Datawarehousing & ETL – A22

Date: June 4th, 2023

Professor: Emerick Duval

Project contributors:

Lan LE : le.lan@edu.dsti.institute

Shi-Jiun HUANG : mori.huang@edu.dsti.institute

Emmanuel GAKOSSO : emmanuel.gakosso@edu.dsti.institute

Cyrille NWABO : cyrille.nwabo@edu.dsti.institute

Mamadou DIALLO : mamadou.diallo@edu.dsti.institute

PROJECT REPORT

Contents

*****

1. Project overview and identifying solution plan

ServiceSpot, an IT company, has approached us to assist them in analyzing their call center data. The company's call center receives daily calls from customers, and they require our expertise to develop an ETL project using SQL Server Integration Services (SSIS). The goal of this project is to load the call center data into the data warehouse of ServiceSpot for further analysis and reporting.

Business requirements

- Call Analysis: This involves identifying call volume trends over time (monthly, yearly), determining peak call volumes per day or per month, and calculating the call abandonment rate.
- Employee Performance Analysis: This includes identifying top-performing and low-performing employees based on waiting time within and outside the service level agreement (SLA), total number of calls handled, and number of calls abandoned per employee.
- Cost Analysis: This involves identifying call types with high charges by calculating the total charges incurred for each call type.

Solution plan

These business requirements provide a starting point for designing the ETL project with SSIS. The ETL process should extract the data from the provided CSV files, transform it as needed, such as: data type conversions, joining employee information, calculating call charges, etc, and load it into the data warehouse.

By implementing these requirements, ServiceSpot will be able to perform comprehensive analysis and gain valuable insights into their call center operations such as employee performance, call volume handled, call abandonment rate, cost analysis, and identify areas for improvement in order to optimize their service delivery.

2. Data Source Analysis and Mapping

Identifying key metrics and dimensions

- Together with the above-identified business requirements, the key metrics and dimensions that need to be captured in the data warehouse will be: TotallCalls; CallAbandoned; CallDuration; Waiting Time within and outside SLA; Total Charges
- Dimensions : Call Charges dimension ; Date dimension ; Time Dimension ; Employee dimension

Schema of data warehouse

Fact Table:

- CallTimeStamp, Datetime
- CallChargeKey, int
- CallDateKey, int
- EmployeeKey, int
- TimeKey, int
- CallDuration int
- WaitTime, int
- CallAbandoned, varchar (1)
- SLA Status varchar (11)

Dimension Tables:

(1) DimCallCharges

- CallChargeKey PK, int
- Call charge, numeric
- Year, varchar (4)
- CallType, varchar (50)
- CallTypeID, varchar (10)

(2) DimDate

- Datekey PK, int
- Date, Date
- Day, tinyint
- Daysuffix, char (2)
- Weekday, tinyint
- WeekDayName, varchar (10)
- WeekDayName_short, char (3)
- WeekDay_FirstLetter, char(1)
- DOWInMonth, tinyint
- DayOfYear, smallint
- WeekOfMonth, tinyint
- WeekOfYear, tinyint
- Month, tinyint

- MonthName, varchar (10)
- MonthName_short char (3)
- MonthName_FirstLetter char (1)
- Quarter, tinyint
- QuarterName, varchar (6)
- Year, int
- MMYYYY, char (6)
- MonthYear, char (7)
- IsWeekend, bit

(3) DimEmployees

- EmployeeKey PK, int
- StateCode, varchar (2)
- EmployeeID, varchar (10)
- Name, varchar (50)
- Region, varchar (50)
- SiteName, varchar (50)
- EmployeeName, varchar (50)
- ManagerName, varchar (50)

(4) DimTime

- TimeKey PK, int
- Hour, tinyint
- Minute, tinyint
- Second, tinyint
- TimeValue, time (7)
- TimePeriod, varchar (20)
- AMPM, varchar (2)

3. ETL Development with SSIS

First step: Data Staging Area

At this stage, we have used the SQL statement "TRUNCATE TABLE" to remove all data from the table. Following that, we imported 4 flat CSV files as data sources: Call charge, Call Types, Employees, States and all files from a specific folder containing call data infos.

For the staging area of call charges, Employees, US States, call types, we have used classic import. For call data, we have used Foreach loop container that allows us to loop over the folder containing the data over the years.

At this point, our focus is on temporarily storing the data in the staging area. We have successfully loaded the data without encountering any significant issues except with the original encoding of Call Charges file that we change to “1252 Latin”. The next step will involve performing the Data Transformation.

Second step: Operating Data Store

(i) ODS_DimCallCharges

- After loading the STA – CallCharges table as OLE source, we cleaned the name of CallType by using the TRIM function to remove all blank spaces and then removed any null values.
- The next step involves unpivoting the CallCharges table.
- A transformation is performed here to facilitate potentially further calculation of costs. Specifically, we removed "/ min" using the REPLACE function.
- Additionally, we validated if the CallCharge adheres to the NUM format and filtered out all invalid values into the "Technical rejects" table.
- Then, we did a lookup on Call Type label between Call charges table and Call Type to retrieve CallTypeID in the table of Call charges in order to have all these informations in the same table.
- Data resizing was performed before loading the data into the ODS CallCharges table.

(ii) ODS_DimCallData

- After loading the STA – CallData table as OLE source, we split the datetime column into Date and Time using the appropriate functions:
  - Date: TOKEN([CallTimestamp],” ”,1)
  - Time: TOKEN([CallTimestamp],” ”,2)
- Next, we individually validated the data types for the following columns: Date (DATE format); Time (TIME format); CallDuration (INT format); WaitTime (INT format); CallAbandoned (STR format), CallTimeStamp (DATETIME format);
  Any incorrect values found in these columns were consolidated and transferred to the "Technical rejects" table.
- To determine whether a call falls within or outside the SLA (Service Level Agreement), we applied the relevant function on WaitTime column : WaitTime_INT < 35 ? "Within SLA" : "Outside SLA"
- Extract year: We extracted the year from Date of each call, to retrieve then the corresponding call charge for a given type of call and year from call charges.
- Data resizing was performed before loading the data into the ODS CallData table.

(iii) ODS_DimEmployee

- The STA – Employees table was loaded as OLE source
- Then the "Site" column was split into "StateCode" and "SiteName" using the LEFT and RIGHT functions.
- The "EmployeeID" was validated to ensure it conforms to the required 7-character string length. Any incorrect data was rejected and sent to the "Technical rejects" table.
- The "StateCode" was validated to ensure it conforms to the required 2-character string length. Any incorrect data was rejected and sent to the "Technical rejects" table.
- A lookup transformation was utilized to retrieve the corresponding Name of the state and region based on the StateCode from US States table.
- Data resizing was performed before loading the data into the ODS Employees table.

Third step: Data Warehouse

(i) DWH_DimCallCharges

- ODS - callCharges table was loaded as OLE source.
- A lookup operation is performed to determine if there is an existing callCharge for a specific year and call type.
  - If the lookup operation does not match, the valid values are loaded into the Dimension table - CallCharges.
  - If match, then perform another lookup for any changes on a specific amount for the given year and call type data. If changes are found, update the callCharges table accordingly.

In detail, if a new row is added to the ODS table and this row is missing from the DWH table, this row is added to the corresponding table in the database because the search feeds the table with the elements that do not match. If, however, items are found, another lookup is performed to see if there has been a change in this row. If so, the row is updated. (SCD 1 application)

- The dimension table DimCallCharges is finally loaded into the data warehouse.

(ii) DWH_DimEmployee

- ODS - Employees table was loaded as OLE source.
- A lookup operation is performed to check if an employee already exists in the dimension table within the data warehouse by using the unique identifier (employee ID).
  - If the lookup operation does not match the existence of an employee in the current dimension table, the valid values are loaded into the Dimension table - Employee.
  - If not, further check will be performed to identify any changes for that employee. If it is the case, the employee record is updated with the new information. This step ensures that the dimension table stays up to date with the latest information.
- The dimension table DimEmployee is finally loaded into the data warehouse.

(iii) DWH_FactCallData

- The ODS – CallData table was loaded as OLE source.
- Once the data is loaded, it needs to be enriched with additional information from lookup tables specifically the technical key from each dimension. In this case, the lookup tables are:
  - DimCallCharges – Lookup on Year and Call type ID
  - DimDate: - Lookup on Date
  - DimEmployee – Look up on EmployeeID
  - DimTime – Lookup on time

Any records that cannot be matched are stored on a separate table called Functional Rejects.

- Finally, the valid records are loaded into the Fact table.

(iv) DWH_DimTime : Generated with SQL statement (see attached Appendix).

(v) *DWH_DimDate :* Generated with SQL statement (see attached Appendix).

Forth step: Testing - validation and Results

After performing the above steps, we have no error messages or warnings from SSIS after running each package.

- Our fact table contains 98,975 rows.
- Our DimCallCharges dimension table contains 12 rows. Each row contains a price per minute for each call type for a specific year.
- Our DimDate table contains 1,460 rows with dates ranging from 01 January 2018 to 30 December 2021. This covers the scope of the data in the facts table.
- Our DimTime table contains 86,400 rows representing each second of hours in a day.
- Our DimEmployees table contains 64 rows.

Conclusion

In conclusion, the ETL solution plan with SSIS for ServiceSpot has been successfully executed, resulting in the development of a comprehensive data analysis solution for their call center operations. And it has encompassed key business requirements such as call analysis, employee performance analysis, and cost analysis. By implementing the solution plan, ServiceSpot can now gain valuable insights into call volume trends, peak call volumes, employee performance, call abandonment rates, and cost analysis. The data warehouse, comprising the fact table and dimension tables, has been designed and populated with the necessary metrics and dimensions to support the analysis. The ETL development process involved stages such as data staging, data transformation, and loading into the operating data store and data warehouse. Through meticulous data validation, cleaning, and enrichment, the ETL process ensured the accuracy and reliability of the data.

Overall, the successful execution of the ETL project equips ServiceSpot with the tools to optimize their call center operations, identify areas for improvement, and enhance their service delivery.

4. SQL queries

See Appendix

SSIS Project - Call center analysis: SQL queries

CONTENTS

SQL queries for creating STA Tables

STA - callCharges

```
CREATE TABLE [callCharges] (
    [call Type] varchar(255),
    [Call Charges (2018)] varchar(255),
    [Call Charges (2019)] varchar(255),
    [Call Charges (2020)] varchar(255),
    [Call Charges (2021)] varchar(255)
)
```

STA - callData

```
CREATE TABLE [callData] (
    [CallTimestamp] varchar(255),
    [Call Type] varchar(255),
    [EmployeeID] varchar(255),
    [CallDuration] varchar(255),
    [WaitTime] varchar(255),
    [CallAbandoned] varchar(255)
)
```

STA - callTypes

```
CREATE TABLE [callTypes] (
    [CallTypeID] varchar(255),
    [CallTypeLabel] varchar(255)
)
```

STA - Employees

```
CREATE TABLE [Employees] (
    [EmployeeID] varchar(255),
    [EmployeeName] varchar(255),
    [Site] varchar(255),
    [ManagerName] varchar(255)
)
```

STA - US States

```
CREATE TABLE [US_States] (
    [StateCD] varchar(255),
    [Name] varchar(255),
    [Region] varchar(255)
)
```

SQL queries for creating ODS Tables

ODS - callCharges

```
CREATE TABLE [callCharges] (
    [Call charge] numeric(10,2),
    [Year] varchar(4),
    [Call Type] varchar(50),
    [CallTypeID] varchar(10)
)
```

ODS - callData

```
CREATE TABLE [callData] (
    [Date] date,
    [Time] time,
    [CallDuration] int,
    [WaitTime] int,
    [CallTimeStamp] datetime,
    [Call Type] varchar(1),
    [EmployeeID] varchar(7),
    [Year] varchar(4),
    [SLA Status] varchar(11),
    [CallAbandoned] varchar(1),
)
```

ODS - Employees

```
CREATE TABLE [Employees] (
    [StateCode] varchar(2),
    [EmployeeID] varchar(10),
    [Name] varchar(50),
    [Region] varchar(50),
```

```
    [SiteName] varchar(50),
    [EmployeeName] varchar(50),
    [ManagerName] varchar(50)
)
```

SQL queries for creating tables in DWH

DWH - DimCallCharges table

```
USE [A22_DWH]
GO

/****** Object:  Table [dbo].[callCharges]    Script Date: 25/05/2023 20:38:49 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[DimCallCharges](
  [CallChargeKey] INT PRIMARY KEY IDENTITY(1,1),
  [Call charge] [numeric](10, 2) NULL,
  [Year] [varchar](4) NULL,
  [Call Type] [varchar](50) NULL,
  [CallTypeID] [varchar](10) NULL
) ON [PRIMARY]
GO
```

Update query to update DimCallCharges

The purpose of this query is to update the DimCallCharges table if any changes occur for an amount on a type and year (SCD 1 applied).

```
UPDATE [dbo].[DimCallCharges]
   SET [Call charge] = ?
      ,[Call Type] = ?
 WHERE ([Year] = ? AND [CallTypeID] = ?)
```

DWH - FactCallData table

```
CREATE TABLE [FactCallData] (
    [CallTimestamp] Datetime,
    [CallChargeKey] int,
    [CallDateKey] int,
    [EmployeeKey] int,
    [TimeKey] int,
    [CallDuration] int,
    [WaitTime] int,
    [CallAbandoned] varchar(1),
    [SLA Status] varchar(11)
)
```

DWH - DimEmployee table

```
USE [A22_DWH]
GO

/****** Object:  Table [dbo].[Employees]    Script Date: 25/05/2023 19:43:29 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[DimEmployee](
  [EmployeeKey] INT PRIMARY KEY IDENTITY(1,1),
  [StateCode] [varchar](2) NULL,
  [EmployeeID] [varchar](10) NULL,
  [Name] [varchar](50) NULL,
  [Region] [varchar](50) NULL,
  [SiteName] [varchar](50) NULL,
  [EmployeeName] [varchar](50) NULL,
  [ManagerName] [varchar](50) NULL
) ON [PRIMARY]
GO
```

Update query to update DimEmployee

The purpose of this query is to update the DimEmployee table if any changes occur (SCD 1 applied)

```
USE [A22_DWH]
GO

UPDATE [dbo].[DimEmployee]
   SET [StateCode] = ?
      ,[Name] = ?
      ,[Region] = ?
      ,[SiteName] = ?
      ,[EmployeeName] = ?
      ,[ManagerName] = ?
 WHERE [EmployeeID] = ?
GO
```

DWH - DimTime table

The purpose is to keep only the technical key from times as other dimensions.

Create DimTime Table

```
CREATE TABLE DimTime (
  TimeKey INT IDENTITY(1,1) PRIMARY KEY, -- Auto-increment primary key
  Hour TINYINT,
  Minute TINYINT,
  Second TINYINT,
  TimeValue TIME,
  TimePeriod VARCHAR(20), -- Field to store time period
  AMPM VARCHAR(2) -- Field to store AM or PM
);
```

Populate DimTime Table

```
DECLARE @Time INT = 0;
```

```
WHILE @Time < 86400 -- total seconds in a day
BEGIN
  DECLARE @Hour TINYINT = @Time / 3600;
  DECLARE @Minute TINYINT = (@Time % 3600) / 60;
  DECLARE @Second TINYINT = @Time % 60;
  DECLARE @TimeValue TIME = DATEADD(SECOND, @Time, CAST('00:00:00' AS TIME));

  DECLARE @TimePeriod VARCHAR(20) =
    CASE
      WHEN @Hour < 5 THEN 'Night'
      WHEN @Hour < 12 THEN 'Morning'
      WHEN @Hour < 17 THEN 'Afternoon'
      WHEN @Hour < 20 THEN 'Evening'
      ELSE 'Night'
    END;

  DECLARE @AMPM VARCHAR(2) =
    CASE
      WHEN @Hour < 12 THEN 'AM'
      ELSE 'PM'
    END;

  INSERT INTO DimTime (Hour, Minute, Second, TimeValue, TimePeriod, AMPM)
  VALUES (@Hour, @Minute, @Second, @TimeValue, @TimePeriod, @AMPM);

  SET @Time = @Time + 1;
END;
```

DWH - DimDate table

```
CREATE TABLE dbo.DimDate (
   DateKey INT NOT NULL PRIMARY KEY,
   [Date] DATE NOT NULL,
   [Day] TINYINT NOT NULL,
   [DaySuffix] CHAR(2) NOT NULL,
   [Weekday] TINYINT NOT NULL,
   [WeekDayName] VARCHAR(10) NOT NULL,
   [WeekDayName_Short] CHAR(3) NOT NULL,
   [WeekDayName_FirstLetter] CHAR(1) NOT NULL,
   [DOWInMonth] TINYINT NOT NULL,
   [DayOfYear] SMALLINT NOT NULL,
   [WeekOfMonth] TINYINT NOT NULL,
   [WeekOfYear] TINYINT NOT NULL,
   [Month] TINYINT NOT NULL,
   [MonthName] VARCHAR(10) NOT NULL,
   [MonthName_Short] CHAR(3) NOT NULL,
   [MonthName_FirstLetter] CHAR(1) NOT NULL,
   [Quarter] TINYINT NOT NULL,
   [QuarterName] VARCHAR(6) NOT NULL,
   [Year] INT NOT NULL,
   [MMYYYY] CHAR(6) NOT NULL,
   [MonthYear] CHAR(7) NOT NULL,
   IsWeekend BIT NOT NULL,
   )


   GO


   SET NOCOUNT ON

TRUNCATE TABLE DimDate

DECLARE @CurrentDate DATE = '2018-01-01'
DECLARE @EndDate DATE = '2021-12-31'

WHILE @CurrentDate < @EndDate
BEGIN
   INSERT INTO [dbo].[DimDate] (
```

```
      [DateKey],
      [Date],
      [Day],
      [DaySuffix],
      [Weekday],
      [WeekDayName],
      [WeekDayName_Short],
      [WeekDayName_FirstLetter],
      [DOWInMonth],
      [DayOfYear],
      [WeekOfMonth],
      [WeekOfYear],
      [Month],
      [MonthName],
      [MonthName_Short],
      [MonthName_FirstLetter],
      [Quarter],
      [QuarterName],
      [Year],
      [MMYYYY],
      [MonthYear],
      [IsWeekend]
      )
   SELECT DateKey = YEAR(@CurrentDate) * 10000 + MONTH(@CurrentDate) * 100 + DAY(@CurrentDate),
      DATE = @CurrentDate,
      Day = DAY(@CurrentDate),
      [DaySuffix] = CASE
         WHEN DAY(@CurrentDate) = 1
            OR DAY(@CurrentDate) = 21
            OR DAY(@CurrentDate) = 31
            THEN 'st'
         WHEN DAY(@CurrentDate) = 2
            OR DAY(@CurrentDate) = 22
            THEN 'nd'
         WHEN DAY(@CurrentDate) = 3
            OR DAY(@CurrentDate) = 23
            THEN 'rd'
         ELSE 'th'
         END,
      WEEKDAY = DATEPART(dw, @CurrentDate),
      WeekDayName = DATENAME(dw, @CurrentDate),
      WeekDayName_Short = UPPER(LEFT(DATENAME(dw, @CurrentDate), 3)),
      WeekDayName_FirstLetter = LEFT(DATENAME(dw, @CurrentDate), 1),
      [DOWInMonth] = DAY(@CurrentDate),
      [DayOfYear] = DATENAME(dy, @CurrentDate),
      [WeekOfMonth] = DATEPART(WEEK, @CurrentDate) - DATEPART(WEEK, DATEADD(MM, DATEDIFF(MM, 0, @CurrentDate), 0)) + 1,
      [WeekOfYear] = DATEPART(wk, @CurrentDate),
      [Month] = MONTH(@CurrentDate),
      [MonthName] = DATENAME(mm, @CurrentDate),
      [MonthName_Short] = UPPER(LEFT(DATENAME(mm, @CurrentDate), 3)),
      [MonthName_FirstLetter] = LEFT(DATENAME(mm, @CurrentDate), 1),
      [Quarter] = DATEPART(q, @CurrentDate),
      [QuarterName] = CASE
         WHEN DATENAME(qq, @CurrentDate) = 1
            THEN 'First'
         WHEN DATENAME(qq, @CurrentDate) = 2
            THEN 'second'
         WHEN DATENAME(qq, @CurrentDate) = 3
            THEN 'third'
         WHEN DATENAME(qq, @CurrentDate) = 4
            THEN 'fourth'
         END,
      [Year] = YEAR(@CurrentDate),
      [MMYYYY] = RIGHT('0' + CAST(MONTH(@CurrentDate) AS VARCHAR(2)), 2) + CAST(YEAR(@CurrentDate) AS VARCHAR(4)),
      [MonthYear] = CAST(YEAR(@CurrentDate) AS VARCHAR(4)) + UPPER(LEFT(DATENAME(mm, @CurrentDate), 3)),
      [IsWeekend] = CASE
         WHEN DATENAME(dw, @CurrentDate) = 'Sunday'
            OR DATENAME(dw, @CurrentDate) = 'Saturday'
            THEN 1
         ELSE 0
         END

   SET @CurrentDate = DATEADD(DD, 1, @CurrentDate)
END
```

FunctionnalRejects table

```
CREATE TABLE [FunctionalRejects] (
[RejectDate] datetime,
[RejectPackageAndTask] nvarchar(202),
[RejectColumn] nvarchar(255),
[RejectDescription] nvarchar(500),
[Number of rejects] int
)
```