Data Warehouse Systems: Design and Implementation

Second Edition

Alejandro VAISMAN
Department of Information Engineering
Instituto Tecnológico de Buenos Aires
avaisman@itba.edu.ar

Esteban ZIMÁNYI
Department of Computer & Decision Engineering (CoDE)
Université libre de Bruxelles
esteban.zimanyi@ulb.be

Chapter 9: Extraction, Transformation, and Loading

1. Extraction, Transformation, and Loading
2. Business Process Modeling Notation
3. Conceptual ETL Design using BPMN
4. Conceptual Design of the Northwind ETL
5. Integration Services
6. The Northwind ETL in Integration Services
7. The Northwind ETL SQL

Table of Contents

Extraction, Transformation, and Loading (ETL)

- **Extract** data from internal and external sources, **transform** data, and **load** data into a data warehouse
- No agreed way to specify ETL at a conceptual level
- We study conceptual ETL design
- Conceptual model based on the Business Process Modeling Notation (BPMN)
  - Users already familiar with BPMN do not need to learn another language to design ETL
  - BPMN provides a conceptual and implementation-independent specification of processes
  - Processes expressed in BPMN can be translated into executable specifications (e.g., Microsoft's Integration Services or PL/SQL)

Table of Contents

Business Process Modeling Notation (BPMN)

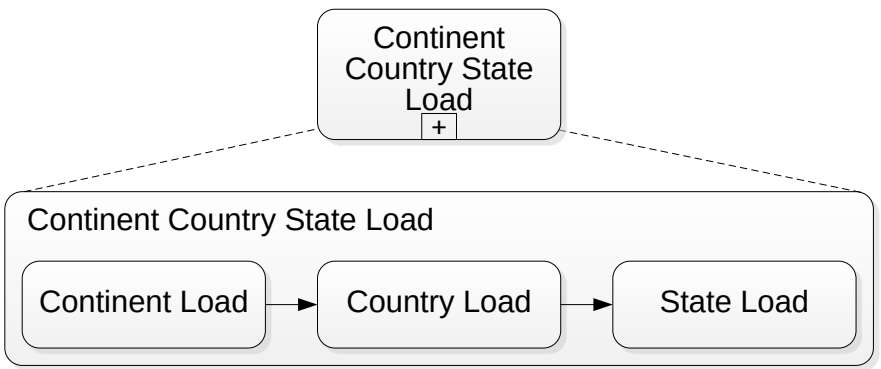
- **Business process**: A collection of related activities or tasks whose goal is to produce a specific service or product
- **Business process modeling**: Activity of representing the business processes of an organization, so that the current processes may be analyzed and improved
- Many techniques to model business processes have been proposed
- No formal semantics for these techniques
- Formal techniques (e.g., Petri Nets): Well-defined semantics but hard to understand by business users
- A standardization process resulted in the **Business Process Modeling Notation** (**BPMN**) released by the Object Management Group (OMG). Current version is BPMN 2.0
- BPMN: Graphical notation for defining, understanding, and communicating the business procedures of an organization in a standard manner
- Four basic categories of elements: **flow objects**, **connecting objects**, **swimlanes**, and **artifacts**

Flow Objects: Activities

- An **activity** is a work performed during a process
  - Can be atomic or nonatomic
  - Can be a task or a subprocess
- **Subprocess**: An encapsulated process whose details we want to hide

Activity

Collapsed and expanded subprocess

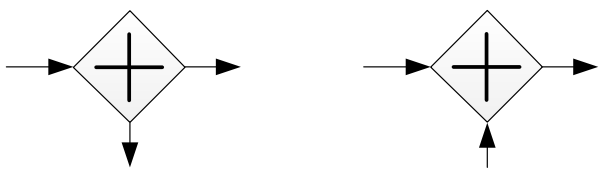


Flow Objects: Gateways

- Control the activity sequence in a process, based on conditions
- Represent only logic, not activities
- **Exclusive gateways** model OR-split decisions
- **Inclusive gateways** select or merge one or more flows
- **Parallel gateways** allow the synchronization between outgoing and incoming flows
  - **Splitting** parallel gateway: Analogous to an AND-split
  - **Merging** parallel gateway: Synchronizes the flow and merges all the incoming flows into a single outgoing one
- **Complex gateways** can represent complex conditions

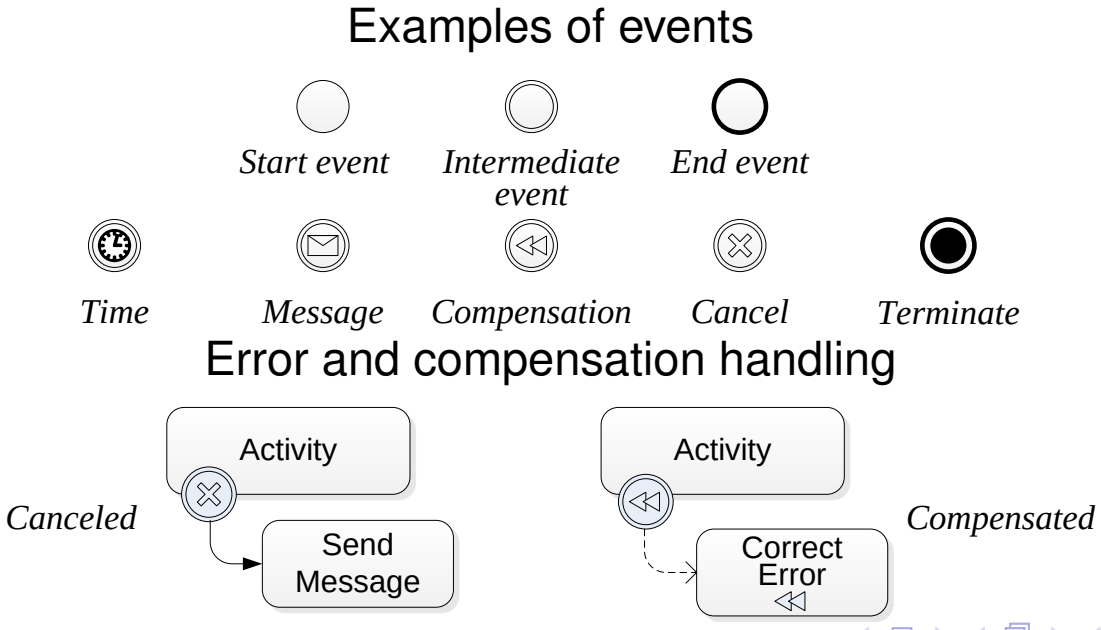
Different types of gateways

Splitting and merging gateways



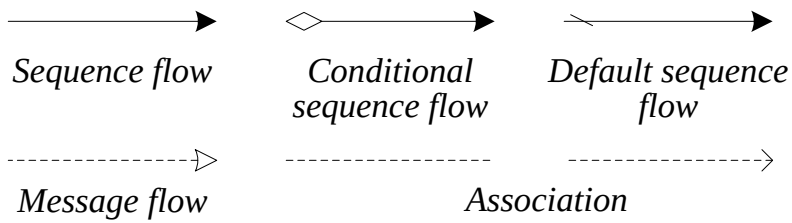
Flow Objects: Events

- Represent something that happens that affects the workflow activities
- **Start** and **end** events indicate the beginning and ending of a process
- **Time** event represents situations when a task must wait some time before continuing
- **Message** event represents communication
- **Compensation** event represents error detection and recovery by launching compensation activities
- **Cancel** event listens to the process errors and notifies them by an explicit or implicit action
- **Terminate** event, when reached, the entire process is stopped, including all parallel processes



Connecting Objects

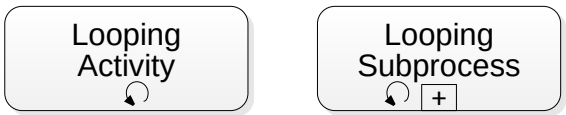
- Represent how objects are connected
- **Sequence flow**: A sequencing constraint between flow objects
  - If two activities are linked by a sequence flow, the target one starts when the source one has finished
  - If multiple sequence flows outgo from a flow object, all of them will be activated after its execution
- **Conditional sequence flow**: Adds a condition to the sequence flow
- A sequence flow may be set as the **default flow** in case of many outgoing flows: if no other condition is true in a gateway, the default flow is followed
- A **message flow** represents the **only way** of sending and receiving of messages between pools
- An **association** relates artifacts (e.g., annotations) to flow objects



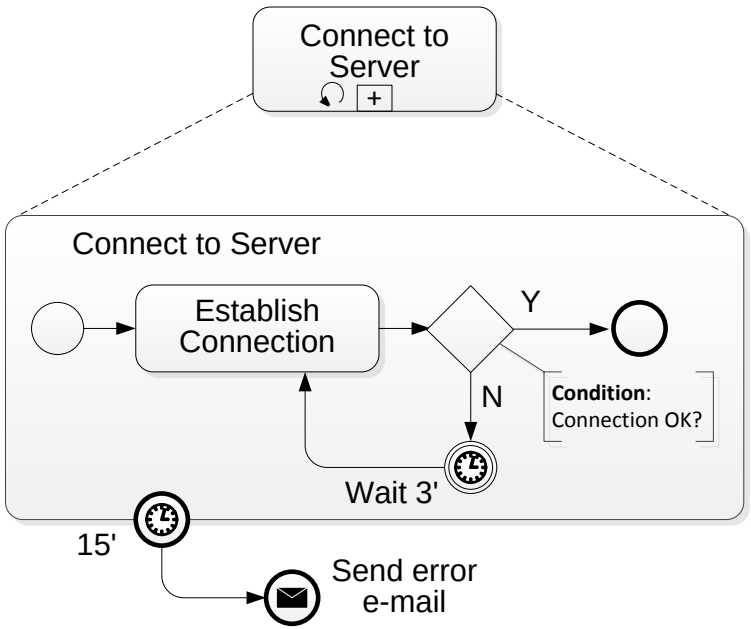
Loops and Subprocesses

- **Loop**: Execution control representing repeated execution of a process
- Conditions checked before or after activity, loop ended if its condition evaluates to false
- Example: ETL process representing the connection to a server task
- At a high abstraction level, the **subprocess activity** hides the details
- Expansion shows details: server waits 3 minutes (**time** event). If connection not established, request launched again. If no connection after 15 minutes, task stopped, and error email sent (**message** event).

Loops

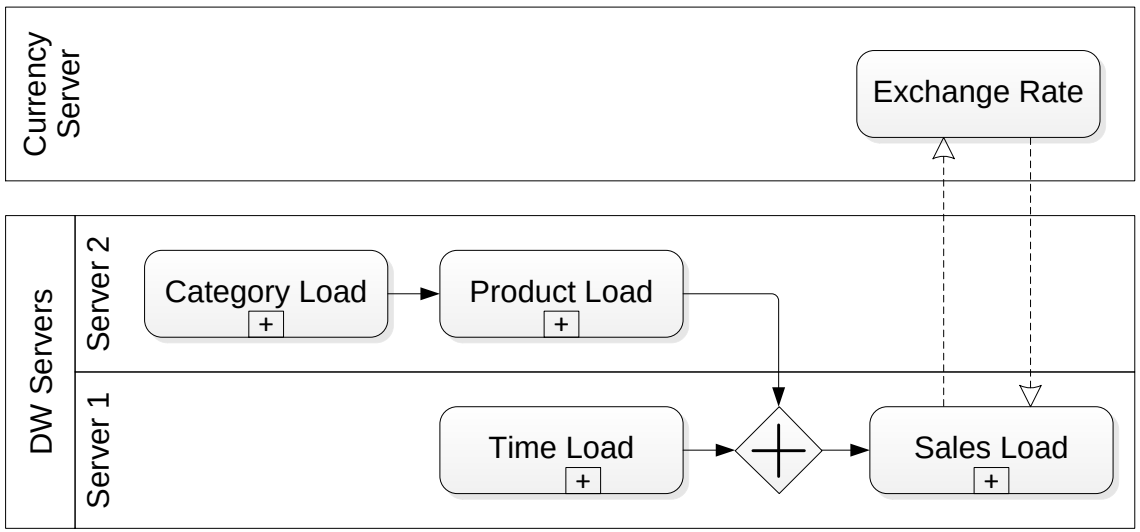


Subprocesses



Swimlanes

- A structuring object that comprises **pools** and **lanes**
- Both allow the definition of process boundaries
- Only messages allowed between two pools, not sequence flows
- A workflow must be contained in only one pool
- One pool may be subdivided into many lanes, which represent roles or services



Artifacts

- Allow to visually represent objects outside the actual process
- Can represent data or notes that describe the process, or they can be used to organize tasks or processes
- Can be data objects, groups, and annotations
- A **data object** represents either data that are input to a process, data resulting from a process, data that needs to be collected, or data that needs to be stored.
- A **group** organizes tasks or processes that have some kind of significance in the overall model
- **Annotations** are used to express semantics about the flow objects, e.g., to indicate the attributes involved in a lookup task, or a gateway condition

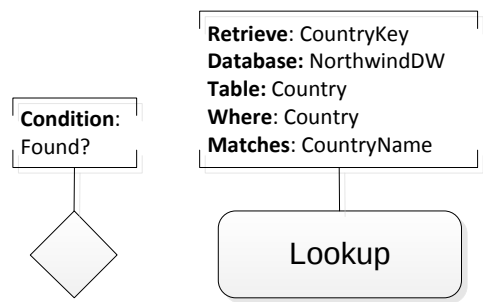


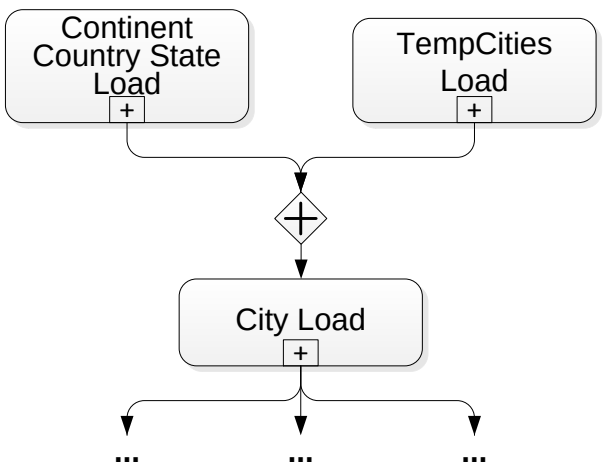
Table of Contents

Conceptual ETL Design using BPMN

- Basic assumption for using BPMN as conceptual model: ETL process is a type of business process
- There is no standard model for defining ETL processes
- Each tool provides its own model, too detailed to be conceptual
- Using BPMN constructs we define the most common ETL tasks and define a BPMN notation for ETL
- ETL process: A combination of **control** and **data processes**
  - Control processes manage the coarse-grained groups of tasks
  - Data processes detail how input data are transformed and output data are produced
- Two kinds of tasks in ETL conceptual modeling
  - **Control tasks** highlight the control procedures provided by BPMN. Represent a **workflow** (arrows represent the precedence between activities)
  - **Data tasks** refer to the tasks that directly manipulate data during an ETL process. Represent a **data flow** (arrows represent data 'flowing' along them)

Control Tasks

- Represent the workflow sequence or **orchestration** of the ETL process independently of the data flow
- Control tasks are represented by means of BPMN constructs
- For example, gateways are used to control the sequence of activities in an ETL process
- The most used types of gateways in an ETL context are exclusive and parallel

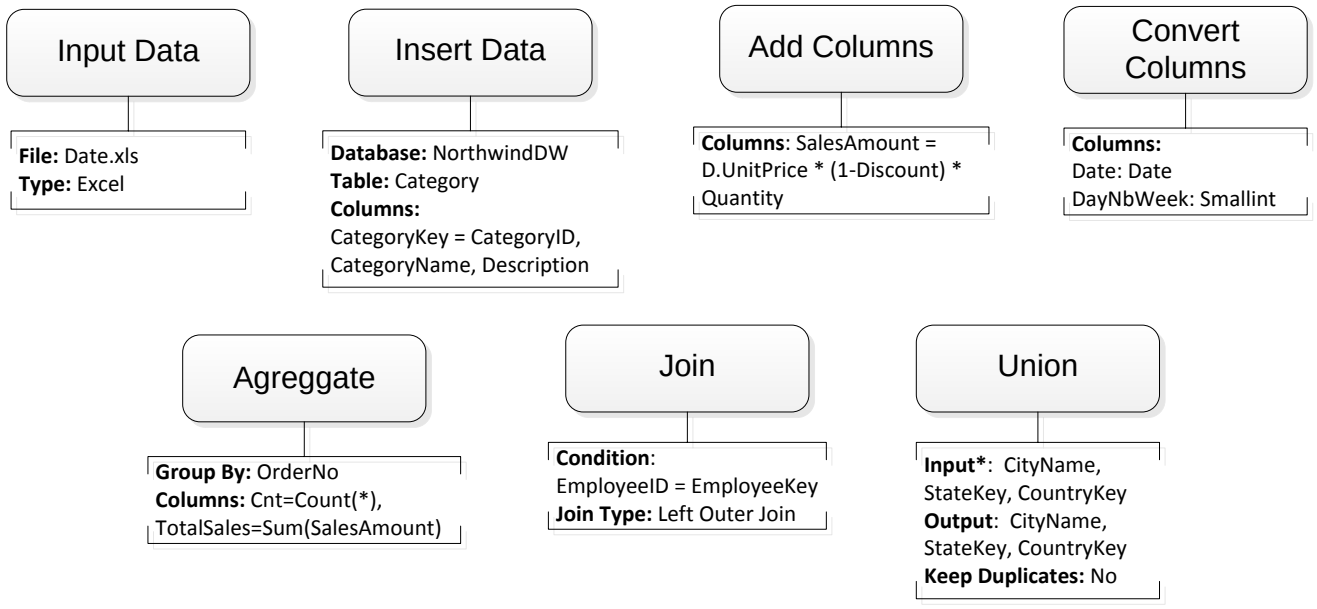


Data Tasks

- Show how data are manipulated **within** an activity
- At lower abstraction level than control tasks
- Represent activities typically carried out to manipulate data: input and output data, data conversion and transformation (for instance, change the data type of an attribute, add a column, remove duplicates, and so on)
- We denote these tasks **unary data tasks** since they receive one input flow
- ***n*-ary data tasks** receive as input more than one flow (e.g., this is the case of union, join, difference,...)
- **Row operations** are transformations applied to the source or target data on a row-by-row basis, e.g., updating the value of a column

Data Tasks (cont.)

- **Rowset operations** deal with a set of rows, e.g., aggregation



- **Lookup data tasks** check if some value is present in a file. Immediately followed by an exclusive gateway with a branching condition.
- Shorthand notation for the lookup task

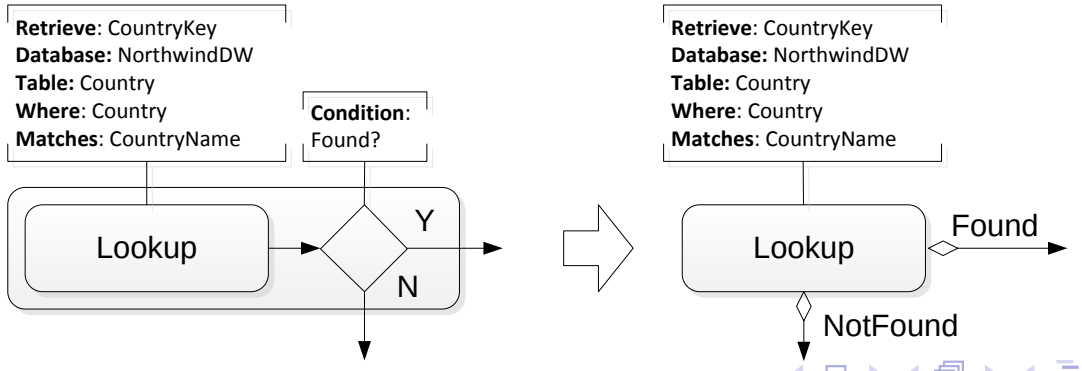
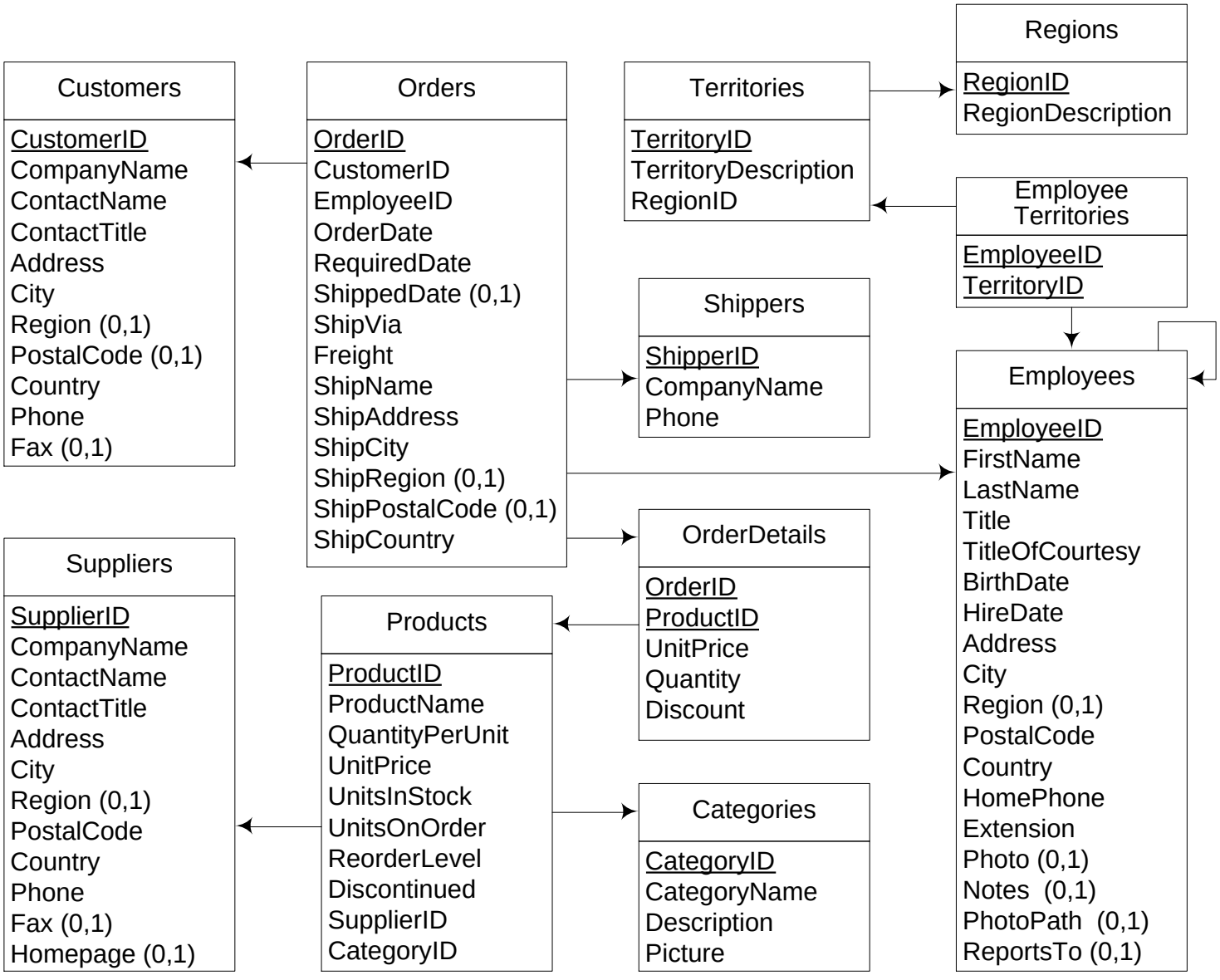
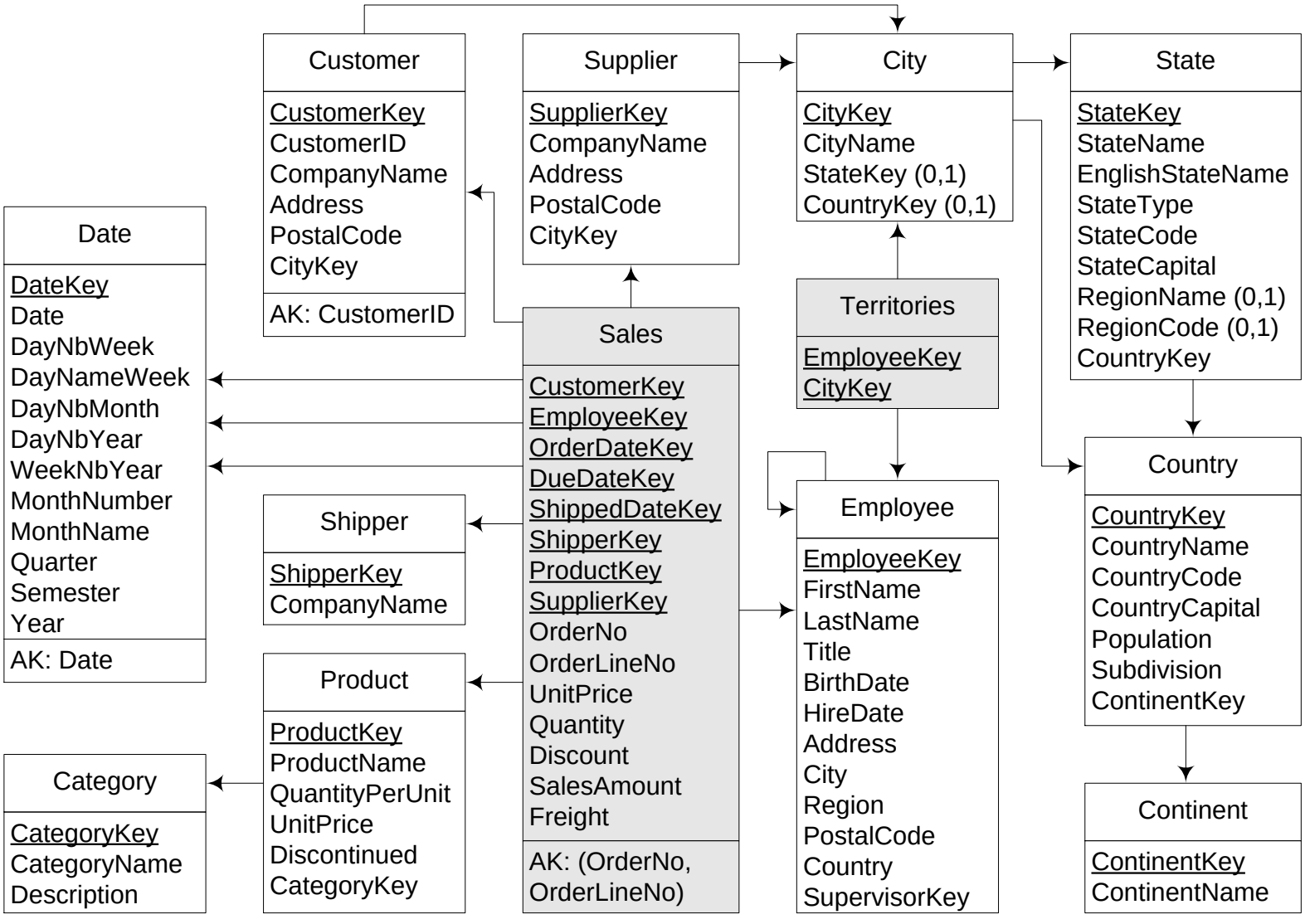


Table of Contents

Schema of the Northwind Operational Database



Schema of the Northwind Data Warehouse



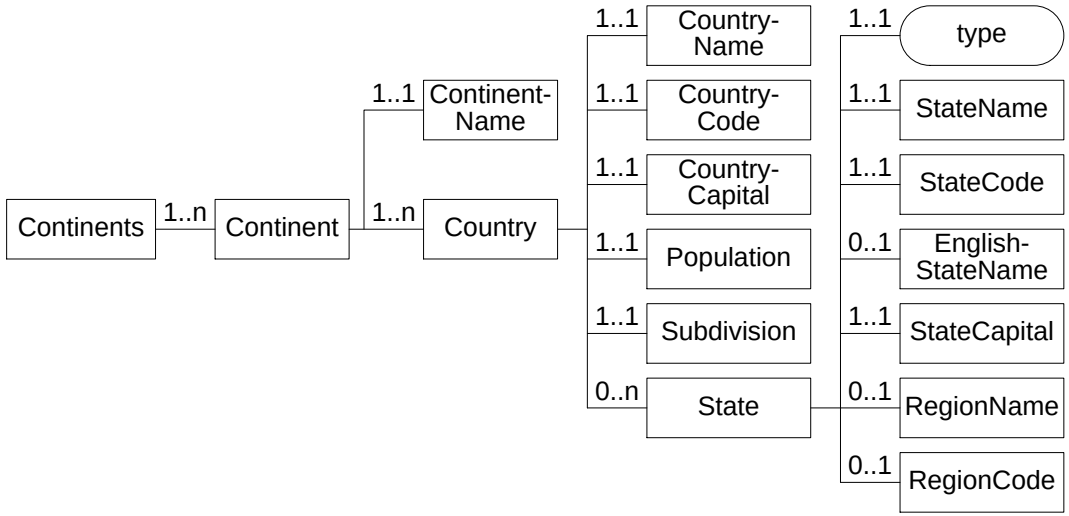
Conceptual Design of the Northwind ETL: Data Sources

- File Date.xls contains data for loading the Date dimension, spanning the dates in table Orders of the operational database
- Dimensions Customer and Supplier share the geographic hierarchy starting at the City level
- Data for the hierarchy State → Country → Continent loaded from Territories.xml
- Start of the file Territories.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Continents>
    <Continent>
        <ContinentName>Europe</ContinentName>
        <Country>
            <CountryName>Austria</CountryName>
            <CountryCode>AT</CountryCode>
            <CountryCapital>Vienna</CountryCapital>
            <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>
            ...
```

Conceptual Design of the Northwind ETL: Data Sources

- XML Schema of Territories.xml



- File called Cities.txt identifies to which state or province a city belongs
- Contains three fields separated by tabs and begins as shown below
- For cities located in countries that do not have states (e.g., Singapore), second field is set to null
- The file is also used to identify to which state corresponds the city in the attribute TerritoryDescription of table Territories

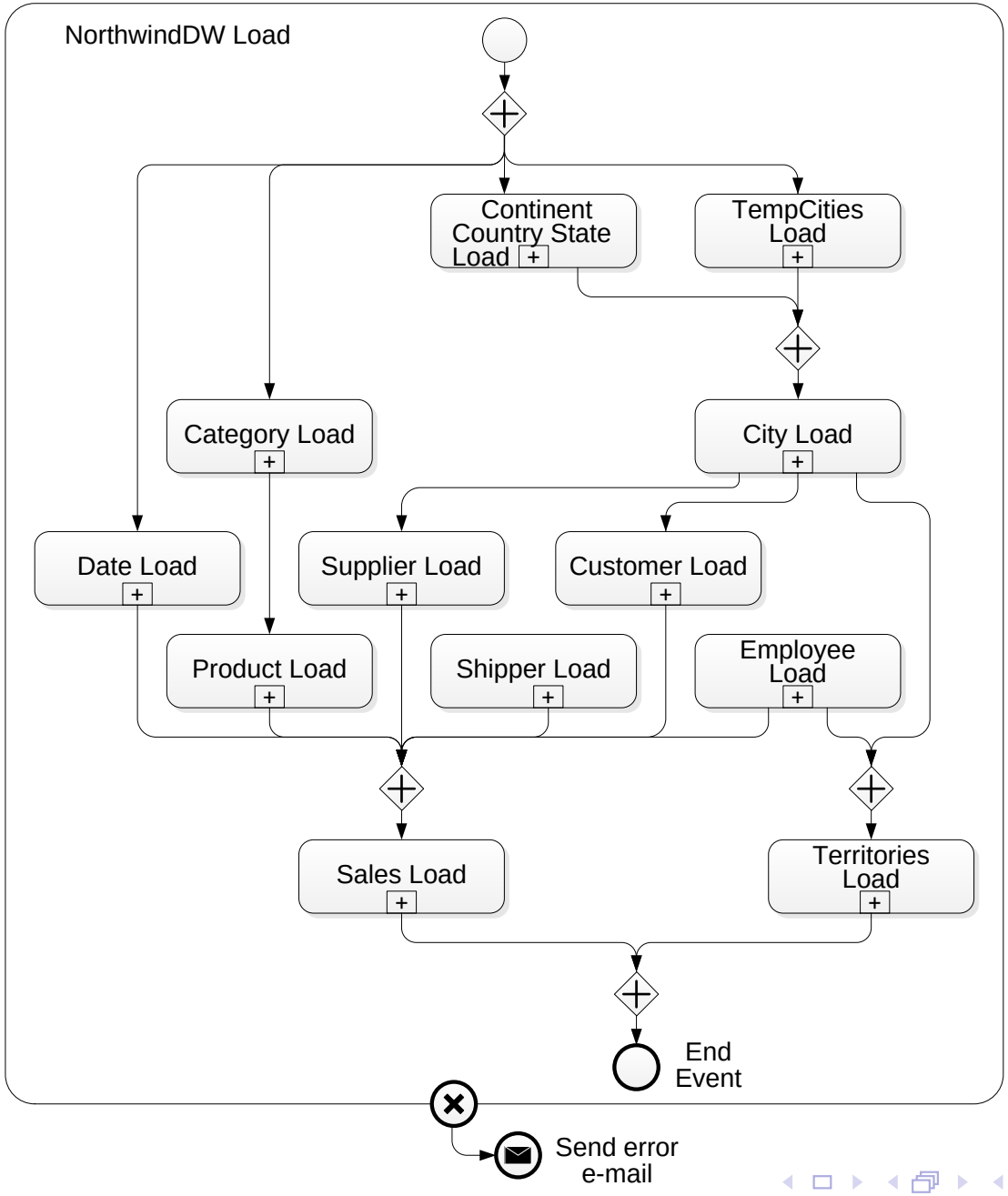
City → State → Country
Aachen → North Rhine-Westphalia → Germany
Albuquerque → New Mexico → USA
Anchorage → Alaska → USA
Ann Arbor → Michigan → USA
Annecy → Haute-Savoie → France
...

Begining of the file Cities.txt

TempCities
City
State
Country

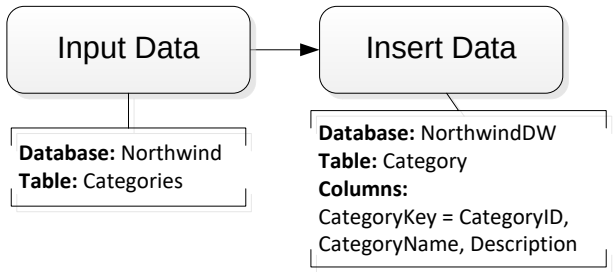
Associated table TempCities

Conceptual Design of the Northwind ETL: Overall View

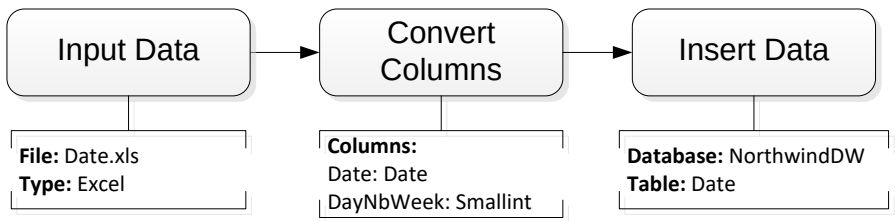


Conceptual Design of the Northwind ETL

- Load of the Category dimension table

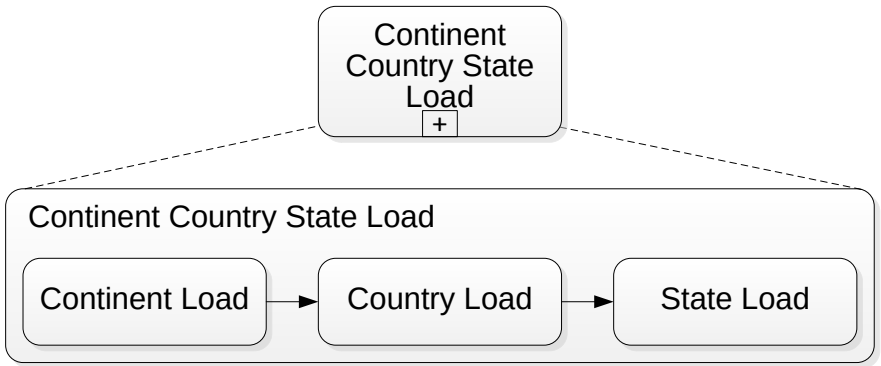


  - Input task loads table Categories from the operational database
  - Insert task loads the table Category in the data warehouse, mapping CategoryID to CategoryKey attribute in the Category table
- Loading the Date dimension table from an Excel file is similar, but includes a data type conversion, and an and an addition of the column DateKey

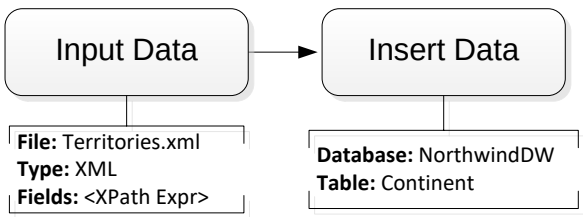


Conceptual Design of the Northwind ETL

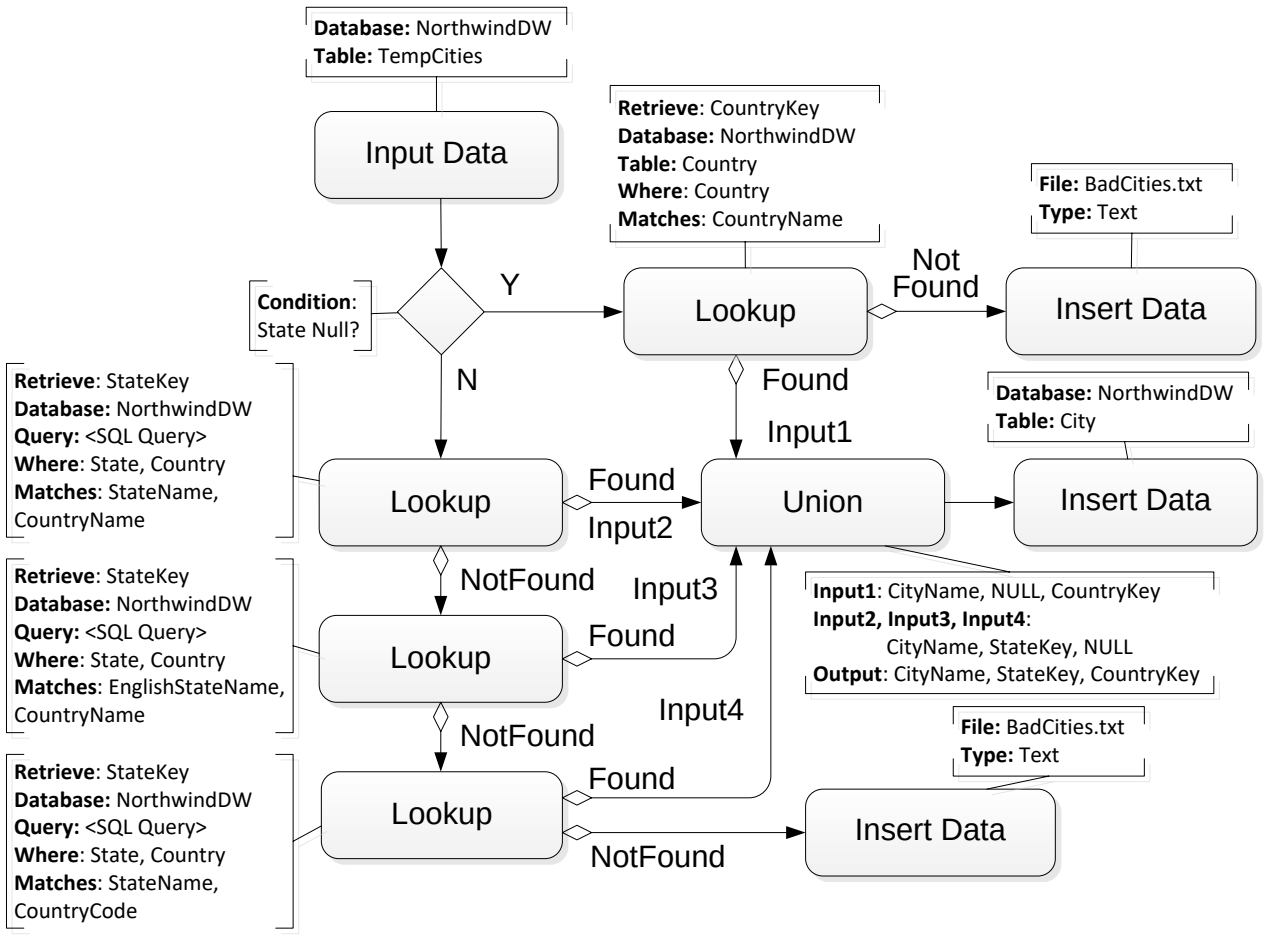
- Loading the City level first requires loading the Geography hierarchy State → Country → Continent
- Associated control task



- Load of the Continent table



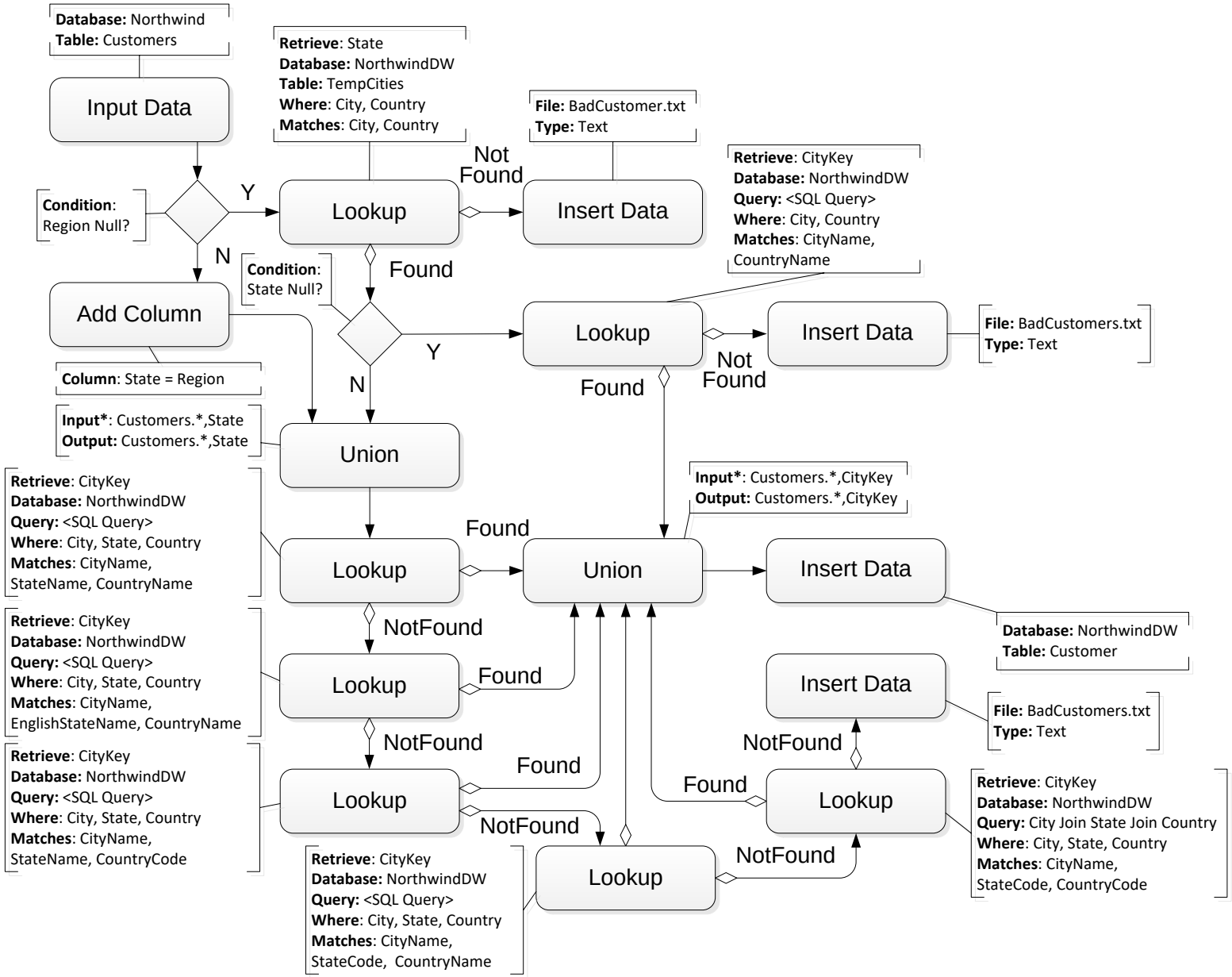
Load of the City Level



Load of the City Level

- Assume that a table TempCities(City,State,Country) has been created and populated from Cities.txt
- First task is an input data over TempCities
- An exclusive gateway tests whether State is null or not
  - If so, lookup obtains the CountryKey
  - If not, we match (State, Country) pairs in TempCities to values in the State and Country tables
- Finally, union performed with the results of the four flows, and table is loaded with an insert data task
- Records for which the state and/or country are not found are stored into a BadCities.txt file.

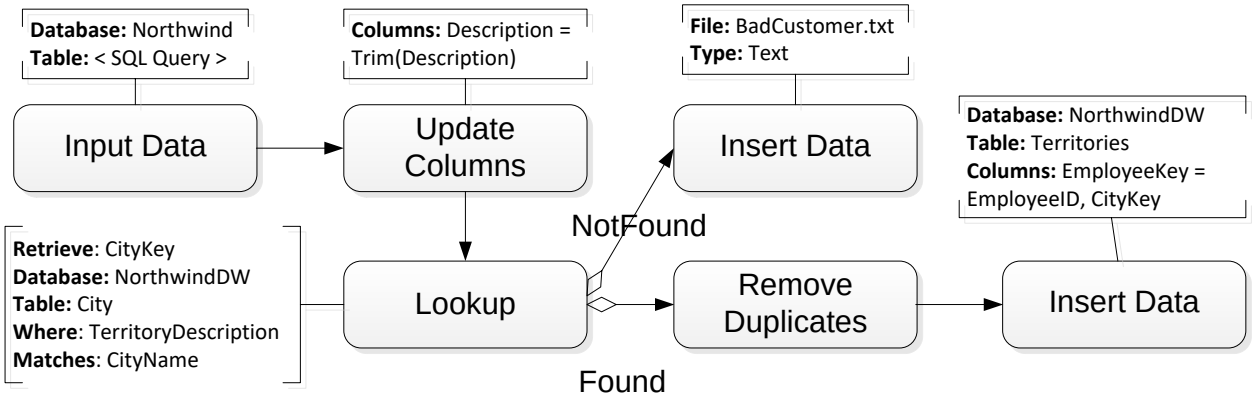
Load of the Customer Level



Load of the Customer Level

- Input table Customers read from the operational database using an input data task
- Region (optional) in Customers is actually a state name or a state code → first exclusive gateway checks whether this attribute is null or not
  - If not null, add new column State initialized with the values of Region
  - Otherwise, check if the (City, Country) pair matches a pair in TempCities, and retrieve the State attribute, creating a new column
- A second exclusive gateway over the new State column accounts for countries without states
- Then perform a union over the two flows
- Finally, perform the union of all flows, and add the column CustomerKey for the surrogate key initialized to null

Load of the Territories Bridge Table

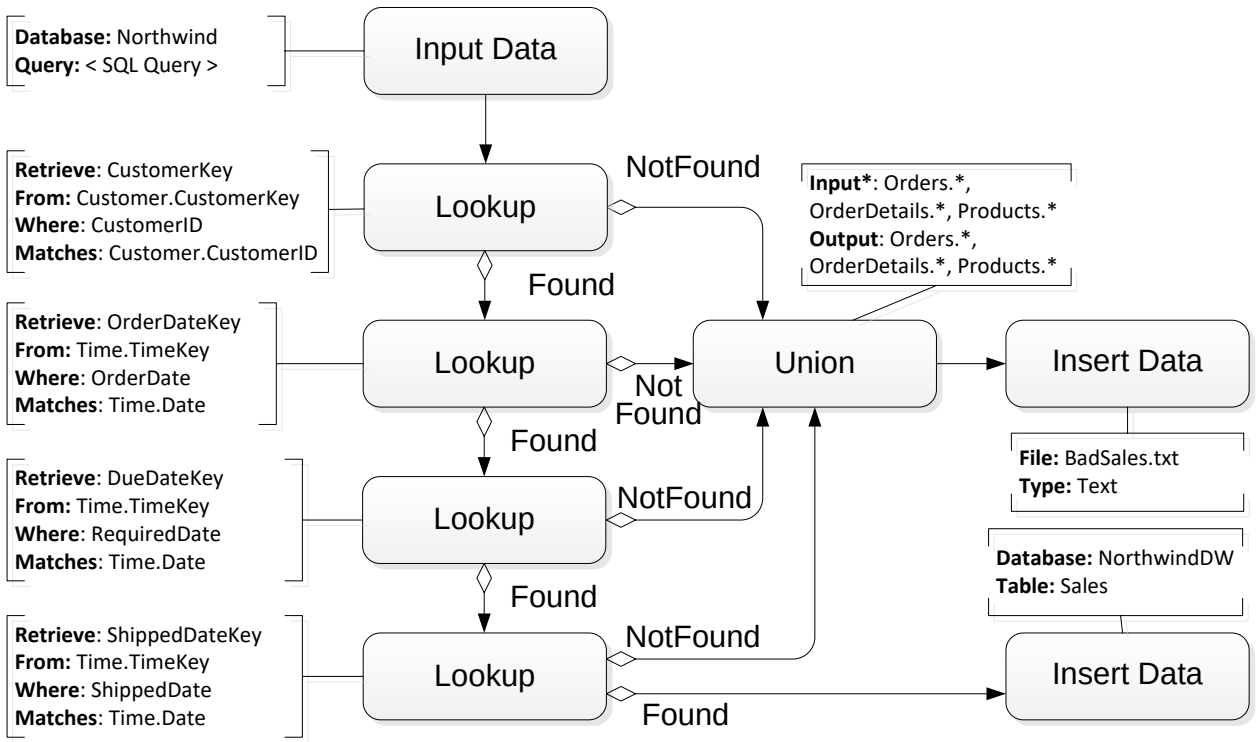


- Input: SQL query joining EmployeeTerritories and Territories

```
SELECT E.*, TerritoryDescription
FROM EmployeeTerritories E JOIN Territories T
        ON E.TerritoryID = T.TerritoryID
```

- An update column task removes (‘trims’) leading spaces from TerritoryDescription
- The city key is obtained with a lookup over City in the DW
- Duplicates need to be removed: employees may be assigned more than once to the same territory in the operational DB
- Finally, Territories is populated with an insert data task

Load of the Sales Fact Table



Load of the Sales Fact Table

- Task performed once all the other ones done
- Columns for order line number, sales amount, and freight must be created (Add Columns data tasks)
- The process starts with an input data task that obtains data from the operational database via the query:

```
SELECT O.CustomerID, EmployeeID AS EmployeeKey, O.OrderDate,
       O.RequiredDate AS DueDate, O.ShippedDate,
       ShipVia AS ShipperKey, P.ProductID AS ProductKey,
       P.SupplierID AS SupplierKey, O.OrderID AS OrderNo,
       ROW_NUMBER() OVER (PARTITION BY D.OrderID
       ORDER BY D.ProductID) AS OrderLineNo, D.UnitPrice, Quantity,
       Discount, D.UnitPrice * (1-Discount) * Quantity AS SalesAmount,
       O.Freight/COUNT(*) OVER (PARTITION BY D.OrderID) AS Freight
FROM   Orders O, OrderDetails D, Products P
WHERE  O.OrderID = D.OrderID AND D.ProductID = P.ProductID
```

- A sequence of lookups follows, which obtains the missing foreign keys for the dimension tables
- Finally, the fact table is loaded with the data retrieved

Table of Contents

Integration Services

- SQL Server component to perform data migration tasks, and implement and execute ETL processes
- **Components of Integration Services**
  - **Package**: A workflow containing a collection of tasks executed in an orderly fashion
  - A package consists of a control flow and, optionally, one or more data flows
  - **Control flow**: three kinds of elements
  - **Tasks**: Individual units of work that provide functionality to a package
  - Tasks: data flow tasks, data preparation tasks, Analysis Services tasks, workflow tasks
  - **Containers**: Group tasks logically into units of work, and are used to define variables and events
  - Ex: Sequence Container and For Loop Container
  - **Precedence constraints**: Connect tasks, containers, and executables defining execution order
- Creating a control flow in Integration Services requires:
  - Adding containers
  - Adding tasks
  - Connecting containers and tasks, using precedence constraints
  - Adding connection managers, when a task connects to a data source

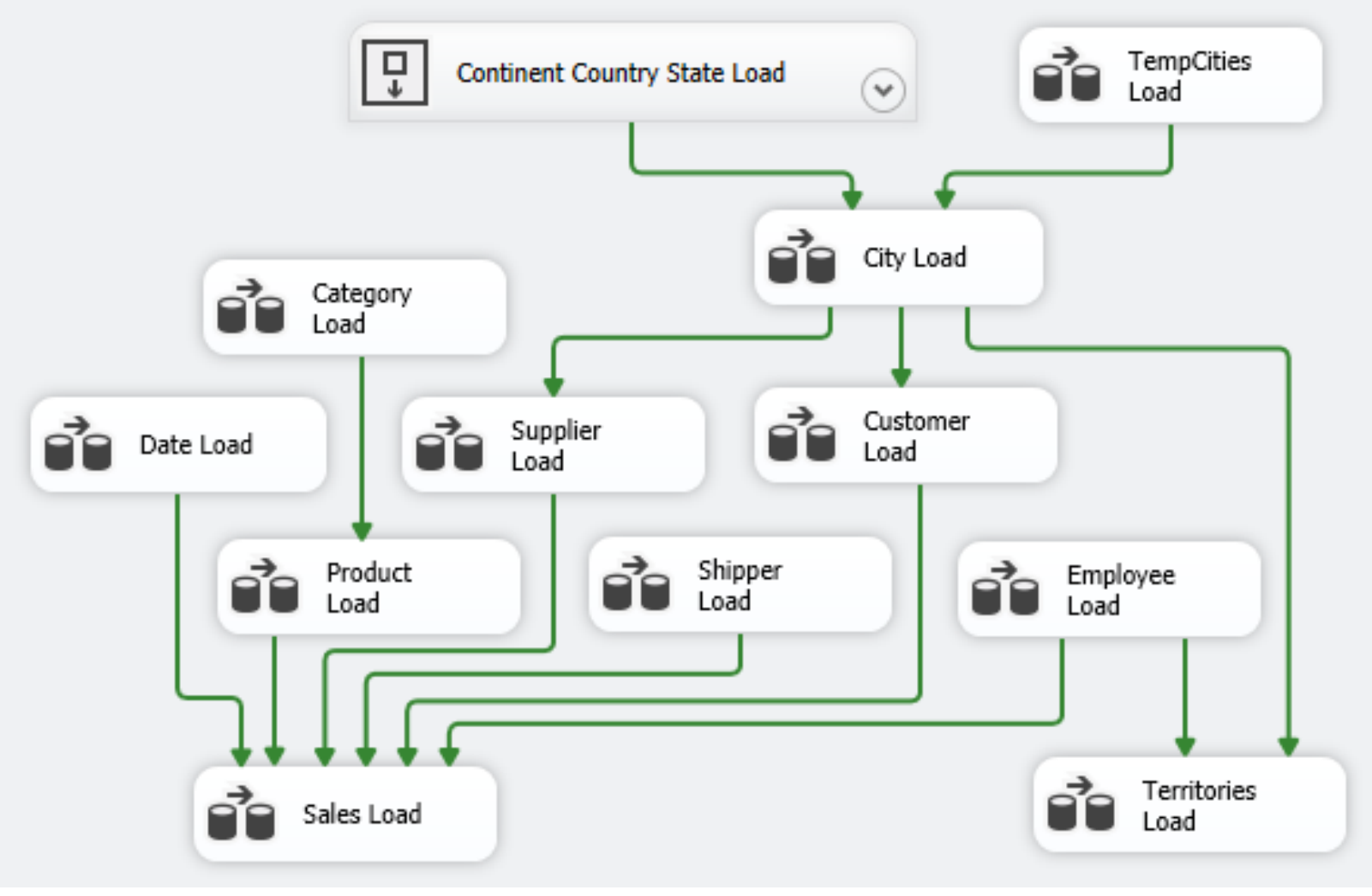
Integration Services: Data Flows

- Extract data into memory, transform them, and write them to a destination
- Three kinds of components:
  - **Sources**: Extract data from data stores (OLE DB data sources, Excel files, flat files, and XML files, among other)
  - **Transformations**: Modify, summarize, and clean data (split, divert, or merge the flow)
    - Example: Conditional Split, Copy Column, and Aggregate.
  - **Destinations**: Load data into data stores or create in-memory datasets
- Creating a data flow includes the following steps
  - Adding one or more sources
  - Adding the transformations to satisfy the package requirements
  - Connecting data flow components
  - Adding one or more destinations to load data into data stores
  - Configuring error outputs
  - Including annotations to document the data flow

Table of Contents

The Northwind ETL in Integration Services

- We just need to translate the conceptual constructs to the equivalent Integration Services ones
- Overall view of the ETL process



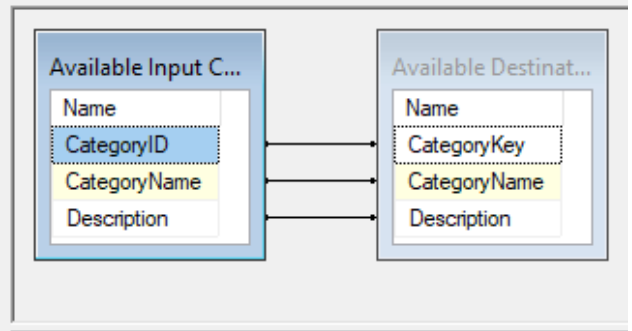
Data Flow Tasks

- Many data flow tasks are simple
- These data flow tasks are composed of an OLE DB Source task that reads the table from the operational database and an OLE DB Destination task that receives the output and stores it in the DW
- Loading the Category dimension table

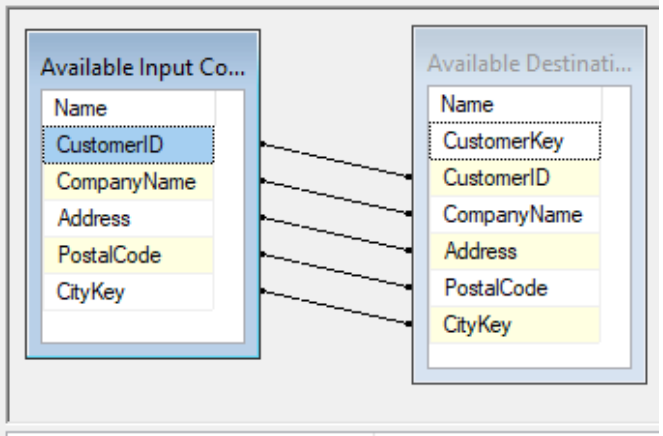
- Similar data flows are used for loading the Product, Shipper, and Employee tables
- Also straightforward is the data flow that loads the Date dimension from the source Excel file after a data conversion

Keys in the Data Warehouse

- Keys of the operational database are reused in the DW where dimensions do not have an alternate key
  - For table Category we reuse CategoryID as CategoryKey key in the DW
  - For table Customer the CustomerID key becomes CustomerAltKey in the DW
  - A new value for CustomerKey is generated during the insert in the DW
- Mappings of source and destination columns depend on the reuse of the key



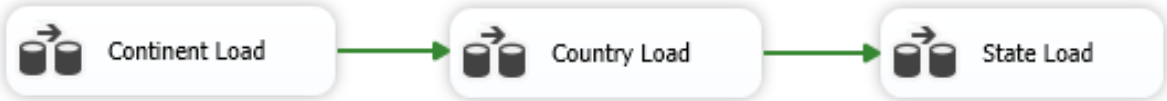
Input Column	Destination Column
CategoryID	CategoryKey
CategoryName	CategoryName
Description	Description



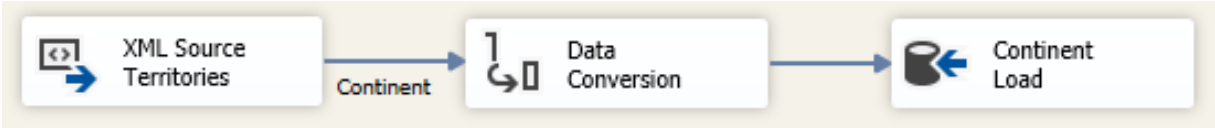
Input Column	Destination Column
<ignore>	CustomerKey
CustomerID	CustomerID
CompanyName	CompanyName
Address	Address
PostalCode	PostalCode
CityKey	CityKey

Load of the Continent → Country → State Hierarchy

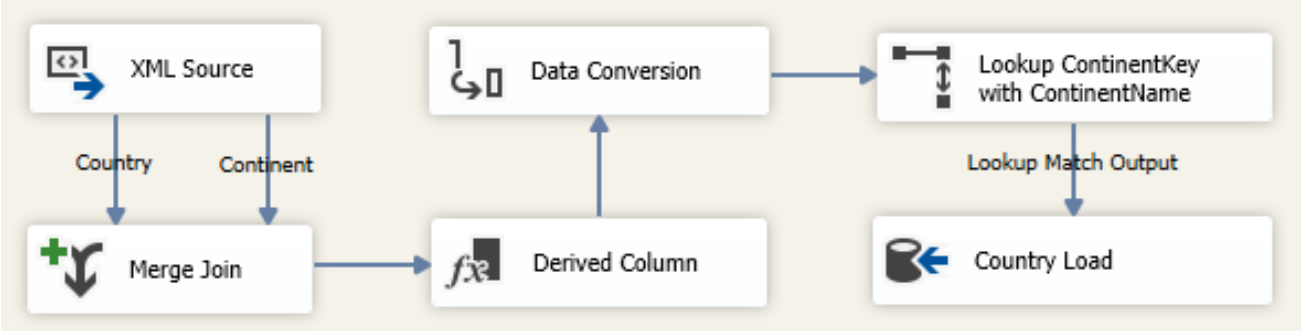
- Sequence container groups the three data flows that load hierarchy tables



- Load of the Continent level



- Load of the Country level

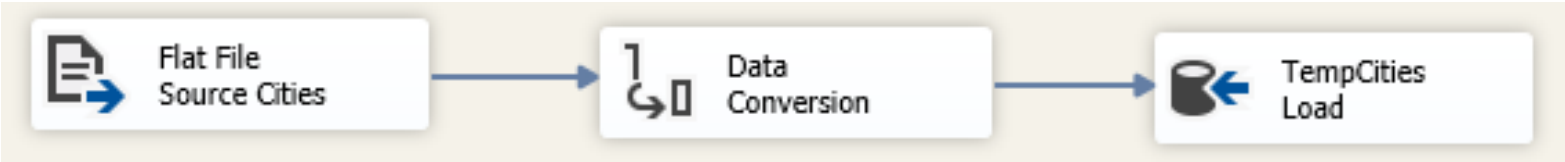


- First produce a key to reference Continent from Country
- Data conversion tasks detailed in the next slide
- In the data flow that loads Country a merge join obtains the ContinentName for a given Country

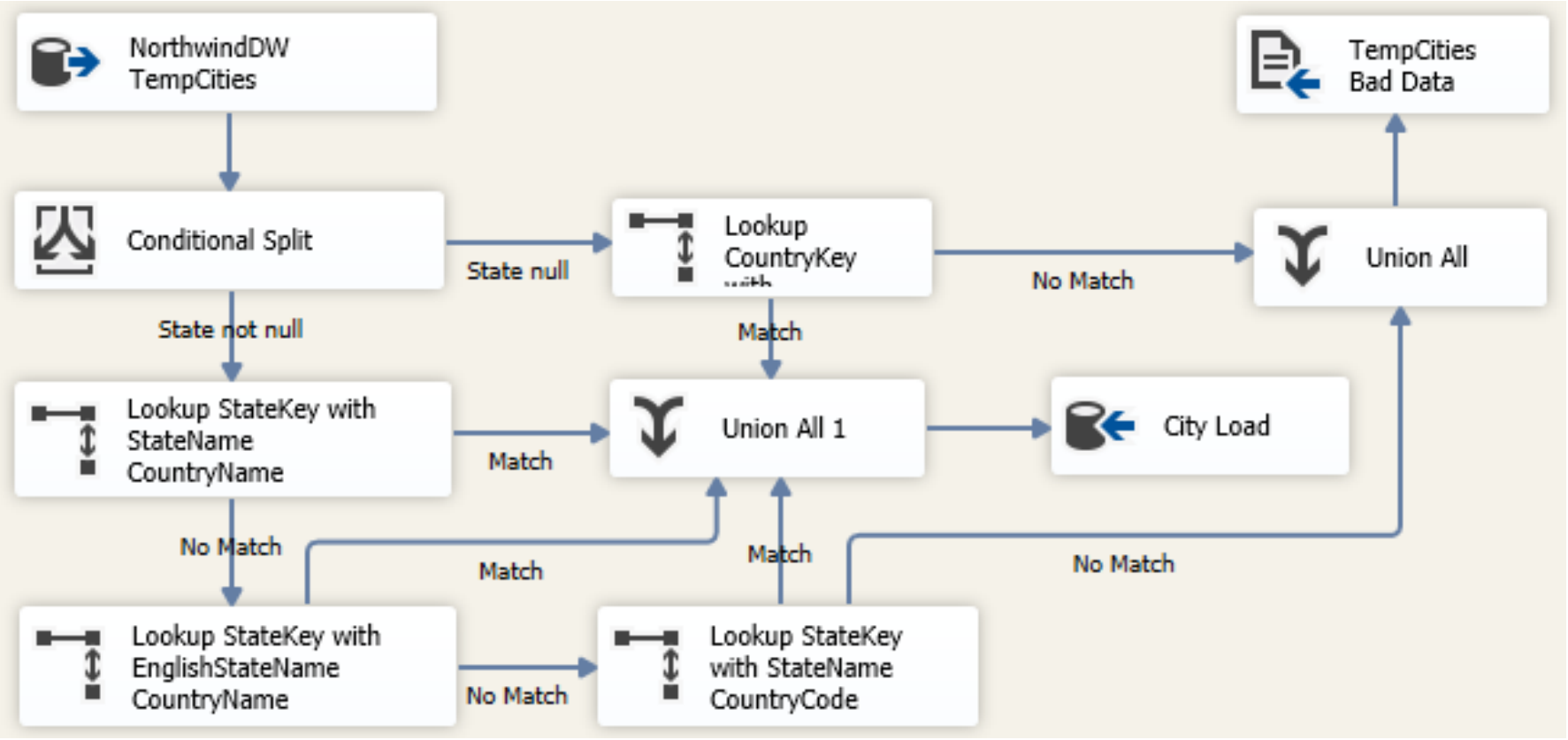
Load of the TempCities Table

TempCities
City State Country

- TempCities: Temporary table needed to load the geographic hierarchy associated to dimensions Customer and Supplier
- Its data is obtained from the text file Cities.txt
- We assume that this table already exists in the database
- A data conversion transformation is needed to transform the default types obtained from the text file into the database types

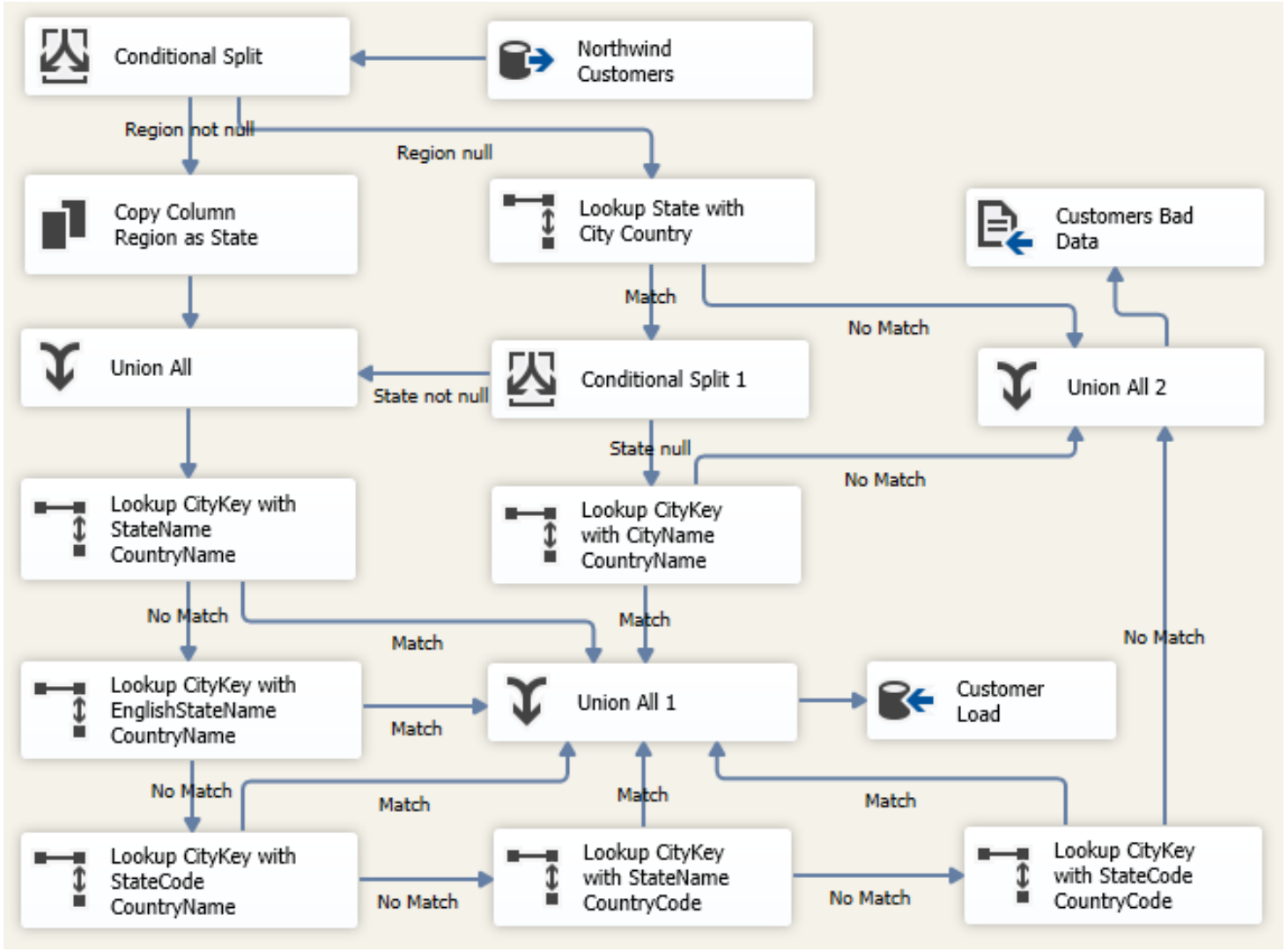


Load of the City Level



- Associate to each city in TempCities a CountryKey or StateKey depending on whether or not the corresponding country is divided in states
- Conditional split tests if the State is null or not
- Different paths in the data flow for obtaining the CountryKey or the StateKey

Load of the Customer Level



Load of the Customer Level

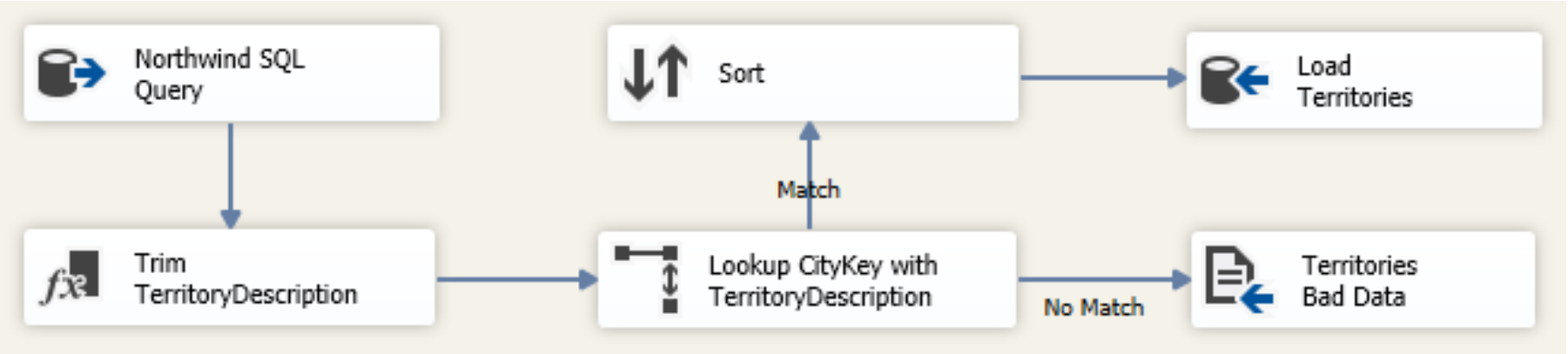
- Starts with a conditional split: if a customer has a null value in Region, a lookup adds a column State by matching City and Country from Customers with City and Country from TempCities
- The value State obtained may be null for countries without states ⇒ a conditional split is needed
- If state is null, then a lookup tries to find a CityKey matching values of City and Country in a lookup table built as a join between City and Country

```
SELECT CityKey, CityName, CountryName
FROM   City C JOIN Country T ON
       C.CountryKey = T.CountryKey
```

- For customers with nonnull Region, the values of this column are copied into a new column State
- Then, 5 lookup tasks are needed, where each one tries to match a couple of values of State and Country to values in the lookup table built as a join between the City, State, and Country tables:

```
SELECT C.CityKey, C.CityName, S.StateName, S.EnglishStateName,
       S.StateCode, T.CountryName, T.CountryCode
FROM   City C JOIN State S ON C.StateKey = S.StateKey
       JOIN Country T ON S.CountryKey = T.CountryKey
```

Load of the Territories Fact Table

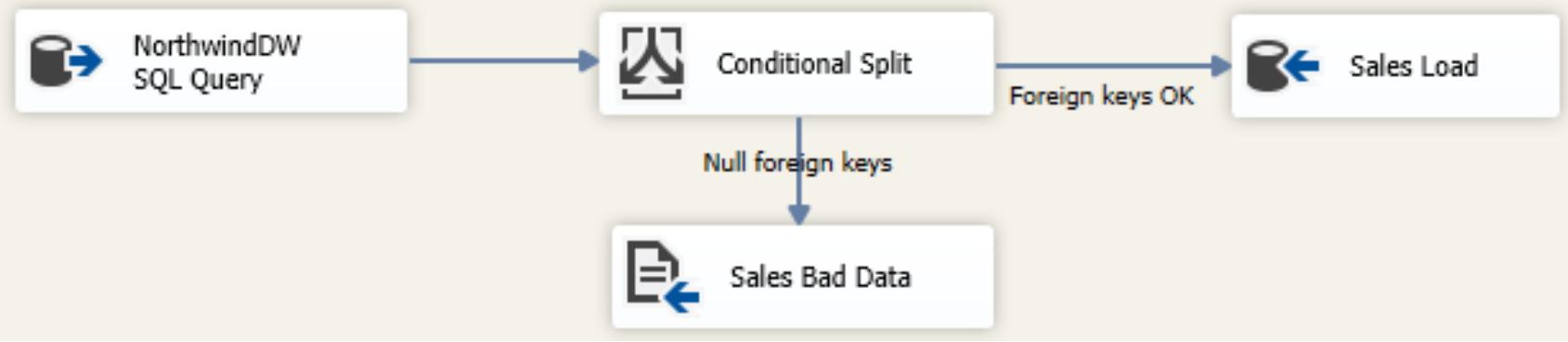


- Data flow task starts with a SQL query in the OLE DB Source task

```
SELECT E.*, TerritoryDescription
FROM    EmployeeTerritories E JOIN Territories T
            ON E.TerritoryID = T.TerritoryID
```

- A derived column transformation removes trailing spaces in the values of TerritoryDescription
- A lookup transformation searches corresponding values of CityKey in City
- Then a sort transformation removes duplicates

Load of the Sales Fact Table



- The first OLE DB Source task is a query that combines data from the operational DB and the DW (see the query in next slide)
- A conditional split transformation task selects the erroneous records from the query containing a null value in CustomerKey or ShippedDateKey and stores them in a flat file.
- The correct records are inserted in the DW

Load of the Sales Fact Table

- Query of the OLE DB source task

```
SELECT
    ( SELECT CustomerKey FROM dbo.Customer C
      WHERE C.CustomerID = O.CustomerID) AS CustomerKey,
    EmployeeID AS EmployeeKey,
    ( SELECT DateKey FROM dbo.Date T
      WHERE T.Date = O.OrderDate) AS OrderDateKey,
    ( SELECT DateKey FROM dbo.Date T
      WHERE T.Date = O.RequiredDate) AS DueDateKey,
    ( SELECT DateKey FROM dbo.Date T
      WHERE T.Date = O.ShippedDate) AS ShippedDateKey,
    ShipVia AS ShipperKey, P.ProductID AS ProductKey,
    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.OrderDetails D,
        Northwind.dbo.Products P
WHERE O.OrderID = D.OrderID AND D.ProductID = P.ProductID
```

Table of Contents

Implementing ETL Processes in SQL

- Approach:
  - Implement each control task with an SQL procedure,
  - Implement each data task with an SQL statement
- We can leverage the expressive power of SQL
- Overall Northwind ETL process

```
DROP FUNCTION IF EXISTS NorthwindDWLoad();
CREATE FUNCTION NorthwindDWLoad()
RETURNS VOID LANGUAGE plpgsql AS $$
BEGIN
    PERFORM ContinentCountryStateLoad(); PERFORM TempCitiesLoad();
    PERFORM CategoryLoad(); PERFORM CityLoad();
    PERFORM DateLoad(); PERFORM SupplierLoad();
    PERFORM CustomerLoad(); PERFORM ProductLoad();
    PERFORM ShipperLoad(); PERFORM EmployeeLoad();
    PERFORM SalesLoad(); PERFORM TerritoriesLoad();
END; $$;
```

- The ETL process is divided into procedures that correspond to the tasks in the BPMN specification

Implementing ETL Processes in SQL

- Loading the Category level

```
DROP FUNCTION IF EXISTS CategoryLoad();
CREATE FUNCTION CategoryLoad()
RETURNS VOID AS $$
BEGIN
    INSERT INTO Category(CategoryKey, CategoryName, Description)
    SELECT Categories.*
    FROM dblink('dbname=Northwind port=5432 user=xxx password=xxx',
        'SELECT CategoryID, CategoryName, Description FROM Categories') AS
        Categories(CategoryID int, CategoryName varchar(40), Description text);
END; $$;
```

- dblink is used for executing a query in a remote database

Implementing ETL Processes in SQL

- Loading the City level (beginning)

```
DROP FUNCTION IF EXISTS CityLoad();
CREATE FUNCTION CityLoad()
RETURNS VOID AS $$
BEGIN
    -- Drop temporary tables
    DROP TABLE IF EXISTS StateCountry;
    -- Insert cities without state
    INSERT INTO City (CityName, CountryKey)
    SELECT City, CountryKey
    FROM TempCities, Country
    WHERE State IS NULL AND Country = CountryName;
    -- Insert non-matching tuples into error table
    INSERT INTO BadCities (City, State, Country)
    SELECT City, State, Country
    FROM TempCities
    WHERE State IS NULL AND NOT EXISTS (
        SELECT * FROM City WHERE StateKey IS NULL AND City = CityName );
        ...
```

Implementing ETL Processes in SQL

- Loading the City level (cont.)

```
    ...
    -- Create lookup table
    CREATE TEMPORARY TABLE StateCountry AS
    SELECT * FROM State S NATURAL JOIN Country C;
    -- Insert cities with state
    INSERT INTO City (CityName, StateKey)
    SELECT City, StateKey
    FROM TempCities, StateCountry
    WHERE (State = StateName AND Country = CountryName) OR
        (State = EnglishStateName and Country = CountryName) OR
        (State = StateName and Country = CountryCode);
    -- Insert non-matching tuples into error table
    INSERT INTO BadCities (City, State, Country)
    SELECT City, State, Country
    FROM TempCities
    WHERE State IS NOT NULL AND NOT EXISTS (
        SELECT * FROM City
        WHERE StateKey IS NOT NULL AND City = CityName );
END; $$;
```

- Temporary tables: exists for the duration of a database session
- Automatically dropped at the end of a session, or optionally at the end of the current transaction