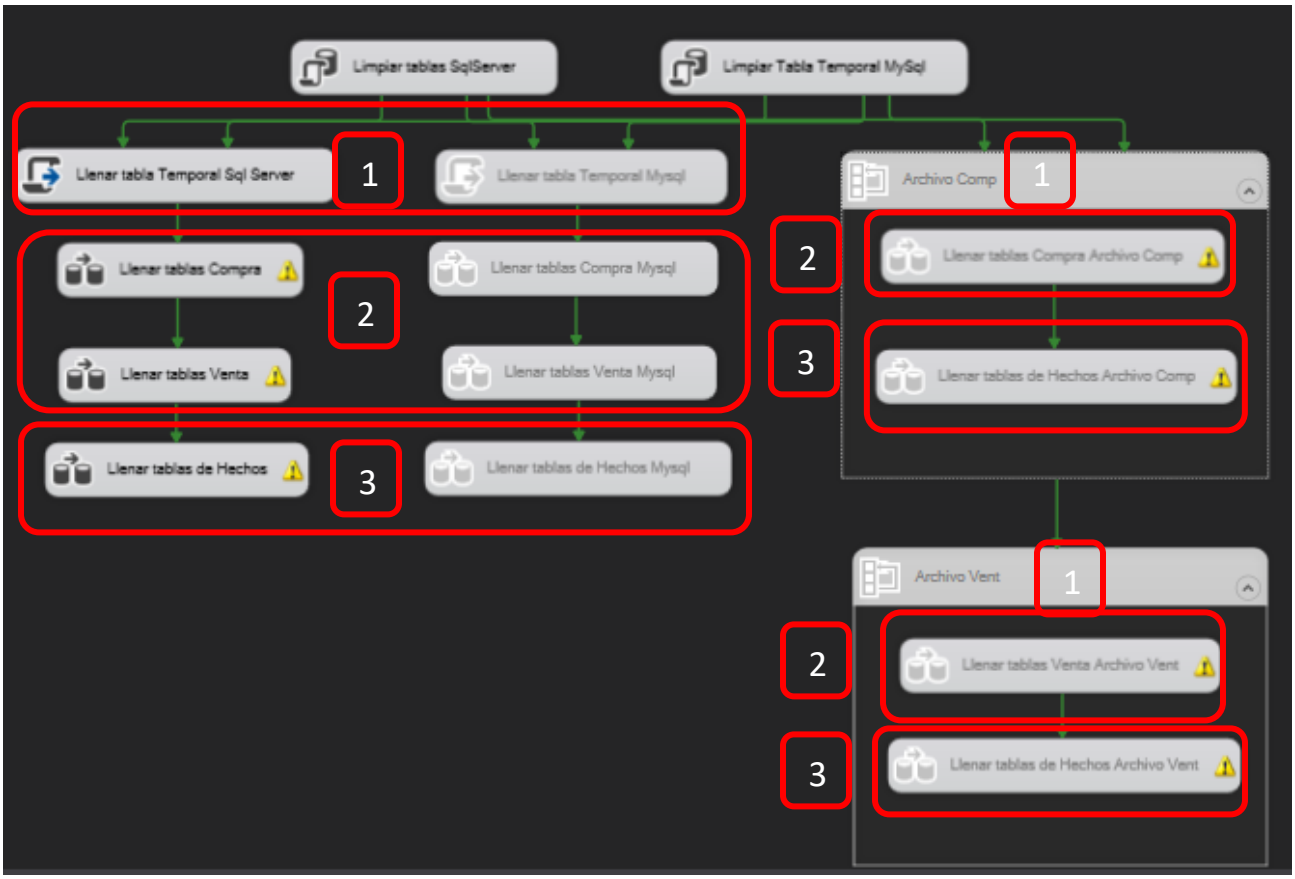
GuateFood



Este grafico representa las siguientes partes donde se aplicó **ETL**

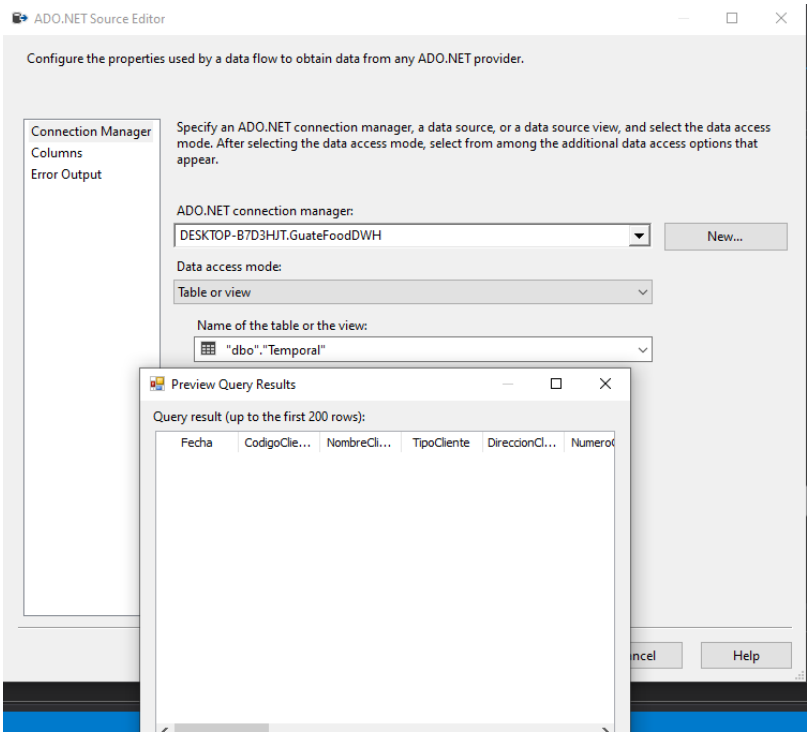
1) Extract
2) Transform
3) Load

Extract

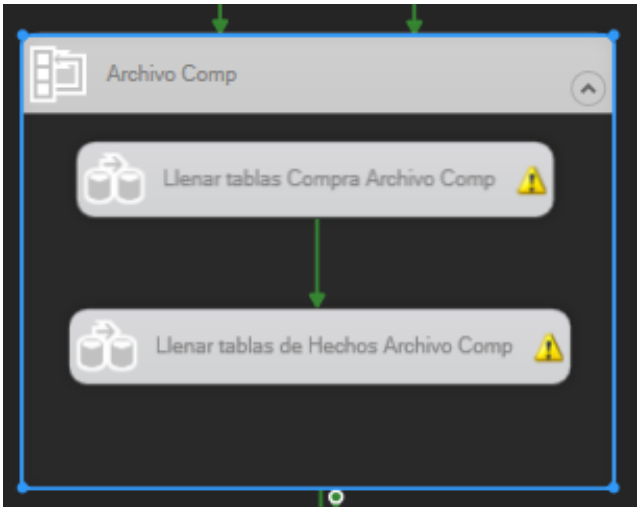
Para la parte de la extracción de datos se utiliza **SCRIPT TASK** que nos sirve para cargar todos los datos a la tabla temporal que están ubicados en SqlServer y MySql. Aquí escogemos nuestra variables que se crearon anteriormente donde se guardan tanto las rutas y extensiones de los archivos. Por ultimo editamos el script que es en código C# y escribimos el código necesario para leer los archivos y meterlos en sus respectivas tablas temporales.

```
//========================= PRIMERO METO LOS DATOS DE LOS ARCHIVOS VENTA ========================
string[] fileEntries = Directory.GetFiles(FolderOrigen, "GuateFood*" + VentExtension);
foreach (string fileName in fileEntries)
{
    SqlConnection myADONETConnection = new SqlConnection();
    myADONETConnection = (SqlConnection)(Dts.Connections[ ConexionSqlServer ].AcquireConnection(Dts.Transactio

    //string connStr = "server=localhost;user=root;database=seminariodos201602723;port=3306;password=root";
    //MySqlConnection myADONETConnection2 = new MySqlConnection(connStr);
    //myADONETConnection2.Open();

    int counter = 0;
    string line;
    System.IO.StreamReader SourceFile = new System.IO.StreamReader(fileName);
    while ((line = SourceFile.ReadLine()) != null)
    {
        if (counter > 0) // SE IGNORA LA PRIMER LINEA, USUALMENTE USADO COMO CABECERA
        {
            string[] campos = line.Split(Delimitador.ToCharArray()[0]);
            string query = "INSERT INTO " + tablasdestino
                + "(Fecha,CodigoCliente,NombreCliente,TipoCliente,DireccionCliente," +
                "NumeroCliente,CodVendedor,NombreVendedor,Vacacionista,CodProducto," +
                "NombreProducto,MarcaProducto,Categoria,SodSuSursal,NombreSucursal," +
                "DireccionSucursal,Region,Departamento,Unidades,PrecioUnitario) VALUES('"
                + campos[0] + "','" + campos[1] + "','" + campos[2] + "','" + campos[3] + "','" + campos[4] +
                + campos[5] + "','" + campos[6] + "','" + campos[7] + "','" + campos[8] + "','" + campos[9] +
                + campos[10] + "','" + campos[11] + "','" + campos[12] + "','" + campos[13] + "','" + campos[1
                + campos[15] + "','" + campos[16] + "','" + campos[17] + "','" + campos[18] + "','" + campos[1
                + "')";
            SqlCommand myCommand = new SqlCommand(query, myADONETConnection);
            myCommand.ExecuteNonQuery();
```

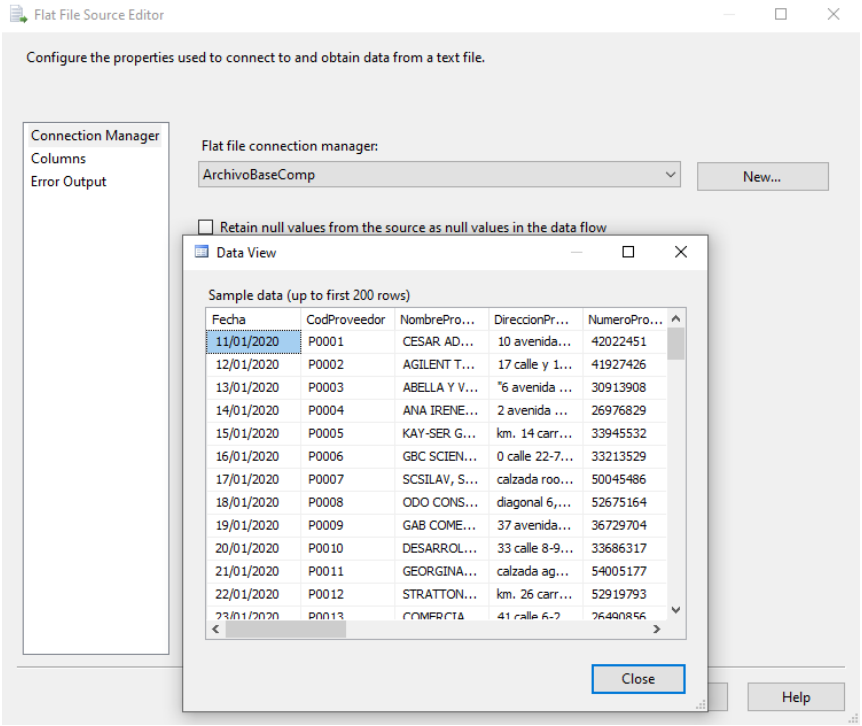
Para utilizar los datos que almacenamos en nuestra tabla Temporal se usa el componente **ADO NET Source** donde lo único que hacemos es escoger la conexión de la base de datos que deseamos usar y escogemos su tabla. Si todo sale bien nos mostrara la tabla temporal con todos sus datos almacenas (si es que tiene) es un vista previa



Ahora para obtener los datos y utilizarlos directamente de los archivos se usara el componente **Foreach loop container**. Este componente nos servirá para poder usar todos los archivos de una carpeta que le especifiquemos, se ejecutara n-veces dependiendo la cantidad de archivos tenga la carpeta. Este componente se ayuda de un archivo base, para saber la estructura de los archivos.

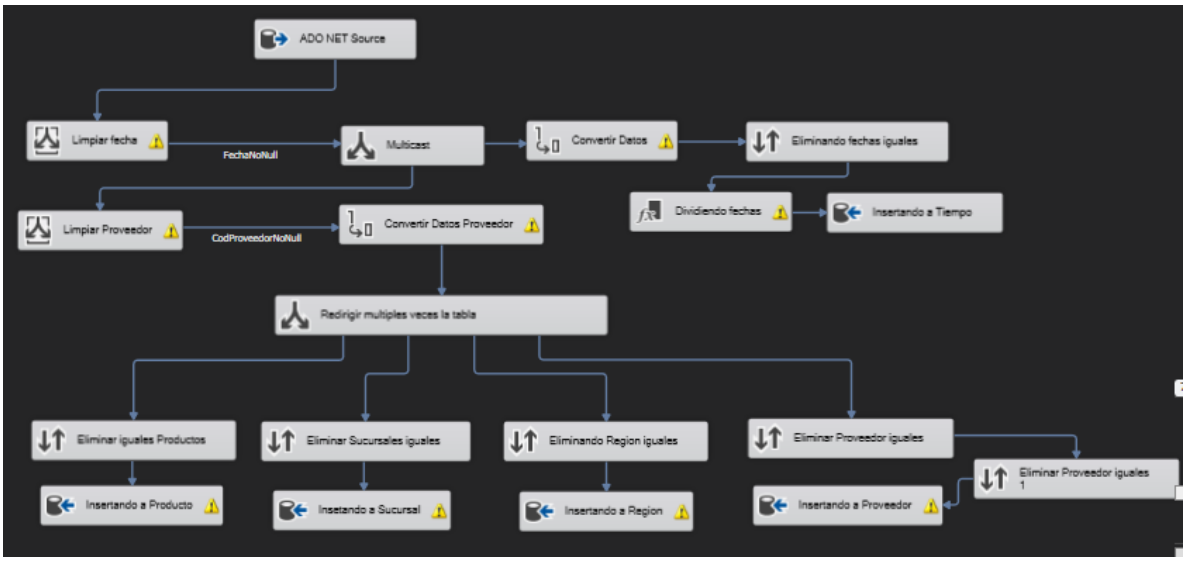


Para utilizar los datos de los archivos es similar a cuando se hacía con tablas temporales, solo que el componente adecuado es **Flat File Source** donde lo único que hacemos es escoger la conexión de nuestros archivos. Si todo sale bien nos mostrara los datos guardados (si es que tiene) es un vista previa. Este componente solo se puede usar dentro de **Data Flow Task**



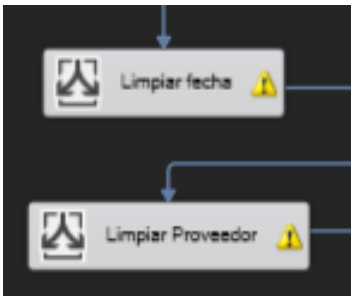
Transform

Una vez ya cargados los datos es hora de transfórmalos antes de cargar los datos a sus respectivas tablas dimensiones. A continuación se muestra la transformación que se realizó en los archivos de compra.



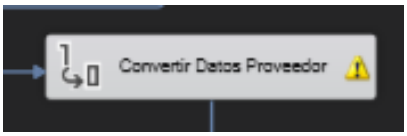
Para las transformaciones de datos, se utilizaron los siguientes componentes:

- **Conditional Split:** Este componente sirve para evaluar una condición y comenzar a filtrar algunos datos. Es comparable a la condición **if** en el mundo de la programación. En nuestro caso servirá para para eliminar los campos Nulos o vacíos y dejar pasar las líneas correctas, así de una vez se comienzan a eliminar las filas que tienen este problema



- **Derived Column**: Este Componente sirve para hacer alguna operación con algún campo. Se pueden hacer operaciones para datos de tipo String, Date, Numéricos, etc. Cuando realizamos alguna operación este módulo nos crea una nueva columna, en donde estará el resultado de nuestras operaciones y por eso es importante identificar adecuadamente la nueva columna.

- **Data Conversion:** Este componente sirve para convertir el tipo de dato a otro de un campo. Puede ser como convertir un string a int, string a date, int a string y así sucesivamente. Al igual que Derived Column esta genera una nueva columna con el dato ya parseado, por eso es importante identificar adecuadamente la nueva columna.



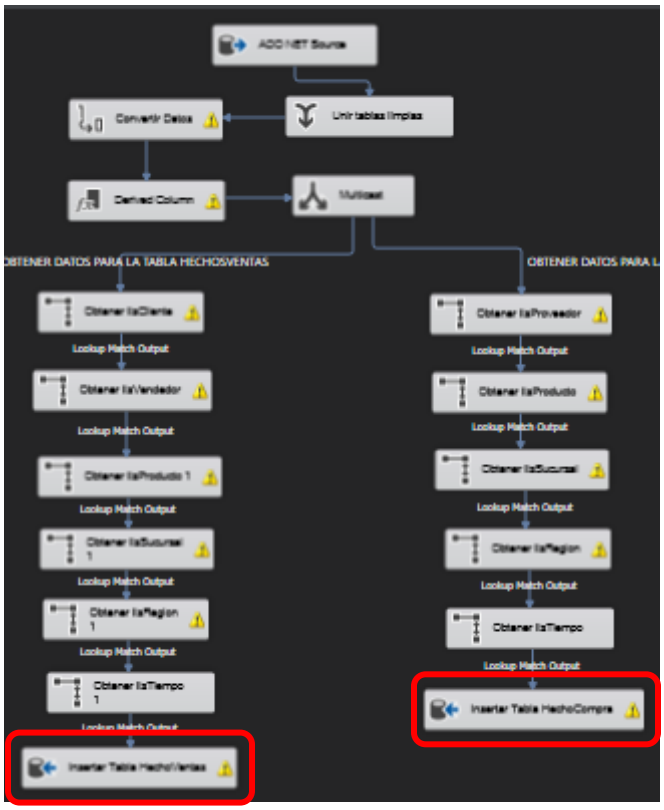
- **Multicast:** Este componente sirve para redirigir una salida a múltiples salida, esto es para usar esa misma salida a diferentes componentes.

- **Sort:** Este componente sirve para ordenar los datos a base de un campo seleccionado, pero lo más importante quitar filas con identificador duplicado.

- **Lookup:** Este componente sirve para buscar datos ya almacenados del Datawarehouse que está almacenado en la base de datos. Es de ayuda ya que a veces necesitamos saber si algún dato con un identificador único, ya fue almacenado en el Datawarehouse previamente y así evitar datos duplicados. Al final usamos los datos que no tuvieran relación con los ya guardados. También nos sirve para almacenar datos a las tablas de hechos ya que nos puede devolver la llave subrogada de dicha fila y así poder usarla.

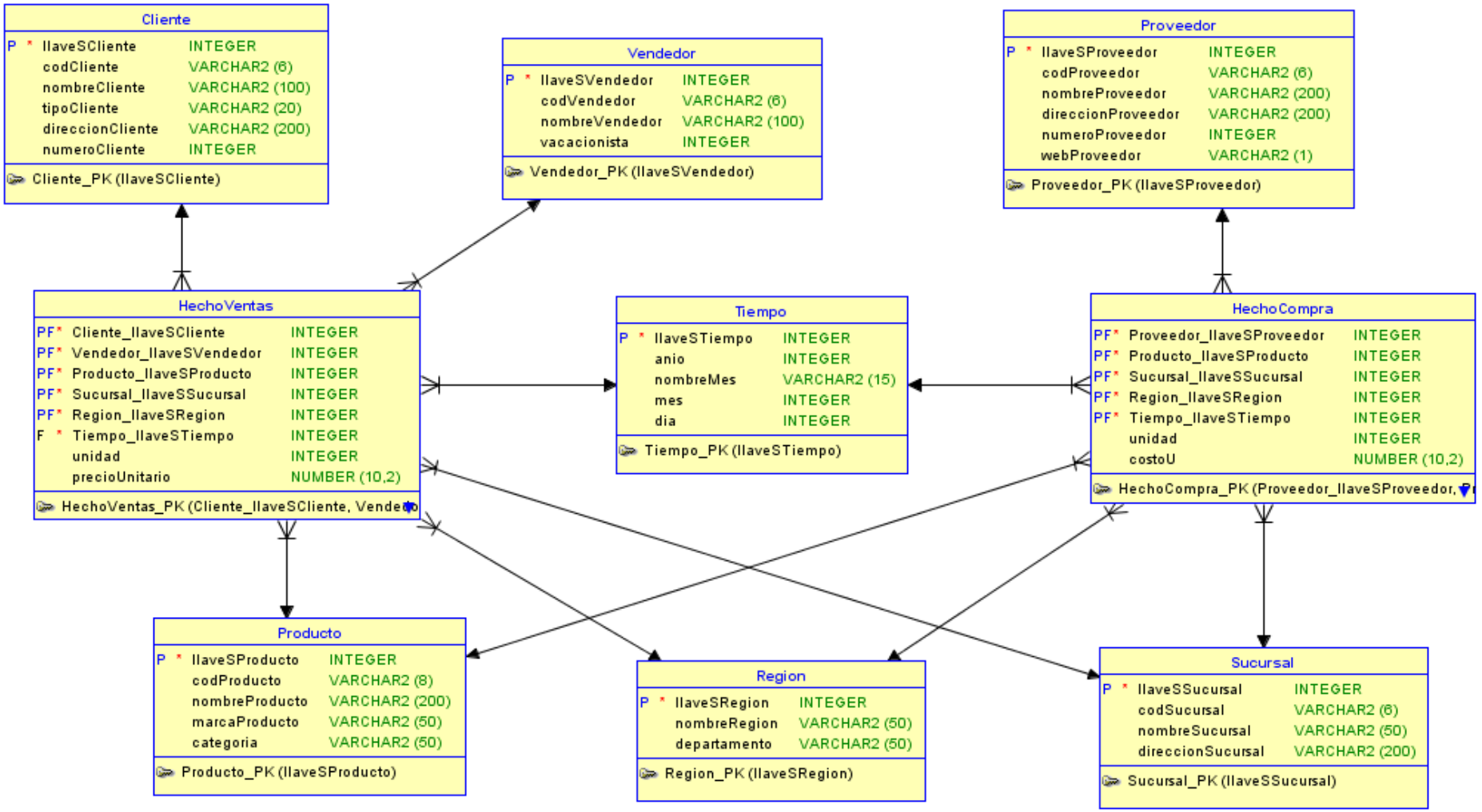
Load

Una vez ya cargados y transformados los datos solo nos falta guardarlos a las tablas respectivas dentro del Datawarehouse. A continuación se señala la carga de los datos transformados y se carga a las tablas de hechos.



- **ADO Net Destination:** Este componente nos sirve para carga los datos transformados a las sus respetivas tablas dentro del Datawarehouse. Es parecido a **ADO NET Source** solo que esta en vez de consulta a guardar datos. Aquí hacemos un mapeo de los datos que tenemos y a donde se van a ir a almacenar dentro de la tabla.

Modelo del Datawarehouse


Cliente
P * llaveSCliente INTEGER
codCliente VARCHAR2 (6)
nombreCliente VARCHAR2 (100)
tipoCliente VARCHAR2 (20)
direccionCliente VARCHAR2 (200)
numeroCliente INTEGER
Cliente_PK (llaveSCliente)
Vendedor
P * llaveSVendedor INTEGER
codVendedor VARCHAR2 (6)
nombreVendedor VARCHAR2 (100)
vacacionista INTEGER
Vendedor_PK (llaveSVendedor)
Proveedor
P * llaveSProveedor INTEGER
codProveedor VARCHAR2 (6)
nombreProveedor VARCHAR2 (200)
direccionProveedor VARCHAR2 (200)
numeroProveedor INTEGER
webProveedor VARCHAR2 (1)
Proveedor_PK (llaveSProveedor)
HechoVentas
PF* Cliente_llaveSCliente INTEGER
PF* Vendedor_llaveSVendedor INTEGER
PF* Producto_llaveSProducto INTEGER
PF* Sucursal_llaveSSucursal INTEGER
PF* Region_llaveSRegion INTEGER
F * Tiempo_llaveSTiempo INTEGER
unidad INTEGER
precioUnitario NUMBER (10,2)
HechoVentas_PK (Cliente_llaveSCliente, Vende
Tiempo
P * llaveSTiempo INTEGER
anio INTEGER
nombreMes VARCHAR2 (15)
mes INTEGER
dia INTEGER
Tiempo_PK (llaveSTiempo)
HechoCompra
PF* Proveedor_llaveSProveedor INTEGER
PF* Producto_llaveSProducto INTEGER
PF* Sucursal_llaveSSucursal INTEGER
PF* Region_llaveSRegion INTEGER
PF* Tiempo_llaveSTiempo INTEGER
unidad INTEGER
costoU NUMBER (10,2)
HechoCompra_PK (Proveedor_llaveSProveedor,
Producto
P * llaveSProducto INTEGER
codProducto VARCHAR2 (8)
nombreProducto VARCHAR2 (200)
marcaProducto VARCHAR2 (50)
categoria VARCHAR2 (50)
Producto_PK (llaveSProducto)
Region
P * llaveSRegion INTEGER
nombreRegion VARCHAR2 (50)
departamento VARCHAR2 (50)
Region_PK (llaveSRegion)
Sucursal
P * llaveSSucursal INTEGER
codSucursal VARCHAR2 (6)
nombreSucursal VARCHAR2 (50)
direccionSucursal VARCHAR2 (200)
Sucursal_PK (llaveSSucursal)

Justificación

Para este problema se optó por un modelo **Starflake,** porque se necesitaba dos tablas de hechos para diferenciar los datos de compra y los datos de venta. Primero se pensó por un modelo de estrella tradicional pero los datos que se tiene en las ventas y compras son ligeramente distintos y no podrían existir en una misma tabla de hechos. Se pensó en agregar una columna “Tipo” para poder identificar de una de la otra pero seguían teniendo el mismo problema ya que esta ligereza de datos podría dejar incluso llaves subrogadas vacías en la tabla de hechos, lo cual no estaría permitido. El único modelo que permitía tener más de una tabla de hechos es el ya mencionado Starflake, ya que es una combinación del modelo estrella y Snowflake. Al final el modelo termino con dos tablas de hechos, una para representar los datos de las ventas y la otra para representar los datos de las compras. Las tablas de dimensiones todas están separadas y ninguna de ellas se relaciona con otra de dimensiones, solo con las tablas de hechos. Esto se hizo así para que el modelo siga mantenido rapidez a la hora de consultas ya que solo tendría una relación y por esa razón se evitó relacionarla con otra tabla de dimensión para evitar una posible