**Data Preparation and Generation**

After completing the data model for the database and creating the tables, the next step is to validate the Olist dataset and ensure that it fits our model. In this part, we will check the columns that serve as primary keys to confirm that they are unique and not null. We will also examine the cardinality and verify that the relationships between the tables in the Olist dataset align with those defined in our data model. Any issues that violate the database constraints will be identified and resolved. Afterward, we will generate data for the additional columns we included in our data model that do not exist in the Olist dataset. Finally, we will enrich some columns in the Olist dataset that contain null values and could provide valuable insights for our analysis.

**Data Assessing**

A screenshot of a computer

AI-generated content may be incorrect.In this part, I examined each table in the Olist dataset to identify key information such as the number of rows, the number of columns, the number of duplicates, and the number of null values, along with which columns contain those nulls.

We found that the only table containing duplicates is the **geolocation** table, and there are null values in the **closed\_deals**, **orders**, **order\_reviews**, and **products** tables. In the next steps, we will remove the duplicates and check whether any of the null values exist in candidate primary key columns so we can handle them properly.

After that, we looked at each table to see which columns have missing values. The **closed\_deals** table has several columns with more than 90% null values, such as *lead\_behaviour\_profile*, *has\_company*, and *declared\_product\_catalog\_size*. The **order\_reviews** table has the *review\_comment\_title* column, where about 88% of the values are null, and *review\_comment\_message*, where 58.7% of the values are null.

Also, in this step, we looked at each column that will serve as a primary key and verified that none of them have missing values.

**Primary Key Integrity Check**

A white rectangular object with black text

AI-generated content may be incorrect.In this part, we focus on verifying the **uniqueness** and **completeness** of primary keys. We already confirmed that all primary key columns do not contain missing values. To check their uniqueness, I applied the same logic shown in the image below to each primary key column.

If the result is **0**, it means there are no duplicates, and the column can be used as a primary key.

**Primary Key Validation and Corrections:**

* All candidate primary key columns were unique, except for the geolocation\_zip\_code\_prefix column in the geolocation table, which contained 719,317 duplicate values.
* Since the geolocation, customers, and sellers datasets shared three common columns: geolocation\_zip\_code\_prefix, city, and state, and our data model did not include latitude or longitude, I decided to drop those two columns.
* After that, I checked the customers and sellers datasets to see if they contained any records that did not exist in the geolocation dataset. Any missing records were appended to the geolocation dataset as part of a data enrichment step to ensure there were no missing values when performing joins.
* Then, I removed the duplicate rows from the geolocation dataset so that it contained a unique combination of geolocation\_zip\_code\_prefix, city, and state.
* After that, I created a new primary key column named geo\_id.
* A screenshot of a computer code

  AI-generated content may be incorrect.Finally, I joined the geolocation dataset twice, once with the customers table and once with the sellers table, using the three shared columns to add the geo\_id field. Once this was done, I dropped the geolocation\_zip\_code\_prefix, city, and state columns from both the customers and sellers datasets.

**Checking Cardinality Constraint**

**Orders and Reviews**

In our data model, the relationship between the reviews and orders tables was designed to be one-to-one with total participation from both sides. Therefore, I joined the reviews and orders tables to align with this design.

A screenshot of a computer

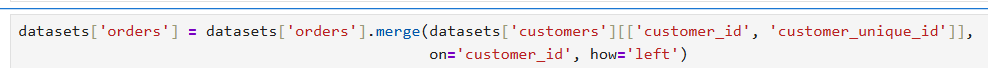
AI-generated content may be incorrect.Before performing the join, I examined the order\_id column in the reviews table to check if it was unique, but I found that there were multiple reviews for the same order. This issue needed to be handled before joining the two tables.

To align with the data model, I kept only the review with the most recent review\_answer\_timestamp, as it represented the customer’s final feedback and reflected the most accurate impression.

**Customers and Orders**

The relationship between the customers table and the orders table in the data model was designed to be one-to-many. However, in the Olist dataset, it was one-to-one because a new record was added for each customer whenever they made an order. There was also another column named customer\_unique\_id that tracked each unique customer.

To fix this, I joined the two tables on the customer\_id column to bring the customer\_unique\_id into the orders table.



After that, I dropped the customer\_id column from both tables and removed duplicates from the customers table to ensure that each customer was represented only once.

A screenshot of a computer code

AI-generated content may be incorrect.

Next, I checked the relationship between the sellers and closed\_deals tables to make sure it was one-to-one, as indicated in the data model, and I confirmed that it was correct.

**Data Generation and Enrichment**

In this part, we added new columns from other tables to align with our data model. We also used a Python package named Faker to generate new data for the columns that were newly created and did not exist in the Olist dataset. In addition, we filled in missing values in existing columns that were considered useful for the analysis.

**Products Table**

A screenshot of a computer code

AI-generated content may be incorrect.First, I obtained the English name of each product category by joining the products table with the product\_category\_name\_translation table.

**Closed\_Deals Table**

We addressed the data integrity issue where some sellers existed in the closed\_deals table but not in the main sellers table. We imputed these missing seller records using data from valid sellers found in the sellers table.

We also filled in the missing values in the declared\_product\_catalog\_size column for each seller by calculating their actual count of unique products sold.

In addition, we filled in the missing values in the declared\_monthly\_revenue column for each seller using their total revenue, which we calculated as the sum of price and freight\_value for all items they sold and multiplied it by a random number to look like a different value than the one we calculated from the Olist dataset.

**Customers Table**

A screenshot of a computer code

AI-generated content may be incorrect.We generated data for the customer\_name and customer\_gender columns to enrich the customers table and make it more suitable for analysis.

**Sellers Table**

We generated data for the seller\_name and seller\_gender columns to complete the missing information and enhance the dataset for future reporting and analysis.

**Employees Table**

We created a new employee’s table based on the sales representatives and sales development representatives’ IDs extracted from the data. We then generated names for those employees and added Facebook page URLs for some of the sales representatives.

A screen shot of a computer code

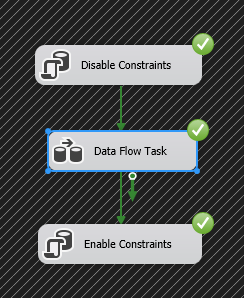
AI-generated content may be incorrect.

**Using Quadratic for Data Enrichment**

In this part we used Quadratic to generate data to support our analysis. We generated customer age, product name, product brand

**Loading the data into the database**

After we completed the database model, we prepared the data and implemented the database. Now, it is time to load the data into the database and to do that I used SQL Server Integration Service (SSIS). The results of the data preparation were 10 Excel files as shown in the image below, one for each table in the database **schema**. I saved the data in Excel format to avoid any problems that may arise because of the data type in SSIS. In SSIS, I created a package to load the data from each file to its intended table in the database.



First, I added an Execute SQL Task Component to disable database constraints and truncate the tables from any data. Then, I added a dataflow which contains **10 separate flows** to load the data from an Excel source to an **OLE DB Destination**. At the end, I added an Execute SQL Task to enable the database constraints.

A screenshot of a computer program

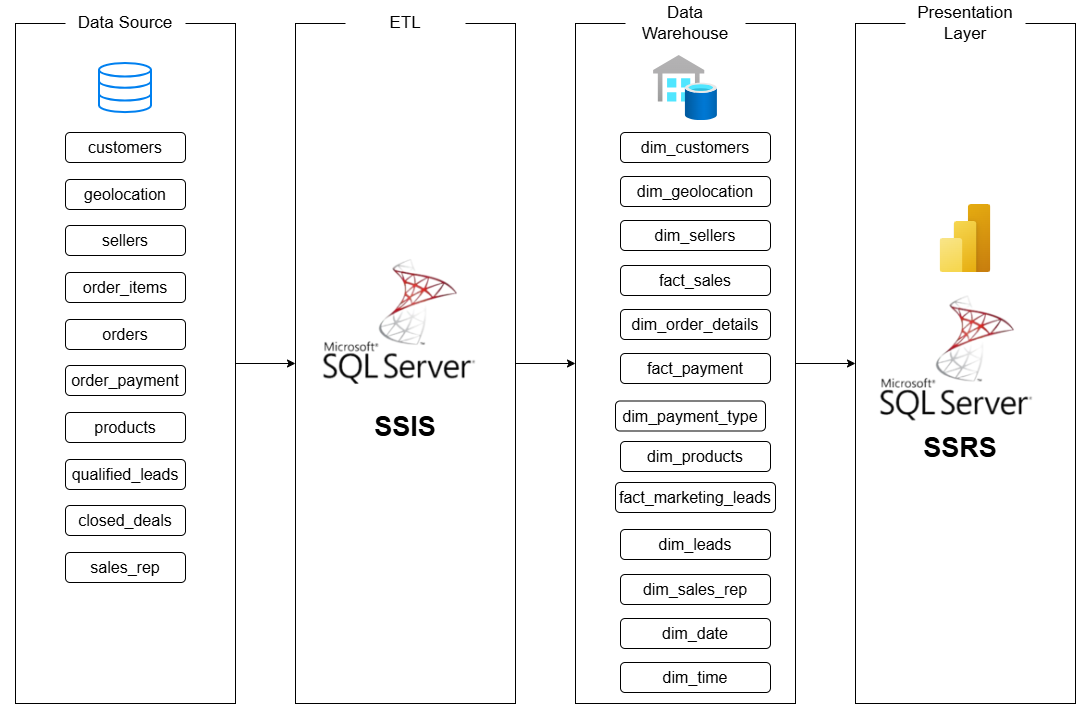
AI-generated content may be incorrect.

**Data Warehouse Implementation**

For this project, we first needed to design the high-level architecture. This plan illustrated how all the different parts of the system were connected, starting from the original raw data and ending with the final reports.

**Data Architecture**

I decided to use a four-layer architecture. The diagram I created shows the complete flow of data through the system. Below is a simple breakdown of what each layer does:

1. **Data Source:** This was the starting point. It represented the OLTP database of the Olist system.
2. **ETL (Extract, Transform, Load):** In this layer, I used SQL Server Integration Services (SSIS) to build packages that extracted, transformed, and loaded the data into the data warehouse.
3. **Data Warehouse:** This was the central database where the transformed data was stored after being processed by SSIS. I designed the schema for this data warehouse using a Galaxy Schema that included three fact tables and several shared dimension tables.
4. **Presentation Layer:** In this final layer, I used SQL Server Reporting Services (SSRS) and Power BI to connect to the data warehouse to build reports and dashboards to present the results clearly and effectively.

**Data Warehouse Schema**

To design the data warehouse, I began by analyzing the Olist database tables to determine which ones serve as **fact tables** (storing measurable business events) and which ones serve as **dimension tables** (storing descriptive context). Based on the business processes, I developed **three data marts**, **Sales**, **Payments**, and **Marketing Leads**, each designed as a star schema or snowflake schema depending on the relationships among dimensions. When combined, they form a **Galaxy Schema** that supports cross-functional analysis across sales, payments, and marketing performance.

**Sales Data Mart**

**Fact Sales**

This fact table stores one record per item sold, enabling analysis of key performance metrics such as item price, freight value, review score, and delivery time.  
It was created by joining the orders and order\_items tables.  
Although the table includes two levels of granularity (order and item), this issue was handled in the presentation layer to ensure accurate measure calculations.

**Dimensions**

**Order Details Dimension**

Tracks descriptive information about orders, such as order status, review title, and review message. This dimension is shared between the Sales Data Mart and the Payments Data Mart, ensuring consistency across analyses.

**Products Dimension**

Contains descriptive details about the products sold by sellers on the Olist e-commerce platform, including product name, brand, and category name.

**Customers Dimension**

Stores descriptive information about customers who placed orders, such as name, gender, age, and location.

**Sellers Dimension**

Holds descriptive details about sellers, including name, gender, and location.This dimension is shared between the Sales Data Mart and the Marketing Data Mart to allow seller-level performance analysis across both domains.

**Geolocation Dimension**

Contains geographical information about states and cities where customers and sellers are located.

**Time Dimension**

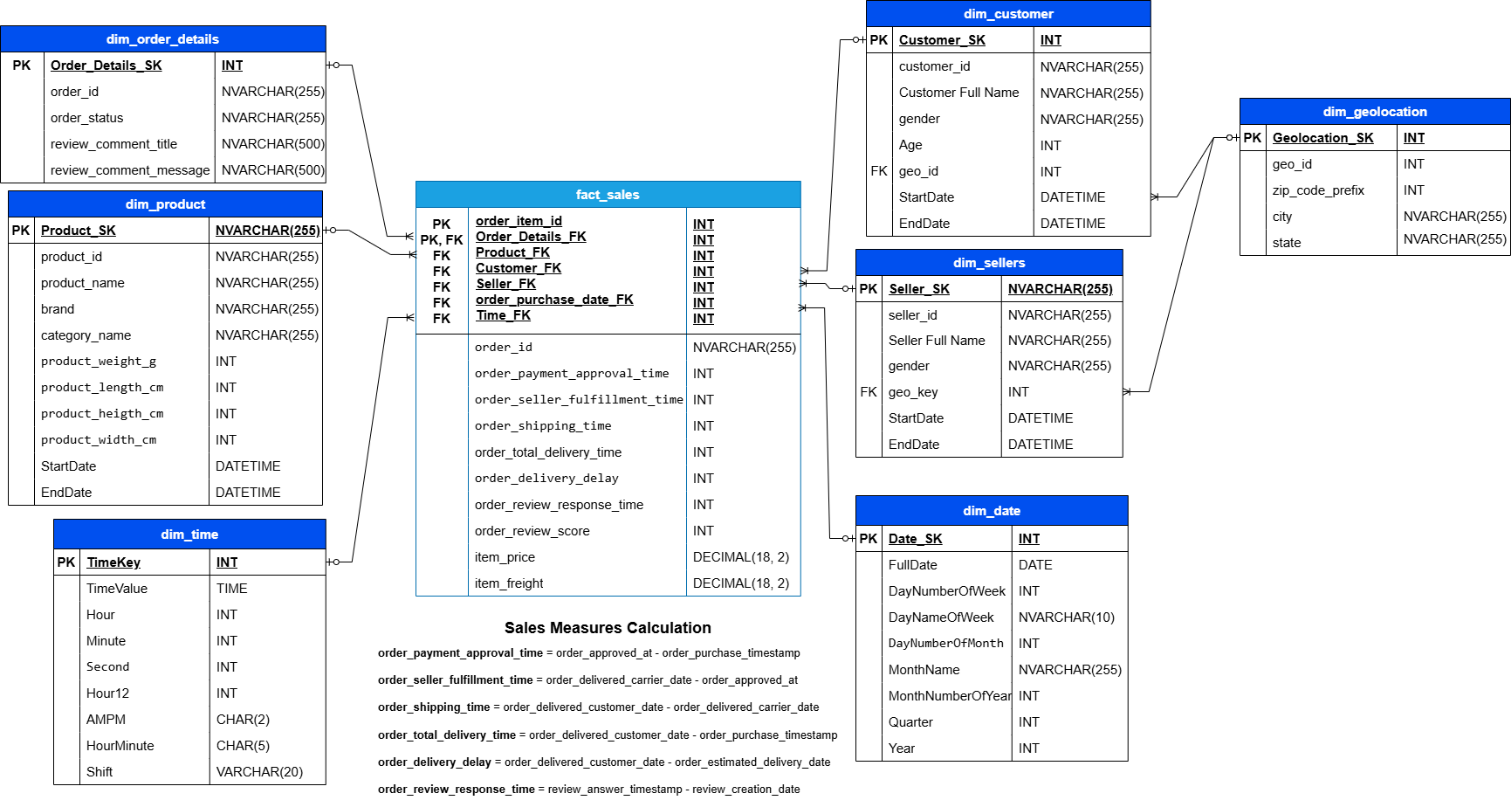
Tracks the time of day when orders are placed, allowing analysis of peak or rush hours throughout the day.

**Date Dimension**

Tracks the date when an order was placed and the date when a deal was closed with a seller.  
It is a shared dimension between the Sales Data Mart and the Marketing Data Mart, enabling time-based analysis across both areas.

**Sales Calculated Measures**

* **order\_payment\_approval\_time** = order\_approved\_at - order\_purchase\_timestamp  
  Measures how long it takes to approve a customer’s payment.
* **order\_seller\_fulfillment\_time** = order\_delivered\_carrier\_date - order\_approved\_at  
  Tracks the time sellers take to prepare and ship an order after approval.
* **order\_shipping\_time** = order\_delivered\_customer\_date - order\_delivered\_carrier\_date  
  Measures the duration of shipping from carrier pickup to customer delivery.
* **order\_total\_delivery\_time** = order\_delivered\_customer\_date - order\_purchase\_timestamp  
  Calculates the total time from order placement to delivery.
* **order\_delivery\_delay** = order\_delivered\_customer\_date - order\_estimated\_delivery\_date  
  Shows how late or early the order was delivered compared to the estimated date.
* **order\_review\_response\_time** = review\_answer\_timestamp - review\_creation\_date  
  Measures the time taken to respond to a customer review.

****The image below shows the Sales Data Mart

**Payment Data Mart**

**Fact Payment**  
This fact table captures each payment transaction, allowing analysis of payment value and installment information.

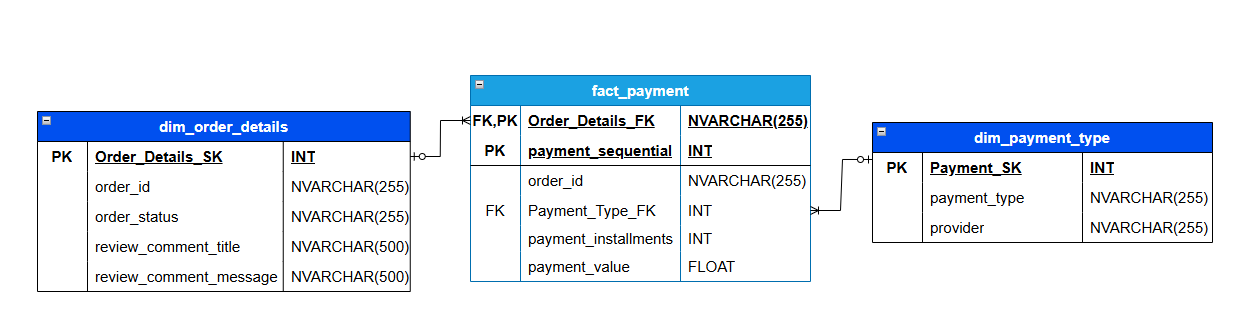
**Dimensions**

**Order Details Dimension**

As previously described, this dimension is shared between the Sales Data Mart and the Payment Data Mart to maintain consistency across both analyses.

**Payment Type Dimension**

Contains descriptive information about the payment methods used by customers, including the payment type and its associated provider.

The image below shows the Payment Data Mart

**Marketing Data Mart**

**Fact Marketing Leads**  
This fact table tracks the marketing funnel, allowing me to measure conversion rates, average time to close a deal, and evaluate the performance of marketing channels and sales representatives.

**Leads Dimension**

Contains descriptive information about the leads such as the lead type, lead behavior profile, business type, business segment, and origin.

**Sales Rep Dimension**

Contains descriptive information about the sales representative responsible and sales development representative such as name, gender, role, Facebook user id.

**Sellers, Geolocation and Date Dimensions**

Common dimensions between the Sales Data Mart and the Marketing Data Mart.

**Marketing Calculated Measures**

* **deal\_close\_time** = *won\_date* − *first\_contact\_date*  
  *Measures how long it takes to close a deal from the first contact date.*
* **lead\_closed** = *IF(mql\_id NOT IN closed\_deals, 0, 1)*  
  *Indicates whether a lead was successfully closed (1) or not (0).*

A screenshot of a computer

AI-generated content may be incorrect.The image below shows the Marketing Data Mart

A blue and white document with text

AI-generated content may be incorrect.After completing the data marts, I connected them using the shared dimensions to form the final schema for the entire data warehouse.

**ETL Process (SSIS)**

We used SQL Server Integration Services (SSIS) to extract, transform, and load the data into the database. I created a separate package for each dimension or group of related dimensions, as well as for each fact table. Then, I built a master package that runs all of them in the correct order. In this master package, the dimensions are loaded first, followed by the facts, since the fact tables rely on the dimensions to retrieve their surrogate keys. I’m going to explain the purpose of each package and what it does.

**ETL Package: Dim\_Geolocation\_Sellers\_Customers.dtsx**

This package is responsible for loading three dimensions: dim\_geolocation, dim\_customers, and dim\_sellers. Since there is a relationship between dim\_geolocation and the other two tables, dim\_geolocation must be loaded first.

A diagram of a software development process

AI-generated content may be incorrect.**Control Flow**

1. **Execute SQL Task:** Runs a SQL script to clear the data in the staging tables as part of the Slowly Changing Dimension (SCD) logic used to load dim\_customers.
2. **Data Flow Task:** Extracts, transforms, and loads the data into dim\_geolocation.
3. **Data Flow Task:** Extracts, transforms, and loads the data into dim\_customers and dim\_sellers.
4. **Execute SQL Task:** Runs a SQL script to insert the data from the staging tables into dim\_customers.

**Data Flow (Load dim\_location)**

A screenshot of a computer

AI-generated content may be incorrect.In this Data Flow, we extracted the data from the **geolocation** table and then used a Lookup component to get the full state name and then we used a **Slowly Changing Dimension (SCD)** component to handle changing attributes. The **state** and **city** attributes were treated as changing attributes so that if their names change, the SCD component will automatically handle the update logic.

**Data Flow (Load dim\_customers and dim\_sellers)**

A screenshot of a computer

AI-generated content may be incorrect.In this Data Flow, we loaded two dimensions. In the **dim\_sellers** table, we treated **geo\_id** as a historical attribute and **seller\_name** as a changing attribute. We used an **SCD** component to handle the logic for tracking these changes.

For the **dim\_customers** table, I used a different approach to handle the different types of attributes. The **customers** table contains around 100,000 rows, and the standard **SCD** component processes data row by row, which makes it slow for large datasets.

Instead, I used a **Lookup** component to compare the incoming data with the existing records in the dimension. Then, I applied a **Conditional Split** component to classify the records based on the type of change (Type 1 or Type 2).

I created two **staging tables** to temporarily store the data for Type 1 and Type 2 attributes, using **OLE DB Destination** components for faster bulk loading instead of the **OLE DB Command**, which processes each row individually.

A screenshot of a computer

AI-generated content may be incorrect.Finally, the new data was loaded into the dimension directly using **OLE DB Destination**, with the data access mode set to **Table or View – Fast Load** to enable batch inserts and improve performance. The updated data will be loaded using an Execute SQL task at the end of the control flow.

**ETL Package: Dim\_Products.dtsx**  
This package is responsible for loading the **dim\_products** table.

**Control Flow**  
It contains a single **Data Flow** task that extracts, transforms, and loads the data from the **products** table into **dim\_products**.

**Data Flow**

A screenshot of a computer

AI-generated content may be incorrect.We used a **Script Component** to convert attribute values from lowercase to title case. In this dimension, the **product\_category** and **brand** attributes were treated as **historical attributes**, while the remaining attributes were considered **changing attributes**. An **SCD** component was used to handle this logic.

**ETL Package: Dim\_Order\_Details.dtsx**  
The purpose of this package is to load the descriptive attributes of the orders from the **orders** table.

**Control Flow**

1. **Execute SQL Task:** Clears the staging table used to store Type 1 attribute changes.
2. **Data Flow Task:** Extracts, transforms, and loads the data into the **dim\_order\_details** table.
3. A diagram of a process

   AI-generated content may be incorrect.**Execute SQL Task:** Updates the **dim\_order\_details** table if there are any changes to existing records.

A screenshot of a computer

AI-generated content may be incorrect.**Data Flow**  
All attributes in this dimension were treated as **changing attributes**. Since the table contains a large number of rows and the **SCD** component would be slow, we implemented the same logic used earlier for the **dim\_customers** package to handle updates efficiently.

**ETL Package: Dim\_Payment\_Type.dtsx**  
The purpose of this package is to load descriptive information related to payment transactions, such as **payment type** and **provider**, from the **order\_payments** table.

**Control Flow**

1. **Execute SQL Task:** Clears the **dim\_payment\_type** table before loading the data, since this dimension is fully reloaded each time.
2. A screenshot of a computer

   AI-generated content may be incorrect.**Data Flow Task:** Extracts, transforms, and loads the data into the **dim\_payment\_type** table.

A screenshot of a computer

AI-generated content may be incorrect.**Data Flow**  
We applied transformations to clean and standardize the column values, such as removing underscores and converting text to **Title Case** for consistency.

**ETL Package: Dim\_Sales\_Rep.dtsx**

The purpose of this package is to load sales representative data from the sales\_rep source table into the dim\_sales\_rep dimension table.

**Control Flow**

The package contains a single **Data Flow Task** responsible for extracting, transforming, and loading the data into the dim\_sales\_rep table.

**Data Flow**

**Derived Column Transformation:**  
A new column was created to combine the first and last names into a single **Full Name** field for better readability and analysis.

**Slowly Changing Dimension (SCD) Component:**

A screenshot of a computer

AI-generated content may be incorrect.The **Name** and **Facebook User ID** were treated as **changing attributes**. The **Role** was treated as a **historical attribute** to preserve history whenever it changes. The **SCD component** was used to implement this logic, ensuring proper handling of both Type 1 and Type 2 changes.

**ETL Package:** Dim\_Leads

The purpose of this package is to load the descriptive attributes for marketing leads into the dim\_leads table.

**Control Flow**

1. **Execute SQL Task:** Clears the dim\_leads table before loading the data.
2. **Data Flow Task:** Extracts, transforms, and loads the data into the dim\_leads table.

**Data Flow**

1. **Derived Column Transformation:** Replaces all NULL values with "N/A".
2. A screenshot of a computer

   AI-generated content may be incorrect.**Script Component:** Converts all text values to **Title Case** instead of lowercase.

**ETL Package: Dim\_Date\_Time**  
The purpose of this package is to create and load the date and time dimension tables used across the data warehouse.

**Control Flow**

* A diagram of a task

  AI-generated content may be incorrect.**Two Execute SQL Task:** Runs a SQL script that generates the dim\_date and dim\_time tables, including all necessary attributes for date and time analysis.

**ETL Package: Fact\_Sales.dtsx**

**Purpose:**  
The purpose of this package is to load the **Fact\_Sales** table by extracting and calculating all necessary measures using a SQL query as the data source.

**Control Flow**

1. **Execute SQL Task:** Clears the existing data from the fact\_sales table before loading new records.
2. **Data Flow Task:** Extracts, transforms, and loads data into the fact\_sales table.

**Data Flow**

* The data flow begins with an **OLE DB Source** that retrieves and calculates all required sales measures using a SQL script.
* Several **Lookup** components are used to fetch the corresponding **foreign keys** for each dimension table by matching business keys.
* A screenshot of a computer

  AI-generated content may be incorrect.After all transformations and lookups are completed, the data is loaded into the fact\_sales table using an **OLE DB Destination** configured for fast batch loading.

**ETL Package: Fact\_Payment.dtsx**

The purpose of this package is to load the **Fact\_Payment** table with all payment-related transactional data.

**Control Flow**

1. **Execute SQL Task:** Clears the existing data from the fact\_payment table before loading new records.
2. **Data Flow Task:** Extracts, transforms, and loads the data into the fact\_payment table.

**Data Flow**

* A **Script Component** is used to apply the same transformations that were performed on the related dimension table columns. This ensures that lookups and key matching function correctly in the following steps.
* A screenshot of a computer

  AI-generated content may be incorrect.Multiple **Lookup Components** are used to retrieve the corresponding **foreign keys** from each dimension table by matching business keys.

**ETL Package: Fact\_Marketing\_Leads.dtsx**

The purpose of this package is to load the **fact\_marketing\_leads** table.

**Control Flow**

1. **Execute SQL Task:** Clears the existing data from the **fact\_marketing\_leads** table before loading new records.
2. **Data Flow Task:** Extracts, transforms, and loads the data into the **fact\_marketing\_leads** table.

**Data Flow**

* A **Script Component** is used to apply the same transformations that were performed on the related dimension table columns. This ensures that lookups and key matching function correctly in the following steps.
* A screenshot of a computer

  AI-generated content may be incorrect.Multiple **Lookup Components** are used to retrieve the corresponding **foreign keys** from each dimension table by matching business keys.

**ETL Package: Master.dtsx**

A screenshot of a computer

AI-generated content may be incorrect.The master package automates the **complete ETL process** by controlling the execution of all dimension and fact packages in the correct sequence. It ensures **referential integrity** by managing the order of loading and by disabling/enabling constraints as needed.

**Control Flow**

**1. Disable Constraints**

This task temporarily disables **foreign key constraints** in the database. It allows dimension and fact tables to be truncated and reloaded without constraint violations.

**2. Sequence Container (Dimensions)**

The purpose is to ensure that all dimensions are fully loaded before any fact tables, since **fact tables depend on dimension surrogate keys**.

**3. Sequence Container 1 (Facts)**

After all dimensions have been successfully loaded, this container executes the packages that load **fact tables**.

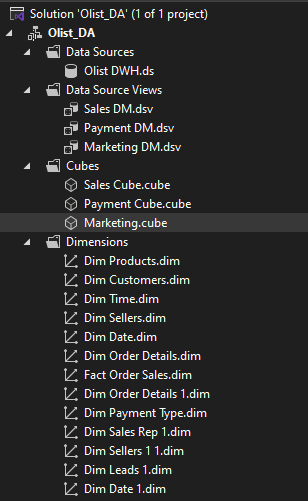
**4. Enable Constraints**

This task re-enables the **foreign key constraints** in the database after all ETL operations are completed.

**Analyzing Data in SSAS**

After successfully loading all the data into the data warehouse, the next step was to perform multidimensional analysis to answer key business questions. To achieve this, I used **SQL Server Analysis Services (SSAS)** to connect to the **Olist Data Warehouse** as the data source.

Within SSAS, I created three views (one for Sales, one for Marketing, and one for Payments) and then built three cubes based on these views. Each cube was designed to support analysis within its specific business domain.



The views serve as an abstraction layer between the data warehouse and the cubes, simplifying data modeling, improving maintainability, and ensuring efficient cube processing.

The cubes were developed to deliver powerful analytical capabilities, enabling deeper insight into sales performance, payment trends, and marketing effectiveness through measures, calculated members, and dimension hierarchies.

**Sales View**

This view brings together all the tables needed to analyze the entire sales process. Its purpose is to answer questions like:

* What are the total sales by product, category, and brand?
* Who are my top customers by region (state and city)?
* Whatt is the average delivery time and review score for different states?

A screenshot of a computer program

AI-generated content may be incorrect.

Since the Fact\_Sales table contains two levels of granularity: item level and order level, it was important to ensure that order-level measures were calculated correctly without duplication.

A computer screen shot of a program

AI-generated content may be incorrect.To address this, I created a Named Query in SSAS that aggregates data at the orderlevel for each customer within a specific time. This query summarizes order-related measures. By doing this, I eliminated duplicate entries and ensured that all order-level calculations and aggregations in the cube are accurate and consistent.

Figure : Fact Order Sales Logical Table Code

**Payment View**

This view is designed for financial analysis. Its purpose is to answer questions about customer payments, such as:

* What is the total payment value by payment type?
* What is the average number of payment installments?A screenshot of a computer

  AI-generated content may be incorrect.

**Marketing View**

This view is focused on the marketing funnel. Its purpose is to track the performance of leads and the sales representatives who manage them. It's designed to answer specific questions, such as:

* What is the conversion rate per origin?
* A screenshot of a computer

  AI-generated content may be incorrect.What is the average time to close a deal per sales rep?

**Sales Cube**

A screenshot of a computer

AI-generated content may be incorrect.This cube was built using Sales View. It combines measures from both the order level and the item level to ensure accurate and flexible analysis. The order-level measures were selected from the logical table created to resolve granularity issues, while the item-level measures, such as Item Freight and Item Price, were sourced directly from the Fact\_Sales table. This design allows the cube to support detailed item-level insights while maintaining accurate aggregated metrics at the order level.

In this cube, I created several calculated measures to support three analytical reports:

**1. Geolocation Report**

* **Description:** A report that aggregates sales data by customer state. It shows the Avg Order Value, Freight %, and Total Revenue for each state.
* **Insights Provided:** This report is for geographical analysis. It helps identify the most valuable states and states with the highest average spending (Avg Order Value). You can also analyze the relationship between Freight % and revenue to see if high shipping costs in certain areas might be impacting sales.

**2. Seller Performance Report**

* **Description:** A report that lists individual sellers and their key performance metrics: No Orders, Freight %, and Total Seller Revenue.
* **Insights Provided:** This report is used to identify top and bottom performers. You can easily sort by Total Seller Revenue or No Orders to find your most valuable sellers. You can also use it to find sellers who may be problematic, for example, a seller with a very high Freight % but low revenue.

**3. Delivery Performance Report**

* **Description:** A report that tracks sales and logistics performance over time. It shows Orders Count, Total Revenue, Late Orders %, and the average time for fulfillment and shipping.
* **Insights Provided:** This report is perfect for identifying seasonal trends in sales and for monitoring logistics efficiency. You can spot problem months with high late orders % and see if delays are caused by sellers or the shipping carriers.

**Calculated Members**

A screenshot of a computer

AI-generated content may be incorrect. The following calculated measures were created in SSAS using **MDX** to support these reports:

|  |  |  |
| --- | --- | --- |
| **Measure** | **Formula** | **Description** |
| **Total Revenue** | [Measures].[Item Freight] + [Measures].[Total Seller Revenue] | Combines product and freight revenue for total sales value. |
| **On-time Orders Count** | SUM(FILTER([Fact Order Sales].[Order Id].[Order Id].MEMBERS, [Measures].[Order Delivery Delay - Fact Order Sales] <= 0), [Measures].[Orders Count]) | Counts all orders delivered on or before the estimated date. |
| **Late Orders Count** | SUM(FILTER([Fact Order Sales].[Order Id].[Order Id].MEMBERS, [Measures].[Order Delivery Delay - Fact Order Sales] > 0), [Measures].[Orders Count]) | Counts all orders delivered after the estimated delivery date. |
| **Late Orders %** | [Measures].[Late Orders Count] / [Measures].[Orders Count] | Calculates the percentage of delayed deliveries. |
| **Avg Delivery Time** | [Measures].[Order Total Delivery Time - Fact Order Sales] / [Measures].[Orders Count] | Measures average total delivery duration per order. |
| **Avg Seller Fulfillment Time** | [Measures].[Order Seller Fulfillment Time - Fact Order Sales] / [Measures].[Orders Count] | Measures average time sellers take to prepare and dispatch orders. |
| **Avg Carrier Shipping Time** | [Measures].[Order Shipping Time - Fact Order Sales] / [Measures].[Orders Count] | Calculates average shipping duration handled by the carrier. |
| **Avg Order Value** | [Measures].[Total Revenue] / [Measures].[No Orders] | Measures average revenue per order. |
| **Freight %** | [Measures].[Item Freight] / [Measures].[Total Revenue] | Indicates how much freight contributes to total revenue. |

**Payment Cube**

A screenshot of a computer

AI-generated content may be incorrect.This cube was built from the Payment View and is designed to support financial transaction analysis. It provides insights into payment patterns, provider performance, and order cancellations.

**Payment Report**

**Description:**  
This report displays the Number of Transactions, Total Revenue, and Number of Canceled Orders, grouped by Payment Provider and filtered by Payment Type.

**Insights Provided:**  
It helps identify the most valuable payment providers for the business and serves as a risk analysis tool by highlighting providers with a high number of canceled orders.

**Calculated Member**

[No Canceled Orders] = IIF(

IsEmpty(([Measures].[No Transactions], [Dim Order Details 1].[Order Status].&[canceled])),

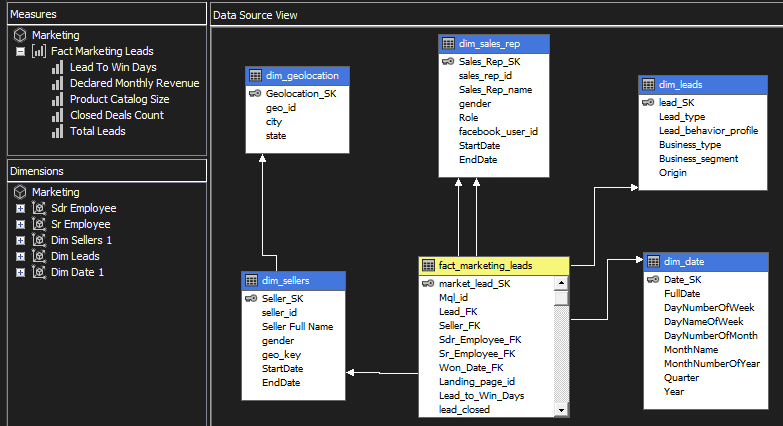
0,

([Measures].[No Transactions], [Dim Order Details 1].[Order Status].&[canceled])

)

This calculated member counts the number of canceled orders by checking transactions with an order status of "canceled" and assigning zero where no such transactions exist.

**Marketing Cube**

This cube was built from Marketing View to support analysis of the marketing funnel and sales representative performance. It provides insights into lead generation, deal conversion, and team efficiency.

We used this cube to create the following reports:

**1. Origin Performance Report**

* **Description:** A report that analyzes the effectiveness of different marketing channels by grouping leads by their Origin. It displays the Closed Deals Count, % Leads (Origin), Conversion Rate, and Avg Deal Close Time for each channel.
* **Insights Provided:** This report is essential for marketing strategy. It shows which channels generate the most closed deals and which channels have the highest Conversion Rate It also identifies which channels produce the *fastest* deals.

**2. Sales Rep Performance Report**

* **Description:** This report breaks down performance by individual sales representative. It shows Sales Rep Name, Role, Gender, Closed Deals Count, and Avg Deal Close Time.
* **Insights Provided:** This report is used to identify top-performing sales reps, both by the volume of deals they close and their efficiency.

**Calculated Members**

|  |  |  |
| --- | --- | --- |
| **Measure Name** | **Description** | **Formula (MDX Expression)** |
| **Avg Deal Close Time** | Calculates the average number of days it takes to close a deal from the time a lead is created. | [Measures].[Lead To Win Days] / [Measures].[Closed Deals Count] |
| **Conversion Rate** | Calculates the percentage of leads that were successfully converted into closed deals. | [Measures].[Closed Deals Count] / [Measures].[Total Leads] |
| **% Leads (Origin)** | Calculates the proportion of leads from each marketing origin compared to total leads. | [Measures].[Total Leads] / ([Measures].[Total Leads], [Dim Leads].[Origin].[All]) |

A screenshot of a computer

AI-generated content may be incorrect.

**Reporting in SSRS**

After we completed the analysis and prepared the necessary measures for six reports. It is time to create the reports in SSRS.

**Geolocation Report**

A screenshot of a computer

AI-generated content may be incorrect.A report that aggregates sales data by customer state. It shows the Avg Order Value, Freight %, and Total Revenue for each state. It gives the user the ability to filter by year and month.

**Insights**

**State with the Top Average Order Value:**São Paulo has the highest average order value of 59.57, which is nearly three times higher than the next state.

**State with the Lowest Freight %:**São Paulo also records the lowest freight percentage at 12.14%.

**State with the Top Total Revenue:**São Paulo leads in total revenue, generating BRL 5,923,525.

**State with the Top Late Orders %:**Alagoas has the highest percentage of late orders at 20.63%.

**Seller Performance Report**

A report that lists individual sellers and their key performance metrics: No Orders, Freight %, and Total Seller Revenue.

A screenshot of a computer

AI-generated content may be incorrect.

**Insights**

**Seller with the Highest Number of Items Sold:**

Sharon Walton has the highest number of items sold, totaling 2,033 items.

**Seller with the Highest Number of Orders:**

Sharon Walton also holds the highest number of orders, with 1,854 orders.

**Seller with the Highest Freight Percentage:**

Eric Gonzalez sold 2 items, with freight costs representing 77% of the total cost.

**Seller with the Highest Revenue:**

Denise Dudley achieved the highest total revenue, amounting to BRL 229,473.

**Delivery Performance Report**

A report that tracks sales and logistics performance over time. It shows Orders Count, Total Revenue, Late Orders %, and the average time for fulfillment and shipping.

A screenshot of a computer

AI-generated content may be incorrect.

**Insights**

From this report, we can gain insights into various business aspects such as delivery performance, revenue trends, and order volume over time.

In 2016, there were only 312 orders, indicating that this was the year the company began operations. By 2017, the number of orders increased significantly to 44,579, with 5.5% of them being late. The average seller fulfillment time was 2.85 days, and the average carrier shipping time was 9.33 days.

By drilling down to the quarter and month levels, we can see that in February 2017, there were 1,733 orders placed, with 2.83% of them being late.

When observing the trend across months, August shows the highest number of orders overall, while the period between September and December experiences a noticeable decline in order volume compared to other months.

**Payment Report**

A screenshot of a computer

AI-generated content may be incorrect.A report that displays the Number of Transactions, Total Revenue, and Number of Canceled Orders, grouped by Payment Provider and filtered by Payment Type.

**Insights**

The Central Bank of Spain recorded the highest number of transactions, totaling 19,784, with a total payment value of BRL 2,869,361 and 95 canceled orders. The highest number of canceled orders was associated with Olist as a provider, suggesting that these cancellations were connected to vouchers issued by Olist. As shown in the chart, the number of undelivered orders is very low, indicating strong delivery overall.

**Marketing Channel Performance Report**

A report that analyzes the effectiveness of different marketing channels by grouping leads by their Origin. It displays the Closed Deals Count, % Leads (Origin), Conversion Rate, and Avg Deal Close Time for each channel.

A screenshot of a computer

AI-generated content may be incorrect.

I**nsights**

* The Unknown marketing channel achieved the highest conversion rate at 16.29%, followed by Paid Search with a 12.3% conversion rate.
* Organic Search generated the highest number of closed deals and maintained a conversion rate of 11.80%, representing the largest share of leads at 28.7%.
* Leads originating from Social channels took the longest average time to close, averaging 61 days. Most closed deals were associated with the Reseller business type.
* Among lead types, the “Other” category had the longest average close time, while Online Top leads had the shortest average deal close time.

**Sales Rep Performance Report**

A report breaks down performance by individual sales representatives. It shows Sales Rep Name, Role, Gender, Closed Deals Count, and Avg Deal Close Time. We can filter by Business Segment and Lead Type.

A screenshot of a computer

AI-generated content may be incorrect.

**Insights**

* Randy Hunt closed the highest number of deals, totaling 133, with an average close time of 30 days.
* Joseph Pacheco closed 74 deals with an average close time of 25 days, meaning he closed deals faster than Randy Hunt and Sara Mack.
* Employees who served as both Sales Representatives and Sales Development Representatives tended to take longer to close deals on average compared to those who worked solely as Sales Representatives.