot of analysis or reporting from files in the data lake, using the OPENRAWSET function can result in complex code that includes data sources and file paths.
  + To simplifyused

|  |
| --- |
| CREATE EXTERNAL TABLE dbo.products  (  product\_id INT,  product\_name VARCHAR(20),  list\_price DECIMAL(5,2)  )  WITH  (  DATA\_SOURCE = files,  LOCATION = 'products/\*.csv',  FILE\_FORMAT = CsvFormat  );  GO  -- query the table  SELECT \* FROM dbo.products; |

* + By creating a database that contains the external objects discussed in this unit, you can provide a relational database layer over files in a data lake, making it easier for many data analysts and reporting tools to access the data by using standard SQL query semantics.

**5. Use Azure Synapse serverless SQL pools to transform data in a data lake**

5.1 The CETAS statement

* Azure Synapse serverless SQL pools can be used to run SQL statements that transform the data and persist the results as a file in a data lake for further processing and querying.
* You can use a CREATE EXTERNAL TABLE AS SELECT (CETAS) statement in a dedicated SQL pool or serverless SQL pool to persist the results of a query in an external table stored in a file in the data lake.
  + The results of the SELECT statement are persisted in an external table, which is a metadata object in a database that provides a relational abstraction over the data stored in files.
* Creating external database object to support CETAS
  + To use CETAS expressions, you must create the following types of object in a database for either a serverless or dedicated SQL pool.
    - When using a serverless SQL pool, create these objects in a custom database (created using the CREATE DATABASE statement), not the **built-in** database.
  + External data source
    - An external data source encapsulates a connection to a file system location in a data lake.
    - You can then use this connection to specify a relative path in which the data files for the external table created by the CETAS statement are saved

|  |
| --- |
| -- Create an external data source for the Azure storage account  CREATE EXTERNAL DATA SOURCE files  WITH (  LOCATION = 'https://mydatalake.blob.core.windows.net/data/files/',  TYPE = HADOOP, -- For dedicated SQL pool  -- TYPE = BLOB\_STORAGE, -- For serverless SQL pool  CREDENTIAL = storageCred  ); |
| CREATE DATABASE SCOPED CREDENTIAL storagekeycred  WITH  IDENTITY='SHARED ACCESS SIGNATURE',  SECRET = 'sv=xxx...';  CREATE EXTERNAL DATA SOURCE secureFiles  WITH (  LOCATION = 'https://mydatalake.blob.core.windows.net/data/secureFiles/'  CREDENTIAL = storagekeycred  ); |

* + External file format
    - The CETAS statement creates a table with its data stored in files. You must specify the format of the files you want to create as an external file format.

|  |
| --- |
| CREATE EXTERNAL FILE FORMAT ParquetFormat  WITH (  FORMAT\_TYPE = PARQUET,  DATA\_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'  ); |
| * In this example, the files will be saved in Parquet format. * You can also create external file formats for other types of file. |

* + Using the CETAS statement
    - After creating an external data source and external file format, you can use the CETAS statement to transform data and stored the results in an external table.
    - Example:
      * The source data you want to transform consists of sales orders in comma-delimited text files that are stored in a folder in a data lake.
      * You want to filter the data to include only orders that are marked as "special order".
      * You want to save the transformed data as Parquet files in a different folder in the same data lake.

|  |
| --- |
| CREATE EXTERNAL TABLE SpecialOrders  WITH (  -- details for storing results  LOCATION = 'special\_orders/',  DATA\_SOURCE = files,  FILE\_FORMAT = ParquetFormat  )  AS  SELECT OrderID, CustomerName, OrderTotal  FROM  OPENROWSET(  -- details for reading source files  BULK 'sales\_orders/\*.csv',  DATA\_SOURCE = 'files',  FORMAT = 'CSV',  PARSER\_VERSION = '2.0',  HEADER\_ROW = TRUE  ) AS source\_data  WHERE OrderType = 'Special Order'; |
| * The LOCATION and BULK parameters in the previous example are relative paths for the results and source files respectively. * The paths are relative to the file system location referenced by the **files** external data source. |

* + - You must use an external data source to specify the location where the transformed data for an external table is to be saved.
    - When file-based source data is stored in the same folder hierarchy, you can use the same external data source.
    - Otherwise, you can use a second data source to define a connection to the source data or use the fully qualified path, as shown in this example:

|  |
| --- |
| CREATE EXTERNAL TABLE SpecialOrders  WITH (  -- details for storing results  LOCATION = 'special\_orders/',  DATA\_SOURCE = files,  FILE\_FORMAT = ParquetFormat  )  AS  SELECT OrderID, CustomerName, OrderTotal  FROM  OPENROWSET(  -- details for reading source files  BULK 'https://mystorage.blob.core.windows.net/data/sales\_orders/\*.csv',  FORMAT = 'CSV',  PARSER\_VERSION = '2.0',  HEADER\_ROW = TRUE  ) AS source\_data  WHERE OrderType = 'Special Order'; |

* + - Dropping external tables
      * It is important to understand that external tables are a metadata abstraction over the files that contain the actual data.
      * Dropping an external table does not delete the underlying files.

|  |
| --- |
| DROP EXTERNAL TABLE SpecialOrders; |

5.2 Encapsulate data transformations in a stored procedure

* Whenever you need to transform data, it is a good practice to encapsulate the transformation operation in stored procedure.
* Store procedures make it easier to operationalize data transformations by enabling you to supply parameters, retrieve outputs, and include additional logic in a single procedure call.
* For example, the following code creates a stored procedure that drops the external table if it already exists before recreating it with order data for the specified year

|  |
| --- |
| CREATE PROCEDURE usp\_special\_orders\_by\_year @order\_year INT  AS  BEGIN  -- Drop the table if it already exists  IF EXISTS (  SELECT \* FROM sys.external\_tables  WHERE name = 'SpecialOrders'  )  DROP EXTERNAL TABLE SpecialOrders  -- Create external table with special orders  -- from the specified year  CREATE EXTERNAL TABLE SpecialOrders  WITH (  LOCATION = 'special\_orders/',  DATA\_SOURCE = files,  FILE\_FORMAT = ParquetFormat  )  AS  SELECT OrderID, CustomerName, OrderTotal  FROM  OPENROWSET(  BULK 'sales\_orders/\*.csv',  DATA\_SOURCE = 'files',  FORMAT = 'CSV',  PARSER\_VERSION = '2.0',  HEADER\_ROW = TRUE  ) AS source\_data  WHERE OrderType = 'Special Order'  AND YEAR(OrderDate) = @order\_year  END |
| * Dropping an existing external table does not delete the folder containing its data files. You must explicitly delete the target folder if it exists before running the stored procedure, or an error will occur. |

5.3 Include a data transformation stored procedure in a pipeline

* In Azure Synapse Analytics and Azure Data Factory, you can create pipelines that connect to *linked services*, including Azure Data Lake Store Gen 2 storage accounts that hosts data lake files, and serverless SQL pools – enabling you to call your stored procedures as part of an overall data ETL pipeline.

**6. Create a lake database in Azure Synapse Analytics**

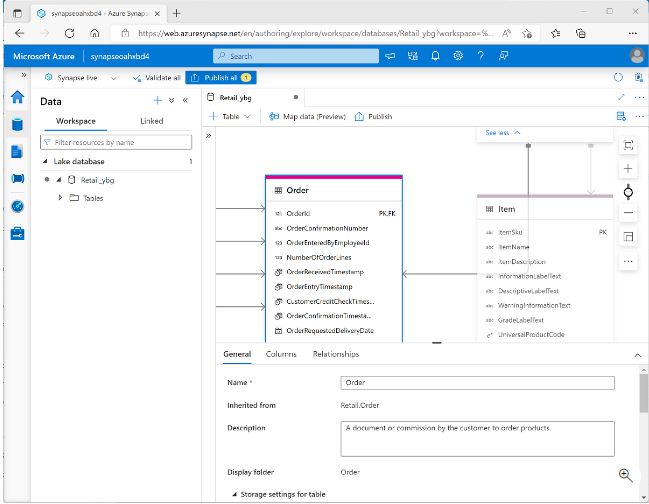
6.1 Understand Lake database concepts

|  |  |  |
| --- | --- | --- |
| **Relational Database** | **Data Lake** | **Data Lakehouse** |
| * Data for the tables is stored in the DB and it is tightly coupled to the table definition; which enforces data types, nullability, key uniqueness, and referential integrity between related keys. * All the queries and data manipulations must be performed through the database system. | * No fixed schema * Data stored in files (structured, semi, un) * Applications and data analysts can work directly with the files in the data lake using the tools of their choice, without the constraints of a relational database system. | * It provides a relational metadata layer over one or more files in a data lake. * You can create a lake database that includes definition for tables, including column names and data types as well as relationships between primary and foreign key columns. * The tables reference files in the data lake, enabling you to apply relational semantics to working with the data and querying it using SQL. * However the storage of the data files is decoupled from the database schema, enabling more flexibility than a relational database system. |

* Lake database schema
  + You can create a lake database in Azure Synapse Analytics, and define the tables that represent the entities for which you need to store the data.
  + You can apply proven data modelling principles to create relationships between tables and use appropriate naming convention for tables, columns, and other database objects.
  + You can create lake database from an empty schema, to which you add definitions for tables and the relationships between them. However, Azure Synapse Analytics provides a comprehensive collection of database templates that reflect common schemas found in multiple business scenarios.
    - For instance: Retail, Manufacturing, Fund Management, etc.
* Lake database storage
  + The data for the tables in your lake database is stored in the data lake as Parquet or CSV files.
  + The files can be managed independently of the database tables, making easier to mange data ingestion and manipulation with a variety of data processing tools and technologies.
* Lake database compute
  + To query and manipulate the data through the tables you have defined, you can use an Azure Synapse serverless SQL pool to run SQL queries or an Azure Synapse Apache Spark pool to work with the tables using the Spark SQL API.

6.3 Create and use a Lake Database

* You can create a LDB using the lake database designer in Azure Synapse Studio.
* Start by Adding a new lake database on the **Data** page, selecting a template from the gallery or starting with a blank lake database; and then add and customize tables using the visual database designer interface.
* As you create each table, you can specify the type and location of the files you want to use to store the underlying data or create a table from existing files that are already in the data lake.
  + In most cases, it is advisable to store all of the database files in a consistent format within the same root folder in the data lake.
* Database designer
  + Azure Synapse Studio’ DB designer provides a drag-and-drop on which you can edit the tables in your database and the relationships between them.



* Using the DB designer, you can define the schema for your DB by adding or removing tables and:
  + - Specifying the name and storage settings for each table.
    - Specifying the names, key usage, nullability, and data types for each column.
    - Defining relationships between key columns in tables.
* When your DB schema is ready for use, you can publish the DB and start using it.
* After creating a lake database, you can store data files that match the table schemas, in the appropriate folders in the data lake, and quey them using SQL.
  + - Using a serverless SQL pool

|  |
| --- |
| USE RetailDB;  GO  SELECT CustomerID, FirstName, LastName  FROM Customer  ORDER BY LastName; |
| * There is no need to use an OPENROWSET function or include any additional code to access the data from the underlying file storage. The serverless SQL pool handles the mapping to the files for you. |

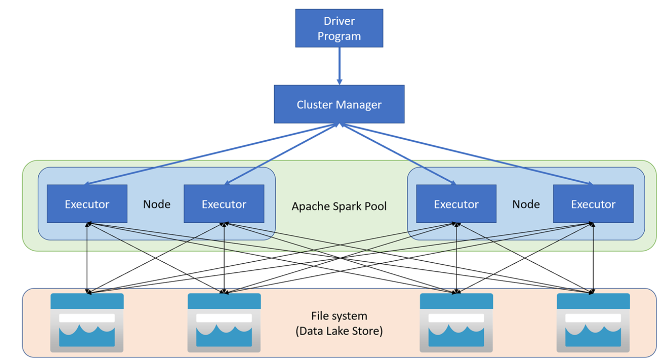
* + - Using an Apache Spark pool: You can work with lake database tables using Spark SQL in an Apache Spark pool.

|  |
| --- |
| %%sql  SELECT \* FROM `RetailDB`.`Customer` WHERE CustomerID = 123 |

**7. Analyze data with Apache Spark in Azure Synapse Analytics**

7.1 How Spark works

* Apache Spark is a distributed data processing framework that enables large-scale data analytics by coordinating work across multiple processing nodes in a cluster.
* Apache Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the driver program).
* The SparkContext connects to the cluster manager, which allocates resources across applications using an implementation of Apache Hadoop YARN. Once connected, Spark acquires executors on nodes in the cluster to run your application code.
* The SparkContext is responsible for converting an application to a directed acyclic graph (DAG). The graph consists of individual tasks that get executed within an executor process on the nodes. Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads.



* In Azure Synapse Analytics, a cluster is implemented as a *Spark pool*, which provides a runtime for Spark operations.
* You can create one or more Spark pools + you can specify the following configuration options:
  + A name for the spark pool.
  + The size of virtual machine (VM) used for the nodes in the pool, including the option to use accelerated GPU-enabled nodes.
  + The Number of nodes in the pool, and whether the pool size is fixed or individual nodes can be brought online dynamically to auto-scale the cluster, in which case you can specify the minimum and the maximum numbers of active nodes.
  + The version of the Spark Runtime to be used in the pool.

7.2 Use Spark in Azure Synapse Analytics

* You can run many kinds of application on Spark, incl. code in Python, Scala, JAR scripts.
* Spark is commonly used in two kinds of workloads:
  + Batch or stream processing jobs to ingest, clean, transform data.
  + Interactive analytics sessions to explore, analyse, and visualize the data.
* Azure Synapse Studio includes an integrated notebook interface for working with Spark.
* Accessing data from a Synapse Spark pool – you can use Spark in Azure Synapse Analytics to work with data from various sources, including:
  + A data lake based on the primary storage account for the Azure Synapse Analytics workspace;
  + A data lake based on storage defined as a *linked service* in the workspace;
  + A dedicated or serverless SQL pool in the workspace;
  + An Azure SQL or serverless SQL pool in the workspace;
  + An Azure Cosmos DB analytical database defined as a *linked service* and configured using *Analytics Synapse Link for Cosmos DB*;
  + An Azure Data Explorer Kusto database defined as a linked service in the workspace.
  + An external Hive metastore defines as linked service in the workspace.
* One of the most common uses of Spark is to work with data in a data lake, where you can read and write files in multiple commonly used formats, included delimited text, Parquet, Avro, and others.

7.3 Analyse data with Spark

* You can write and run code in various programming languages.
* The default is *PySpark*, a Spark-optimized version of Python. Additionally, you can use languages such as Scala (a Java-derived language) and SQL.

*Exploring data with dataframes*

* The most commonly used data structure for working with structured data in Spark is the *dataframe*, which is provided as part of the *Spark SQL* library. For example:

|  |
| --- |
| ProductID,ProductName,Category,ListPrice  771,"Mountain-100 Silver, 38",Mountain Bikes,3399.9900  772,"Mountain-100 Silver, 42",Mountain Bikes,3399.9900  773,"Mountain-100 Silver, 44",Mountain Bikes,3399.9900  ...   * This is ‘products.csv’, saved in the primary storage account for an Azure Synapse Analytics workspace. |
| %%pyspark  df = spark.read.load('abfss://container@store.dfs.core.windows.net/products.csv',  format='csv',  header=True  )  display(df.limit(10))   * In a Spark notebook, you could use the following PySpark code to load the data into a dataframe and display the first 10 rows. * The %%pyspark line at the beginning is called a magic, and tells Spark that the language used in this cell is PySpark. |
| from pyspark.sql.types import \*  from pyspark.sql.functions import \*  productSchema = StructType([  StructField("ProductID", IntegerType()),  StructField("ProductName", StringType()),  StructField("Category", StringType()),  StructField("ListPrice", FloatType())  ])  df = spark.read.load('abfss://container@store.dfs.core.windows.net/product-data.csv',  format='csv',  schema=productSchema,  header=False)  display(df.limit(10))   * The following PySpark example shows how to specify a schema for the dataframe to be loaded from a file named **product-data.csv** in this format. |

* You can use the methods of the dataframe class to filter / sort / group, and otherwise manipulate the data it contains.

*Using SQL expressions in Spark*

* The Dataframe API is part of a Spark library named Spark SQL, which enables data analysts to use SQL expressions to query and manipulate data:
  + Create a database object in the Spark catalog
    - The Spark catalog is a metastore for relational data objects such as views and tables.
    - The Spark runtime can use the catalog to seamlessly integrate code written in any Spark-supported language with SQL expressions that may be more natural to some data analysts or developers.
    - One of the simplest ways to make data in a dataframe available for querying in the Spark catalog is to create a temporary view.

|  |
| --- |
| df.createOrReplaceTempView("products") |

* + - A view is temporary, meaning that it's automatically deleted at the end of the current session. You can also create tables that are persisted in the catalog to define a database that can be queried using Spark SQL.
  + Use the Spark SQL API to query data
    - You can use the Spark SQL API in code written in any language to query data in the catalog. For example, the following PySpark code uses a SQL query to return data from the products view as a dataframe.

|  |
| --- |
| bikes\_df = spark.sql("SELECT ProductID, ProductName, ListPrice \  FROM products \  WHERE Category IN ('Mountain Bikes', 'Road Bikes')")  display(bikes\_df) |

* + Use SQL code
    - In a notebook, you can also use the %%sql magic to run SQL code that queries objects in the catalog, like this:

|  |
| --- |
| %%sql  SELECT Category, COUNT(ProductID) AS ProductCount  FROM products  GROUP BY Category  ORDER BY Category |

*Using SQL expressions in Spark*

* Notebooks in Azure Synapse Analytics provide some basic charting capabilities in the user interface, and when that functionality doesn't provide what you need, you can use one of the many Python graphics libraries to create and display data visualizations in the notebook.

**8. Transform data with Spark in Azure Synapse Analytics**

* Apache Spark provides a powerful platform for performing data cleansing and transformation tasks on large volumes of data.
* By using the Spark dataframe object, you can easily load data from files in a data lake and perform complex modifications. You can then save the transformed data back to the data lake for downstream processing or ingestion into a data warehouse.
* Azure Synapse Analytics provides Apache Spark pools that you can use to run Spark workloads to transform data as part of a data ingestion and preparation workload.
* You can use natively supported notebooks to write and run code on a Spark pool to prepare data for analysis. You can then use other Azure Synapse Analytics capabilities such as SQL pools to work with the transformed data.

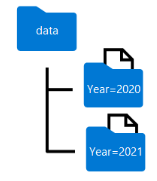
|  |
| --- |
| **Modify and save dataframes**  order\_details = spark.read.csv('/orders/\*.csv', header=True, inferSchema=True)  display(order\_details.limit(5))   * To load data into a dataframe, you use the spark.read function, specifying the file format, path, and optionally the schema of the data to be read. For example, the following code loads data from all .csv files in the orders folder into a dataframe named order\_details and then displays the first five records. |
| **Transform the data structure**  from pyspark.sql.functions import split, col  # Create the new FirstName and LastName fields  transformed\_df = order\_details.withColumn("FirstName", split(col("CustomerName"), " ").getItem(0)).withColumn("LastName", split(col("CustomerName"), " ").getItem(1))  # Remove the CustomerName field  transformed\_df = transformed\_df.drop("CustomerName")  display(transformed\_df.limit(5))   * You can use the full power of the Spark SQL library to transform the data by filtering rows, deriving, removing, renaming columns, and any applying other required data modifications. |
| **Save the transformed data**  transformed\_df.write.mode("overwrite").parquet('/transformed\_data/orders.parquet')  print ("Transformed data saved!") |

*Partition data files*

* Partitioning is an optimization technique that enables spark to maximize performance across the worker nodes.
* More performance gains can be achieved when filtering data in queries by eliminating unnecessary disk IO.
* To save a dataframe as a partitioned set of files, use the *partitionBy* method when writing the data.

|  |
| --- |
| from pyspark.sql.functions import year, col  # Load source data  df = spark.read.csv('/orders/\*.csv', header=True, inferSchema=True)  # Add Year column  dated\_df = df.withColumn("Year", year(col("OrderDate")))  # Partition by year  dated\_df.write.partitionBy("Year").mode("overwrite").parquet("/data") |

* The folder names generated when partitioning a database include the partitioning column name and value in a **column=value** format



* You can partition data by multiple columns, which results in a hierarchy of folders for each partitioning key.
* When reading data from parquet files into a dataframe, you have the ability to pull data from any folder within the hierarchical folders. The filtering process is done with the use explicit values and wildcards against the partitioned fields.

|  |
| --- |
| orders\_2020 = spark.read.parquet('/partitioned\_data/Year=2020')  display(orders\_2020.limit(5))   * The partitioning columns specified in the file path are omitted in the resulting dataframe. The results produced by the example query would not include a Year column - all rows would be from 2020. |

*Transform data with SQL*

* The SparkSQL library, which provides the dataframe structure also enables you to use SQL as a way of working with data.
* With this approach, You can query and transform data in dataframes by using SQL queries, and persist the results as tables.
  + Tables are metadata abstractions over files. The data is not stored in a relational table, but the table provides a relational layer over files in the data lake.

|  |
| --- |
| # Create derived columns  sql\_transform = spark.sql("SELECT \*, YEAR(OrderDate) AS Year, MONTH(OrderDate) AS Month FROM sales\_orders")  # Save the results  sql\_transform.write.partitionBy("Year","Month").saveAsTable('transformed\_orders', format='parquet', mode='overwrite', path='/transformed\_orders\_table')   * The following code creates two new derived columns named Year and Month and then creates a new table transformed\_orders with the new derived columns added. |
| %%sql  SELECT \* FROM transformed\_orders  WHERE Year = 2021  AND Month = 1   * Because this new table was created in the metastore, you can use SQL to query it directly with the %%sql magic key in the first line to indicate that the SQL syntax will be used |

* Drop tables
  + When working with external tables, you can use the DROP command to delete the table definitions from the metastore without affecting the files in the data lake.
  + This approach enables you to clean up the metastore after using SQL to transform the data, while making the transformed data files available to downstream data analysis and ingestion processes.

|  |
| --- |
| %%sql  DROP TABLE transformed\_orders;  DROP TABLE sales\_orders; |

**9. Use Delta Lake in Azure Synapse Analytics**

* Delta Lake is an open-source storage layer for Spark that enables relational database capabilities for batch and streaming data.
* By using Delta Lake, you can implement a data lakehouse architecture in Spark to support SQL\_based data manipulation semantics with support for transactions and schema enforcement.
* The result is an analytical data store that offers many of the advantages of a relational database system with the flexibility of data file storage in a data lake.

9.1 Understand Delta Lake

* Delta Lake is an open-source storage layer that adds relational database semantics to Spark-based data lake processing.
* The benefits of using Delta Lake in a Synapse Analytics Spark pool include:
  + Relational tables that support querying and data modification
    - You can store data in tables that support CRUD (create, read, update, and delete).
    - In other words, you can select, insert, update, and delete rows of data in the same way you would in a relational database system.
  + Support for ACID transactions
    - Delta Lake brings this same transactional support to Spark by implementing a transaction log and enforcing serializable isolation for concurrent operations.
  + Data versioning and time travel
    - Because all transactions are logged in the transaction log, you can track multiple versions of each table row and even use the time travel feature to retrieve a previous version of a row in a query.
  + Support for batch and streaming data.
    - While most relational databases include tables that store static data, Spark includes native support for streaming data through the Spark Structured Streaming API.
    - Delta Lake tables can be used as both sinks (destinations) and sources for streaming data.
  + Standard formats and interoperability
    - The underlying data for Delta Lake tables is stored in Parquet format, which is commonly used in data lake ingestion pipelines.
    - Additionally, you can use the serverless SQL pool in Azure Synapse Analytics to query Delta Lake tables in SQL

9.2 Create Delta Lake tables

* Delta lake is built on tables, which provide a relational storage abstraction over files in a delta lake.
* One of the easiest ways to create a Delta Lake table is to save a dataframe in the *delta* format, specifying a path where the data files and related metadata information for the table should be stored.

|  |
| --- |
| # Load a file into a dataframe  df = spark.read.load('/data/mydata.csv', format='csv', header=True)  # Save the dataframe as a delta table  delta\_table\_path = "/delta/mydata"  df.write.format("delta").save(delta\_table\_path)   * After saving the delta table, the path location you specified includes parquet files for the data (regardless of the format of the source file you loaded into the dataframe) and a \_delta\_log folder containing the transaction log for the table. |
| new\_df.write.format("delta").mode("overwrite").save(delta\_table\_path)   * You can replace an existing Delta Lake table with the contents of a dataframe by using the overwrite mode   new\_rows\_df.write.format("delta").mode("append").save(delta\_table\_path)   * You can also add rows from a dataframe to an existing table by using the append mode: |

*Making conditional updates*

* While you can make data modifications in a dataframe and then replace a Delta Lake table by overwriting it, a more common pattern in a database is to insert, update or delete rows in an existing table as discrete transactional operations.
* To make such modifications to a Delta Lake table, you can use the DeltaTable object in the Delta Lake API, which supports update, delete, and merge operations. For example, you could use the following code to update the price column for all rows with a category column value of "Accessories"

|  |
| --- |
| from delta.tables import \*  from pyspark.sql.functions import \*  # Create a deltaTable object  deltaTable = DeltaTable.forPath(spark, delta\_table\_path)  # Update the table (reduce price of accessories by 10%)  deltaTable.update(  condition = "Category == 'Accessories'",  set = { "Price": "Price \* 0.9" })   * The data modifications are recorded in the transaction log, and new parquet files are created in the table folder as required. |

*Querying a previous version of a table*

* Delta Lake tables support versioning through the transaction log.
* The transaction log records modifications made to the table, noting the timestamp and version number for each transaction.
* You can use this logged version data to view previous versions of the table - a feature known as *time travel*.

|  |
| --- |
| df = spark.read.format("delta").option("versionAsOf", 0).load(delta\_table\_path)   * You can retrieve data from a specific version of a Delta Lake table by reading the data from the delta table location into a dataframe, specifying the version required as a versionAsOf option   df = spark.read.format("delta").option("timestampAsOf", '2022-01-01').load(delta\_table\_path)   * Alternatively, you can specify a timestamp by using the timestampAsOf option |

9.3 Create catalog tables

* So far we've considered Delta Lake table instances created from dataframes and modified through the Delta Lake API.
* You can also define Delta Lake tables as *catalog tables* in the Hive metastore for your Spark pool, and work with them using SQL.
* Tables in a Spark catalog, including Delta Lake tables, can be managed or external
  + Managed
    - Defined without a specified location, and the data files are stored within the storage used by the metastore.
    - Dropping the table, not only removes its metadata from the catalog, but also deletes the folder in which its data are stored.
  + External
    - Defined for a custom file location, where the data for the table is store. The metadata for the table is defined in the Spark catalogue.
    - Dropping the table deletes the metadata from the catalog, but does not affect the data files

*Creating catalog tables*

* There are several ways to create catalog tables:
  + from a dataframe

|  |
| --- |
| # Save a dataframe as a managed table  df.write.format("delta").saveAsTable("MyManagedTable")  ## specify a path option to save as an external table  df.write.format("delta").option("path", "/mydata").saveAsTable("MyExternalTable")   * You can create managed tables by writing a dataframe using the saveAsTable |

* + using SQL

|  |
| --- |
| spark.sql("CREATE TABLE MyExternalTable USING DELTA LOCATION '/mydata'")   * You can also create a catalog table by using the CREATE TABLE SQL statement with the USING DELTA clause, and an optional LOCATION parameter for external tables. You can run the statement using the SparkSQL API. |
| %%sql  CREATE TABLE MyExternalTable  USING DELTA  LOCATION '/mydata'   * Alternatively you can use the native SQL support in Spark to run the statement |

* + DeltaTableBuilder API
    - You can use the DeltaTableBuilder API (part of the Delta Lake API) to create a catalog table

|  |
| --- |
| from delta.tables import \*  DeltaTable.create(spark) \  .tableName("default.ManagedProducts") \  .addColumn("Productid", "INT") \  .addColumn("ProductName", "STRING") \  .addColumn("Category", "STRING") \  .addColumn("Price", "FLOAT") \  .execute() |

*Using catalog tables*

* You can use catalog tables like tables in any SQL-based relational database, querying and manipulating them by using standard SQL statements.

|  |
| --- |
| %%sql  SELECT orderid, salestotal  FROM ManagedSalesOrders |

9.4 Use Delta Lake with streaming data (TO BE FINISHED)

**10. Analyse data in a relational data warehouse**

* Relational data warehouses are at the centre of most enterprise business intelligence (BI) solutions. While the details may vary across data warehouse implementations, a common pattern based on a denormalized, multidimensional schema has emerged as the standard design for a relational data warehouse.
* Azure Synapse Analytics includes a highly scalable relational database engine that is optimized for data warehousing workloads.
* By using *dedicated SQL pools* in Azure Synapse Analytics, you can create databases that are capable of hosting and querying huge volumes of data in relational tables.

10.1 Design a data warehouse schema

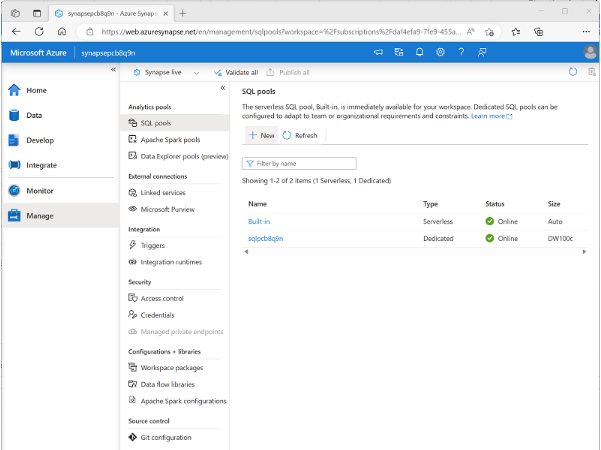
* Like all relational databases, a data warehouse contains tables in which the data you want to analyze is stored.
* These tables are organized in a schema that is optimized for multidimensional modeling, in which numerical measures associated with events known as facts can be aggregated by the attributes of associated entities across multiple dimensions.
* A common pattern for relational data warehouses is to define a schema that includes two kinds of tables: (i) dimension, (ii) facts tables.
  + Dimensions tables
    - They describe business entities.
    - They contain a unique key that uniquely identify each row in the table.
    - It's common for a dimension table to include **two** key columns:
      * *A surrogate key*, specific to the DW, usually am incrementing integer number.
      * *A natural key* (or *alternate, business)* used to identify a specific instance of an entity in the transactional source system from which the entity record originated.

|  |
| --- |
| Surrogate & Natural keys   * The data warehouse may be populated with data from multiple source systems, which can lead to the risk of duplicate or incompatible business keys. * Simple numeric keys generally perform better in queries that join lots of tables - a common pattern in data warehouses. * Attributes of entities may change over time - for example, a customer might change their address. Since the data warehouse is used to support historic reporting, you may want to retain a record for each instance of an entity at multiple points in time. |

* + - In addition to dimension tables that represent business entities, it's common for a data warehouse to include a dimension table that represents *time*. This table enables data analysts to aggregate data over temporal intervals. Depending on the type of data you need to analyze, the lowest granularity (referred to as the *grain*) of a time dimension could represent times (to the hour, second, millisecond, nanosecond, or even lower), or dates.
  + Facts tables
    - Fact tables store details of observations or events.
    - A fact table contains columns for numeric values that can be aggregated by dimensions.
    - In addition to the numeric columns, a fact table contains key columns that reference unique keys in related dimension tables.
    - A fact table's dimension key columns determine its grain.
* Data warehouse schema design
  + the dimension data is generally de-normalized to reduce the number of joins required to query the data.
  + Often a data warehouse is organized as a *star* schema, in which a fact table is directly related to the dimension table.
  + The attributes of an entity can be used to aggregate measures in fact tables over multiple hierarchical levels.
  + The attributes for each level can be stored in the same dimension table.
  + When entities has a large number of hierarchical attribute levels or some attributes are shared across dimensions (e.g. customers and stores which have a geographical address), it can make sense to apply some normalization to the dimension tables and create a snowflake schema.

10.2 Create data warehouse tables

* To create a relational data warehouse in Azure Synapse Analytics, you must create a dedicated SQL Pool. The simplest way to do this in an existing Azure Synapse Analytics workspace is to use the **Manage** page in Azure Synapse Studio.



* When provisioning a dedicated SQL pool, you can specify the following configuration settings:
  + A unique name for the dedicated SQL pool.
  + A performance level for the SQL pool, which can range from *DW100c* to *DW30000c*, impacting the cost per hour for the pool is running.
  + Whether to start with an empty pool or restore an existing DB from the backup.
  + The *collation* of the SQL pool, which determines sort order and string comparison rules for the DB (cannot be modified after creation).
* After creating a dedicated SQL pool, you can control its running state in the **Manage** page of Synapse Studio; pausing it when not required to prevent unnecessary costs.
* When designing a star schema model for small or medium sized datasets, you can use you preferred database, such as Azure SQL.
* For larger data sets, you may benefit from your data warehouse in Azure Synapse Analytics, instead of SQL server. It is important to understand some key differences when creating tables in Synapse Analytics.

|  |  |
| --- | --- |
|  | Dedicated SQL pools in Synapse Analytics |
| Data integrity constraints | * Dedicated SQL pools in Synapse Analytics don't support foreign key and unique constraints as found in other relational database systems like SQL Server. * This means that jobs used to load data must maintain uniqueness and referential integrity for keys, without relying on the table definitions in the database to do so. |
| Indexes | * While Synapse Analytics dedicated SQL pools support clustered indexes as found in SQL Server, the default index type is *clustered columnstore*. * This index type offers a significant performance advantage when querying large quantities of data in a typical DW schema and should be used where possible. * However, some tables may include data typed that can’t be included in a clustered *columnstore* index (for example, VARBINARY(MAX)), in which case a clustered index can be used instead). |
| Distribution | Azure Synapse dedicated SQL pools uses a **massively parallel processing (MPP)** architecture, as opposed to the symmetric multiprocessing (SMP) architecture used in most OLTP database systems.  In a MPP system, the data in a table is distributed for processing across a pool of nodes. Synapse analytics supports the following kinds of distribution:   * **Hash**: A deterministic hash value is calculated for the specified column and used to assign the row to a compute node. * **Round-robin:** Rows are distributed evenly across all compute nodes. * **Replicated:** A copy of the table is stored on each compute node. |

* The table type often determines which option to choose for distributing the table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table type** | **Table dimension** | **Recommended distribution option** | **Comments** |
| Dimension | Small | Replicated distribution | To avoid data shuffling when joining to distributed fact tables |
| Large | Hash distribution |  |
| Fact |  | Hash distribution | with clustered *columnstore* index to distribute fact tables across compute nodes. |
| Staging |  | round-robin |  |

* Creating dimension tables

|  |
| --- |
| CREATE TABLE dbo.DimCustomer  (  CustomerKey INT IDENTITY NOT NULL,  CustomerAlternateKey NVARCHAR(15) NULL,  CustomerName NVARCHAR(80) NOT NULL,  EmailAddress NVARCHAR(50) NULL,  Phone NVARCHAR(25) NULL,  StreetAddress NVARCHAR(100),  City NVARCHAR(20),  PostalCode NVARCHAR(10),  CountryRegion NVARCHAR(20)  )  WITH  (  DISTRIBUTION = REPLICATE,  CLUSTERED COLUMNSTORE INDEX  ); |
| * When you create a dimension table, ensure that the table definition includes surrogate and alternate keys. * It's often easiest to use an IDENTITY column to auto-generate an incrementing surrogate key. |
| CREATE TABLE dbo.DimGeography  (  GeographyKey INT IDENTITY NOT NULL,  GeographyAlternateKey NVARCHAR(10) NULL,  StreetAddress NVARCHAR(100),  City NVARCHAR(20),  PostalCode NVARCHAR(10),  CountryRegion NVARCHAR(20)  )  WITH  (  DISTRIBUTION = REPLICATE,  CLUSTERED COLUMNSTORE INDEX  );  CREATE TABLE dbo.DimCustomer  (  CustomerKey INT IDENTITY NOT NULL,  CustomerAlternateKey NVARCHAR(15) NULL,  GeographyKey INT NULL,  CustomerName NVARCHAR(80) NOT NULL,  EmailAddress NVARCHAR(50) NULL,  Phone NVARCHAR(25) NULL  )  WITH  (  DISTRIBUTION = REPLICATE,  CLUSTERED COLUMNSTORE INDEX  ); |
| * If you intend to use a snowflake schema in which dimension tables are related to one another, you should include the key for the parent dimension in the definition of the child dimension table (geographical address details from the **DimCustomer** table to a separate **DimGeography** dimension). |

* Most DW include a t*ime* dimension table that enables you to aggregate data by multiple hierarchical levels of time interval.
* A common pattern when creating a dimension table for dates is to use the numeric date in YYYYMMDD format as an integer surrogate key, and the date as DATE or DATETIME datatype as the alternative key.
* Fact tables include keys for each dimension to which they are related, and the attributes and numeric measures for specific event or observations that you want to analyze.
* Staging tables are used as temporary storage for data as it is being loaded into the data warehouse.
* A typical pattern is to structure the table to make it as efficient as possible to ingest the data from its external source (often files in a data lake) into the relational DB, and then use SQL statements to load the data from the staging tables into the dimension and fact tables.
* In some cases, if the data to be loaded is in files with an appropriate structure, it can be more effective to create external tables that reference the file location. This way, the data can be read directly from the source files instead of being loaded into the relational store

|  |
| --- |
| -- External data source links to data lake location  CREATE EXTERNAL DATA SOURCE StagedFiles  WITH (  LOCATION = 'https://mydatalake.blob.core.windows.net/data/stagedfiles/'  );  GO  -- External format specifies file format  CREATE EXTERNAL FILE FORMAT ParquetFormat  WITH (  FORMAT\_TYPE = PARQUET,  DATA\_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'  );  GO  -- External table references files in external data source  CREATE EXTERNAL TABLE dbo.ExternalStageProduct  (  ProductID NVARCHAR(10) NOT NULL,  ProductName NVARCHAR(200) NOT NULL,  ProductCategory NVARCHAR(200) NOT NULL,  Color NVARCHAR(10),  Size NVARCHAR(10),  ListPrice DECIMAL NOT NULL,  Discontinued BIT NOT NULL  )  WITH  (  DATA\_SOURCE = StagedFiles,  LOCATION = 'products/\*.parquet',  FILE\_FORMAT = ParquetFormat  );  GO |

10.3 Load data warehouse tables

* At a basic level, loading a data warehouse is typically achieved by adding new data from files in a data lake into tables in the data warehouse.
* The COPY statement is an effective way to accomplish this task

|  |
| --- |
| COPY INTO dbo.StageProducts  (ProductID, ProductName, ProductCategory, Color, Size, ListPrice, Discontinued)  FROM 'https://mydatalake.blob.core.windows.net/data/stagedfiles/products/\*.parquet'  WITH  (  FILE\_TYPE = 'PARQUET',  MAXERRORS = 0,  IDENTITY\_INSERT = 'OFF'  ); |

* In most cases, you should implement a data warehouse load process that performs tasks in the following order:
  + Ingest the new data to be loaded into a data lake, applying pre-load cleansing or transformations as required.
  + Load the data from files into staging tables in the relational data warehouse.
  + Load the dimension tables from the dimension data in the staging tables, updating existing rows or inserting new rows and generating surrogate key values as necessary.
  + Load the fact tables from the fact data in the staging tables, looking up the appropriate surrogate keys for related dimensions.
  + Perform post-load optimization by updating indexes and tables distribution statistics.
* After using the COPY statement to load data into staging tables, you can use a combination of INSERT, UPDATE, MERGE, and CREATE TABLE AS SELECT (CTAS) statements to load the staged data into dimension and fact tables.

**11. Load data into a relational data warehouse**

11.1 Load staging tables

|  |  |
| --- | --- |
| CREATE TABLE dbo.StageProduct  (  ProductID NVARCHAR(10) NOT NULL,  ProductName NVARCHAR(200) NOT NULL,  ProductCategory NVARCHAR(200) NOT NULL,  Color NVARCHAR(10),  Size NVARCHAR(10),  ListPrice DECIMAL NOT NULL,  Discontinued BIT NOT NULL  )  WITH  (  DISTRIBUTION = ROUND\_ROBIN,  CLUSTERED COLUMNSTORE INDEX  ); | |
| COPY INTO dbo.StageProduct  (ProductID, ProductName, ...)  FROM 'https://mydatalake.../data/  products\*.parquet'  WITH  (  FILE\_TYPE = 'PARQUET',  MAXERRORS = 0,  IDENTITY\_INSERT = 'OFF'  ); | CREATE EXTERNAL TABLE dbo.ExternalStageProduct  (  ProductID NVARCHAR(10) NOT NULL,  ProductName NVARCHAR(10) NOT NULL,  ...  )  WITH  (  DATE\_SOURCE = StagedFiles,  LOCATION = 'folder\_name/\*.parquet',  FILE\_FORMAT = ParquetFormat  );  GO |
| * This is generally the recommended approach to load staging tables due to its high performance throughput. | * if the data to be loaded is stored in files with an appropriate structure, it can be more effective to create external tables that reference the file location. * This way, the data can be read directly from the source files instead of being loaded into the relational store. |

11.2 Load dimension table

* After staging dimension data, you can load it into dimension tables using SQL.
* Two main way to load dimension tables

|  |  |
| --- | --- |
| Using a CREATE TABLE AS (CTAS) statement  CREATE TABLE dbo.DimProduct  WITH  (  DISTRIBUTION = REPLICATE,  CLUSTERED COLUMNSTORE INDEX  )  AS  SELECT ROW\_NUMBER() OVER(ORDER BY ProdID) AS ProdKey,  ProdID as ProdAltKey,  ProductName,  ProductCategory,  Color,  Size,  ListPrice,  Discontinued  FROM dbo.StageProduct;  You can also load a combination of new and updated data into a dimension table by using a CTAS statement to create a new table that UNIONs the existing rows from the dimension table with the new and updated records from the staging tables  You can't use IDENTITY to generate a unique integer value for the surrogate key when using a CTAS statement, so this example uses the ROW\_NUMBER function to generate an incrementing row number for each row in the results ordered by the ProductID business key in the staged data. | Using an INSERT statement  When you need to load staged data into an existing dimension table, you can use an INSERT statement. This approach works if the staged data contains only records for new dimension entities (not updates to existing entities).  INSERT INTO dbo.DimCustomer  SELECT CustomerNo AS CustAltKey,  CustomerName,  EmailAddress,  Phone,  StreetAddress,  City,  PostalCode,  CountryRegion  FROM dbo.StageCustomers  Assuming the DimCustomer dimension table is defined with an IDENTITY CustomerKey column for the surrogate key (as described in the previous unit), the key will be generated automatically and the remaining columns will be populated using the values retrieved from the staging table by the SELECT query. |

11.3 Load time dimension table

* Time dimension tables store a record for each time interval based on the grain by which you want to aggregate data over time. For example, a time dimension table at the date grain contains a record for each date between the earliest and latest dates referenced by the data in related fact tables.
* Scripting this in SQL may be time-consuming in a dedicated SQL pool – it may be more efficient to prepare the data in Microsoft Excel or an external script and import it using the COPY statement.
* As the data warehouse is populated in the future with new fact data, you periodically need to extend the range of dates related time dimension tables.

11.4 Load slowly changing dimension tables

* Types of slowly changing dimension
  + Type 0: Data can’t be changed.
  + Type 1: Dimension record is updated in-place. Changes are made to an existing dimension row applied to all previously loaded facts to the dimension.
  + Type 2: Change to a dimension result in a new dimension row.
    - Existing rows for previous versions of the dimension are retained for historical fact analysis and the new row is applied to future fact table entries.
    - Type 2 dimensions often include columns to track the effective time periods for each version of an entity, and/or a flag to indicate which row represents the current version of the entity.

|  |
| --- |
| -- New Customers  INSERT INTO dbo.DimCustomer  SELECT stg.\*  FROM dbo.StageCustomers AS stg  WHERE NOT EXISTS  (SELECT \* FROM dbo.DimCustomer AS dim  WHERE dim.CustomerAltKey = stg.CustNo)  -- Type 1 updates (name)  UPDATE dbo.DimCustomer  SET CustomerName = stg.CustomerName  FROM dbo.StageCustomers AS stg  WHERE dbo.DimCustomer.CustomerAltKey = stg.CustomerNo;  -- Type 2 updates (StreetAddress)  INSERT INTO dbo.DimCustomer  SELECT stg.\*  FROM dbo.StageCustomers AS stg  JOIN dbo.DimCustomer AS dim  ON stg.CustNo = dim.CustomerAltKey  AND stg.StreetAddress <> dim.StreetAddress; |
| * it's assumed that an incrementing surrogate key based on an IDENTITY column identifies each row, and that the highest value surrogate key for a given alternate key indicates the most recent or "current" instance of the dimension entity associated with that alternate key. * In practice, many data warehouse designers include a Boolean column to indicate the current active instance of a changing dimension or use DateTime fields to indicate the active time periods for each version of the dimension instance. * With these approaches, the logic for a type 2 change must include an INSERT of the new dimension row and an UPDATE to mark the current row as inactive. |
| MERGE dbo.DimProduct AS tgt  USING (SELECT \* FROM dbo.StageProducts) AS src  ON src.ProductID = tgt.ProductBusinessKey  WHEN MATCHED THEN  -- Type 1 updates  UPDATE SET  tgt.ProductName = src.ProductName,  tgt.ProductCategory = src.ProductCategory,  tgt.Color = src.Color,  tgt.Size = src.Size,  tgt.ListPrice = src.ListPrice,  tgt.Discontinued = src.Discontinued  WHEN NOT MATCHED THEN  -- New products  INSERT VALUES  (src.ProductID,  src.ProductName,  src.ProductCategory,  src.Color,  src.Size,  src.ListPrice,  src.Discontinued); |
| * As an alternative to using multiple INSERT and UPDATE statements, you can use a single MERGE statement to perform an "upsert" operation to insert new records and update existing ones. |

11.5 Load fact tables

* Load operation loads fact tables after dimension tables. This approach ensures that the dimensions to which facts will be related are already present in the data warehouse.
* When working with slowly changing dimensions tables, the appropriate version of the dimension record must be identified to ensure the correct surrogate key is used to match the event recorded in the fact table with the state of the dimension at the time the fact occurred.
* In many cases, you can retrieve the latest “current” version of the dimension; but in some cases you might need to find the right dimension record based on DateTime that indicate the period of validity for each version of the dimension.

|  |
| --- |
| INSERT INTO dbo.FactSales  SELECT (SELECT MAX(DateKey)  FROM dbo.DimDate  WHERE FullDateAlternateKey = stg.OrderDate) AS OrderDateKey,  (SELECT MAX(CustomerKey)  FROM dbo.DimCustomer  WHERE CustomerAlternateKey = stg.CustNo) AS CustomerKey,  (SELECT MAX(ProductKey)  FROM dbo.DimProduct  WHERE ProductAlternateKey = stg.ProductID) AS ProductKey,  (SELECT MAX(StoreKey)  FROM dbo.DimStore  WHERE StoreAlternateKey = stg.StoreID) AS StoreKey,  OrderNumber,  OrderLineItem,  OrderQuantity,  UnitPrice,  Discount,  Tax,  SalesAmount  FROM dbo.StageSales AS stg |
| * The following example assumes that the dimension records have an incrementing surrogate key, and that the most recently added version of a specific dimension instance (which will have the highest key value) should be used. |

11.6 Perform post load optimization

* After loading new data into a DW, it is a good idea to rebuild the table indexes and update statistics on commonly queried columns.

|  |
| --- |
| ALTER INDEX ALL ON dbo.DimProduct REBUILD |

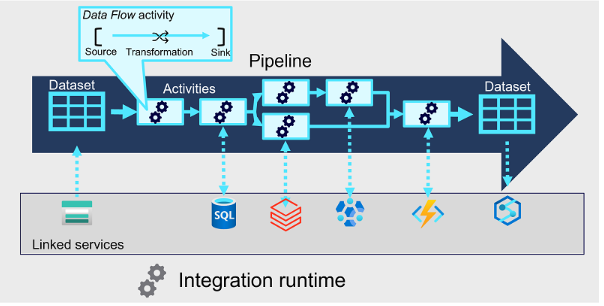
**12. Build a data pipeline in Azure Synapse Analytics**

* Azure Synapse Analytics Pipelines are built on the same technology as Azure Data Factory, and offer a similar authoring experience. The authoring process described in this model are also applicable to ADF.

12.1 Understand pipelines in Azure Synapse Analytics

* Pipelines encapsulate a sequence of *activities* that perform data movement and processing tasks.
* You can use a pipeline to define data transfer and transformation activities and orchestrate these activities through control flow activities that manage branching, looping, and other typical processing logic.

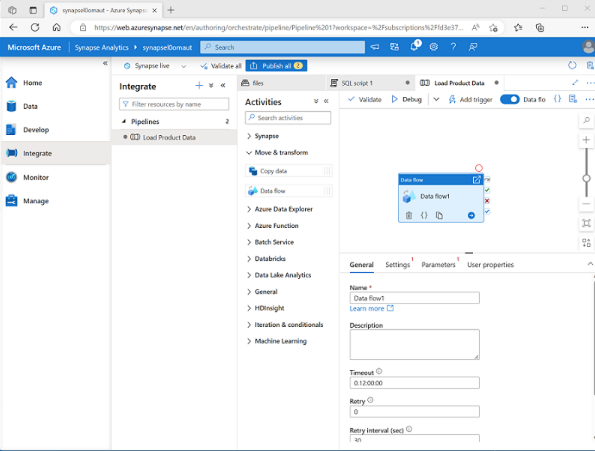
*Core pipeline concepts*



* Activities
  + Activities are the executable tasks in a pipeline.
  + You can define a flow of activities by connecting them in a sequence.
  + The outcome of a particular activity (success, failure, or competition) can be used to direct the flow to the next activity in the sequence.
  + Activities can encapsulate:
    - data transfer operations, including simple data copy operations that extract data from a source and load it to a target (or sink)
    - more complex data flows that apply transformation to the data as part of an ETL operation.
    - Processing tasks on specific systems, such as running a Spark notebook or calling an Azure function.
  + Finally, there are control flow activities that you can use to implement loops, conditional branching, or managing variable and parameter values.
* Integration runtime
  + The pipeline requires compute resources and an execution context in which to run.
  + The pipeline’s integration runtime provides the context, and is used to initiate and coordinate the activities in the pipeline.
* Linked services
  + While many of the activities are run directly in the integration runtime for the pipeline, some activities depend on external services.
    - For example, a pipeline might include an activity to run a notebook in Azure Databricks or to call a stored procedure in Azure SQL Database.
    - To enable secure connections to the external services used by your pipelines, you must define linked services for them.
    - Linked services are defined at the Azure Synapse Analytics workspace level and can be shared across multiple pipelines.
* Datasets
  + Most pipelines process data, and the specific data that is consumed and produced by activities in a pipeline is defined using *datasets*.
  + A dataset defines the schema for each data object that will be used in the pipeline and has an associated linked service to connect to its source. Activities can have datasets as inputs or outputs.
  + Similarly to linked services, datasets are defined at the Azure Synapse Analytics workspace level, and can be shared across multiple pipelines.

12.2 Create a pipeline in Azure Synapse Studio

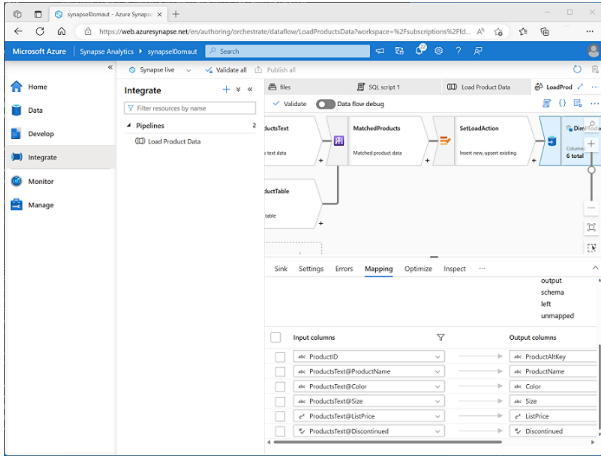
* You can create a pipeline in Azure Synapse Studio by using shortcuts on the Home page, but the primary place where pipelines are created and managed is the **Integrate page**, using the graphical design interface.



* The pipeline designer includes a set of activities, organized by categories, which you can drag onto a visual design canvas.
* You can select each activity on the canvas and use the properties pane beneath the canvas to configure the settings for that activity.
* To define the logical sequence of activities, you can connect them by using the Succeeded, Failed, and Completed dependency conditions, which are shown as small icons on the right-hand edge of each activity.
* Defining a pipeline with JSON
  + While the graphical development environment is the preferred way to create a pipeline, you can also create or edit the underlying JSON definition of a pipeline.

12.3 Define data flows

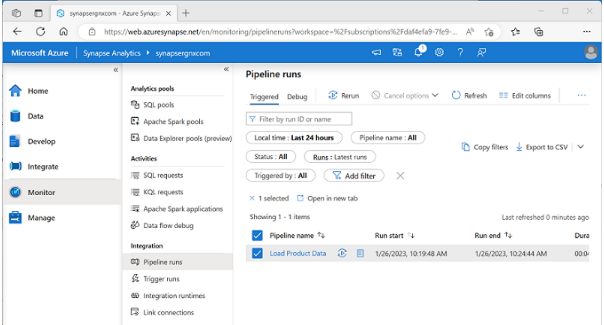
* A Data Flow is a commonly used activity type to define data flow and transformation.
* Data flows consists of:
  + *Sources*: The input to be transferred
  + *Transformations*: Various operations that you can apply to data as it streams through the data flow.
  + *Sinks*: Targets into which the data will be loaded.
* When you add a data flow activity to a pipeline, you can open it in a separate graphical design interface in which to create and configure the required data flow elements.



* An important part of creating a data flow is to define mappings for the columns as the data flows through the various stages, ensuring column names and data types are defined appropriately.
* While developing a data flow, you can enable the Data Flow debug option to pass a subset of data through the flow – which can be useful to test that your columns are mapped correclty

12.4 Run a pipeline

* When you are ready, you can publish a pipeline and use a trigger to run it.
* Triggers can be defined to run the pipeline:
  + Immediately
  + At explicitly scheduled intervals
  + In response to an event, such as new data files being added to a folder in a data lake.
* You can monitor each individual run of a pipeline in the **Monitor** page in Azure Synapse Studio.
* The ability to monitor past and ongoing pipeline runs is useful for troubleshooting purposes. Additionally, when combined with the ability to integrate Azure Synapse Analytics and Microsoft Purview, you can use pipeline run history to track data lineage data flows.



**13. Use Spark notebooks in an Azure Synapse Pipeline**

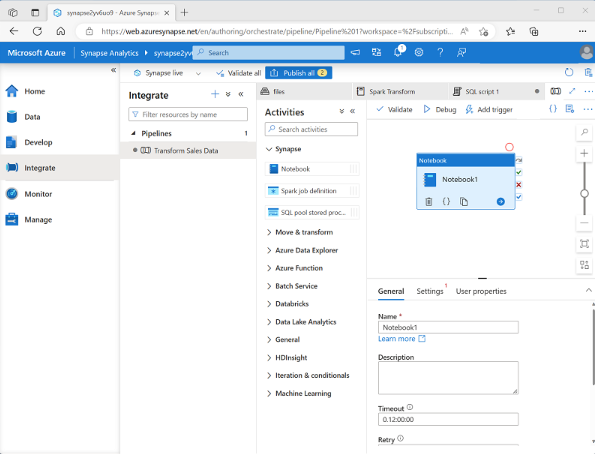
* With Azure Synapse Analytics pipelines, you can orchestrate data transfer and transformation activities and build data integration solutions across multiple systems.
* When you are working with analytical data in a data lake, Apache Spark provides a scalable, distributed processing platform that you can use to process huge volumes of data effectively.
* The Synapse Notebook activity enables you to run data processing code in Spark notebooks as a task in a pipeline; making it possible to automate big data processing and integrate into ETL workloads.

13.1 Understand Spark Notebooks in an Azure Synapse Pipelines

* Azure Synapse Pipelines enable you to create run and manage data integration and data flow activities.
* However, you can also use external processing resources to perform specific task. For instance, Apache Spark pool, on which you can run code in a notebook.
* It is common to use Spark notebooks for initial data exploration when designing data transformation processes. When it completed, you can include the notebook in a pipeline. The pipeline can then be run on a schedule or in response to an event (such as new data files being loaded into the data lake).
* The notebook is run on a Spark pool, which you can configure with the appropriate compute resources and Spark runtime for your specific workload. The pipeline is run in an integration runtime that orchestrates the activities in the pipeline, coordinating the external services needed to run them.

13.2 Use a Synapse notebook activity in a pipeline

* To run a Spark notebook in a pipeline, you must add a notebook and configure it appropriately. You’ll find the Notebook activity in the Synapse section of the activities pane in the Azure Synapse Analytics pipeline designer.



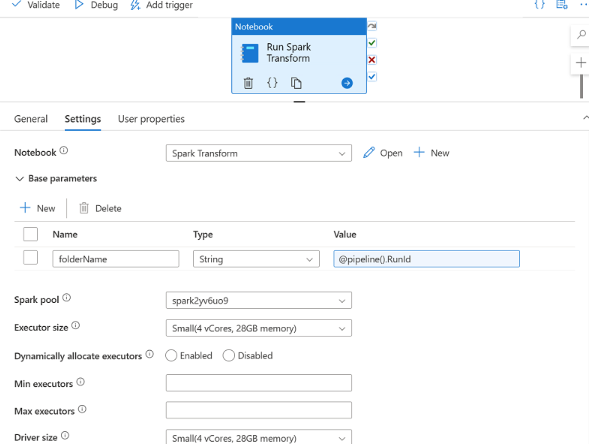
* To configure the notebook activity, edit the settings in the properties pane beneath the pipeline designer canvas. Notebook activity specific settings include:
  + Notebook
    - The notebook you want to run (existing one or new)
  + Spark pool
    - The Apache Spark pool on which the notebook should be run
  + Executor size
    - The node size for the worker nodes in the pool, which determines the number of processor cores and the amount of memory allocated to worker nodes.
  + Dynamically allocate executors
    - Configures Spark dynamic allocation, enabling the pool to automatically scale up and down to support the workload
  + Min and Max executors
    - The min and the max number of executors to be allocated.
  + Driver size
    - The node size for the driver node

13.3 Parameters in a notebook

* Parameters enable you to dynamically pass values for variables in the notebook each time is run. This approach provides flexibilities, enabling you to adjust the logic encapsulated in the notebook for each run.
* Create a parameters cell in the notebook
  + To define the parameters for a notebook, you declare and initialize variables in a cell, which you then configure as a **Parameters** cell by using the toggle option in the notebook editor interface.



* + Initializing a variable ensures that it has a default value, which will be used if the parameter isn't set in the notebook activity.
* Set base parameters for the notebook activity
  + After defining a parameters cell in the notebook, you can set values to be used when the notebook is run by a notebook activity in a pipeline. To set parameter values, expand and edit the Base parameters section of the settings for the activity.



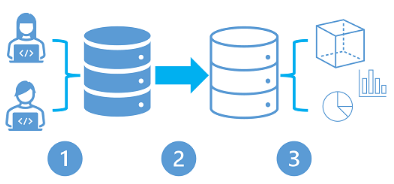
* + You can assign explicit parameter values or use an expression to assign a dynamic value. For example, the expression *@pipeline().RunId* returns the unique identifier for the current run of the pipeline.

**14. Plan hybrid transactional and analytical processing using Azure Synapse Analytics**

* HTAP is a style of data processing that combines transactional and analytical processing.
* In Azure Synapse Analytics, HTAP capabilities are provided by multiple Azure Synapse Link services, each connecting a commonly used transactional data store to your Azure Synapse Analytics workspace and making the data available for processing using Spark or SQL.

14.1 Understand Hybrid transactional and analytical processing patterns

* Many business application architectures separate transactional and analytical processing into separate systems with data stored and processed on separate infrastructures.
  1. OLTP
     + Systems are optimized for dealing with discrete systems or user requests immediately and responding as quickly as possible.
  2. OLAP
     + Systems are optimized for analytical processing, ingesting, synthesizing, and managing large sets of historical data.
     + The data processed by OLAP systems largely originates from OLTP systems and needs to be loaded into the OLAP systems by ETL.
     + Due to their complexity and the need to physically copy large amounts of data, this approach creates a delay in data being available to analyse in OLAP systems.
  3. HTAP
     + As more business move to digital processes, they increasingly recognize the value of being able to respond to opportunities by making faster and well-informed decisions.
     + HTAP enables business to run advanced analytics in near-real-time on data stored and processed by OLTP



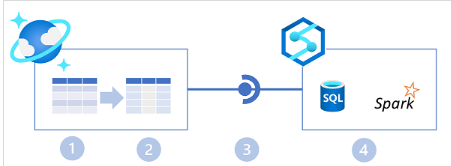
* + - A business application processes user input and stores data in a transactional DB that is optimized for a mix of data reads and writes based on the application’s expected usage profile.
    - The application data is automatically replicated to an analytical store with low latency.
    - The analytical store supports data modelling, analytics, and reporting without impacting the transactional system.

14.2 Describe Azure Synapse Link

* HTAP are supported through Azure Synapse Link, a general term for a set of linked services that support HATAP data sync into Azure Synapse Analytics workspace.

*Azure Synapse Link for Cosmos DB*

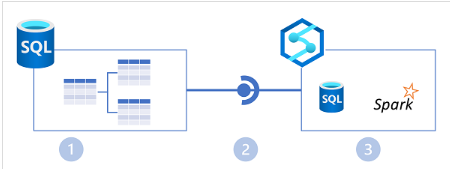
* Azure Cosmos DB is a global-scale NoSQL data service in Azure that enables applications to store and access operational data by using a choice of an APIs.
* Azure Synapse Link for Azure Cosmos DB is a cloud-native HTAP capability that enables you to run near-real-time analytics over operational data stored in a Cosmos DB container.
* Azure Synapse Link creates a tight seamless integration between Azure Cosmos DB and Azure Synapse Analytics.



* The architecture:
  + An Azure Cosmos DB container provides:
    - (1) a row-based transactional store that is optimized for read/write operations.
    - (2) a column-based analytical store that it is optimized for analytical workloads.
  + Azure Synapse Link provides a linked service that connects the analytical store enabled container in Azure Cosmos DB to an Azure Synapse Analytical workspace (3).
  + Azure Synapse Analytics provides Synapse SQL and Apache Spark runtimes in which you can run code to retrieve, process, ana analyse data from Azure Cosmos DB analytical store without impacting the transactional data store in Azure Cosmos DB (4).

*Azure Synapse Link for SQL*

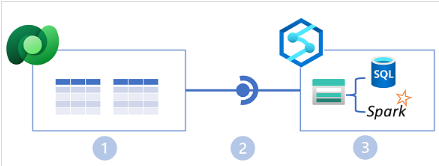
* Azure Synapse Link for SQL enables HTAP integration between data in SQL Server or Azure SQL Database and an Azure Synapse Analytics workspace.



* The architecture:
  + An Azure SQL Database or SQL Server instance contains a relational database in which transactional data is stored in tables (1).
  + Azure Synapse Link for SQL replicates the table data to a dedicated SQL pool in an Azure Synapse workspace (2).
  + An Azure SQL Database or SQL Server instance contains a relational database in which transactional data is stored in tables (3).

*Azure Synapse Link for Dataverse*

* Microsoft Dataverse is a data storage service within the Microsoft Power Platform (Power Apps, Power Bi, Power Virtual Agents, and other applications and services across M365, Dynamics 365, and Azure.
* Azure Synapse Link for Dataverse enables HTAP integration by replicating table data to Azure Data Lake storage, where it can be accessed by runtimes in Azure Synapse Analytics
  1. directly from the data lake or
  2. through a Lake Database defined in a serverless SQL pool.

****

* The architecture:
  + Business applications store data in Microsoft Dataverse tables (1).
  + Azure Synapse Link for Dataverse replicates the table data to an Azure Data Lake Gen2 storage account associated with an Azure Synapse workspace (2).
  + The data in the data lake can be used to define tables in a lake database and queried using a serverless SQL pool, or read directly from storage using SQL or Spark (3).

**15.** [**Implement Azure Synapse Link with Azure Cosmos DB**](https://learn.microsoft.com/en-gb/training/modules/configure-azure-synapse-link-with-azure-cosmos-db/)

* Azure Synapse Analytics Link for Cosmos DB enables hybrid transactional / analytical processing (HTAP) integration between Azure Cosmos DB and Azure Synapse Analytics.
* By using this HTAP solution, organizations can make operational data in real-time, without the need to develop a complex ETL pipeline.

15.1 [Implement Azure Synapse Link with Azure Cosmos DB](https://learn.microsoft.com/en-gb/training/modules/configure-azure-synapse-link-with-azure-cosmos-db/)

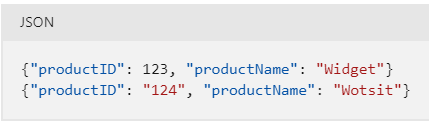
* The first step in using Azure Synapse Link for Cosmos DB is to enable it in an Azure Cosmos DB account. Azure Synapse Link is supported in the following types:
  + Azure Cosmos DB for NoSQL
  + Azure Cosmos DB for MongoDB
  + Azure Cosmos DB for Apache Gremlin (preview)
* You can enable Azure Synapse Link:
  + In the Azure portal page for your Cosmos DB account
    - ‘Integration’ section, direct link
  + By using the Azure CLI Azure PowerShell from a command line or in script.
* Considerations for enabling Azure Synapse Link
  + After enabling it, you can’t disable it.
  + Enabling Azure Synapse Link does not start sync of operational data to an analytical store – you must also create or update a container with support for an analytical store.
  + When enabling Azure Synapse Link for a Cosmos DB for NoSQL account using Azure CLI or PowerShell you can use the parameter to specify the schema type as *WellDefined* (default) or *FullFidelity*. For a Cosmos DB for MongoDB account, the default (and only supported) schema type is *FullFidelity*.
  + After a schema type has been assigned, you can’t change it.

15.2 [Create](https://learn.microsoft.com/en-gb/training/modules/configure-azure-synapse-link-with-azure-cosmos-db/) an analytical store enabled container

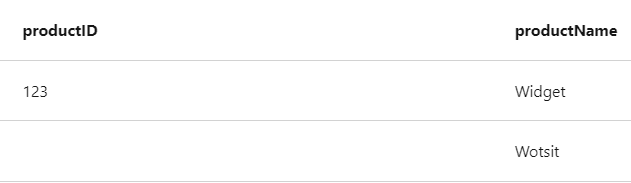
* After enabling Azure Synapse Link in an Azure Cosmos DB account, you can create or update a container with support for an analytical store.
* An analytical store is a column-based store within the same container as a row-based operational store.
* An *auto-sync* process synchronizes changes in the operational store to the analytical store – from where it can be queried without incurring processing overhead in the operational store.

*Analytical store schema types*

* As the data from the operational store is sync to the analytical store, the schema is updated dynamically to reflect the structure of the documents being sync.
* The specific behaviour of this dynamic schema maintenance depends on the analytical store schema type configured for the Azure Cosmos DB account.
* Two types of schema representation are supported:
  + Well-defined
    - The default schema type for an Azure Cosmos DB for NoSQL account.
  + Full fidelity
    - The default (and only supported) schema type for an Azure Cosmos DB for MongoDB account.
* The analytical store receives JSON data from the operational store and organizes it into a column-based structure.



* + In a well-defined schema
    - The first non-null occurrence of a JSON field determines the data type for that field.
    - Sub-sequent occurrences of the field that are not compatible with the assigned data type are not ingested into the analytical store.



* + In a full fidelity schema, the data type is appended to each instance of the field, with new columns created as necessary; enabling the analytical store to contain multiple occurrences of a field, each with a different data type, as shown in the following table.

