Northwind Database
Extraction Transformation Load

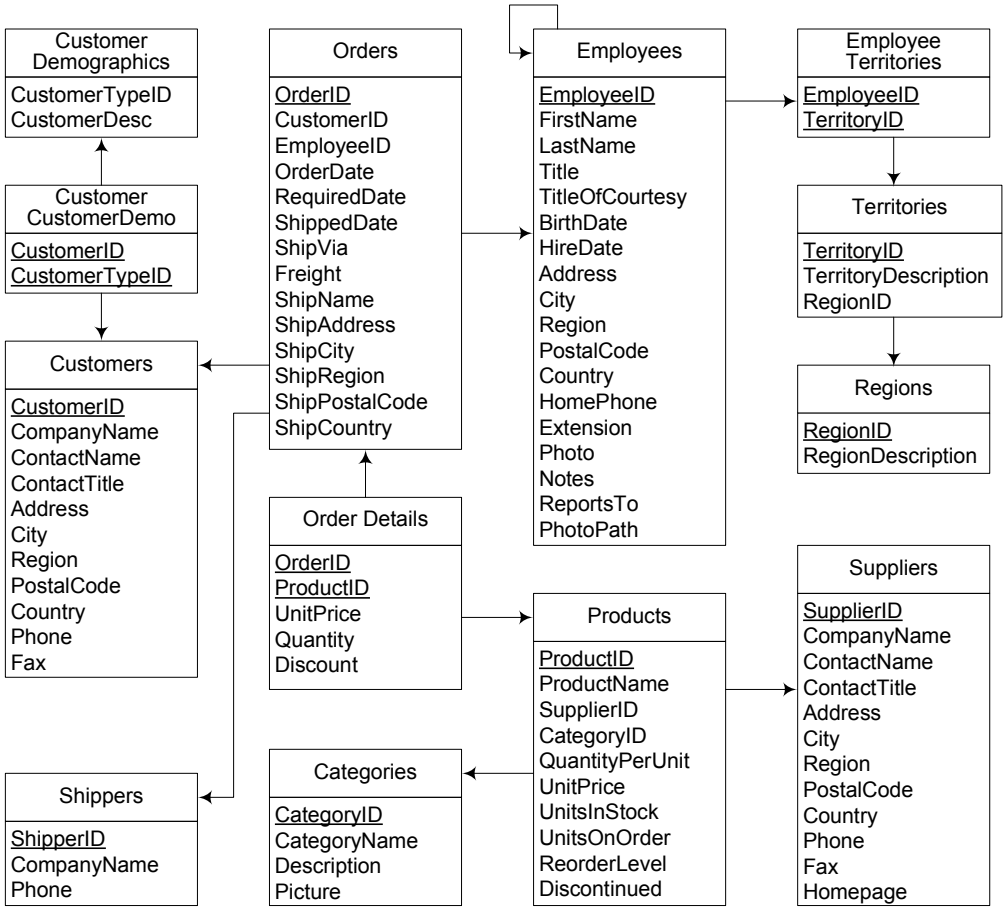


Figure 1: Schema of the Northwind operational database

Figures 1 and 2 represent, respectively, the schema of an operational database and the schema of its associated data warehouse. Implement in Integration Services 2005 the ETL process that allows to load the data warehouse.

For the dimension `DimTime`, create an Excel file that will contain the data. The time interval of this dimension must cover the dates contained in the table `Orders` of the operational database.

The data for the hierarchy `DimState`, `DimCountry`, `DimArea` is input from an XML file called `Territories.xml` that begins as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Areas>
  <Area>
    <AreaName>Europe</AreaName>
    <Country>
      <CountryName>Austria</CountryName>
      <CountryCode>AT</CountryCode>
      <CountryCapital>Vienna</CountryCapital>
```

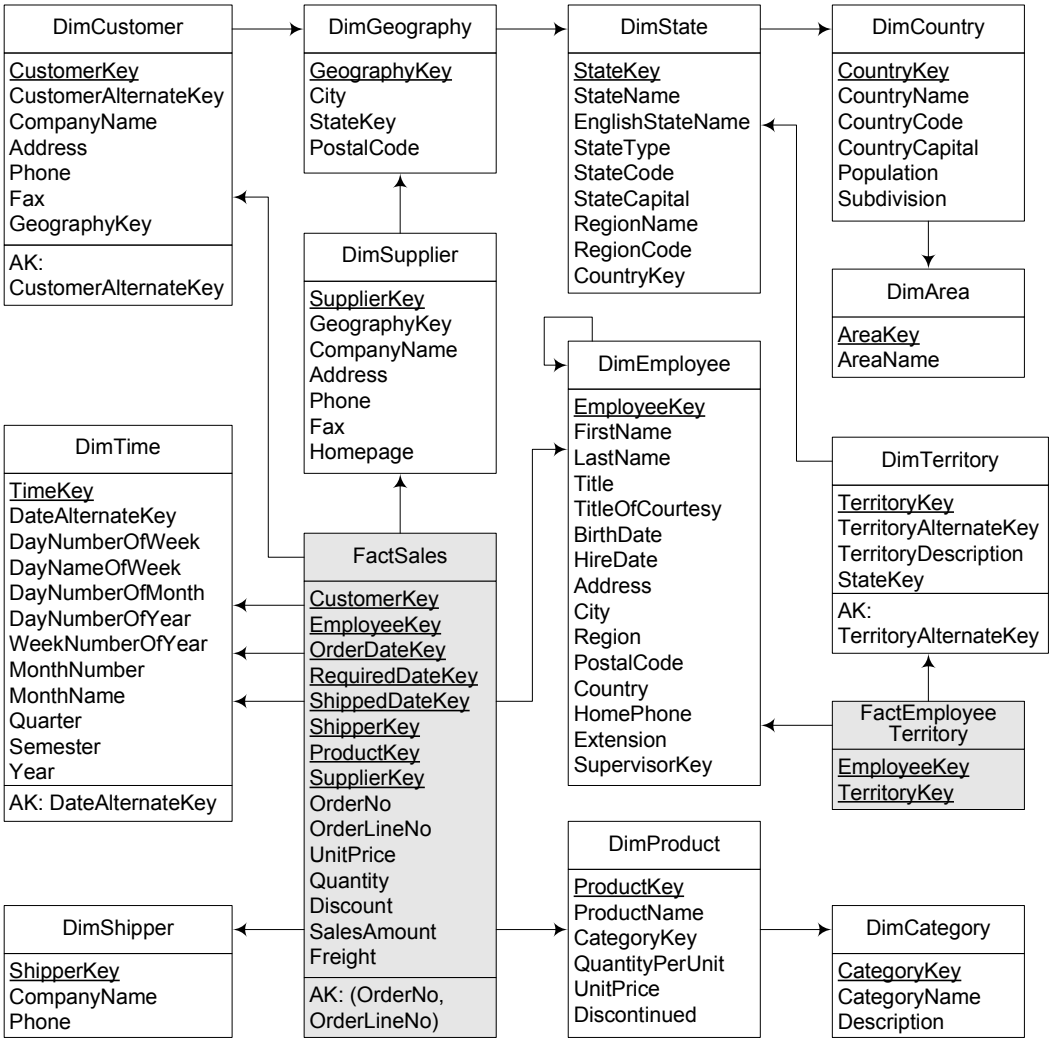


Figure 2: Schema of the Northwind data warehouse

```
<Population>8316487</Population>
<Subdivision>Austria is divided into nine Bundesländer,
  or simply Länder (states; sing. Land).</Subdivision>
<State type="state">
  <StateName>Burgenland</StateName>
  <StateCode>BU</StateCode>
  <StateCapital>Eisenstadt</StateCapital>
</State>
<State type="state">
  <StateName>Kärnten</StateName>
  <StateCode>KA</StateCode>
  <EnglishStateName>Carinthia</EnglishStateName>
  <StateCapital>Klagenfurt</StateCapital>
</State>
...
```

The schema of the XML file is shown in Figure 3. Notice that `type` is an attribute of

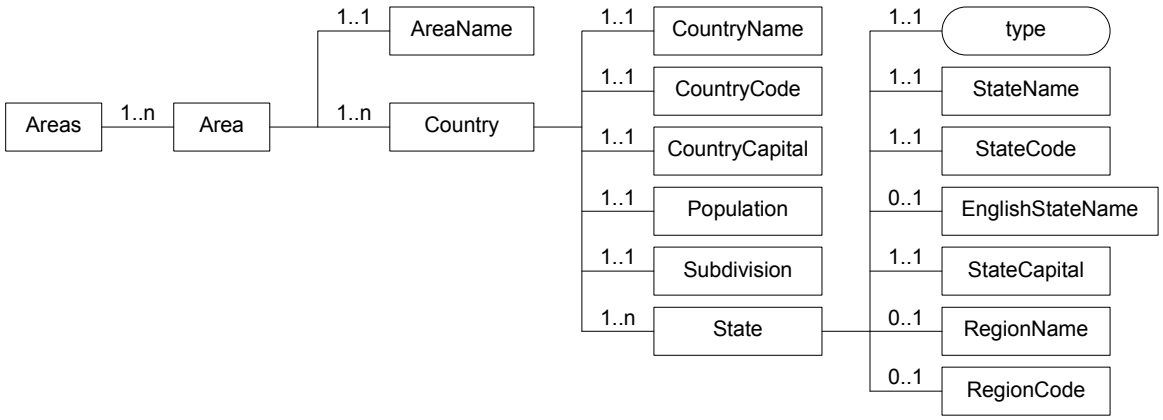


Figure 3: XML Schema of the file `Territories.xml`

`State` and that `EnglishStateName`, `RegionName`, and `RegionCode` are optional.

The `DimGeography` dimension is obtained from the `City`, `Region`, `PostalCode`, and `Country` from both `Customers` and `Suppliers`. Notice that the attribute `Region` contains in fact a state name (e.g., `Québec`) or a state code (e.g., `CA`); similarly, the attribute `Country` contains a country name (e.g., `Canada`) or a country code (e.g., `USA`). To identify to which state corresponds a city the file `cities.txt` is used. The file contains three fields separated by tabs and begins as follows:

```
Atlanta → Georgia → USA
Austin → Texas → USA
Beachwood → Ohio → USA
Bedford → Indiana → USA
Bellevue → Kentucky → USA
...
```

This file is also used to identify to which state correspond the attribute `TerritoryDescription` in `DimTerritory`, which in fact contains city names from the United States.

For the `FactSales` table, the following transformations are needed.

- The `OrderLineNo` must be generated in ascending order of `ProductID`.
- The `SalesAmount` must be calculated taking into account the unit price, the discount, and the quantity.
- The `Freight`, which in the operational database is related to the whole order, must be equally distributed among the lines of the order.

Integration Services 2005 Solution

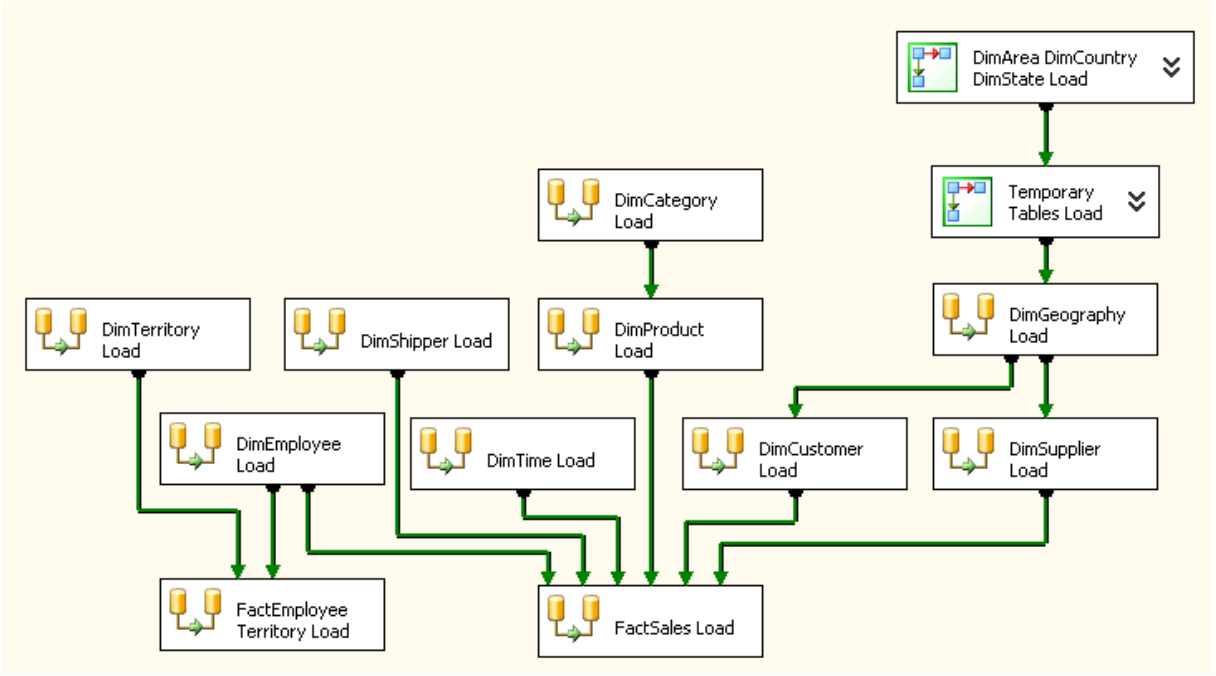


Figure 4: Overall view of the ETL process

Figure 4 shows the overall ETL process. It is composed of ten data flow tasks and two sequence container tasks connected by precedence constraints.

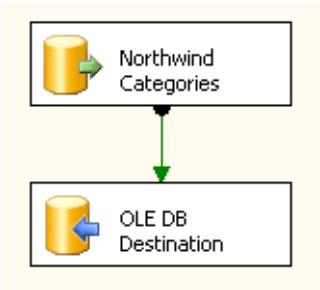
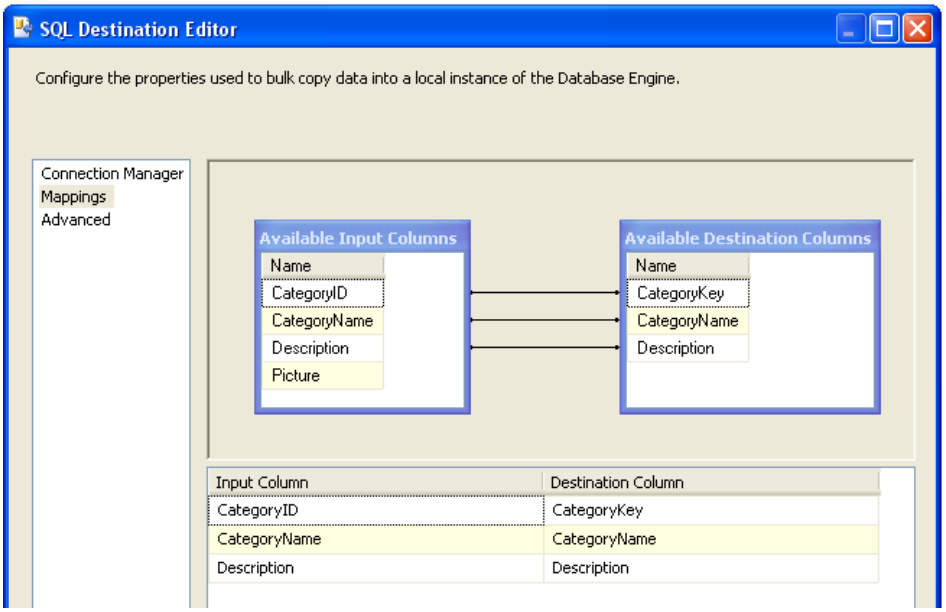


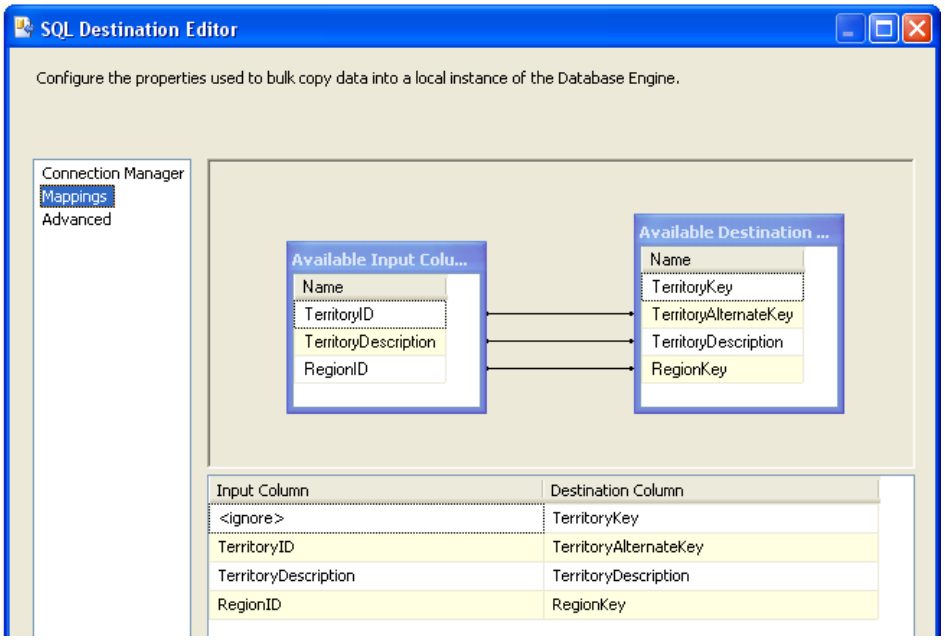
Figure 5: Loading of the `DimCategory` dimension

Most data flow tasks are simple, an example for loading the `DimCategory` is given in Figure 5. These data flow tasks are composed of an OLE DB Source task that reads the entire table from the operational database and an SQL Server Destination task that receives the output from the previous task.

In these simple data flow tasks it is necessary to determine whether the operational database already contains a key that can be reused in the data warehouse. This is the case for all dimensions that do not have an alternate key. Depending on whether the key can be reused in the data warehouse, the mapping of colums in the SQL Server Destination tasks should be done as shown in Figure 6 (a) or (b).



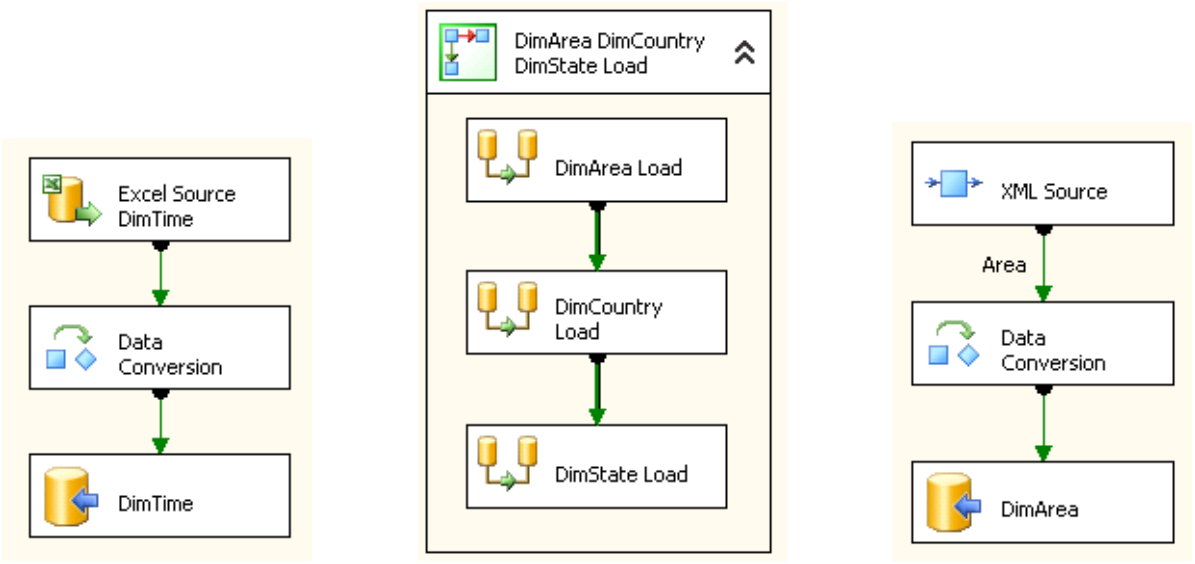
(a)



(b)

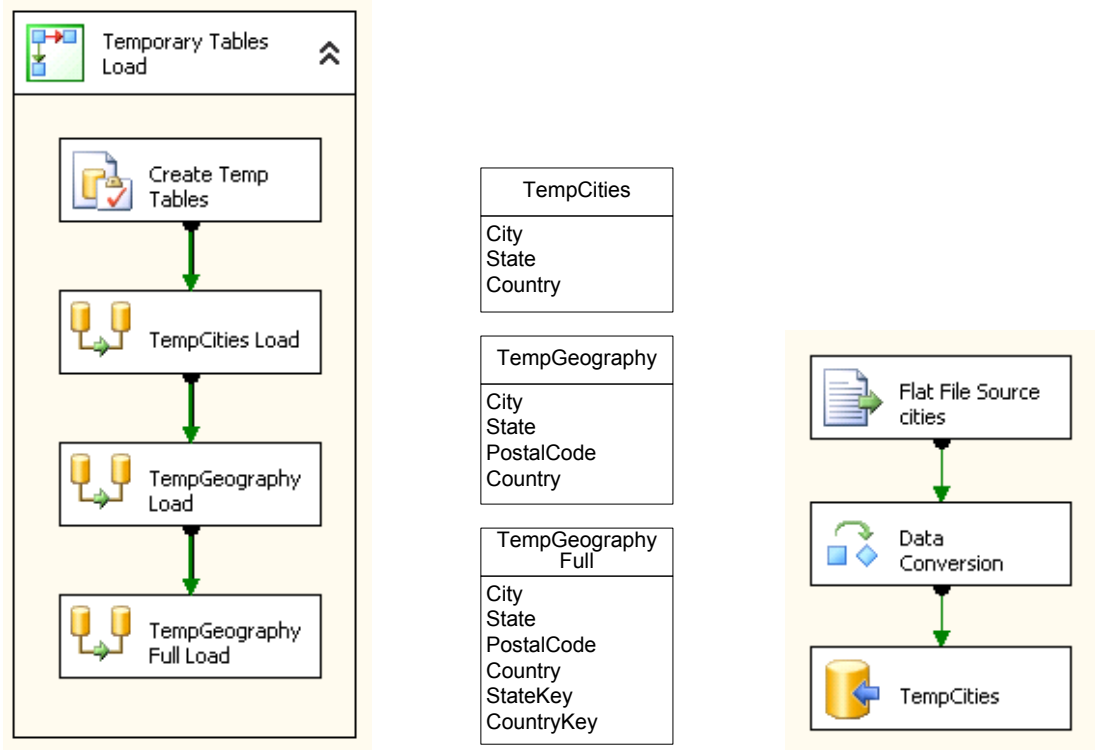
Figure 6: Mappings of the source and destination columns, depending on whether the key in the operational database is reused in the data warehouse

The data flow task that loads the `DimTime` dimension is shown in Figure 7 (a). After loading the source Excel file, a data conversion transformation is needed to convert the data types from the Excel file into the data types of the database. Figure 7 (b) shows the data flow task that loads the hierarchy composed of `DimArea`, `DimCountry` and `DimState`. A Sequence Container is used for the three data flows that load the tables of the hierarchy. The data composed for loading the table `DimArea` is given in Figure 7 (c). Conversion



(a) (b) (c)

Figure 7: (a) Loading of the `DimTime` dimension; (b) Loading of the hierarchy composed of `DimArea`, `DimCountry`, and `DimState`; (c) Loading of the `DimArea` dimension



(a) (b) (c)

Figure 8: (a) Loading of the temporary tables needed for the `DimGeography` dimension; (b) Structure of the temporary tables; (c) Loading of the `TempCities` table

transformations are needed to convert the data types from the XML file into the data types of the database.

Figure 8 (a) shows the Sequence Container used for loading the temporary tables needed for the `DimGeography` dimension. Figure 8 (b) shows the structure of the temporary tables. The first task in the sequence executes an SQL script that creates the temporary tables. Figure 8 (c) shows the data flow used for loading the `TempCities` table from the text file `cities.txt`. A data conversion transformation is needed to convert the default types obtained from the text file into the database types.

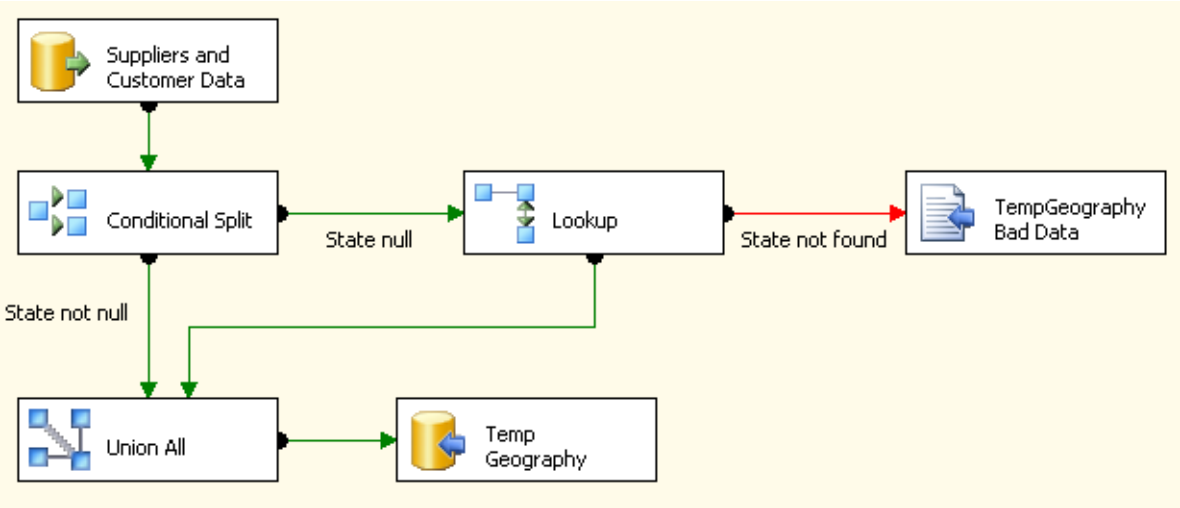


Figure 9: Loading of the `TempGeography` table

Figure 9 shows the data flow for loading of the `TempGeography` table. The source data is obtained from the following SQL query.

```
select distinct City, Region as State, PostalCode, Country
from Suppliers
union
select distinct City, Region as State, PostalCode, Country
from Customers
```

Since some of the rows obtained have a null valued in the `State` attribute a lookup is needed to complete them using the data from the `TempCities` table.

Figure 10 shows the data flow for loading the `TempGeographyFull` table. This data flow completes the the `TempGeography` table with the corresponding `StateKey` and `CountyKey`. For this, four look up tasks are needed as follows.

1. The first look up process records from `TempGeographyFull` where `State` appears in `DimState.StateName`. An example of such a state is `Loire-Atlantique`.
2. The second look up process records from `TempGeographyFull` where `State` appears in `DimState.EnglishStateName`. An example of such a state is `Lower Saxony` whose German name is `Niedersachsen`.
3. The third look up process records from `TempGeographyFull` where `State` and `Country` correspond, respectively, to `StateCode` and `CountryCode` of the lookup table defined by the following query

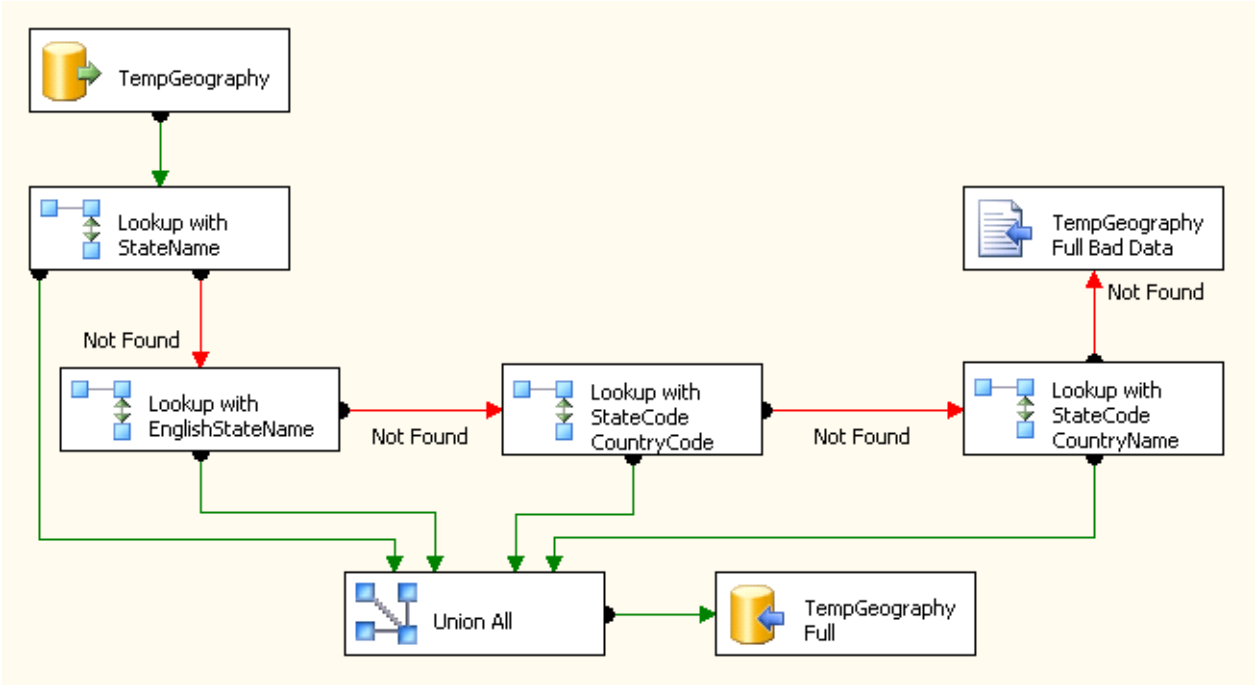


Figure 10: Loading of the `TempGeographyFull` table

```
select S.*, CountryName, CountryCode
from DimState S join DimCountry C
on S.CountryKey = C.CountryKey
```

An example of `State` and `Country` values captured in this look up are `AK` and `USA` that correspond to `Arkansas` and `United States of America`.

4. Finally, the fourth look up process records from `TempGeographyFull` where `State` and `Country` correspond, respectively, to `StateCode` and `CountryName` of the lookup table defined above. An example of `State` and `Country` values captured in this look up are `BC` and `Canada` that correspond to `British Columbia` and `Canada`.

As result, the following two records are rejected

```
Singapore,NULL,0512,Singapore
Cork,Co. Cork,NULL,Ireland
```

that correspond to supplier `'Leka Trading'` and customer `'Hungry Owl All-Night Grocers'`, respectively.

The data flow task that loads the `DimCustomer` dimension is shown in Figure 11. It includes a lookup transformation that starting from the following SQL query

```
select T.City, T.PostalCode, T.Country, GeographyKey
from TempGeographyFull T join DimGeography G
  on T.City=G.City and T.StateKey=G.StateKey
  and T.PostalCode=G.PostalCode
```

transforms the `City`, `PostalCode`, and `Country` of a customer into its corresponding `GeographyKey` of the `DimGeography`. A similar data flow task is used for loading the `DimSupplier` dimension.

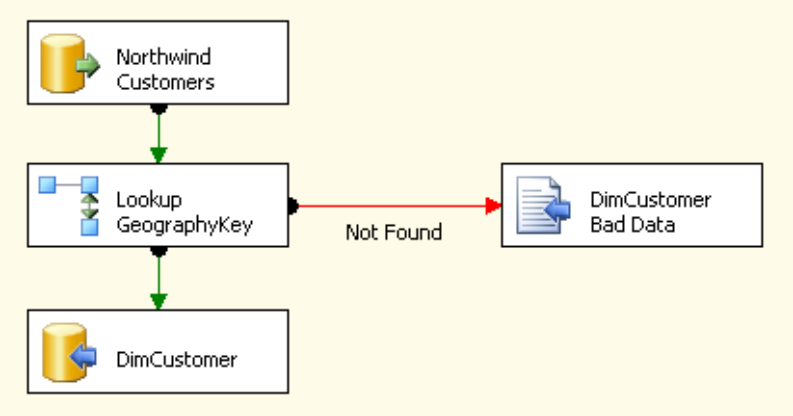


Figure 11: Loading of the `DimCustomer` dimension

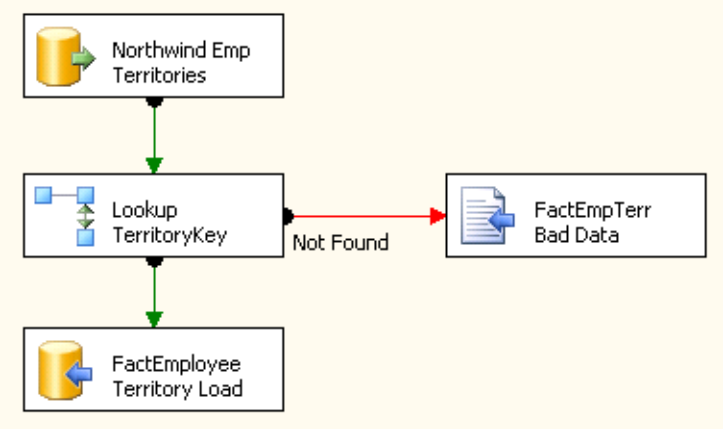


Figure 12: Loading of the `FactEmployeeTerritory` fact table

The data flow task that loads the `FactEmployeeTerritory` fact table is shown in Figure 12. It includes a Lookup transformation that transforms the `TerritoryID` of the operational database into the `TerritoryKey` of the `DimTerritory`.

Finally, the data flow task that loads the `FactSales` fact table is shown in Figure 13. The first OLE DB Source task includes an SQL query that combines data from the operational database and the data warehouse. This SQL query is given next.

```
select
  (select [CustomerKey] from [dbo].[DimCustomer] DC
    where DC.[CustomerAlternateKey] = O.[CustomerID]) as [CustomerKey],
  EmployeeID as EmployeeKey,
  (select [TimeKey] from [dbo].[DimTime] DT
    where DT.[DateAlternateKey] = O.[OrderDate]) as [OrderDateKey],
  (select [TimeKey] from [dbo].[DimTime] DT
    where DT.[DateAlternateKey] = O.[RequiredDate]) as [RequiredDateKey],
  (select [TimeKey] from [dbo].[DimTime] DT
    where DT.[DateAlternateKey] = O.[ShippedDate]) as [ShippedDateKey],
  ShipVia as ShipperKey, P.ProductID as ProductKey,
  (select [SupplierKey] from [dbo].[DimSupplier] DS
```

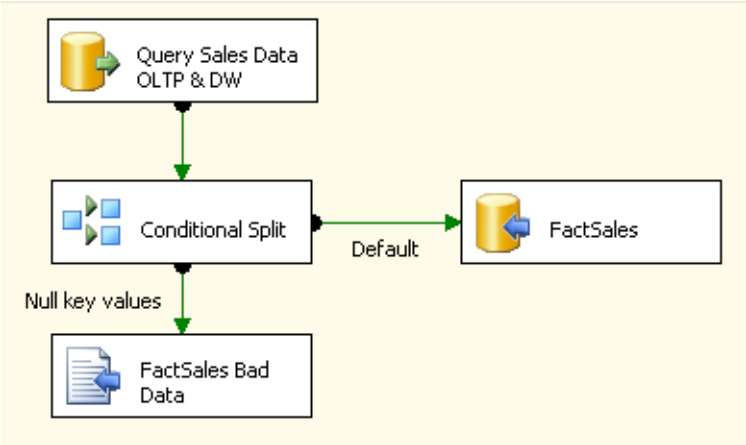


Figure 13: Loading of the `FactSales` fact table

```
    where DS.[SupplierKey] = P.[SupplierID]) as [SupplierKey],
  O.OrderID as OrderNo,
  convert(int, ROW_NUMBER() over
    (partition by D.OrderID order by D.ProductID)) as OrderLineNo,
  D.UnitPrice, Quantity, Discount,
  convert(money, D.UnitPrice * (1-Discount) * Quantity) as SalesAmount,
  convert(money, O.Freight/Count(*) over (partition by D.OrderID)) as Freight
from Northwind.dbo.Orders O, Northwind.dbo.[Order Details] D,
     Northwind.dbo.Products P
where O.OrderID=D.OrderID and D.ProductID=P.ProductID
order by supplierKey
```

A conditional split transformation task is then used to select the records obtained from the previous query that have a null value in the columns `CustomerKey`, `SupplierKey`, or `ShippedDateKey` and store them in a flat file. These are 206 records that correspond to order details from the rejected supplier `'Leka Trading'`, the rejected customer `'Hungry Owl All-Night Grocers'` or that have a null value in `ShippedDate`. The correct records are inserted in the data warehouse.