Documentation Tool for Business Intelligence Systems

Implementing a Mechanism to Parse BI-Related Files

LAB University of Applied Sciences
Bachelor of Engineering, Information and Communication Technology
2023
Roula Almouayad-Alazem

Abstract

Author(s)	Type of Publication	Published
Roula Almouayad-Alazem	Thesis, UAS	2023
	Number of Pages 424643**!Syntax Error, ERROR**4643	

Title of Publication

Documentation Tool for Business Intelligence Systems

Implementing a Mechanism to Parse BI-Related Files

Degree, Field of Study

Bachelor's Degree Program in Information and Communication Technology

Organization of the client

Pinja Digital Oy

Abstract

This thesis aims to continue the development of a business intelligence (BI) system documentation tool. The tool should collect data from different BI solution components so that users can relate data warehouse tables accessed and used by the BI solution. The paper focuses on BI solutions implemented in the Microsoft SQL Server using its related tools Microsoft provides. Data warehouses are held in SQL server database engines, and the Microsoft BI tools are SQL server integration services (SSIS), SQL server reporting services (SSRS), SQL server analysis services (SSAS) and PowerBI.

The tool was implemented using the C# language as a console application which executes manually or automatically as scheduled.

The tool parses the physical files of each BI component and inserts collected data into database tables devoted to this purpose. According to the user's preferences, collected data are modelled and analyzed, and eventually, visualized using any reporting tool.

Keywords

Business intelligence, BI, SSIS, SSRS, SSAS, PowerBI, C#

Contents

1 Introduction 1
2 Business Intelligence 2
2.1 Business Intelligence in Brief 2
2.2 Data Integration 2
2.3 Data Analysis 3
2.4 Reporting and Data Visualization 3
3 Terminologies and Technologies in Programming 4
3.1 Visual Studio 4
3.2 C Sharp 4
3.3 Libraries 4
3.4 Deserialization 5
4 SQL Server 6
4.1 What is SQL Server 6
4.2 Database Objects 7
5 SQL Server Services and Tools 8
5.1 Management Studio 8
5.2 Integration Services 8
5.2.1 SSIS Packages 9
5.3 Analysis Services 11
5.4 Reporting Services 12
5.5 SQL Server Agent 14
6 Other Microsoft Tools 15
6.1 PowerBI 15
6.2 Azure 15
7 BI Documentation Tool 17
7.1 Background: A Briefing of Current State Application 17
7.1.1 SSIS Parser 20
7.1.2 SSRS Parser 21
7.1.3 SSMS Parser 22
7.1.4 SQL Parser 22
7.2 The Implementation 23
7.2.1 SSAS Parser 24
7.2.2 PowerBI Parser (PbiParser) 25
7.2.3 SQL Server Jobs Collector 33

7.3 Changes/Enhancements to Existing Parsers 34
8 Test Case (End-user case) 37
9 Results and Conclusion 38
References 40

1 Introduction

As a business grows, the more data the business systems generate and the need for business intelligence, hereafter referred to as BI, dominates for efficient decision-making. BI comprises building a data warehouse which is used for analysis and reporting. Maintaining a clean data warehouse would be challenging when using different BI system components and tools. It requires continuous effort to check which database tables in the data warehouse are updated regularly and used in analysis and reporting services, or the data warehouse will build up inefficiently.

Pinja Group Oy is an international software company with over than 500 professionals with an estimated turnover of 59 million euros. It has offices around Finland and the headquarter is in Jyväskylä. Pinja provides software as a service (SaaS) solutions and digital services to customers in different industries such as manufacturing, forest, energy, and trade industries. Pinja Digital Oy is the unit that has over than 80 experts in BI. Pinja Digital professionals create BI solutions for each of their customers based on the requirements of these customers. The majority of BI solutions are built using the Microsoft products, such as SQL Server and its related business intelligence services, Integration Services (SSAS), Reporting Services (SSRS), and Analysis Services (SSAS), in addition to PowerBI. Some customers use Azure Data Factory in their BI system implementation.

The goal of this project was to continue the development and enhance an already implemented tool to facilitate the maintenance of the data warehouse. The implemented tool is a .Net console application, which parses different BI system-related files such as SSIS packages files and SSRS reports files along with other data in the SQL server, then inserts the collected data into dedicated database tables as a focal point of reference. The tool should be enhanced so that it collects data from all files the BI system generates, such as tabular and PowerBI reports, in addition to the previously mentioned sources. The tool should be dynamic so that it is usable by different customers of Pinja Group based on their diverse needs and preferences. Nevertheless, the tool is subject to continuous enhancements as more customers use it.

A customer of Pinja Group was considered the pilot end user, in which running the application successfully on their production environment should consider the project completed.

2 Business Intelligence

2.1 Business Intelligence in Brief

Business Intelligence is the process of collecting raw data and transforming them into better informative forms for efficient decision-making. According to Frankenfield (2020), business intelligence (BI) parses all data generated by a business into readable insights in the form of reports, performance measures and trends for better management decisions.

Figure 1 shows how the BI process works. The process involves collecting organization-related data from various internal or external sources, modelling them, and storing them in a centralized data warehouse. The data are then analysed, and findings are visualized in dashboards and reports, which will help decision-makers to make better business decisions. (Stedman, 2021.)

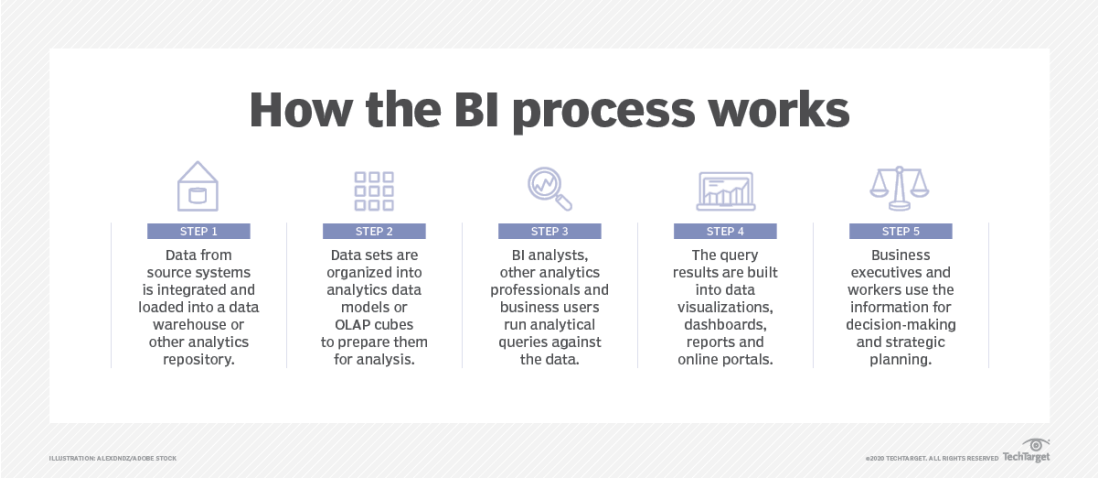

Figure 1. How BI works (Stedman, 2021)

2.2 Data Integration

ETL stands for extract, transform and load, is a process used by BI systems in integrating data. It extracts data from different systems, transforms data to ensure consistency, and loads data into the target database. (IBM.) The process involves collecting unstructured data from various sources, cleansing the collections of errors and mismatches, remodelling, and transforming data to a unified form, and validating the data before storing them in the data warehouse. (Stedman, Burns, & Pratt; IBM.)

2.3 Data Analysis

Data analysis involves performing queries against a data warehouse and breaking large data volumes to build structured smaller parts for faster analysis. These structured data parts are analysed database models ready for use by various visualization clients. Data models can be tabular or multidimensional models that include tables. Figure 2 illustrates the architecture of an SQL analysis service. Further analysis is conducted on these tables using measures. (Peterson.)

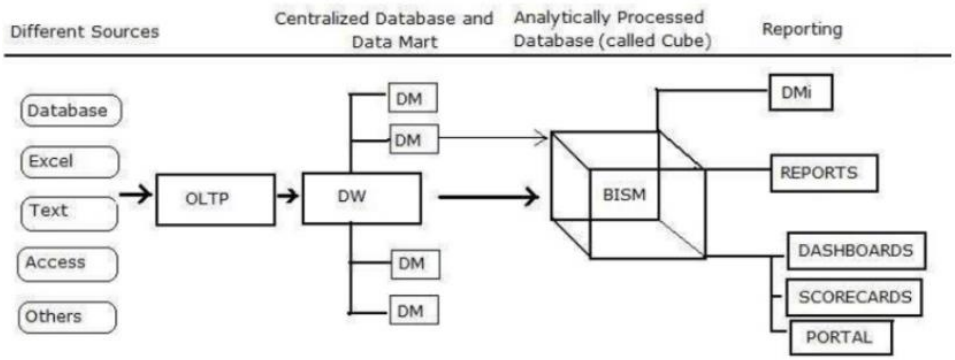

Figure 2. The architecture of SQL analysis services (Peterson)

2.4 Reporting and Data Visualization

Data visualization is translating analysed data into visual formats and reports, which help make informed decisions. The process involves relating different data and providing insights into patterns and trends. Reporting can include graphs, charts, tables, and others. (Power-BI.)

3 Terminologies and Technologies in Programming

3.1 Visual Studio

Visual Studio is an integrated development environment used by developers to create diverse types of applications. It is a tool that supports different programming languages such as C#, C++, JavaScript, and others. It is a desktop application used to develop, debug, test and deploy an application. (Visual Studio.)

3.2 C Sharp

C Sharp (C#) is a modern object-oriented programming language developers use to build secure applications (Microsoft h). The .Net platform should be installed so that C# can be used. The .Net is an open-source platform Microsoft provides to enable developers to create different applications (.Net).

In object-oriented programming a class is a primary concept. It is a template or a data structure for creating objects. It defines attributes and methods in one unit (Knowledgehut.) An Object is an instance of the class, which includes data as the values of the attributes. A method is a set of code statements that do not run unless the method is called. They are functions declared as members of a class and executed only from the instant of that object.

3.3 Libraries

Libraries in programming are a set of prewritten codes intended for shared use across different applications. Developers create instances of classes and can access their methods to perform the required task. (Sheldon). Visual Studio includes the NuGet package manager, built for the .Net platform to include thousands of .Net packages. Figure 3 shows the NuGet manager with the list of packages available.

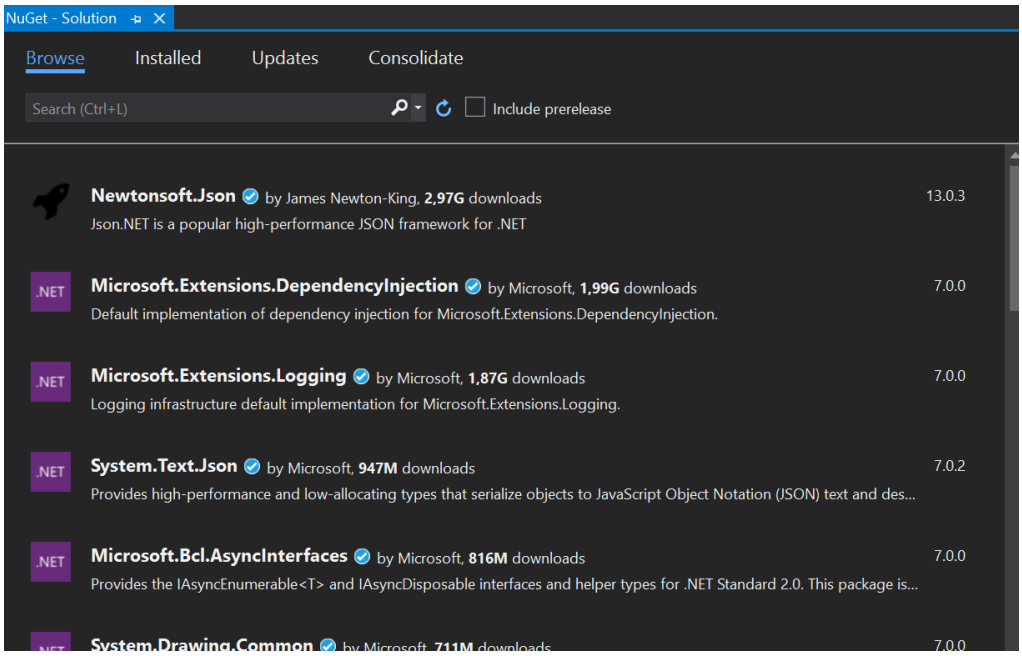

Figure 3. NuGet manager in Visual Studio

3.4 Deserialization

Deserialization is the process used to read the contents of a file into an object. It is the opposite of serialization, which converts an object to a byte stream to save it into memory (Figure 4). (Pedamkar.) In C# there are several types of serialization such as XML (Extensible Markup Language) and JSON (JavaScript Object Notation) serialization.

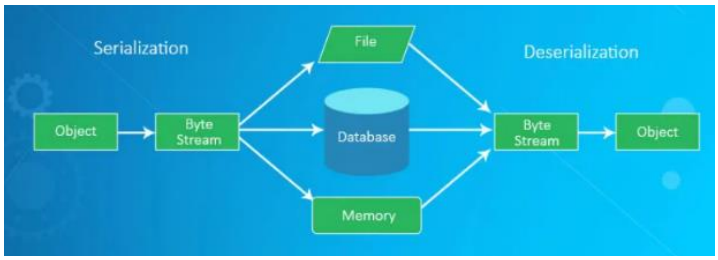

Figure 4. Serialization and deserialization process (Pedamkar)

4 SQL Server

4.1 What is SQL Server

SQL Server is a Microsoft product that manages relational databases. The server uses the T-SQL language, an extension of the standard query language (SQL) built by Microsoft. T-SQL offers functionalities that are not offered by standard SQL. The SQL server consists of two main components, the database engine, and the SQL operating system. The database engine is the core service for data storage and processing. It contains the storage engine, which is responsible for storing and fetching database data, and the relational engine, also called the query processor, which performs different tasks and memory management tasks in addition to processing queries against the storage engine. The database engine also manages other objects, such as views, stored procedures, and functions. Figure 5 shows the SQL server architecture, which contains the database engine with its two main components and the operating system SQLOS underneath it. The operating system provides several services, such as I/O and memory management, exception handling, and others. (SQLServerTutorial.) The SQL server includes variant data management and business intelligence tools and services.

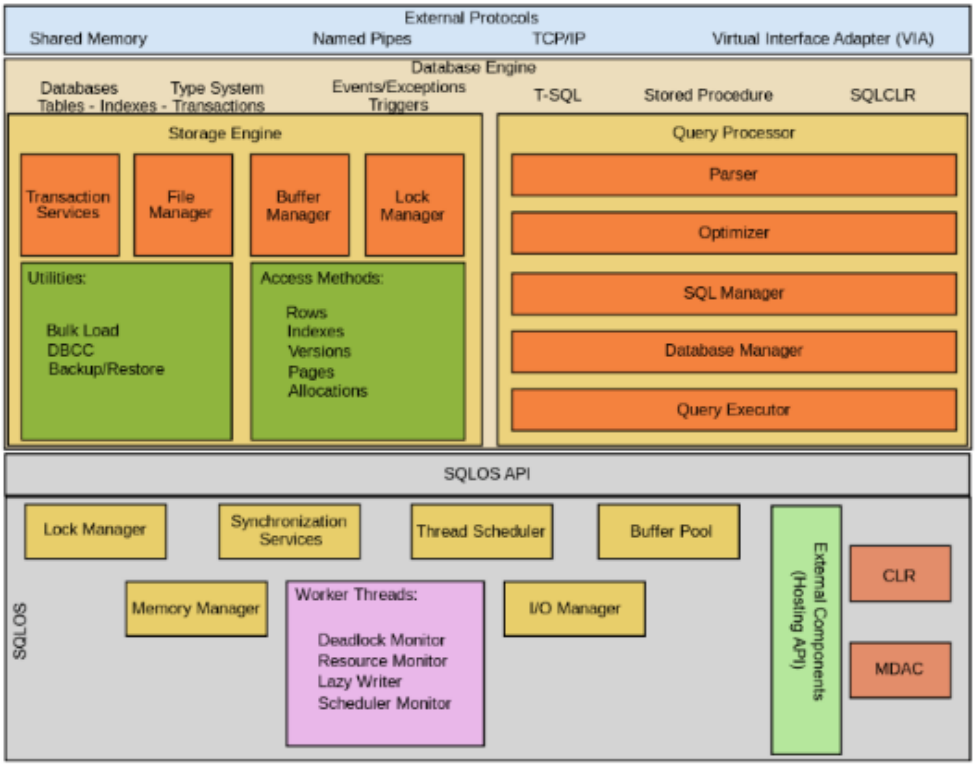

Figure 5. The architecture of the SQL server (SQLServerTutorial)

4.2 Database Objects

Database objects are data structures created into a database using the clause Create in the SQL statement. They are used to store or reference data. Tables, views, stored procedures, and functions are database objects. (GeeksForGeeks.)

In addition to database tables, BI solutions use some objects as data sources. The objects are:

- View: The view is a virtual table, which does not hold actual data but collects data from different physical tables (Microsoft a).
- Procedure: A stored procedure is a previously written transact SQL statement (T-SQL) that is saved into the server and can be reused at any time by executing it (W3Schools).

5 SQL Server Services and Tools

5.1 Management Studio

SQL Server Management Studio hereafter referred to as SSMS, is a user Interface desktop application provided by Microsoft as part of the SQL server to manage, configure and administer all components of an SQL server. Using SSMS, one can create databases and tables, execute SQL queries, and manage privileges on databases. In addition, it includes tools for deploying and monitoring databases. (JavaTPoint a.) Through SSMS, a user can connect to different types of servers. Figure 6 shows what server types the SSMS interface supports.

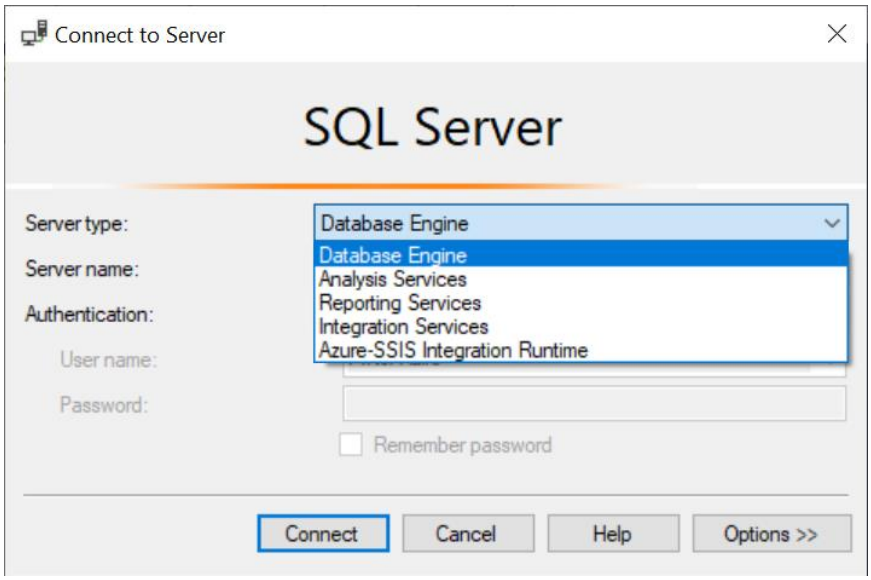

Figure 6. SSMS front page to connect to an SQL server

5.2 Integration Services

SQL server provides an integration services platform, hereafter referred to as SSIS, to implement ETL solutions in data warehousing. The tool includes various graphical tools and window wizards to create different components in an SSIS package (JavaTPoint b.) SSIS projects are created using the SQL server integration services extension in Visual Studio (Figure 7).

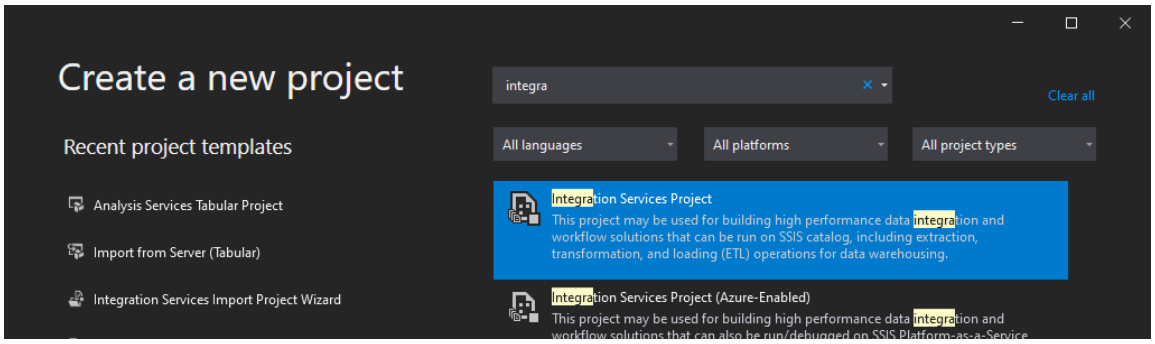

Figure 7. Creating a service integration project in Visual Studio

5.2.1 SSIS Packages

A package is the central part of an integration service project where the data integration and transformations are performed (JavaTPoint b). It assembles connections, control flow elements, dataflow components, parameters, variables, and event handlers (Microsoft b).

Each package consists of three main parts

- Control flow
- Dataflow
- Connection managers.

The Control flow mainly manages workflows. It includes various kinds of executables connected with constraints. Executables are either containers or tasks. A Container can include different tasks, which run in sequence or loops. Constraints determine the sequence to execute executables. Figure 8 shows an example of the control flow view for a created SSIS package.

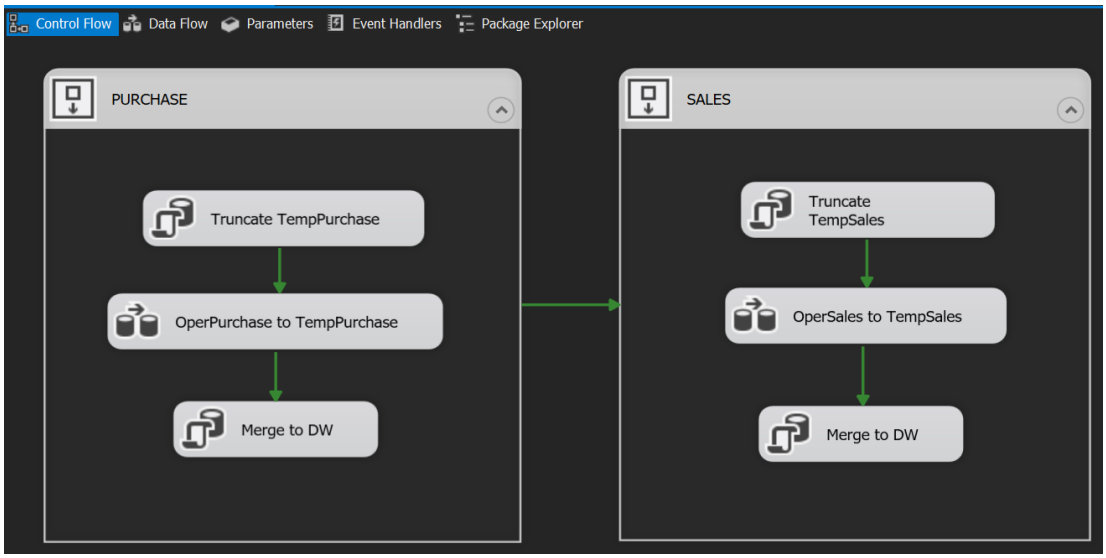

Figure 8.The control flow view implemented in a Visual Studio project

The Dataflow is a task that consists of source, destination, and transform objects (Microsoft b). It handles the ETL process, which performs the operations of extract, transform, and load. Developers use source objects for extract operations and destination objects for load operations. Figure 9 shows an example view of a dataflow task.

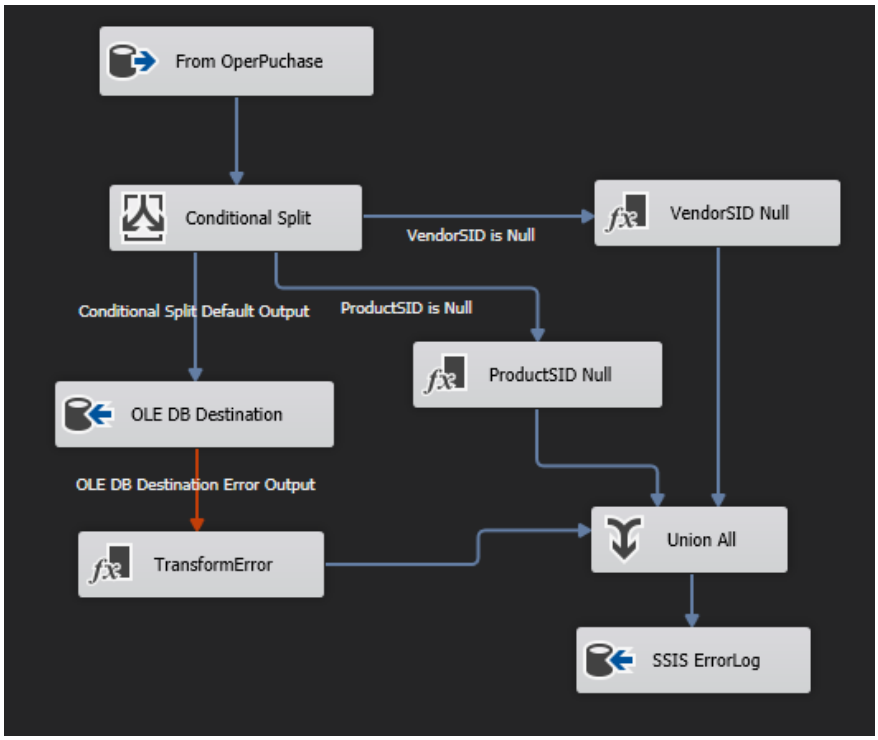

Figure 9. Dataflow view in a Visual Studio project

The connection managers are used to connect packages to data sources. Several types of connections are provided, including excel files, relational databases, analysis services databases, and more. (Microsoft b.) Connections can be created on the project level so multiple packages can use them under the same project or locally in the package, which is used only by that package.

5.3 Analysis Services

SQL Server analysis services hereafter referred to as SSAS, is a tool provided by Microsoft. It is installed as a server instance part of SQL server installations. An SSAS project is created in Visual Studio using the analysis services extension. Figure 10 shows a diagram for the tables included in a tabular model with their relations. The model is created in Visual Studio, processed to fetch data from data sources, and then deployed as a database into the analysis service instance (Figure 11). With proper permissions, deployed models are accessible to different visualization clients. (Microsoft c.)

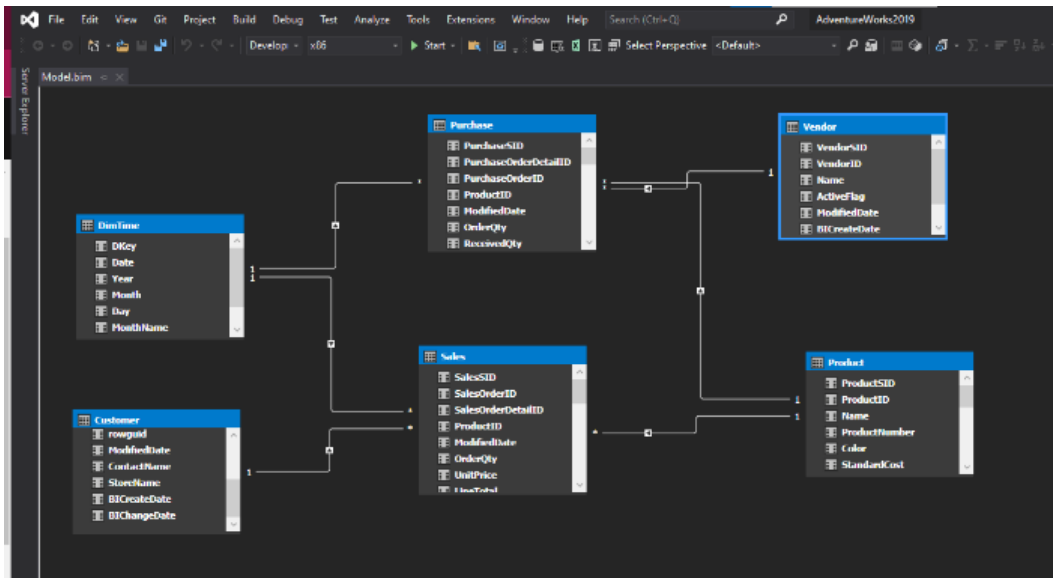

Figure 10. Tabular project implemented in Visual Studio

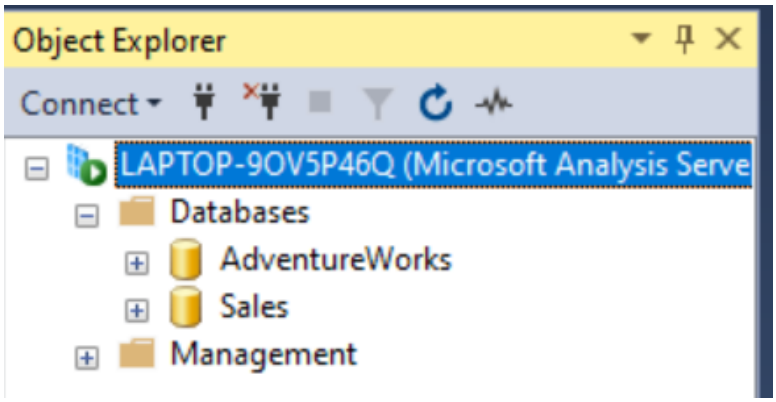

Figure 11. Accessing SQL analysis service server instance on localhost from SSMS

5.4 Reporting Services

SQL server reporting services, hereafter referred to as SSRS, is a tool Microsoft provides to create paginated formatted reports that include graphs, charts, tables, KPIs and others (JavaTPoint c). An SSRS project is created using the Microsoft reporting services extension in Visual Studio (Figure 12).

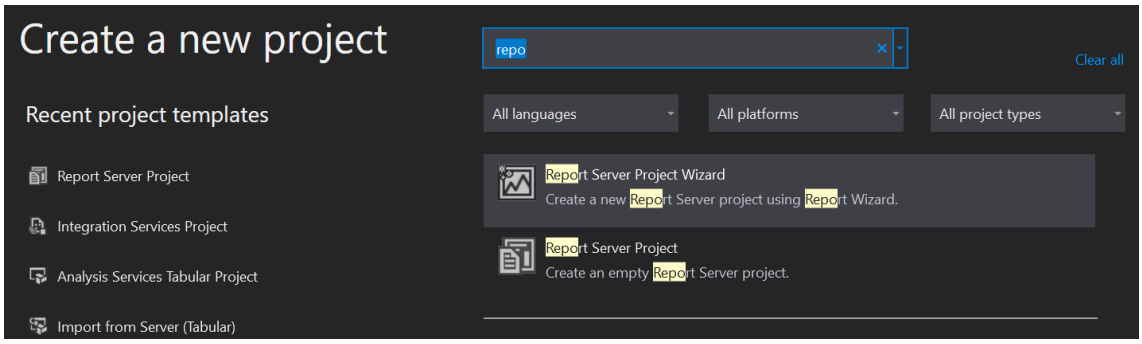

Figure 12. Creating a report server project in Visual Studio

The main component of an SSRS project is the report. Each report includes different elements, such as data sources, datasets, parameters, and others. Figure 13 is an example report view in Visual Studio.

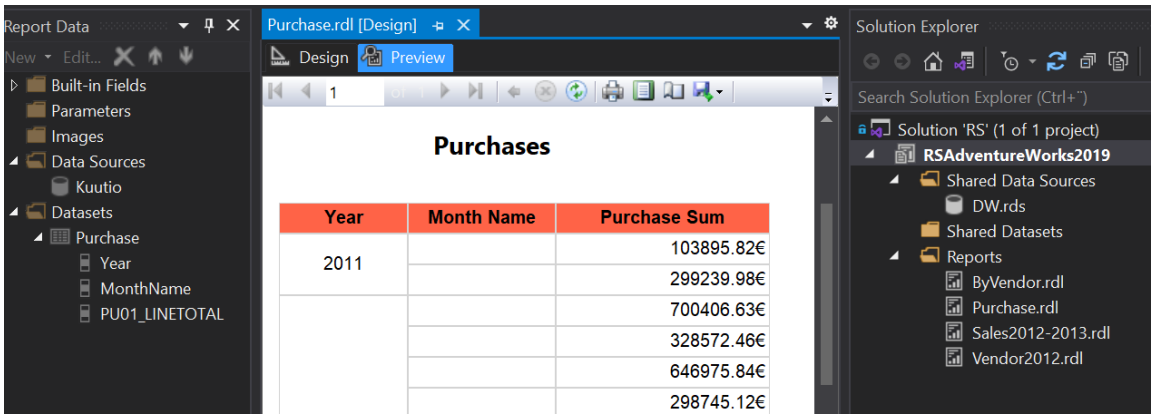

Figure 13. Example report view in Visual Studio

A dataset is the data returned by a query against a data source. The data source within a report is either a reference to a shared data source configured on the project level or a report-specific data source that establishes a connection that is used by this report only. (Microsoft d.)

Figure 14 shows the window to configure a data source for a report.

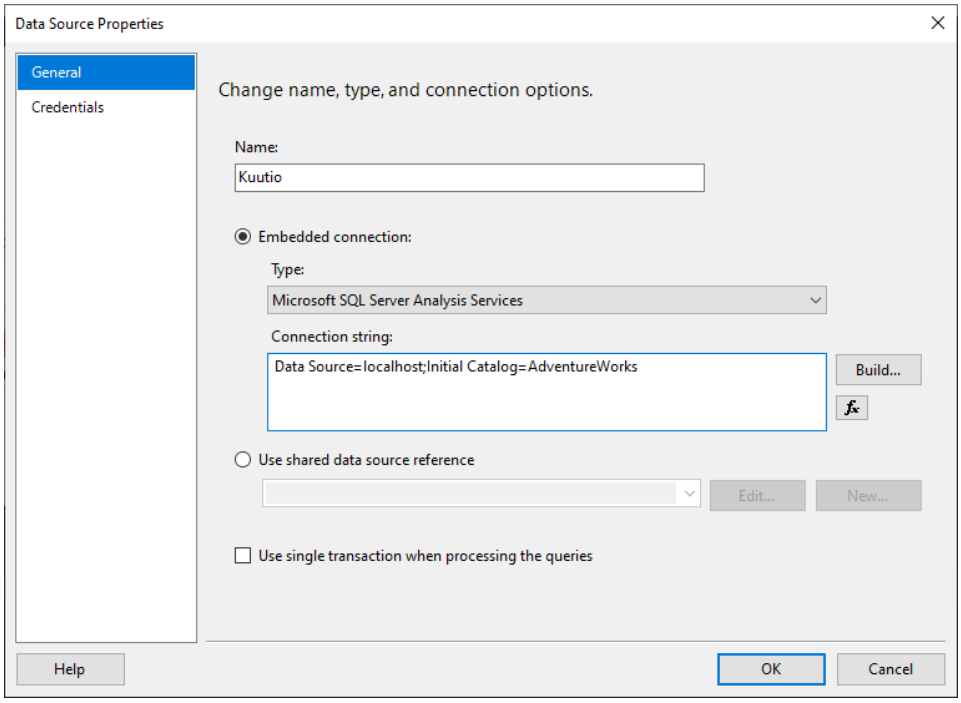

Figure 14. Configuring a data source within a report

5.5 SQL Server Agent

The SQL server agent is a window service for managing and running jobs. A job is a set of steps where each step performs one task. Jobs can run manually or be scheduled to run automatically or based on the occurrence of an event. (Microsoft e.) Steps can be of different types. The type named SQL Server Integration Services Package is responsible for executing the SSIS packages. The SQL server agent can be accessed from SSMS. Figure 15 shows the job properties window from which one can access a configured job and add or modify steps using SSMS.

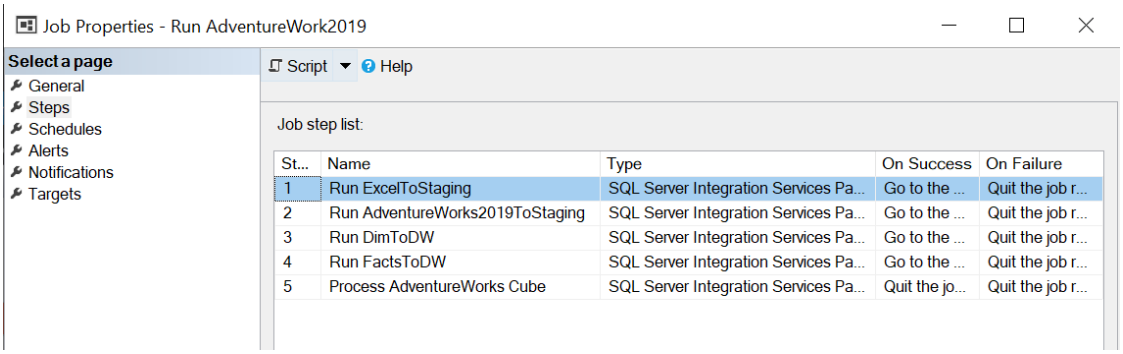

Figure 15. SQL server agent job properties in SSMS

6 Other Microsoft Tools

6.1 PowerBI

PowerBI is a collection of applications Microsoft provides to collect data from data sources to provide comprehensive visuals eventually. Using the PowerBI desktop application, data from various sources are collected and transformed to create a data model, which will be used to create visuals in the report. The report can be shared with others in the organization using the PowerBI service. The model developed in PowerBI desktop is an analysis service tabular model, often referred to as a dataset. Datasets on the PowerBI service either connect to external analysis service models, such as the SQL analysis service model (SSAS) and the Azure analysis service model. Alternatively, they are internally hosted in the Power–BI service by uploading them. (Microsoft f; Microsoft g.) A model developed on the PowerBI desktop is typically uploaded to the service. An example of a report published in PowerBI Service is shown in Figure 16.

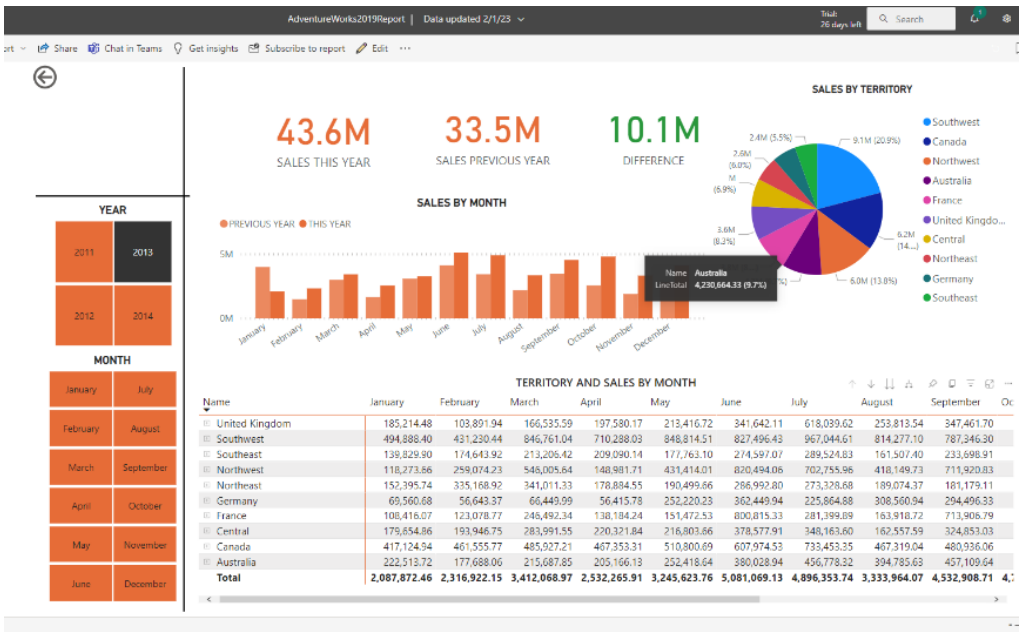

Figure 16. Example of PowerBI report published to PowerBI service portal

6.2 Azure

Azure is a Microsoft cloud platform that provides plenty of services through its portal, including computing, analytics, storage, and networking (Bigelow). Some of the services Azure provides are

- Azure Active Directory: a cloud-based directory and identity management service which allows the employees of an organization to access Azure services.
- Azure Data Factory: a cloud ETL service for data integration and transformation provided in the Azure portal.

7 BI Documentation Tool

7.1 Background: A Briefing of Current State Application

The development of a console application was initiated around a year ago. A .Net application was developed using the C# language. The application includes SSIS, SSRS, SSMS, and SQL parsers to parse SSIS packages, and SSRS reports, collect stored views and procedures from the SQL server database engine and parse SQL statements. The application also downloads the BI project files from the Azure DevOps Git server. Running the application will initiate each parser asynchronously.

An executable application is a result of publishing the project through Visual Studio. Upon publishing the project, a folder for the published project is created, which includes an executable file with extension exe, a settings file of type JSON (JavaScript Object Notation), and dependencies required to run the application (Figure 17).

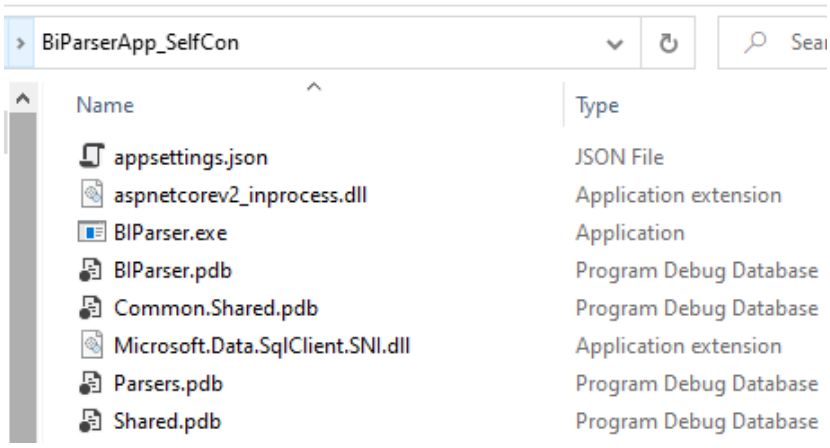

Figure 17. The contents of the published application

The appsettings.json file is a JSON file accessible by the user and includes the configurations required to run the application (Figure 18). The user should provide the values for each property before running the application the first time. The application will parse files located in the directory specified in DevOpsfileRoot property in the settings file. The connection string is used to establish a connection to the destination database where the data collected by the application gets inserted. The Servers property is a list of servers accessed by the SSMS parser.

```
{
  "Servers": [
    "localhost"
  ],
  "AppDbConfig": {
    "ConnectionString": "Data Source=localhost;Initial Catalog=OldParser;Trusted_Connection=True",
    "SsisMetadataTableName": "SsisMetadata",
    "SchemaName": "dbo",
    "SqlMetadataTableName": "ServerMetadata"
  },
  "DevOpsfileRoot": "C:\\Users\\RoulaAlmouayadAlazem\\Test2",
  "DevOpsPersonalAccessToken": "",
  "DevOpsApiURL": ""
}
```

Figure 18. appsettings.json file structure for initial implementation

The BIParser.exe is executed using the command line application to run the application. The user should provide couple of parameters which will be passed to the application during the execution (Figure 19). The parameters are:

- The operation to perform, which represents which parser will run
- The path to the settings file, which refers to the location where the application is physically located.

```
_SelfCon>BIParser.exe run "C:\Users\RoulaAl _SelfCon\appsettings.json"
```

Figure 19. Running the BIParser.exe in the command line

The XML deserialization is used in SSIS and SSRS parsers to convert XML files to an XML-typed object, using the System.Xml.Serialization library provided by Microsoft. Figure 20 is an example of a code for simple XML file deserialization. Figure 21 shows a simple XML file content, and Figure 22 shows the result of running the code.

```
XmlSerializer xmlSerializer = new XmlSerializer(typeof(Employee));
using StreamReader reader = new StreamReader(@"C:\Users\R \Employee.xml");

Employee employee= (Employee) xmlSerializer.Deserialize(reader);

Console.WriteLine($"Name: {employee.Name}, Age: {employee.Age}, Department: {employee.Department}");

public class Employee
{
    public string Name { get; set; }
    public string Age { get; set; }
    public string Department { get; set; }
}
```

Figure 20. Sample code for XML deserialization

```
<?xml version="1.0" encoding="utf-8" ?>
<Employee>
  <Name>JohnS</Name>
  <Age>30</Age>
  <Department>IT</Department>
</Employee>
```

Figure 21. XML file contents

```
Microsoft Visual Studio Debug Console
Name: JohnS, Age: 30, Department: IT
```

Figure 22. Object details printed into the console after running the code

In the implementation of the projects, different libraries are used. These libraries are .Net packages found and installed using the NuGet manager. To have the application access and interact with the SQL server database engine, the Microsoft.Data.SqlClient library was used. The library includes various classes that are instantiated to establish a connection with the database, read data from the database, and insert data into the database.

Upon the initiation of each parser, a database table gets created into the database specified previously in the ConnectionString property in the settings file. Each parser will have a dedicated table to include the data collected by that parser (Figure 23). The data collected are represented in a collection of objects. Each object translates into a table row, and each object property represents a table column.

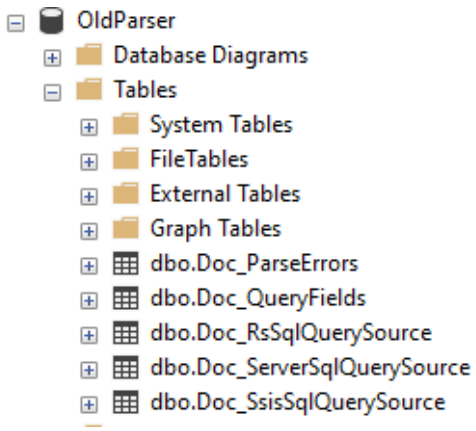

Figure 23. Tables created in the database after running the application

7.1.1 SSIS Parser

An SSIS project is a directory which includes at least one package file and might include connection manager files if a connection manager was configured on the project level. Else, at least one local connection manager is configured on the package level. Figure 24 shows a project directory.

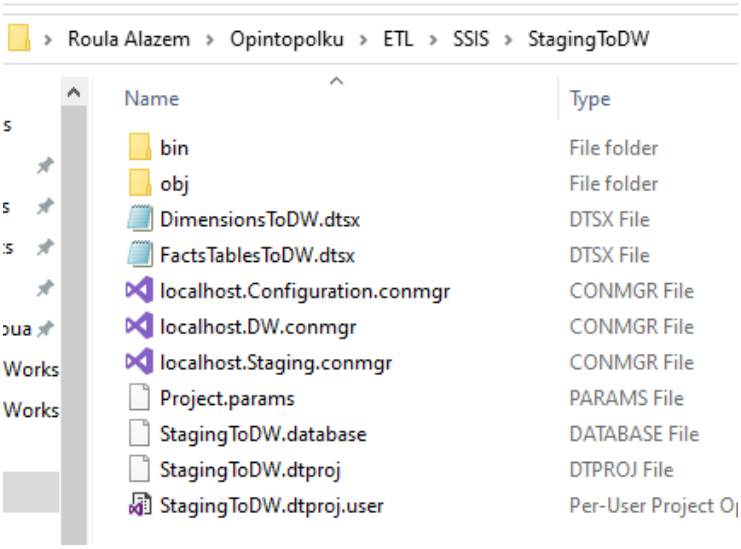

Figure 24. Contents of an SSIS project directory

The package file has the extension dtsx, and the connection manager file has the extension conmgr. Dtsx and conmgr files are XML files, thus, XML deserialization is used during the parsing.

SSIS parser identifies SSIS packages and connection manager files using their extensions, parses each file, and collects the package and connection metadata. The application parses only two types of executables, the **data flow tasks** and their relative components and the **execute SQL tasks**.

When initiating the SSIS parser, the parser loops through each file identified earlier, deserializes the file, and then executes a series of methods to collect the metadata of a package and map it to the connection managers it uses. The result is a list of objects; each represents one component/executable in the SSIS package. Objects are then inserted into the database table. Figure 25 shows the result of an SQL query done in SSMS from the table, which resulted from the SSIS parser.

esults | Messages

Projec...	PackageName	FileName	FullPathToFile	Prefix	SourceType	Server	Database	Query	PackageReference...	SqlCo...	SourceID
ETL	AdventureWork...	Adventure...	ETL\SSIS\OperToSt...	Adven...	SSIS SOURCE	localhost	AdventureWorks2019	SELECT * ...	Package\PRODUC...	SELE...	417E4B8
ETL	AdventureWork...	Adventure...	ETL\SSIS\OperToSt...	Adven...	SSIS SOURCE	localhost	AdventureWorks2019	SELECT * ...	Package\PRODUC...	SELE...	3B4C62E
ETL	AdventureWork...	Adventure...	ETL\SSIS\OperToSt...	Adven...	SSIS SOURCE	localhost	AdventureWorks2019	SELECT * ...	Package\PURCHA...	SELE...	55E13927
ETL	AdventureWork...	Adventure...	ETL\SSIS\OperToSt...	Adven...	SSIS SOURCE	localhost	AdventureWorks2019	SELECT * ...	Package\PURCHA...	SELE...	9F59E86
ETL	AdventureWork...	Adventure...	ETL\SSIS\OperToSt...	Adven...	SSIS SOURCE	localhost	AdventureWorks2019	SELECT * ...	Package\SALESO...	SELE...	51278BF
ETL	AdventureWork...	Adventure...	ETL\SSIS\OperToSt...	Adven...	SSIS SOURCE	localhost	AdventureWorks2019	SELECT * ...	Package\SALESO...	SELE...	0BC2E39
ETL	AdventureWork...	Adventure...	ETL\SSIS\OperToSt...	Adven...	SSIS SOURCE	localhost	AdventureWorks2019	SELECT * ...	Package\VENDOR...	SELE...	FDB52CB
ETL	AdventureWork...	Adventure...	ETL\SSIS\OperToSt...	Adven...	SSIS TARGET	localhost	Staging	SELECT * ...	Package\CUSTOM...	INSERT	BCF7536
ETL	AdventureWork...	Adventure...	ETL\SSIS\OperToSt...	Adven...	SSIS TARGET	localhost	Configuration	SELECT * ...	Package\CUSTOM...	INSERT	76F5D43

Figure 25. Result of an SQL query in SSMS from the table created by the SSIS parser

7.1.2 SSRS Parser

The SSRS project includes at least one report file and might include data source files if shared data sources are configured on the project level, as Figure 26 shows. If shared data sources are not used in the project, each report should include at least one local data source saved inside the report file.

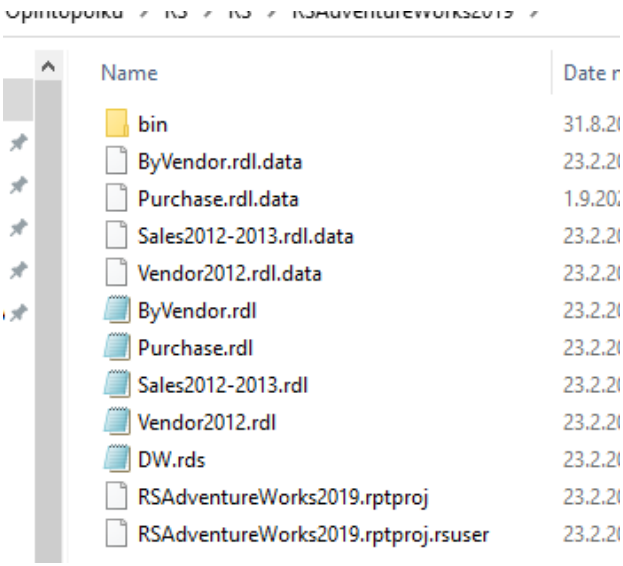

Figure 26. Contents of an SSRS project

Report files have the extension rdl, while the shared data source files have the extension rds. Both types are XML formatted files. The SSRS Parser identifies report files using their extension and deserializes them into XML-typed objects. Then, it executes different methods to collect the metadata of the datasets and collect them into a list of objects, where each object represents a dataset. These objects are then inserted into the database. Figure

27 shows the results of an SQL query in SSMS from the table created by running the SSRS parser.

esults ssages ssages

Project	PackageNa	PackageN	FileNam	FullPathToFile	Prefix	SourceType	Connectio	ConnectionString[	Query	isExpre	DatasetNa	SqlCom	SourceID
RS	ByVendor	ByVendor	ByVenc	RS\RSAdventure	ByVe	PROCEDUF		DW	Purchase	0	ByVendorl	SELEC	E59FBC
RS	ByVendor	ByVendor	ByVenc	RS\RSAdventure	ByVe	SQL		DW	SELECT	0	VendorsLi	SELEC	BD1A53
RS	Purchase	Purchase	Purcha	RS\RSAdventure	Purcl	MDX		Kuutio	EVALUA	0	Purchase	SELEC	9D3527
RS	Sales2012-	Sales201	Sales2	RS\RSAdventure	Sales	SQL		Sum	SELECT	0	Sum	SELEC	C4F109
RS	Vendor201	Vendor20	Vendor	RS\RSAdventure	Venc	SQL		ProcedureVendor	SELECT	0	VendorsP	SELEC	FD51F6
RS	ByVendor	ByVendor	ByVenc	RS\RSAdventure	ByVe	PROCEDUF		DW	Purchase	0	ByVendorl	SELEC	48465C
RS	ByVendor	ByVendor	ByVenc	RS\RSAdventure	ByVe	SQL		DW	SELECT	0	VendorsLi	SELEC	7603824
RS	Purchase	Purchase	Purcha	RS\RSAdventure	Purcl	MDX		Kuutio	EVALUA	0	Purchase	SELEC	FD836D
RS	Sales2012-	Sales201	Sales2	RS\RSAdventure	Sales	SQL		Sum	SELECT	0	Sum	SELEC	9515479

Figure 27. Result of an SQL query in SSMS from the table created by the SSRS parser

7.1.3 SSMS Parser

The SSMS parser executes chain of methods to loop through all SQL servers specified in the settings file and fetch all view and procedure objects from the database engine. Required parsed data are collected into objects inserted into the targeted database table. Figure 28 shows the resulting table from running the SSMS parser.

Results Messages

	SourceType	SourceSer...	SourceSche...	SourceData...	SourceName	Query	SourceID
1	VIEW	localhost	Temp	Staging	Read_Purchase	CREATE VIEW [Temp].[R...	98C38827-B4E4-...
2	VIEW	localhost	SSIS	Configuration	PBI_ExecutionLog	Create View [SSIS].[PBI_E...	ED1E3A12-6F7B-...
3	VIEW	localhost	SSIS	Configuration	PBI_ErrorLog	Create View [SSIS].[PBI_E...	4F5EDCB7-4838-...
4	VIEW	localhost	SSIS	Configuration	PBI_ExecutionPhases	Create View [SSIS].[PBI_E...	4220212F-2169-4...
5	VIEW	localhost	SSIS	Configuration	PBI_JobProperties	Create View [SSIS].[PBI_Jo...	C99ED8B2-0ABF-...
6	VIEW	localhost	SSIS	Configuration	PBI_Packages	CREATE view [SSIS].[PBI_...	530C0402-709F-4...
7	PROCEDURE	localhost	dbo	DW	AllVendorsPurchases2012	CREATE PROCEDURE All...	F8E223F9-3727-4...
8	PROCEDURE	localhost	dbo	DW	PurchasesByVendor	CREATE PROCEDURE [db...	5B39B8C8-2840-...

Figure 28. SQL query in SSMS from the table created by the SSMS parser

7.1.4 SQL Parser

The SQL parser parses each SQL statement collected by other parsers. The column Query in each table, resulting from SSIS, SSRS and SSMS parsers, represents these statements. The purpose of parsing the query field is to collect which column in which table from which database is used in each statement. The list of objects is inserted into the database. The

SQL parser runs with each of the SSIS, SSRS and SSMS parsers while they are running. The SQL parser uses the Microsoft.SqlServer.Management.SqlParser.SqlCodeDom library to identify the existence of main SQL clauses in forming a valid SQL statement. It identifies the existence of a select, from and others clause. If an error is identified in the SQL statement, the error gets inserted into a separate table created when the SQL parser initializes. Figure 29 shows the table for data collected by the SQL parser, while Figure 30 shows the errors table resulting from failures to parse SQL statements.

esults | Messages

Server	Database	Schema	Table	Column	Context	expression	UsageConte:	hashCo	ParentID
	DW	dbo	Vendor	Name	READ	Name	SELECT	14265:	1F948CEC
	ProcedureVen	dbo	DimTime	VendorNar	READ	VendorName	SELECT	-16838	9D461FA8
	ProcedureVen	dbo	DimTime	LineTotal	READ	SUM(LineTot	SELECT	12424:	9D461FA8
	ProcedureVen	dbo	Purchase	OrderDate	READ	P.OrderDate	FROM	11629:	9D461FA8
	ProcedureVen	dbo	DimTime	DKey	READ	P.OrderDate	FROM	24187:	9D461FA8
	ProcedureVen	dbo	DimTime	Year	READ	dt.Year = 201	WHERE	-16190	9D461FA8
	DW	dbo	Vendor	Name	READ	Name	SELECT	14265:	7A90B500
	ProcedureVen	dbo	DimTime	VendorNar	READ	VendorName	SELECT	-16838	D5FEED1E
	ProcedureVen	dbo	DimTime	LineTotal	READ	SUM(LineTot	SELECT	12424:	D5FEED1E
	ProcedureVen	dbo	Purchase	OrderDate	READ	P.OrderDate	FROM	11629:	D5FEED1E
	ProcedureVen	dbo	DimTime	DKey	READ	P.OrderDate	FROM	24187:	D5FEED1E
	ProcedureVen	dbo	DimTime	Year	READ	dt.Year = 201	WHERE	-16190	D5FEED1E

Figure 29. SQL query in SSMS from the table resulted from the SQL parser

Results | Messages

	ErrorSource	ErrorLocation	ErrorQuery	ErrorMessage	StackTrace
1	SSRS	RS\RS\RSAdventureWorks2019\B...	PurchasesByV...	NO QUERY...	at Commo...
2	SSRS	RS\RS\RSAdventureWorks2019\V...	EXEC AllVend...	NO QUERY...	at Commo...

Figure 30. SQL error table resulted from invalid SQL statements

7.2 The Implementation

Part of the implementation of this project is to have the application runnable on an end-user environment. One customer of Pinja was chosen as the pilot user. The implementation included parsing tabular and PowerBI data models and collecting data related to jobs running in the SQL server. Also, to ensure that already implemented features are functioning correctly. In addition, it was intended to make the application dynamic so that any customer can use it. A unified approach was followed in the new implementations. Tables for the

collected data from parsed tabular and PowerBI models and jobs are created at the beginning of each process. In the implementation, each parser identifies the existence of any compressed file under the root directory using the file extension zip. If any file with the extension zip is found, the ZipArchive class provided by Microsoft is used to read the compressed file contents. Contents are then parsed using the parsing process.

7.2.1 SSAS Parser

The SSAS or tabular parser is the mechanism to collect metadata from tabular model files into a dedicated database table. The tabular model project includes a model file identified with the extension bim and the tabular project file with extension smproj, as shown in Figure 31. The model file bim includes tabular model metadata, such as tables, measures, partitions, data sources, and more. The project files include details related to the deployment of the tabular model.

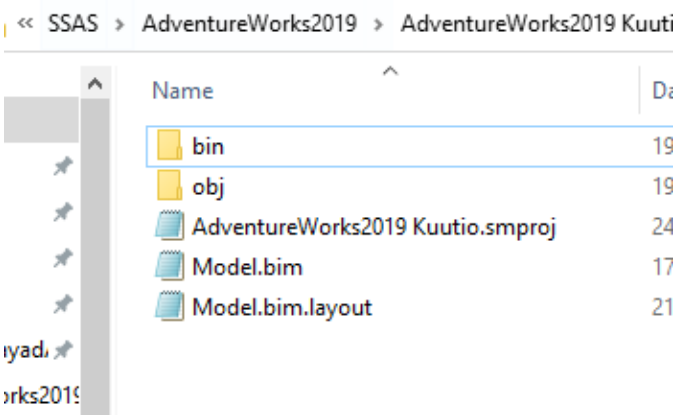

Figure 31. Tabular project contents

The parser identifies the files using their extensions. The bim model file is a JSON formatted file; hence file is deserialized into a JSON-typed object using the Newtonsoft.Json library. Then the yielded JSON object is parsed through a chain of methods to collect the required metadata. Since the tabular name in the analysis service server is determined during the deployment process, the project file gets parsed. The project file smproj is an XML file and gets deserialized using the XML deserializer. The collected metadata results in a list of objects, each representing a table in the model. Objects are then inserted into the database table. The table is created once the parser initializes. Figure 32 shows the results of successfully running the SSAS parser. The SQL parser methods also execute during the parsing process to parse any SQL statement found.

Results | Messages

	Project...	FileName	FullPathToFile	TabularName	Connection...	Con...	QueryType	Query	Sch...	Table	SourceID
1	Kuutio	Model	Kuutio\SSAS\Ad...	AdventureW...	localhost	DW	query	SELECT [dbo].[...	dbo	Customer	46042387
2	Kuutio	Model	Kuutio\SSAS\Ad...	AdventureW...	localhost	DW	query	SELECT [dbo].[...	dbo	DimTime	97D6F66
3	Kuutio	Model	Kuutio\SSAS\Ad...	AdventureW...	localhost	DW	query	SELECT [dbo].[...	dbo	Product	D7D503F
4	Kuutio	Model	Kuutio\SSAS\Ad...	AdventureW...	localhost	DW	query	SELECT [dbo].[...	dbo	Purchase	21832D2
5	Kuutio	Model	Kuutio\SSAS\Ad...	AdventureW...	localhost	DW	query	SELECT [dbo].[...	dbo	Sales	192B510
6	Kuutio	Model	Kuutio\SSAS\Ad...	AdventureW...	localhost	DW	query	SELECT [dbo].[...	dbo	Vendor	83340769

Figure 32. SQL query in SSMS against the table created by the SSAS parser

7.2.2 PowerBI Parser (PbiParser)

In PbiParser implementation, it was initially intended to use the same approach to identify PowerBI files with the extension pbix. However, after investigating and exploring the contents of the pbix file, it was found that the pbix file is a compressed file, which includes multiple contents. To get the required data, the file named DataModel is examined. The DataModel file is a binary, compressed file, and reading the data was hard to achieve. Therefore, PowerBI datasets are collected from the PowerBI service. Microsoft provides two approaches to accessing datasets from the PowerBI service, The PowerBI REST APIs (Application Programming Interface) and the Tabular Object Model library.

The tabular object model (TOM) can programmatically access and manage models at compatibility level 1200 or higher. The library accesses the model metadata, such as tables and columns. The object in this model is a hierarchy, with the server as a root object, which contains a collection of databases. Each database includes different groups of objects, such as tables, data sources and others. At a lower level, each object also contains a collection of objects. Figure 33 shows the hierarchy of a tabular object model object. The server is represented by the workspace in the PowerBI service, while the database is the dataset. (Microsoft i; Microsoft j.)

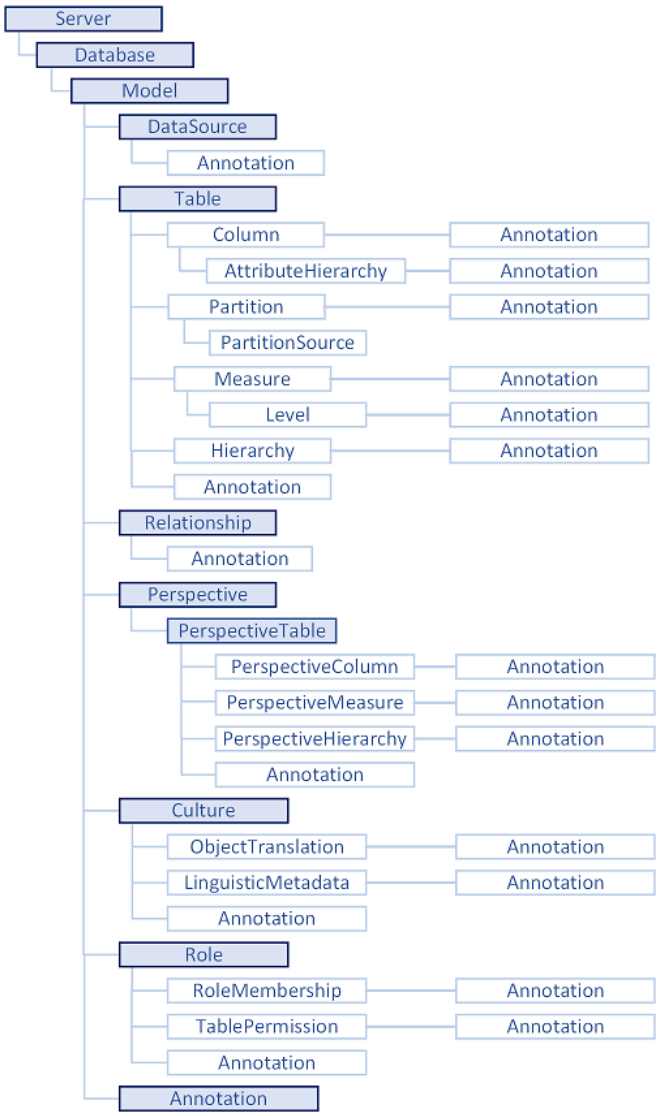

Figure 33. Tabular Object Model hierarchy (Microsoft j)

PowerBI REST APIs are endpoints to embed report contents in web or desktop applications (Microsoft k). Microsoft offers two solutions for embedding PowerBI reports into an application. The first one is the embed-for-your-organization, known as user-own-data, which requires a licensed PowerBI user interactive login. The second is the embed-for-your-customers, also known as app-own-data, which uses a non-interactive method to authenticate the application into PowerBI service. (Microsoft i.)

After going through the functionalities of both libraries Microsoft provides, it was found that the REST API does not offer functionalities to access the model metadata, such as tables, columns, and so on. On the other hand, the Tabular Object Model (TOM) library does not provide information on the datasets which use a live connection to connect to analysis

service tabular models as their data source. Figure 34 shows the contents of the workspace named Test in the PowerBI portal. It is known that AdevntureWorks2019CubeReport datasets use an SSAS tabular model as a data source. Two sample codes were executed to retrieve data from the workspace named Test. Figure 35 shows the results of running sample code that used the tabular object model, while Figure 36 shows the results of running the code that used the PowerBI REST API. The code using the tabular model object did not return the datasets that use a tabular model as a data source. Based on that, a complete solution that loops over all workspaces and datasets was built using the PowerBI REST API and the Tabular Object Model.

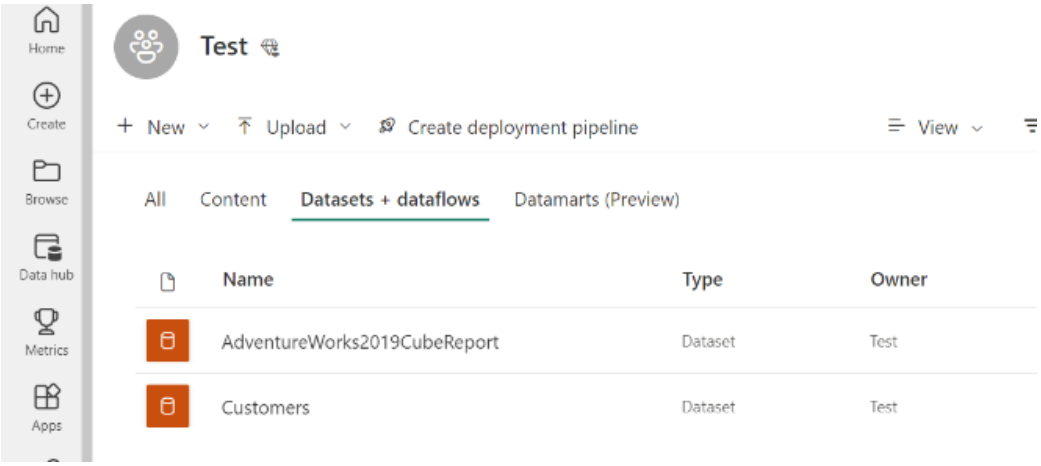

Figure 34. Example workspace contents in the PowerBI service portal

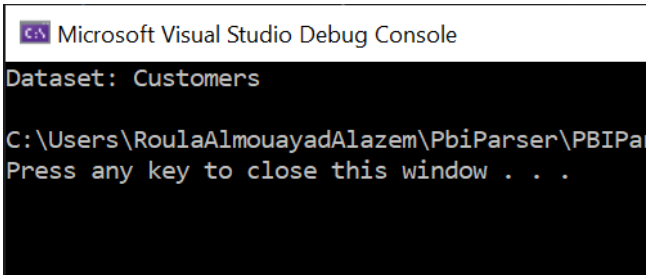

Figure 35. Running a sample code using the tabular object model library

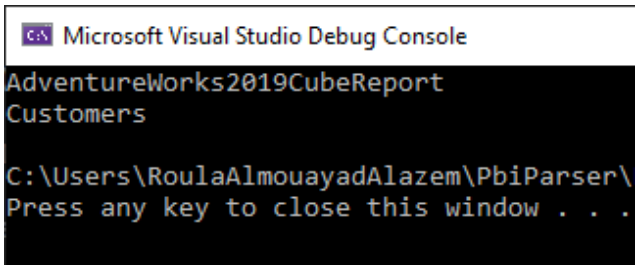

Figure 36. Running a sample code using PowerBI API

In the implementation related to the REST API, the embed-for-your-customer solution using the service principal authentication method was used to avoid the need for interactive login to PowerBI service. According to Microsoft instructions for the API to access the PowerBI services, several steps needed to occur. First, the application was registered in the Azure active directory. For this purpose and to test the functionalities of the implemented PbiParser, a personal free trial subscription to Microsoft and Azure portal was used. Figure 37 shows the registered BiApp application in Azure AD.

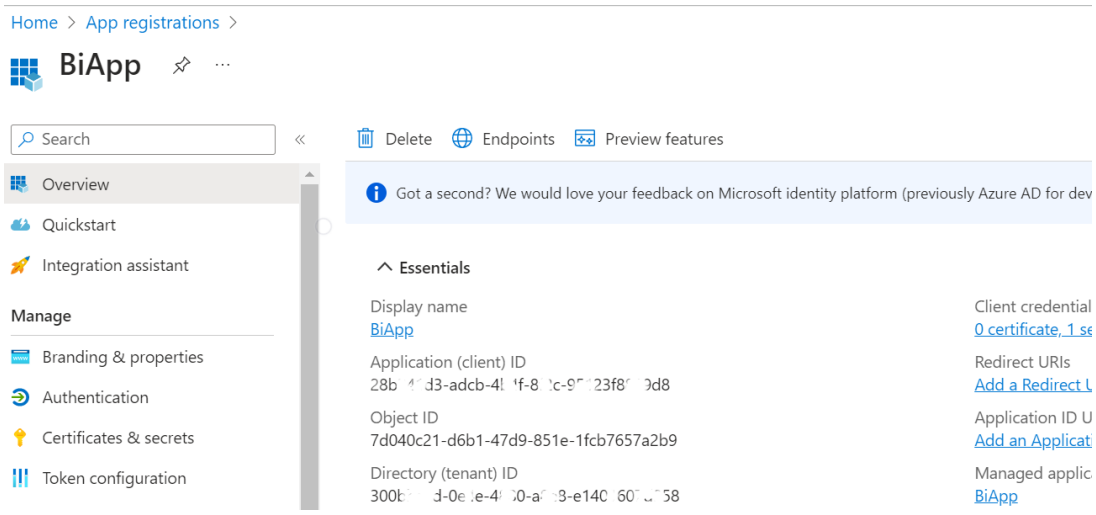

Figure 37. Registered application in Azure active directory

After registering the application, a new client secret was added under the certification and secrets. The string under the value field is the client secret that will be used later in the parser. It should be copied immediately and kept in a safe place, as it will not be displayed again once the page is refreshed, as shown in Figure 38.

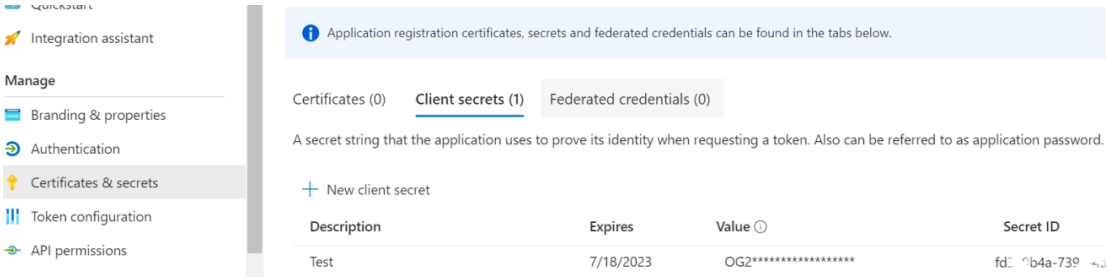

Figure 38. A client secret added to the registered application

Read or write permissions should be specified and granted to the registered application from the Azure portal. Read permissions to PowerBI services were given from the API permission option (Figure 39).

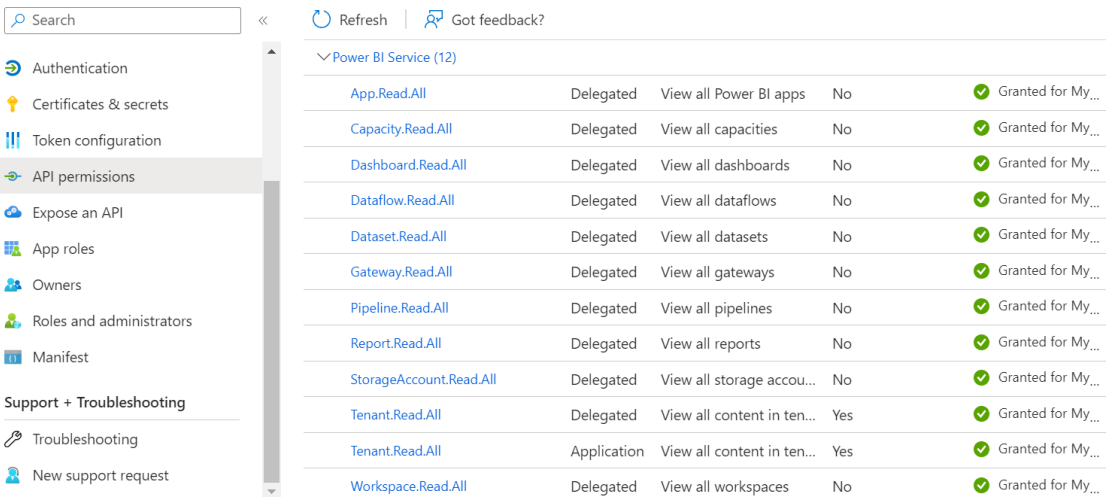

Figure 39. PowerBI read permission was granted to the application

On the PowerBI portal side, two options were enabled by an admin of the PowerBI portal. The embed content in apps option and allow service principals to use the PowerBI APIs option (Figure 44).

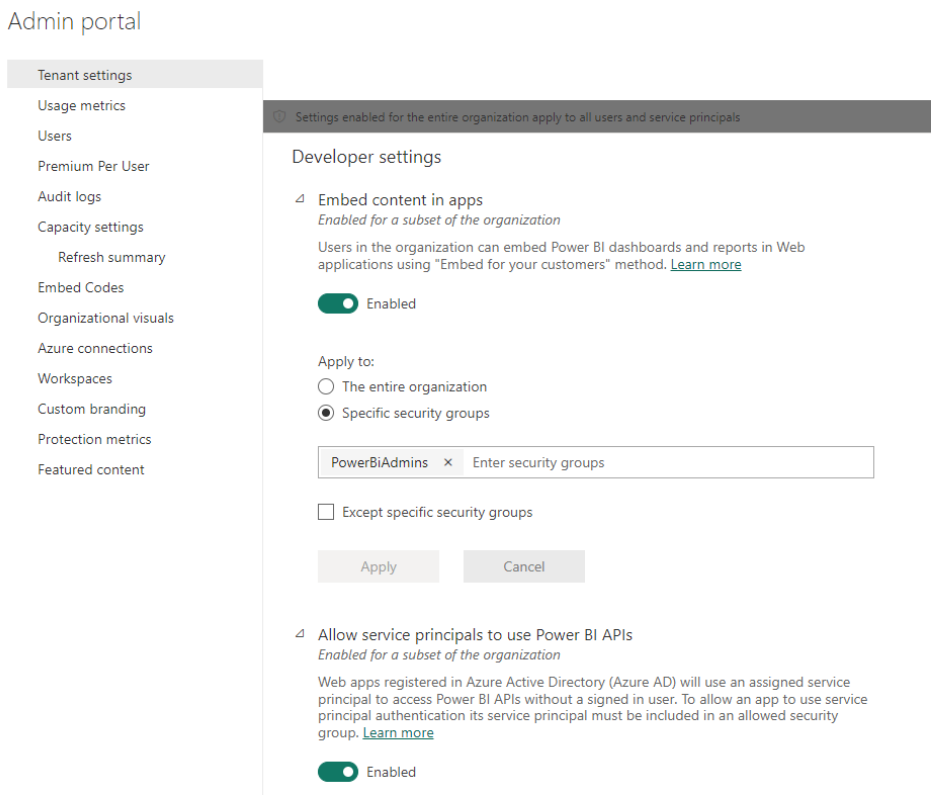

Figure 40. Enable permission in the PowerBI admin portal

Also the application as a service principle should be added to the access the workspaces. Figure 41 shows the BiApp service principle added to access the required workspaces.

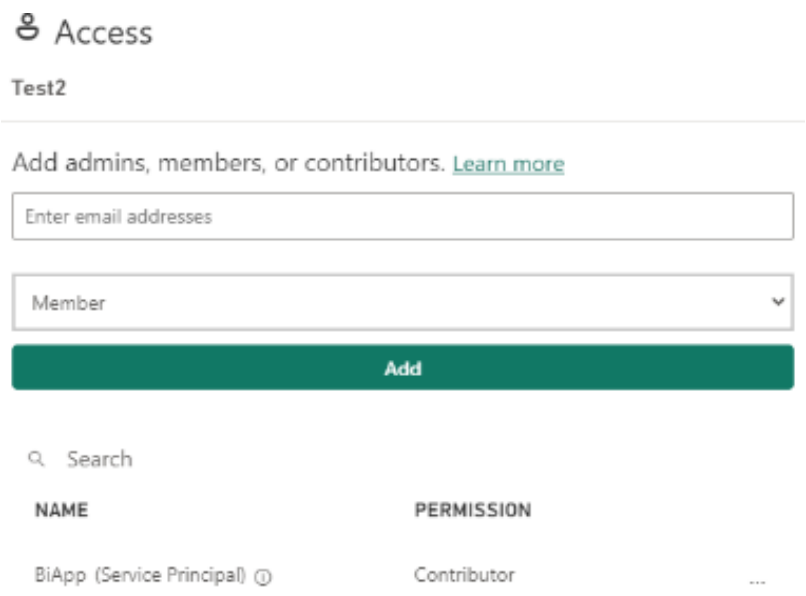

Figure 41. Allowing the application to access the workspaces

Some of the application details are required to authenticate the application. These details are called the embedding parameters and are found in the Azure portal. The parameters were added to the appsettings.json file (Figure 42).

```
//The setting needed to have PowerBI parser work

"AzureAD": {
  "clientId": "28b[illegible]18",
  "clientSecret": "0G2[illegible]X1",
  "tenantId": "36[illegible]E3"
},
```

Figure 42. Embedding parameters in the application settings file

Microsoft provides the Microsoft.PowerBI.Api library to implement the functionalities of the REST API. The library includes the class PowerBIClient which contains various properties representing the endpoints of the API, each including several methods. This implementation focuses on Datasets and Groups endpoints. The Groups endpoint provides access to workspaces in the PowerBI service, while the Datasets endpoint provides access to the models in the report. Creating an instant of PowerBIClient object requires token credentials to authenticate the application to access the services (Figure 43). The Microsoft.Identity.Client and the Microsoft.Rest libraries were used to implement a token manager object with several methods which produce an access token.

```
string baseURI = "https://api.powerbi.com";
var tokenCredentials = new TokenCredentials(GetAccessToken(), "Bearer");
var client = new PowerBIClient(new Uri(baseURI), tokenCredentials);
```

Figure 43. Sample code for creating a PowerBIClient object

When using the tabular object model library, a connection should be established to a workspace (a server) to access the PowerBI services. Figure 44 is an example code for establishing a connection to the workspace named Test. This method requires an interactive login to the PowerBI service since the connection string did not include credentials. The login window appears, and a user is prompted to log in to the service.

```
using Microsoft.AnalysisServices.Tabular;

string workSpaceString = @"powerbi://api.powerbi.com/v1.0/myorg/Test";
string connectionString = $"datasource = {workSpaceString}";

Server server = new();
server.Connect(connectionString);
```

Figure 44. Sample code to connect to a server using the tabular object model library

However, to avoid user interference in logging into the service in our implementation, the connection established to workspaces used the authentication with a token method (Figure 45).

```
string connectionString = $"datasource = {workSpaceString};Password={token}";
```

Figure 45. Authenticate with a token in the connection string

When running the PbiParser, a database table that collects the parsed data gets created. Then a series of methods get executed to parse and collect the required metadata from PowerBI datasets. Data get then inserted into the target table. Figure 46 shows the database table, including the data collected by the PbiParser. The first row represents a PowerBI dataset which uses SQL server analysis service tabular model as a data source. The blank values in the table and expression columns are because the parser does not collect metadata of that model. Since the model is tabular, it gets parsed using the SSAS parser.

esults | Messages

ReportName	Workspac	DatasourceT	DataSourceC	DataSourceConne	Table	Expression		Query	SourceID
AdventureWorks2019CubeRep	Test	AnalysisServ	localhost	adventureworks					8A5C976E-0C7F-4
Customers	Test	Sql	localhost	dw	Query1	let	Source = Sql.Da	select * from	C13A052F-DA78-4
AdventureWorks2019Report	Test2	Sql	localhost	adventureworks20	Customers	let	Source = Sql.Da		F0EBDE15-CEAC-
AdventureWorks2019Report	Test2	Sql	localhost	adventureworks20	Sales SalesTerrit	let	Source = Sql.Da		61F7E66B-6CFD-4
AdventureWorks2019Report	Test2	Sql	localhost	adventureworks20	Sales Store	let	Source = Sql.Da		C822B2DB-246B-4
AdventureWorks2019Report	Test2	Sql	localhost	adventureworks20	Sales SalesOrde	let	Source = Sql.Da		14E77BEA-94EF-4
AdventureWorks2019Report	Test2	Sql	localhost	adventureworks20	Sales SalesOrde	let	Source = Sql.Da		67676484-A988-4
AdventureWorks2019Report	Test2	Sql	localhost	dw	DimTime	let	Source = Sql.Da		F3F7AB7F-D540-4

Figure 46. Entries of the table resulted from PbiParser

During the execution of the parser, the values collected into the expression properties get parsed too. The expression parsing methods identify and extract details related to SQL statements. These details refer to database tables used as data sources when building the report model. Data are inserted into a different database table as expressions are parsed, and details are collected. The table is also created once the PbiParser initializes. Figure 47 shows the database table, which includes the details of each parsed expression. There are six entries in the table which equals the number of entries in the PbiParser table. The expression parser looks for the value in the Query column. If the field is not empty, then the query is parsed using the SQL parser, else the expression parser is used to parse the expression field.

Results | Messages

	ServerName	DatabaseName	SchemaName	TableName	ParentID
1	localhost	AdventureWorks2019	Sales	Customer	F0EBDE15-CEAC-4CC4-9228-D6749F4637E6
2	localhost	AdventureWorks2019	Sales	SalesTerritory	61F7E66B-6CFD-4739-AC30-73A8188E4EAF
3	localhost	AdventureWorks2019	Sales	Store	C822B2DB-246B-4D87-A2A4-9EF2C6FA3675
4	localhost	AdventureWorks2019	Sales	SalesOrderDetail	14E77BEA-94EF-43C3-ADDC-5694997AFF1A
5	localhost	AdventureWorks2019	Sales	SalesOrderHeader	67676484-A988-42D3-A829-EC9E7A1AB84A
6	localhost	DW	dbo	DimTime	F3F7AB7F-D540-460D-B2D1-3A042CC07125

Figure 47. Table of parsed expressions related to parsed PowerBI reports

7.2.3 SQL Server Jobs Collector

Part of the application was to collect data related to running jobs in the SQL server agent, particularly jobs which process SSIS packages. The implementation was done using the Microsoft.Data.SqlClient library to connect to the msdb system database, which is the database used by the SQL server agent. A Connection to the target database is established using the SQL Connection class. While a T-SQL query statement is performed using an SQL command object. After the query is conducted against the target database, a stream reader reads a stream of rows retrieved from the database. The results are inserted into a database table for SSIS-related jobs (Figure 48).

job_id	Job_Nam	SSIS_Package_	SsisPackageName	ServerNar	last_run_dat	step_uid	step_id	step_name	enabl	date_creat	date_modif
6C948D5	Run Adv	AdventureWork	ExcelToStaging.dtsx		20221212	D39ED3CD-F	1	Run Excel	1	2022-08-2	2023-01-2
6C948D5	Run Adv	AdventureWork	AdventureWorks201		20221212	9ED83CBC-5f	2	Run Adve	1	2022-08-2	2023-01-2
6C948D5	Run Adv	AdventureWork	DimensionsToDW.dt		20221212	A8AD1313-81	3	Run DimT	1	2022-08-2	2023-01-2
6C948D5	Run Adv	AdventureWork	FactsTablesToDW.c		20221212	A3C5A74A-D	4	Run Facts	1	2022-08-2	2023-01-2
6C948D5	Run Adv	AdventureWork	ProcessCube Adven		20221212	DB051273-DF	5	Process A	1	2022-08-2	2023-01-2

Figure 48. Resulting table from SQL Server Jobs Collector

7.3 Changes/Enhancements to Existing Parsers

Some changes were applied to the SSIS, SSRS, SSMS and SQL parsers to ensure a uniform structure across all parsers and to fix some discovered problems upon running the parsers.

Each parser source code included its method for writing parsed data into the database. One class to write to database tables was created and used among all parsers to avoid duplication and unnecessary code lines. Modifications were done in the methods of each parser that were impacted by this change.

In the initial implementation of parsers, the user should specify whether the files that need to be parsed are included in a compressed directory. Then the user chooses which operation to run when executing the application. This approach assumes that the main directory does not include any compressed subdirectories. If the user input was set to true (compressed directory), any file not part of a compressed directory will not be parsed, and vice versa. The implementation of SSIS and SSRS parsers was changed to identify files with the extension zip. In this case, any compressed file found under the root directory will be handled using the ZipArchive class to access its contents, and then the contents get parsed through the same parsing process.

In the case of an exception happening during the execution of the application, the application throws the exception and terminates the execution. Exceptions are captured and inserted into a separate database table using the try and catch blocks in the source code, which gets the application to continue parsing other files.

The names of database tables resulting from previously implemented parsers were hardcoded. Modifications were done to read the table names from the appsettings.json file, increasing the dynamicity of the application.

The SSIS parsers implementation was updated to include the Service Analysis processing task to executables that the parser recognizes. Initially, the parser recognizes only two types of tasks, the execute SQL task and dataflow tasks. The analysis service processing task is used to process SSAS tabular models. Unrecognizing the service analysis tasks resulted in missing data related to numerous SSIS packages.

The SSRS parser did not parse data source files (rds files) from which connection strings related to each report existed. Parsing the rds files provided more details about the data source used in the report and helped to relate each report to the SQL server data source. The SSRS parser was updated to identify rds files, deserialize them to XML schema objects,

then map them to the parsed report objects. Figure 49 shows the table resulting from the enhanced SSRS parser. The highlighted columns are added based on parsing the data source files.

ProjectN	FileName	FullPathToFile	Prefix	SourceType	DataSourceNam	DataSourceConne	DataSourceConne	Query	isExpres	DatasetName	SqlCom	SourceID
RS	ByVendor	RS\RSAdvent	ByVend	PROCEDUR	DW	localhost	DW	PurchasesByVe	0	ByVendorNam	SELEC	4571EAE
RS	ByVendor	RS\RSAdvent	ByVend	SQL	DW	localhost	DW	SELECT Na	0	VendorsList	SELEC	08F94A3
RS	Purchase	RS\RSAdvent	Purchas	MDX	Kuutio	localhost	AdventureWorks	EVALUATE SU	0	Purchase	SELEC	A7316C3
RS	Sales2012-2013	RS\RSAdvent	Sales	SQL	Sum	localhost	DW	SELECT * FRO	0	Sum	SELEC	E2EBFD1
RS	Vendor2012	RS\RSAdvent	Vendor	SQL	ProcedureVendc	localhost	DW	SELECT Vendo	0	VendorsPurch	SELEC	C4FA87B

Figure 49. Updated SSRS Parser outcome table

SSIS, SSRS and SSMS parsers run asynchronously, in which a parser commences after the preceding finish execution. Since an SQL parser object is created with each BI parser, the table resulting from the SQL parser gets overwritten with the initialization of each BI parser. Ideally, the SQL parser table should include data of parsed SQL statements collected from all BI parsers. This issue was fixed by creating one instance of SQL parser once the application executes and before initiating any parser. BI parsers then use the same SQL parser instant to insert the parsed SQL statement data into the same table. Figure 50 shows the new SQL parser resulting table, which includes parsed SQL statements related to all BI components. Figure 51 shows the old SQL parser resulting table, which included parsed statements related only to SSRS, as the SSRS parser ran the last.

ServerName	DatabaseN	SchemaNar	TableName	ColumnNa	Context	expression	UsageConte	hashCc	Source	ParentID
localhost	Adventure	Sales	Customer	*	READ	*	SELECT	59312	SSIS	AD79EAAC-
localhost	Adventure	Person	Person	*	READ	*	SELECT	11821	SSIS	B34A81A9-
localhost	Adventure	Production	Product	*	READ	*	SELECT	15879	SSIS	ADCFD6C5-
localhost	Adventure	Production	ProductCategc	*	READ	*	SELECT	-13033	SSIS	071340B7-9
localhost	Adventure	Production	ProductSubcat	*	READ	*	SELECT	96187	SSIS	E82D77A2-
localhost	DW	dbo	Vendor	Name	READ	Name	SELECT	90353	SSRS	08F94A3E-
localhost	DW	dbo	SUMMA	*	READ	*	SELECT	-17656	SSRS	E2EBFD1D-
localhost	DW	dbo	DimTime	VendorNa	READ	VendorName	SELECT	42065	SSRS	C4FA87BD-
localhost	DW	dbo	DimTime	LineTotal	READ	SUM(LineTotal) a	SELECT	14620	SSRS	C4FA87BD-
localhost	DW	dbo	Purchase	OrderDate	READ	P.OrderDate = dt	FROM	-60498	SSRS	C4FA87BD-
localhost	Adventure	Person	Person	BusinessE	READ	[BusinessEntityID	SELECT	-31032	SSMS	0B42E11D-
localhost	Adventure	Person	Person	FirstName	READ	[FirstName]	SELECT	17595	SSMS	0B42E11D-
localhost	Adventure	Person	Person	MiddleNar	READ	[MiddleName]	SELECT	87147	SSMS	0B42E11D-
localhost	Adventure	Person	Person	LastName	READ	[LastName]	SELECT	16574	SSMS	0B42E11D-
localhost	Adventure	Person	Person	rowguid	READ	[rowguid]	SELECT	-25227	SSMS	0B42E11D-
localhost	DW	dbo	Customer	*	READ	[dbo].[Customer].	SELECT	-17563	Tabular	CD34F908-
localhost	DW	dbo	DimTime	*	READ	[dbo].[DimTime].*	SELECT	-43082	Tabular	37C44378-E
localhost	DW	dbo	Product	*	READ	[dbo].[Product].*	SELECT	-74867	Tabular	D5472713-

Figure 50. Outcome table after fixing SQL parser

esults | Messages

Server	Database	Schema	Table	Column	Context	expression	UsageConte:	hashCo	ParentID
	DW	dbo	Vendor	Name	READ	Name	SELECT	14265:	1F948CEC
	ProcedureVen	dbo	DimTime	VendorNar	READ	VendorName	SELECT	-16838	9D461FA8
	ProcedureVen	dbo	DimTime	LineTotal	READ	SUM(LineTot	SELECT	12424:	9D461FA8
	ProcedureVen	dbo	Purchase	OrderDate	READ	P.OrderDate	FROM	11629:	9D461FA8
	ProcedureVen	dbo	DimTime	DKey	READ	P.OrderDate	FROM	24187:	9D461FA8
	ProcedureVen	dbo	DimTime	Year	READ	dt.Year = 201	WHERE	-16190	9D461FA8
	DW	dbo	Vendor	Name	READ	Name	SELECT	14265:	7A90B500
	ProcedureVen	dbo	DimTime	VendorNar	READ	VendorName	SELECT	-16838	D5FEED1E
	ProcedureVen	dbo	DimTime	LineTotal	READ	SUM(LineTot	SELECT	12424:	D5FEED1E
	ProcedureVen	dbo	Purchase	OrderDate	READ	P.OrderDate	FROM	11629:	D5FEED1E
	ProcedureVen	dbo	DimTime	DKey	READ	P.OrderDate	FROM	24187:	D5FEED1E
	ProcedureVen	dbo	DimTime	Year	READ	dt.Year = 201	WHERE	-16190	D5FEED1E

Figure 51. Outcome table from old SQL parser

More configuration options were added to increase the dynamicity of the application. The user can choose which parsers to use from the appsetting.json file.

Figure 52 shows the database tables produced by the final application when all parsers were used and ran successfully.

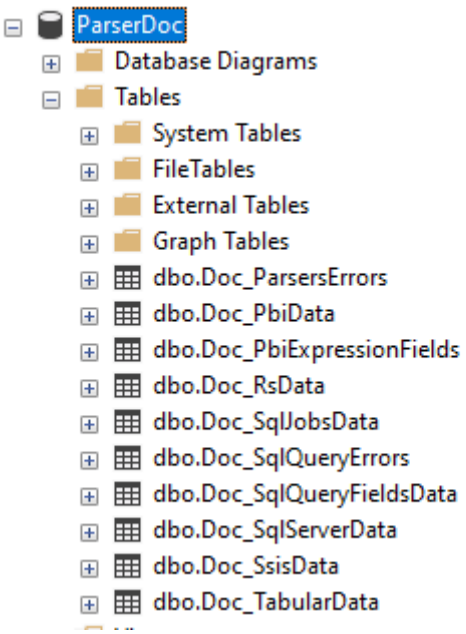

Figure 52. A list of tables resulting from running the updated application

8 Test Case (End-user case)

The project was first initiated according to the requirements of a Pinja customer to maintain their own data warehouse tables. It was intended to have insight into what BI data are related to each database table in the data warehouse. What database table in which database is related to which SSIS package, SSRS report and SSAS tabular model. Also, to know which SQL job runs which package and whether there is no job processing some SSIS packages. These job data will help the customer to maintain and clean up their data warehouse. This customer was considered the pilot customer to grant the user acceptance of the project. The customer decided not to use the PowerBI parser at this stage; hence the PowerBI parser was set to false in the configuration file. The application was installed manually on the customer's environment using copy and paste. Then using the command line, the parser was executed against only one server of the production environment servers. The SSIS parser executed successfully, but the SSRS parser failed. This failure caused the application to exit without continuing to parse other components, such as SSAS. An investigation was conducted on the failure, and it was found that reports which use shared data sources were not parsed, and only local data sources were parsed. After fixing the issue, an updated version of the application got published and executed against the same server in the customer's environment. The application ran successfully, and all SSIS, SSRS and SSAR components got parsed, in addition to the SQL jobs data.

The result of the project was an SQL procedure that collects data from all tables created upon the successful execution of the application. The procedure uses database and table names as parameters and then returns all data related to that table. The procedure was created in the production SQL server of the customer. Figure 53 shows the results after running the procedure on the local host.

esults | [:ssages

ServerN	Datab	Sche	TableNar	Source	ParentID	SourceName	SourceType	Relations	JobName	StepName	JobStatus	LastRunDat
localhos	DW	dbo	Purchase	SSMS	952E46C	AllVendorsPurcha	PROCEDUR	RsReport:				0
localhos	DW	dbo	Purchase	SSMS	79F392C	PurchasesByVenc	PROCEDUR	RsReport: ByVendor				0
localhos	DW	dbo	Purchase	SSRS	E4B224	RS\RSAdventure	SQL					0
localhos	DW	dbo	Purchase	SSIS	12CCC2	ETL\SSIS\Stagin	SSIS TASK		Run Adventu	Run FactsTc	enabled	20221212
localhos	DW	dbo	Purchase	Tabular	2CFFBB	Kuutio\SSAS\Adv		Ssis Package: Proce	Run Adventu	Process Adv	enabled	20221212
localhos	DW	dbo	Purchase	Tabular	2CFFBB	Kuutio\SSAS\Adv		RsReport: Purchase				0

Figure 53. Result of running the procedure on the local host

9 Results and Conclusion

This thesis aimed to create a tool to document the BI system and be able to relate database tables with BI components that read from or write to these tables. The paper discussed BI systems which are created using SQL server tools. As such, the tool collects data from all files generated by the BI solutions, which includes SSIS packages, SSRS reports, and analysis service tabular files. In addition, the tool collects data related to view and stored procedure objects in the database engine and the data related to jobs configured in the SQL server agent. The tool should create database tables to insert the collected data from each BI component. As customers maintain their BI files in Git repositories, the application should also include functionality to download projects from git repositories, which will facilitate running the tool automatically without a need for user interference in the execution process.

The result of this project is a console application which uses user inputs to run correctly. User inputs are gathered in the application settings file and then passed to the functionalities implemented as parameters. The tool downloads projects from the DevOps git repository in a zip file format. However, the functionality was found not to be working as expected. Zip files for the targeted projects are created, yet, the files are empty, which causes the application to fail. Temporarily, projects are copied to the root directory manually. Also, the application includes multiple main parsers that a user decides to enable or disable from the settings file; enabled parsers run upon the execution of the application. These parsers are SSIS, SSAS, SSRA and PowerBI parsers, which collect data from each BI component. However, other functionalities are not configurable by a user. These functionalities, such as the SSMS parser, complement the BI parsers' functionalities.

Another complementary functionality is the job collector. It runs if the SSIS parser is used. Furthermore, an SQL parser is also implemented to relate BI components with database tables which they access. This parser reads each SQL statement that is part of the data collected by the main parsers, then fetches database, tables and columns names used in each statement. The parser does not run separately but runs along with each of the SSIS, SSAS, SSRS, PowerBI and SSMS parsers. Successfully running the application produces several database tables, including data collected from each main parser and complementary functionality in a separate table. Files which fail to parse and collect their data are inserted into a separate table with error details for later investigation. A Pinja customer was selected for the user acceptance of the project. Part of the user acceptance was to create an SQL server procedure. The procedure aims to provide insight into data relations. The application was executed on the production environment of that customer manually using

the command line program. PowerBI parser (PbiParser) was not used in the pilot testing; hence user acceptance was not acquired for that specific feature.

During the implementation of the project, multiple enhancements have been foreseen. The requirements of the pilot customer initially drove the application scope; hence it is built based on their practices in developing the BI solution. The function to download projects from git repositories should be fixed and enhanced to include other types of git repositories besides the DevOps. The SSMS parser can be enhanced to collect other objects in the database engine, such as functions since other customers use them in SSIS packages. Not reading the data from function objects results in missing relations between these SSIS packages and the user database tables. SQL parser enhancements can also be considered. Currently, the SQL parser identifies the exitance of Select-, From- and Where- clauses. Other syntaxes are not recognized, which causes the SQL statement parsing to fail. Also, some less-common clauses are not recognized, and SQL statements do not parse accurately in these cases. SSIS parser improvements to recognize and parse any task in the control flow can be considered. Errors collected during the parsing processes can be investigated individually, and based on the findings, enhancements to parsers can be implemented in future releases. Appsetting.json file should not include the security data such as Azure embedding data and git tokens. Thus, a better security approach should be considered. The application can be improved by using the configuration providers provided by Microsoft. These providers can read configuration data from any configuration source rather than providing the path to the appsettings.json file as a parameter in the command line during the application execution. Also, parsing SSIS solutions implemented in Azure data factory is considered part of future releases, given the increased number of customers using the Azure services.

References

.Net. What is .Net? Accessed on 1.3.2023. Available at https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

Bigelow, S. Microsoft Azure. TechTarget. Accessed on 3.3.2023. Available at https://www.techtarget.com/searchcloudcomputing/definition/Windows-Azure

Frankenfield, J. 2022. What Is Business Intelligence (BI)? Types, Benefits, and Examples. Accessed on 13.2.2023. Available at https://www.investopedia.com/terms/b/business-intelligence-bi.asp

GeeksForGeeks a. Database Objects in DBMS. Accessed on 2.3.2023. Available at https://www.geeksforgeeks.org/database-objects-in-dbms/

IBM. ETL (Extract, Transform, Load). Accessed on 2.3.2023. Available at https://www.ibm.com/my-en/topics/etl

JavaTPoint a. SQL Server Management Studio (SSMS). Accessed on 20.2.2023. Available at https://www.javatpoint.com/sql-server-management-studio

JavaTPoint b. SSIS Tutorial. Accessed on 20.2.2023. Available at https://www.javatpoint.com/ssis

JavaTPoint c. SSRS Tutorial. Accessed on 20.2.2023. Available at https://www.javatpoint.com/ssrs#whatisssrs

Knowledgehut. Classes and Objects in C#. Accessed on 1.3.2023. Available at https://www.knowledgehut.com/tutorials/csharp/csharp-classes-objects

Microsoft a. Views. Accessed on 2.3.2023. Available at https://learn.microsoft.com/en-us/sql/relational-databases/views/views?view=sql-server-ver16

Microsoft b. Integration Services (SSIS) Packages. Accessed on 20.2.2023. Available at https://learn.microsoft.com/en-us/sql/integration-services/integration-services-ssis-packages?view=sql-server-ver16

Microsoft c. SQL Server Analysis Services Overview. Accessed on 23.02.2023. Available at https://learn.microsoft.com/en-us/analysis-services/ssas-overview?view=asallproducts-allversions

Microsoft d. Reporting Services Concepts (SSRS). Accessed on 21.2.2023. Available at https://learn.microsoft.com/en-us/sql/reporting-services/reporting-services-concepts-ssrs?view=sql-server-ver16

Microsoft e. SQL Server Agent. Accessed on 28.2.2023. Available at https://learn.microsoft.com/en-us/sql/ssms/agent/sql-server-agent?view=sql-server-ver16

Microsoft f. What is Power BI? Accessed on 23.2.2023. Available at https://learn.microsoft.com/en-us/power-bi/fundamentals/power-bi-overview

Microsoft g. Datasets in the Power BI service. Accessed on 28.2.2023. Available at https://learn.microsoft.com/en-us/power-bi/connect-data/service-datasets-understand

Microsoft h. A tour of the C# language. Accessed on 1.3.2023. Available at https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Microsoft i. Tabular Object Model (TOM). Accessed on 24.2.2023. Available at https://learn.microsoft.com/en-us/analysis-services/tom/introduction-to-the-tabular-object-model-tom-in-analysis-services-amo?view=asallproducts-allversions

Microsoft j. Programming Power BI datasets with the Tabular Object Model (TOM). Referenced on 24.2.2023 https://learn.microsoft.com/en-us/analysis-services/tom/tom-pbi-datasets?view=asallproducts-allversions

Microsoft k. Using the Power BI REST APIs. Accessed on 24.4.2023. Available at https://learn.microsoft.com/en-us/rest/api/power-bi/

Microsoft l. What is Power BI embedded analytics? Accessed on 24.2.2023. Available at https://learn.microsoft.com/en-us/power-bi/developer/embedded/embedded-analytics-power-bi

Pedamkar, P. Deserialization in C#. Accessed on 1.3.2023. Available at https://www.educba.com/deserialization-in-c-sharp/

Peterson, R. SAS Tutorial: What is SSAS Cube, Architecture & Types. Accessed on 21.2.2023. Available at https://www.guru99.com/ssas-tutorial.html

PowerBI. What is data visualization? Accessed on 6.3.2023. Available at https://powerbi.microsoft.com/en-us/data-visualization/

Sheldon, R. Class library (in object-oriented programming). Accessed on 1.3.2023. Available at https://www.techtarget.com/whatis/definition/class-library

Stedman, C. 2021. The ultimate guide to business intelligence in the enterprise. TechTarget. Accessed on 13.2.2023. Available at https://www.techtarget.com/searchbusinessanalytics/Ultimate-guide-to-business-intelligence-in-the-enterprise

Stedman, C; Burns, E.; Pratt, M. What is data preparation? An in-depth guide to data prep. Accessed on 13.2.2023. Available at https://www.techtarget.com/searchbusinessanalytics/definition/data-preparation

SQLServer Tutorial. What is SQL Server? Accessed on 20.2.2023. Available at https://www.sqlservertutorial.net/getting-started/what-is-sql-server/

Visual Studio. Accessed on 1.3.2023. Available at https://visualstudio.microsoft.com/

W3Schools. SQL Stored Procedures for SQL Server. Accessed on 1.3.2023. Available at https://www.w3schools.com/sql/sql_stored_procedures.asp