ASSIGNMENT NUMBER: A1

TITLE	Design multidimensional Data cube
PROBLEM STATEMENT /DEFINITION	For an organization of your choice, choose a set of business processes. Design star / snow flake schemas for analyzing these processes. Create a fact constellation schema by combining them. Extract data from different data sources, apply suitable transformations and load into destination tables using an ETL tool. For Example: Business Origination: Sales, Order, Marketing Process
OBJECTIVE	• To understand concept of Data cube • Understand different pre-processing techniques • To study ETL tool.
S/W PACKAGES AND HARDWARE APPARATUS USED	• Fedora 20 /Windows 10 • Open source ETL tool for Linux/Windows - Pentaho
REFERENCES	• Han, Jiawei Kamber, Micheline Pei and Jian, "Data Mining: Concepts and Techniques", Elsevier Publishers Second Edition, ISBN: 9780123814791, 9780123814807. • https://help.pentaho.com/Documentation/5.1/0L0/0Y0/040/000 • **https://help.pentaho.com/Documentation/7.1/0J0/0C0/020**
STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept, Architecture,Syntax etc • Test cases • Output • Conclusion

Date: 04/07/2018

Title: Design multidimensional Data cube

Problem Definition: For an organization of your choice, choose a set of business processes. Design star / snow flake schemas for analysing these processes. Create a fact constellation schema by combining them. Extract data from different data sources, apply suitable transformations and load into destination tables using an ETL tool. For Example: Business Origination: Sales, Order, and Marketing Process

Learning Objectives

- To understand concept of Data cube
- Understand different preprocessing techniques
- To study ETL tool.

Learning Outcomes

- To design Multidimensional data cube
- To identify & apply different preprocessing techniques
- To construct data cube using ETL tool.

Theory-

A data warehouse is a **subject-oriented, integrated, time-variant, and nonvolatile** collection of data in support of management's decision-making process. A data warehouse is usually modeled by a multidimensional data structure, called a **data cube**, in which each dimension corresponds to an attribute or a set of attributes in the schema, and each cell stores the value of some aggregate measure such as count, or sum(sales amount). A data cube provides a multidimensional view of data and allows the pre-computation and fast access of summarized data.

Data warehouse systems use back-end tools and utilities to populate and refresh their data. These tools and utilities include the following functions:

Data extraction: This typically gathers data from multiple, heterogeneous, and external sources

Data cleaning: This detects errors in the data and rectifies them when possible

Data transformation: This converts data from legacy or host format to warehouse format.

Load: This sorts, summarizes, consolidates, computes views, checks integrity, and builds indices and partitions.

Refresh: This propagates the updates from the data sources to the warehouse.

A data cube for **AllElectronics**: A data cube for summarized sales data of AllElectronics is presented in Figure 1(a). The cube has three dimensions: address (with city values Mumbai,

Dehli, Gurgaon), time (with quarter values Q1, Q2, Q3, Q4), and item(with item type values Mouse, Mobile, Modem). The aggregate value stored in each cell of the cube is sales amount (in thousands).

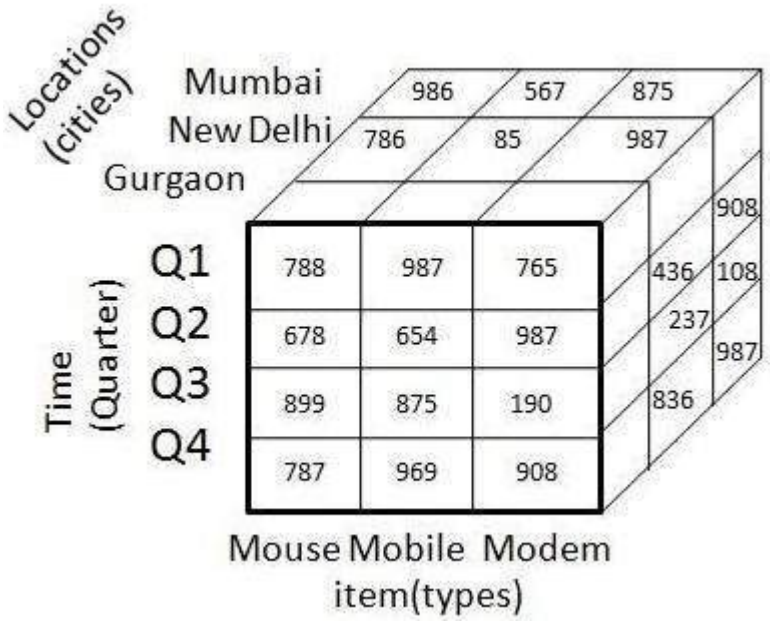


Fig. 1(a) Data Cube

In general terms, **dimensions** are the perspectives or entities with respect to which an organization wants to keep records. For example, AllElectronics may create a sales data warehouse in order to keep records of the store's sales with respect to the dimensions time, item, branch, and location. These dimensions allow the store to keep track of things like monthly sales of items and the branches and locations at which the items were sold. Each dimension may have a table associated with it, called a **dimension table**, which further describes the dimension. For example, a dimension table for item may contain the attributes item name, brand, and type. Dimension tables can be specified by users or experts, or automatically generated and adjusted based on data distributions. A multidimensional data model is typically organized around a central theme, such as sales. This theme is represented by a **fact table**. **Facts** are numeric measures. Think of them as the quantities by which we want to analyze relationships between dimensions.

Examples of facts for a sales data warehouse include dollars sold (sales amount in dollars), units sold (number of units sold), and amount budgeted. The fact table contains the names of the facts, or measures, as well as keys to each of the related dimension tables.

The most popular data model for a data warehouse is a multidimensional model, which can exist in the form of a star schema, a snowflake schema, or a fact constellation schema.

Star schema: The most common modeling paradigm is the star schema, in which the data warehouse contains (1) a large central table (fact table) containing the bulk of the data, with no redundancy, and (2) a set of smaller attendant tables (dimension tables), one for each

dimension.

A star schema for AllElectronics sales is shown in Figure 1(b). Sales are considered along four dimensions: time, item, branch, and location. The schema contains a central fact table for sales that contains keys to each of the four dimensions, along with two measures: dollars sold and units sold. To minimize the size of the fact table, dimension identifiers (e.g., time key and item key) are system-generated identifiers.

Snowflake schema: The snowflake schema is a variant of the star schema model, where some dimension tables are normalized, thereby further splitting the data into additional tables. The major difference between the snowflake and star schema models is that the dimension tables of the snowflake model may be kept in normalized form to reduce redundancies. Such a table is easy to maintain and saves storage space. However, this space savings is negligible in comparison to the typical magnitude of the fact table. Furthermore, the snowflake structure can reduce the effectiveness of browsing, since more joins will be needed to execute a query. Consequently, the system performance may be adversely impacted. Hence, although the snowflake schema reduces redundancy, it is not as popular as the star schema in data warehouse design. A snowflakes schema for AllElectronics sales is shown in Figure 1(c).

Fact constellation: Sophisticated applications may require multiple fact tables to share dimension tables. This kind of schema can be viewed as a collection of stars, and hence is called a galaxy schema or a fact constellation. A fact constellation schema for AllElectronics sales is shown in Figure 1(d).

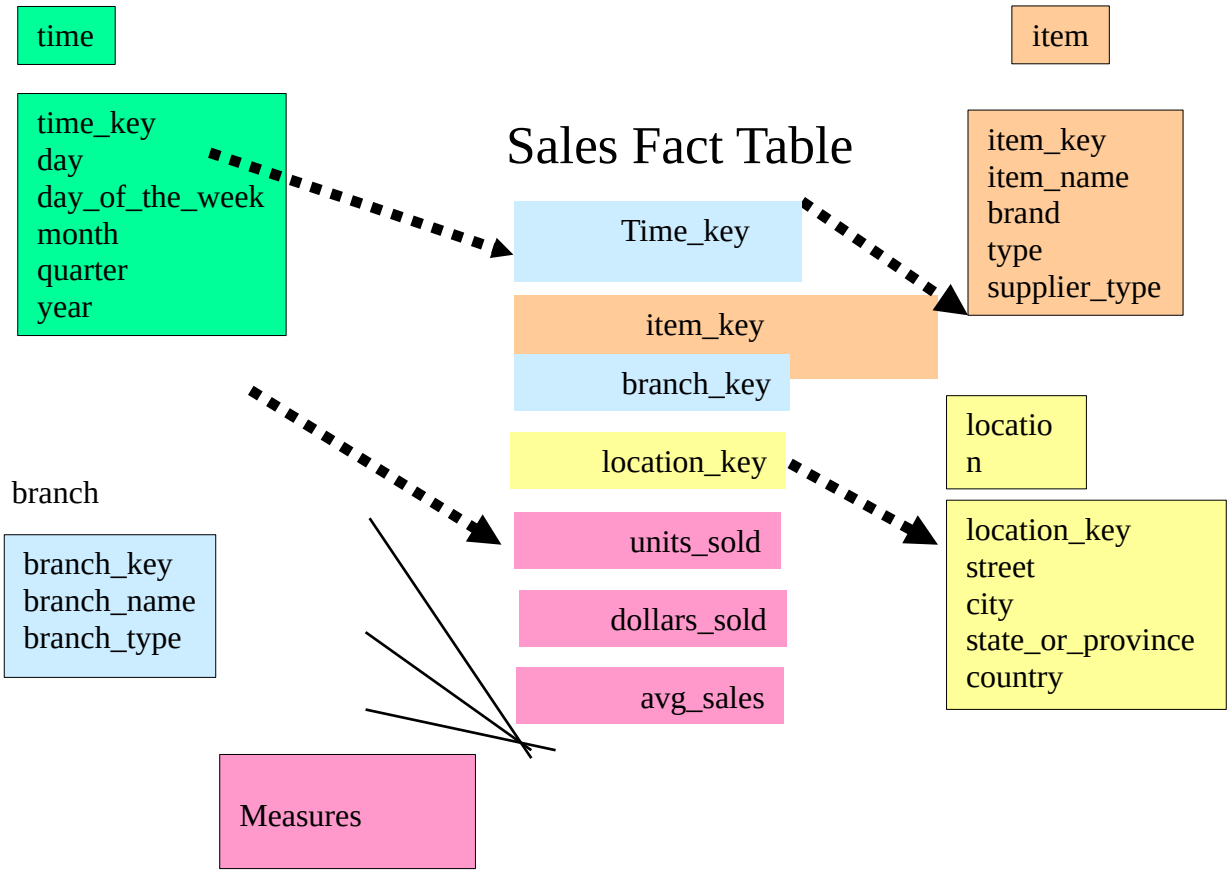


Fig. 1(b) Star Schema for AllElectronics

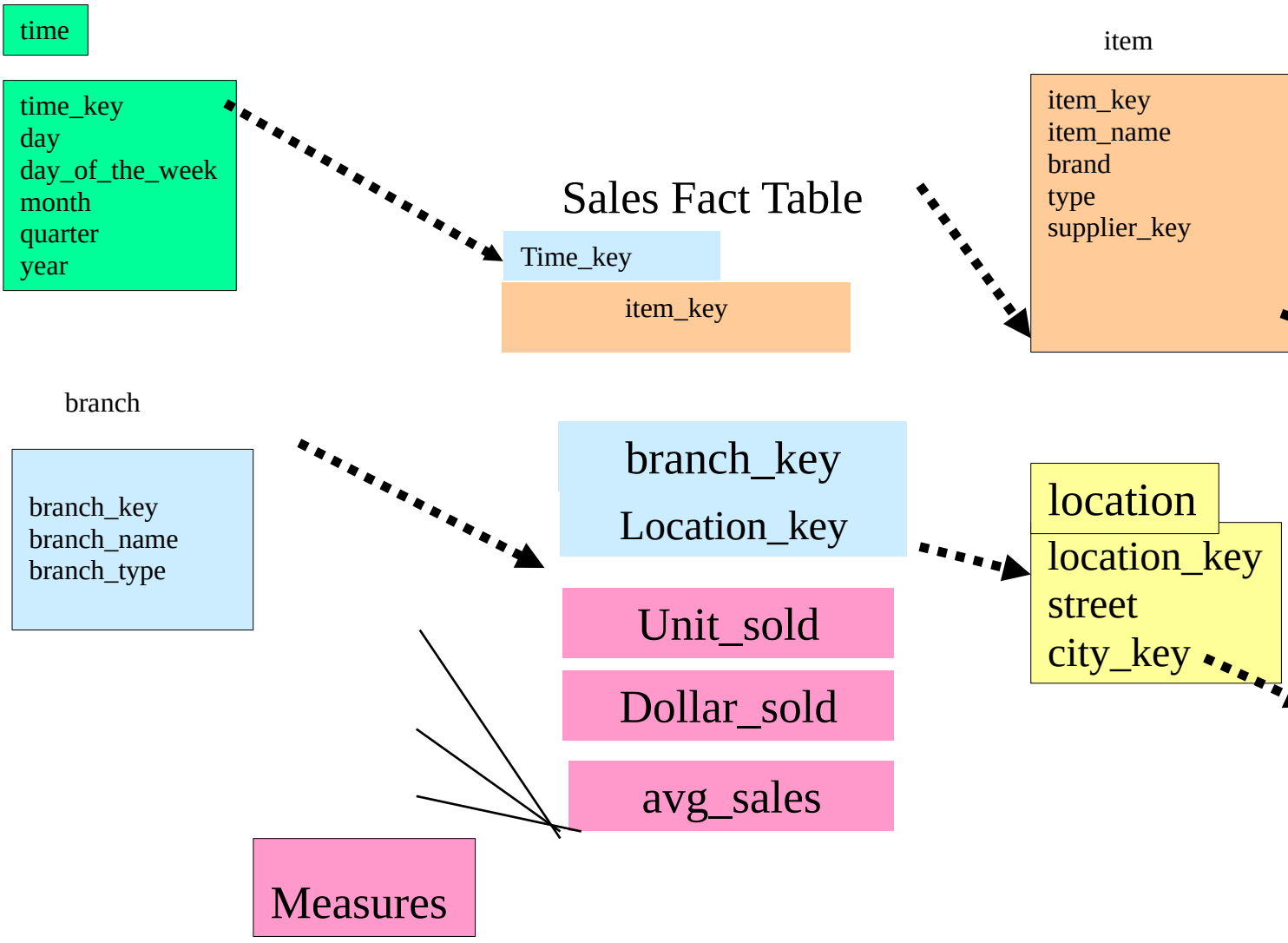
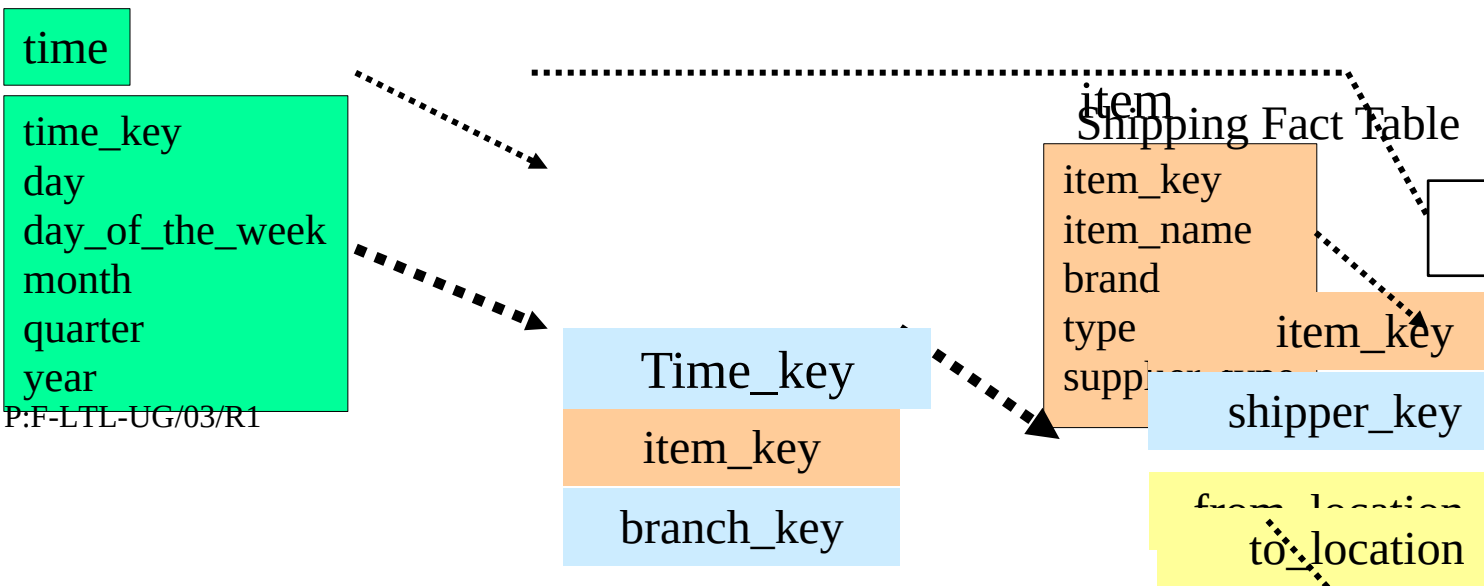


Fig. 1(b) Snowflakes Schema for AllElectronics



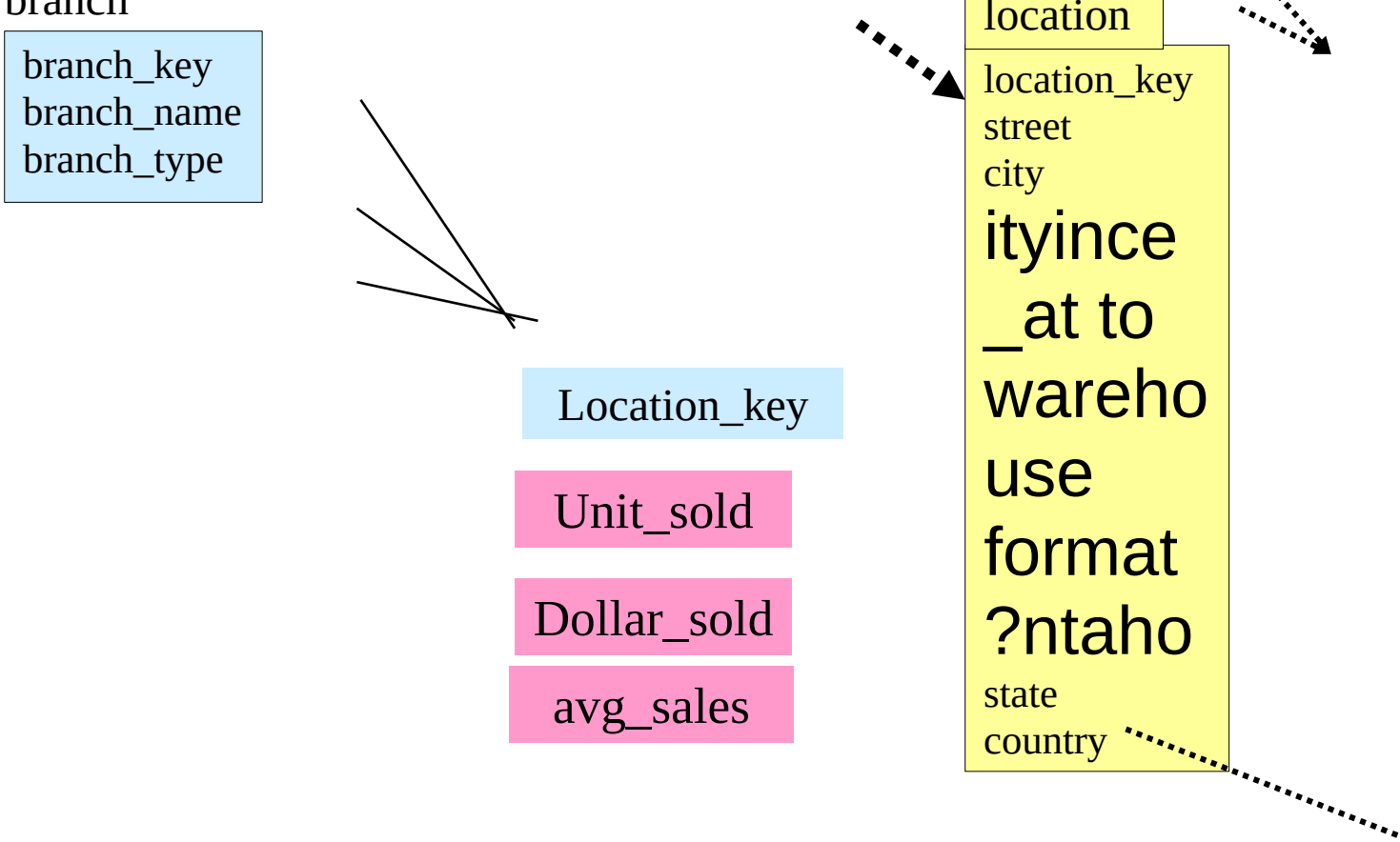


Fig. 1(b) Fact constellation Schema for AllElectronics

Steps to Install Pentaho

Data Integration:- Pentaho Data Integration (PDI) provides access to an Extraction, Transformation, and Loading (ETL) engine that captures the right data, cleanses the data, and stores data using a uniform format that is accessible and relevant to end users and IoT technologies.

`Download` Pentaho Data Integration (PDI) Tool from following url
https://excellmedia.dl.sourceforge.net/project/pentaho/Data%20Integration/4.1.0-stable/pdi-ce-4.1.0-stable.zip

`Follow following steps to install Pentaho on Linux. The detail information for all the steps mentioned in url :` https://help.pentaho.com/Documentation/7.0/0F0/0P0/020/0B0

1. Create the Pentaho User.
2. Create Linux Directory Structure.
3. Install Java.
4. Install the Web Application Server, if you are installing on your own web application server.
5. Install the Pentaho Repository Host Database.
6. Download and Unpack the Installation Files.
7. Set Environment Variables.
8. Advanced Linux Considerations.

To start Pentaho :
go to terminal
cd Home/data-integration/
run spoon.sh on linux >>./soopn.sh

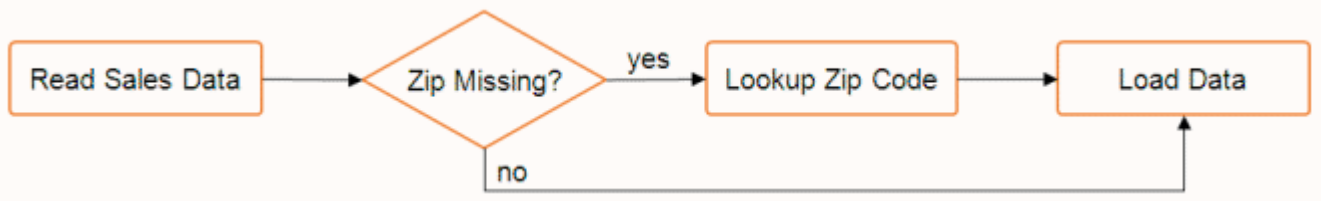
To install Pentaho on Windows

Install JAVA
Install Apache/Tomcat
Run spoon.bat from Data Integration folder

Steps for Preprocessing using Pentaho

The Data Integration perspective of Spoon allows you to create two basic file types: transformations and jobs. Transformations are used to describe the data flows for ETL such as reading from a source, transforming data and loading it into a target location. Jobs are used to coordinate ETL activities such as defining the flow and dependencies for what order transformations should be run, or prepare for execution by checking conditions such as, "Is my source file available?" or "Does a table exist in my database?"

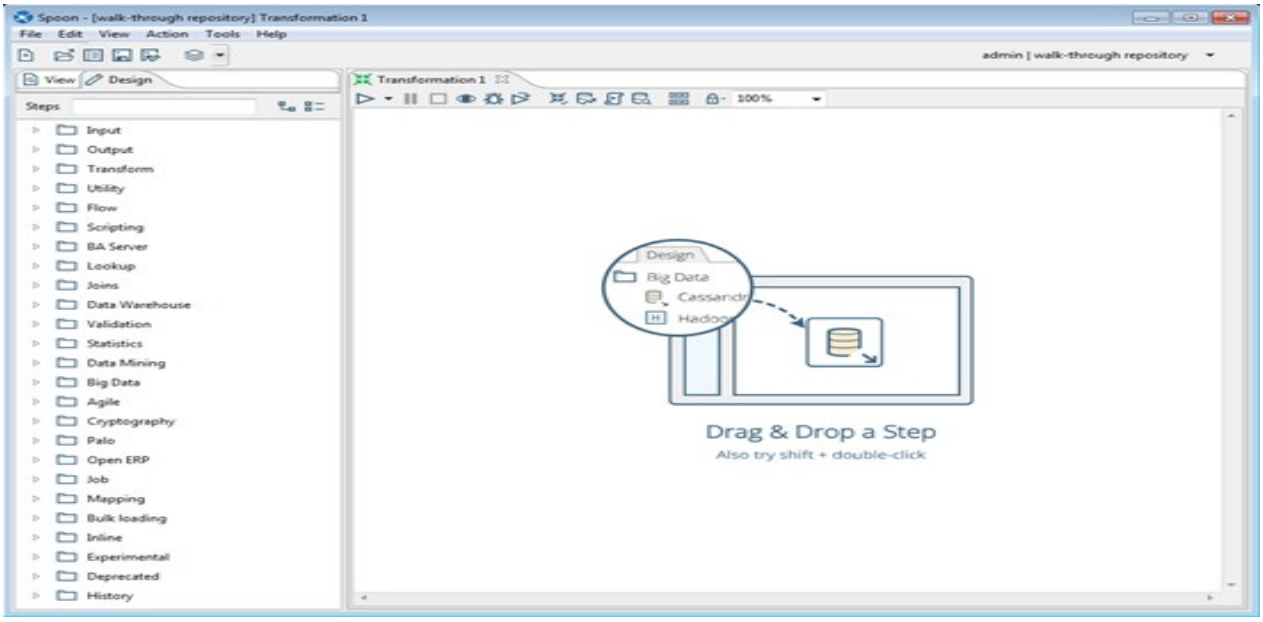
This exercise will step you through building your first transformation with Pentaho Data Integration introducing common concepts along the way. The exercise scenario includes a flat file (.csv) of sales data that you will load into a database so that mailing lists can be generated. Several of the customer records are missing postal codes (zip codes) that must be resolved before loading into the database. The logic looks like this:



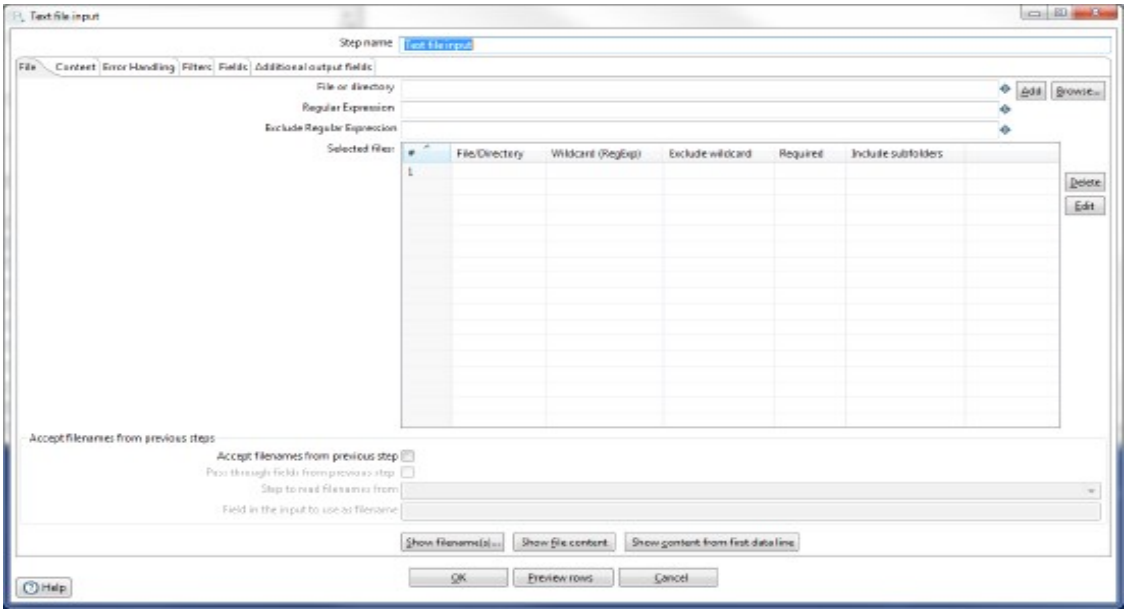
Retrieving Data from a Flat File

First connect to a repository, then follow the instructions below to retrieve data from a flat file.

1. Select **File > New > Transformation** in the upper left corner of the **Spoon** window to create a new transformation.



2. Under the **Design** tab, expand the **Input** node; then, select and drag a **Text File Input** step onto the canvas.
3. Double-click on the **Text File** input step. The **Text file input** window appears. This window allows you to set the properties for this step.



4. In the **Step Name** field, type **Read Sales Data**. This renames the **Text file input** step to **Read Sales Data**.
5. Click **Browse** to locate the source file, **sales_data.csv**, available at ...\design-tools\data-integration\samples\transformations\files. (The **Browse** button appears near the top right side of the window near the **File or Directory** field.) Click **Open**. The path to the source file appears in the **File or directory** field.
6. Click **Add**. The path to the file appears under **Selected Files**.
7. To look at the contents of the sample file perform the following steps:

a. Click the **Content** tab, then set the **Format** field to **Unix**.

b. Click the **File** tab again and click the **Show file content** near the bottom of the window.

c. The **Nr of lines to view** window appears. Click the **OK** button to accept the default.

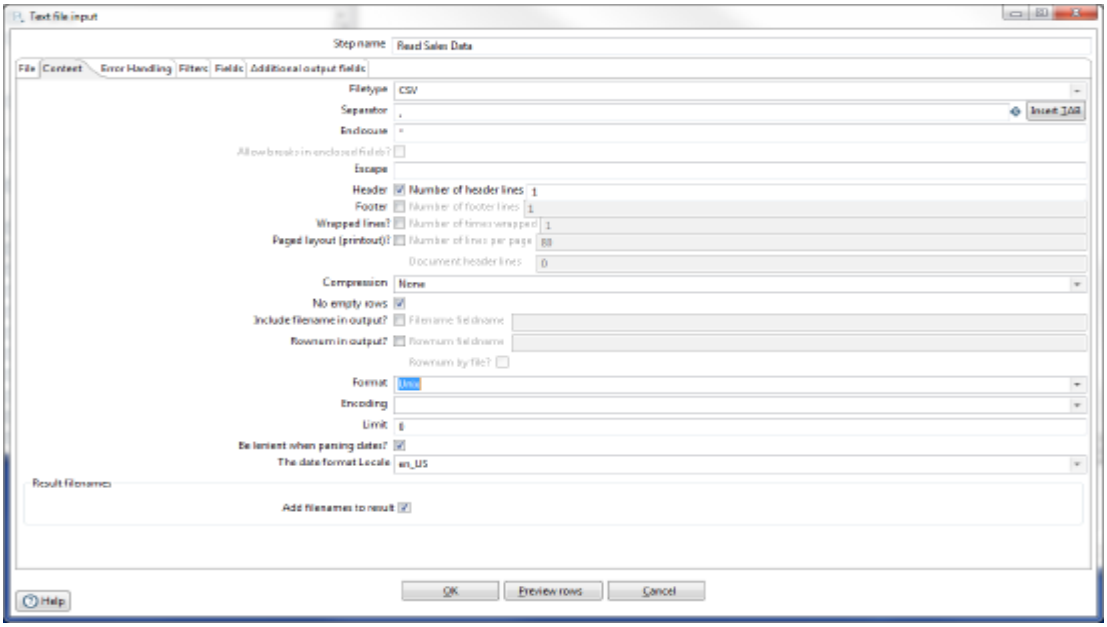
d. The **Content of first file** window displays the file. Examine the file to see how that input file is delimited, what enclosure character is used, and whether or not a header row is present. In the sample, the input file is comma (,) delimited, the enclosure character being a quotation mark (“) and it contains a single header row containing field names.

e. Click the **Close** button to close the window.

8. To provide information about the content, perform the following steps:

a. Click the **Content** tab. The fields under the **Content** tab allow you to define how your data is formatted.

b. Make sure that the **Separator** is set to comma (,) and that the **Enclosure** is set to quotation mark ("). Enable **Header** because there is one line of header rows in the file.



c. Click the **Fields** tab and click **Get Fields** to retrieve the input fields from your source file. When the **Nr of lines to sample** window appears, enter **0** in the field then click **OK**.

d. if the **Scan Result** window displays, click **Close** to close the window.

e. To verify that the data is being read correctly:

a. Click the **Content** tab, then click **Preview Rows**.

b. In the **Enter preview size** window click **OK**. The **Examine preview data** window appears.

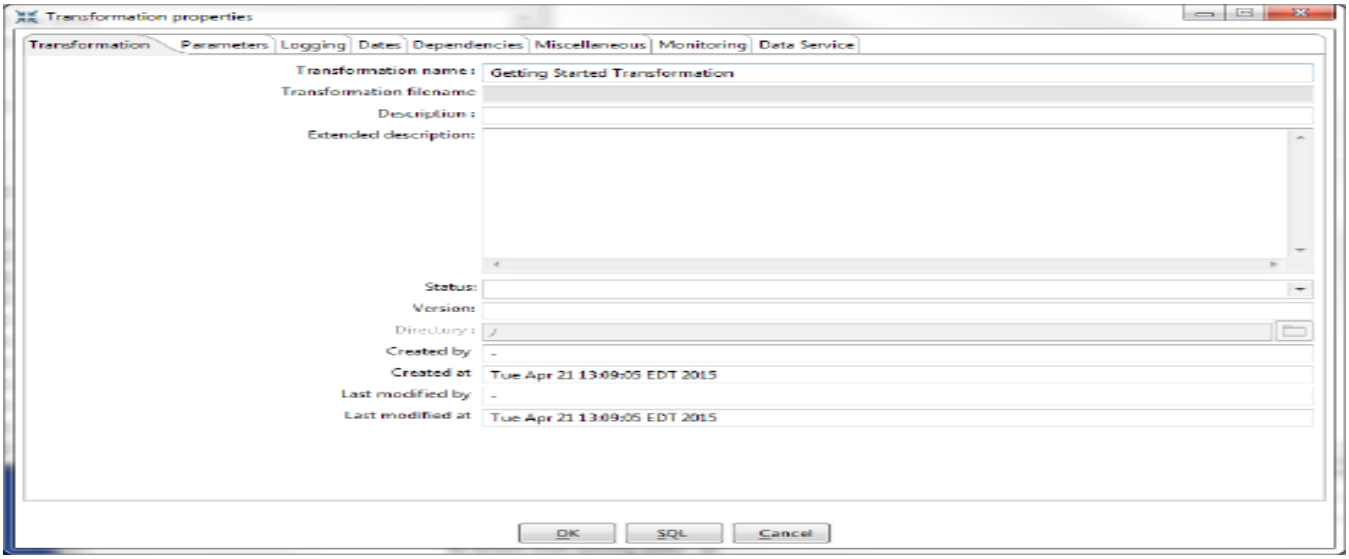
c. Review the data, then click **Close**.

f. Click **OK** to save the information that you entered in the step.

g. To save the transformation, do these things.

a. Select **File > Save** to save the transformation.

b. The **Transformation Properties** window appears. In the **Transformation Name** field, type **Getting Started Transformation**. (Note that the **Transformation Properties** window appears because you are connected to a repository. If you were not connected to the repository, the standard save window would appear.)



c. In the **Directory** field, click the folder icon.

d. Expand the **Home** directory and double-click the folder in which you want to save the transformation. Your transformation is saved in the Pentaho Repository.

e. Click **OK** to close the **Transformation Properties** window.

Filter Records with Missing Postal Codes

After completing Retrieve Data from a Flat File, you are ready to add the next step to your transformation. The source file contains several records that are missing postal codes. Use the Filter Rows transformation step to separate out those records so that you can resolve them in a later exercise.

1. Add a **Filter Rows** step to your transformation. Under the **Design** tab, select **Flow** > **Filter Rows**.
2. Create a hop between the **Read Sales Data** step and the **Filter Rows** step. Hops are used to describe the flow of data in your transformation. To create the hop, click the **Read Sales Data** (Text File input) step, then press the <**SHIFT**> key down and draw a line to the **Filter Rows** step.

3. Double-click the **Filter Rows** step. The **Filter Rows** window appears.

4. In the **Step Name** field type, **Filter Missing Zips**.
5. Under **The condition**, click <field>. The **Fields** window appears. These are the conditions you can select.
6. In the **Fields** window select **POSTALCODE** and click **OK**.
7. Click on the comparison operator (set to = by default) and select the **IS NOT NULL** from the **Functions:** window that appears.
8. Click **OK** to close the **Functions:** window.
9. Click **OK** to exit the **Filter Rows** window.
   *Note: You will return to this step later and configure the **Send true data to step** and **Send false data to step** settings after adding their target steps to your transformation.*
10. Select **File > Save** to save your transformation.

Loading Your Data into a Relational Database

After completing Filter Records with Missing Postal Codes, you are ready to take all records exiting the **Filter rows** step where the **POSTALCODE** was not null (the **true**condition), and load them into a database table.

1. Under the **Design** tab, expand the contents of the **Output** node.
2. Click and drag a **Table Output** step into your transformation. Create a hop between the **Filter Missing Zips** and **Table Output** steps. In the dialog that appears, select **Result is TRUE**.
3. Double-click the **Table Output** step to open its edit properties dialog box.
4. Rename your Table Output Step to **Write to Database**.
5. Click **New** next to the **Connection** field. You must create a connection to the database. The **Database Connection** dialog box appears.
6. Provide the settings for connecting to the database.

Field	Setting
Connection Name:	Sample Data
Connection Type:	Hypersonic
Host Name	localhost

Database Name	sampledata
Port Number	9001
User Name	pentaho_admin
Password	password (If "password" does not work, please check with your system administrator.)

1. Click **Test** to make sure your entries are correct. A success message appears. Click **OK**.
   Note: If you get an error when testing your connection, ensure that you have provided the correct settings information as described in the table and that the sample database is running. See Start and Stop the Pentaho Server for information about how to start the Pentaho Server.
2. Click **OK**, to exit the **Database Connections** window.
3. Type **SALES_DATA** in the **Target Table** text field.
4. Since this table does not exist in the target database, you will need use the software to generate the Data Definition Language (DDL) to create the table and execute it. DDLs are the SQL commands that define the different structures in a database such as CREATE TABLE.

a. In the **Table Output** window, enable the **Truncate Table** property.

b. Click the **SQL** button at the bottom of the **Table output** dialog box to generate the DDL for creating your target table.

c. The **Simple SQL editor** window appears with the SQL statements needed to create the table.

d. Click **Execute** to execute the SQL statement.

e. The **Results of the SQL statements** window appears. Examine the results, then click **OK** to close the **Results of the SQL statements** window.

f. Click **Close** in the **Simple SQL editor** window to close it.

g. Click **OK** to close the **Table output** window.

5. Save your transformation.

Retrieving Data from Your Lookup File

After Loading Your Data into a Relational Database, you are ready to retrieve data from your lookup file. You have been provided a second text file containing a list of cities, states, and postal codes that you will now use to look up the postal codes for all of the records where they were missing (the ‘false’ branch of your Filter rows step). First, you will use a Text file input step to read from the source file, then you will use a Stream lookup step to bring the resolved Postal Codes into the stream.

1. Add a new **Text File Input** step to your transformation. In this step you will retrieve the records from your lookup file. Do not add a hop yet.
2. Open the **Text File Input** step window, then enter **Read Postal Codes** in the **Step name** property.
3. Click **Browse** to locate the source file, **Zipssortedbycitystate.csv**, located at ...\design-tools\data-integration\samples\transformations\files.
4. Click **Add**. The path to the file appears under **Selected files**.
5. To look at the contents of the sample file:

a. Click the **Content** tab, then set the **Format** field to **Unix**.

b. Click the **File** tab again and click the **Show file content** near the bottom of the window.

c. The **Nr of lines to view** window appears. Click the **OK** button to accept the default.

d. The **Content of first file** window displays the file. Examine the file to see how that input file is delimited, what enclosure character is used, and whether or not a header row is present. In the example, the input file is comma (,) delimited, the enclosure character being a quotation mark (“) and it contains a single header row containing field names.

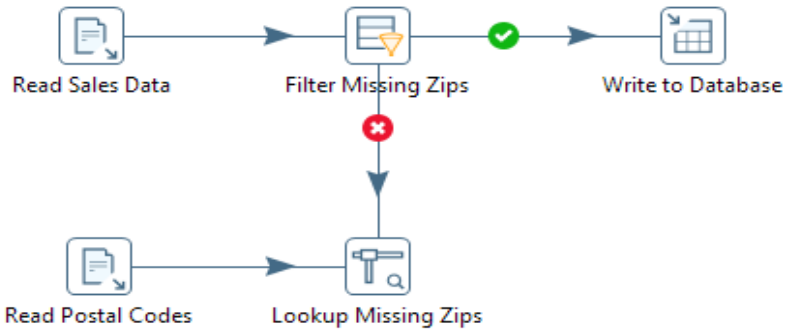
e. Click the **Close** button to close the window.

6. In the **Content** tab, change the **Separator** character to a comma (,). and confirm that the **Enclosure** setting is a quotation mark ("). Make sure the **Header** option is selected.
7. Under the **Fields** tab, click **Get Fields** to retrieve the data from your .csv file.
8. The **Nr of lines to sample** window appears. Enter 0 in the field, then click **OK.**
9. If the **Scan Result** window displays, click **Close** to close it.
10. Click **Preview rows** to make sure your entries are correct. When prompted to enter the preview size, click **OK**. Review the information in the window, then click **Close.**
11. Click **OK** to exit the **Text File input** window.
12. Save the transformation.

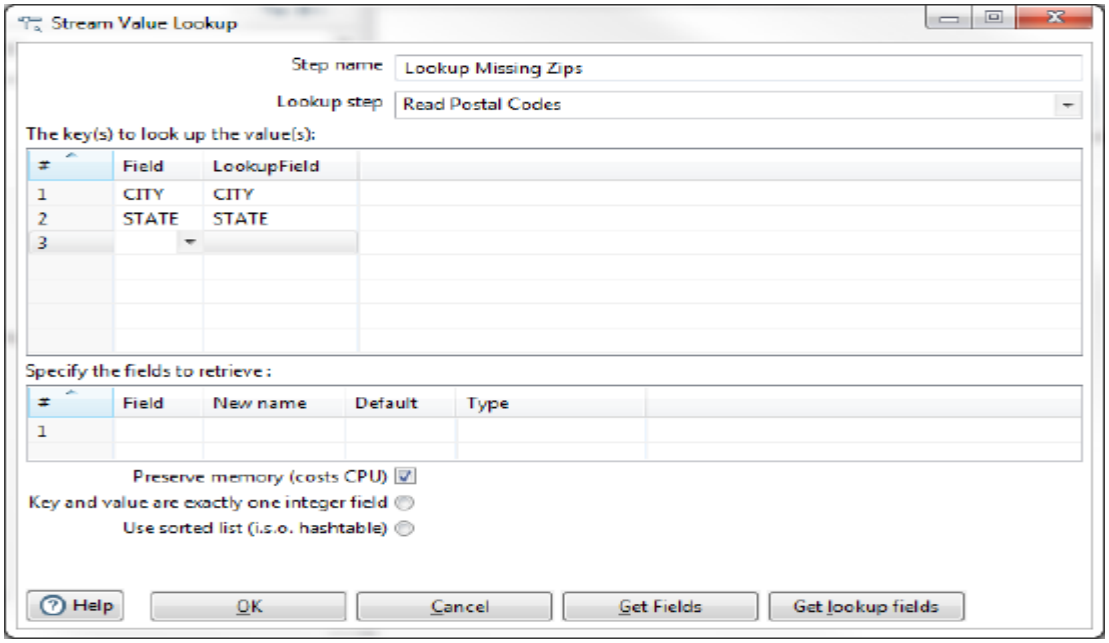
Resolving Missing Zip Code Information

After Retrieving Data from Your Lookup File, you can begin to resolve the missing zip codes.

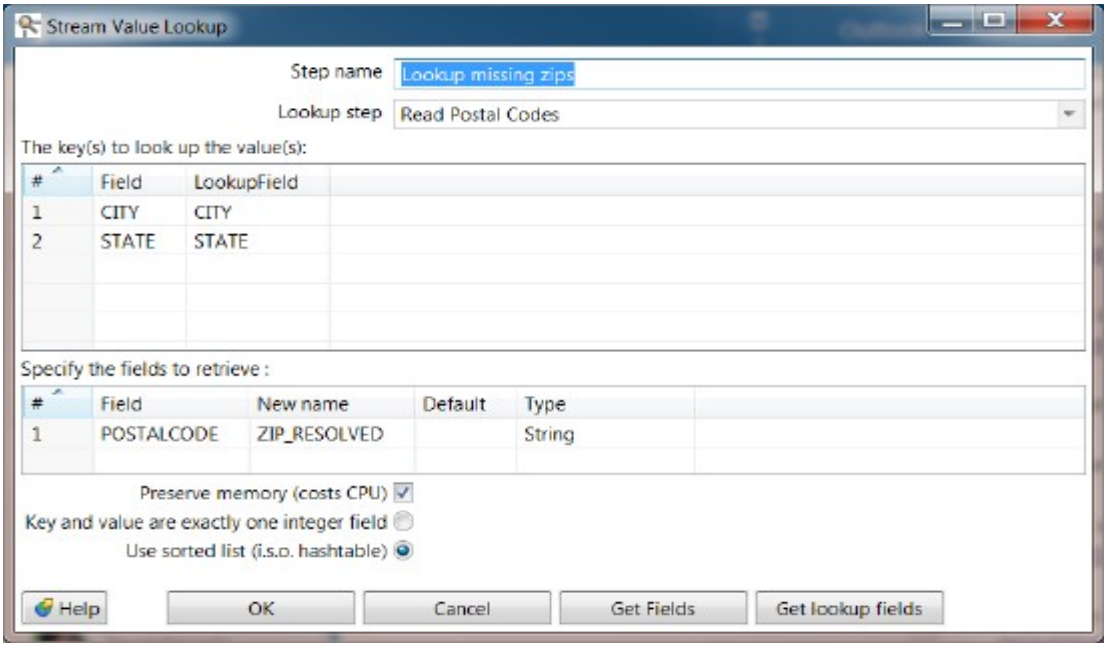
1. Add a **Stream Lookup** step to your transformation. To do this, click the **Design** tab, then expand the **Lookup** folder and choose **Stream Lookup**.
2. Draw a hop from the **Filter Missing Zips** to the **Stream lookup** step. In the dialog box that appears, select **Result is FALSE**.
3. Create a hop from the **Read Postal Codes** step to the **Stream lookup** step.



4. Double-click on the **Stream lookup** step to open the **Stream Value Lookup** window.
5. Rename **Stream Lookup** to **Lookup Missing Zips**.
6. From the **Lookup step** drop-down box, select **Read Postal Codes** as the lookup step.
7. Define the **CITY** and **STATE** fields in the **key(s) to look up the value(s)** table. In **row #1**, click the drop down in the **Field** column and select **CITY**. Then, click in the **LookupField** column and select **CITY**. In **row #2**, click the drop down field in the **Field** column and select **STATE**. Then click in the **LookupField** column and select **STATE**.



8. Click **Get Lookup Fields**.
9. **POSTALCODE** is the only field you want to retrieve. To delete the **CITY** and **STATE** lines, right-click in the line and select **Delete Selected Lines**.
10. In the **New Name** field, give **POSTALCODE** a new name of **ZIP_RESOLVED** and make sure the **Type** is set to **String**.
11. Enable **Use sorted list (i.s.o. hashtable)**.



12. Click **OK** to close the **Stream Value Lookup** edit properties dialog box.
13. Save your transformation.
14. To preview the data:

a. In the canvas, select the **Lookup Missing Zips** step, then right-click. From the menu that appears, select **Preview**.

b. In the **Transformation debug dialog** window, click **Quick Launch** to preview the data flowing through this step.

c. The **Examine preview data** window appears. Notice that the new field, **ZIP_RESOLVED**, has been added to the stream containing your resolved postal codes.

d. Click **Close** to close the window.

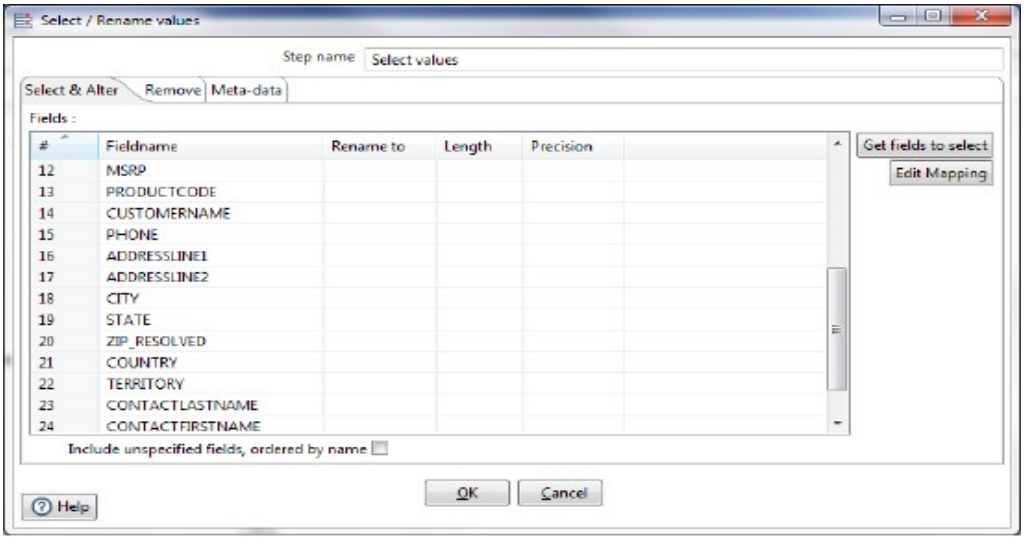
e. If the **Select the preview step** window appears, click the **Close** button.

f. Note that the execution results near the bottom of the **Spoon** window shows updated metrics in the **Step Metrics** tab.

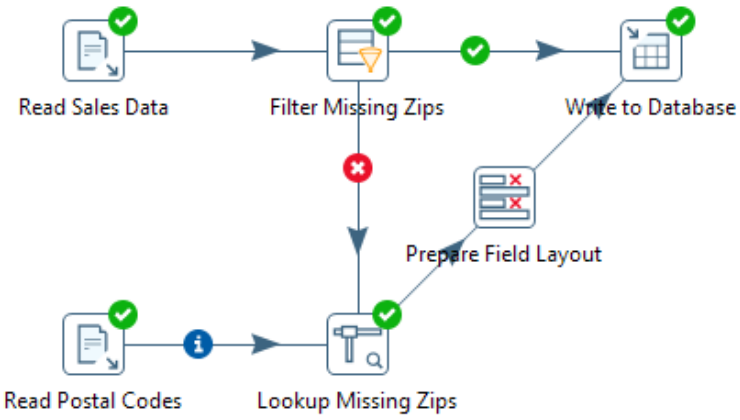
Completing Your Transformation

After you resolve missing zip code information, the last task is to clean up the field layout on your lookup stream. Cleaning up makes it so that it matches the format and layout of your other stream going to the **Write to Database** step. Create a **Select values** step for renaming fields on the stream, removing unnecessary fields, and more.

1. Add a **Select Values** step to your transformation by expanding the **Transform** folder and choosing **Select Values**.
2. Create a hop from the **Lookup Missing Zips** to the **Select Values** step.
3. Double-click the **Select Values** step to open its properties dialog box.
4. Rename the Select Values step to **Prepare Field Layout**.
5. Click **Get fields to select** to retrieve all fields and begin modifying the stream layout.
6. In the **Fields** list, find the **#** column and click the number for the **ZIP_RESOLVED** field. Use <**CTRL**><**UP**> to move **ZIP_RESOLVED** just below the **POSTALCODE** field (the one that still contains null values).
7. Select the old **POSTALCODE** field in the list (line 20) and delete it.



8. The original POSTALCODE field was formatted as an 9-character string. You must modify your new field to match the form. Click the **Meta-Data** tab.
9. In the first row of the **Fields to alter table the meta-data for** section, click in the **Field-name** column and select **ZIP_RESOLVED**.
10. Type **POSTALCODE** in the **Rename** to column; select **String** in the Type column, and type **9** in the **Length** column. Click **OK** to exit the edit properties dialog box.
11. Draw a hop from the **Prepare Field Layout** (Select values) step to the **Write to Database** (Table output) step.
12. When prompted, select the **Main output of the step** option.
13. Save your transformation.

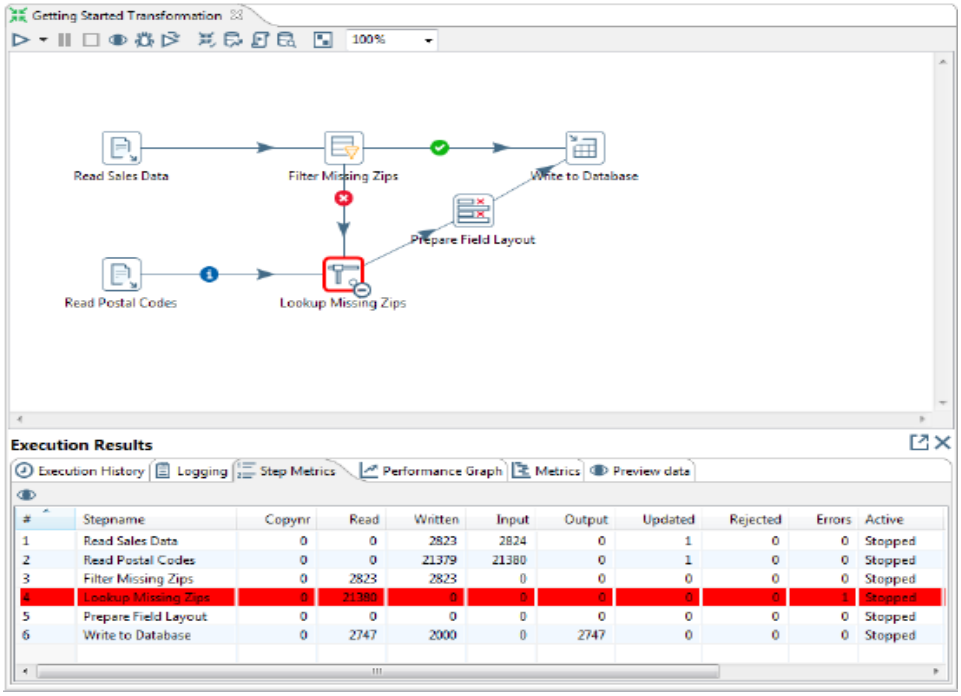


Run Your Transformation

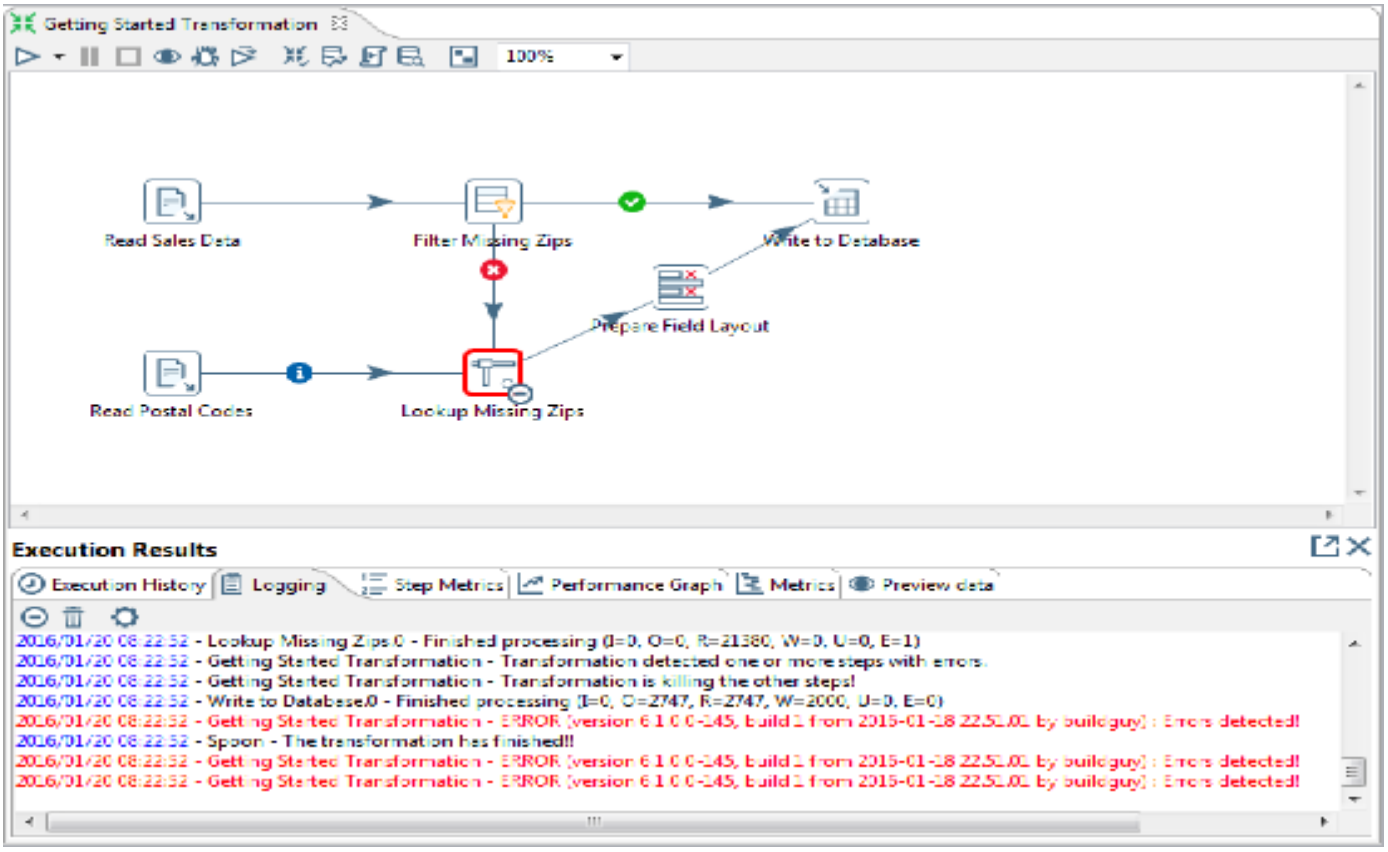
Data Integration provides a number of deployment options. Running a Transformation explains these and other options available for execution. This final part of this exercise to create a transformation focuses exclusively on the **Local** run option.

1. In the PDI client window, select **Action > Run**.
2. The **Run Options** window appears. Keep the default **Pentaho local** option for this exercise. It will use the native Pentaho engine and run the transformation on your local machine. See Run Configurations if you are interested in setting up configurations that use another engine, such as Spark, to run a transformation.
3. Click **Run**. The transformation executes. Upon running the transformation, the **Execution Results** panel opens below the canvas.
4. The **Execution Results** section of the window contains several different tabs that help you to see how the transformation executed, pinpoint errors, and monitor performance.

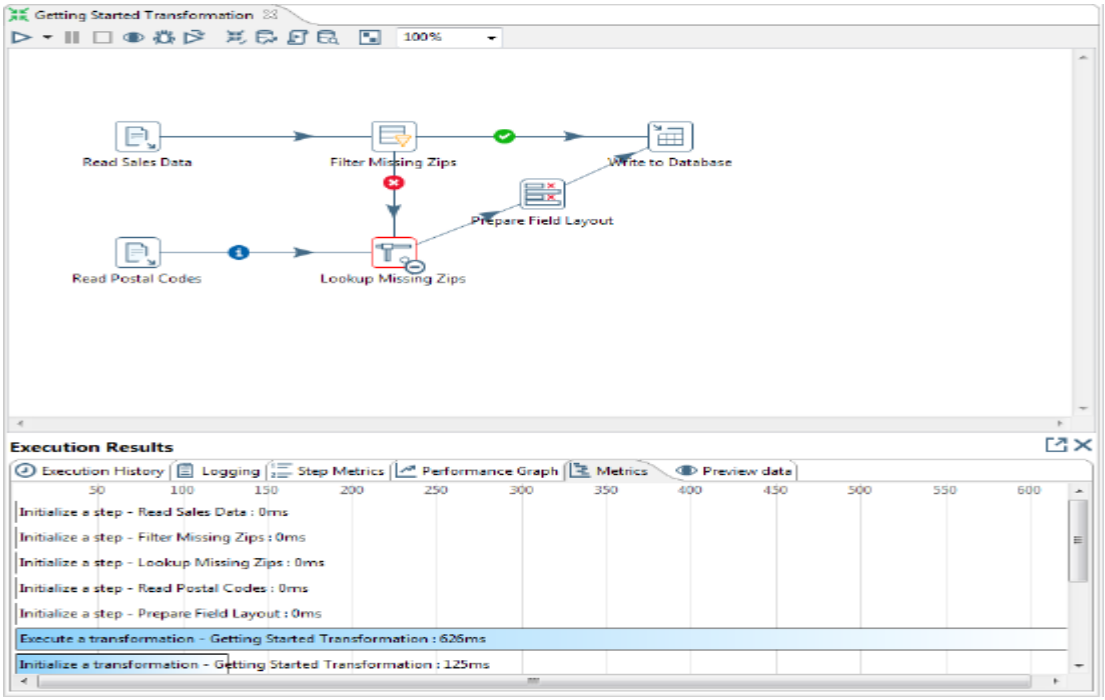
- **Step Metrics** tab provides statistics for each step in your transformation including how many records were read, written, caused an error, processing speed (rows per second) and more. This tab also indicates whether an error occurred in a transformation step. We did not intentionally put any errors in this tutorial so it should run correctly. But, if a mistake had occurred, steps that caused the transformation to fail would be highlighted in red. In the example below, the **Lookup Missing Zips** step caused an error.



- The **Logging** tab displays the logging details for the most recent execution of the transformation. It also allows you to drill deeper to determine where errors occur. Error lines are highlighted in red. In the example below, the **Lookup Missing Zips** step caused an error because it attempted to lookup values on a field called **POSTALCODE2**, which did not exist in the lookup stream.



- The **Execution History** tab provides you access to the Step Metrics and log information from previous executions of the transformation. This feature works only if you have configured your transformation to log to a database through the Logging tab of the Transformation Settings dialog. For more information on configuring logging or viewing the execution history, see Create DI Solutions.
- The **Performance Graph** allows you to analyze the performance of steps based on a variety of metrics including how many records were read, written, caused an error, processing speed (rows per second) and more. Like the Execution History, this feature requires you to configure your transformation to log to a database through the **Logging** tab of the **Transformation Settings** dialog box.
- The **Metrics tab** allows you to see a Gantt chart after the transformation or job has run. This shows you information such as how long it takes to connect to a database, how much time is spent executing a SQL query, or how long it takes to load a transformation.



- The **Preview Data** tab displays a preview of the data.