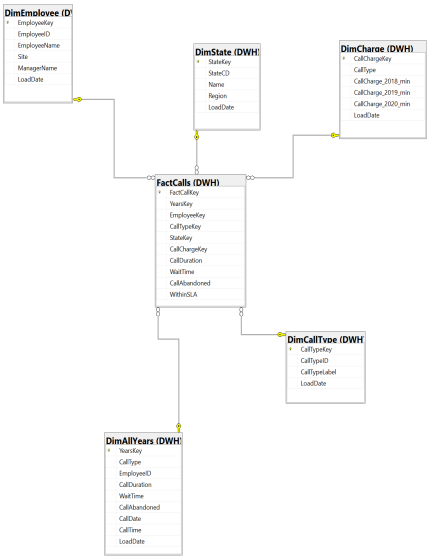
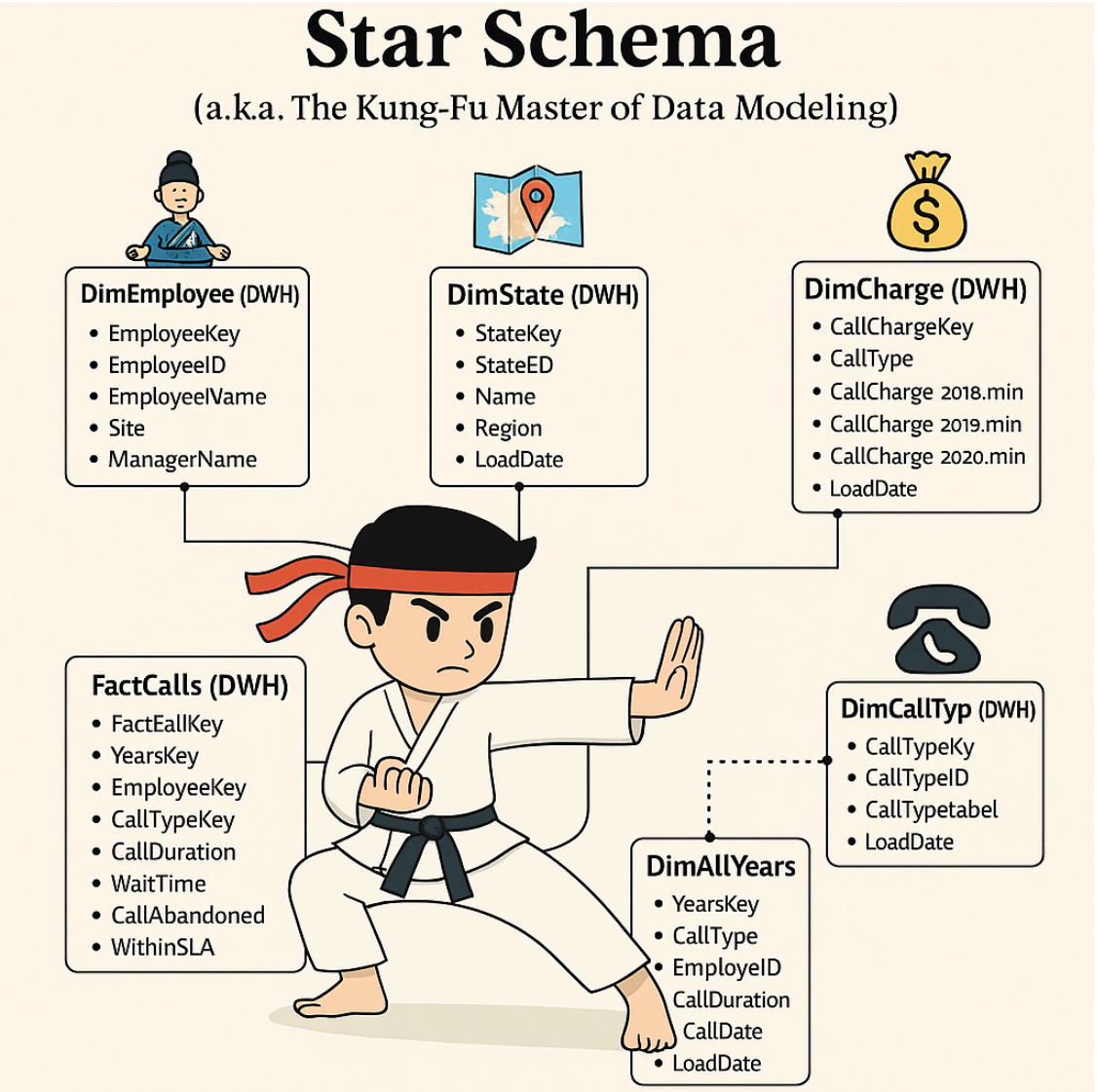
# A24

# ETL Data Warehouse

# &

# Power BI Dashboard Project



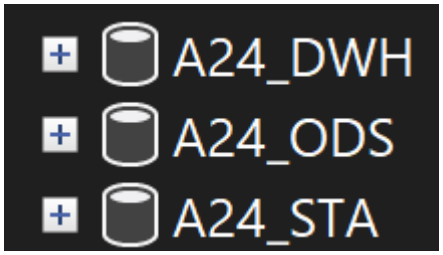



# Table of Contents

# Introduction

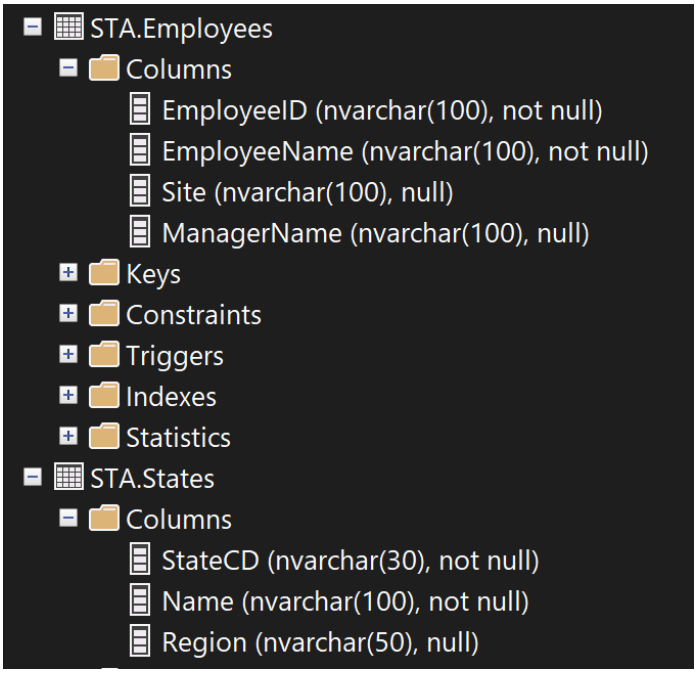
This report presents the comprehensive design, development, and analysis of the ETL (Extract, Transform, Load) process for ServiceSpot's call center dataset. The primary objective of this project is to convert multiple years of raw call data into a well-structured Data Warehouse (DWH) to support efficient data analysis and reporting. The ETL pipeline was designed and implemented using SQL Server Integration Services (SSIS), enabling automated data extraction, transformation, and loading into a star-schema data model. The resulting dataset serves as the foundation for interactive analytical dashboards developed in Power BI, providing actionable insights into call center performance and operational trends.

# 0. Database creation (STA)




Before implementing the ETL process in SSIS, the necessary databases and table structures were created in SQL Server. At this stage, the tables were not populated with data; instead, the focus was on establishing the database framework to ensure that the data could later be populated automatically through the SSIS workflows.

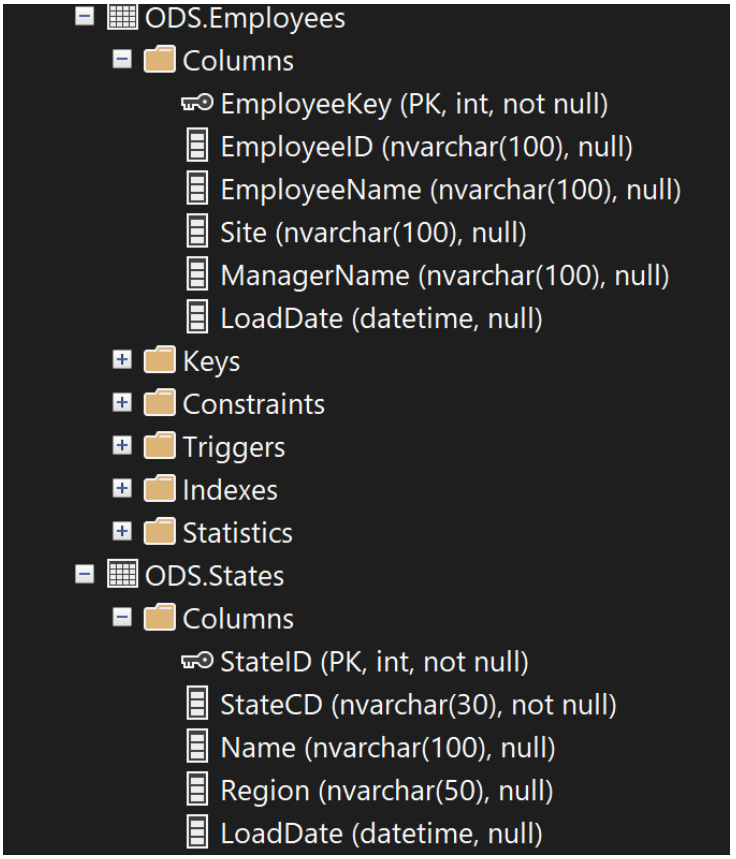
For the initial Stage of STA all the data format used in the Columns is in NVARCHAR

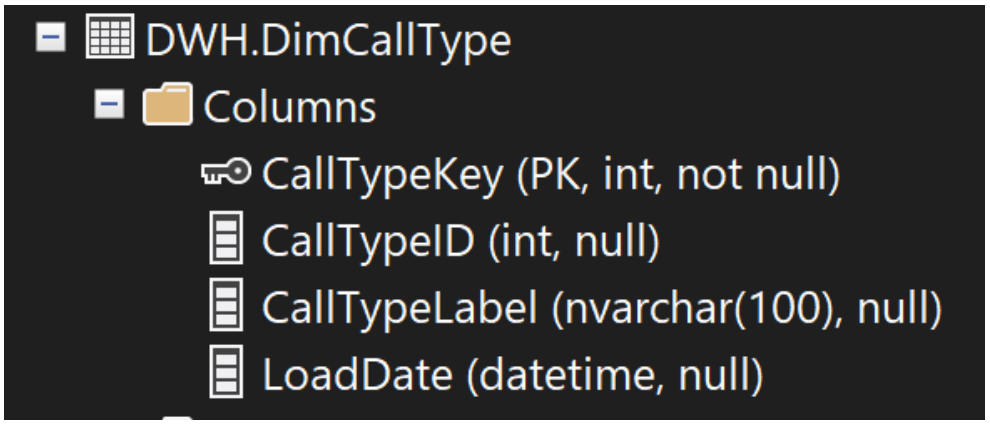



Although the use of **NVARCHAR** data types is not generally considered the optimal choice for numeric or date fields, we intentionally adopted it during the initial database creation phase to prevent compatibility issues with SSIS. During early testing, we encountered format mismatches where SSIS outputs were not being accepted by SQL Server tables due to strict data type constraints. To streamline the loading phase, we therefore chose to define columns as NVARCHAR to ensure flexible data ingestion without conversion errors.

This approach significantly reduced development time, as it eliminated the need to review and assign specific data types for each column at this stage. Since the **Fact tables** ultimately store numeric identifiers (primarily **INTEGER** values), the data types can be refined and transformed appropriately during the later stages of the ETL process. Thus, this decision was a deliberate trade-off between strict schema enforcement and practical efficiency during the early stages of development.

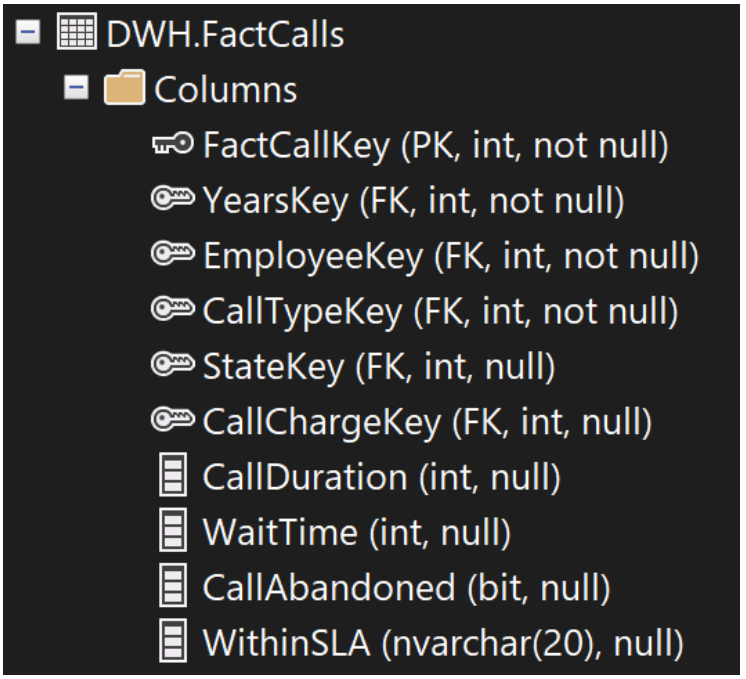
This is applicable to the ODS as well.




The **Data Warehouse (DWH)** was not created until the completion of the **Operational Data Store (ODS)** phase. By that stage, we had a clear understanding of the data requirements and relationships, allowing us to define the schema more precisely. For example, we determined that fields such as **CallTypeID** should be stored as **INT** data types to ensure efficient joins and optimized query performance within the DWH. This step marked the transition from flexible, intermediate staging to a structured and optimized data model ready for analytical processing.

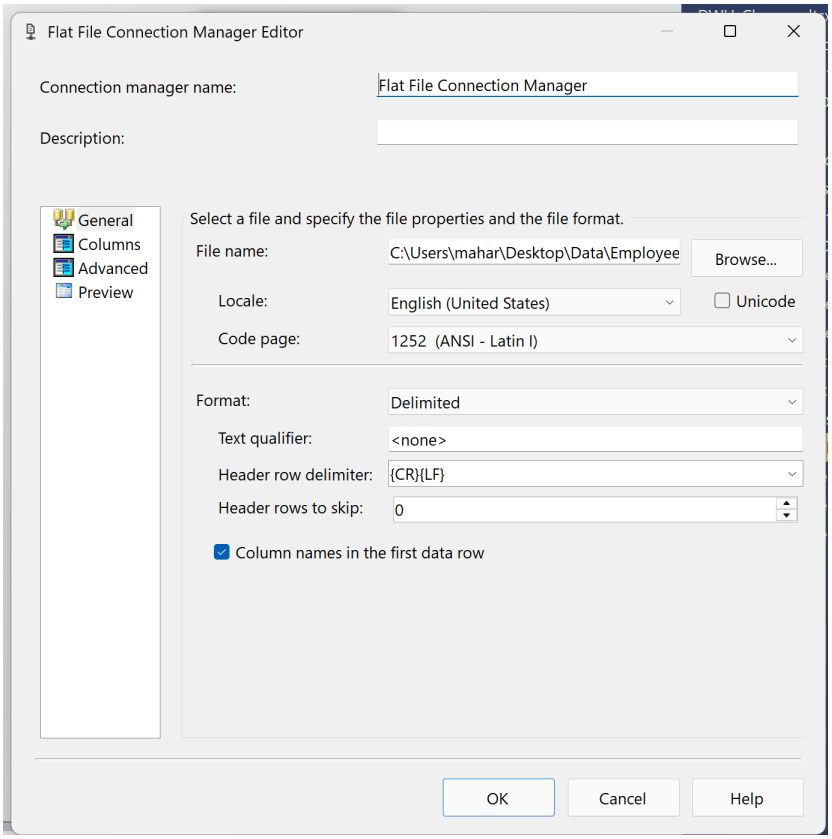
Below is the first version of our **Fact Table**, which was developed iteratively as the project progressed. Instead of enforcing a rigid schema from the beginning, we created and adjusted column profiles as needed throughout the ETL process. This adaptive approach provided greater flexibility, allowing us to focus more on refining the data flow and transformation logic rather than being constrained by predefined structures. As a result, the Fact Table evolved naturally to align with the actual analytical requirements identified during implementation.

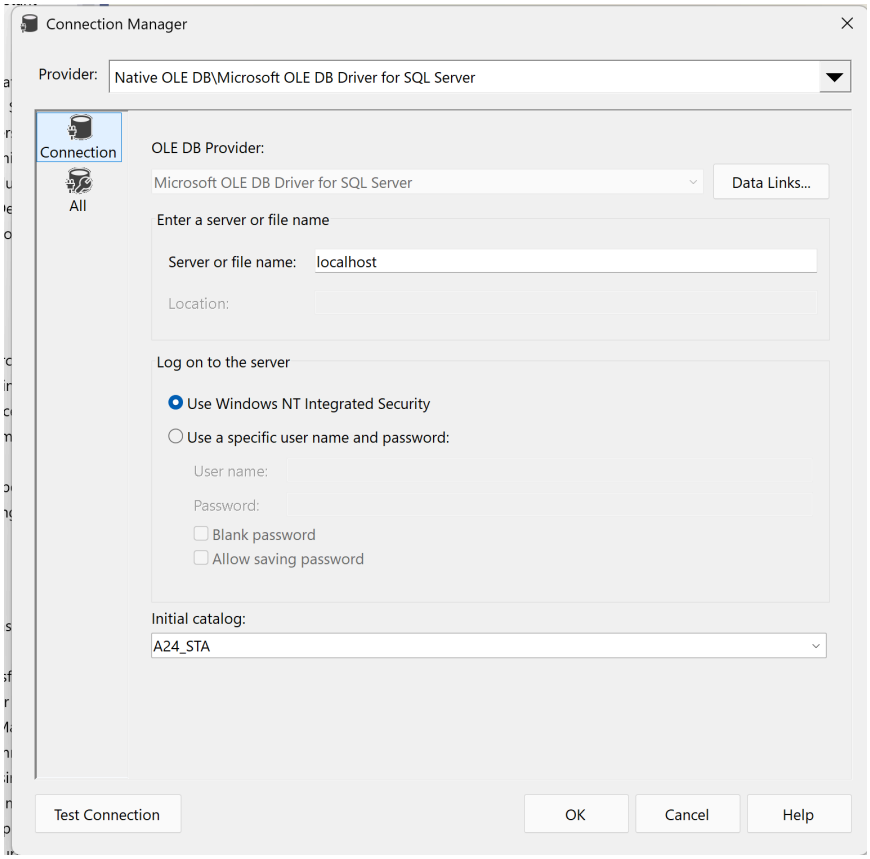



# I. Staging (STA)

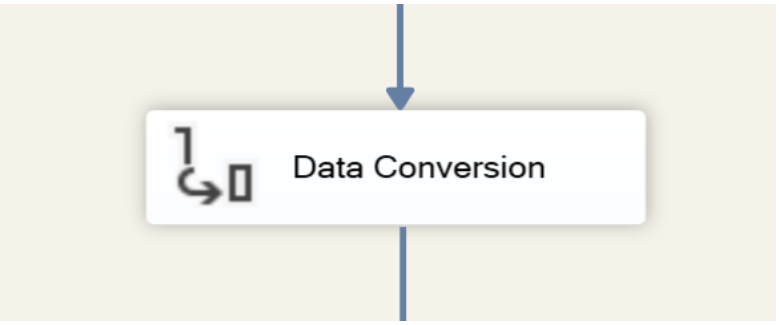
The **Staging Area (STA)** serves as the initial layer of the ETL process, designed to temporarily store raw data before transformation. In this phase, we imported the CSV files containing multiple years of call, employee, and charge data using **Flat File Connection Managers** in SSIS. A **For Each Loop Container** was implemented to dynamically iterate through each yearly dataset, enabling automation and scalability for future data imports.

This stage represents the **loading phase** of the CSV files into the SQL Server environment, ensuring that all raw data is centralized and readily available for subsequent transformation and cleansing operations.



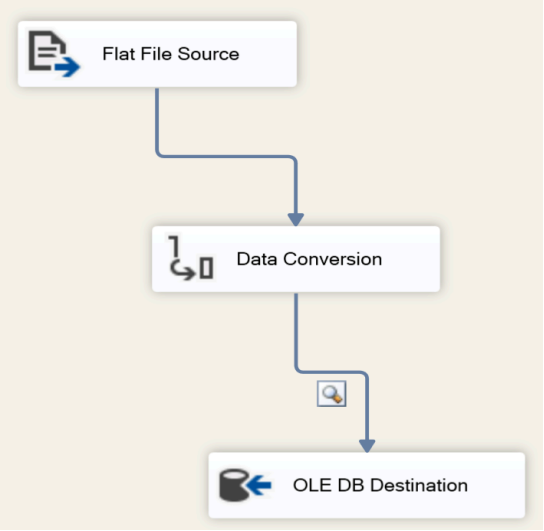



Next, we configured a **Destination Connection Manager** in SSIS to load the extracted data into the **STA (Staging Area) database** that we had previously created in SQL Server. This component establishes the link between the source files and the target database tables, ensuring that all imported data is correctly mapped and transferred. By defining the STA database as the destination, we successfully completed the data ingestion phase, preparing the foundation for the subsequent transformation and cleansing stages within the ETL workflow.



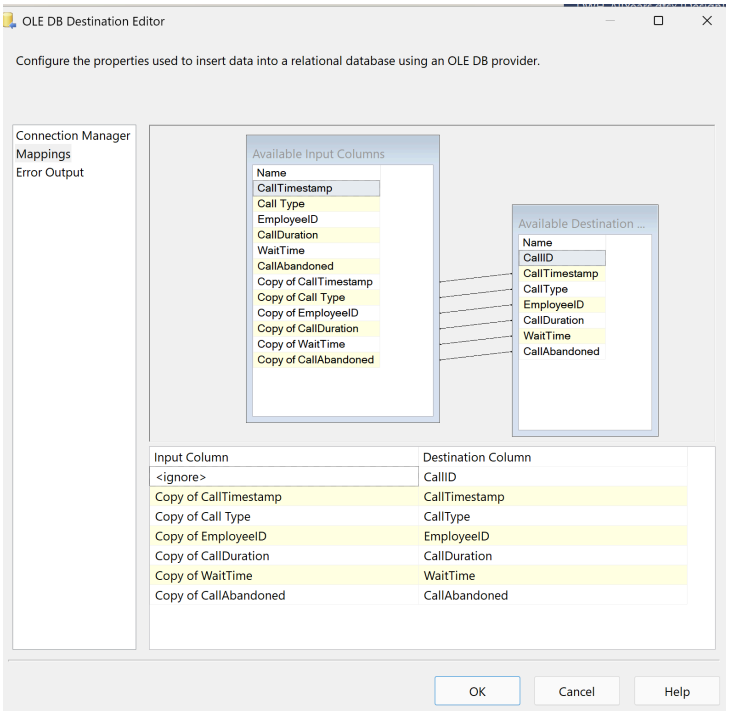

To ensure compatibility and prevent data type conflicts during the loading process, we added a **Data Conversion** transformation between the **Source** and **Destination** components in SSIS. This step was implemented as a precautionary measure to standardize data formats and ensure smooth data flow into the STA database. By converting data types at this stage, we minimized potential errors related to mismatched column formats and ensured that all records were properly validated before being loaded into the staging environment.

| Data Type | Length |
|---|---|
| Unicode string [DT_WSTR] | 50 |
| Unicode string [DT_WSTR] | 50 |
| Unicode string [DT_WSTR] | 50 |
| | |

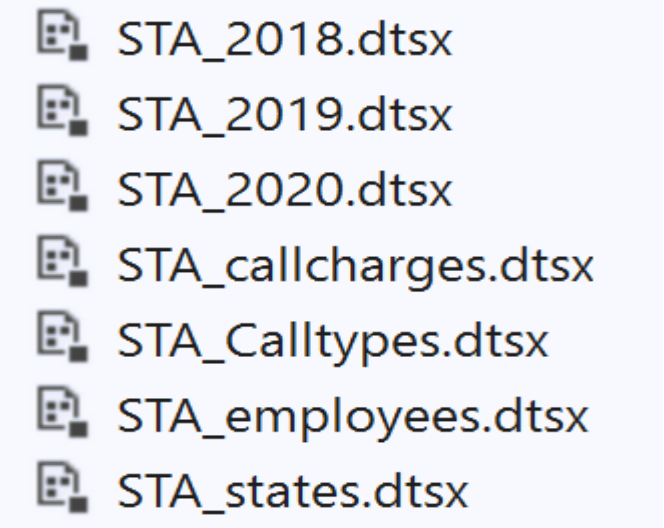



The same approach was applied consistently across all stages of the ETL process, including the loading of the **2018**, **2019**, and **2020** CSV data files. By maintaining this standardized configuration—with Flat File Sources, Data Conversion transformations, and defined Destinations—we ensured uniform data handling, reduced the risk of schema inconsistencies, and streamlined future data integrations.

Using the **Data Conversion** transformation in SSIS, we converted the necessary columns to the **DT_WSTR** data type. This ensured that all text-based data was represented as Unicode strings, aligning with the **NVARCHAR** data type used in the SQL Server **STA** database. The conversion prevented encoding and truncation issues, guaranteeing that special characters and varying text formats from the source CSV files were correctly handled during the data loading process.




After performing the data conversion, we mapped the converted output columns to their corresponding fields in the **STA database** tables. This ensured that each data element from the source files aligned correctly with the predefined schema in the staging area. Proper column mapping was essential to maintain data integrity and consistency across all records loaded through the SSIS workflow.

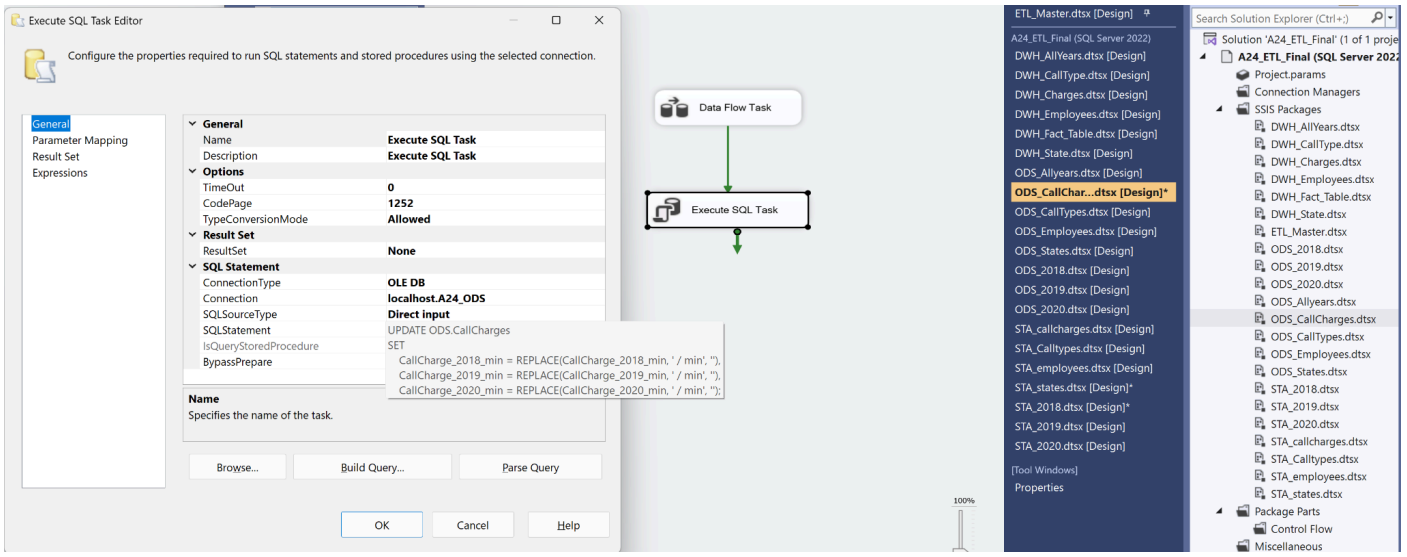



This process was iterated across all the CSV files, ensuring that each dataset was loaded into the **STA database** in a consistent manner. At this stage, no transformations were applied; the focus was solely on importing and standardizing the raw data structure. This approach allowed us to establish a reliable foundation of unaltered data before proceeding to the transformation and cleansing phases in later stages of the ETL process.

# II. Transformation (ODS)

In the **Operational Data Store (ODS)** layer, data undergoes transformation to ensure consistency, accuracy, and readiness for integration into the Data Warehouse. Each table from the **Staging Area (STA)** was processed through **SSIS Data Flow Tasks** and **Execute SQL Tasks**, enabling controlled and structured transformations. Similar to the staging phase, we applied transformations only where necessary—focusing on correcting data types, removing inconsistencies, and standardizing key attributes while preserving the integrity of the original records.

The following example illustrates one such transformation applied to the **CallCharges** table.




In the picture above we used a SQL execute block at the end of the dataflow.
After execution, '1.52 / min' becomes '1.52'

Similarly, we used **SQL script blocks** within SSIS to separate the combined **date and time** fields in the datasets for the years **2018**, **2019**, and **2020**. This transformation ensured that each record contained distinct **Date** and **Time** columns, allowing for more efficient filtering, aggregation, and time-based analysis in later stages of the ETL process. By handling this separation during the

ODS phase, we maintained a standardized temporal structure across all years of call data.

**Direct input**

```
UPDATE ODS.CallsData2018
SET CallDate = CAST(CallTimestamp AS DATE),
    CallTime = CAST(CallTimestamp AS TIME);
```

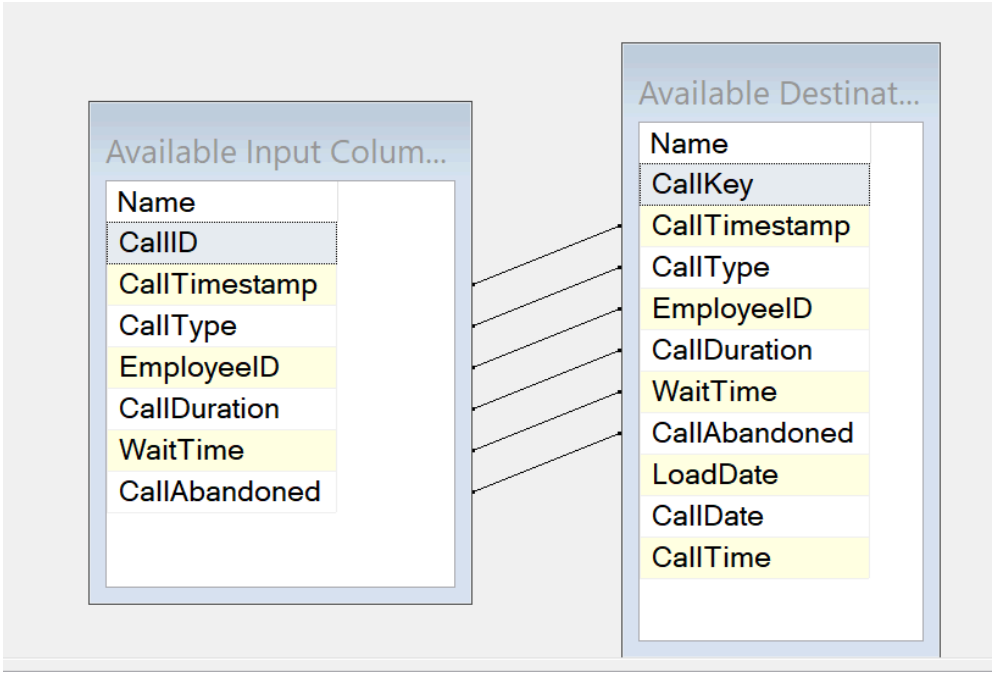
**Direct input**

```
UPDATE ODS.CallsData2019
SET CallDate = CAST(CallTimestamp AS DATE),
    CallTime = CAST(CallTimestamp AS TIME);
```

**Direct input**

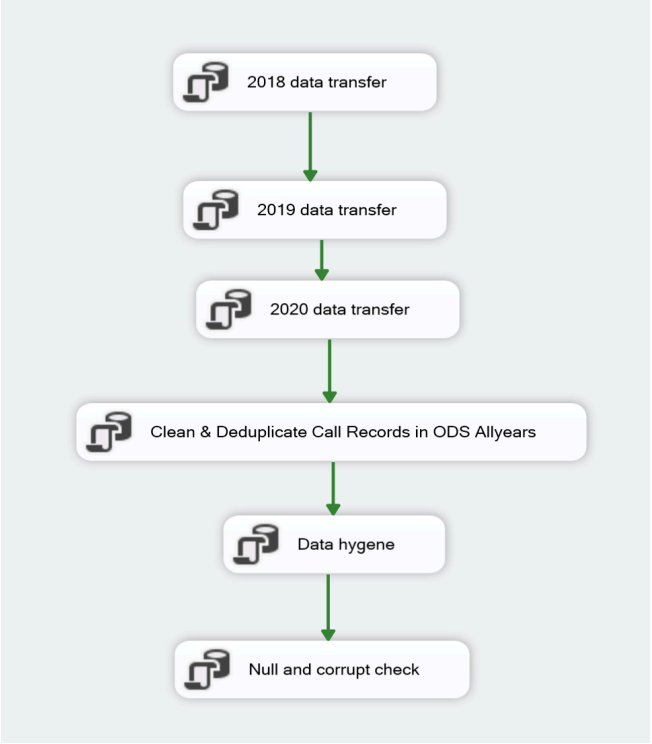
```
UPDATE ODS.CallsData2020
SET CallDate = CAST(CallTimestamp AS DATE),
    CallTime = CAST(CallTimestamp AS TIME);
```

These SQL queries were executed **after the column mapping** was completed in the **Data Flow** stage. This ensured that all data fields were correctly aligned and loaded before any transformation logic was applied. By running the transformation scripts post-mapping, we maintained data integrity and avoided potential mismatches between source columns and target



As a result, these transformations produced the **desired table format**, with properly structured and standardized columns ready for integration into the Data Warehouse. The cleaned and well-organized data in the ODS now served as a reliable intermediary layer, ensuring that only validated and consistent records were passed forward to the DWH for analytical modeling.

| | CallKey | CallTimestamp | CallType | EmployeeID | CallDuration | WaitTime | CallAbandoned | LoadDate | CallDate | CallTime |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2020-02-29 08:39:00.000 | 3 | D411745 | 1289 | 0 | 0 | 2025-11-08 13:45:03.833 | 2020-02-29 | 08:39:00.000 |
| 2 | 2 | 2020-02-13 10:03:00.000 | 2 | X825147 | 968 | 0 | 0 | 2025-11-08 13:45:03.833 | 2020-02-13 | 10:03:00.000 |
| 3 | 3 | 2020-06-08 16:42:00.000 | 3 | E243130 | 300 | 0 | 0 | 2025-11-08 13:45:03.837 | 2020-06-08 | 16:42:00.000 |
| 4 | 4 | 2020-10-02 19:18:00.000 | 3 | B861430 | 1500 | 30 | 0 | 2025-11-08 13:45:03.837 | 2020-10-02 | 19:18:00.000 |
| 5 | 5 | 2020-06-14 15:01:00.000 | 2 | S704443 | 182 | 7 | 0 | 2025-11-08 13:45:03.837 | 2020-06-14 | 15:01:00.000 |
| 6 | 6 | 2020-08-16 19:57:00.000 | 1 | C319958 | 1208 | 0 | 0 | 2025-11-08 13:45:03.840 | 2020-08-16 | 19:57:00.000 |
| 7 | 7 | 2020-03-15 19:46:00.000 | 3 | X409223 | 311 | 9 | 0 | 2025-11-08 13:45:03.840 | 2020-03-15 | 19:46:00.000 |




The most significant transformation performed during the ODS phase was the **consolidation of the 2018, 2019, and 2020 datasets into a single unified table**. This process was implemented using **SQL code blocks** within SSIS to merge records from each year while maintaining data consistency and integrity

```
INSERT INTO ODS.Allyears (
    CallTimestamp,
    CallType,
    EmployeeID,
    CallDuration,
    WaitTime,
    CallAbandoned,
    LoadDate,
    CallDate,
    CallTime
)
SELECT
    CallTimestamp,
    CallType,
    EmployeeID,
    CallDuration,
    WaitTime,
    CallAbandoned,
    LoadDate,
    CallDate,
    CallTime
FROM ODS.CallsData2018;
```

——————————————————>

We used this code in the 2018, 2019, and 2020 to send all data after their date and time were separated.

We created a new ODS file in SQL Studio named ODS.Allyears and they were populated with these SQL command ——————————>

Our 4th code block “Clean & Deduplicate Call Records in ODS Allyears

```
-- Convert duration & wait time to INT
ALTER TABLE ODS.Allyears ALTER COLUMN CallDuration INT;
ALTER TABLE ODS.Allyears ALTER COLUMN WaitTime INT;


ALTER TABLE ODS.AllYears ALTER COLUMN CallAbandoned INT;



DELETE FROM ODS.Allyears
WHERE CallKey NOT IN (
   SELECT MIN(CallKey)
   FROM ODS.Allyears
   GROUP BY EmployeeID, CallDate, CallTime
);


DELETE FROM ODS.Allyears
WHERE CallTimestamp IS NULL;
```

←These statements change the data type of three columns (CallDuration, WaitTime, CallAbandoned) from text (like '123') or another type to integer (INT).

←This deletes duplicate rows from the table ODS.AllYears. It groups records by EmployeeID, CallDate, and CallTime.For each group, it keeps only the first (minimum) CallKey.All other duplicate rows are deleted.

←This deletes rows where the call timestamp is missing.

Our 5th code block “Data hygene”

```
UPDATE ODS.Allyears
SET EmployeeID = LTRIM(RTRIM(EmployeeID)),
   CallType = LTRIM(RTRIM(CallType));
```

← This does more cleaning on the ODS Allyears

RTRIM() → removes spaces from the right end of the text (after the last visible character).

LTRIM() → removes spaces from the left end of the text (before the first visible character).

The combination LTRIM(RTRIM(...)) removes spaces from both sides of the text.

Our 6th code block “Null and Corrupt Check”

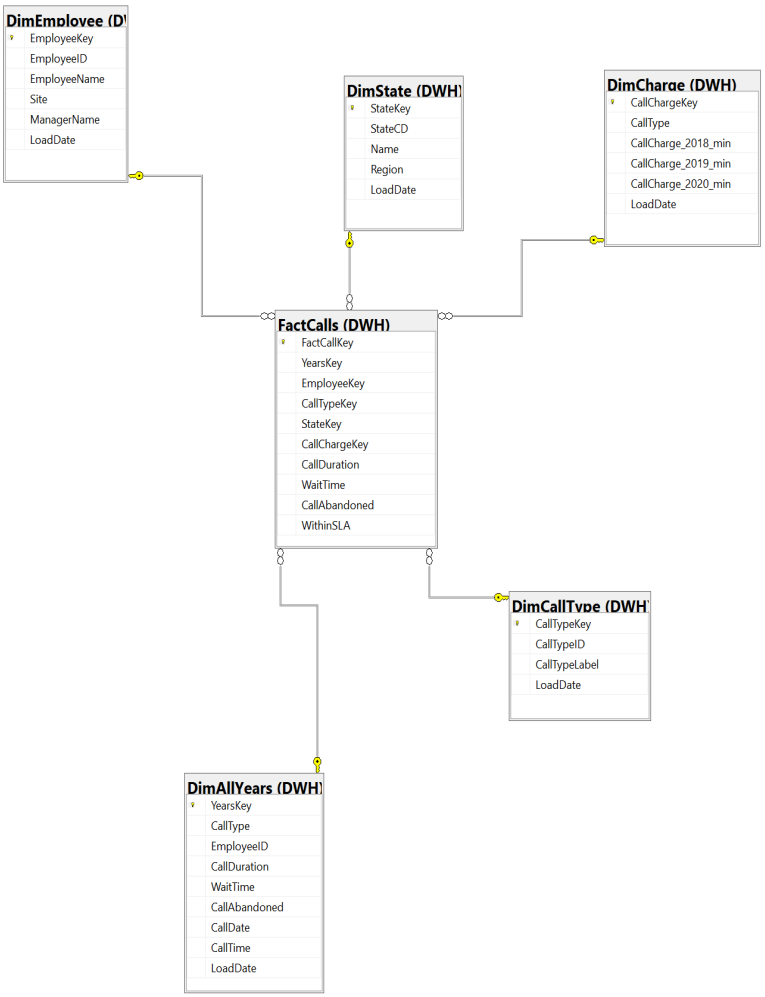
```
SELECT
   COUNT(*) AS Null_EmployeeID FROM ODS.Allyears WHERE EmployeeID IS NULL OR EmployeeID = '';

SELECT
   COUNT(*) AS Null_CallType FROM ODS.Allyears WHERE CallType IS NULL;

SELECT
   COUNT(*) AS NegativeDurations FROM ODS.Allyears WHERE CallDuration < 0 OR WaitTime < 0;
```

Here the first SELECT checks for missing Employee IDs and ensures all call records are linked to an employee.
2nd and 3rd checks for missing Call Types and negative durations or wait times, which are

logically impossible. These are just checks for quality assurance and do not fix the errors. These are just what if scenarios. But since we already did necessary pruning these errors should not arise.

The rest of the data were forwarded as default as we deemed them unnecessary.

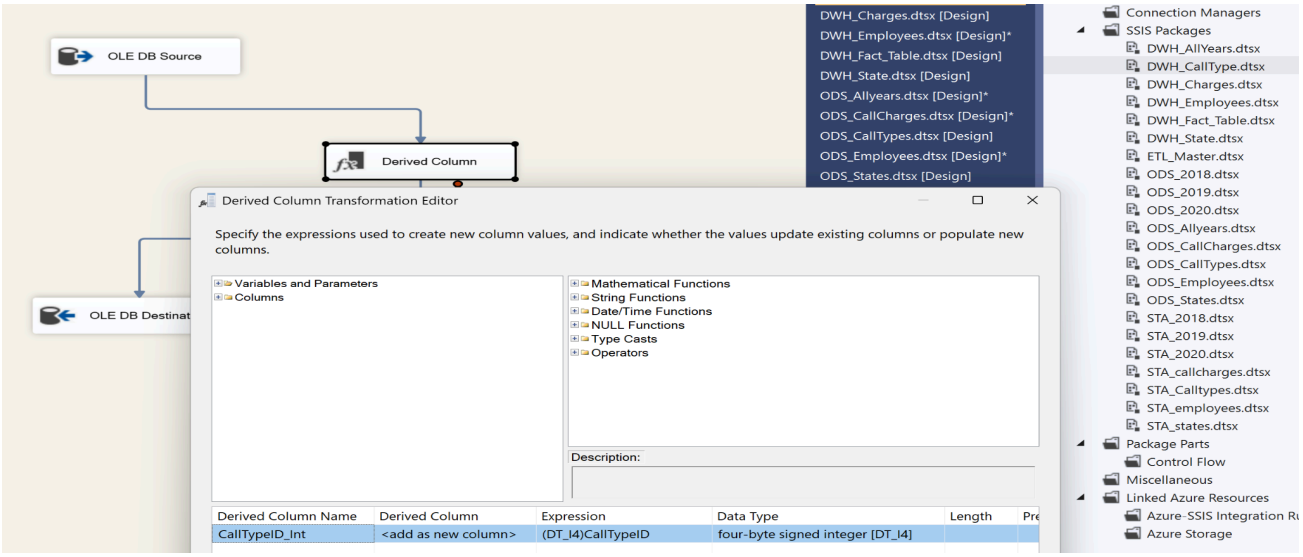
# III. Load (DWH)




The **Data Warehouse (DWH)** layer serves as the final, structured repository optimized for analytical queries and business intelligence reporting. It was designed following a **star schema architecture**, consisting of one central **Fact table** surrounded by multiple **Dimension tables**. Data was loaded in a controlled sequence to maintain referential integrity and respect foreign key dependencies.

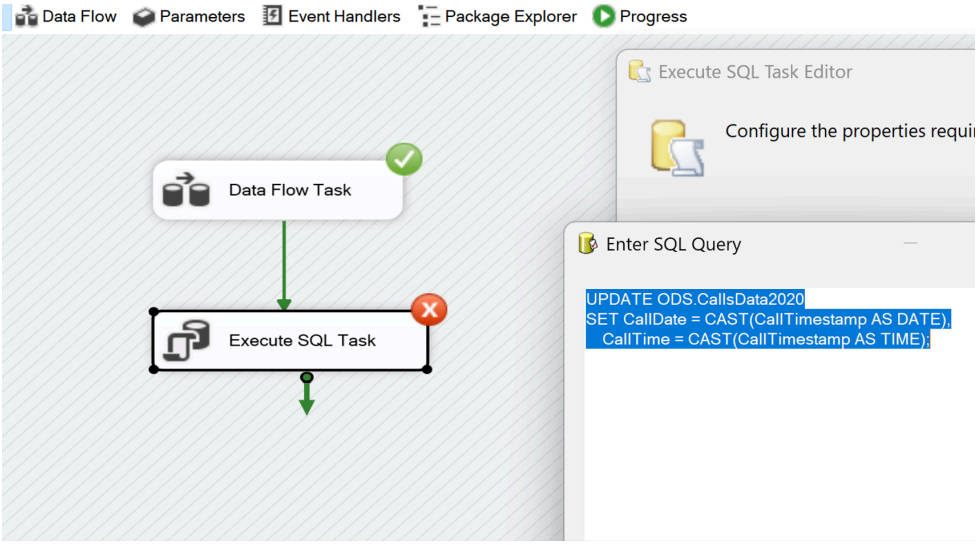
A clear separation was maintained between **Fact** and **Dimension** tables to promote scalability and clarity in data relationships. Dimension tables — such as **Employees**, **CallTypes**, and **Charges** — were populated first, followed by the **FactCalls** table. This systematic loading approach ensured data consistency and simplified analytical modeling in **Power BI**, allowing for efficient visualization, filtering, and performance analysis across multiple dimensions.

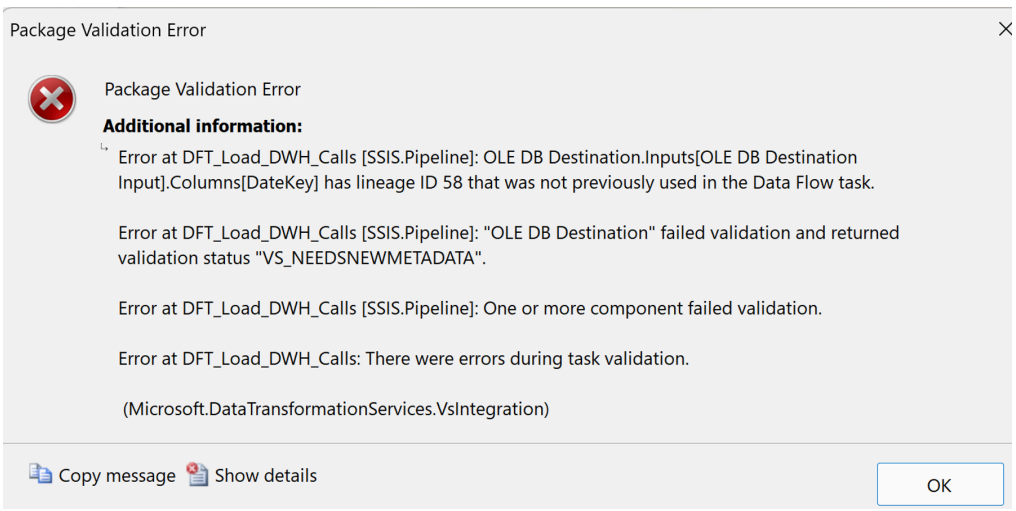
The **DWH phase** was relatively straightforward, as most of the data cleansing and transformation had already been completed in the ODS layer. In this stage, we used **Derived Column transformations** within SSIS wherever minor format adjustments were required. These expressions were applied to standardize column formats, refine calculated fields, and ensure consistency with the schema defined in the Data Warehouse. This lightweight transformation approach helped maintain the DWH's simplicity while ensuring that all data conformed precisely to the analytical model used in **Power BI**.




The figure above illustrates an example of this process applied to the **CallType** data, where derived columns were used to adjust and format values as required.

It is important to note that our ETL development process was **not entirely linear or error-free**. We frequently revisited the **ODS tables and files** to correct data types and resolve inconsistencies encountered during loading and transformation. These iterative refinements were a normal part of ensuring data quality and compatibility across all layers of the ETL pipeline. Each adjustment improved the accuracy and reliability of the final dataset loaded into the Data Warehouse.



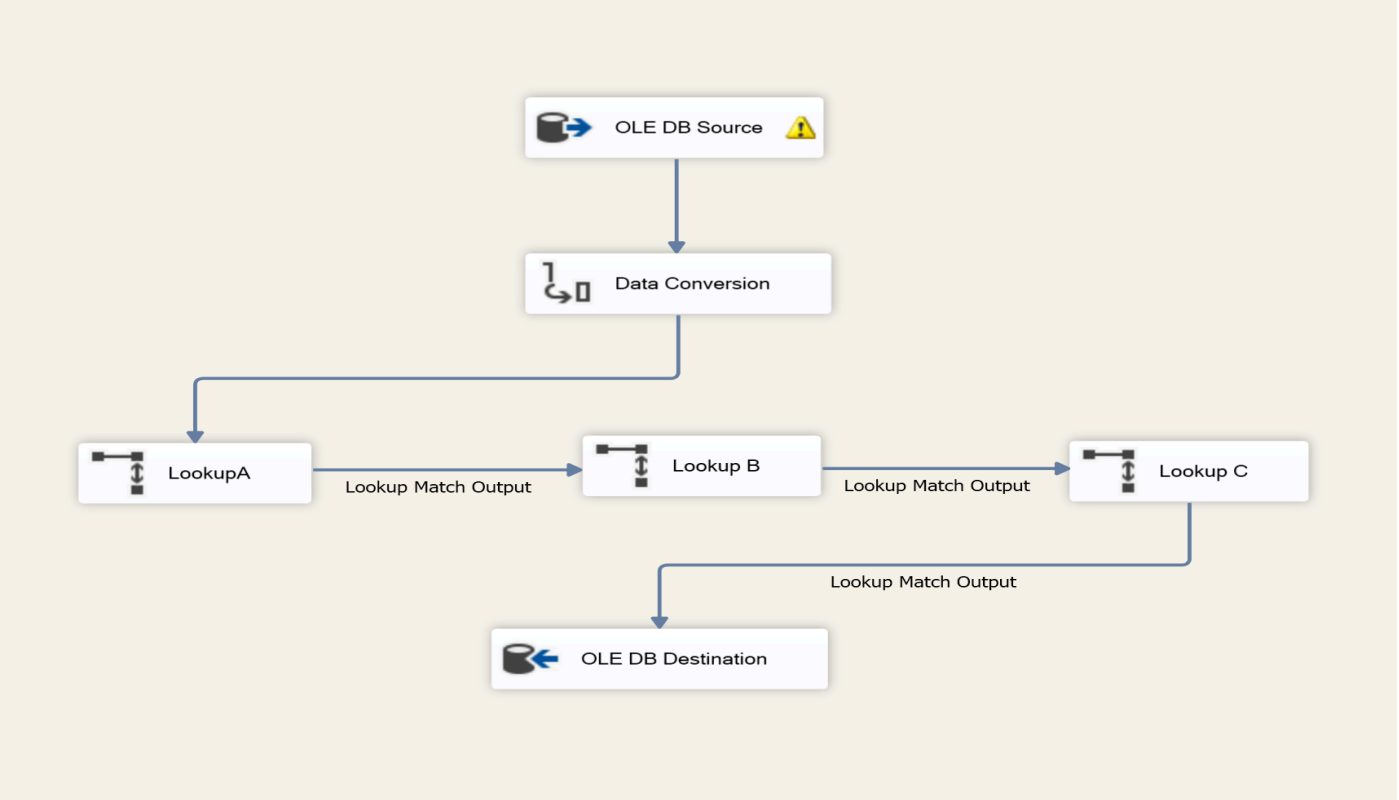



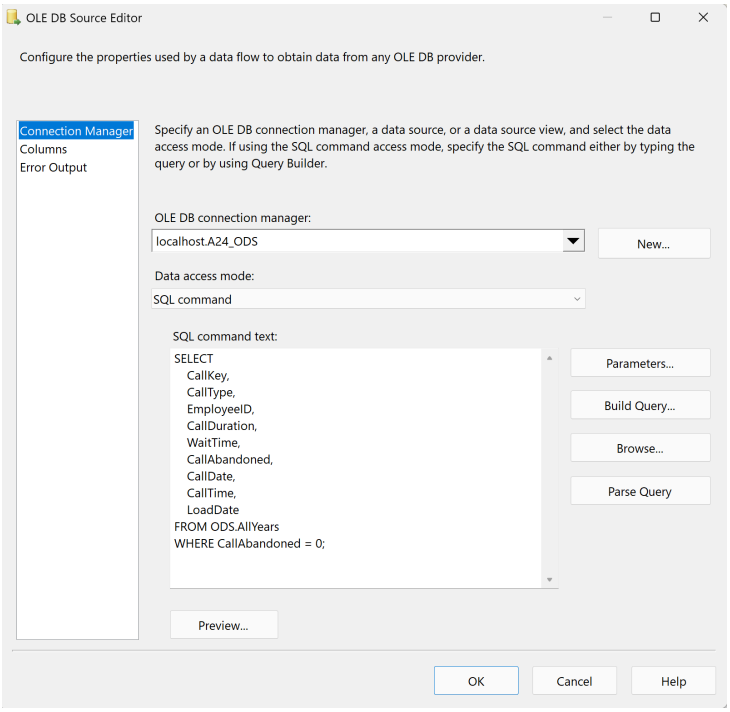
In fact, out of the total **55 hours** spent on this project, approximately **14 hours** were dedicated to **troubleshooting and debugging** various issues encountered throughout the ETL process. These challenges primarily involved resolving data type mismatches, handling null values, and reconfiguring mappings between SSIS components.

To ensure data accuracy before progressing to the **Fact Table** stage, we performed thorough **data filtering and validation using SQL queries** directly to see how our outputs were before we moved into the next section Fact Table.

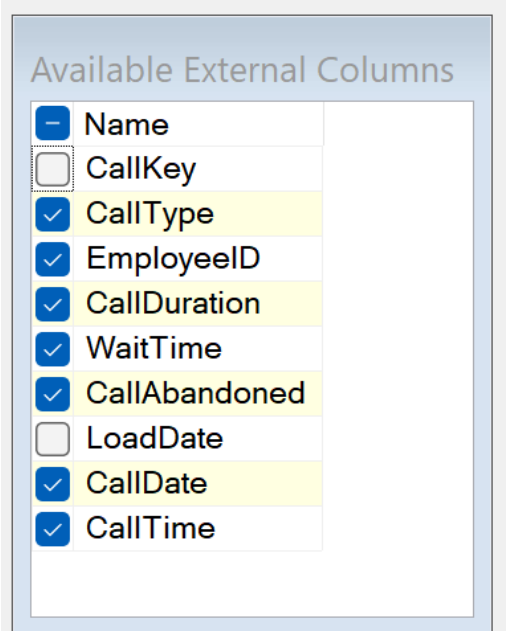
FACT TABLE

This is the Data Flow that created our Fact table



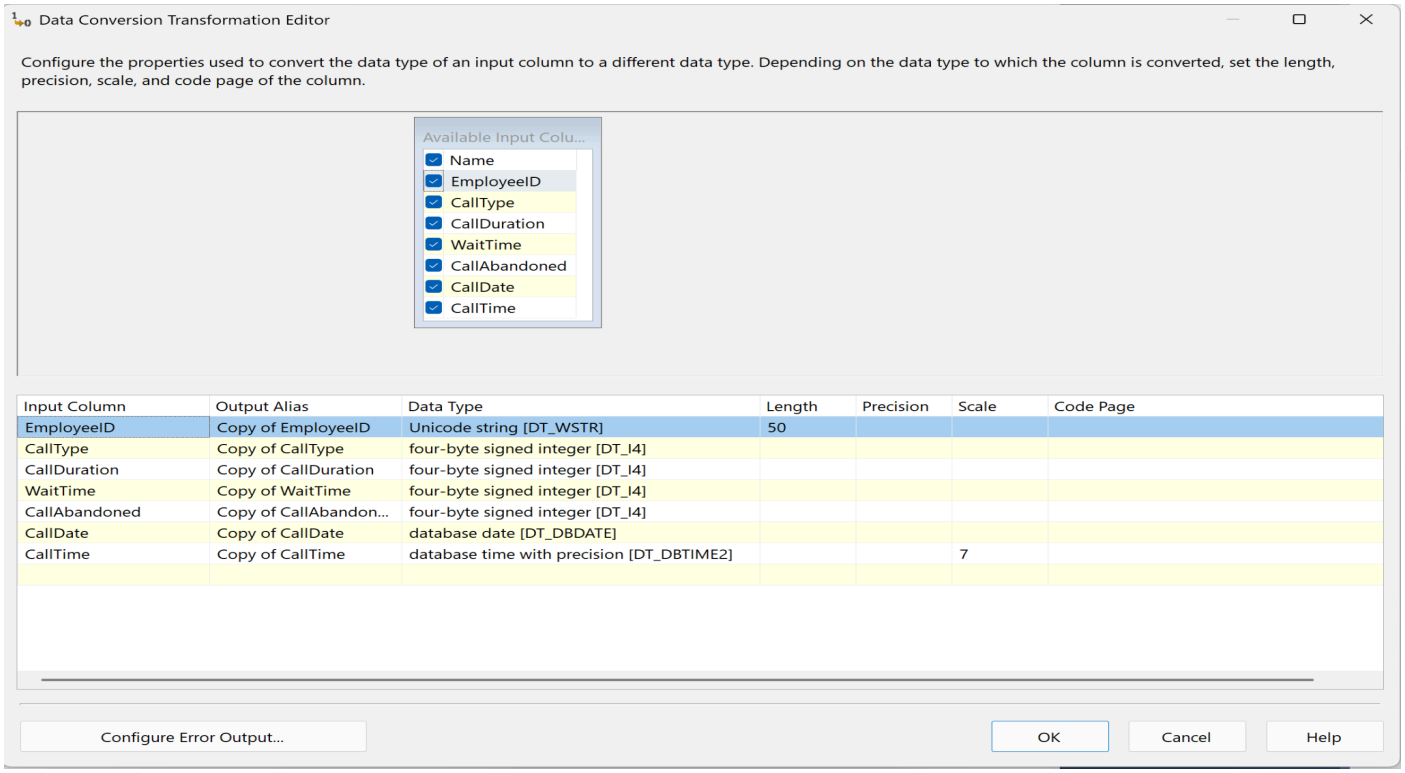




First in the Source we used SQL to retrieve our data.



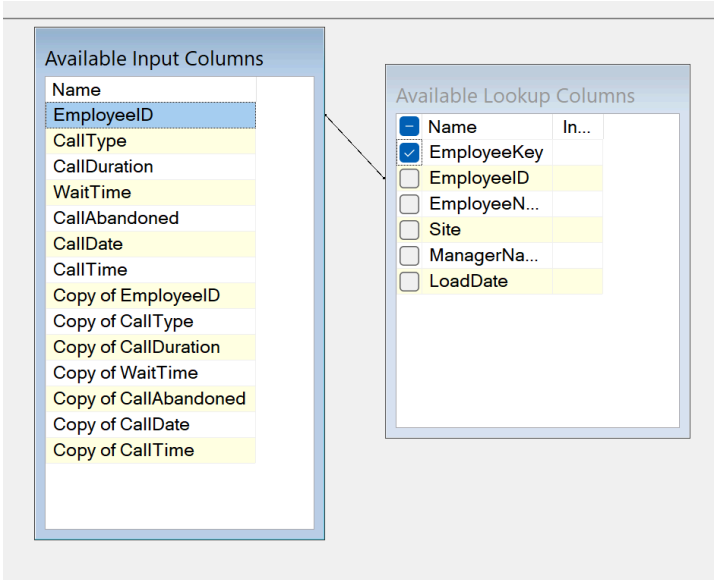

We did not use CallKey as it is automatically assigned by SSIS at the end and LoadDate was removed as we deemed it unnecessary for our Fact table.

Next comes our Data Transformer Editor



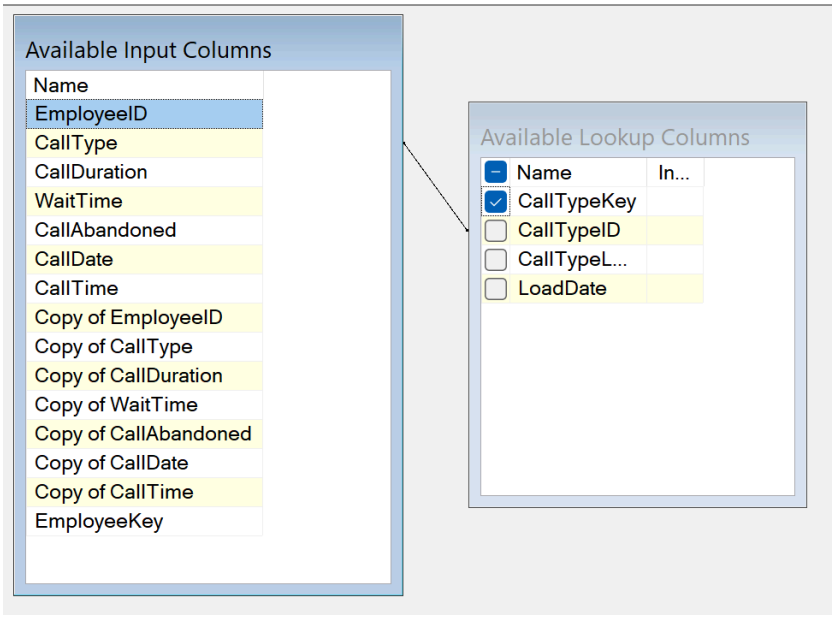

At this stage, we performed the necessary **transformations** to prepare the data for integration into the **Fact Table**. These transformations focused on aligning data formats, ensuring referential integrity with the dimension tables, and applying calculated measures where required to support analytical queries in the Data Warehouse.

Next in Lookup A




The primary purpose of this transformation was to **match each call record with its corresponding employee**, establishing the relational link between the **FactCalls** table and the **Employee Dimension**. This step ensured that every call entry could be accurately associated with the responsible employee, enabling performance tracking, workload analysis, and employee-based reporting within Power BI.

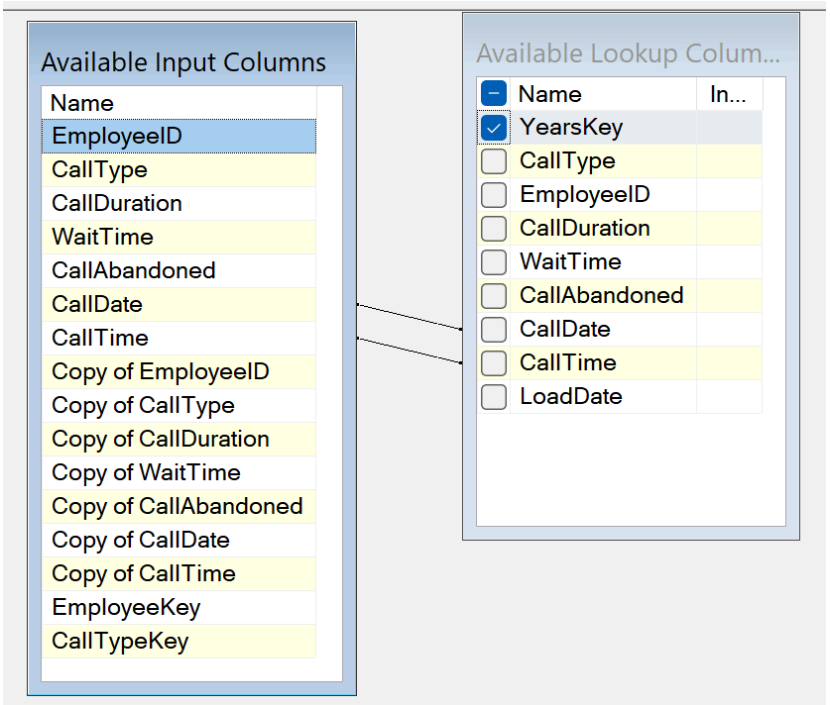
Next in Lookup B




The purpose of this transformation was to **replace the textual call type values** (e.g., *"Billing"*, *"Sales"*, *"Tech Support"*) with their corresponding **CallTypeKey** from the **CallType Dimension**. This step ensured referential integrity between the **FactCalls** table and the **DimCallType** table, allowing the Data Warehouse to use numeric foreign keys instead of textual descriptors. This not only optimized query performance but also maintained consistency across the analytical model used in Power BI.

Next in Lookup C

This step was responsible for **linking the fact data to the DimAllYears table** within the **DWH layer**. By establishing this connection, we ensured that each call record in the **FactCalls** table was correctly associated with its corresponding time dimension entry.

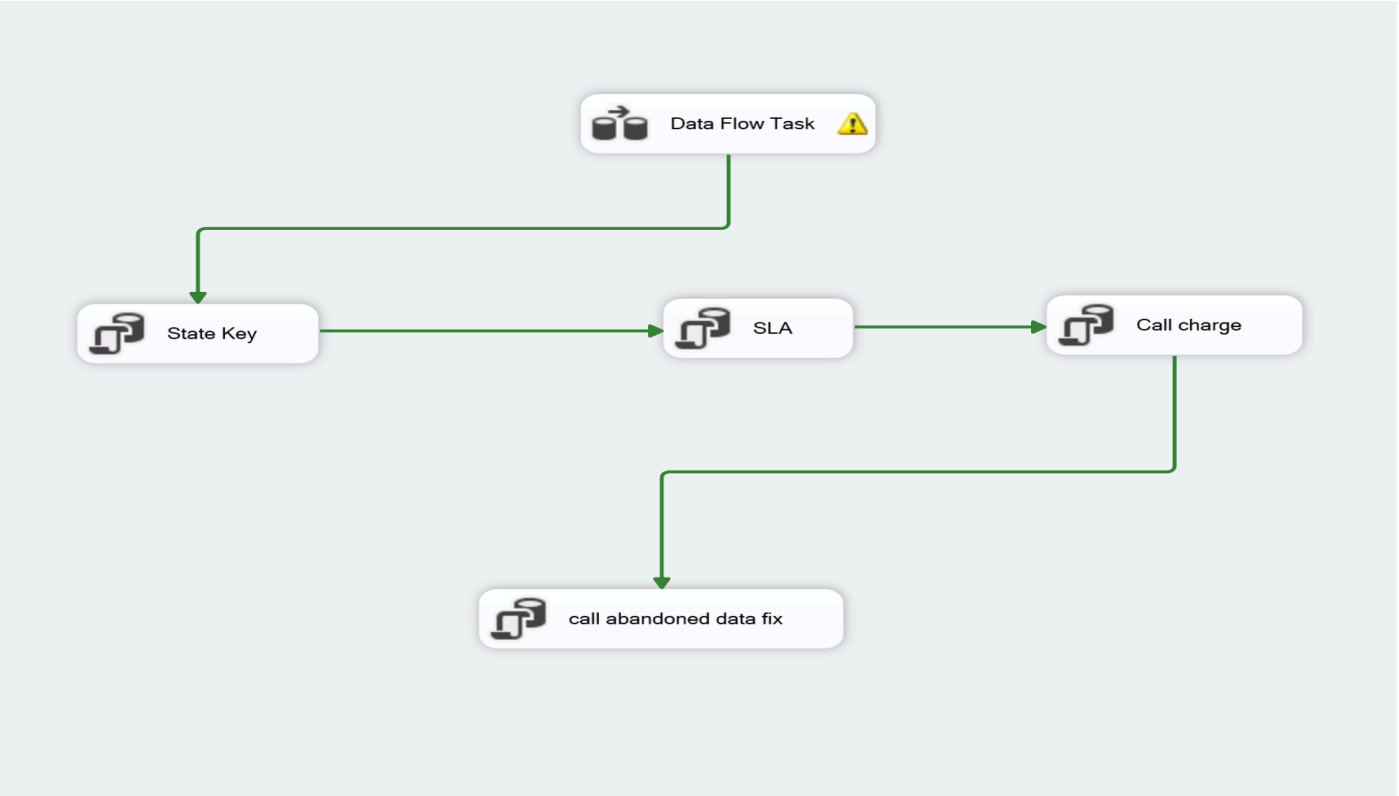



At the conclusion of this stage, our **FactCalls table** was fully structured and populated, reflecting all applied transformations and key mappings.

| | FactCallKey | YearsKey | EmployeeKey | CallTypeKey | StateKey | CallChargeKey | CallDuration | WaitTime | CallAbandoned | WithinSLA | LoadDate |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 1 | 54 | 3 | NULL | NULL | 486 | 2 | 0 | NULL | NULL |
| 2 | 5 | 2 | 1 | 3 | NULL | NULL | 945 | 0 | 0 | NULL | NULL |
| 3 | 6 | 3 | 5 | 1 | NULL | NULL | 379 | 11 | 0 | NULL | NULL |
| 4 | 7 | 4 | 55 | 3 | NULL | NULL | 1044 | 0 | 0 | NULL | NULL |
| 5 | 8 | 5 | 44 | 1 | NULL | NULL | 1357 | 0 | 0 | NULL | NULL |
| 6 | 9 | 6 | 36 | 3 | NULL | NULL | 570 | 23 | 0 | NULL | NULL |
| 7 | 10 | 7 | 35 | 3 | NULL | NULL | 26 | 26 | 0 | NULL | NULL |
| 8 | 11 | 8 | 19 | 2 | NULL | NULL | 8 | 8 | 0 | NULL | NULL |
| 9 | 12 | 9 | 23 | 1 | NULL | NULL | 800 | 0 | 0 | NULL | NULL |
| 10 | 13 | 10 | 15 | 2 | NULL | NULL | 651 | 111 | 0 | NULL | NULL |
| 11 | 14 | 11 | 26 | 2 | NULL | NULL | 229 | 10 | 0 | NULL | NULL |
| 12 | 15 | 12 | 53 | 3 | NULL | NULL | 467 | 23 | 0 | NULL | NULL |
| 13 | 16 | 13 | 16 | 3 | NULL | NULL | 1290 | 7 | 0 | NULL | NULL |
| 14 | 17 | 14 | 40 | 3 | NULL | NULL | 504 | 0 | 0 | NULL | NULL |
| 15 | 18 | 15 | 26 | 3 | NULL | NULL | 1445 | 28 | 0 | NULL | NULL |
| 16 | 19 | 16 | 31 | 2 | NULL | NULL | 1255 | 2 | 0 | NULL | NULL |
| 17 | 20 | 17 | 10 | 3 | NULL | NULL | 1069 | 5 | 0 | NULL | NULL |

Although some **NULL values** were still present at this stage, the majority of the data was successfully loaded and accurately reflected in the **FactCalls** table.

The Null columns exist because we never mapped them to any other Column.
For this we come to the Control flow layer




So after the data goes through the data flow layer. To address the Null column we have used SQL blocks to extract and populate those columns.

**First is the State Key**

```
UPDATE F
SET F.StateKey = S.StateKey
FROM A24_DWH.DWH.FactCalls AS F
INNER JOIN A24_DWH.DWH.DimState AS S
   ON F.EmployeeKey = S.StateKey;
```

Here to find the State Keys we use the Update query.

**2nd the SLA**

```
UPDATE DWH.FactCalls
SET WithinSLA =
  CASE
     WHEN CallAbandoned = 1 THEN 'No'
     WHEN WaitTime <= 30 THEN '1'
     ELSE '0'
  END;
```

This is one of the most important sections of our data.
SLA (Service Level Agreement) represents a critical performance metric for call centers

This code transforms raw operational data into KPI insights by calculating whether each call met the predefined 35-second SLA rule

forming the basis for our SLA % metric in Power BI.

**3rd the Callcharge**

```
UPDATE F
SET F.CallChargeKey = C.CallChargeKey
FROM DWH.FactCalls AS F
INNER JOIN DWH.DimCharge AS C
   ON (CASE F.CallTypeKey
         WHEN 1 THEN 'Sales'
         WHEN 2 THEN 'Billing'
         WHEN 3 THEN 'Tech Support'
      END) = C.CallType;
```

This code ensures that every record in FactCalls is connected to its correct call charge rate.

This was supposed to be our last section but we encountered error with call abandon rate later in Power BI, So we made this quick fix

```
UPDATE DWH
SET DWH.CallAbandoned = ODS.CallAbandoned
FROM DWH.FactCalls AS DWH
INNER JOIN A24_ODS.ODS.AllYears AS ODS
   ON DWH.FactCallKey = ODS.CallKey;
```

This code pulls call Abandoned data directly from the ODS layer. As we found out that we had missed it .

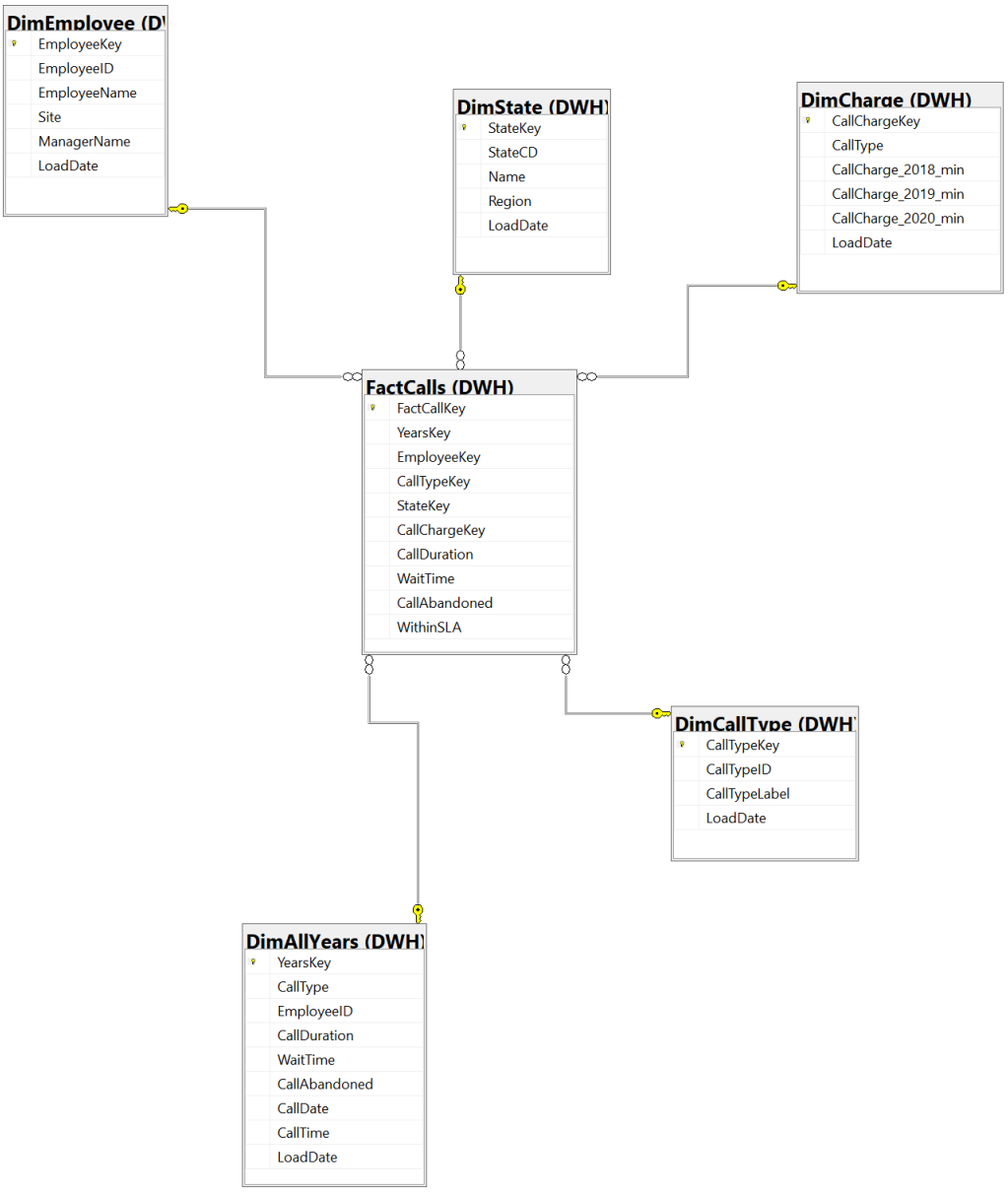
By the end of this stage we had very clean looking data tables.

100 % 1 0

Results Messages

| | FactCallKey | YearsKey | EmployeeKey | CallTypeKey | StateKey | CallChargeKey | CallDuration | WaitTime | CallAbandoned | WithinSLA |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 1 | 54 | 3 | 54 | 7 | 486 | 2 | 0 | 1 |
| 2 | 5 | 2 | 1 | 3 | 1 | 7 | 945 | 0 | 0 | 1 |
| 3 | 6 | 3 | 5 | 1 | 5 | 5 | 379 | 11 | 0 | 1 |
| 4 | 7 | 4 | 55 | 3 | 55 | 7 | 1044 | 0 | 0 | 1 |
| 5 | 8 | 5 | 44 | 1 | 44 | 5 | 1357 | 0 | 0 | 1 |
| 6 | 9 | 6 | 36 | 3 | 36 | 7 | 570 | 23 | 0 | 1 |
| 7 | 10 | 7 | 35 | 3 | 35 | 7 | 26 | 26 | 0 | 1 |
| 8 | 12 | 9 | 23 | 1 | 23 | 5 | 800 | 0 | 0 | 1 |
| 9 | 15 | 12 | 53 | 3 | 53 | 7 | 467 | 23 | 0 | 1 |
| 10 | 16 | 13 | 16 | 3 | 16 | 7 | 1290 | 7 | 0 | 1 |
| 11 | 17 | 14 | 40 | 3 | 40 | 7 | 504 | 0 | 0 | 1 |
| 12 | 18 | 15 | 26 | 3 | 26 | 7 | 1445 | 28 | 0 | 1 |
| 13 | 20 | 17 | 10 | 3 | 10 | 7 | 1069 | 5 | 0 | 1 |
| 14 | 21 | 18 | 36 | 3 | 36 | 7 | 758 | 0 | 0 | 1 |
| 15 | 23 | 20 | 37 | 3 | 37 | 7 | 681 | 10 | 0 | 1 |
| 16 | 25 | 22 | 50 | 3 | 50 | 7 | 252 | 0 | 0 | 1 |
| 17 | 26 | 23 | 45 | 3 | 45 | 7 | 84 | 271 | 0 | 0 |
| 18 | 28 | 25 | 55 | 1 | 55 | 5 | 432 | 6 | 0 | 1 |
| 19 | 30 | 27 | 19 | 3 | 19 | 7 | 1226 | 9 | 0 | 1 |

# Star schema




The **Data Warehouse (DWH)** was modeled using a **star schema** centered around the **FactCalls** table. This design approach simplifies analytical querying and significantly enhances **Power BI** performance by reducing the complexity of joins and query execution time. The **FactCalls** table is linked to several **Dimension tables** — including **DimEmployee**, **DimCallType**, **DimCharge**, **DimState**, and **DimAllYears** — through foreign key relationships.

This schema structure supports a wide range of analytical capabilities, such as **time-series trend analysis**, **Service Level Agreement (SLA) monitoring**, and **cost evaluation by call type**. By combining fact data with descriptive attributes from the dimension tables, the model enables comprehensive performance insights across different operational perspectives.

# ETL Process Overview

## ETL Layer Summary

### 1. STA (Staging Area)

In the **STA layer**, data from raw CSV files was imported into SQL Server using **SSIS Data Flow Tasks**. During this process, initial validation steps ensured that all columns adhered to the expected data types and formats. The staging layer acted as a temporary repository for unprocessed data, providing a controlled environment for subsequent cleaning and transformation operations.

### 2. ODS (Operational Data Store)

The **ODS** functioned as an intermediate data-cleaning and transformation layer. Here, we applied a series of **SQL queries** and **SSIS transformations** to remove duplicates, trim inconsistent entries, and standardize field values. Additional transformations, such as **unpivoting CallCharges** and verifying **NULL values**, ensured data quality and consistency before integration into the Data Warehouse.

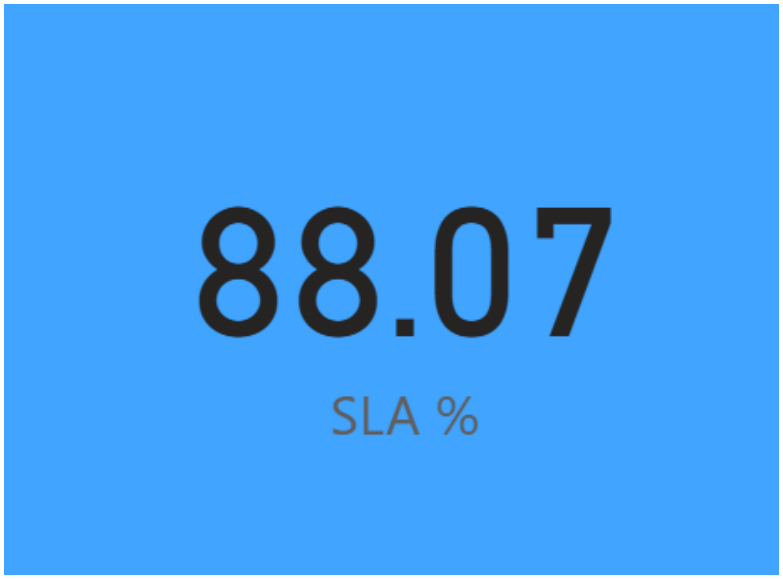
### 3. DWH (Data Warehouse)

In the final **DWH stage**, data was loaded into **Fact and Dimension tables** following a controlled order of dependency. Referential integrity checks were performed to validate key relationships between tables, ensuring accurate and consistent linkages. We also conducted **summary-level validations** to confirm the correctness of aggregated metrics before visualizing the data in **Power BI**.

# Power BI Analysis

## SLA and Performance

In the **Power BI dashboard**, several **Key Performance Indicator (KPI)** cards were developed to present high-level metrics, including **Total Calls**, **Total Duration**, **Average Duration**, **Abandon Rate**, and **SLA %**. To visualize performance trends, a **line chart** was created to display **Abandon Rate variations by year**, enabling year-over-year comparison of customer engagement and operational efficiency. Additionally, a **gauge visual** was implemented to highlight **SLA compliance**, which stood at **88.07%**, providing an immediate snapshot of service reliability.

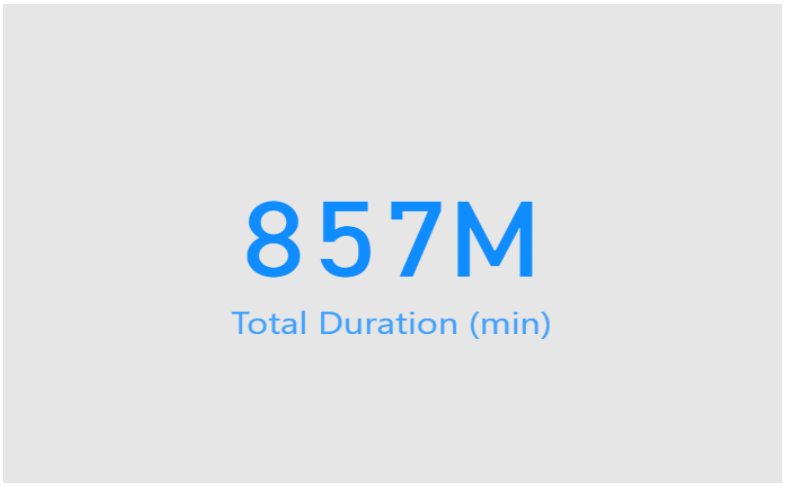



Together, these visuals offer a comprehensive overview of **service quality, call center responsiveness, and performance consistency**, allowing management to make data-driven decisions and identify areas for improvement.

In **Power BI**, we developed a series of visualizations to derive meaningful insights from the Data Warehouse. These visuals allowed us to analyze key performance indicators such as call volume, response times, abandonment trends, and overall SLA compliance.

The most significant finding from our analysis was that the **Service Level Agreement (SLA)** stood at **88.07%**, indicating that approximately **9 out of 10 calls were answered within 35 seconds**. This reflects a strong level of operational efficiency and responsiveness within the call center, demonstrating that the majority of customer inquiries are being handled promptly and effectively.


The analysis revealed a **steady decline in phone call abandonment rates** over the three-year period. Compared to 2018, the abandon rate has significantly decreased, reflecting improved call handling and reduced customer wait times. The **current overall abandonment rate** stands at **0.34%**, representing an aggregate across all three years of data. This demonstrates substantial enhancement in service efficiency and indicates that customer calls are now being successfully connected and addressed.




Total duration of time spent on calls

Total calls received in 2018 - 2020

## Conclusion

This ETL and Data Warehouse project successfully automated data integration and analysis for ServiceSpot’s call center operations. Through SSIS, SQL, and Power BI, we built a complete data pipeline from raw files to interactive dashboards. The project demonstrates the importance of structured data warehousing in achieving accurate business intelligence, efficient reporting, and scalable analytics. And our most important output is the SLA % which proves that the call center is performing well,. 11.93% were outside SLA, meaning customers had to wait longer or the call was abandoned.