CHAPTER 5

Data-Mining Predictions

Improving human-to-computer interaction through speech processing is just one area of computing that can benefit from enhanced computing. On the other side of the interface is the backend, which usually ties in to a database. It is here that enhanced computing can help users get the most from their data.

Over the past ten years, there has been a dramatic increase in computer usage—and in the number of home users. Electronic commerce has resulted in the collection of vast amounts of customer and order information. In addition, most businesses have automated their processes and converted legacy data into electronic formats. Businesses large and small are now struggling with the question of what to do with all the electronic data they have collected.

Data warehousing is a multi-billion-dollar industry that involves the collection, organization, and storage of large amounts of data. Data cubes—structures comprising one or more tables in a relational database—are built so that data can be examined through multiple dimensions. This allows databases containing millions of records and hundreds of attributes to be explored instantly.

Data mining is the process of extracting meaningful information from large quantities of data. It involves uncovering patterns in the data and is often tied to data warehousing because it makes such large amounts of data usable. Data elements are grouped into distinct categories so that predictions can be made about other pieces of data. For example, a bank may wish to ascertain the characteristics that typify customers who pay back loans. Although this could be done with database queries, the bank would first have to know what customer attributes to query for. Data min-

ing can be used to identify what those attributes are and then make predictions about future customer behavior.

Data mining is a technique that has been around for several years. Unfortunately, many of the original tools and techniques for mining data were complex and difficult for beginners to grasp. Microsoft and other software makers have responded by creating easier-to-use data-mining tools. A 2004 report titled "The Golden Vein" by the Economist.com states:

> As the cost of storing data plummets and the power of analytic tools improves, there is little likelihood that enthusiasm for data mining, in all its forms, will diminish.

This is the first of two chapters that will examine how a fictional retailer named Savings Mart was able to utilize Microsoft's **Analysis Services**, included with Microsoft SQL Server**,** to improve operational efficiencies and reduce costs. The present chapter will examine a standalone Windows program named LoadSampleData which is used to load values into a database and generate random purchases for several of the retailer's stores. A mining model will then be created based on shipments to each store. The mining model will be the first step toward revising the way Savings Mart procedurally handles product orders and shipments.

Chapter 6 will extend the predictions made in this chapter through the use of a Windows service designed to automate mining-model processing and the application of processing results. Finally, a modified version of the LoadSampleData program will be used to verify that Savings Mart was able to successfully lower its operating costs.

The chapter also includes a Microsoft case study which examines a real company that used Analysis Services to build a data-mining solution. In the case study, a leaser of technology equipment needed to predict when clients would return their leased equipment. By using Analysis Services, it was able to quickly build a data-mining solution that helped to reduce costs and more accurately predict the value of assets.

Introducing Data Mining with SQL Server

Although SQL Server 7.0 offered Online Analytical Processing (OLAP) as OLAP Services, it was not until the release of SQL Server 2000 that data-mining algorithms were included. Analysis Services comes bundled with SQL Server as a separate install. It allows developers to build complex

OLAP cubes and then utilize two popular data-mining algorithms to process data within the cubes.

Of course, it is not necessary to build OLAP cubes in order to utilize data-mining techniques. Analysis Services also allows mining models to be built against one or more tables from a relational database. This is a big departure from traditional data-mining methodologies. It means that users can access data-mining predictions without the need for OLAP services.

Data mining involves the gathering of knowledge to facilitate better decision-making. It is meant to empower organizations to learn from their experiences—or in this case their historical data—in order to form proactive and successful business strategies. It does not replace decision-makers, but instead provides them with a useful and important tool.

The introduction of data-mining algorithms with SQL Server represents an important step toward making data mining accessible to more companies. The built-in tools allow users to visually create mining models and then train those models with historical data from relational databases.

Data-Mining Algorithms

Data mining with Analysis Services is accomplished using one of two popular mining algorithms—decision trees and clustering. These algorithms are used to find meaningful patterns in a group of data and then make predictions about the data. Table 5.1 lists the key terms related to data mining with Analysis Services.

Decision Trees

Decision trees are useful for predicting exact outcomes. Applying the decision trees algorithm to a training dataset results in the formation of a tree that allows the user to map a path to a successful outcome. At every node along the tree, the user answers a question (or makes a "decision"), such as "years applicant has been at current job (0–1, 1–5, > 5 years)."

The decision trees algorithm would be useful for a bank that wants to ascertain the characteristics of good customers. In this case, the predicted outcome is whether or not the applicant represents a bad credit risk. The outcome of a decision tree may be a Yes/No result (applicant is/is not a bad credit risk) or a list of numeric values, with each value assigned a probability. We will see the latter form of outcome later in this chapter.

The training dataset consists of the historical data collected from past loans. Attributes that affect credit risk might include the customer's educational level, the number of kids the customer has, or the total household

Table 5.1 Key terms related to data mining with Analysis Services.

Term	Definition
Case	The data and relationships that represent a single object you wish to analyze. For example, a product and all its attributes, such as Product Name and Unit Price. It is not necessarily equivalent to a single row in a relational table, because attributes can span multiple related tables. The product case could include all the order detail records for a single product.
Case Set	Collection of related cases. This represents the way the data is viewed and not necessarily the data itself. One case set involving products could focus on the product, whereas another may focus on the purchase detail for the same product.
Clustering	One of two popular algorithms used by Analysis Services to mine data. Clustering involves the classification of data into distinct groups. As opposed to the other algorithm, decision trees, clustering does not require an outcome variable.
Cubes	Multidimensional data structures built from one or more tables in a relational database. Cubes can be the input for a data-mining model, but with Analysis Services the input could also be based on an actual relational table(s).
Decision Trees	One of two popular algorithms used by Analysis Services to mine data. Decision trees involves the creation of a tree that allows the user to map a path to a successful outcome.
Testing Dataset	A portion of historical data that can be used to validate the predictions of a trained mining model. The model will be trained using a training dataset that is representative of all historical data. By using a testing dataset, the developer can ensure that the mining model was designed correctly and can be trusted to make useful predictions.
Training Dataset	A portion of historical data that is representative of all input data. It is important that the training dataset represent input variables in a way that is proportional to occurrences in the entire dataset. In the case of Savings Mart, we would want the training dataset to include all the stores that were open during the same time period so that no bias is unintentionally introduced.

income. Each split on the tree represents a decision that influences the final predicted variable. For example, a customer who graduated from high school may be more likely to pay back the loan. The variable used in the first split is considered the most significant factor. So if educational level is in the first split, it is the factor that most influences credit risk.

Clustering

Clustering is different from decision trees in that it involves grouping data into meaningful clusters with no specific outcome. It goes through a looped process whereby it reevaluates each cluster against all the other clusters looking for patterns in the data. This algorithm is useful when a large database with hundreds of attributes is first evaluated. The clustering process may uncover a relationship between data items that was never suspected. In the case of the bank that wants to determine credit risk, clustering might be used to identify groups of similar customers. It could reveal that certain customer attributes are more meaningful than originally thought. The attributes identified in this process could then be used to build a mining model with decision trees.

OLE DB for Data-Mining Specification

Analysis Services is based on the **OLE DB for Data Mining** (OLE DB for DM) specification. OLE DB for DM, an extension of OLE DB, was developed by the Data Mining Group at Microsoft Research. It includes an Application Programming Interface (API) that exposes data-mining functionality. This allows third-party providers to implement their own data-mining algorithms. These algorithms can then be made available through the Analysis Services Manager application when building new mining models.

TIP: Readers interested in learning more about the OLE DB for Data Mining Specification can download documentation from the Microsoft Web site at http://www.microsoft.com/downloads/details.aspx?FamilyID=01005f92-dba1-4fa4-8ba0-af6a19d30217&displaylang=en.

Savings Mart

Savings Mart is a fictitious discount retailer operating in a single American state. It has been in business since 2001 and hopes to open new stores by achieving greater operational efficiencies. Since its inception, Savings

Mart has relied on a system of adjusting product inventory thresholds to determine when shipments will be made to stores. Every time a product is purchased, the quantity for that product and store is updated. When the quantity dips below the minimum threshold allowed for that product and store, an order is automatically generated and a shipment is made three days later.

Although this process seemed like a good way to ensure that the stores were well stocked, it resulted in shipments being made to each store almost every day. This resulted in high overhead costs for each store. Management now wishes to replace the order/shipment strategy with a system designed to predict shipment dates rather than rely on adjustable thresholds.

A sample application presented in this chapter allows the reader to create a training dataset for Savings Mart based on randomly generated purchases. The reader can then step through the process of creating a mining model based on the dataset.

Loading the Savings Mart Database

To execute the sample code with SQL Server, you will need to create a database using a script file available on the book's Web site. The installation steps are as follows:

1. Open SQL Server's **Query Analyzer** and connect to the server where you wish to install the database.
2. Click **File** and **Open**…
3. From the **Open Query File** dialog, browse to location of the **InstallDB.sql** file available on the book's Web site. Once selected, click **OK**.
4. Click **Query** and **Execute** or hit **F5**.
5. Once the script has completed executing, a new database named SavingsMart will be visible in the drop-down box on the toolbar.

Figure 5.1 is a diagram of the SavingsMart database. The Products table contains a record for every product available in the stores. Each product is associated with a certain vendor and assigned to a product type, such as beverage or medicine. A field named DiscontinuedDate will contain either a null, meaning it is not discontinued, or the date that it should no longer be available for order. Every product should have a UnitQuantity of one or greater, which indicates the number of items packaged with that product. The products UnitPrice represents the retail

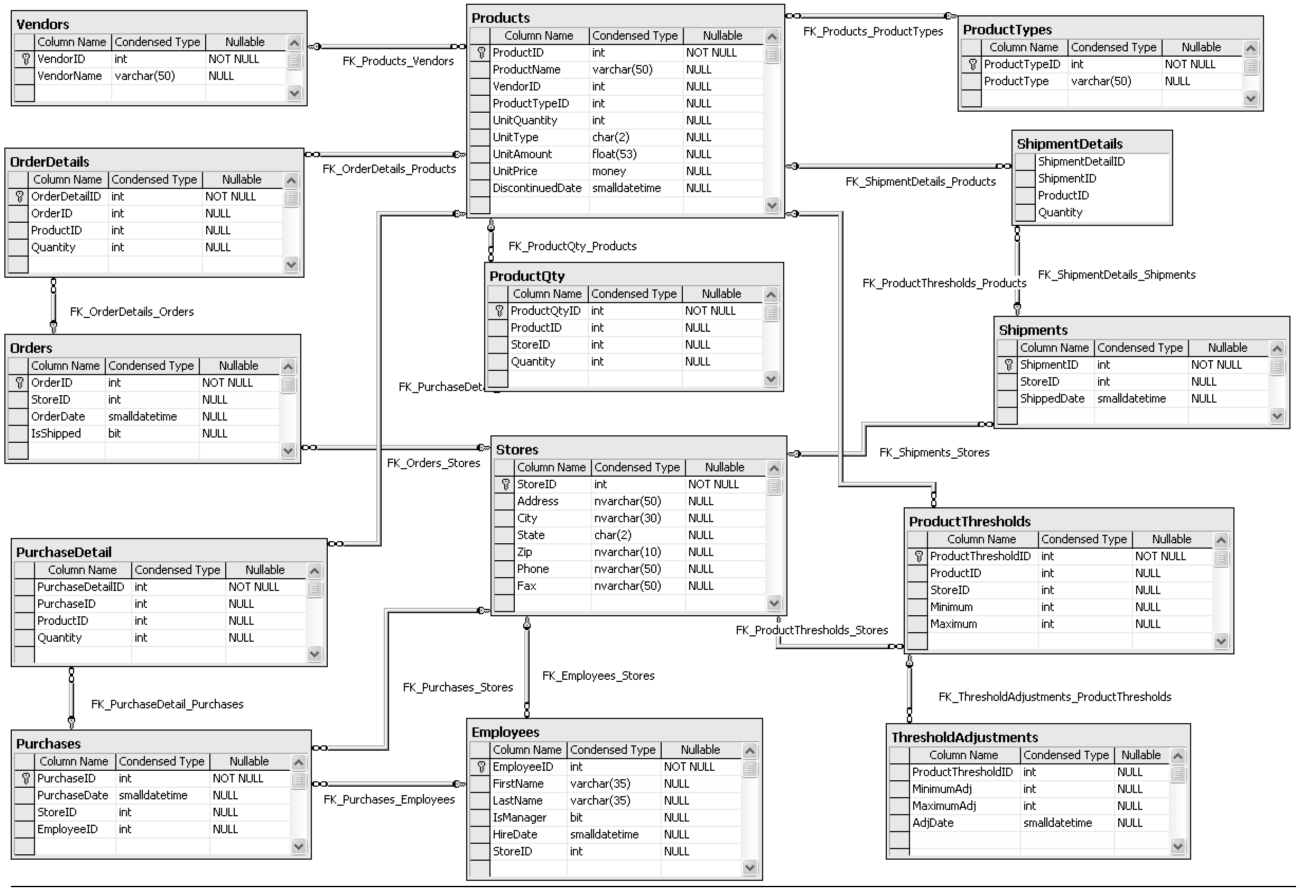

Figure 5.1 The SavingsMart database. The sample database contains five hundred products and five stores. Stores are stocked with all products according to threshold values contained in the ProductThresholds table. Each time a product is ordered and a shipment completed, the quantity field in ProductQty is updated.

price before any discounts are applied. The UnitType and UnitAmount fields may or may not contain values, depending on the product. For instance, a bottled water product will have a UnitType of "oz" and a UnitAmount of 16.4, indicating that it weighs 16.4 fluid ounces. It is not necessary to record the weight of a mop, so for this product these values would be null.

The Purchases table is written to every time a customer makes a single purchase. It records information common to all purchases, such as when the purchase took place, what store it was made in, and what employee rang up the purchase. Purchases are made for products available within a particular store. Availability is determined by examining the Quantity field in the table ProductQty. A purchase can include multiple products and more than one unit of each product. The ProductDetail table is a child of products, and it contains a record for each product associated with a single purchase. If the product purchased is on sale during the time of purchase, then the discount percentage, specified in the ProductDiscounts table is applied.

Once a product is sold beneath the minimum threshold allowed for the store, as indicated by the ProductThresholds table, an order is automatically placed. The quantity for the order is based on the maximum amount found in the ProductThresholds table. Each shipment is the direct result of an order and is typically completed three days after the order is placed.

Populating the Database

Once the SavingsMart database is created, the next step is to populate the database. Unlike the sample databases in Chapters 2 and 3, the SavingsMart database needs to be populated with a large quantity of data. To facilitate this process and provide a method for generating unique training datasets, a sample data-loading application is included on the book's Web site at http:// www.awprofessional.com/title/0321246268. The sample Windows application, named LoadSampleData, will allow you to simulate random purchases as well as to initiate orders and shipments needed to restock products.

Utilizing the LoadSampleData program ensures that you are dealing with a clean database. Very often, the most difficult and time-consuming part of successful data mining involves cleaning the historical database to remove or replace records holding invalid values. Refer to the next section, "Cleaning the Database," for more information about this.

The LoadSampleData program also gives you an opportunity to adjust factors affecting the mining model and therefore produce different results. For instance, the program allows you to select certain product types and vendor names that will be purchased more often.

The LoadSampleData application consists of one form, Form1.vb (see Figure 5.2). It utilizes the Microsoft Data Application block to handle data access and the Microsoft Exception Application block to handle writing exceptions to the event log. The **Load Data** button is used to populate tables with values from text files, available for download from the book's Web site. The following is a list of these text files along with a brief description of what they contain:

- Stores.txt—Data for a total of five stores.
- Employees.txt—Assigns three employees to each store.
- Vendors.txt—Data for a total of thirty-four vendors or product brands.
- ProductTypes.txt—A total of fourteen product types, including such items as Beverages and Kitchen Supplies.
- Products.txt—A total of five hundred products representing each of the product types and vendors.

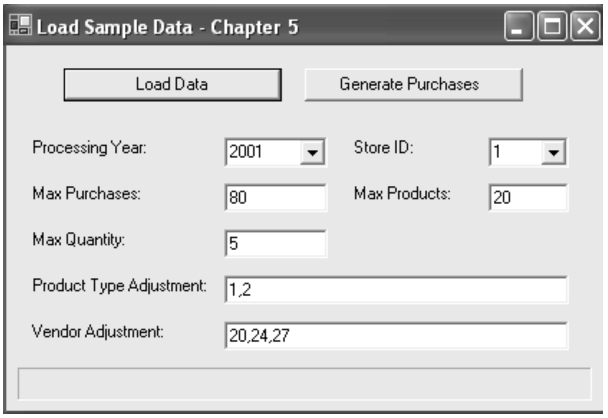

Figure 5.2 Screenshot of the main form used in the LoadSampleData program. This program will be used to load initial data values into the SavingsMart database. It will also allow the reader to simulate random customer purchases in order to populate a large historical dataset.

The LoadData routine is also used to populate the ProductThresholds table with set values for minimum and maximum threshold amounts. The minimum field represents the minimum quantity of product that should be available in a certain store. The maximum field is the quantity that will be ordered for that store when the minimum threshold is broken. Initially, the minimum and maximum values will be set at ten and two hundred respectively. This will be the case for each product and each store, resulting in a total of twenty-five hundred records (500 products × 5 stores = 2500 records).

Finally, the LoadData routine will generate orders and shipments for each of the five stores. The initial orders will stock the stores with the maximum threshold for all five hundred products. The shipment date will occur three days after the order date to ensure that all stores are fully stocked on the first day of the year.

To begin loading data, execute the following steps:

1. Copy the contents of the LoadSampleData directory, available for download from the book's Web site, to a location accessible by your development machine. Note the location because it will be used to set the sPath variable in step 4.
2. Open the LoadSampleData project file with **Visual Studio .NET.**
3. From **Solution Explorer**, right-click Form1 and select **View Code**.
4. The top of the form contains two variables that will be unique to your installation. Ensure that the **sConn** and **sPath** variables are set correctly. sConn is a string variable containing the connection string used to connect to the SavingsMart database on SQL Server . sPath is a string variable containing the file path to the text files. The text files reside in a subdirectory name TextFiles. This subdirectory is located inside the LoadSampleData directory (created in step 1).
5. Execute the application by selecting **Start** from the **Debug** Menu. Figure 5.2 is a screenshot of form1.
6. To begin, click the **Load Data** button and ensure that the message box "Initial Data Load is complete" appears. Note that the form contains several combo and textboxes that will determine what and how data is loaded.

TIP: If you do not wish to load the database utilizing the sample application provided, you can instead attach the database file supplied on the book's Web site. To attach the file, execute the following steps:

1. Copy the **SavingsMart.mdf** and **SavingsMart.ldf** files from the book's Web site to a local directory on the server where Microsoft SQL Server is installed.
2. Open SQL Server's **Enterprise Manager**.
3. Right-click the **Databases** node and click **All Tasks...** and **Attach Database**.
4. From the **Attach Database** dialog, browse to the directory where you copied the database files in step 1 and select the SavingsMart.mdf file. Click **OK**.
5. Click **OK** to attach the database.
6. To see the newly added database, you will need to click **Action** and **Refresh**.

The next step is to generate purchases for each of the five stores. Data mining is most effective in dealing with large datasets. Therefore, the GeneratePurchases routine, seen as follows, will insert approximately 100,000 records in the PurchaseDetail table for each store and calendar year. Purchases are generated for one store—one year at a time.

```
'Maximum # of purchases per day
Dim nMaxPurchases As Int16 = txtMaxPurchases.Text
'Maximum # of products per purchase
Dim nMaxProducts As Int16 = txtMaxProducts.Text
'Maximum value of quantity per product
Dim nMaxQuantity As Int16 = txtMaxQuantity.Text
Dim sYear As String = cboYear.Text
Dim nStoreID As Int16 = cboStoreID.Text

'These are the Product Types in which there is an
'increased chance of product selection
'The default of 1,2 represents snack foods and beverages
Dim sProductTypeAdj As String = txtProductTypeAdj.Text

'These are the Vendors in which there is an increased
'chance of product selection
'The default of 20,24,27 represents Kamp, Notch, and PNuts as Vendors
Dim sVendorAdj As String = txtVendorAdj.Text
```

```
ProgressBar1.Minimum = 1
ProgressBar1.Maximum = 366

Try
  Dim params(2) As SqlParameter
  params(0) = New _
   SqlParameter("@ID", SqlDbType.Int)
  params(1) = New _
   SqlParameter("@ProdTypeAdj", SqlDbType.VarChar, 50)
  params(2) = New _
  SqlParameter("@VendorAdj", SqlDbType.VarChar, 50)
 params(0).Value = nStoreID
 params(1).Value = sProductTypeAdj
 params(2).Value = sVendorAdj
 Dim ds As DataSet = _
 SqlHelper.ExecuteDataset(sConn, _
  CommandType.StoredProcedure, "GetProductIDS", params)
 Dim i As Int16 = 1
 Dim dtDate As DateTime
 dtDate = Convert.ToDateTime("01/01/" + sYear)
 'Loop through everyday of the year
 'We assume the store is open every day
 Randomize()
 Do Until i = 366
   'First thing is check to see if orders needs to
   'be fulfilled for this day and store
   'We assume that all orders are shipped 3 days
   'after the orderdate in one shipment
   Dim params1(1) As SqlParameter
   params1(0) = New _
     SqlParameter("@StoreID", SqlDbType.Int)
   params1(1) = New _
     SqlParameter("@Date", SqlDbType.SmallDateTime)
   params1(0).Value = nStoreID
   'order was placed 3 days ago
   params1(1).Value = dtDate.AddDays(-3)
   SqlHelper.ExecuteReader(sConn, _
    CommandType.StoredProcedure, "InsertShipment", params1)
   Dim x As Int16 = 1
   'This will be the total number of purchases for this day
   Dim nPurchases As Int16
   nPurchases = CInt(Int((nMaxPurchases * Rnd()) + 1))
   Do Until x = nPurchases + 1
```

```
Dim y As Int16 = 1
Dim nEmployeePos As Int16
nEmployeePos = CInt(Int((ds.Tables(1).Rows.Count * Rnd())))
Dim nEmployeeID As Integer = _
   Convert.ToInt32(ds.Tables(1).Rows(nEmployeePos).Item(0))
Dim params2(2) As SqlParameter
params2(0) = New SqlParameter("@ID1", SqlDbType.Int)
params2(1) = New _
  SqlParameter("@Date", SqlDbType.SmallDateTime)
params2(2) = New SqlParameter("@ID2", SqlDbType.Int)
params2(0).Value = nStoreID
params2(1).Value = dtDate
params2(2).Value = nEmployeeID
Dim nPurchaseID As Integer = _
  SqlHelper.ExecuteScalar(sConn, _
 CommandType.StoredProcedure, "InsertPurchase", params2)
'This is total number of products for this purchase
Dim nProducts As Int16 = _
  CInt(Int((nMaxProducts * Rnd()) + 1))
Do Until y = nProducts + 1
  'This is quantity for this purchase
  Dim nQty As Int16 = _
    CInt(Int((nMaxQuantity * Rnd()) + 1))
  'This is the product for this detail record
  Dim nProductPos As Int16 = _
    CInt(Int((ds.Tables(0).Rows.Count * Rnd())))
  Dim nProductID As Integer = _
   Convert.ToInt32(ds.Tables(0).Rows(nProductPos).Item(0))
  'Generate the detail record
  Dim params3(4) As SqlParameter
   params3(0) = New SqlParameter("@StoreID", SqlDbType.Int)
  params3(1) = New _
    SqlParameter("@ProductID", SqlDbType.Int)
  params3(2) = New _
    SqlParameter("@PurchaseID", SqlDbType.Int)
  params3(3) = New SqlParameter("@Qty", SqlDbType.Int)
  params3(4) = New _
    SqlParameter("@Date", SqlDbType.SmallDateTime)
  params3(0).Value = nStoreID
  params3(1).Value = nProductID
  params3(2).Value = nPurchaseID
  params3(3).Value = nQty
  params3(4).Value = dtDate
```

```
        SqlHelper.ExecuteScalar(sConn, _
          CommandType.StoredProcedure, _
         "InsertPurchaseDetail", params3)
        y += 1
      Loop
      x += 1
    Loop

    i += 1
    ProgressBar1.Value = i
    'Go to the next day
    dtDate = dtDate.AddDays(1)
   Loop
   MessageBox.Show("Purchases for store " _
    + Convert.ToString(cboStoreID.Text) + _
    " were generated successfully")
Catch ex As Exception
   MessageBox.Show(ex.Message)
   ExceptionManager.Publish(ex)
End Try
```

The amount of records is approximate because the routine utilizes a random number generator to determine the number of purchases per day along with the number of products per purchase. The number of records also varies depending on what input variables are chosen on form1.

The program utilizes default values specifying that purchases will be generated for Store 1 in the year 2001. The GeneratePurchases routine contains a main loop that will execute 365 times for each day of one calendar year. The variable max purchases defaults to 80 and is used to provide the maximum value for the random number generator when determining how many purchases will be generated for a single day.

The variable max products determines the number of distinct products that will be used for a single purchase. Max quantity is used to determine the quantity used in a single purchase detail record. By utilizing the random number generator and then adjusting these values for each store that is processed, we can simulate random purchase activity. In the section titled "Interpreting the Results," we will examine the results of one mining model. To ensure that your results are consistent with the explanations in this section, use the values in Table 5.2 when loading your database.

Table 5.2 Values to be used in the LoadSampleData application when generating purchases for all five stores.

Store	Field Caption	Value (only use the number values and not the literal values in parentheses)
1	Processing Year	2001
	Max Purchases	80
	Max Products	20
	Max Quantity	5
	Product Type Adjustment	1, 2 (Snack Foods and Beverages)
	Vendor Adjustment	20, 24, 27 (Kamp, Notch, and Pnuts)
2	Processing Year	2001
	Max Purchases	60
	Max Products	12
	Max Quantity	7
	Product Type Adjustment	2, 6 (Beverages and Baking Goods)
	Vendor Adjustment	13, 18 (Gombers and Joe's)
3	Processing Year	2001
	Max Purchases	70
	Max Products	12
	Max Quantity	7
	Product Type Adjustment	2 (Beverages)
	Vendor Adjustment	34 (Store Brand)
4	Processing Year	2001
	Max Purchases	100
	Max Products	5
	Max Quantity	2
	Product Type Adjustment	(leave blank)
	Vendor Adjustment	34, 18 (Store Brand and Joe's)
5	Processing Year	2001
	Max Purchases	50
	Max Products	15
	Max Quantity	8
	Product Type Adjustment	6 (Baking Goods)
	Vendor Adjustment	24, 27, 34 (Notch, Pnuts, and Store Brand)

Of course, since we are using a random number generator, the purchases generated will represent all products equally well over the long run. The Product Type Adjustment and Vendor Adjustment variables are introduced because equal distribution of product purchases is not realistic. These variables contain a comma-delimited list of ProductTypeID and VendorID values. The stored procedure **GetProductIDs** uses these values when returning the dataset of available ProductID's. If a ProductTypeID is specified, then every product that relates to that product type will be included in the list of available product id's more than once. This will increase the chances that the product will be selected for the PurchaseDetail record. The Vendor Adjustment works similarly in that for each VendorID specified, all products assigned to that vendor will appear in the available product list more than once.

If you do not alter the values on form1, the GeneratePurchases routine will take approximately twenty minutes to load data for each store and calendar year. A progress bar is used to indicate the status of the data load because the process is somewhat time-consuming.

TIP: Although it is not necessary to execute the GeneratePurchases routine for each store and all three years, make sure to load at least one store for one calendar year before continuing. This will provide you with enough data to use for processing.

Once the data has loaded, you should see a dialog with the message "Purchases were generated successfully."

If you do not receive the successful message and instead receive an error message, you will first need to resolve the error. Then you will need to delete the database from SQL Server Enterprise Manager and repeat the steps used to load data.

If you continue to receive an error, consider attaching the database per the instructions in the previous tip.

Case Study: ComputerFleet

ComputerFleet, (www.computerfleet.com.au), based in Sydney, Australia, leases technology equipment to companies in a number of different industries and to government agencies. Its wide range of clients includes small, medium, and large organizations.

Since it was necessary to store a large amount of data (twelve years worth), ComputerFleet was using a data warehouse built on top of Mi-

crosoft SQL Server 2000. The company was already using analysis tools like Microsoft Data Analyzer and Microsoft Excel to report on the data.

What ComputerFleet needed was a way to predict when its clients would return leased equipment. Unfortunately, this does not always occur on the agreed lease-end date. Having this information would allow ComputerFleet to better manage its equipment and save money. The company would also be in a better position to value its assets.

ComputerFleet asked a consulting company named Angry Koala (www.angrykoala.com.au) and Microsoft Consulting Services to help it obtain this information. Within a few weeks the consultants had built a data-mining solution with Microsoft Analysis Services.

The data-mining solution uses such attributes as the type of industry, type of equipment, and whether the client is new as input variables to the mining model. The predicted results are stored back in the data warehouse as the predicted asset-arrival time.

ComputerFleet was pleased not only because it was able to make valuable predictions, but because it did not have to purchase any extra proprietary software to do so. Also, it was easy to integrate the data-mining solution with the existing data warehouse in Microsoft SQL Server 2000.

This is a real example of a large company that was able to quickly take advantage of the built-in functionality of Microsoft SQL Server to decrease its operational costs. For more information about ComputerFleet and this Microsoft case study, go to www.microsoft.com/resources/casestudies/casestudy.asp?CaseStudyID=14375.

Cleaning the Database

Cleaning the database is one of the most important tasks in successful data mining. Databases to be mined are often constructed from multiple data sources. These data sources often involve data that is prone to a variety of errors that can destroy any chance you have of making useful predictions. Most everyone has heard or used the phrase "Garbage in, Garbage out." This phrase applies more than ever to data mining.

Possible errors include records with impossible values or values that fall outside the expected range. Also, records that include null values where nulls will negatively impact the decision-making process. Most of these errors could be prevented with database restrictions or application code, but this does not always happen. Since our sample database was artificially created, we can be reasonably sure that these errors do not exist.

The following is a list of the errors that could have occurred if our database existed in the real world:

- Store sale in which the ending data occurs before the starting date.
- A purchase handled by a store employee before their hire date or for a store they were not assigned to.
- An order made for a product that was discontinued before the order date.
- A shipment or order date that is invalid or outside the days of operation for the concerned store.
- A negative quantity in either PurchaseDetail, OrderDetail, ShipmentDetail, or ProductQty.
- A product not associated with a vendor or a purchase with no purchase date and quantity.
- A maximum amount that is greater than the minimum.

The methods used to clean a database can vary. Often, values can be corrected with a few update queries made in **Query Analyzer**. The hardest part is determining what the correct values should be. More than likely, outside help will be needed from people intimately familiar with the data, such as a store manager.

Creating Views

In order to ease the process of building a mining model, a special view will be created. The view, vw_Shipments, combines fields from five different tables and will be used in the next section to create the mining model. The view utilizes the function fn_GetLastShipmentDate to calculate the number of days between shipments. The Transact-SQL code (viewable from the User Defined Function tab in Enterprise Manager) for this function is as follows:

```
CREATE FUNCTION fn_GetLastShipmentDate
 (
 @ShipmentID int,
 @ProductID int
 )
RETURNS datetime
AS
BEGIN
```

```
 DECLARE @ShippedDate smalldatetime, @TShipmentID int, @ret
smalldatetime
 DECLARE cursor1 CURSOR SCROLL
 FOR select shippeddate, s.shipmentid from shipments s
    left join shipmentdetails sd on s.shipmentid = sd.shipmentid
    where s.storeid IN (SELECT StoreID FROM Shipments
       WHERE shipmentid = @Shipmentid)
    AND sd.productid = @ProductID
    ORDER BY shippeddate
 OPEN cursor1
 FETCH NEXT FROM cursor1 INTO @ShippedDate, @TShipmentID

 WHILE @@FETCH_STATUS = 0
 BEGIN
    IF @ShipmentID = @TShipmentID
        BEGIN
       FETCH PRIOR FROM cursor1 INTO @ShippedDate, @TShipmentID
       SET @ret = @ShippedDate
       GOTO Close_Cursor
        END

    FETCH NEXT FROM cursor1 INTO @ShippedDate, @TShipmentID
 END

 Close_Cursor:
 CLOSE cursor1
 DEALLOCATE cursor1

 RETURN(@ret)
END
```

The function accepts @ShipmentID and @ProductID as input variables. It then opens a scrollable cursor (similar to an ADO resultset) based on the following SQL statement:

```
SELECT shippeddate, s.shipmentid FROM shipments s
 LEFT JOIN shipmentdetails sd ON s.shipmentid = sd.shipmentid
 WHERE s.storeid IN (SELECT StoreID FROM Shipments
    WHERE shipmentid = @Shipmentid)
 AND sd.productid = @ProductID
 ORDER BY shippeddate
```

The function loops through the cursor results until it locates the ShipmentID supplied as an input variable. Once located, it moves to the

preceding record and returns that shipment date. This shipment date is used as the first variable for the built-in SQL function DATEDIFF. The resulting variable, DaysSinceLastShipped, will be an important column for the Analyze Shipments mining model.

Working with Mining Models

Building a Mining Model

Now that the database has been created and populated, the next step is to create a mining model. Mining models can be created with the **Mining Model Editor** in **Analysis Manager** or programmatically with Decision Support Objects (DSO). Using DSO to create mining models is useful when you need to programmatically automate the mining-model process. For the most part, you will use Analysis Manager to create mining models.

Analysis Manager (see Figure 5.3) allows you to create and manage databases used by Analysis Services. A database node in Analysis Manager does not represent the physical storage of a large database. Instead, it represents the database that Analysis Services will use to hold the mining-model definitions and the results of processing these mining models. It will, however, reference a physical data source.

Each database in Analysis Manager is associated with one or more data sources. These data sources can be either relational databases or data warehouses. Data sources are created by right-clicking the **Data Sources** node and selecting **New Data Source**. From the **Data Link Properties** dialog, a data provider is selected along with the connection information. Analysis Services supports SQL Server, Access, and Oracle databases as data sources.

Mining models are the blueprint for how the data should be analyzed or processed. Each model represents a case set, or a set of cases (see Table 5.1). The mining-model object stores the information necessary to process the model—for instance, what queries are needed to get the data fields, what data fields are input columns or predictable columns, and what relationship each column has with other columns. Input columns are attributes whose values will be used to generate results for the predictable columns. In some cases, the attribute may serve both as an input column and a predictable column.

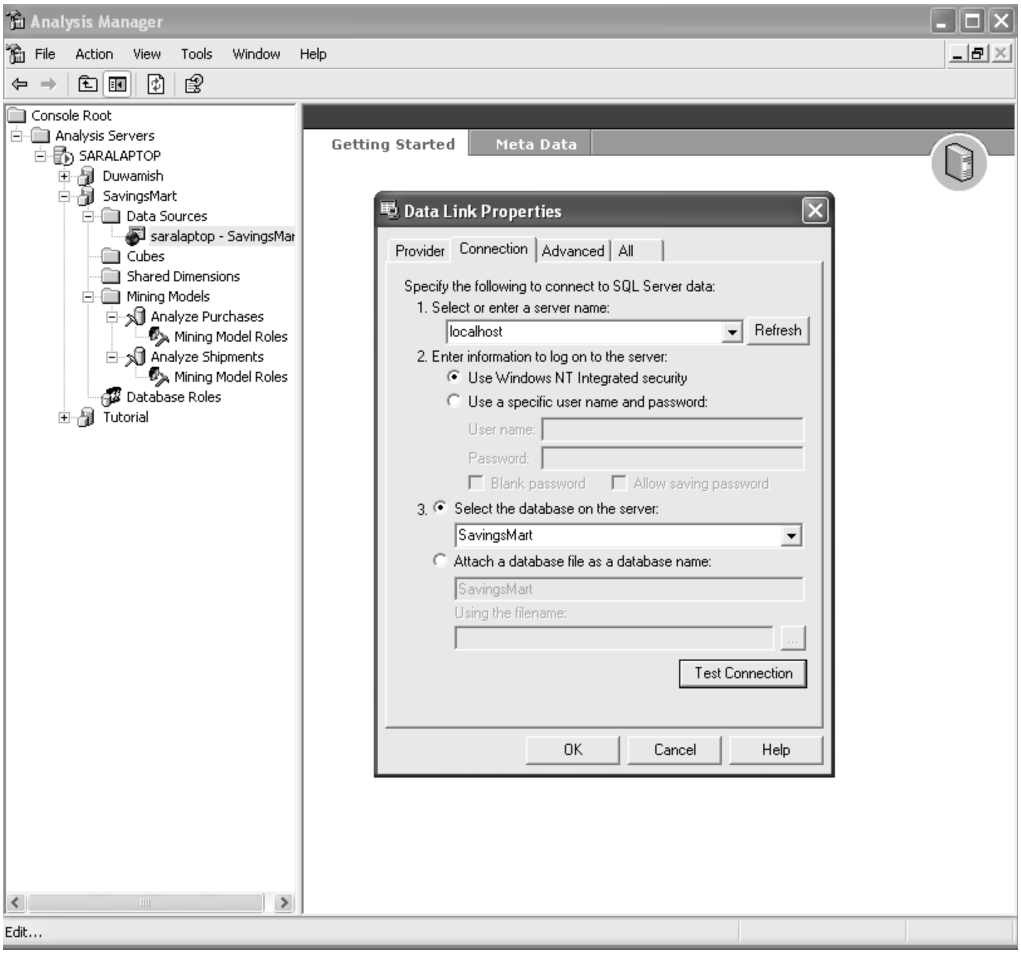

Figure 5.3 Screenshot of Analysis Manager, the utility used to create and manage mining models with Analysis Services. The Data Link Properties dialog box, used to specify the data source, is visible.

Once the model is processed, the data associated with the mining model represents what was learned from the data. The actual data from the training dataset is not stored in the Analysis Services database. The results of analyzing that data, however, are saved.

To quickly demonstrate the process of creating these models, we will walk through the process of creating a mining model using Analysis Manager.

TIP: If you have not already done so, you will need to install Analysis Services. It is available as a separate install with the SQL Server 2000 setup. Make sure that you install the latest Service Pack release for SQL Server. Information about how to obtain the latest SQL Server service pack can be found at http://support.microsoft.com/default.aspx?kbid=290211.

If you have problems connecting with Analysis Services, refer to the article by Alexander Chigrik titled, "Troubleshooting OLAP Problems" at http://databasejournal.com/features/mssql/article.php/1582491. This online article, featured in the Database Journal, is a troubleshooting checklist for solving common problems.

To begin, open **Analysis Manager** from the **Analysis Services** menu item. You will then need to create a new database and specify the data source by executing the following steps:

1. Right-click the server name in the left-hand pane and select **New Database…**
2. Specify 'SavingsMart' as the Database Name and click **OK**
3. Expand the newly added SavingsMart node, right-click Data Sources, and select **New Data Source…**
4. From the Data Link Properties dialog box, select Microsoft OLE DB Provider for SQL Server and click **Next**
5. Enter the SQL connection information for your SQL Server and test the connection before closing the Data Link Properties dialog

The next thing to do is create the mining model using the mining-model wizard. To do so, execute the following steps:

1. Right-click Mining Models in the left-hand pane and select **New Mining Model…**
2. Click **Next** on the Welcome Dialog
3. Click **Next** on the Select Source Dialog because we will be using Relational Data
4. Select the vw_Shipments view from the Available Tables list box in the Select Case Tables dialog and click **Next**. It would have been possible to select multiple tables, but utilizing the view allows access to a calculated field indicating the number of days between shipments.
5. Click **Next** to accept the default of Microsoft Decision Trees as the data-mining technique.
6. Click **Next** to accept ShipmentID as the default Case Key Column.

7. Select the **Finish the mining model in the editor** checkbox and click **Next**.
8. Name the model "Analyze Shipments" and click **Finish**.
9. From the Relational Mining Model Editor, as seen in Figure 5.4, click **Insert** and **Column…** and then select the column named DaysSinceLastShipped. Once added, change the usage to Input and Predictable (note that a diamond icon now appears next to the column). Then go to the **Advanced Properties** and enter DISCRETIZED(CLUSTERS) as the content type.

NOTE: Choosing discretized as the content type allows a continuous variable to be grouped discretely instead. Continuous variables are usually numeric-based values that have an infinite range of possibilities. Since we need predictable results, utilizing a discretization method allows for grouped results. DISCRETEIZED accepts two parameters, such as:

DISCRETIZED(<method>, <#buckets>)

where method could contain one of the following values:

EQUAL_AREAS—Divides into equal buckets
THRESHOLDS—Uses inflection points to estimate bucket boundaries
CLUSTERS—Uses a clustering algorithm to estimate buckets
AUTOMATIC (default)—Tries all algorithms and uses the first one that suggests number buckets

---

10. Click **Insert** and **Column…** and then select the column named StoreID. Once added, change the usage to Input and Predictable.
11. Click **Insert** and **Column…** and then select the column named Quantity. Once added, change the usage to Predictable, and from the Advanced Properties tab, enter DISCRETIZED(CLUSTERS) as the content type.
12. Click **Insert** and **Column…** and then select the column named VendorName.
13. Click **Insert** and **Column…** and then select the column named ProductType.
14. Click **Tools** and **Process Mining Model…** Click **OK** when asked to save the mining model. Then click OK to start a full process of the mining model. This process will take several minutes to run if you loaded data for all five stores. When complete, the message "Processing Complete" will appear in green text.

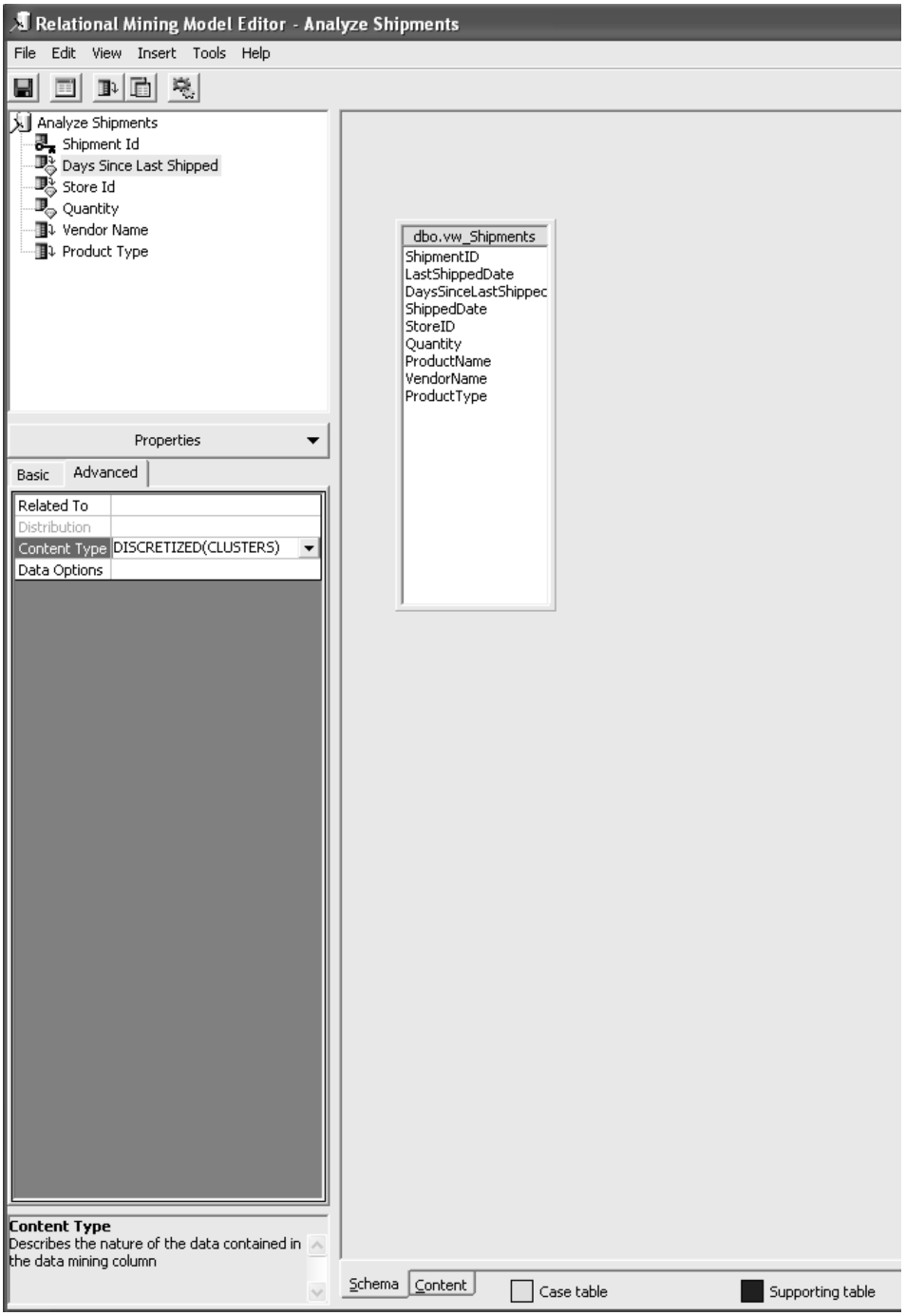

Figure 5.4 Screenshot of the Schema tab in the Relational Mining Model Editor after the columns have been added for the Analyze Shipments mining model.

Training the Mining Model

Training the mining model is accomplished by processing the results of a mining model using Analysis Manager. Alternatively, the same thing could be accomplished using a scripting language known as Data Definition Language (DDL) and a connection to the Analysis Server. We can see what DDL commands are used to train the model through the Process dialog box, as shown in Figure 5.5.

DDL is useful in cases when you want to programmatically process a mining model. The language can be executed through a connection to the Analysis Server. It is also useful for demonstrating how Analysis Manager processes a mining model.

A mining model is created using the CREATE MINING MODEL syntax. The syntax is similar to Transact SQL and should be instantly familiar to SQL developers. The CREATE statement for this mining model is as follows:

```
CREATE MINING MODEL [Analyze Shipments](
    [Shipment Id] LONG KEY,
    [Days Since Last Shipped] LONG DISCRETIZED(CLUSTERS) PREDICT,
    [Store Id] LONG DISCRETE,
    [Quantity] LONG DISCRETIZED(CLUSTERS) PREDICT_ONLY,
```

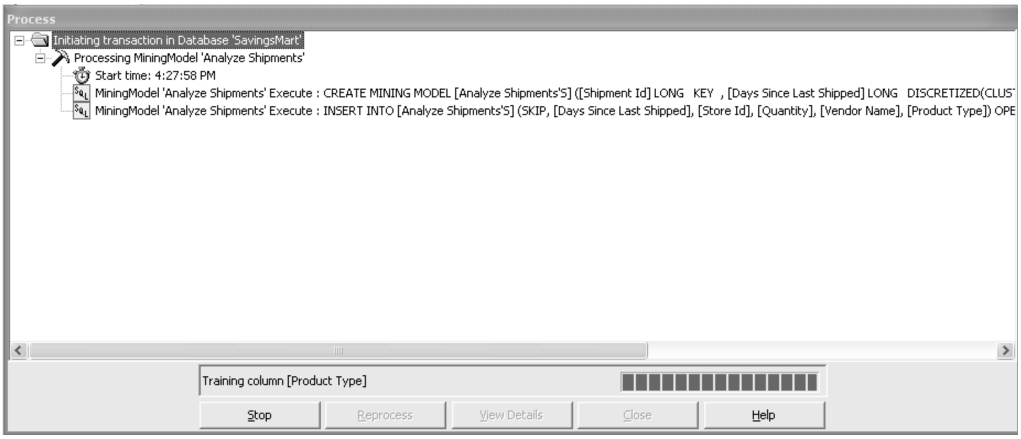

Figure 5.5 Screenshot of the dialog that appears when full process is initiated for a mining model. Note the DDL syntax used to create the model and then train it by populating it with historical data.

```
    [Vendor Name] TEXT DISCRETE,
    [Product Type] TEXT DISCRETE)
USING Microsoft_Decision_Trees
```

With the preceding statement, we are creating a new mining model named Analyze Shipments. The model utilizes Shipment ID as the case key. Days Since Last Shipped, and Quantity are each defined as predictable columns, but Days Since Last Shipped also functions as an input column. The remaining columns, Store ID, Vendor Name, and Product Type, are input columns only. Mining-model columns are defined as either input, predictable, or input and predictable.

The process of training a model involves the insertion of data into the mining model using the INSERT INTO syntax, as follows:

```
INSERT INTO [Analyze Shipments]
   (SKIP,[Days Since Last Shipped], [Store Id], [Quantity],
   [Vendor Name], [Product Type])
OPENROWSET('SQLOLEDB.1','Provider=SQLOLEDB.1;Integrated
   Security=SSPI;Persist Security Info=False;Initial
   Catalog=SavingsMart;Data Source=(local)',
   'SELECT DISTINCT "dbo"."vw_Shipments"."ShipmentID"
   AS "Shipment Id", "dbo"."vw_Shipments"."DaysSinceLastShipped"
   AS "Days Since Last Shipped", "dbo"."vw_Shipments"."StoreID"
   AS "Store Id", "dbo"."vw_Shipments"."Quantity" AS "Quantity",
   "dbo"."vw_Shipments"."VendorName" AS "Vendor Name",
   "dbo"."vw_Shipments"."ProductType" AS "Product Type"
   FROM "dbo"."vw_Shipments"')
```

The mining model will not store the actual data, but will store the prediction results instead once the mining algorithm is processed. In the preceding statement, the OPENROWSET keyword was used to specify the location of the physical data source.

Interpreting the Results

To examine the results from processing the model, select the **Content** tab. Figure 5.6 is a screenshot of the content detail when analyzing DaysSinceLastShipped. This screen indicates that VendorName was the most significant factor affecting DaysSinceLastShipped. We know this because it is the first split on the tree. For nodes that have additional branches, two lines will follow the node. To view the additional branches, double-click that node and the detail page will drill down to the next level.

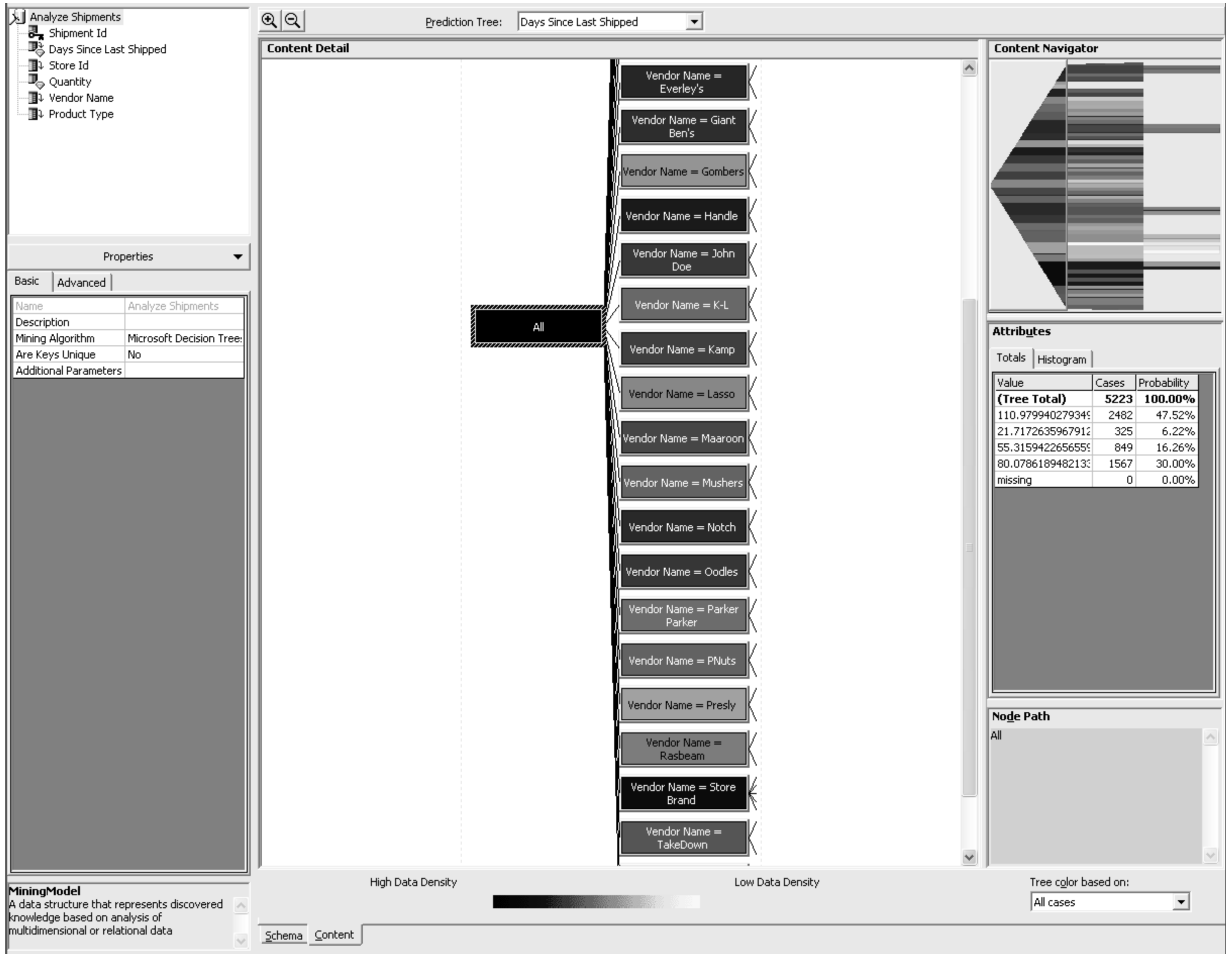

Figure 5.6 Screenshot of the content detail after processing the Analyze Shipments mining model. These results indicate the highest prediction level for the DaysSinceLastShipped column.

The **Content Navigator** box—seen in the top-right corner—offers an easy way to see all the mining-model results and drill down into a certain path. The **Attributes** box shows the totals associated with each node, grouped according to a clustering algorithm. In Figure 5.6, the cursor is selecting the outermost node labeled All. In this example, the attributes are shown for all the cases analyzed.

NOTE: If you did not attach the database file and instead loaded the data using the LoadSampleData program, you will encounter slightly different statistical results. The results presented in this section are specific to the database file available on the book's Web site.

If you attached your database using the file provided, your processing results should be the same as the ones we are about to interpret. The first thing to notice is that the darkest-shaded node is the one where the Vendor Name is Store Brand. Nodes that resulted in a higher data density, or more cases analyzed, will be shaded in a darker color. This result is not surprising, because 127 of the 500 products available, or 25 percent, are represented by the Store Brand. This can be confirmed in **Query Analyzer** with the following query:

```
SELECT v.VendorName,(COUNT(ProductID)/500.0) AS 'Percent'
FROM Products p
LEFT JOIN Vendors v ON p.VendorID = v.VendorID
GROUP BY v.VendorName
ORDER BY 'Percent' DESC
```

If the Store Brand node is double-clicked, the detail pane will show the next branching of the tree (see Figure 5.7). For the Store Brand Node, the first branching distinguishes between the different stores. If we click on the node Store ID = 2 and look at the attributes, the value with the highest probability is 119.33. This indicates that for all products where the Vendor name is Store Brand and the Store ID is 2, it is highly probable that there should be 119 days between shipments.

If we examine the attributes for the remaining nodes, we will see that predictions can be made for all the stores. For Store 1, there is one additional branching that distinguishes between a product type of Snack Foods versus all other product types. When the Store ID is 1, vendor name is Store Brand, and product type is snack foods, there is a 58 percent probability that there will be 60 days between shipments. When we examine the attributes

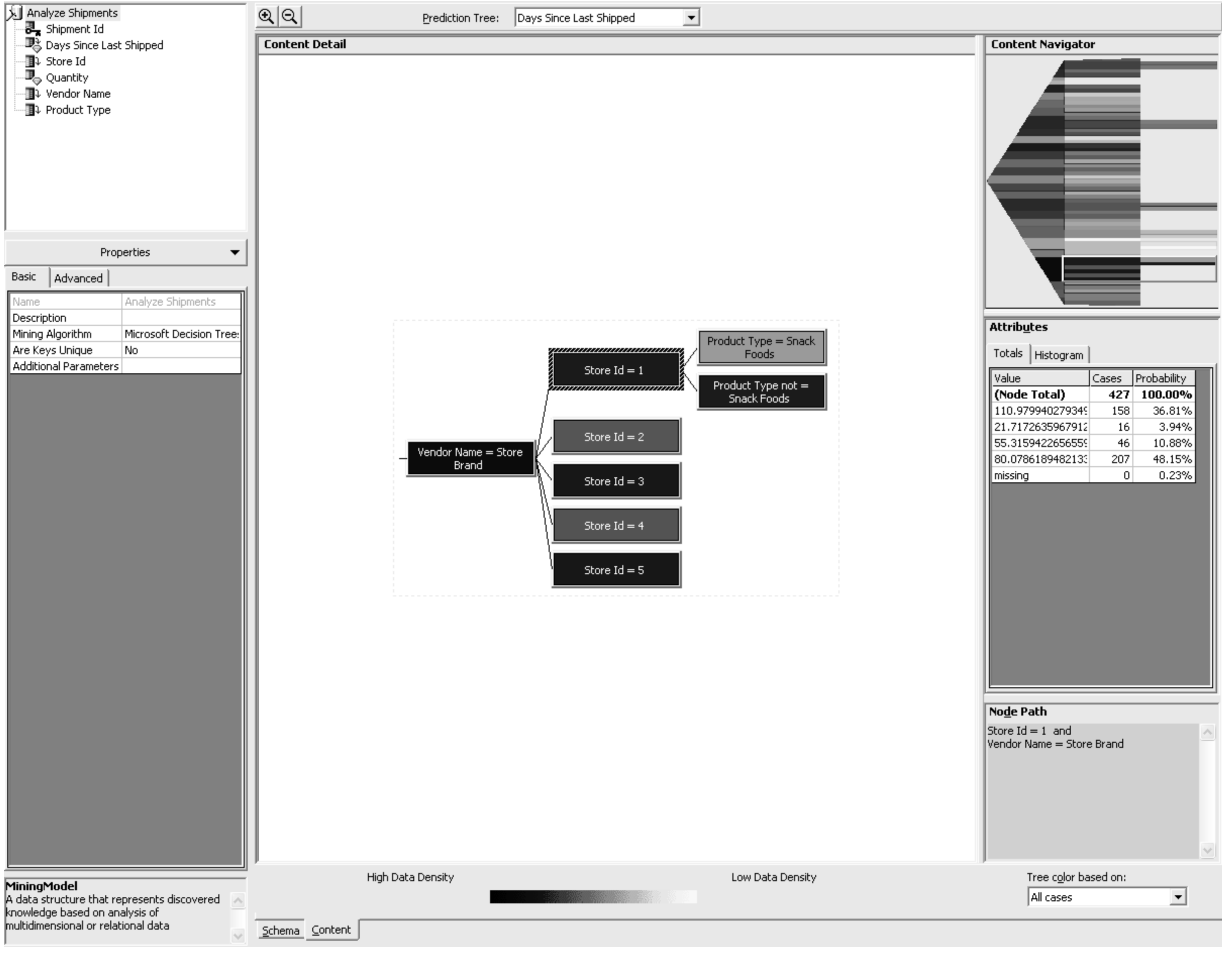

Figure 5.7 Screenshot of the Content Detail Editor as it displays the predictions for days since last shipped. In this example, the node path is where Store ID is 2 and the vendor name is Store Brand.

where product type is not snack foods, there is a 43 percent probability that there will be 119 days between shipments and a 53 percent probability that there will be 85 days between shipments. In this case, we could say that the 53 percent probability wins the toss, but that might not always be the best decision. This will be discussed further in the next chapter.

If you use the drop-down box above the Content Detail pane named **Prediction Tree** to select the quantity column, you will see that the main factor affecting quantity is the days since last shipped (see Figure 5.8).

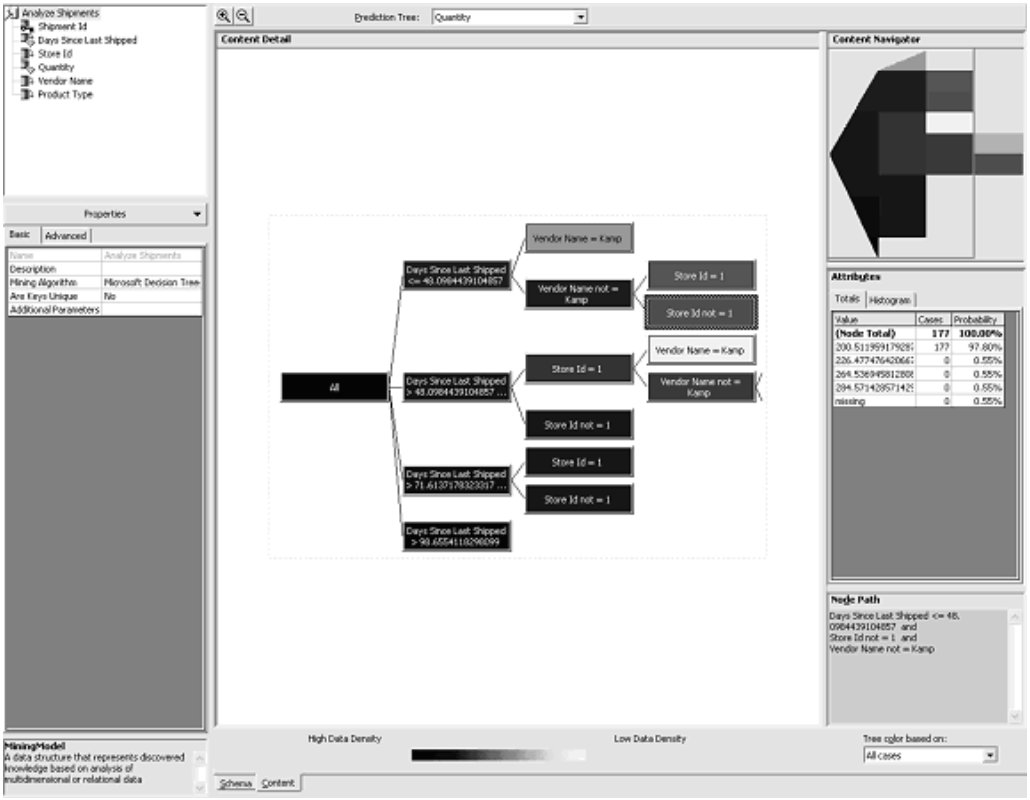

Figure 5.8 Screenshot of the Content Detail Editor as it displays the predictions for Quantity. In this example, the node path examined is where Days Since Last Shipped is less than or equal to 48 and Store ID is not equal to 1 and vendor name is not equal to Kamp.

This is possible because the column DaysSinceLastShipped was defined as an input and a predictable column.

The next factor affecting quantity is the vendor name. In the case where the vendor name is NOT Kamp, Store ID is an additional factor. In Figure 5.8 we can see that when the days since last shipped is less than or equal to 48 and the Store ID is NOT 1 and the vendor name is NOT Kamp, there is a 98 percent probability that the quantity should be 200. When the Store ID is equal to 1, the prediction drops to a 72 percent probability that the quantity will be 200.

The next chapter will involve interpreting the results from the mining model and then applying the predictions to a new shipment strategy. The goal of the new shipment strategy will be to reduce Savings Mart's operational costs by reducing the total number of shipments.

Summary

- The technology known as data warehousing has developed as a means of making use of the huge quantities of data available these days. In addition to storing all this data, data mining is needed to make useful predictions about it. Analysis Services, a separate install for SQL Server 2000, offers data-mining capabilities in an easy-to-use and scalable way.
- This chapter is one of two that examines the effort of a fictional discount retailer named Savings Mart to improve its operational efficiencies. A sample application named LoadSampleData, provided on the book's Web site, allows readers to generate a unique dataset for the data-mining model. Optionally, the reader can also attach a database file provided on the Web site.
- One of the biggest problems affecting successful data mining is invalid or incorrect data. Therefore, the process of cleaning a database is often the most time-consuming aspect of preparing a dataset.
- We step through the process of creating a mining model using Analysis Services. This involves creating a database, naming the data source, and using the mining-model wizard to create the actual model.
- Once a model is created, it can be trained with a training dataset to produce prediction results. The training dataset in this chapter represents one year's worth of purchases and shipments to all five stores. These results will be the basis for a Windows service created in the next chapter.