**Datawarehousing & ETL**

**Call Center ETL Report**

Prepared by

**Caique Dias**

[**caique.miranda-dias@edu.dsti.institute**](mailto:caique.miranda-dias@edu.dsti.institute)

**Immaculate Nabuggwawo**

**immaculate.nabuggwawo@edu.dsti.institute**

**Oluwafemi Oladapo Adeleke**

**oluwafemi-oladapo.adeleke@edu.dsti.institute**

November 24, 2022

**Instructions for Project Setup**

This report is to explain step by step how to use the project files. You can extract he **.rar** file anywhere, the important thing is not to separate the contents of the *CallCenterProject\_ETL* folder. After extracting, you will see the folder organized as follows:

Table

Description automatically generated

The *Data* folder is the supplied data (**.xls** and **.csv** files) for the project. The *CallCenterProject\_ETL.sln* is the project solution, which allows you to open the visual studio.

We also have four scripts to create the tables for the SQL databases in SSMS (SQL Server Management Studio).

Graphical user interface, text, table

Description automatically generated

The rest of the content is used by the SSIS solution.

The first step is open SSMS and create 4 databases:

* **Staging** – The staging area is where we will extract the data directly from the source and transform them into structured tables in the SQL database.
* **Operational Data Store** – This is the area where we will do some technical operations extracting staging data, such as changing data types.
* **Datawarehouse** – This is the final area of ​​transformation. The function of the data warehouse is to centralize information that can be analyzed to make better decisions. It must be clean and structured in fact tables and dimension tables.
* **Administration** – The administration database will receive the lines where there were technical and functional errors during the transformations.

Diagram

Description automatically generated

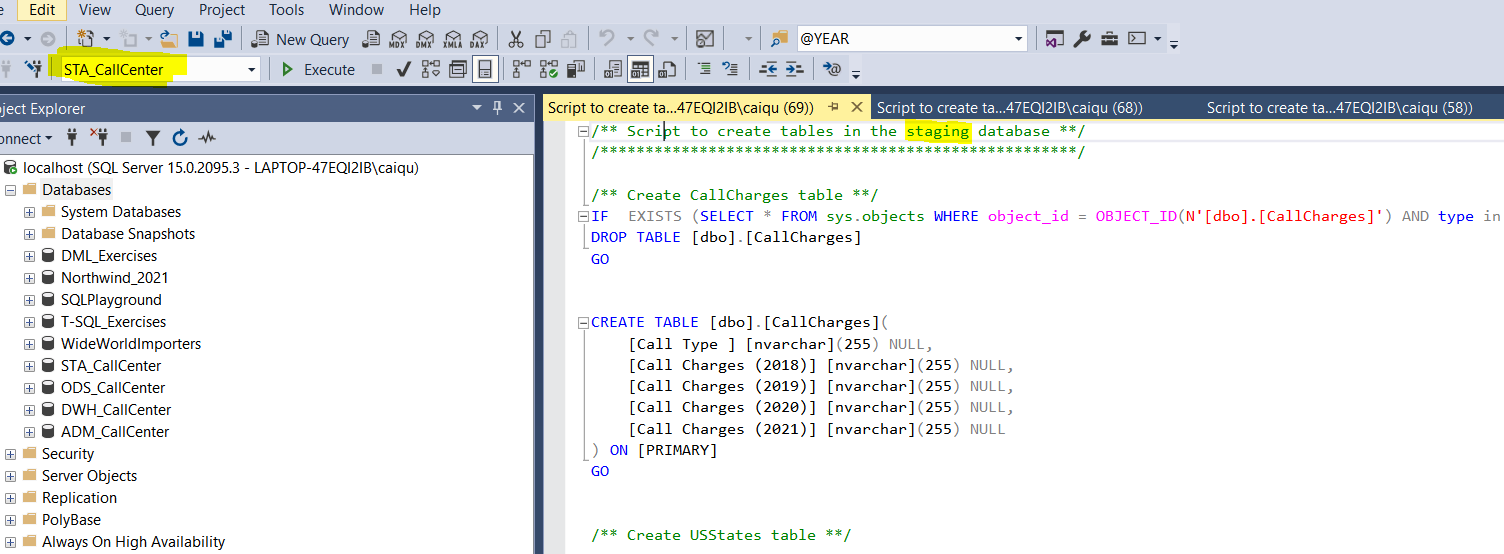
You don't have to worry about the name of the bases that you are going to create, because you will parameterize them later, however, we recommend that you create them with a name something like this:

Graphical user interface, text, application

Description automatically generated

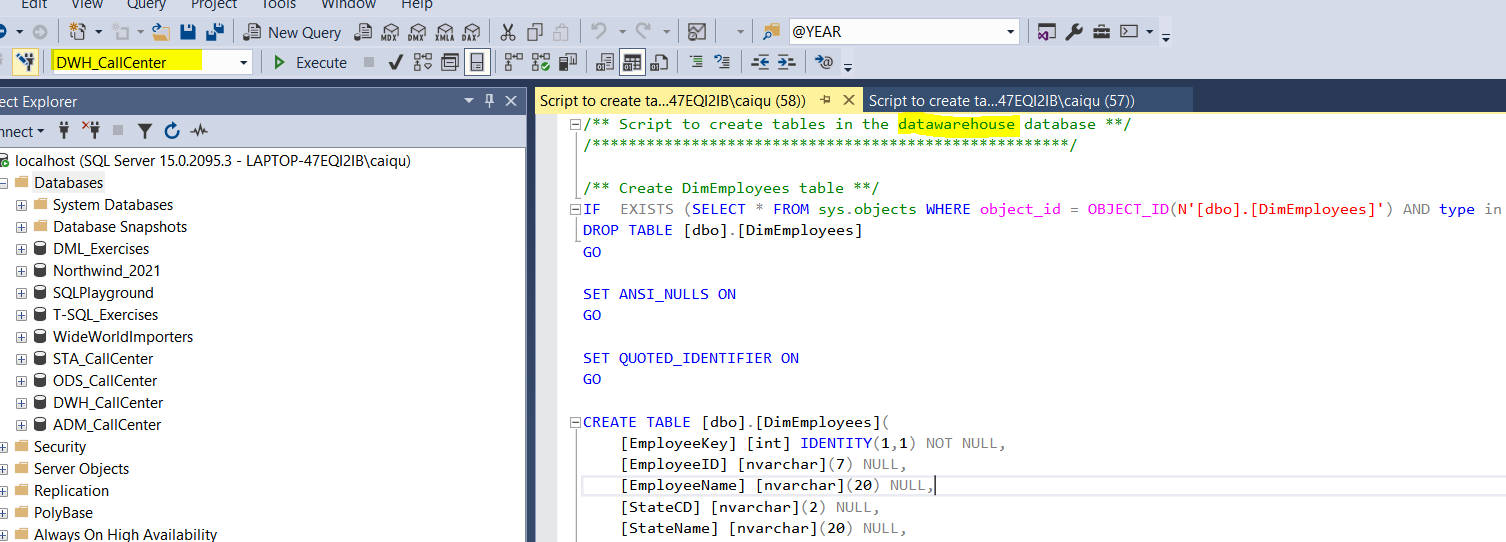
The second step is open the SQL table creation scripts in SSMS, note that there is one script for each database.

It’s necessary to execute the scripts selecting respectively the corresponding databases, like this:



Graphical user interface, text, application

Description automatically generated



Graphical user interface, application

Description automatically generated

The third step is to open the project solution, the **.sln** file. After opening the solution, go to solution explorer and open the project parameters.

Graphical user interface, text, application, chat or text message

Description automatically generated

At this moment, you will input parameters such as the name of databases created, the server’s name and the directory of the *Data* folder.

Graphical user interface

Description automatically generated with medium confidence

Now save and open the package named **Scheduler**.

Text

Description automatically generated

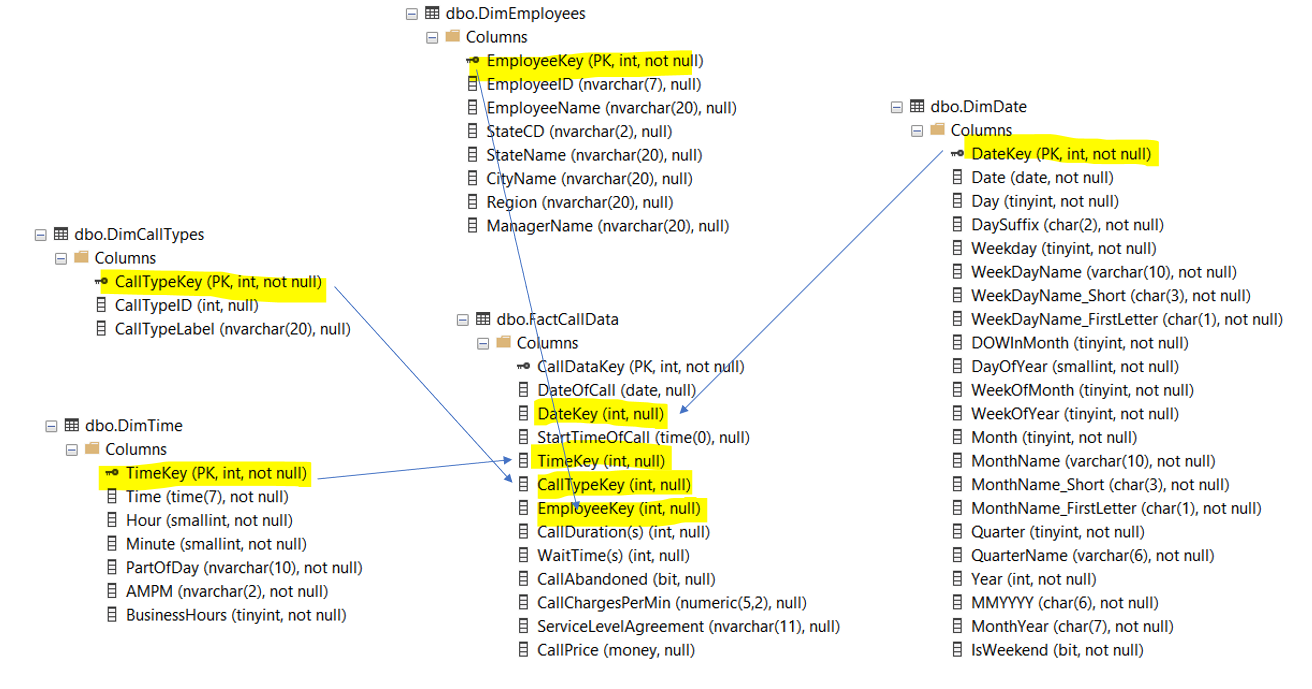
Just run this package and all databases will be fed.

**Datawarehouse Design**

The final DWH structure is shown below. It consists of one fact table and four dimension tables.

Implied relationships are indicated on the diagram, however, please note these have not been defined in the database.

We decided that the data warehouse should be built with a star schema.



* **DimEmployees** – This dimension consists of Employee details (ID, name), their work location and their manager’s name, together with a technical key. Full location details were added by looking up the USStates data set during the ETL process. We chose to model this dimension as a type 1 slowly changing dimension, because it’s more appropriate to overwrite the existing data with the new data.

Graphical user interface, text

Description automatically generated

* **DimCallTypes** – This dimension consists of the technical key CallTypeKey, Call Type Description and a source key Call Type ID. We chose to model this dimension as a type 1 slowly changing dimension, because it’s more appropriate to overwrite the existing data with the new data.

Text, letter

Description automatically generated

* **DimTime** – This dimension is a table of all possible times in a day, broken down into components, along with value-added derived fields for analysts. This allows analysts to directly filter or query the data on them, without needing to apply functions to parse timestamps. The smallest granularity is minutes, as the data which will reference this dimension does not include seconds. The table size is therefore 1440 rows. does not include seconds. The table size is therefore 1440 rows.

Text

Description automatically generated

* **DimDate** –As for the time dimension, DimDate is a table of all possible dates within a given period, broken down into useful components and value-added fields, plus a technical key.

Text

Description automatically generated

* **FactCallData** –Our fact table consists of (foreign) keys corresponding to each dimension (it’s not configured in SSMS), all call metrics (including some derived fields) like the Call price and a Service Level Agreement.

Text

Description automatically generated with medium confidence

**SSIS Packages Considerations**

We will start by explaining the package **Scheduler.** It’s nothing more than a sequence of package executions, which can be programmed through a job to run periodically.

We will now explain all the packages following this sequence.

* **STA – US States** – This package initially empties an existing US States table on staging and extracts the data from the excel file found in the data folder and loads them into staging.
* **STA - Employees** – This package initially empties an existing Employees table on staging and extracts the data from the excel file found in the data folder and loads them into staging.
* **STA – Call Types** – This package initially empties an existing Call Types table on staging and extracts the data from the excel file found in the data folder. After that, it converts the float value “CallTypeID” into unicode.

At this point we create an intermediate key to do a lookup with the call charges table later. It was a solution we found, because the Call Charges table does not have a column as a key and will be needed later. The destination is a staging database.

* **STA – Call Charges** – This package initially empties an existing Call Charges table on staging and extracts the data from the excel file found in the data folder. After that, it selects only the non-null values ​​of the column “Call Type”, and the destination is a staging database.
* **STA – Call Data** – This package is a little more complex, as we extract data from several .csv files in a folder called "Calls data". That's why we use a for each loop container that maps a variable used as the connection string of the flat file connection. This package initially empties an existing Call Data table on staging and with each loop, we extract and load the staging data.
* **ODS – Employees**– This package initially empties an existing Employees table on operational data store and extracts the data from the staging. We do some operations such as checking if the source key is empty or not, we divide the site column into StateCD and StateName, we resize the string columns, we collect all possible errors in the table Technical\_Rejects that is in the administration database and finally we load the data into the ODS database.
* **ODS – Call Types** – This package initially empties an existing Call Types table on operational data store and extracts the data from the staging. We do some operations such as checking if the source key is empty or not, we resize the string columns, we collect all possible errors in the table Technical\_Rejects that is in the administration database, we change the data type to integer and finally we load the data into the ODS database.
* **ODS – Call Charges** – This package initially empties an existing Call Charges table on operational data store and extracts the data from the staging. We do some operations such as create a lookup key to retrieve the Call Type ID column in another table, we do the unpivoting task, we change the data type to integer, we extract only the numeric value at Charges column, we resize the string columns, we collect all possible errors in the table Technical\_Rejects that is in the administration database and finally we load the data into the ODS database.
* **ODS – Call Data** – This package initially empties an existing Call Data table on operational data store and extracts the data from the staging. We do some operations such as convert unicode to datetime, we divide the CallTimeStamp column into DateOfCall and StartTimeOfCall, convert unicode to int and bool, we resize the string columns, we collect all possible errors in the table Technical\_Rejects that is in the administration database and finally we load the data into the ODS database.
* **DWH – Dim Employees** – This package extracts data from the ODS database and loads it into the datawarehouse. Basically we do a type 1 slow dimension change to feed the Dim Employees.
* **DWH – Dim Call Types** – This package extracts data from the ODS database and loads it into the datawarehouse. Basically we do a type 1 slow dimension change to feed the Dim Call Types.
* **DWH – Fact Call Data** – This package initially empties an existing Call Data table on datawarehouse and extracts the data from the ODS database. We do a lookup to recover all technical keys of all dimensions, after we extract the call charge information to calculate the call price. After we add a ServiceLevelAgreement column that classifies the call waiting time. We collect all possible errors in the table Functional\_Rejects that is in the administration database and finally we load the data into the Fact table on datawarehouse.