BLUE SKY ONLINE RETAILER - CASE STUDY

BLUE SKY ONLINE RETAILER -CASE STUDY

INTRODUCTION:

According to the given case study, Blue Sky Online is a consumer electronics retailer that offers more than five hundred brands. The company conducts online business across Europe and the UK. They have multiple selling channels like Amazon, eBay, Tesco, and their website. Blue Sky has a huge customer base. Due to the competitive nature of the electronics retail industry, businesses in this industry are facing multiple challenges like pricing, targeting of customers, customer satisfaction, and so on. Blue Sky Company is facing an issue with accessing the customer base as some valuable information is missing due to minimal maintenance. Therefore, the company requires the analyst to analyze and manage the customer data to improve the business and relationship with its customers, to promote customer retention, and to improve their sales. The following business requirements have been identified:

1. Maintain customer database.
2. Analyze purchase or transaction history based on demographic data such as age group, income group, marital status, and postcode/City/Country.
3. Identify customer preferences in terms of selling channels/ payment payments/.
4. Identify the best and most profitable customers at different periods (month, quarter, and year).

According to the brief, this project aims to:

1.1 Identify the grain detail of the fact table and explain in brief the hierarchies.

1.2 Identify the dimensions of the central fact table.

1.3 Populate the fact table and dimension table.

1.4 Create three graphical representations of simple star schemas.

1. Analysis and design of dimensional data model:

In this analysis and design phase, we are required to understand the given data and produce simple star schemas.

1.1 Identify and define the grain of a central fact table, to represent the primary business activity. Explain the hierarchies if there are any.

To effectively design a dimensional data model for Blue Sky's Online data warehouse, the team has collaboratively identified and defined critical elements relating to the central fact table(s) and hierarchies.

Central Fact Table(s) / Grain Detail:

We first identified that we needed three fact tables to meet the three defined business requirements of Blue Sky Online. Each fact table would need to have its granularity – the level of detail in which the data is stored. The granularity decides the grain level and a grain refers to the specific unit of analysis for a fact table within the dimensional data model.

We first looked at the second business requirement. To meet this, we decided to focus on customer transactions, specifically individual customer sales transactions named. We named this the 'CustomerSalesTransactions' fact. This central fact table captures the most granular details at the atomic level, allowing for subsequent aggregation or grouping.

The agreed-upon grain is set to the most granular level, each row captures an individual transaction made by a customer. This ensures a comprehensive representation of the primary business activity.

For the third business requirement, we identified the need for a fact table that captures customer preferences for selling channels and payment – we called this the 'CustomerPreferences' fact. Again, the grain level was set on the customer level. Each row represents a customer transaction, including information on the selling channel and payment type. The individual rows can then be aggregated by grouping and counting to identify the most popular selling channels and payment methods.

Lastly, for the fourth business requirement, we identified the usefulness of a period snapshot fact table that can capture the profit per customer aggregated over a set period. This will allow for quicker and easier management reports instead of having to write repetitive queries every (by month, quarter, year). We call this the 'CustomerProfitability' fact.

For this table, the grain level is the period (by customer), and each row represents a periodic snapshot of the profitability per customer. For the requirement, we could pick 'Month' as the grain. However, it could also be useful to consider 'daily' as the lowest grain, should business requirements change, and it becomes necessary to find the most profitable customer on a given day or week. It is then possible to aggregate the report into quarters and years if needed.

Hierarchies:

Hierarchies are of paramount importance in the context of dimensional data modeling. They play a crucial role in structuring and organizing data for efficient querying and reporting within data warehousing systems. These hierarchies serve as the structural framework that represents

interrelationships and pathways to dive deep into various attributes and data levels. They empower users to effectively explore and analyze data, enhancing user understanding and overall usability and utilization of the data model.

There are a few hierarchies that we should be aware of when deciding the grain of the fact table. Dates are a fixed depth hierarchy – a series of many-to-one relationships – with agreedupon hierarchy level names, such as Year -> Quarter -> Month -> Day. The level we chose needs to fit the grain of the fact(s). in the case of 'CustomerSalesTransactions' and 'CustomerPreferences' the grain is each customer purchase, and this is captured on the daily hierarchy. We can later use the different hierarchies to 'drill up' to higher levels. Note that if a customer submits more than one order per day, it might be useful to consider including more identifiable attributes such as invoice number or timestamp (the data source does not include a timestamp).

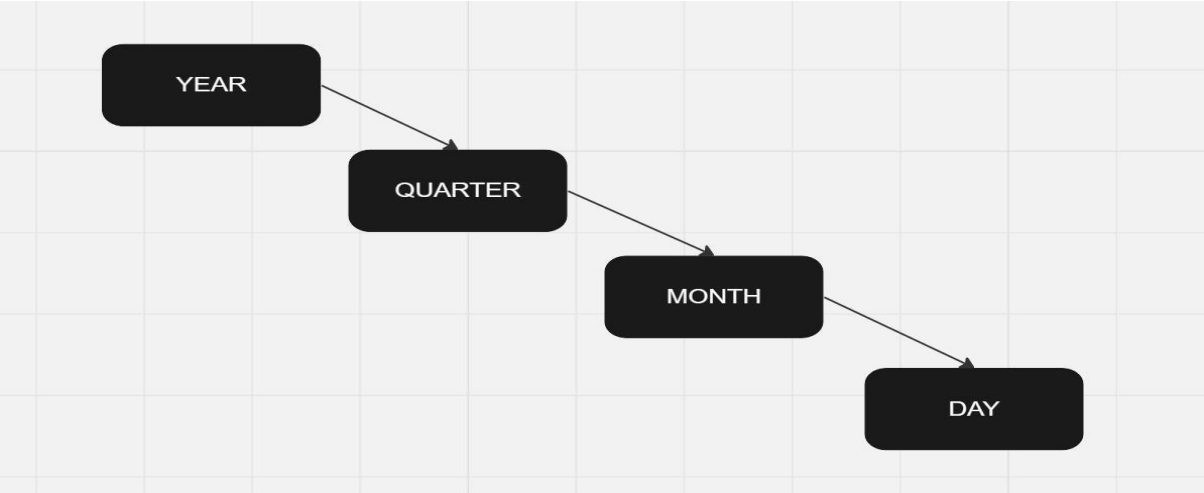


Figure 1: Time Hierarchy

The source data also includes information on the geographic location of the customer. Geographic hierarchies are considered *slighter ragged* hierarchies because they have variable depth. Our data source only includes geographic information as part of customer data, and the hierarchies are city ->postcode. As Blue Sky Online's transactions happen online, the customer's geographic location will not be a part of the fact table but instead used as a description of the customer for filtering and grouping.

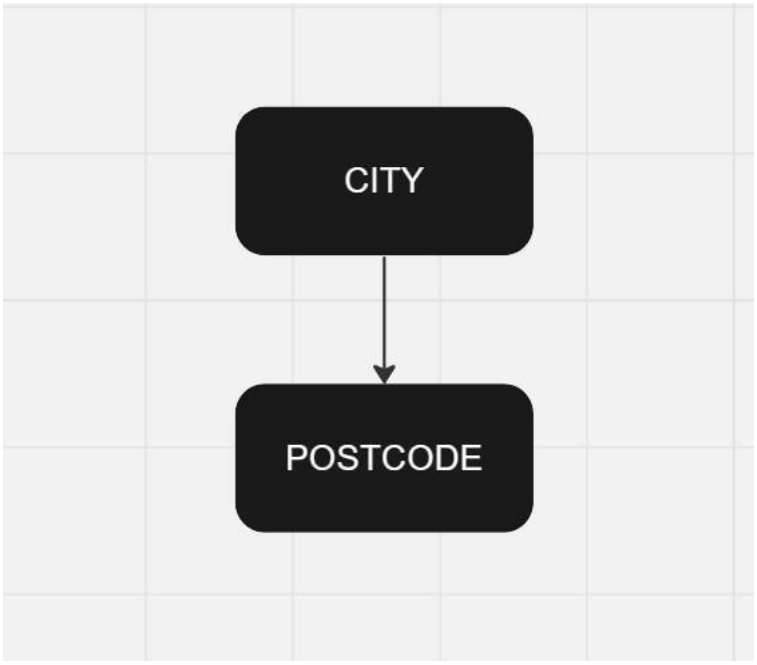


Figure 2: Location Hierarchy

1.2 Identify dimensions of central fact table(s) based on the available data source that will meet the reporting and analysis requirements mentioned in the business case scenario.

Before identifying the dimensions of our fact table, it can be useful to understand the given data sources. They are:

1. **CustomerDetails-1:** A CSV file containing the customer's full name, customer codes, date of birth, marital status, gender, postcode, and city.
2. **CustomerDetails-2:** A text file containing the customer's first name, last name, and income.
3. **SellingChannels:** A text file containing the channel name, channel code, and commission rate.
4. **GeneratedDateTime:** A text file containing 21 columns; Date Key, Full Date, Date Name, Day of Week, Day of Month, Day of Year, WeekdayWeekend, Week of Year, Month Name, Month of Year, Last Day of Month, Calendar Quarter, Calendar Year, Calendar Year Month, Calendar Year Quarter, Fiscal Month of Year, Fiscal Quarter, Fiscal Year, Fiscal Year Mont, Fiscal Year Quarter.
5. **PaymentDate**: A CSV file containing the payment type name, and Retailer Payment Type ID.
6. **CustomerSaleTransactions:** A CSV file containing the Date, Customer Code, Invoice Number, Total Cost, Toral Retail Price, Payment Type ID, and Selling Channel.

The dimensions of the fact table were identified by asking 'when,' 'who,' 'where,' 'what,' 'how,' and 'why' related to the event. They represent the descriptions of the attributes for filtering and grouping the facts.

For the first fact table – CustomerTransactions – the event is capturing transactions daily (when), by customers (who). The 'what' – product – is not provided in the source information.

Based on the above, the following dimension can be identified based on the available data source and requirements:

- Date dimension
- Customer dimension

The second fact table – CustomerPreferences –also captures events daily (when), by customers (who), on online sales channels (where), and by payment methods (how).

- Date Dimension
- Customer Dimension
- Selling channel Dimension
- Payment Dimension

For the third fact table – CustomerProfitability – the answer to the above question is shorter; The event is captured monthly (when) and by (who). This gives the following dimensions:

- Date dimension
- Customer dimension

1.3 Identify attributes of dimensions and measures of the fact table(s) based on the available data source descriptions that will meet the reporting and analysis requirements mentioned in the business case scenario.

To meet the business requirements, it is necessary to populate the dimension models with attributes that can be used for filtering and grouping. We also need to add numeric values as measurements to our fact table.

Date dimension:

Despite the source data containing more than twenty columns/attributes for each date, it is not necessary to add all of these to meet Blue Sky Online's business requirements. The minimum number of attributes for this dimension are:

- FullDate (PK)

- Month of Year
- Calendar Quarter
- Calendar Year

Though this is the minimum number of attributes to meet the business requirements, it can be useful to consider adding more when building the data warehouse, in case the business requirements change.

Customer Dimension:

The customer dimension has combined attributes from two sources – CustomerDetails-1 and CustomerDetails-2 – to create a comprehensive customer dimension. It includes the following:

- Customer Key (PK)
- Customer Code
- First Name
- Last Name
- Birth Date
- Marital Status
- Gender
- Post Code
- City
- Income

Note that commission is a numeric value. However, it is a ratio rather than a fully additive numeric value. It has been included in the Selling Channel dimension because it is used for filtering and grouping.

Selling channel dimension:

- Selling Channel Key (PK)
- Selling Channel Name
- Selling Channel Code
- Commission

Note: Commission is a numeric value. However, it is a ratio rather than a fully additive numeric value. It has been included in the Selling Channel dimension because, in addition to being a measurement (see fact measurements below), it can also be used for filtering and grouping.

Payment Type Dimension:

- Payment Type Key (PK)
- Payment Type Name
- Payment Type ID

1.4 Use simple star schema to define the structure that shows how the fact tables are related to their dimensions. Create a graphical representation of the three simple star schemas on one page.

After careful consideration, we decided on a structure that combines the three business requirements into one fact table. The justifications for the design of the simple star schema below are:

First, as both 'CustomerSalesTransactions' and 'CustomerPreferences' have the same granularity – customer transaction level – it is useful to combine these two fact tables into one single fact table. The fact table will be connected to all four dimensions as we will need date, customer details, selling channel details, and payment type details to meet business requirements 2 and 3. Business requirement two needs transaction information, date, and details from the customer dimension to analyze purchase history by customer demographics. To meet the third business requirement, we will need information on the selling channel, payment method, customer, and date. We would then have to aggregate the individual transactions to identify the most used selling channel or payment type.

We were then left with two fact tables, 'CustomerSalesTransactions' and 'CustomerProfitability.' The initial thought was to design two simple star schemas based on these, as the preferred granularity would be on transaction level for the former, and period and customer for the latter. The idea was the table would show the total profit per customer for that given period. We could then meet the fourth business requirement by filtering the customers by profit to get the most profitable customer.

However, this design produced a challenge. Profit is calculated by taking the 'Total Retail Price' and subtracting 'Total Costs' and 'Commission.' Both retail prices and costs can easily be aggregated across time and customer. Commission, on the other hand, is a ratio, so it cannot be aggregated in the same way across dimensions. In addition, the commission is based on what type of selling channel has been used for a specific transaction. So, unless the customer has used the same selling channel across the period it will also be harder to calculate the exact commission rate for the month without calculating this on the transaction level.

To get around this issue most simply, we decided that it would be better to combine all three fact tables into one. The grain would be the sales transaction level for all; each row corresponds to one customer sales transaction. We can then calculate the profitability for each transaction, and then aggregate this by customer and time to get the most profitable customer in a month, quarter, or year. This design requires a little more aggregation than separating profitability as an individual fact table, but it allows us to meet business requirement four without having to make more complicated commission calculations.

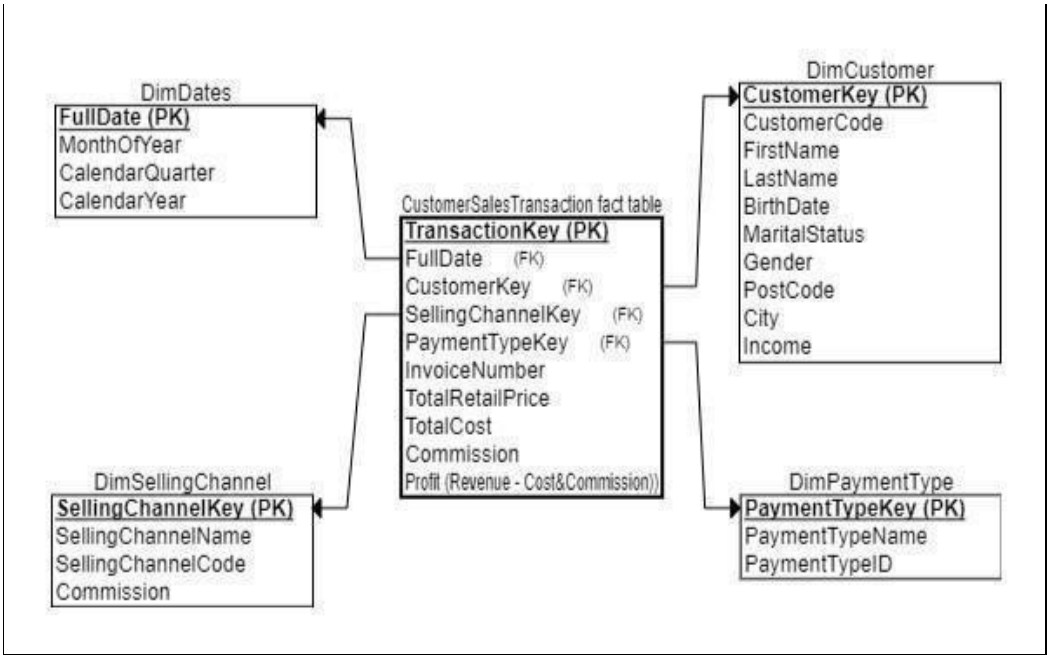


Figure 3: Simple Star Schema

We have also added the following SQL queries that show that our design can meet the business requirements.

Business Requirement: Identify customer preferences in terms of selling channels/ payment payments.

```
SELECT pt. PaymentTypeName as most_popular_payment_type,  sc.

SellingChannelName as most_popular_selling_channel,

COUNT (*) as frequency

FROM CustomerSalesTransaction cst

JOIN DimPaymentType pt ON cst. PaymentTypeKey = pt. PaymentTypeKey
```

JOIN DimSellingChannel sc ON cst. SellingChannelKey= sc. SellingChannelKey

GROUP BY pt. PaymentTypeName, sc. SellingChannelName

ORDER BY frequency DESC.

This query counts the frequency of each combination of payment type and selling channel together. To find the most popular payment type and selling channel individually, the query needs to be two separate subqueries for each dimension.

Business Requirement: Identify the best and most profitable customers at different periods (month, quarter, and year).

SELECT dc. LastName, SUM

(cst. Profit) as TotalProfit

FROM CustomerSalesTransaction

JOIN DimCustomer dc ON cst. CustomerKey = dc. CustomerKey

JOIN DimFullDate dd ON cst. FullDate = dd. FullDate

WHERE EXTRACT (month FROM dd. FullDate) = 1

GROUP BY DC. LastName

ORDER BY TotalProfit DESC.

This query will return based on the customer's last name. We might want to consider either joining first name and last name together to a name attribute or writing a more complex query to make sure that the results show exact customers in cases where there are customers with the same last name.

Business Requirement: Analyse purchase or transaction history based on demographic data such as age group, income group, marital status, and postcode/City/Country.

One example could be to find the TotalRetailPrice for customers who are married.

SELECT SUM (cst. TotalRetailPrice) as totalretailsales_married.

FROM CustomerSalestTransactions cst

JOIN DimCustomer dc ON cst. CustomerKey = dc. CustomerKey

WHERE dc. MaritalStatus = 'Married.'

From the above Conceptual Design, a physical design needs to be implemented as individual work.

2. Create a Relational Database to store dimensions and fact tables using Microsoft SQL Server Management Studio.

To create a relational database for Blue Sky Online, the database must first be created in Microsoft SQL Server Management Studio (SSMS). SSMS offers a graphical interface and a SQL query for performing this task.

2.1 Create a database.

Before starting to create a database there few rules that need to be followed, SQL Server can create up to 32,767 databases. It is necessary to execute the **CREATE DATABASE** command in auto-commit mode rather than in explicit or implicit transactions. The master database should be backed up when we create, modify, or drop the database.

To create a database Open Object Explorer, connect to a SQL Server Database Engine instance, and then expand it.

- To Select New Database by right-clicking on **Databases**.
- Give a database name as **"BlueSkyOnlineDB"** in the New Database. • Click **OK** to create the database using all of the default options.

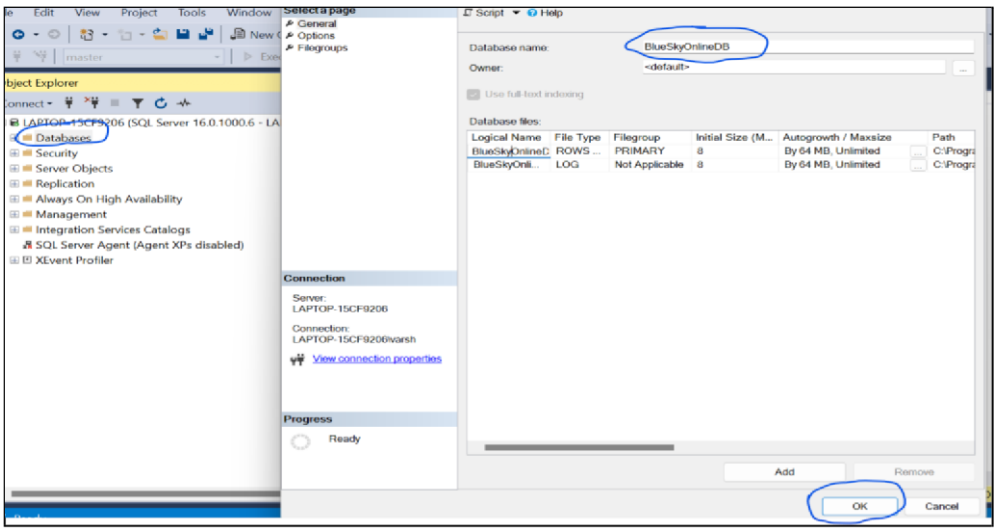


Fig: Creating Database "BlueSkyOnlineDB"

In a similar way to create a staging area, follow the previously mentioned steps for "BlueSkyOnlineDB" and name as "**BS_Stagging"**

It acts as a temporary storage place for raw data extraction from source systems, and a staging area is an essential intermediate stage in the ETL process. It facilitates data cleansing, transformation, and validation, ensuring accuracy and consistency. To prevent direct impacts during the data extraction and transformation phases, the staging area separates the extraction process from the data warehouse. Additionally, before the data is loaded into the final destination, it provides verification and data validation, making it easier to find inconsistencies or abnormalities.

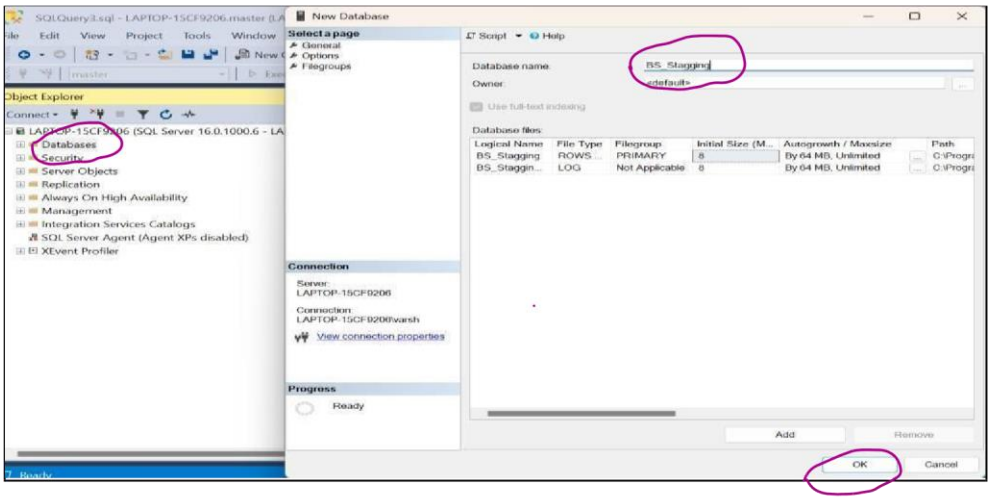


Fig: Creating staging area "BS_Stagging"

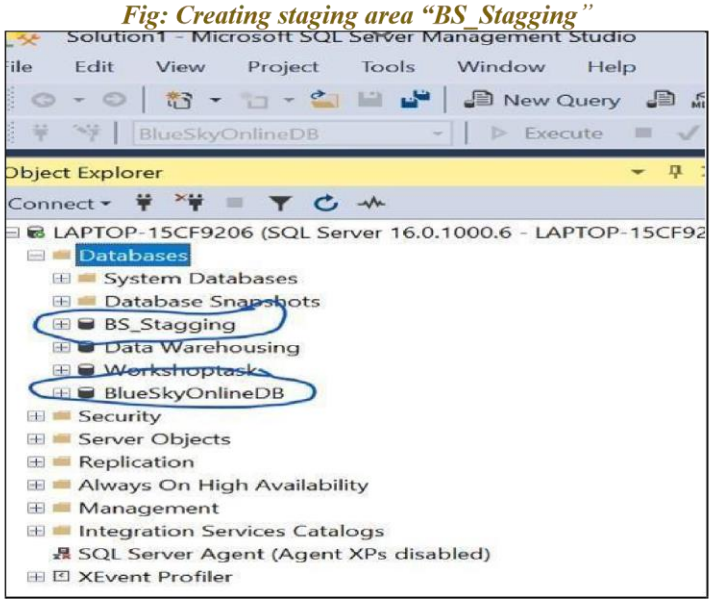


Fig: Databases "BlueSkyOnlineDB" and "BS_Stagging"

2.2 Create dimension tables.

To create a new table in a given database, SQL provides the create TABLE statement. The structure of a table must be specified in an SQL query before one can be created. The structure consists of the table name, the column names, and the data type for each column in the table.

Note: We have to make sure each table in a database needs a unique name.

After the database has been created, the dimension tables must be designed and implemented. These dimensional tables represent the main entities like Customers, Dates, Payments, and Selling Channels. Each dimension table has been created to include specific attributes related to its entity. Primary keys are assigned for uniquely identifying records inside each dimension table.

Creating the Customer Dimension table in SSMS. These are the customer dimensions that have been created by combining attributes from two sources.

Customer Key: It is the primary key for uniquely identifying for each customer.

Customer Details: FirstName, LastName, Gender and BirthDate.

Demographic Data: Age Group, Income, and Marital Status.

Location Details: Postcode and City

These are the steps for creating the Customer Dimension that have been followed.

- Connect to the database server by launching SQL Server Management Studio.
- Click on **"New Query"** in the toolbar to open a new query window.
- Make sure we are using the right database when creating the Customer Dimension **"BlueSkyOnline."**
- To run the query, use the **"Execute"** button in the toolbar.

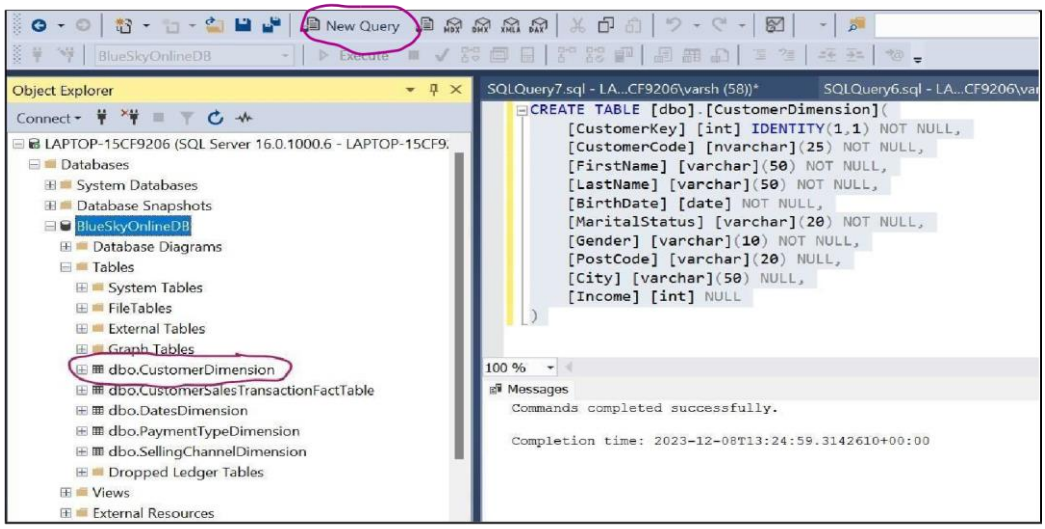


Fig: Creating CustomerDimension Table

To create a customer dimension table, use this SQL Query.

Follow the above steps for the other dimension tables and the fact table. These lines of code have been used for creating a table with date-related attributes. The query creates a **"DateDimension"** table with FullDate, MonthOfYear, CalendarQuarter, and CalendarYear.

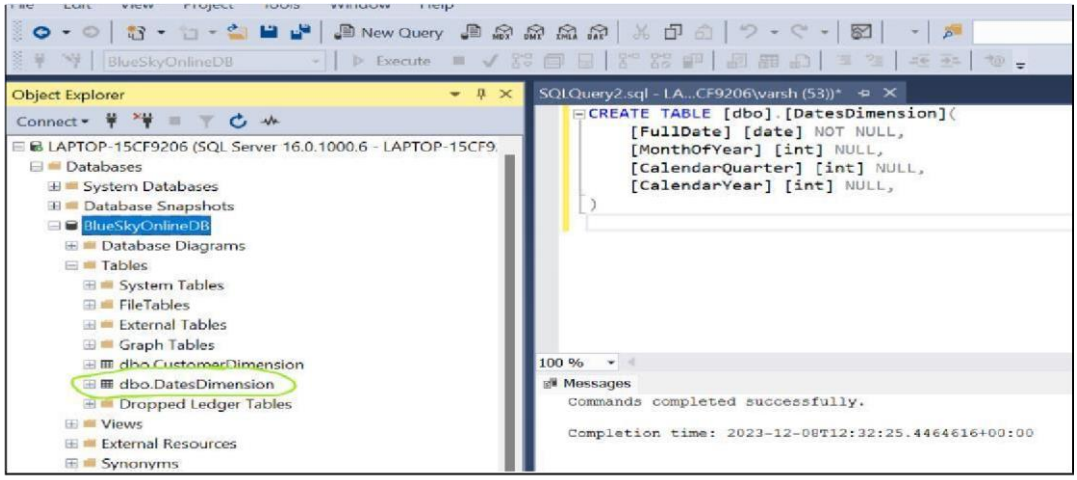


Fig: Creating DatesDimension Table

To create the **"PaymentTypeDimension"** table in the database. These lines of SQL query have been used for payment-related attributes like PaymentTypeKey, PaymentTypeName, and PaymentTypeID.

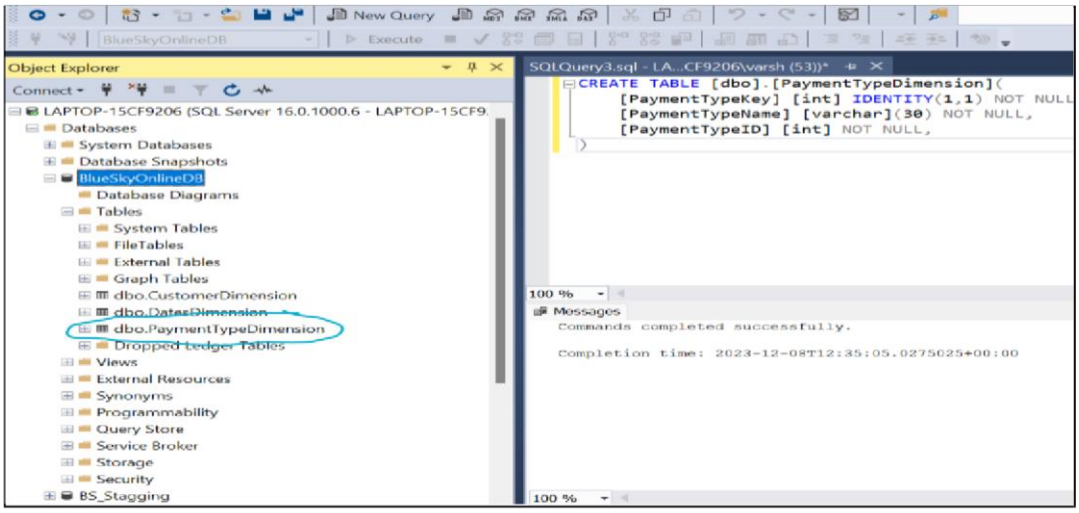


Fig: Creating PaymentDimension Table

For creating specific a table for selling channels as **"SellingChannelDimension"** attributes like sellingchannelkey, sellingchannelname, sellingchannelcode, and commission in the database. This dimension will provide insights into the various channels through which Blue Sky Online conducts its business.

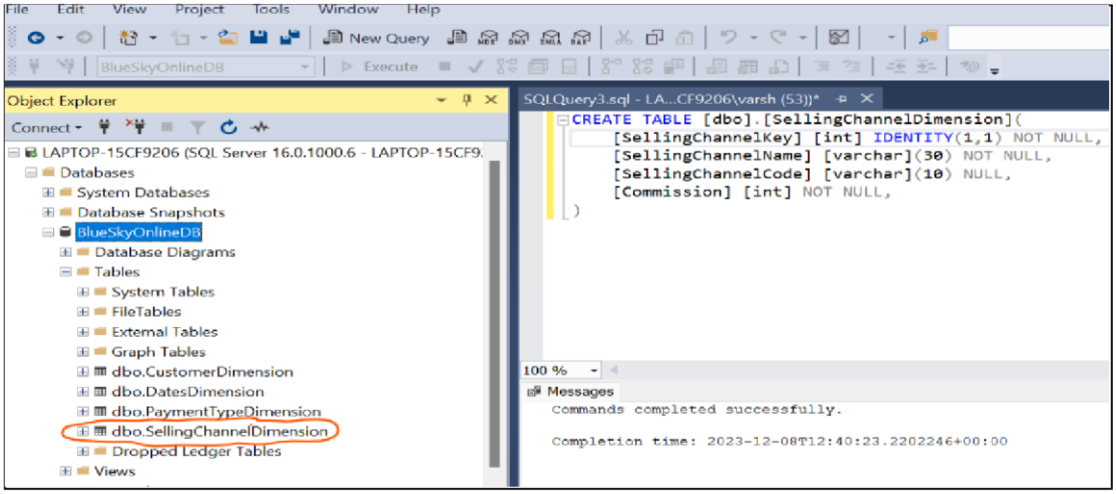


Fig: Creating SellingChannelDimension Table

2.3 Create fact tables.

In the Simple star schema of a data warehouse, the fact table is the central table. It stores quantitative information for analysis and is often denormalized. Dimension tables are working with this fact table. In addition, it contains the data that needs to be analyzed and a dimension table stores data about how the data in the fact table can be analyzed. As a result, the fact table has two distinct types of columns. The foreign keys column helps connect with dimension tables, while the measures columns contain the data that is being analyzed.

To create a Fact table in SSMS to store key details about customer sales transactions. The details including Customer details, Invoice Number, purchase Total retail price, and related data will be stored in this database as **"CustomerSalesTransactionFactTable."** Open a new SSMS query window and execute the following SQL code.

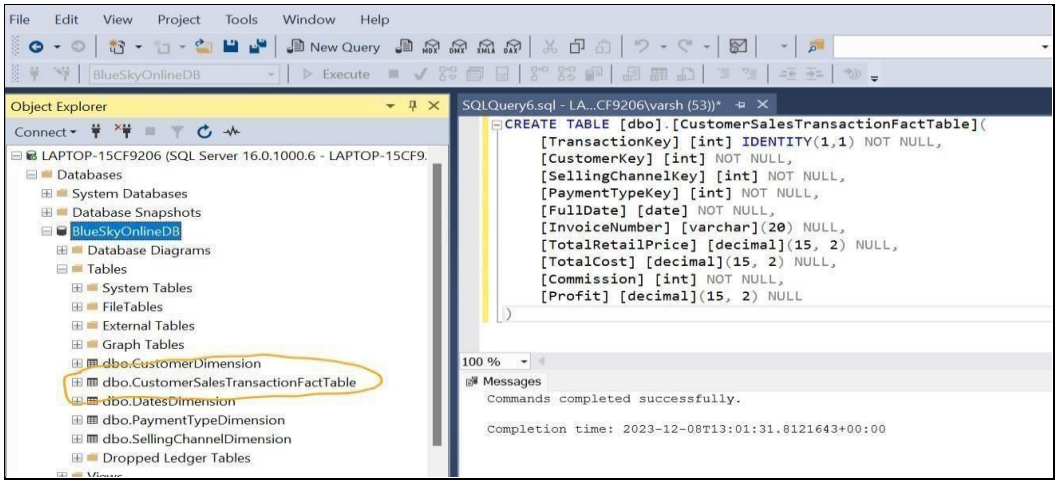


Fig: Creating CustomerSalesTransactionFact Table

2.4 Add appropriate primary keys and foreign key constraints.

To Build Primary Keys: We can use the *ALTER TABLE* command to change a primary key in a dimensional table after it has been created. Suppose if we want to modify the primary key of the CustomerDimension table from the column *CustomerKey* as *PrimaryKey.*

In this SQL code, the *ALTER TABLE* statement is used to modify as primary key constraint *(PK_CustomerDimension)* from the *CustomerDimension* table along with other dimensional tables like *DateDimension, PaymentTypeDimension*, and *SellingChannelDimension*.

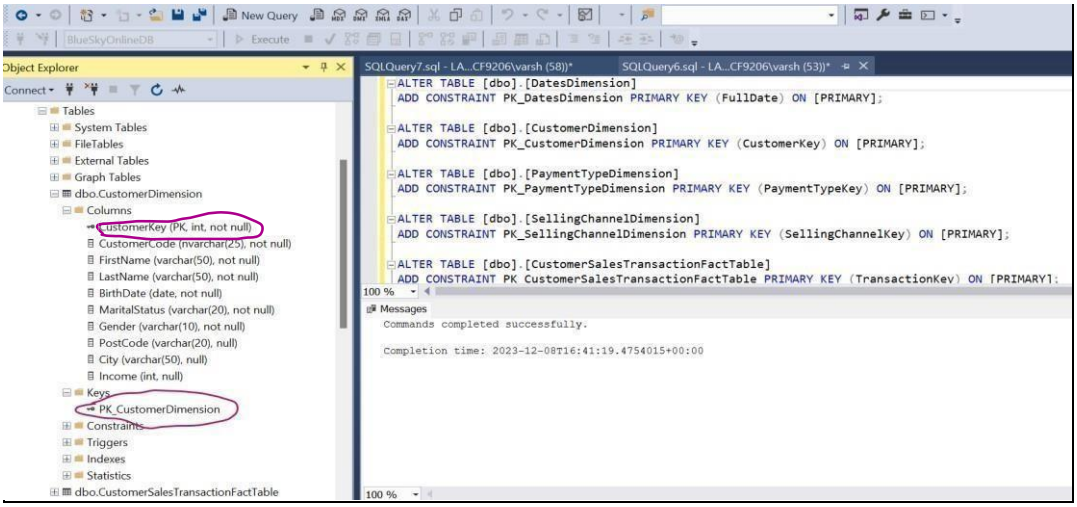


Fig: Altering CustomerDimension table to add PrimaryKey

By following the above code, the same as adding constraints for the customer dimension table other dimensional table constraints have been modified.

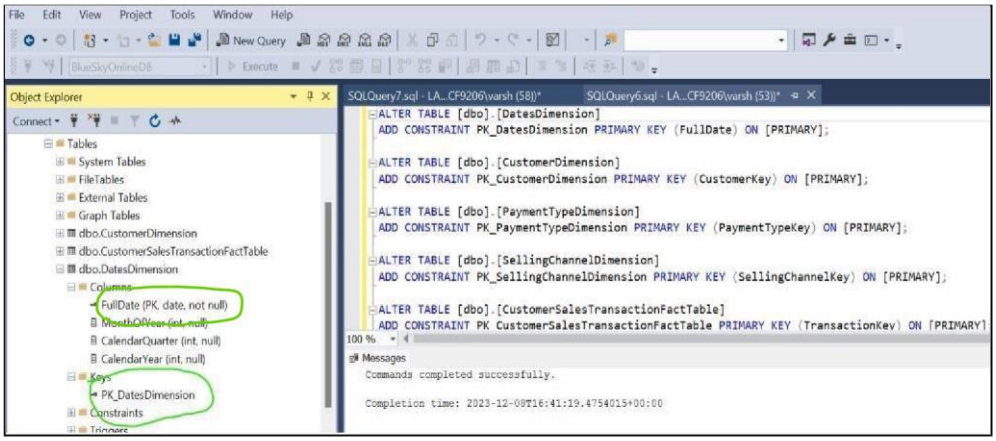


Fig: Altering DateDimension table to add PrimaryKey

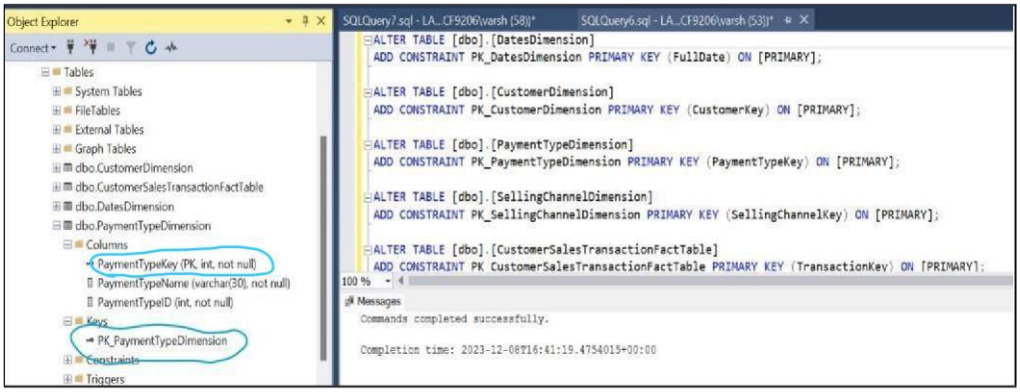


Fig: Altering PaymentTypeDimension table to add PrimaryKey

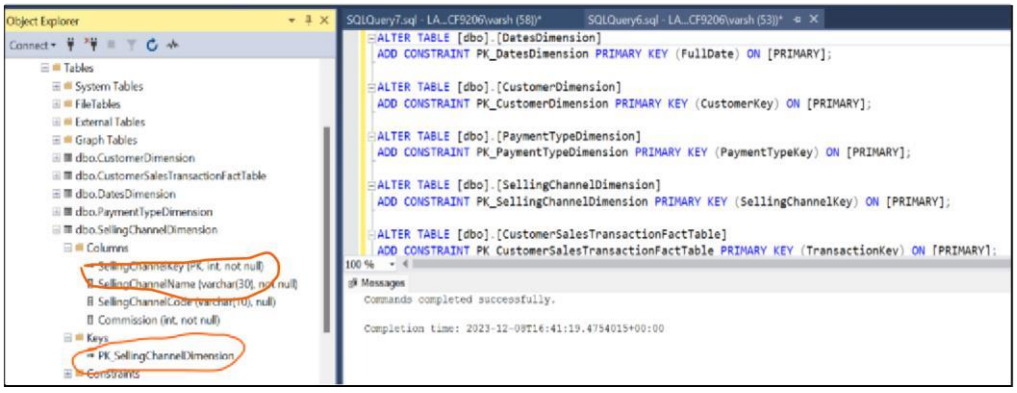


Fig: Altering SellingChannelDimension table to add PrimaryKey

To Build Foreign Keys: When we create foreign keys in the fact table, we are making connections to data stored in other dimension tables. Here, we have a *CustomerSalesTransactionFactTable*, which captures details about the customer transactions, such as which channel and when.

The below SQL code refers to the primary keys of relevant dimension tables through the use of foreign keys (*CustomerKey, FullDate, SellingChannelKey*, and *PaymentTypeKey*) in the *CustomerSalesTransactionFactTable*.

We have added the PRIMARY KEY constraint to the **"TransactionKey"** column to make sure that each transaction has a unique identification. Furthermore, each foreign key constraint refers to the associated primary key in the dimension tables *(CustomerDimension, DateDimension, SellingChannelDimension, and PaymentTypeDimension).*

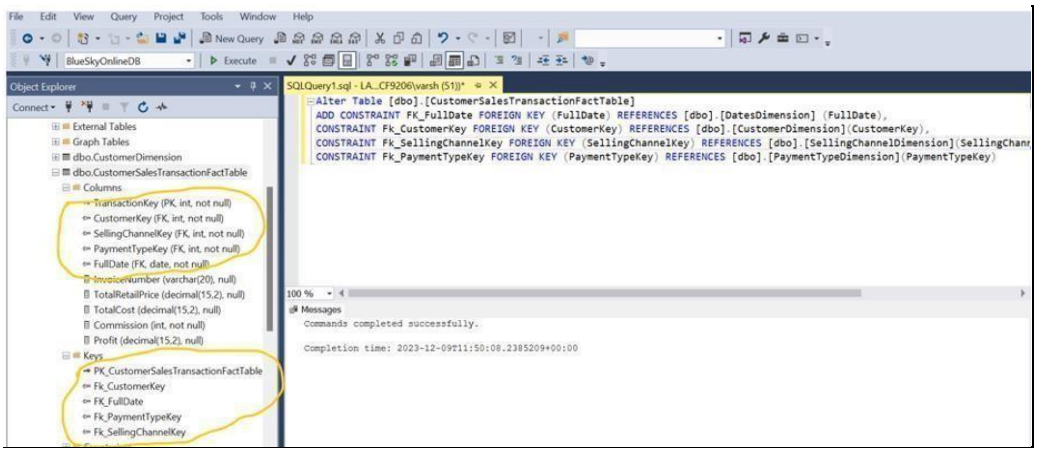


Fig: Altering CustomerSalesTransactionFactTable to add PrimaryKey and ForeignKey

3. Populate the implemented data warehouse based on the provided data sources using SQL Server Management Studio.

A sequence of steps must be followed before loading data into dimension tables to perform ETL processes in SQL Server Management Studio. A staging area has been created and data has been inserted in the staging area as **"BS_Stagging."**

3.1 Implement ETL (extract, transform, and load) processes to populate dimension tables.

Once the staging is created, the available data will be temporarily stored in the staging area in the database. Before inserting the data into the final dimension tables, this staging area serves as an intermediate stage where we can clean, transform, and validate the data. In these cases, we could manage missing values, standardize date formats, or carry out other data-cleaning tasks.

Importing customer details:

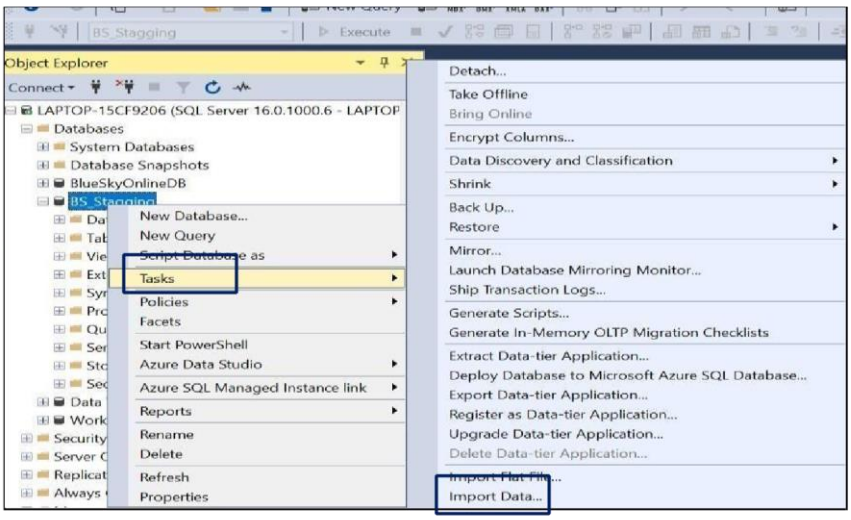


Fig: Importing customer details by selecting Task

- Select the Task to start the SQL Server import and export wizard, and then import.

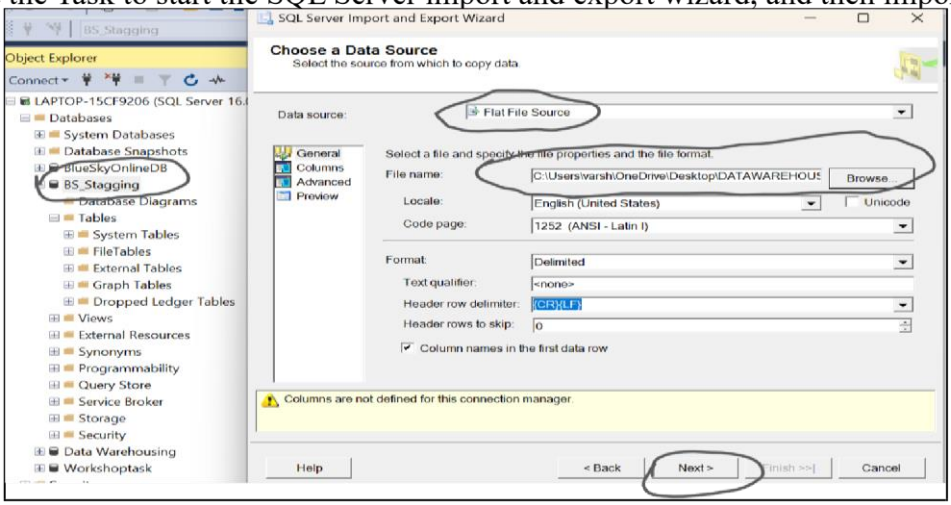


Fig: Importing data files into BS_Stagging

- Select the customer details data source "*Flat File Source*."
- Select a flat file source because the customer data that is generated has been stored in a CSV file with comma separated.

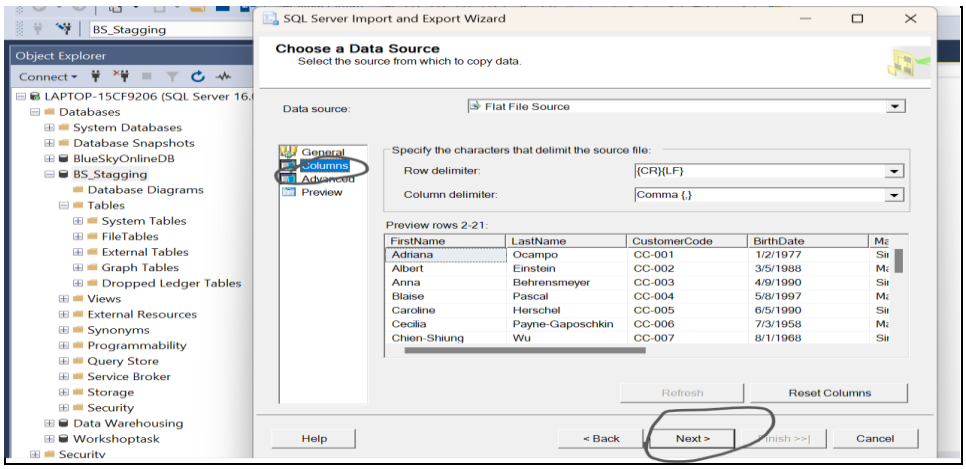


Fig: Displaying the columns and preview of the data source

- When we select a file, information about that file is displayed. We can also see the columns and preview the data source. It is suggested to preview the data source to quickly know the data type of each column.

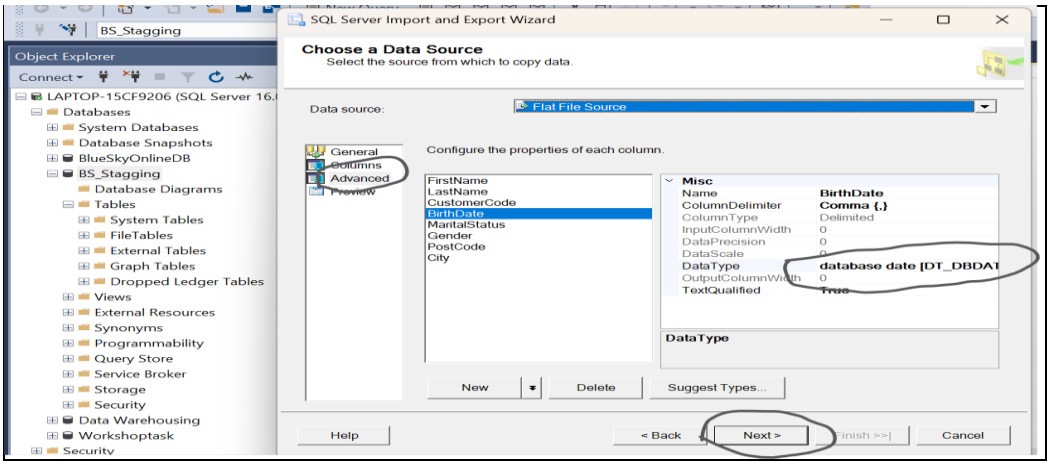


Fig: Showing an advanced option to change datatype

- To specify the data type and length, use the advanced feature. When importing or extracting data from a flat file source.

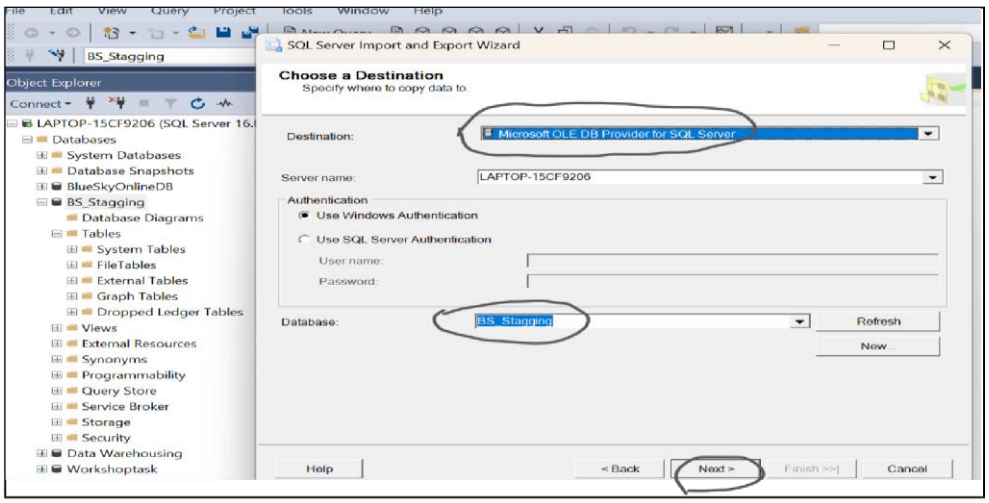


Fig: Choosing a destination for the ETL process

- The final destination is the Microsoft OLE DB provider for SQL Server for the Customer Details
- Select a destination for the ETL process. Although the purpose is to populate the customer details in Stagging, the final destination is the data warehouse, which is managed by an SQL server.
- Customer Details 1 is the destination dimension table to be specified. If the ETL process is successful, a popup will appear indicating that the execution was successful and how many records were extracted and loaded to the Customer Details.

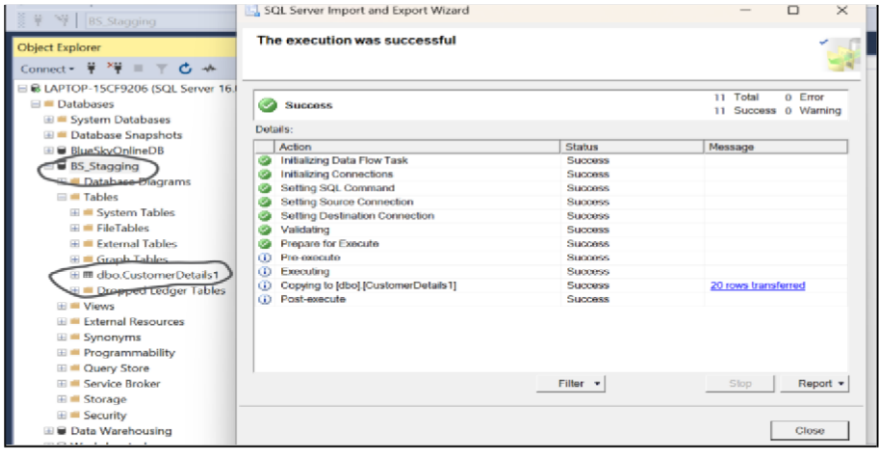


Fig: Execution window showing successful

Similarly for importing other details available in these case studies, to populate the other details available for Blue Sky online using flat file sources, repeat the procedure used for the previous customer details.

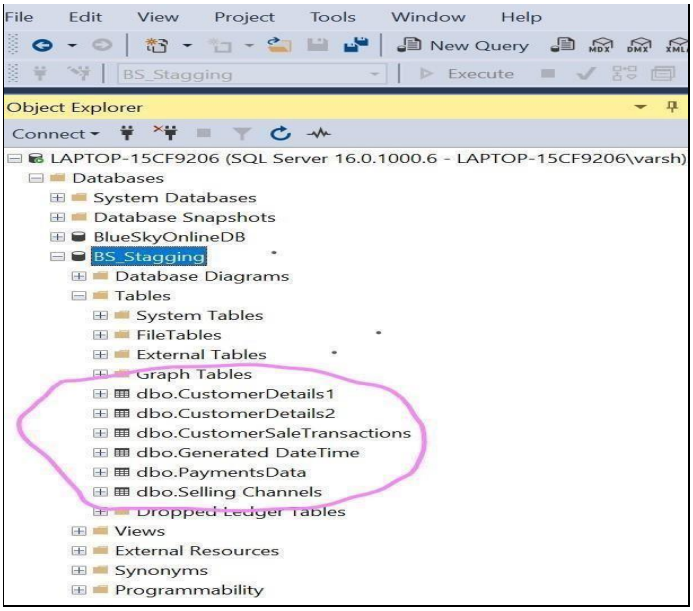


Fig: Imported data sources in BS_Stagging

Once the data files are imported into the final dimension tables, it has been validated and loaded into the staging area. With this approach, we can ensure that thoroughly cleaned and structured data is utilized in data warehouses by controlling the quality of the data before inserting it into dimension tables. We may transfer data from the staging to the original dimension tables using INSERT INTO...SELECT query.

3.2 Implement ETL (extract, transform, and load) processes to populate a fact table.

The staging area is a temporary storage space, where raw data enters before moving it to the dimension tables. The required data from this staging area is subsequently extracted and transformed into a format that is suitable for distinct dimension tables.

We have customer data in the staging area, so we extract it, apply any necessary transformations (such as formatting addresses or providing unique customer IDs), and then place it into the "**CustomerDimension"** table.

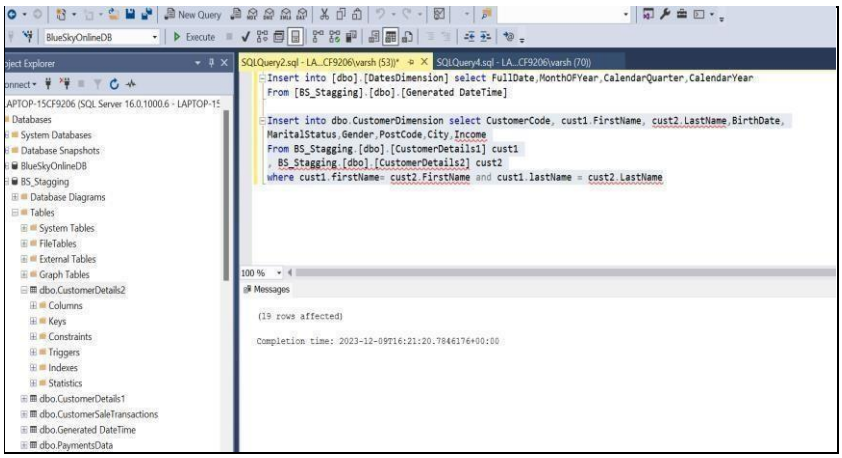


Fig: Inserting data from BS_Stagging to CustomerDimiension

In this SQL code, we have used a *SELECT* query to select relevant columns from the staging area "BS_Stagging" and *INSERT* them into the "*CustomerDimension*" table. *WHERE* clause filters the data to include only records where the customer's *firstname* and *lastname* are same.

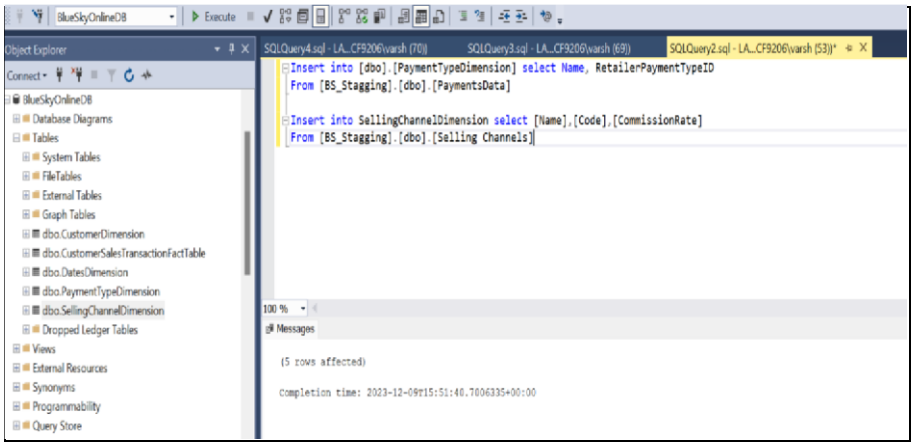


Fig: Inserting data from BS_Stagging to PaymentTpeDimiension and SellingChannelDimension

This procedure is done for each dimension table and is a key step in keeping a data warehouse up-to-date and dependable.

In this SQL script, we are inserting particular data from the BS_Staging to the *DateDimension* table, such as *FullDate, CalendarQuarter, MonthOfYear, and CalenderYear***.**

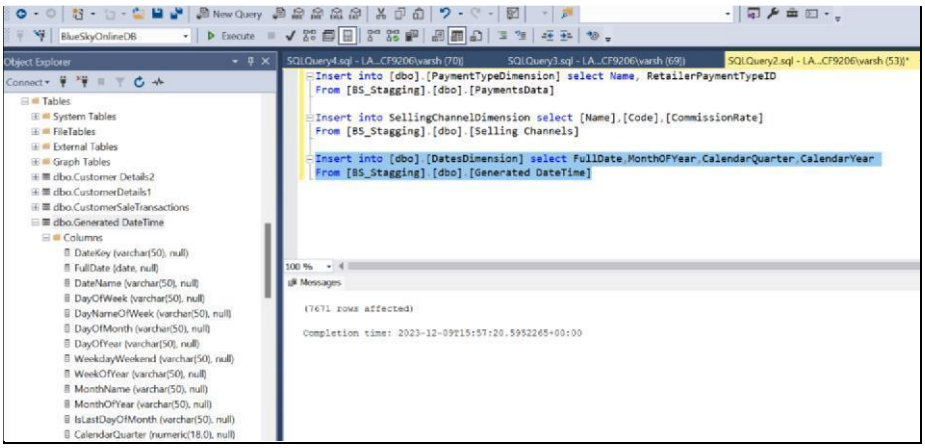


Fig: Inserting data from BS_Stagging to DateDimiension

To import data from the Blue-Sky Staging area (BS_Staging) into the *CustomerSalesTransactionFactTable*, use a similar procedure. This process involves extracting relevant data from the staging area, possibly transforming it if needed, and then loading it into the fact table.

We can get transactional data from the staging area, alter it as needed, and load it into our fact table to gain insights into customer transactions. The "*CustomerSalesTransactionFactTable*" stores data such as *TransactionKey, CustomerKey, SellingChannelKey, FullDate, InvoiceNumber, Commission and TotalCost*.

In this SQL code, we use a **SELECT** statement to extract data from the staging area (BS_Staging) and **INSERT** them into the *CustomerSalesTransactionFactTable* along with that we used a **WHERE** statement to filter the records from the BS_Staging

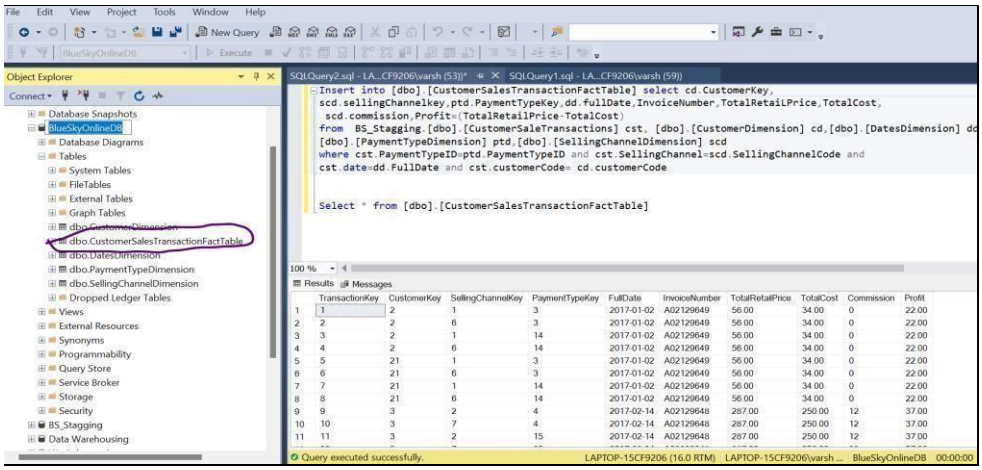


Fig: Inserting data from BS_Stagging to CustomerSalesTransactionFactTable output

4. Migrate test data from the data warehouse to an Apache Hadoop platform for further analysis of Big Data using Hortonworks Data Platform (HDP)

An individual physical machine's storage capacity becomes exhausted when the amount of data increases. This expansion requires the need to split our data across multiple devices. Distributed File Systems are File Systems that handle data storage over a network of machines. HDFS is an integral part of Apache Hadoop that is designed to store huge data files with streaming data access patterns on commodity hardware clusters. Hortonworks Data Platform (HDP) has enhanced HDFS to allow heterogeneous storage media within the HDFS cluster.

4.1 Export the data warehouse database data into an external data file.

SQL Server Import and Export Wizard is a graphical tool for transferring data between SQL Server databases and other data sources, as well as exporting data to CSV files. To start with, launch SQL Server Management Studio (SSMS). Connect to SQL Server machine.

To get the customer dimension table data in a CSV file, In the Object Explorer, right-click on the database containing the table that want to export, go to "Task," and then select "Export Data."

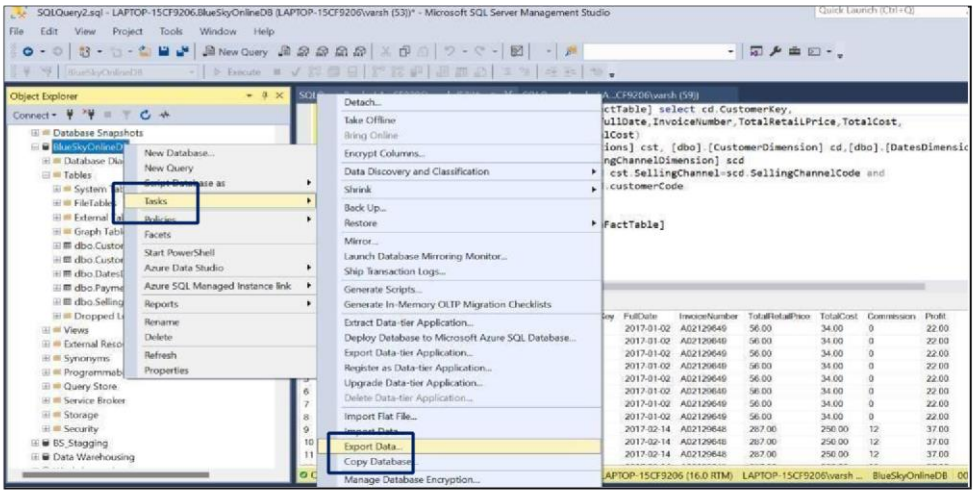


Fig: Exporting customer's data by selecting Task

The data source is the Microsoft OLE DB provider for SQL Server for the Customer Details

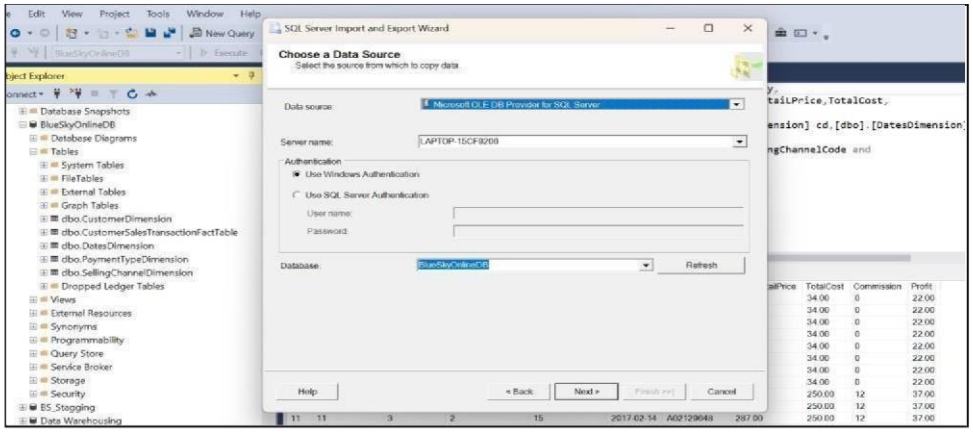


Fig: Selecting data source for data files

For the Customer dimension, choose a data source. Select a flat file source because the date data that is created remains stored as tab-separated text.

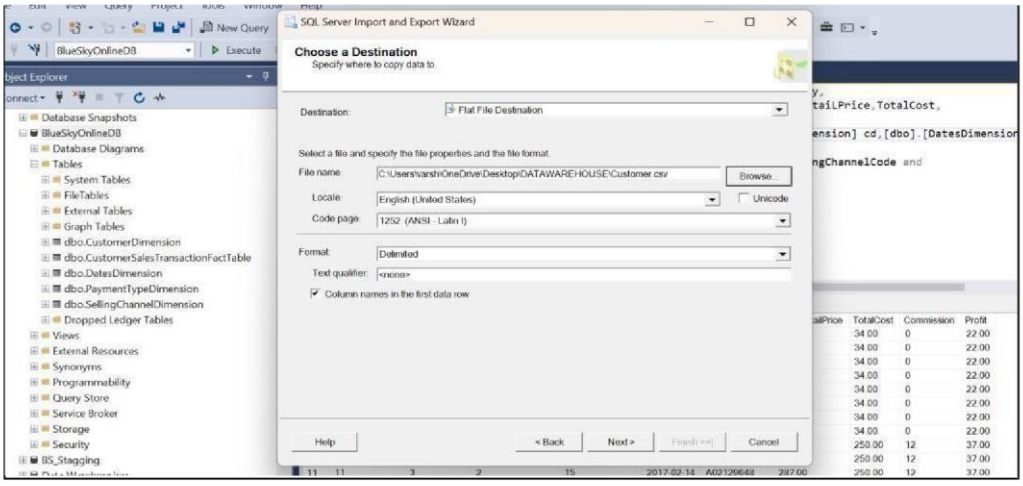


Fig: Selecting a destination for Customer details

Specifying the customer dimension as the target dimension table

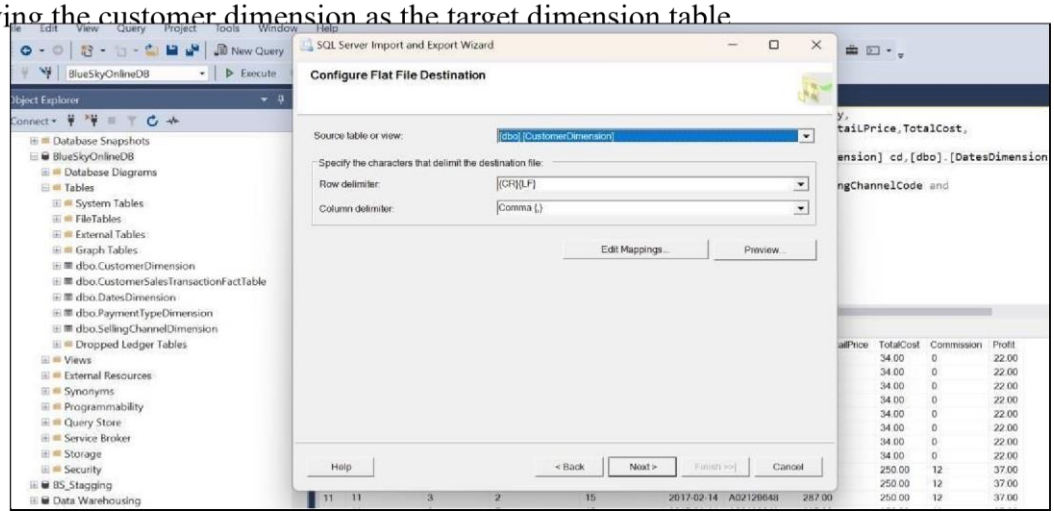


Fig: Seecting Flat File Destination for Customer dimension

If the exported data process is successful, a dialog box will appear showing that the execution was successful as well as the number of records generated and stored in the Customer dimension.csv.

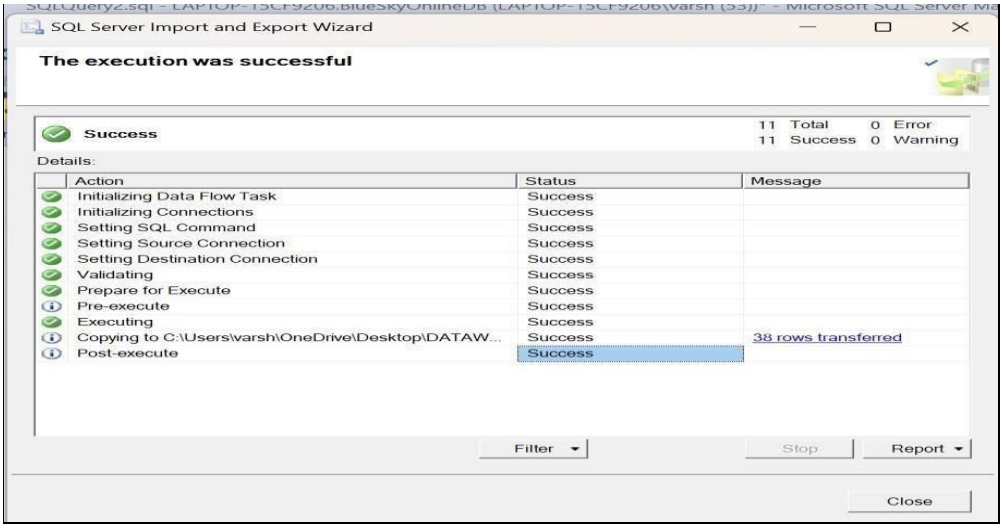


Fig: Execution window showing successful for customer dimension

In a similar way to exporting data to the other data available for Blue Sky online utilizing flat file sources, repeat the process used for the previous customer dimension.

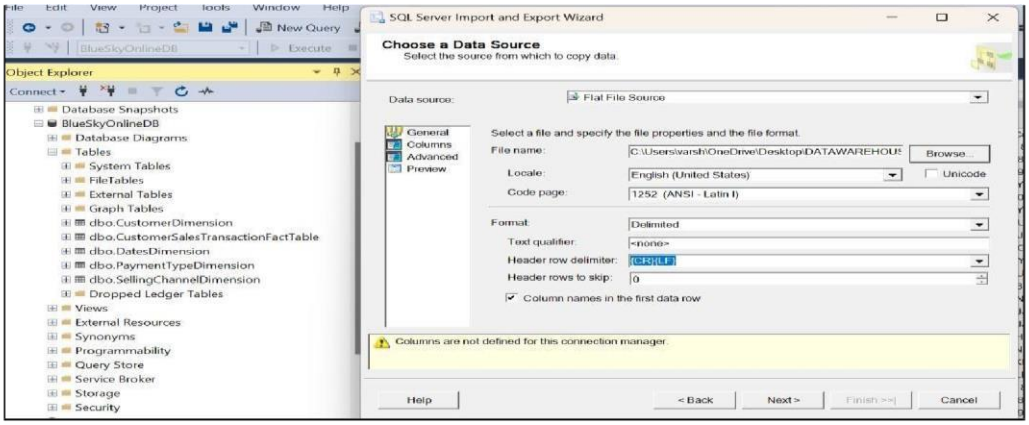


Fig: Choosing data source for customer sales transaction fact table

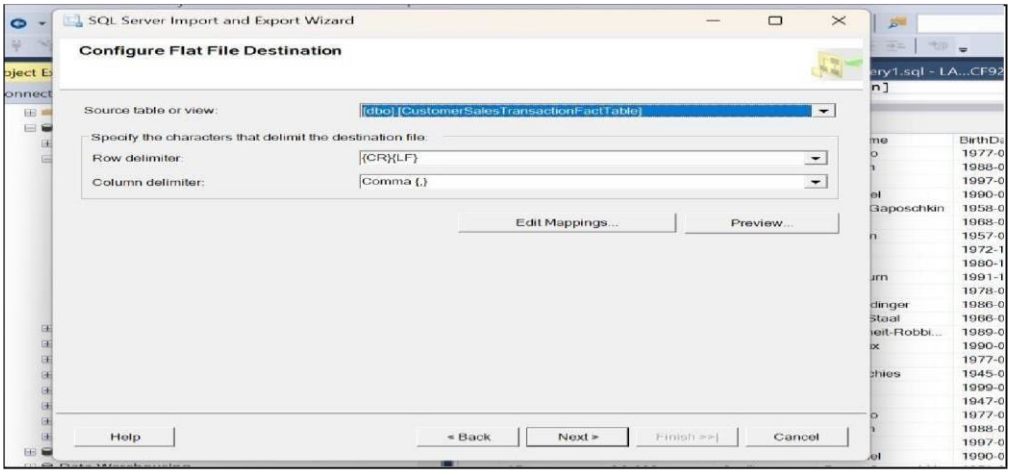


Fig: Selecting destination file for customer sales transaction fact

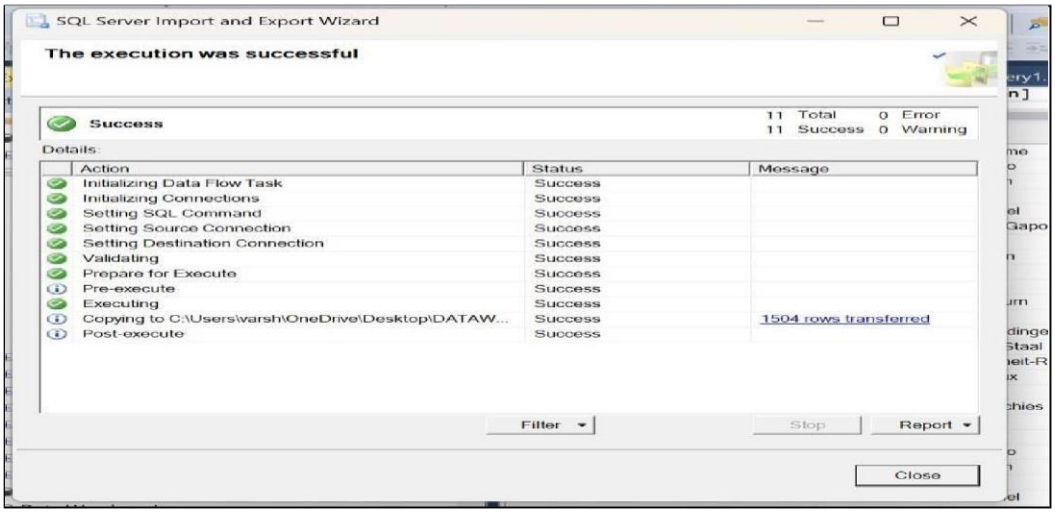


Fig: Execution window for Customer Sales Transaction Fact

4.2 Migrate the data file from the file system to Apache HDFS

The Hadoop Distributed File System (HDFS) is a critical component in the field of big data. Consider it a virtual space that spans multiple machines, allowing huge amounts of data to be

stored and processed efficiently. In simple words, instead of storing all of our data on a single computer, HDFS breaks it into smaller portions and distributes them over a cluster of machines. This technique has two major advantages:

- It allows parallel processing, which means allowing many computers to work on various sections of the data at the same time, speeding up tasks.
- It provides fault tolerance, which means that even if one computer fails, the data is still accessible and may be retrieved from other nodes in the cluster.

HDFS serves as the foundation for maintaining and analyzing large datasets in big data applications.

Load Data in HDFS: To begin with, select the HDFS Files to view from the Side panel or Offcanvas menu at the top. We can see the Hortonworks Data Platform (HDP) file store using the HDFS Files view. This is a standalone of the local file system. It will be included as part of the file system in the Hortonworks Sandbox VM.

After getting CSV files from the SQL server, we have to load the data into HDFS. To load data files in HDFS we have to log in to Ambari.

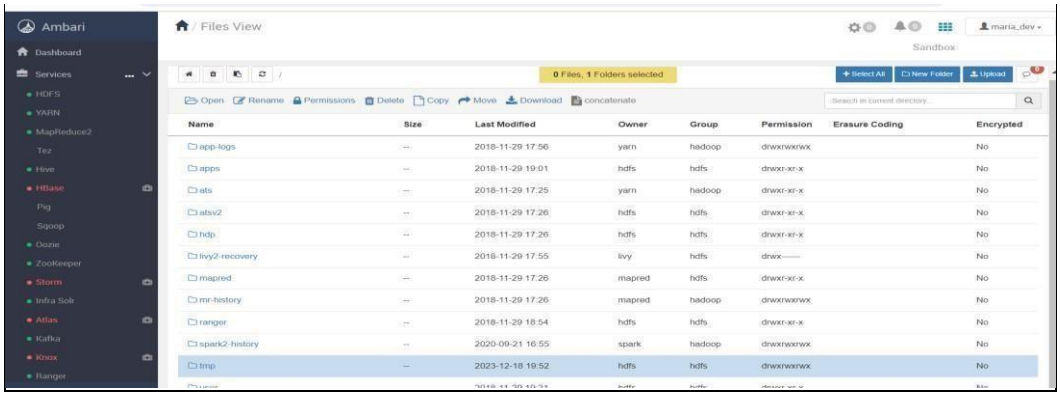


Fig: Ambari Dashboard

- Log in to the Ambari web interface using your web browser http://localhost:8080
- Enter the login credentials Maria_dev for both username and password.
- Go to the burger icon and select file by clicking on **"File View."**
- Here we can see all of the files to which the logged-in user maria_dev has access.
- Switch to the **"tmp"** directory by clicking on the directory links.

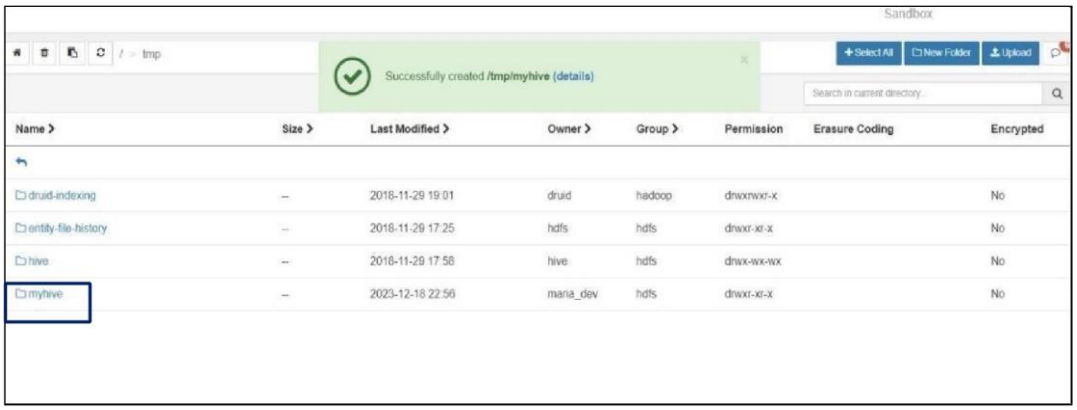


Fig: Creating a new folder for data files

- Create a separate folder for data files as my hive. To add that directory, click the New Folder option, then name the folder and create it, along with giving it permission to read, write, and execute. The folder's path we should see /tmp/myhive.

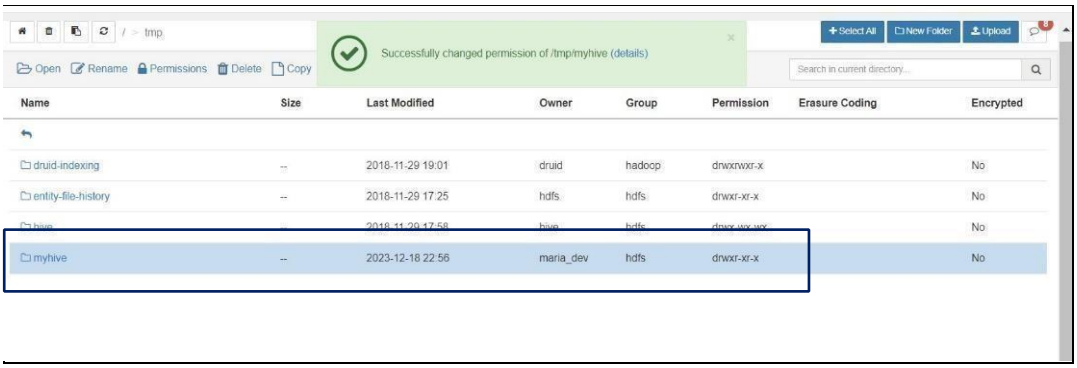


Fig: Giving permissions to the folder

- We are on the newly created directory path /tmp/myhive, so navigate to the myhive folder. Then, click the upload button to upload into the Hortonworks Sandbox environment with suitable data files ***CustomerDetails.csv*** and ***PaymentDetails.csv*** files.

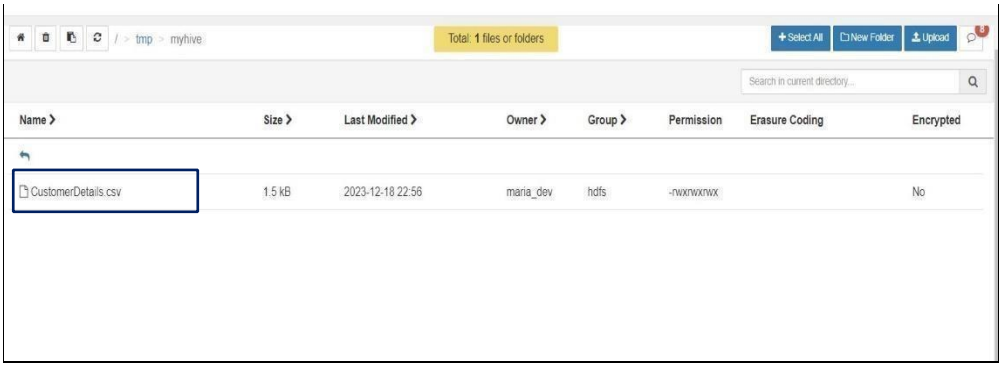


Fig: Uploading Customer Details file in HDFS

By clicking on the entity's row, we can execute the following operations such as *open, rename, permissions, delete, copy, move, download, and concatenate*.

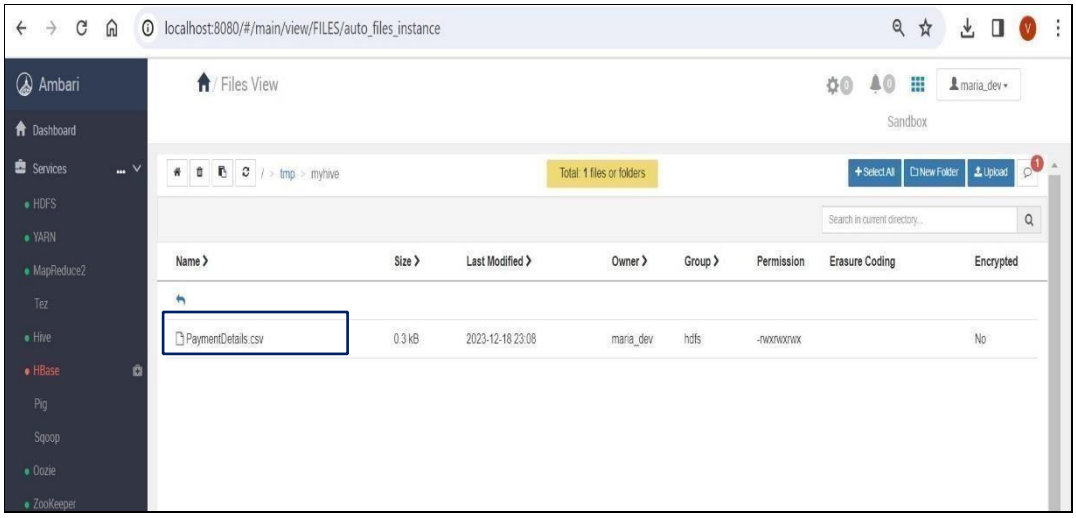


Fig: Uploading Payment Details file in HDFS

4.3 Create a suitable data structure for loading the data file into the HIVE.

Apache Hive is a data warehouse infrastructure built on Hadoop Environment, enabling data summarization, query, and analysis using the Hive Query Language (HiveQL). It facilitates reading, writing, and managing large datasets residing in distributed storage using a SQL-like language.

This SQL-like language allows users to express queries using a SQL-like syntax. Hive translates HiveQL queries into MapReduce tasks, allowing them to be executed on a Hadoop cluster for efficient data management.

To perform HIVE, Hortonworks provides Data Analytics Studio (DAS) as a platform for doing a variety of analytics activities, such as interfacing with Hive to query and analyze data. DAS is an easy-to-use web-based interface for browsing data, writing, and executing Hive queries, and carrying out analysis on the Hadoop environment.

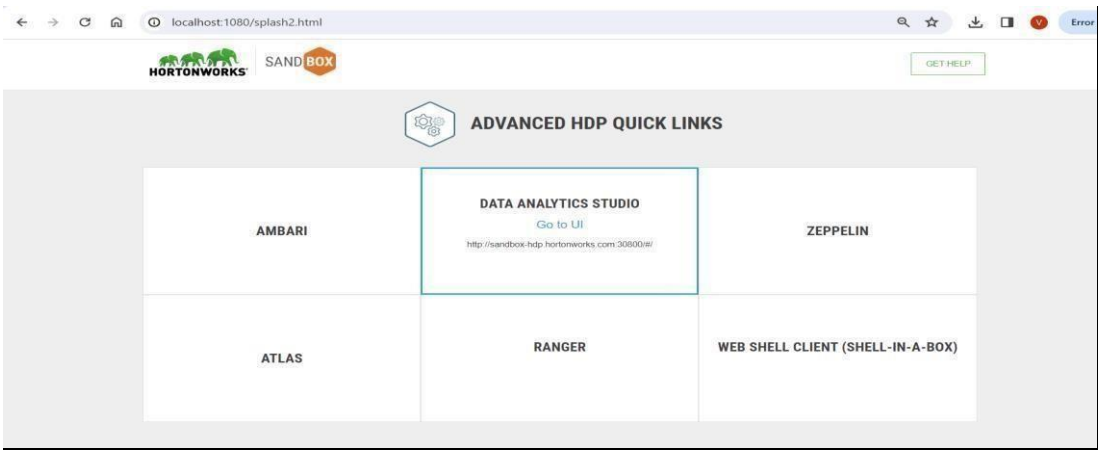


Fig: HortonsWorks Dashboard

For accessing the Data Analytics Studio user interface. Open the internet browser and type DAS's URL into the address bar. http://localhost:1080 sandbox-hdp. when DAS running hortonworks.com at port 30800. enter our login information to sign in.

Hive is suitable for data warehousing tasks using a language similar to SQL and uses HiveQL. Using the Apache Hadoop data store as its operating platform, Hive offers a data workspace for analyzing, modifying, and altering data. It performs data processing operations one query or line at a time, it gives the ability to a user to solve problems and rearrange priorities in response to results. This adaptability makes it possible to manipulate data quickly and efficiently.

At first, we have created a table to store the data. Using the Query Editor, we will write the query. Click the **Execute** button located at the bottom of the query once you have completed it.

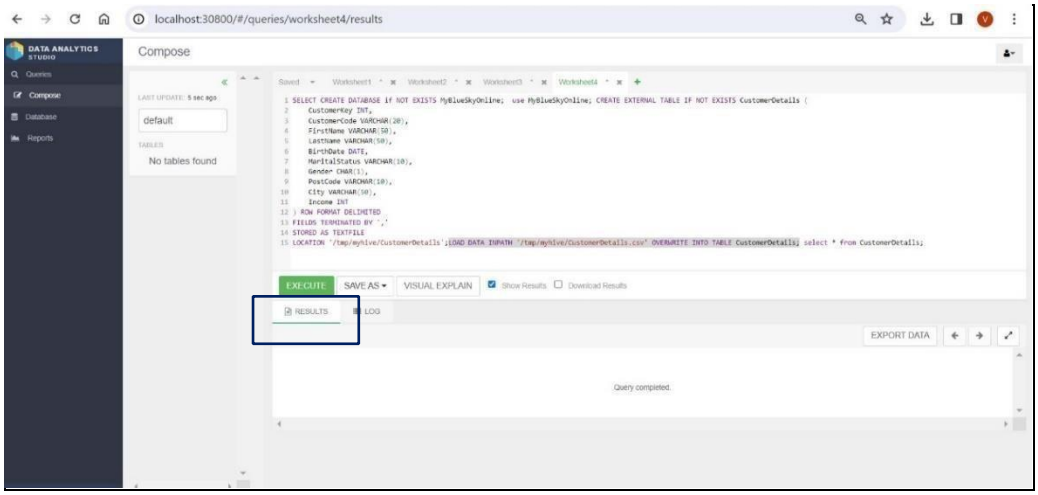


Fig: Created a Query to Populate Hive Table CustomerDetails with CustomerDetails.csv Data

After the execution of LOAD DATA, we can see that the table CustomerDetails has been filled with data from the CSV file, and Hive consumed the data file CustomerDetails.csv into these stages. CustomerDetails.csv is no longer available in the File Browser. Here the LOAD command in HIVE provides the structure for the data after it has been stored in Hadoop. This is the result we got after executing the select * from CustomerDetails command which holds CustomerCode, Firstname, Lastname, Birthdate, MartitalStatus, Gender and few more.

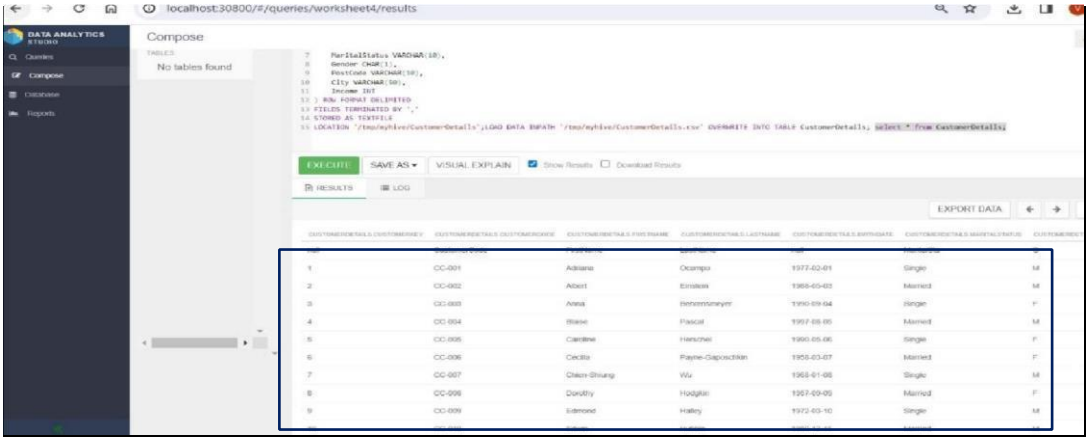


Fig: Output of Customer Details query in Hive

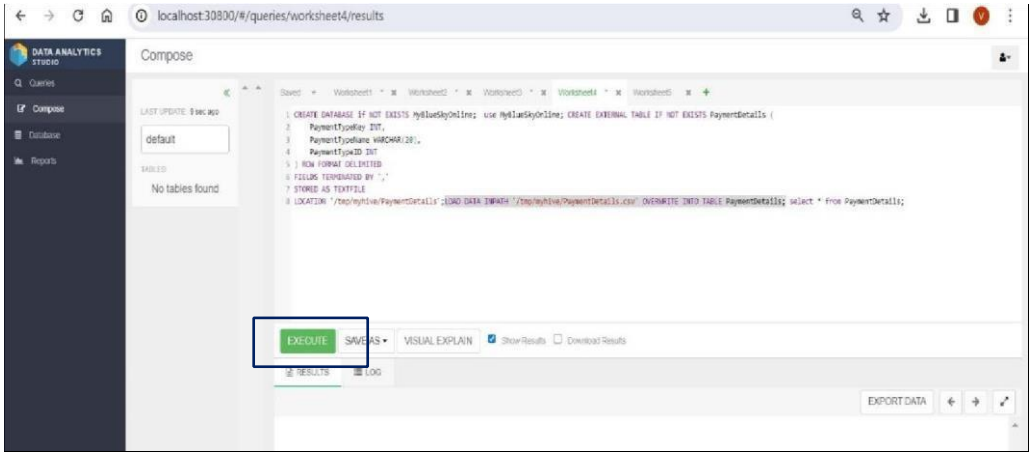


Fig: Created a Query to Populate Hive Table PaymentDetails with PaymentDetails.csv Data

Similarly for Payment Details, we created an additional table and then loaded the PaymentDetails.csv file in HDFS. By executing the above query, we have seen our results using the select * from payment details command. The results which stored in the table like PaymenttypeID and Name with paymentkeys. As previously explained, we managed the task using Hive in step by step.

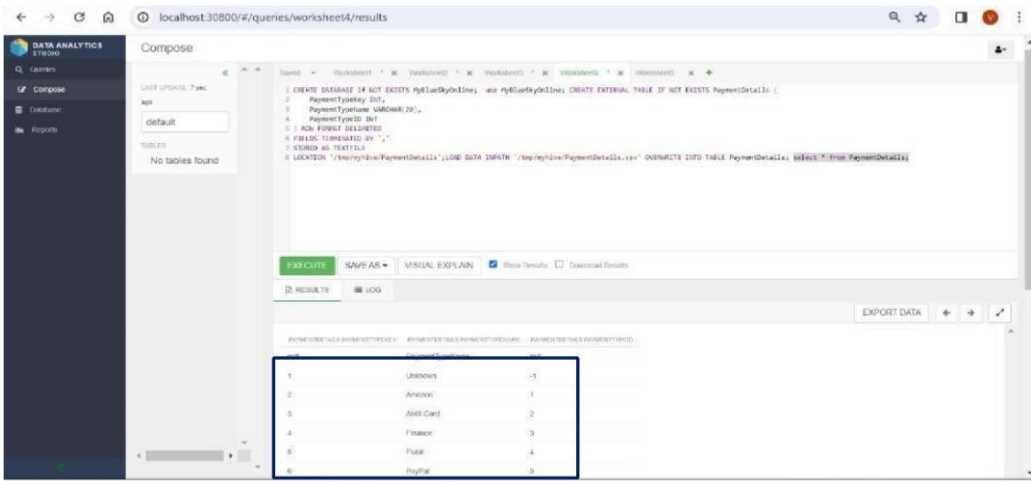


Fig: Output of PaymentDetails query in Hive

4.4 Demonstrate the use of Apache Pig for manipulating the loaded data.

Apache Pig is a high-level platform for Hadoop processing and analysis of large data sets. It abstracts the MapReduce programming design, allowing developers to express data processing tasks using Pig Latin, as a scripting language. Pig Latin is a scripting language for data flows that is used in Pig. It is similar to SQL and is intended to be easy to use. Pig scripts are used to define a set of data transformations that are carried out on Hadoop clusters. Pig Latin is a scripting language that is excellent for data processing tasks.

In the previous task, we have used Hive, a dataflow-focused scripting language. Pig and Hive are both part of the Hadoop ecosystem and provide a higher-level abstraction for data processing tasks, making it more accessible to users who may not have significant programming or MapReduce skills.

Pig Platform is developed by Yahoo, to make programming with MapReduce, Hadoop's assembler task easier. Pig Latin commands can be executed in one of two ways, either in Shell Commands (Grunt shell) to enter commands one at a time, and in Pig scripts we can create as a text file with a Pig script and run it with the Pig program. Here, the Grunt shell can be used to run Pig Latin scripts as well as HIVE and HDFS commands. Pig's logical plan does not run until a DUMP or STORE command is executed.

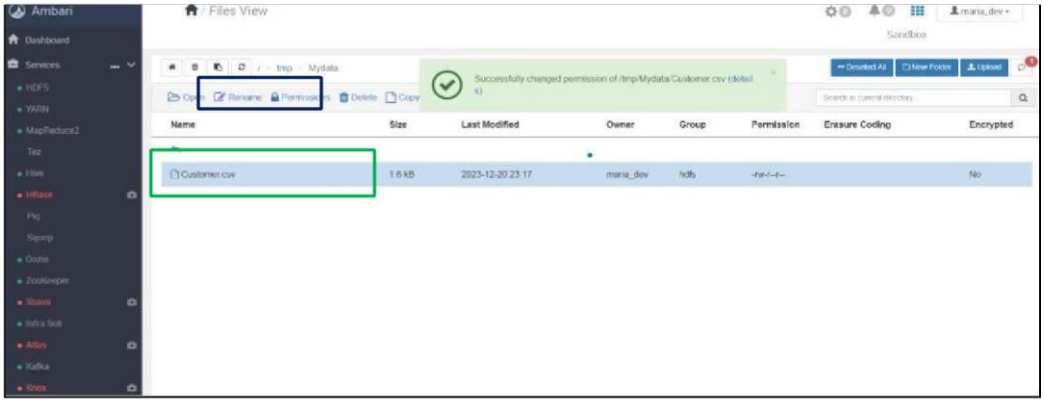


Fig: Uploaded Customer file in HDFS for Pig Task

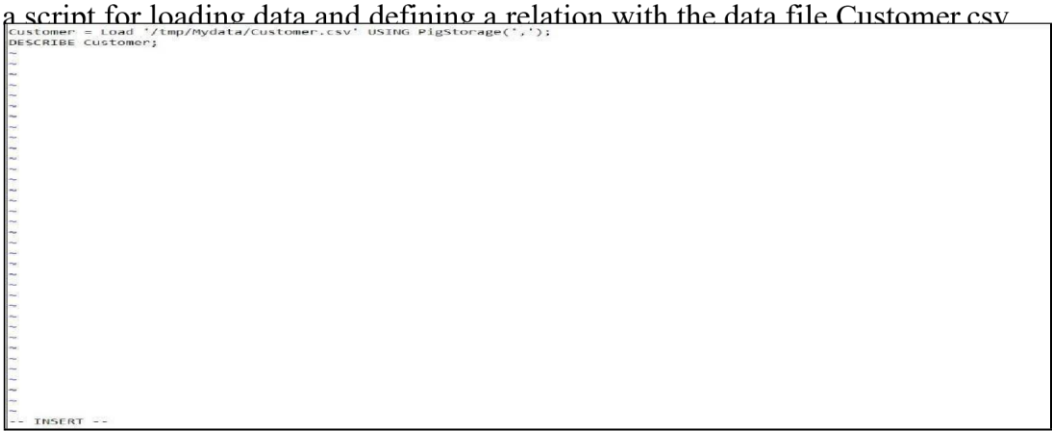
To launch Apache Pig on Hortonworks Data Platform (HDP), browse to localhost:4200 (Shellon-A-Box) and log in with root as the login and password, then change the user name to maria_dev and change the directory to home (cd). Create a file for pig script as *vi Customer* and execute it.

Fig: Creating a script to load data and define the relation

In this step, we created a pig script to load data and construct a relationship with the data file. In the first line, we created a relation named Customer, which represents customer details, and in the following line, we used the DESCRIBE command to review customer details and to

Fig: Web shell in pig with login details

To create a script for loading data and defining a relation with the data file Customer.csv



execute the script pig -f Customers then Enter before that we have to save changes by clicking esc :x.

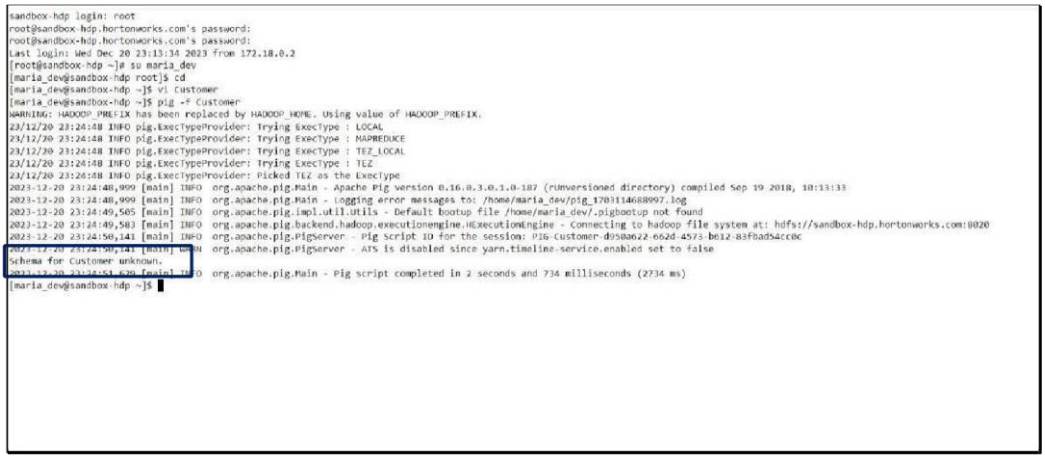


Fig: Script creates map-reduce tasks as unknown

By using the following code in the shell with the schema to give pig datatypes to the Customer and save it. Once the excution done, enter vi script.

Fig: Defining relation with

Fig: Successfully Defined scheme for the Customer Relation

In the next step, we created a new relation with the available Customer data, which is a collection of 10 entries from the Customer relation. Now save and execute the Pig query.

```
Customer = Load '/tmp/Mydata/Customer.csv' USING PigStorage(',') AS (CustomerKey: int,CustomerCode : chararray,FirstName :chararray,LastName :chararray,BirthDate :chararray,MaritalSt
us :chararray,Gender :chararray,PostCode :chararray,City :chararray,Income :long);
DESCRIBE Customer;
Customer_subset = LIMIT Customer 10;
DESCRIBE Customer_subset;
```

```
2023-12-20 23:24:51,629 [main] INFO  org.apache.pig.Main - Pig script completed in 2 seconds and 734 milliseconds (2734 ms)
[maria_dev@sandbox-hdp ~]$ vi Customer
[maria_dev@sandbox-hdp ~]$ pig -f Customer
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
23/12/20 23:43:30 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
23/12/20 23:43:30 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
23/12/20 23:43:30 INFO pig.ExecTypeProvider: Trying ExecType : TEZ_LOCAL
23/12/20 23:43:30 INFO pig.ExecTypeProvider: Trying ExecType : TEZ
23/12/20 23:43:30 INFO pig.ExecTypeProvider: Picked TEZ as the ExecType
2023-12-20 23:43:30,245 [main] INFO  org.apache.pig.Main - Apache Pig version 0.16.0.3.0.1.0-187 (rUnversioned directory) compiled Sep 19 2018, 10:13:33
2023-12-20 23:43:30,245 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/maria_dev/pig_1703115810243.log
2023-12-20 23:43:30,701 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/maria_dev/.pigbootup not found
2023-12-20 23:43:30,813 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://sandbox-hdp.hortonworks.com:8020
2023-12-20 23:43:31,343 [main] INFO  org.apache.pig.PigServer - Pig Script ID for the session: PIG-Customer-d4a284be-66b6-489c-b62f-73535a129bd6
2023-12-20 23:43:31,343 [main] WARN  org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
Customer: {CustomerKey: int,CustomerCode: chararray,FirstName: chararray,LastName: chararray,BirthDate: chararray,MaritalStatus: chararray,Gender: chararray,PostCode: chararray,City: c
hararray,Income: long}
2023-12-20 23:43:32,235 [main] INFO  org.apache.pig.Main - Pig script completed in 2 seconds and 88 milliseconds (2088 ms)
[maria_dev@sandbox-hdp ~]$ vi Customer
[maria_dev@sandbox-hdp ~]$ pig -f Customer
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
23/12/20 23:56:00 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
23/12/20 23:56:00 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
23/12/20 23:56:00 INFO pig.ExecTypeProvider: Trying ExecType : TEZ_LOCAL
23/12/20 23:56:00 INFO pig.ExecTypeProvider: Trying ExecType : TEZ
23/12/20 23:56:00 INFO pig.ExecTypeProvider: Picked TEZ as the ExecType
2023-12-20 23:56:00,596 [main] INFO  org.apache.pig.Main - Apache Pig version 0.16.0.3.0.1.0-187 (rUnversioned directory) compiled Sep 19 2018, 10:13:33
2023-12-20 23:56:00,596 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/maria_dev/pig_1703116560594.log
2023-12-20 23:56:01,277 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/maria_dev/.pigbootup not found
2023-12-20 23:56:01,388 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://sandbox-hdp.hortonworks.com:8020
2023-12-20 23:56:02,030 [main] INFO  org.apache.pig.PigServer - Pig Script ID for the session: PIG-Customer-2d7e954f-f96c-4c93-a598-2d2fdd163089
2023-12-20 23:56:02,030 [main] WARN  org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
Customer: {CustomerKey: int,CustomerCode: chararray,FirstName: chararray,LastName: chararray,BirthDate: chararray,MaritalStatus: chararray,Gender: chararray,PostCode: chararray,City: c
hararray,Income: long}
Customer_subset: {CustomerKey: int,CustomerCode: chararray,FirstName: chararray,LastName: chararray,BirthDate: chararray,MaritalStatus: chararray,Gender: chararray,PostCode: chararray,
City: chararray,Income: long}
2023-12-20 23:56:03,425 [main] INFO  org.apache.pig.Main - Pig script completed in 3 seconds and 16 milliseconds (3016 ms)
[maria_dev@sandbox-hdp ~]$
```

Fig: Successfully executed above query by defining subset Customer

To view the data Customer relation by using the DUMP command as (DUMP Customer_subset), inserting these commands in pig script, and again save and execute the code.

```
Customer_subset: {CustomerKey: int,CustomerCode: chararray,FirstName: chararray,LastName: chararray,BirthDate: chararray,MaritalStatus: chararray,Gender: chararray,PostCode: chararray,
City: chararray,Income: long}
2023-12-20 23:59:37,345 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: LIMIT
2023-12-20 23:59:37,444 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2023-12-20 23:59:37,497 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, ConstantCalculator, GroupByConstPara
llelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, PartitionFilterOptimizer, PredicatePushDownOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, St
reamTypeCastInserter]}
2023-12-20 23:59:37,944 [main] INFO  org.apache.pig.impl.util.SpillableMemoryManager - Selected heap (PS Old Gen) of size 699400192 to monitor. collectionUsageThreshold = 489580128, us
ageThreshold = 489580128
2023-12-20 23:59:38,371 [main] INFO  org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - File Output Committer Algorithm version is 2
2023-12-20 23:59:38,371 [main] INFO  org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - FileOutputCommitter skip cleanup _temporary folders under output directory:false, igno
re cleanup failures: false
2023-12-20 23:59:38,453 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2023-12-20 23:59:38,526 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2023-12-20 23:59:38,534 [main] INFO  org.apache.pig.builtin.PigStorage - Using PigTextInputFormat
2023-12-20 23:59:38,542 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2023-12-20 23:59:38,549 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
2023-12-20 23:59:38,839 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigHadoopLogger - org.apache.pig.builtin.Utf8StorageConverter(FIELD_DISCARDED_TYPE_CON
VERSION_FAILED): Unable to interpret value [67, 117, 115, 116, 111, 109, 101, 114, 75, 101, 121] in field being converted to int, caught NumberFormatException <For input string: "Custo
merKey"> field discarded
2023-12-20 23:59:38,898 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigHadoopLogger - org.apache.pig.builtin.Utf8StorageConverter(FIELD_DISCARDED_TYPE_CON
VERSION_FAILED): Unable to interpret value [73, 110, 99, 111, 109, 101] in field being converted to long, caught NumberFormatException <For input string: "Income"> field discarded
2023-12-20 23:59:39,081 [main] INFO  org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - Saved output of task 'attempt__0001_m_000001_1' to hdfs://sandbox-hdp.hortonworks.com:
8020/tmp/temp990957218/tmp1991750198
2023-12-20 23:59:39,117 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2023-12-20 23:59:39,148 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2023-12-20 23:59:39,149 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(,CustomerCode,FirstName,LastName,BirthDate,MaritalStatus,Gender,PostCode,City,)
(1,CC-001,Adriana,Ocampo ,02/01/1977,Single,Male,NW1 8GV,London,30000)
(2,CC-002,Albert,Einstein,03/05/1988,Married,Male,NE1 8GH,London,45000)
(3,CC-003,Anna,Behrensmeyer,04/09/1990,Single,Female,N8 9GV,London,23000)
(4,CC-004,Blaise,Pascal,08/05/1997,Married,Male,SE8 9GV,London,34000)
(5,CC-005,Caroline,Herschel,05/06/1990,Single,Female,N6 9YH,London,25000)
(6,CC-006,Cecilia,Payne-Gaposchkin,03/07/1958,Married,Female,EC3 0YU,London,23000)
(7,CC-007,Chien-Shiung,Wu,01/08/1968,Single,Male,E8 9YU,London,45000)
(8,CC-008,Dorothy,Hodgkin,09/09/1957,Married,Female,M1 9UU,Manchester,50000)
(9,CC-009,Edmond,Halley,03/10/1972,Single,Male,M8 9JJ,Manchester,23000)
2023-12-20 23:59:39,363 [main] INFO  org.apache.pig.Main - Pig script completed in 4 seconds and 626 milliseconds (4626 ms)
```

Fig: Executed Customer Map reduces task

This process takes a few minutes to finish, and the result should be ten fields from the Customer.csv file.

By adding the FOREACH command in the existing pig script, it defines new relations to the specific columns and remove the dump command from the query, it is no longer needed once it's executed.

Specific_columns = FOREACH Customer_subset GENERATE CustomerCode,PostCode ,Income;DESCRIBE specific_columns;

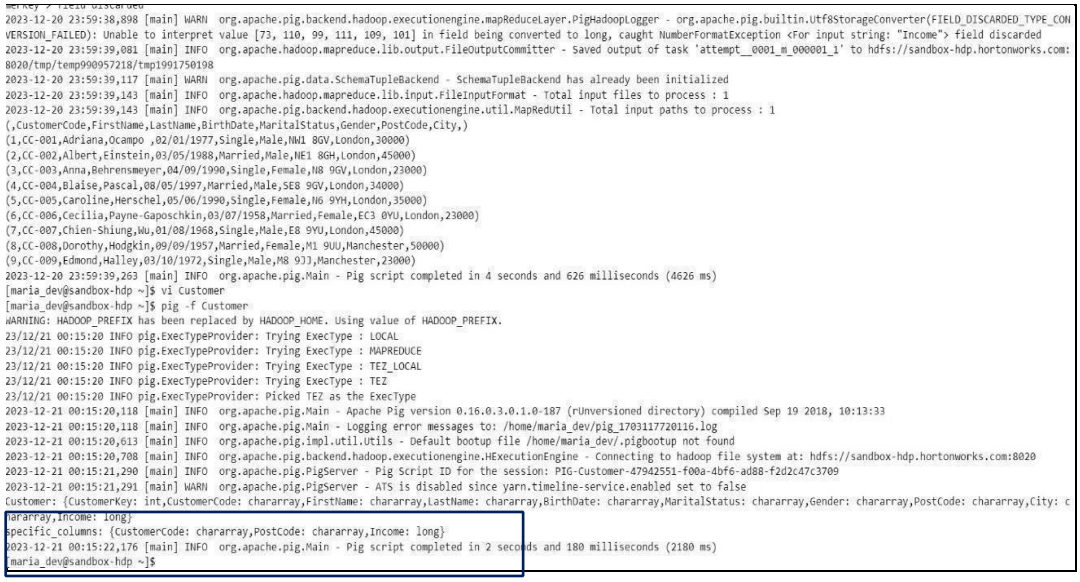


Fig: Successfully Executed Specific Columns from Customer Relation

To fulfill the requirements outlined below, store the data relation into HDFS as a file by using the STORE specific_columns INTO 'output/specific_columns using PigStorage(','); command in pig script, then save and execute it.

Fig: Storing relationship data in HDFS

Once it has been executed, we have to go to HDFS Files View to view the output, which is saved under the user as maria_dev. We need to check for a newly created folder named "Output"

.

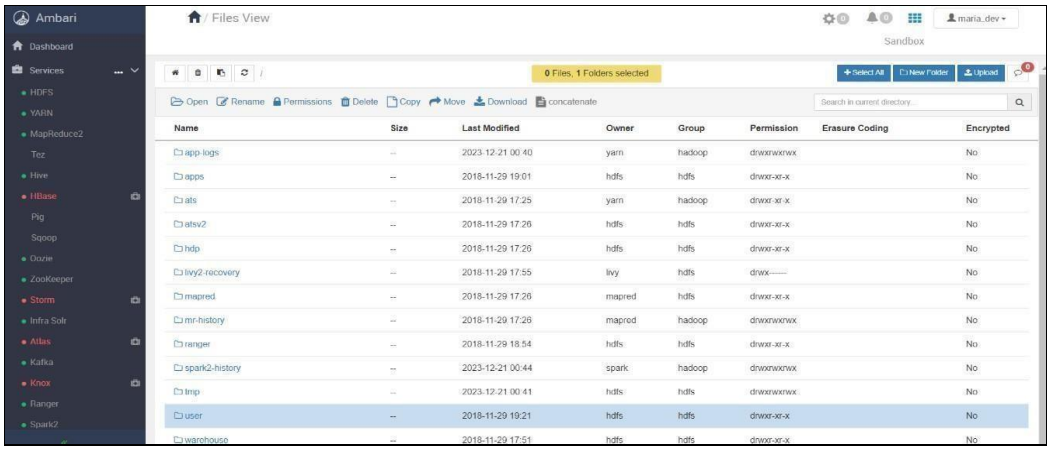


Fig: Navigating the "User" folder in the HDFS file view

Now click on the **Output** folder to see the sub folder named as Specific_columns.

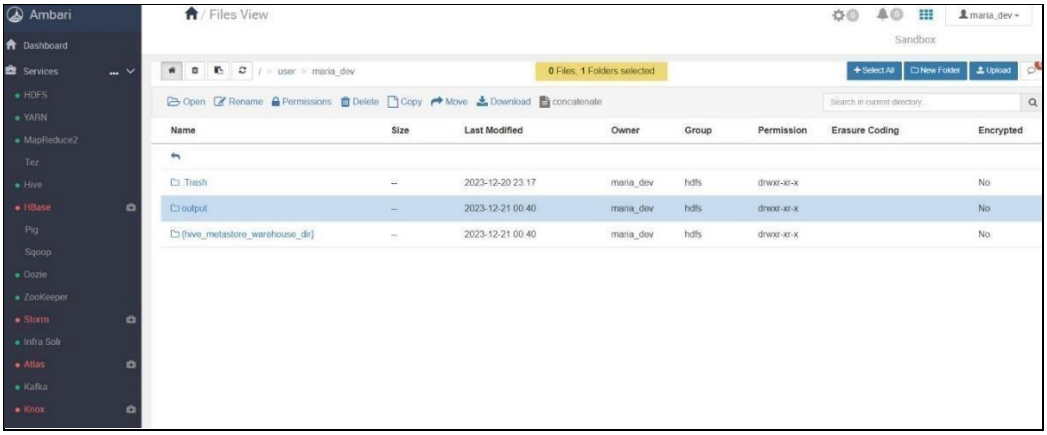


Fig: Output for Customer folder created in HDFS

By clicking the Output folder, we can see the specific _columns folder. The output is named "**part-v001-o000-r-00000"** in that folder.

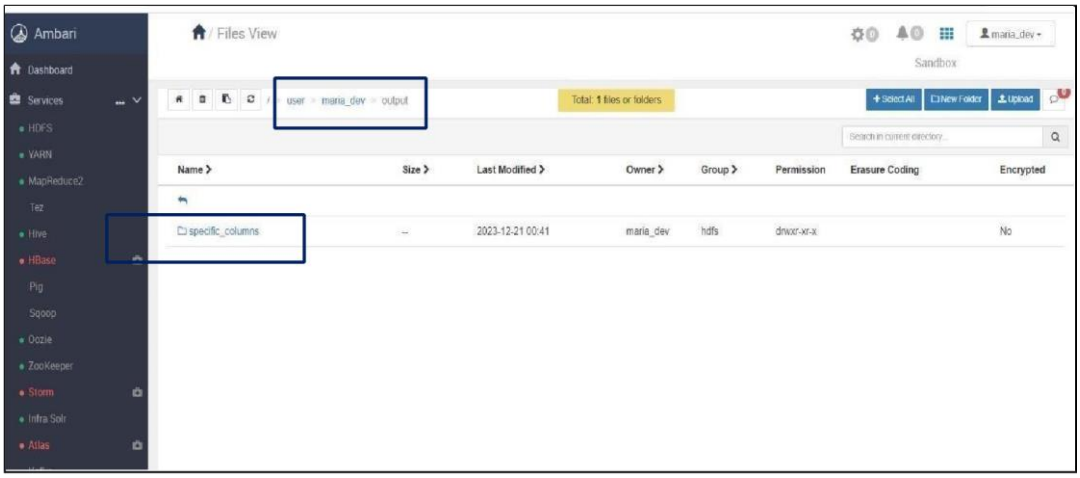
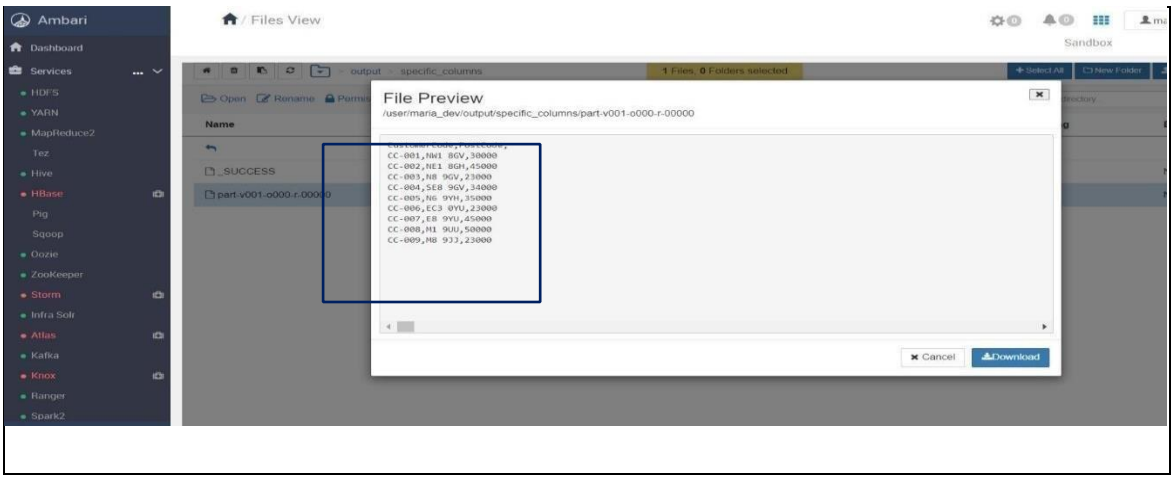


Fig: Specific Column created in HDFS

Once we click on the "part-v001-o000-r-00000" file and then Open we can see output by clicking on Open.



Conclusion:

In conclusion, the Blue-Sky Online Retailer project provided me with great insights into the field of big-data processing using technologies such as SSMS, HDFS, Pig, and Hive in the context of Apache Hadoop. From a technical point of view, I have developed an in-depth understanding of Pig and Hive's features, knowing that they are feature-rich and capable of completing tasks in the big data ecosystem.

One important takeaway from this case study is the flexibility and choice provided by these technologies. The fact that organizations may choose from Pig and Hive based on their individual needs and preferences demonstrates Apache Hadoop's versatility in fulfilling a wide range of objectives. Having the ability to choose between tools improves the simplicity and adaptability of big data processing by allowing businesses to modify their strategy to their specific data processing challenges.

The hands-on experience that I got from this project in uploading data into HDFS files and reviewing data using various Hive queries has helped me to develop my abilities. The usage of queries like CREATE, LOAD, INSERT, SELECT, FROM, and GROUP has given me a good foundation in handling and querying large data sets. The actual implementation of these queries in creating and handling tables for customer and payment data has deepened my understanding of real-world scenarios in big data processing.

Furthermore, working on tasks such as creating tables for customers as well as extracting data from databases, and sorting outcomes based on specified criteria has provided me with a practical understanding of how these technologies can be used to extract useful insights from enormous datasets. The project not only taught me technical skills, but also highlighted the value of proper data management and extraction for making accurate decisions.

Reflecting on my experience working in an agile setting, I have learned to appreciate the iterative and collaborative aspects of agile approach phases. The courses required adaptation, rapid problem-solving, and effective team communication. These abilities are not only important in the team, but they are also highly transferable to real-world professional circumstances where flexibility and teamwork are essential for success.

Overall, by doing this project I have gained a rewarding experience, which taught me technical skills, practical insights, and a deeper understanding of the flexible work environment. The information learned will surely help me manage and contribute effectively to the rapidly changing world of big data and analytics.