Precision is defined as the percentage of documents in the results that are relevant. If by entering keyword `bPhone`, the search engine returns 100 documents, and 70 of them are relevant, the precision of the search engine result is 0.7%.

Recall is the percentage of returned documents among all relevant documents in the corpus. If by entering keyword `bPhone`, the search engine returns 100 documents, only 70 of which are relevant while failing to return 10 additional, relevant documents, the recall is $70 / (70 + 10) = 0.875$.

Therefore, the naïve Bayes classifier from Table 9-7 receives a recall of $195 / (195 + 5) = 0.975$ and a precision of $195 / (195 + 101) \approx 0.659$.

Precision and recall are important concepts, whether the task is about information retrieval of a search engine or text analysis over a finite corpus. A good classifier ideally should achieve both precision and recall close to 1.0. In information retrieval, a perfect precision score of 1.0 means that every result retrieved by a search was relevant (but says nothing about whether all relevant documents were retrieved), whereas a perfect recall score of 1.0 means that all relevant documents were retrieved by the search (but says nothing about how many irrelevant documents were also retrieved). Both precision and recall are therefore based on an understanding and measure of relevance. In reality, it is difficult for a classifier to achieve both high precision and high recall. For the example in Table 9-7, the naïve Bayes classifier has a high recall but a low precision. Therefore, the Data Science team needs to check the cleanliness of the data, optimize the classifier, and find if there are ways to improve the precision while retaining the high recall.

Classifiers determine sentiments solely based on the datasets on which they are trained. The domain of the datasets and the characteristics of the features determine what the knowledge classifiers can learn. For example, *lightweight* is a positive feature for reviews on laptops but not necessarily for reviews on wheelbarrows or textbooks. In addition, the training and the testing sets should share similar traits for classifiers to perform well. For example, classifiers trained on movie reviews generally should not be tested on tweets or blog comments.

Note that an absolute sentiment level is not necessarily very informative. Instead, a baseline should be established and then compared against the latest observed values. For example, a ratio of 40% positive tweets on a topic versus 60% negative might not be considered a sign that a product is unsuccessful if other similar successful products have a similar ratio based on the psychology of when people tweet.

The previous example demonstrates how to use naïve Bayes to perform sentiment analysis. The example can be applied to tweets on ACME's `bPhone` and `bEbook` simply by replacing the movie review corpus with the pretagged tweets. Other classifiers can also be used in place of naïve Bayes.

The movie review corpus contains only 2,000 reviews; therefore, it is relatively easy to manually tag each review. For sentiment analysis based on larger amounts of streaming data such as millions or billions of tweets, it is less feasible to collect and construct datasets of tweets that are big enough or manually tag each of the tweets to train and test one or more classifiers. There are two popular ways to cope with this problem. The first way to construct pretagged data, as illustrated in recent work by Go et al. [41] and Pak and Paroubek [42], is to apply supervision and use emoticons such as `:)` and `:(` to indicate if a tweet contains positive or negative sentiments. Words from these tweets can in turn be used as clues to classify the sentiments of future tweets. Go et al. [41] use classification methods including naïve Bayes, MaxEnt, and SVM over the training and testing datasets to perform sentiment classifications. Their demo is available at `http://www.sentiment140.com`. Figure 9-6 shows the sentiments resulting from a query against the term "Boston weather" on a set of tweets. Viewers can mark the result as accurate or inaccurate, and such feedback can be incorporated in future training of the algorithm.

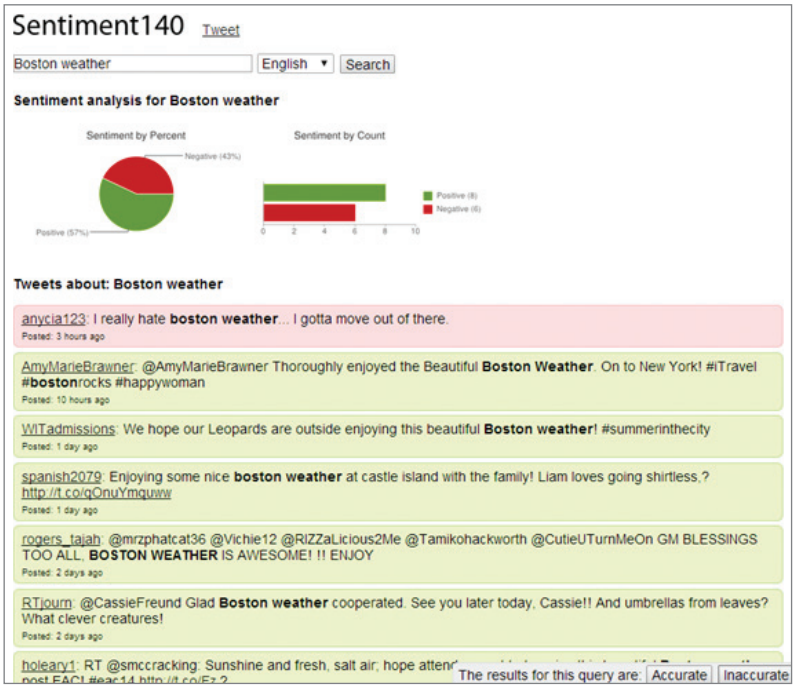

FIGURE 9-6 *Sentiment140 [41], an online tool for Twitter sentiment analysis*

Emoticons make it easy and fast to detect sentiments of millions or billions of tweets. However, using emoticons as the sole indicator of sentiments sometimes can be misleading, as emoticons may not necessarily correspond to the sentiments in the accompanied text. For example, the sample tweet shown in Figure 9-7 contains the :) emoticon, but the text does not express a positive sentiment.

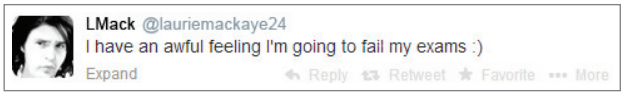

FIGURE 9-7 *Tweet with the :) emoticon does not necessarily correspond to a positive sentiment*

To address this problem, related research usually uses Amazon Mechanical Turk (MTurk) [44] to collect human-tagged reviews. MTurk is a crowdsourcing Internet marketplace that enables individuals or businesses to coordinate the use of human intelligence to perform tasks that are difficult for computers to do. In many cases, MTurk has been shown to collect human input much faster compared to traditional channels such as door-to-door surveys. For the example sentiment analysis task, the Data Science team can publish the tweets collected from Section 9.3 to MTurk as Human Intelligence Tasks (HITs). The team can then ask human workers to tag each tweet as positive, neutral, or negative. The result can be used to train one or more classifiers or test the performances of classifiers. Figure 9-8 shows a sample task on MTurk related to sentiment analysis.

FIGURE 9-8 *Amazon Mechanical Turk*

9.8 Gaining Insights

So far this chapter has discussed several text analysis tasks including text collection, text representation, TFIDF, topic models, and sentiment analysis. This section shows how ACME uses these techniques to gain insights into customer opinions about its products. To keep the example simple, this section only uses `bPhone` to illustrate the steps.

Corresponding to the data collection phase, the Data Science team has used `bPhone` as the keyword to collect more than 300 reviews from a popular technical review website.

The 300 reviews are visualized as a word cloud after removing stop words. A ***word cloud*** (or ***tag cloud***) is a visual representation of textual data. Tags are generally single words, and the importance of each word is shown with font size or color. Figure 9-9 shows the word cloud built from the 300 reviews. The reviews have been previously case folded and tokenized into lowercased words, and stop words have been removed from the text. A more frequently appearing word in Figure 9-9 is shown with a larger font size. The orientation of each word is only for the aesthetical purpose. Most of the graph is taken up by the words `phone` and `bphone`, which occur frequently but are not very informative. Overall, the graph reveals little information. The team needs to conduct further analyses on the data.

FIGURE 9-9 *Word cloud on all 300 reviews on bPhone*

Fortunately, the popular technical review website allows users to provide ratings on a scale from one to five when they post reviews. The team can divide the reviews into subgroups using those ratings.

To reveal more information, the team can remove words such as `phone`, `bPhone`, and `ACME`, which are not very useful for the study. Related research often refers to these words as ***domain-specific stop words***. Figure 9-10 shows the word cloud corresponding to 50 five-star reviews extracted from the data. Note that the shades of gray are only for the aesthetical purpose. The result suggests that customers are satisfied with the `seller`, the `brand`, and the `product`, and they `recommend` bPhone to their friends and families.

Figure 9-11 shows the word cloud of 70 one-star reviews. The words `sim` and `button` occur frequently enough that it would be advisable to sample the reviews that contain these terms and determine what is being said about buttons and SIM cards. Word clouds can reveal useful information beyond the most prominent terms. For example, the graph in Figure 9-11 oddly contains words like `stolen` and `Venezuela`. As the Data Science team investigates the stories behind these words, it finds that these words appear in 1-star reviews because there are a few unauthorized sellers from Venezuela that sell stolen

bPhones. ACME can take further actions from this point. This is an example of how text analysis and even simple visualizations can help gain insights.

Figure 9-10 *Word cloud on five-star reviews*

Figure 9-11 *Word cloud on one-star reviews*

TFIDF can be used to highlight the informative words in the reviews. Figure 9-12 shows a subset of the reviews in which each word with a larger font size corresponds to a higher TFIDF value. Each review is considered a document. With TFIDF, data analysts can quickly go through the reviews and identify what aspects are perceived to make bPhone a good product or a bad product.

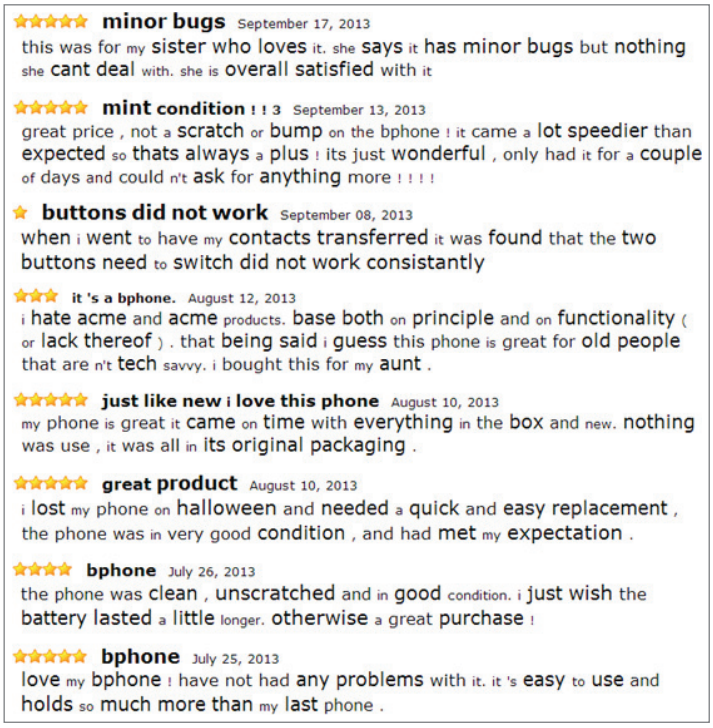

FIGURE 9-12 *Reviews highlighted by TFIDF values*

Topic models such as LDA can categorize the reviews into topics. Figures 9-13 and 9-14 show circular graphs of topics as results of the LDA. These figures are produced with tools and technologies such as Python, NoSQL, and D3.js. Figure 9-13 visualizes ten topics built from the five-star reviews. Each topic focuses on a different aspect that can characterize the reviews. The disc size represents the weight of a word. In an interactive environment, hovering the mouse over a topic displays the full words and their corresponding weights.

Figure 9-14 visualizes ten topics from one-star reviews. For example, the bottom-right topic contains words such as `button`, `power`, and `broken`, which may indicate that bPhone has problems related to button and power supply. The Data Science team can track down these reviews and find out if that's really the case.

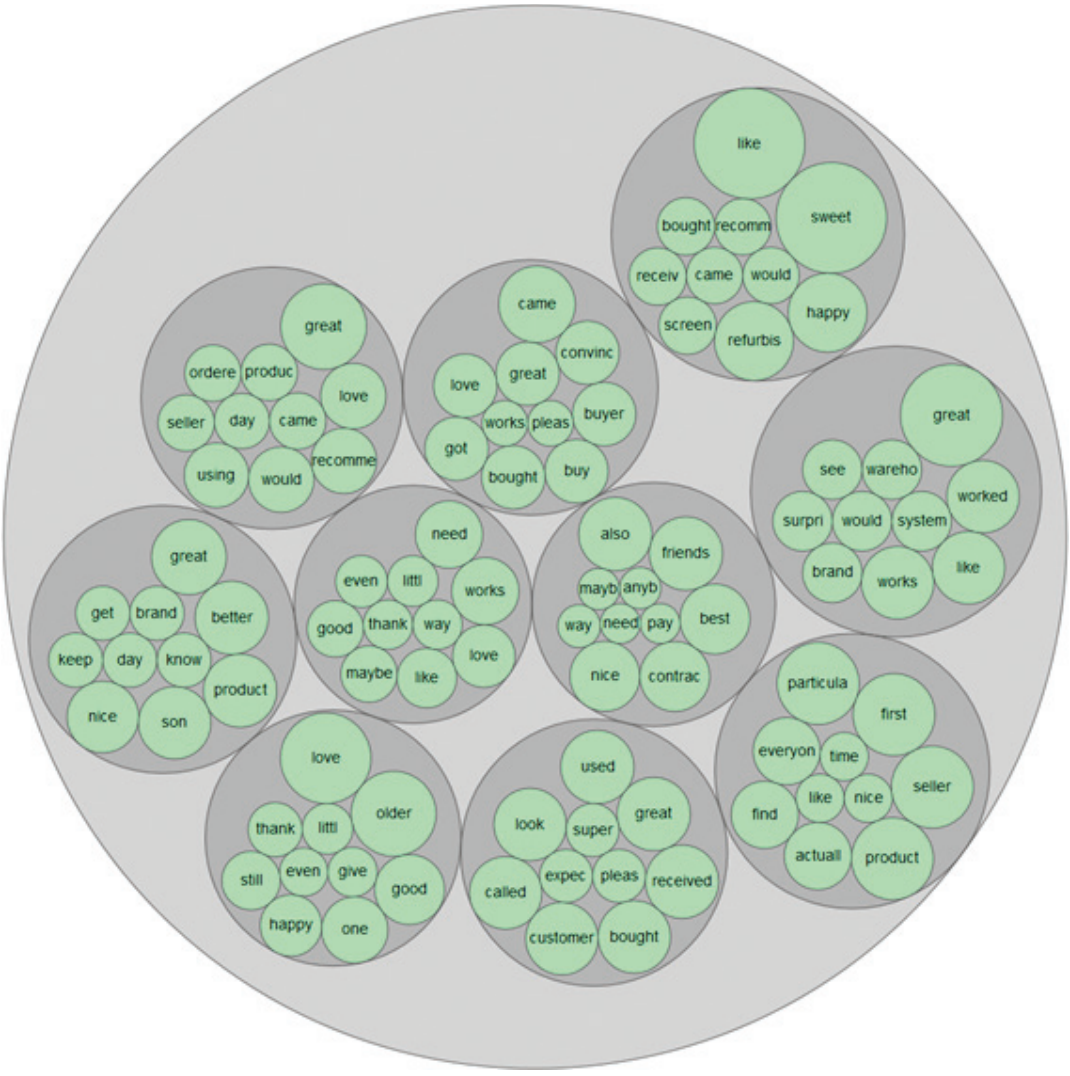

FIGURE 9-13 *Ten topics on five-star reviews*

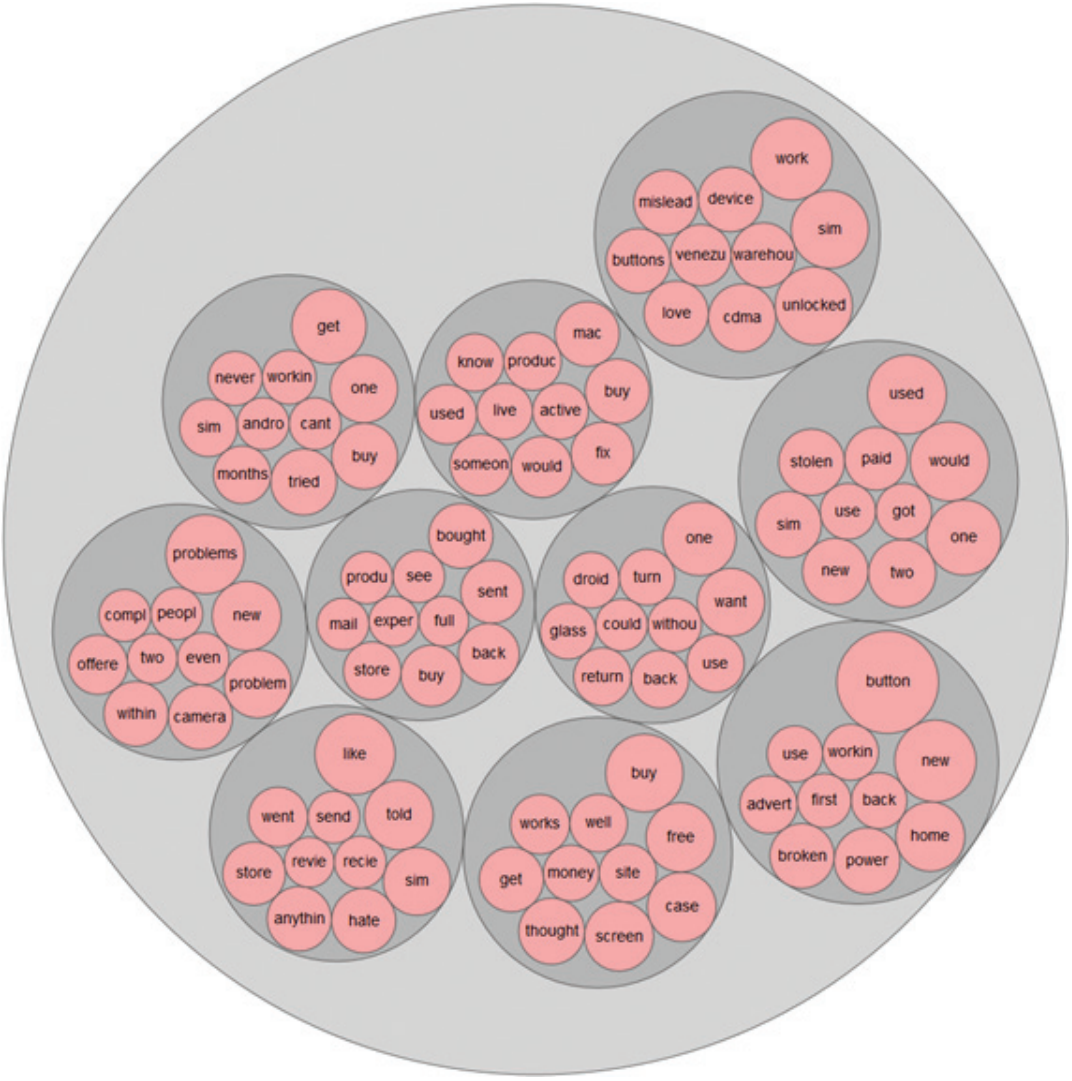

FIGURE 9-14 *Ten topics on one-star reviews*

Figure 9-15 provides a different way to visualize the topics. Five topics are extracted from five-star reviews and one-star reviews, respectively. In an interactive environment, hovering the mouse on a topic highlights the corresponding words in this topic. The screenshots in Figure 9-15 were taken when `Topic 4` is highlighted for both groups. The weight of a word in a topic is indicated by the disc size.

FIGURE 9-15 *Five topics on five-star reviews (left) and 1-star reviews (right)*

The Data Science team has also conducted sentiment analysis over 100 tweets from the popular micro-blogging site Twitter. The result is shown in Figure 9-16. The left side represents negative sentiments, and the right side represents positive sentiments. Vertically, the tweets have been randomly placed for aesthetic purposes. Each tweet is shown as a disc, where the size represents the number of followers of the user who made the original tweet. The color shade of a disc represents how frequently this tweet has been retweeted. The figure indicates that most customers are satisfied with ACME's bPhone.

FIGURE 9-16 *Sentiment analysis on Tweets related to bPhone*

Summary

This chapter has discussed several subtasks of text analysis, including parsing, search and retrieval, and text mining. With a brand management example, the chapter talks about a typical text analysis process: (1) collecting raw text, (2) representing text, (3) using TFIDF to compute the usefulness of each word in the texts, (4) categorizing documents by topics using topic modeling, (5) sentiment analysis, and (6) gaining greater insights.

Overall text analysis is no trivial task. Corresponding to the Data Analytic Lifecycle, the most time-consuming parts of a text analysis project often are not performing the statistics or implementing algorithms. Chances are the team would spend most of the time formulating the problem, getting the data, and preparing the data.

Exercises

1. What are the main challenges of text analysis?
2. What is a corpus?
3. What are common words (such as *a, and, of*) called?
4. Why can't we use TF alone to measure the usefulness of the words?
5. What is a caveat of IDF? How does TFIDF address the problem?
6. Name three benefits of using the TFIDF.
7. What methods can be used for sentiment analysis?
8. What is the definition of *topic* in topic models?
9. Explain the trade-offs for precision and recall.
10. Perform LDA topic modeling on the Reuters-21578 corpus using Python and LDA. The NLTK has already come with the Reuters-21578 corpus. To import this corpus, enter the following comment in the Python prompt:

```
from nltk.corpus import reuters
```

The LDA has already been implemented by several Python libraries such as `gensim` [45]. Either use one such library or implement your own LDA to perform topic modeling on the Reuters-21578 corpus.

11. Choose a topic of your interest, such as a movie, a celebrity, or any buzz word. Then collect 100 tweets related to this topic. Hand-tag them as positive, neutral, or negative. Next, split them into 80 tweets as the training set and the remaining 20 as the testing set. Run one or more classifiers over these tweets to perform sentiment analysis. What are the precision and recall of these classifiers? Which classifier performs better than the others?

Bibliography

[1] Dr. Seuss, "Green Eggs and Ham," New York, NY, USA, Random House, 1960.

[2] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques," ***KDD Workshop on Text Mining,*** 2000.

[3] "The Penn Treebank Project," University of Pennsylvania [Online]. Available: `http://www.cis.upenn.edu/~treebank/home.html`. [Accessed 26 March 2014].

[4] Wikipedia, "List of Open APIs" [Online]. Available: `http://en.wikipedia.org/wiki/List_of_open_APIs`. [Accessed 27 March 2014].

[5] ProgrammableWeb, "API Directory" [Online]. Available: `http://www.programmableweb.com/apis/directory`. [Accessed 27 March 2014].

[6] Twitter, "Twitter Developers Site" [Online]. Available: `https://dev.twitter.com/`. [Accessed 27 March 2014].

[7] "Curl and libcurl Tools" [Online]. Available: `http://curl.haxx.se/`. [Accessed 27 March 2014].

[8] "XML Path Language (XPath) 2.0," World Wide Web Consortium, 14 December 2010. [Online]. Available: `http://www.w3.org/TR/xpath20/`. [Accessed 27 March 2014].

[9] "Gnip: The Source for Social Data," GNIP [Online]. Available: `http://gnip.com/`. [Accessed 12 June 2014].

[10] "DataSift: Power Decisions with Social Data," DataSift [Online]. Available: `http://datasift.com/`. [Accessed 12 June 2014].

[11] G. Salton and C. Buckley, "Term-Weighting Approaches in Automatic Text Retrieval," in *Information Processing and Management*, 1988, pp. 513–523.

[12] G. K. Zipf, *Human Behavior and the Principle of Least Effort*, Reading, MA: Addison-Wesley, 1949.

[13] M. E. Newman, "Power Laws, Pareto Distributions, and Zipf's Law," *Contemporary Physics,* vol. 46, no. 5, pp. 323–351, 2005.

[14] Y. Li, D. McLean, Z. A. Bandar, J. D. O'Shea, and K. Crockett, "Sentence Similarity Based on Semantic Nets and Corpus Statistics," *IEEE Transactions on Knowledge and Data Engineering,* vol. 18, no. 8, pp. 1138–1150, 2006.

[15] W. N. Francis and H. Kucera, "Brown Corpus Manual," 1979. [Online]. Available: `http://icame.uib.no/brown/bcm.html`.

[16] "Critical Assessment of Information Extraction in Biology (BioCreative)" [Online]. Available: `http://www.biocreative.org/`. [Accessed 2 April 2014].

[17] J. J. Godfrey and E. Holliman, "Switchboard-1 Release 2," Linguistic Data Consortium, Philadelphia, 1997. [Online]. Available: `http://catalog.ldc.upenn.edu/LDC97S62`. [Accessed 2 April 2014].

[18] P. Koehn, "Europarl: A Parallel Corpus for Statistical Machine Translation," *MT Summit,* 2005.

[19] N. Seco, T. Veale, and J. Hayes, "An Intrinsic Information Content Metric for Semantic Similarity in WordNet," *ECAI,* vol. 16, pp. 1089–1090, 2004.

[20] P. Resnik, "Using Information Content to Evaluate Semantic Similarity in a Taxonomy," In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95),* vol. 1, pp. 448–453, 1995.

[21] T. Pedersen, "Information Content Measures of Semantic Similarity Perform Better Without Sense-Tagged Text," *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics,* pp. 329–332, June 2010.

[22] C. D. Manning, P. Raghavan, and H. Schütze, "Document and Query Weighting Schemes," in *Introduction to Information Retrieval,* Cambridge, United Kingdom, Cambridge University Press, 2008, p. 128.

[23] M. Porter, "Porter's English Stop Word List," 12 February 2007. [Online]. Available: http://snowball.tartarus.org/algorithms/english/stop.txt. [Accessed 2 April 2014].

[24] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques," *KDD workshop on text mining,* vol. 400, no. 1, 2000.

[25] T. Joachims, "Transductive Inference for Text Classification Using Support Vector Machines," *ICML,* vol. 99, pp. 200–209, 1999.

[26] P. Soucy and G. W. Mineau, "A Simple KNN Algorithm for Text Categorization," *ICDM,* pp. 647–648, 2001.

[27] B. Liu, X. Li, W. S. Lee, and P. S. Yu, "Text Classification by Labeling Words," *AAAI,* vol. 4, pp. 425–430, 2004.

[28] D. M. Blei, "Probabilistic Topic Models," *Communications of the ACM,* vol. 55, no. 4, pp. 77–84, 2012.

[29] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research,* vol. 3, pp. 993–1022, 2003.

[30] T. Minka, "Estimating a Dirichlet Distribution," 2000.

[31] J. Chang, "lda: Collapsed Gibbs Sampling Methods for Topic Models," *CRAN*, 14 October 2012. [Online]. Available: `http://cran.r-project.org/web/packages/lda/`. [Accessed 3 April 2014].

[32] D. M. Blei, "Topic Modeling Software" [Online]. Available: http://www.cs.princeton.edu/~blei/topicmodeling.html. [Accessed 11 June 2014].

[33] A. McCallum, K. Nigam, J. Rennie, and K. Seymore, "A Machine Learning Approach to Building Domain-Specific Search Engines," *IJCAI,* vol. 99, 1999.

[34] P. D. Turney, "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews," *Proceedings of the Association for Computational Linguistics,* pp. 417–424, 2002.

[35] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs Up? Sentiment Classification Using Machine Learning Techniques," *Proceedings of EMNLP,* pp. 79–86, 2002.

[36] M. Hu and B. Liu, "Mining and Summarizing Customer Reviews," *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* pp. 168–177, 2004.

[37] A. Agarwal, F. Biadsy, and K. R. Mckeown, "Contextual Phrase-Level Polarity Analysis Using Lexical Affect Scoring and Syntactic N-Grams," *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics,* pp. 24–32, 2009.

[38] B. O'Connor, R. Balasubramanyan, B. R. Routledge, and N. A. Smith, "From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series," *Proceedings of the Fourth International Conference on Weblogs and Social Media, ICWSM '10,* pp. 122–129, 2010.

[39] A. Agarwal, B. Xie, I. Vovsha, O. Rambow and R. Passonneau, "Sentiment Analysis of Twitter Data," *In Proceedings of the Workshop on Languages in Social Media,* pp. 30–38, 2011.

[40] H. Saif, Y. He, and H. Alani, "Semantic Sentiment Analysis of Twitter," *Proceedings of the 11th International Conference on The Semantic Web (ISWC'12),* pp. 508–524, 2012.

[41] A. Go, R. Bhayani, and L. Huang, "Twitter Sentiment Classification Using Distant Supervision," *CS224N Project Report, Stanford,* pp. 1–12, 2009.

[42] A. Pak and P. Paroubek, "Twitter as a Corpus for Sentiment Analysis and Opinion Mining," *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10),* pp. 19–21, 2010.

[43] B. Pang and L. Lee, "Opinion Mining and Sentiment Analysis," *Foundations and Trends in Information Retrieval,* vol. 2, no. 1–2, pp. 1–135, 2008.

[44] "Amazon Mechanical Turk" [Online]. Available: http://www.mturk.com/. [Accessed 7 April 2014].

[45] R. Řehůřek, "Python Gensim Library" [Online]. Available: http://radimrehurek.com/gensim/. [Accessed 8 April 2014].

10

Advanced Analytics—Technology and Tools: MapReduce and Hadoop

Key Concepts

Hadoop
Hadoop Ecosystem
MapReduce
NoSQL

Chapter 4, "Advanced Analytical Theory and Methods: Clustering," through Chapter 9, "Advanced Analytical Theory and Methods: Text Analysis," covered several useful analytical methods to classify, predict, and examine relationships within the data. This chapter and Chapter 11, "Advanced Analytics—Technology and Tools: In-Database Analytics," address several aspects of collecting, storing, and processing unstructured and structured data, respectively. This chapter presents some key technologies and tools related to the Apache Hadoop software library, "a framework that allows for the distributed processing of large datasets across clusters of computers using simple programming models" [1].

This chapter focuses on how Hadoop stores data in a distributed system and how Hadoop implements a simple programming paradigm known as MapReduce. Although this chapter makes some Java-specific references, the only intended prerequisite knowledge is a basic understanding of programming. Furthermore, the Java-specific details of writing a MapReduce program for Apache Hadoop are beyond the scope of this text. This omission may appear troublesome, but tools in the Hadoop ecosystem, such as Apache Pig and Apache Hive, can often eliminate the need to explicitly code a MapReduce program. Along with other Hadoop-related tools, Pig and Hive are covered in a portion of this chapter dealing with the Hadoop ecosystem.

To illustrate the power of Hadoop in handling unstructured data, the following discussion provides several Hadoop use cases.

10.1 Analytics for Unstructured Data

Prior to conducting data analysis, the required data must be collected and processed to extract the useful information. The degree of initial processing and data preparation depends on the volume of data, as well as how straightforward it is to understand the structure of the data.

Recall the four types of data structures discussed in Chapter 1, "Introduction to Big Data Analytics":

- **Structured:** A specific and consistent format (for example, a data table)
- **Semi-structured:** A self-describing format (for example, an XML file)
- **Quasi-structured:** A somewhat inconsistent format (for example, a hyperlink)
- **Unstructured:** An inconsistent format (for example, text or video)

Structured data, such as relational database management system (RDBMS) tables, is typically the easiest data format to interpret. However, in practice it is still necessary to understand the various values that may appear in a certain column and what these values represent in different situations (based, for example, on the contents of the other columns for the same record). Also, some columns may contain unstructured text or stored objects, such as pictures or videos. Although the tools presented in this chapter focus on unstructured data, these tools can also be utilized for more structured datasets.

10.1.1 Use Cases

The following material provides several use cases for MapReduce. The MapReduce paradigm offers the means to break a large task into smaller tasks, run tasks in parallel, and consolidate the outputs of the individual tasks into the final output. Apache Hadoop includes a software implementation of MapReduce. More details on MapReduce and Hadoop are provided later in this chapter.

IBM Watson

In 2011, IBM's computer system Watson participated in the U.S. television game show *Jeopardy* against two of the best *Jeopardy* champions in the show's history. In the game, the contestants are provided a clue such as "He likes his martinis shaken, not stirred" and the correct response, phrased in the form of a question, would be, "Who is James Bond?" Over the three-day tournament, Watson was able to defeat the two human contestants.

To educate Watson, Hadoop was utilized to process various data sources such as encyclopedias, dictionaries, news wire feeds, literature, and the entire contents of Wikipedia [2]. For each clue provided during the game, Watson had to perform the following tasks in less than three seconds [3]:

- Deconstruct the provided clue into words and phrases
- Establish the grammatical relationship between the words and the phrases
- Create a set of similar terms to use in Watson's search for a response
- Use Hadoop to coordinate the search for a response across terabytes of data
- Determine possible responses and assign their likelihood of being correct
- Actuate the buzzer
- Provide a syntactically correct response in English

Among other applications, Watson is being used in the medical profession to diagnose patients and provide treatment recommendations [4].

LinkedIn

LinkedIn is an online professional network of 250 million users in 200 countries as of early 2014 [5]. LinkedIn provides several free and subscription-based services, such as company information pages, job postings, talent searches, social graphs of one's contacts, personally tailored news feeds, and access to discussion groups, including a Hadoop users group. LinkedIn utilizes Hadoop for the following purposes [6]:

- Process daily production database transaction logs
- Examine the users' activities such as views and clicks
- Feed the extracted data back to the production systems
- Restructure the data to add to an analytical database
- Develop and test analytical models

Yahoo!

As of 2012, Yahoo! has one of the largest publicly announced Hadoop deployments at 42,000 nodes across several clusters utilizing 350 petabytes of raw storage [7]. Yahoo!'s Hadoop applications include the following [8]:

- Search index creation and maintenance
- Web page content optimization

- Web ad placement optimization
- Spam filters
- Ad-hoc analysis and analytic model development

Prior to deploying Hadoop, it took 26 days to process three years' worth of log data. With Hadoop, the processing time was reduced to 20 minutes.

10.1.2 MapReduce

As mentioned earlier, the MapReduce paradigm provides the means to break a large task into smaller tasks, run the tasks in parallel, and consolidate the outputs of the individual tasks into the final output. As its name implies, MapReduce consists of two basic parts—a map step and a reduce step—detailed as follows:

Map:

- Applies an operation to a piece of data
- Provides some intermediate output

Reduce:

- Consolidates the intermediate outputs from the map steps
- Provides the final output

Each step uses key/value pairs, denoted as `<key, value>`, as input and output. It is useful to think of the key/value pairs as a simple ordered pair. However, the pairs can take fairly complex forms. For example, the key could be a filename, and the value could be the entire contents of the file.

The simplest illustration of MapReduce is a word count example in which the task is to simply count the number of times each word appears in a collection of documents. In practice, the objective of such an exercise is to establish a list of words and their frequency for purposes of search or establishing the relative importance of certain words. Chapter 9 provides more details on text analytics. Figure 10-1 illustrates the MapReduce processing for a single input—in this case, a line of text.

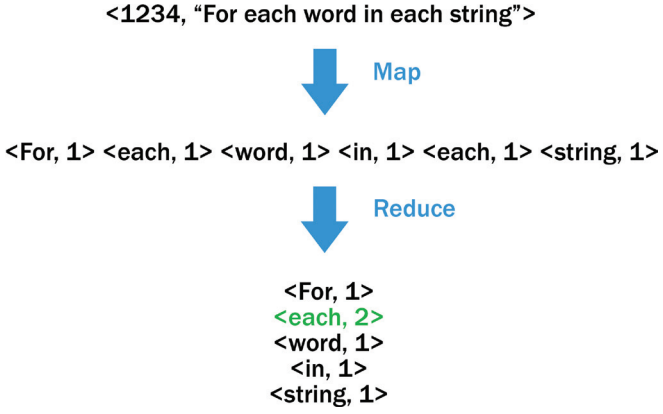

FIGURE 10-1 *Example of how MapReduce works*

In this example, the map step parses the provided text string into individual words and emits a set of key/value pairs of the form `<word, 1>`. For each unique key—in this example, `word`—the reduce step sums the `1` values and outputs the `<word, count>` key/value pairs. Because the word `each` appeared twice in the given line of text, the reduce step provides a corresponding key/value pair of `<each, 2>`.

It should be noted that, in this example, the original key, `1234`, is ignored in the processing. In a typical word count application, the map step may be applied to millions of lines of text, and the reduce step will summarize the key/value pairs generated by all the map steps.

Expanding on the word count example, the final output of a MapReduce process applied to a set of documents might have the `key` as an ordered pair and the `value` as an ordered tuple of length 2n. A possible representation of such a key/value pair follows:

```
<(filename, datetime),(word1,5, word2,7,... , wordn,6)>
```

In this construction, the key is the ordered pair `filename` and `datetime`. The value consists of the n pairs of the words and their individual counts in the corresponding file.

Of course, a word count problem could be addressed in many ways other than MapReduce. However, MapReduce has the advantage of being able to distribute the workload over a cluster of computers and run the tasks in parallel. In a word count, the documents, or even pieces of the documents, could be processed simultaneously during the map step. A key characteristic of MapReduce is that the processing of one portion of the input can be carried out independently of the processing of the other inputs. Thus, the workload can be easily distributed over a cluster of machines.

U.S. Navy rear admiral Grace Hopper (1906–1992), who was a pioneer in the field of computers, provided one of the best explanations of the need for using a group of computers. She commented that during preindustrial times, oxen were used for heavy pulling, but when one ox couldn't budge a log, people didn't try to raise a larger ox; they added more oxen. Her point was that as computational problems grow, instead of building a bigger, more powerful, and more expensive computer, a better alternative is to build a system of computers to share the workload. Thus, in the MapReduce context, a large processing task would be distributed across many computers.

Although the concept of MapReduce has existed for decades, Google led the resurgence in its interest and adoption starting in 2004 with the published work by Dean and Ghemawat [9]. This paper described Google's approach for crawling the web and building Google's search engine. As the paper describes, MapReduce has been used in functional programming languages such as Lisp, which obtained its name from being readily able to process lists (List processing).

In 2007, a well-publicized MapReduce use case was the conversion of 11 million *New York Times* newspaper articles from 1851 to 1980 into PDF files. The intent was to make the PDF files openly available to users on the Internet. After some development and testing of the MapReduce code on a local machine, the 11 million PDF files were generated on a 100-node cluster in about 24 hours [10].

What allowed the development of the MapReduce code and its execution to proceed easily was that the MapReduce paradigm had already been implemented in Apache Hadoop.

10.1.3 Apache Hadoop

Although MapReduce is a simple paradigm to understand, it is not as easy to implement, especially in a distributed system. Executing a MapReduce job (the MapReduce code run against some specified data) requires the management and coordination of several activities:

- MapReduce jobs need to be scheduled based on the system's workload.
- Jobs need to be monitored and managed to ensure that any encountered errors are properly handled so that the job continues to execute if the system partially fails.
- Input data needs to be spread across the cluster.
- Map step processing of the input needs to be conducted across the distributed system, preferably on the same machines where the data resides.
- Intermediate outputs from the numerous map steps need to be collected and provided to the proper machines for the reduce step execution.
- Final output needs to be made available for use by another user, another application, or perhaps another MapReduce job.

Fortunately, Apache Hadoop handles these activities and more. Furthermore, many of these activities are transparent to the developer/user. The following material examines the implementation of MapReduce in Hadoop, an open source project managed and licensed by the Apache Software Foundation [11].

The origins of Hadoop began as a search engine called Nutch, developed by Doug Cutting and Mike Cafarella. Based on two Google papers [9] [12], versions of MapReduce and the Google File System were added to Nutch in 2004. In 2006, Yahoo! hired Cutting, who helped to develop Hadoop based on the code in Nutch [13]. The name "Hadoop" came from the name of Cutting's child's stuffed toy elephant that also inspired the well-recognized symbol for the Hadoop project.

Next, an overview of how data is stored in a Hadoop environment is presented.

Hadoop Distributed File System (HDFS)

Based on the Google File System [12], the Hadoop Distributed File System (HDFS) is a file system that provides the capability to distribute data across a cluster to take advantage of the parallel processing of MapReduce. HDFS is not an alternative to common file systems, such as ext3, ext4, and XFS. In fact, HDFS depends on each disk drive's file system to manage the data being stored to the drive media. The Hadoop Wiki [14] provides more details on disk configuration options and considerations.

For a given file, HDFS breaks the file, say, into 64 MB blocks and stores the blocks across the cluster. So, if a file size is 300 MB, the file is stored in five blocks: four 64 MB blocks and one 44 MB block. If a file size is smaller than 64 MB, the block is assigned the size of the file.

Whenever possible, HDFS attempts to store the blocks for a file on different machines so the map step can operate on each block of a file in parallel. Also, by default, HDFS creates three copies of each block across the cluster to provide the necessary redundancy in case of a failure. If a machine fails, HDFS replicates an accessible copy of the relevant data blocks to another available machine. HDFS is also rack aware, which means that it distributes the blocks across several equipment racks to prevent an entire rack failure from causing a data unavailable event. Additionally, the three copies of each block allow Hadoop some flexibility in determining which machine to use for the map step on a particular block. For example,

an idle or underutilized machine that contains a data block to be processed can be scheduled to process that data block.

To manage the data access, HDFS utilizes three Java daemons (background processes): NameNode, DataNode, and Secondary NameNode. Running on a single machine, the ***NameNode*** daemon determines and tracks where the various blocks of a data file are stored. The ***DataNode*** daemon manages the data stored on each machine. If a client application wants to access a particular file stored in HDFS, the application contacts the NameNode, and the NameNode provides the application with the locations of the various blocks for that file. The application then communicates with the appropriate DataNodes to access the file.

Each DataNode periodically builds a report about the blocks stored on the DataNode and sends the report to the NameNode. If one or more blocks are not accessible on a DataNode, the NameNode ensures that an accessible copy of an inaccessible data block is replicated to another machine. For performance reasons, the NameNode resides in a machine's memory. Because the NameNode is critical to the operation of HDFS, any unavailability or corruption of the NameNode results in a data unavailability event on the cluster. Thus, the NameNode is viewed as a single point of failure in the Hadoop environment [15]. To minimize the chance of a NameNode failure and to improve performance, the NameNode is typically run on a dedicated machine.

A third daemon, the ***Secondary NameNode***, provides the capability to perform some of the NameNode tasks to reduce the load on the NameNode. Such tasks include updating the file system image with the contents of the file system edit logs. It is important to note that the Secondary NameNode is not a backup or redundant NameNode. In the event of a NameNode outage, the NameNode must be restarted and initialized with the last file system image file and the contents of the edits logs. The latest versions of Hadoop provide an HDFS High Availability (HA) feature. This feature enables the use of two NameNodes: one in an active state, and the other in a standby state. If an active NameNode fails, the standby NameNode takes over. When using the HDFS HA feature, a Secondary NameNode is unnecessary [16].

Figure 10-2 illustrates a Hadoop cluster with ten machines and the storage of one large file requiring three HDFS data blocks. Furthermore, this file is stored using triple replication. The machines running the NameNode and the Secondary NameNode are considered ***master nodes***. Because the DataNodes take their instructions from the master nodes, the machines running the DataNodes are referred to as ***worker nodes***.

Structuring a MapReduce Job in Hadoop

Hadoop provides the ability to run MapReduce jobs as described, at a high level, in Section 10.1.2. This section offers specific details on how a MapReduce job is run in Hadoop. A typical MapReduce program in Java consists of three classes: the driver, the mapper, and the reducer.

The ***driver*** provides details such as input file locations, the provisions for adding the input file to the map task, the names of the mapper and reducer Java classes, and the location of the reduce task output. Various job configuration options can also be specified in the driver. For example, the number of reducers can be manually specified in the driver. Such options are useful depending on how the MapReduce job output will be used in later downstream processing.

The ***mapper*** provides the logic to be processed on each data block corresponding to the specified input files in the driver code. For example, in the word count MapReduce example provided earlier, a map task is instantiated on a worker node where a data block resides. Each map task processes a fragment of the text, line by line, parses a line into words, and emits `<word, 1>` for each word, regardless of how many

times `word` appears in the line of text. The key/value pairs are stored temporarily in the worker node's memory (or cached to the node's disk).

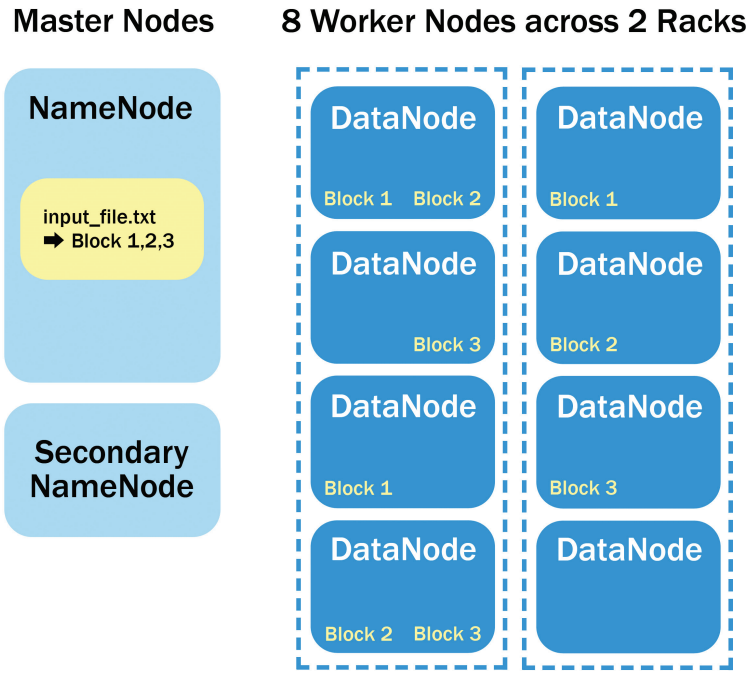

FIGURE 10-2 *A file stored in HDFS*

Next, the key/value pairs are processed by the built-in ***shuffle and sort*** functionality based on the number of reducers to be executed. In this simple example, there is only one reducer. So, all the intermediate data is passed to it. From the various map task outputs, for each unique key, arrays (lists in Java) of the associated values in the key/value pairs are constructed. Also, Hadoop ensures that the keys are passed to each reducer in sorted order. In Figure 10-3, `<each,(1,1)>` is the first key/value pair processed, followed alphabetically by `<For,(1)>` and the rest of the key/value pairs until the last key/value pair is passed to the reducer. The `( )` denotes a list of values which, in this case, is just an array of ones.

In general, each reducer processes the values for each key and emits a key/value pair as defined by the reduce logic. The output is then stored in HDFS like any other file in, say, 64 MB blocks replicated three times across the nodes.

Additional Considerations in Structuring a MapReduce Job

The preceding discussion presented the basics of structuring and running a MapReduce job on a Hadoop cluster. Several Hadoop features provide additional functionality to a MapReduce job.

First, a ***combiner*** is a useful option to apply, when possible, between the map task and the shuffle and sort. Typically, the combiner applies the same logic used in the reducer, but it also applies this logic on the output of each map task. In the word count example, a combiner sums up the number of occurrences of

each word from a mapper's output. Figure 10-4 illustrates how a combiner processes a single string in the simple word count example.

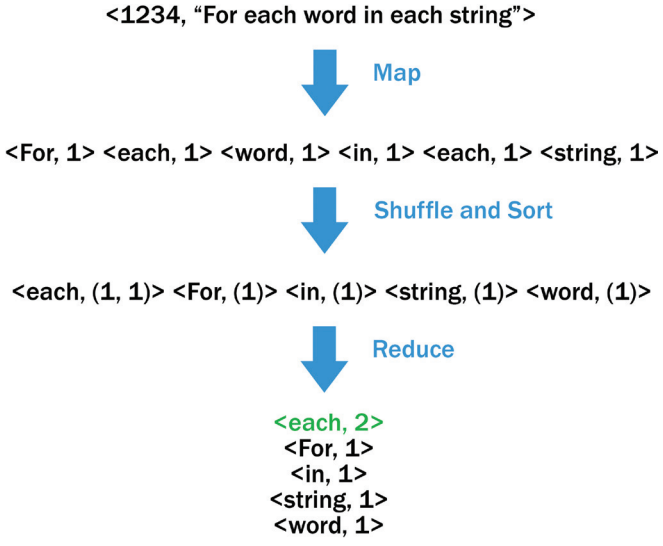

FIGURE 10-3 *Shuffle and sort*

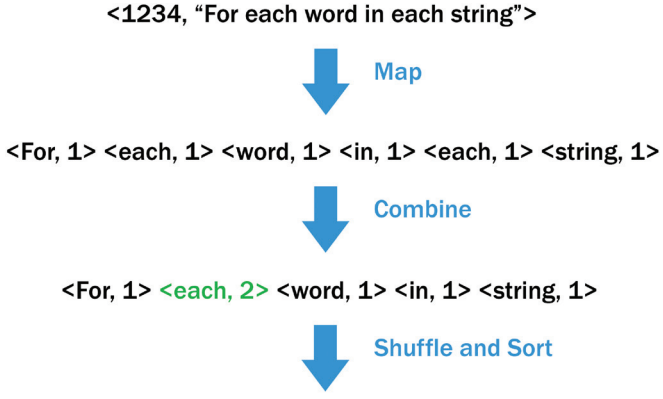

FIGURE 10-4 *Using a combiner*

Thus, in a production setting, instead of ten thousand possible `<the, 1>` key/value pairs being emitted from the map task to the Shuffle and Sort, the combiner emits one `<the, 10000>` key/value pair. The reduce step still obtains a list of values for each word, but instead of receiving a list of up to a million ones `list(1,1,. . .,1)` for a key, the reduce step obtains a list, such as `list(10000,964,. . .,8345)`, which might be as long as the number of map tasks that were run. The use of a combiner minimizes the amount of intermediate map output that the reducer must store, transfer over the network, and process.

Another useful option is the ***partitioner***. It determines the reducers that receive keys and the corresponding list of values. Using the simple word count example, Figure 10-5 shows that a partitioner can send every word that begins with a vowel to one reducer and the other words that begin with a consonant to another reducer.

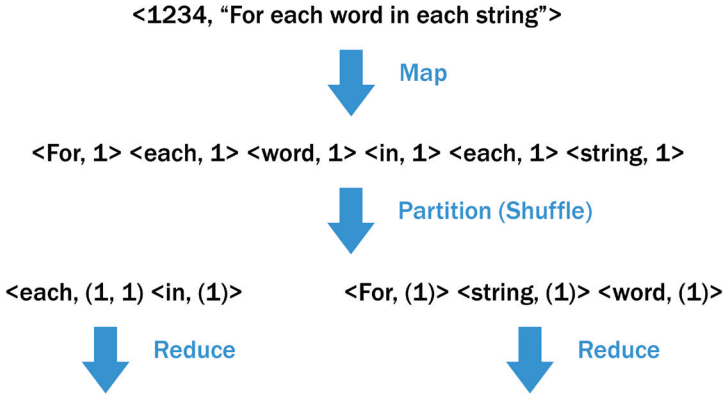

FIGURE 10-5 *Using a custom partitioner*

As a more practical example, a user could use a partitioner to separate the output into separate files for each calendar year for subsequent analysis. Also, a partitioner could be used to ensure that the workload is evenly distributed across the reducers. For example, if a few keys are known to be associated with a large majority of the data, it may be useful to ensure that these keys go to separate reducers to achieve better overall performance. Otherwise, one reducer might be assigned the majority of the data, and the MapReduce job will not complete until that one long-running reduce task completes.

Developing and Executing a Hadoop MapReduce Program

A common approach to develop a Hadoop MapReduce program is to write Java code using an Interactive Development Environment (IDE) tool such as Eclipse [17]. Compared to a plaintext editor or a command-line interface (CLI), IDE tools offer a better experience to write, compile, test, and debug code. A typical MapReduce program consists of three Java files: one each for the driver code, map code, and reduce code. Additional, Java files can be written for the combiner or the custom partitioner, if applicable. The Java code is compiled and stored as a Java Archive (JAR) file. This JAR file is then executed against the specified HDFS input files.

Beyond learning the mechanics of submitting a MapReduce job, three key challenges to a new Hadoop developer are defining the logic of the code to use the MapReduce paradigm; learning the Apache Hadoop Java classes, methods, and interfaces; and implementing the driver, map, and reduce functionality in Java. Some prior experience with Java makes it easier for a new Hadoop developer to focus on learning Hadoop and writing the MapReduce job.

For users who prefer to use a programming language other than Java, there are some other options. One option is to use the ***Hadoop Streaming API***, which allows the user to write and run Hadoop jobs with no direct knowledge of Java [18]. However, knowledge of some other programming language, such as Python, C, or Ruby, is necessary. Apache Hadoop provides the `Hadoop-streaming.jar` file that

accepts the HDFS paths for the input/output files and the paths for the files that implement the map and reduce functionality.

Here are some important considerations when preparing and running a Hadoop streaming job:

- Although the shuffle and sort output are provided to the reducer in key sorted order, the reducer does not receive the corresponding values as a list; rather, it receives individual key/value pairs. The reduce code has to monitor for changes in the value of the key and appropriately handle the new key.
- The map and reduce code must already be in an executable form, or the necessary interpreter must already be installed on each worker node.
- The map and reduce code must already reside on each worker node, or the location of the code must be provided when the job is submitted. In the latter case, the code is copied to each worker node.
- Some functionality, such as a partitioner, still needs to be written in Java.
- The inputs and outputs are handled through stdin and stdout. Stderr is also available to track the status of the tasks, implement counter functionality, and report execution issues to the display [18].
- The streaming API may not perform as well as similar functionality written in Java.

A second alternative is to use ***Hadoop pipes***, a mechanism that uses compiled C++ code for the map and reduced functionality. An advantage of using C++ is the extensive numerical libraries available to include in the code [19].

To work directly with data in HDFS, one option is to use the C API (libhdfs) or the Java API provided with Apache Hadoop. These APIs allow reads and writes to HDFS data files outside the typical MapReduce paradigm [20]. Such an approach may be useful when attempting to debug a MapReduce job by examining the input data or when the objective is to transform the HDFS data prior to running a MapReduce job.

Yet Another Resource Negotiator (YARN)

Apache Hadoop continues to undergo further development and frequent updates. An important change was to separate the MapReduce functionality from the functionality that manages the running of the jobs and the associated responsibilities in a distributed environment. This rewrite is sometimes called MapReduce 2.0, or Yet Another Resource Negotiator (YARN). YARN separates the resource management of the cluster from the scheduling and monitoring of jobs running on the cluster. The YARN implementation makes it possible for paradigms other than MapReduce to be utilized in Hadoop environments. For example, a Bulk Synchronous Parallel (BSP) [21] model may be more appropriate for graph processing than MapReduce [22] is. Apache Hama, which implements the BSP model, is one of several applications being modified to utilize the power of YARN [23].

YARN replaces the functionality previously provided by the JobTracker and TaskTracker daemons. In earlier releases of Hadoop, a MapReduce job is submitted to the JobTracker daemon. The JobTracker communicates with the NameNode to determine which worker nodes store the required data blocks for the MapReduce job. The JobTracker then assigns individual map and reduce tasks to the TaskTracker running on worker nodes. To optimize performance, each task is preferably assigned to a worker node that is storing an input data block. The TaskTracker periodically communicates with the JobTracker on the status of its executing tasks. If a task appears to have failed, the JobTracker can assign the task to a different TaskTracker.

10.2 The Hadoop Ecosystem

So far, this chapter has provided an overview of Apache Hadoop relative to its implementation of HDFS and the MapReduce paradigm. Hadoop's popularity has spawned proprietary and open source tools to make Apache Hadoop easier to use and provide additional functionality and features. This portion of the chapter examines the following Hadoop-related Apache projects:

- **Pig:** Provides a high-level data-flow programming language
- **Hive:** Provides SQL-like access
- **Mahout:** Provides analytical tools
- **HBase:** Provides real-time reads and writes

By masking the details necessary to develop a MapReduce program, Pig and Hive each enable a developer to write high-level code that is later translated into one or more MapReduce programs. Because MapReduce is intended for batch processing, Pig and Hive are also intended for batch processing use cases.

Once Hadoop processes a dataset, Mahout provides several tools that can analyze the data in a Hadoop environment. For example, a k-means clustering analysis, as described in Chapter 4, can be conducted using Mahout.

Differentiating itself from Pig and Hive batch processing, HBase provides the ability to perform real-time reads and writes of data stored in a Hadoop environment. This real-time access is accomplished partly by storing data in memory as well as in HDFS. Also, HBase does not rely on MapReduce to access the HBase data. Because the design and operation of HBase are significantly different from relational databases and the other Hadoop tools examined, a detailed description of HBase will be presented.

10.2.1 Pig

Apache Pig consists of a data flow language, Pig Latin, and an environment to execute the Pig code. The main benefit of using Pig is to utilize the power of MapReduce in a distributed system, while simplifying the tasks of developing and executing a MapReduce job. In most cases, it is transparent to the user that a MapReduce job is running in the background when Pig commands are executed. This abstraction layer on top of Hadoop simplifies the development of code against data in HDFS and makes MapReduce more accessible to a larger audience.

Like Hadoop, Pig's origin began at Yahoo! in 2006. Pig was transferred to the Apache Software Foundation in 2007 and had its first release as an Apache Hadoop subproject in 2008. As Pig evolves over time, three main characteristics persist: ease of programming, behind-the-scenes code optimization, and extensibility of capabilities [24].

With Apache Hadoop and Pig already installed, the basics of using Pig include entering the Pig execution environment by typing **`pig`** at the command prompt and then entering a sequence of Pig instruction lines at the `grunt` prompt.

An example of Pig-specific commands is shown here:

```
$ pig
grunt> records  =  LOAD '/user/customer.txt' AS
                        (cust_id:INT, first_name:CHARARRAY,
```

```
                          last_name:CHARARRAY,
                          email_address:CHARARRAY);
grunt> filtered_records = FILTER records
                          BY email_address matches '.*@isp.com';
grunt> STORE filtered_records INTO '/user/isp_customers';
grunt> quit
$
```

At the first `grunt` prompt, a text file is designated by the Pig variable `records` with four defined fields: `cust_id`, `first_name`, `last_name`, and `email_address`. Next, the variable `filtered_records` is assigned those records where the `email_address` ends with `@isp.com` to extract the customers whose e-mail address is from a particular Internet service provider (ISP). Using the `STORE` command, the filtered records are written to an HDFS folder, `isp_customers`. Finally, to exit the interactive Pig environment, execute the `QUIT` command. Alternatively, these individual Pig commands could be written to the file `filter_script.pig` and submit them at the command prompt as follows:

```
$ pig   filter_script.pig
```

Such Pig instructions are translated, behind the scenes, into one or more MapReduce jobs. Thus, Pig simplifies the coding of a MapReduce job and enables the user to quickly develop, test, and debug the Pig code. In this particular example, the MapReduce job would be initiated after the `STORE` command is processed. Prior to the `STORE` command, Pig had begun to build an execution plan but had not yet initiated MapReduce processing.

Pig provides for the execution of several common data manipulations, such as inner and outer joins between two or more files (tables), as would be expected in a typical relational database. Writing these joins explicitly in MapReduce using Hadoop would be quite involved and complex. Pig also provides a `GROUP BY` functionality that is similar to the `Group By` functionality offered in SQL. Chapter 11 has more details on using `Group By` and other SQL statements.

An additional feature of Pig is that it provides many built-in functions that are easily utilized in Pig code. Table 10-1 includes several useful functions by category.

TABLE 10-1 *Built-In Pig Functions*

Eval	Load/Store	Math	String	DateTime
AVG	BinStorage()	ABS	INDEXOF	AddDuration
CONCAT	JsonLoader	CEIL	LAST_ INDEX_OF	CurrentTime
COUNT	JsonStorage	COS, ACOS	LCFORST	DaysBetween
COUNT_STAR	PigDump	EXP	LOWER	GetDay
DIFF	PigStorage	FLOOR	REGEX_ EXTRACT	GetHour

(*continues*)

Table 10-1 *Built-In Pig Functions* (*Continued*)

Eval	Load/Store	Math	String	DateTime
IsEmpty	TextLoader	LOG, LOG10	REPLACE	GetMinute
MAX	HBaseStorage	RANDOM	STRSPLIT	GetMonth
MIN		ROUND	SUBSTRING	GetWeek
SIZE		SIN, ASIN	TRIM	GetWeekYear
SUM		SQRT	UCFIRST	GetYear
TOKENIZE		TAN, ATAN	UPPER	MinutesBetween
				SubtractDuration
				ToDate

Other functions and the details of these built-in functions can be found at the `pig.apache.org` website [25].

In terms of extensibility, Pig allows the execution of user-defined functions (UDFs) in its environment. Thus, some complex operations can be coded in the user's language of choice and executed in the Pig environment. Users can share their UDFs in a repository called the Piggybank hosted on the Apache site [26]. Over time, the most useful UDFs may be included as built-in functions in Pig.

10.2.2 Hive

Similar to Pig, Apache Hive enables users to process data without explicitly writing MapReduce code. One key difference to Pig is that the Hive language, HiveQL (Hive Query Language), resembles Structured Query Language (SQL) rather than a scripting language.

A Hive table structure consists of rows and columns. The rows typically correspond to some record, transaction, or particular entity (for example, customer) detail. The values of the corresponding columns represent the various attributes or characteristics for each row. Hadoop and its ecosystem are used to apply some structure to unstructured data. Therefore, if a table structure is an appropriate way to view the restructured data, Hive may be a good tool to use.

Additionally, a user may consider using Hive if the user has experience with SQL and the data is already in HDFS. Another consideration in using Hive may be how data will be updated or added to the Hive tables. If data will simply be added to a table periodically, Hive works well, but if there is a need to update data in place, it may be beneficial to consider another tool, such as HBase, which will be discussed in the next section.

Although Hive's performance may be better in certain applications than a conventional SQL database, Hive is not intended for real-time querying. A Hive query is first translated into a MapReduce job, which is then submitted to the Hadoop cluster. Thus, the execution of the query has to compete for resources with any other submitted job. Like Pig, Hive is intended for batch processing. Again, HBase may be a better choice for real-time query needs.

To summarize the preceding discussion, consider using Hive when the following conditions exist:

- Data easily fits into a table structure.
- Data is already in HDFS. (Note: Non-HDFS files can be loaded into a Hive table.)
- Developers are comfortable with SQL programming and queries.
- There is a desire to partition datasets based on time. (For example, daily updates are added to the Hive table.)
- Batch processing is acceptable.

The remainder of the Hive discussion covers some HiveQL basics. From the command prompt, a user enters the interactive Hive environment by simply entering **`hive`**:

```
$ hive
hive>
```

From this environment, a user can define new tables, query them, or summarize their contents. To illustrate how to use HiveQL, the following example defines a new Hive table to hold customer data, load existing HDFS data into the Hive table, and query the table.

The first step is to create a table called `customer` to store customer details. Because the table will be populated from an existing tab ('\t')-delimited HDFS file, this format is specified in the table creation query.

```
hive> create table customer (
                  cust_id bigint,
                  first_name string,
                  last_name  string,
                  email_address  string)
                  row format delimited
                  fields terminated by '\t';
```

The following HiveQL query is executed to count the number of records in the newly created table, `customer`. Because the table is currently empty, the query returns a result of zero, the last line of the provided output. The query is converted and run as a MapReduce job, which results in one map task and one reduce task being executed.

```
hive> select count(*) from customer;
Total MapReduce jobs = 1

Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
Starting Job = job_1394125045435_0001, Tracking URL =
     http://pivhdsne:8088/proxy/application_1394125045435_0001/
Kill Command = /usr/lib/gphd/hadoop/bin/hadoop job
     -kill job_1394125045435_0001
Hadoop job information for Stage-1: number of mappers: 1;
     number of reducers: 1
2014-03-06 12:30:23,542 Stage-1 map = 0%,  reduce = 0%
2014-03-06 12:30:36,586 Stage-1 map = 100%,  reduce = 0%,
     Cumulative CPU 1.71 sec
2014-03-06 12:30:48,500 Stage-1 map = 100%,  reduce = 100%,
     Cumulative CPU 3.76 sec
```

```
MapReduce Total cumulative CPU time: 3 seconds 760 msec
Ended Job = job_1394125045435_0001
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 3.76 sec   HDFS Read: 242
     HDFS Write: 2 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 760 msec
OK
0
```

When querying large tables, Hive outperforms and scales better than most conventional database queries. As stated earlier, Hive translates HiveQL queries into MapReduce jobs that process pieces of large datasets in parallel.

To load the customer table with the contents of HDFS file, `customer.txt`, it is only necessary to provide the HDFS directory path to the file.

```
hive>  load data inpath '/user/customer.txt' into table customer;
```

The following query displays three rows from the `customer` table.

```
hive>  select * from customer limit 3;
34567678     Mary   Jones        mary.jones@isp.com
897572388    Harry  Schmidt      harry.schmidt@isp.com
89976576     Tom    Smith        thomas.smith@another_isp.com
```

It is often necessary to join one or more Hive tables based on one or more columns. The following example provides the mechanism to join the `customer` table with another table, `orders`, which stores the details about the customer's orders. Instead of placing all the customer details in the order table, only the corresponding `cust_id` appears in the `orders` table.

```
hive>  select o.order_number, o.order_date, c.*
              from orders o inner join customer c
              on o.cust_id = c.cust_id
              where c.email_address = 'mary.jones@isp.com';

Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
Starting Job = job_1394125045435_0002, Tracking URL =
     http://pivhdsne:8088/proxy/application_1394125045435_0002/
Kill Command = /usr/lib/gphd/hadoop/bin/hadoop job
     -kill job_1394125045435_0002
Hadoop job information for Stage-1: number of mappers: 2;
     number of reducers: 1
2014-03-06 13:26:20,277 Stage-1 map = 0%,  reduce = 0%
2014-03-06 13:26:42,568 Stage-1 map = 50%, reduce = 0%,
     Cumulative CPU 4.23 sec
2014-03-06 13:26:43,637 Stage-1 map = 100%,reduce = 0%,
     Cumulative CPU 4.79 sec
2014-03-06 13:26:52,658 Stage-1 map = 100%,reduce = 100%,
     Cumulative CPU 7.07 sec
MapReduce Total cumulative CPU time: 7 seconds 70 msec
```

```
Ended Job = job_1394125045435_0002
MapReduce Jobs Launched:
Job 0: Map: 2  Reduce: 1   Cumulative CPU: 7.07 sec   HDFS Read: 602
     HDFS Write: 140 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 70 msec
OK
X234825811 2013-11-15 17:08:43 34567678  Mary Jones mary.jones@isp.com
X234823904 2013-11-04 12:53:19 34567678  Mary Jones mary.jones@isp.com
```

The use of joins and SQL in general will be covered in Chapter 11. To exit the Hive interactive environment, use `quit`.

```
hive> quit;
$
```

An alternative to running in the interactive environment is to collect the HiveQL statements in a script (for example, `my_script.sql`) and then execute the file as follows:

```
$ hive -f my_script.sql
```

This introduction to Hive provided some of the basic HiveQL commands and statements. The reader is encouraged to research and utilize, when appropriate, other Hive functionality such as external tables, explain plans, partitions, and the `INSERT INTO` command to append data to the existing content of a Hive table.

Following are some Hive use cases:

- **Exploratory or ad-hoc analysis of HDFS data:** Data can be queried, transformed, and exported to analytical tools, such as R.
- **Extracts or data feeds to reporting systems, dashboards, or data repositories such as HBase:** Hive queries can be scheduled to provide such periodic feeds.
- **Combining external structured data to data already residing in HDFS:** Hadoop is excellent for processing unstructured data, but often there is structured data residing in an RDBMS, such as Oracle or SQL Server, that needs to be joined with the data residing in HDFS. The data from an RDBMS can be periodically added to Hive tables for querying with existing data in HDFS.

10.2.3 HBase

Unlike Pig and Hive, which are intended for batch applications, Apache HBase is capable of providing real-time read and write access to datasets with billions of rows and millions of columns. To illustrate the differences between HBase and a relational database, this section presents considerable details about the implementation and use of HBase.

The HBase design is based on Google's 2006 paper on Bigtable. This paper described Bigtable as a "distributed storage system for managing structured data." Google used Bigtable to store Google product–specific data for sites such as Google Earth, which provides satellite images of the world. Bigtable was also used to store web crawler results, data for personalized search optimization, and website clickstream data. Bigtable was built on top of the Google File System. MapReduce was also utilized to process

data into or out of a Bigtable. For example, the raw clickstream data was stored in a Bigtable. Periodically, a scheduled MapReduce job would run that would process and summarize the newly added clickstream data and append the results to a second Bigtable [27].

The development of HBase began in 2006. HBase was included as part of a Hadoop distribution at the end of 2007. In May 2010, HBase became an Apache Top Level Project. Later in 2010, Facebook began to use HBase for its user messaging infrastructure, which accommodated 350 million users sending 15 billion messages per month [28].

HBase Architecture and Data Model

HBase is a data store that is intended to be distributed across a cluster of nodes. Like Hadoop and many of its related Apache projects, HBase is built upon HDFS and achieves its real-time access speeds by sharing the workload over a large number of nodes in a distributed cluster. An HBase table consists of rows and columns. However, an HBase table also has a third dimension, version, to maintain the different values of a row and column intersection over time.

To illustrate this third dimension, a simple example would be that for any given online customer, several shipping addresses could be stored. So, the row would be indicated by a customer number. One column would provide the shipping address. The value of the shipping address would be added at the intersection of the customer number and the shipping address column, along with a timestamp corresponding to when the customer last used this shipping address.

During a customer's checkout process from an online retailer, a website might use such a table to retrieve and display the customer's previous shipping addresses. As shown in Figure 10-6, the customer can then select the appropriate address, add a new address, or delete any addresses that are no longer relevant.

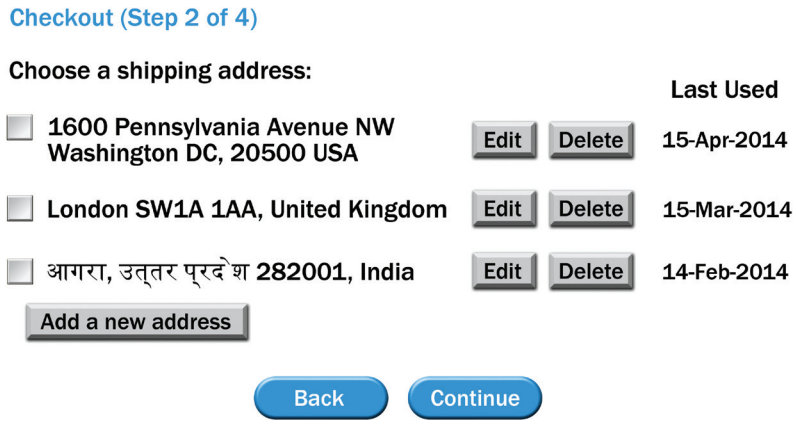

FIGURE 10-6 *Choosing a shipping address at checkout*

Of course, in addition to a customer's shipping address, other customer information, such as billing address, preferences, billing credits/debits, and customer benefits (for example, free shipping) must be stored. For this type of application, real-time access is required. Thus, the use of the batch processing of Pig, Hive, or Hadoop's MapReduce is not a reasonable implementation approach. The following discussion examines how HBase stores the data and provides real-time read and write access.

As mentioned, HBase is built on top of HDFS. HBase uses a key/value structure to store the contents of an HBase table. Each value is the data to be stored at the intersection of the row, column, and version. Each key consists of the following elements [29]:

- Row length
- Row (sometimes called the row key)
- Column family length
- Column family
- Column qualifier
- Version
- Key type

The ***row*** is used as the primary attribute to access the contents of an HBase table. The row is the basis for how the data is distributed across the cluster and allows a query of an HBase table to quickly retrieve the desired elements. Thus, the structure or layout of the row has to be specifically designed based on how the data will be accessed. In this respect, an HBase table is purpose built and is not intended for general ad-hoc querying and analysis. In other words, it is important to know how the HBase table will be used; this understanding of the table's usage helps to optimally define the construction of the row and the table.

For example, if an HBase table is to store the content of e-mails, the row may be constructed as the concatenation of an e-mail address and the date sent. Because the HBase table will be stored based on the row, the retrieval of the e-mails by a given e-mail address will be fairly efficient, but the retrieval of all e-mails in a certain date range will take much longer. The later discussion on regions provides more details on how data is stored in HBase.

A column in an HBase table is designated by the combination of the ***column family*** and the ***column qualifier***. The column family provides a high-level grouping for the column qualifiers. In the earlier shipping address example, the row could contain the `order_number`, and the order details could be stored under the column family `orders`, using the column qualifiers such as `shipping_address`, `billing_address`, `order_date`. In HBase, a column is specified as column family:column qualifier. In the example, the column `orders:shipping_address` refers to an order's shipping address.

A ***cell*** is the intersection of a row and a column in a table. The ***version***, sometimes called the ***timestamp***, provides the ability to maintain different values for a cell's contents in HBase. Although the user can define a custom value for the version when writing an entry to the table, a typical HBase implementation uses HBase's default, the current system time. In Java, this timestamp is obtained with `System.getCurrentTimeMillis()`, the number of milliseconds since January 1, 1970. Because it is likely that only the most recent version of a cell may be required, the cells are stored in descending order of the version. If the application requires the cells to be stored and retrieved in ascending order of their creation time, the approach is to use `Long.MAX_VALUE - System.getCurrentTimeMillis()` in Java as the version number. `Long.MAX_VALUE` corresponds to the maximum value that a long integer can be in Java. In this case, the storing and sorting is still in descending order of the version values.

Key type is used to identify whether a particular key corresponds to a write operation to the HBase table or a delete operation from the table. Technically, a delete from an HBase table is accomplished with a write to the table. The key type indicates the purpose of the write. For deletes, a tombstone marker is

written to the table to indicate that all cell versions equal to or older than the specified timestamp should be deleted for the corresponding row and column `family:column` qualifier.

Once an HBase environment is installed, the user can enter the HBase shell environment by entering **`hbase shell`** at the command prompt. An HBase table, `my_table`, can then be created as follows:

```
$ hbase shell
hbase> create 'my_table', 'cf1', 'cf2',
             {SPLITS =>['250000','500000','750000']}
```

Two column families, `cf1` and `cf2`, are defined in the table. The `SPLITS` option specifies how the table will be divided based on the row portion of the key. In this example, the table is split into four parts, called ***regions***. Rows less than `250000` are added to the first region; rows from `250000` to less than `500000` are added to the second region, and likewise for the remaining splits. These splits provide the primary mechanism for achieving the real-time read and write access. In this example, `my_table` is split into four regions, each on its own worker node in the Hadoop cluster. Thus, as the table size increases or the user load increases, additional worker nodes and region splits can be added to scale the cluster appropriately. The reads and writes are based on the contents of the row. HBase can quickly determine the appropriate region to direct a read or write command. More about regions and their implementation will be discussed later.

Only column families, not column qualifiers, need to be defined during HBase table creation. New column qualifiers can be defined whenever data is written to the HBase table. Unlike most relational databases, in which a database administrator needs to add a column and define the data type, columns can be added to an HBase table as the need arises. Such flexibility is one of the strengths of HBase and is certainly desirable when dealing with unstructured data. Over time, the unstructured data will likely change. Thus, the new content with new column qualifiers must be extracted and added to the HBase table.

Column families help to define how the table will be physically stored. An HBase table is split into regions, but each region is split into column families that are stored separately in HDFS. From the Linux command prompt, running `hadoop fs -ls -R /hbase` shows how the HBase table, *`my_table`*, is stored in HBase.

```
$ hadoop fs -ls -R /hbase

  0 2014-02-28 16:40 /hbase/my_table/028ed22e02ad07d2d73344cd53a11fb4
243 2014-02-28 16:40 /hbase/my_table/028ed22e02ad07d2d73344cd53a11fb4/
                     .regioninfo
  0 2014-02-28 16:40 /hbase/my_table/028ed22e02ad07d2d73344cd53a11fb4/
                     cf1
  0 2014-02-28 16:40 /hbase/my_table/028ed22e02ad07d2d73344cd53a11fb4/
                     cf2
  0 2014-02-28 16:40 /hbase/my_table/2327b09784889e6198909d8b8f342289
255 2014-02-28 16:40 /hbase/my_table/2327b09784889e6198909d8b8f342289/
                     .regioninfo
  0 2014-02-28 16:40 /hbase/my_table/2327b09784889e6198909d8b8f342289/
                     cf1
  0 2014-02-28 16:40 /hbase/my_table/2327b09784889e6198909d8b8f342289/
                     cf2
  0 2014-02-28 16:40 /hbase/my_table/4b4fc9ad951297efe2b9b38640f7a5fd
267 2014-02-28 16:40 /hbase/my_table/4b4fc9ad951297efe2b9b38640f7a5fd/
                     .regioninfo
```

```
  0 2014-02-28 16:40 /hbase/my_table/4b4fc9ad951297efe2b9b38640f7a5fd/
                     cf1
  0 2014-02-28 16:40 /hbase/my_table/4b4fc9ad951297efe2b9b38640f7a5fd/
                     cf2
  0 2014-02-28 16:40 /hbase/my_table/e40be0371f43135e36ea67edec6e31e3
267 2014-02-28 16:40 /hbase/my_table/e40be0371f43135e36ea67edec6e31e3/
                     .regioninfo
  0 2014-02-28 16:40 /hbase/my_table/e40be0371f43135e36ea67edec6e31e3/
                     cf1
  0 2014-02-28 16:40 /hbase/my_table/e40be0371f43135e36ea67edec6e31e3/
                     cf2
```

As can be seen, four subdirectories have been created under `/hbase/mytable`. Each subdirectory is named by taking the hash of its respective region name, which includes the start and end rows. Under each of these directories are the directories for the column families, `cf1` and `cf2` in the example, and the `.regioninfo` file, which contains several options and attributes for how the regions will be maintained. The column family directories store keys and values for the corresponding column qualifiers. The column qualifiers from one column family should seldom be read with the column qualifiers from another column family. The reason for the separate column families is to minimize the amount of unnecessary data that HBase has to sift through within a region to find the requested data. Requesting data from two column families means that multiple directories have to be scanned to pull all the desired columns, which defeats the purpose of creating the column families in the first place. In such cases, the table design may be better off with just one column family. In practice, the number of column families should be no more than two or three. Otherwise, performance issues may arise [30].

The following operations add data to the table using the `put` command. From these three `put` operations, `data1` and `data2` are entered into column qualifiers, `cq1` and `cq2`, respectively, in column family `cf1`. The value `data3` is entered into column qualifier `cq3` in column family `cf2`. The row is designated by row key `000700` in each operation.

```
hbase> put 'my_table', '000700', 'cf1:cq1', 'data1'

0 row(s) in 0.0030 seconds

hbase> put 'my_table', '000700', 'cf1:cq2', 'data2'

0 row(s) in 0.0030 seconds

hbase> put 'my_table', '000700', 'cf2:cq3', 'data3'

0 row(s) in 0.0040 seconds
```

Data can be retrieved from the HBase table by using the `get` command. As mentioned earlier, the timestamp defaults to the milliseconds since January 1, 1970.

```
hbase> get 'my_table', '000700', 'cf2:cq3'

COLUMN      CELL
 cf2:cq3    timestamp=1393866138714, value=data3
1 row(s) in 0.0350 seconds
```

By default, the `get` command returns the most recent version. To illustrate, after executing a second `put` operation in the same row and column, a subsequent `get` provides the most recently added value of `data4`.

```
hbase> put 'my_table', '000700', 'cf2:cq3', 'data4'

0 row(s) in 0.0040 seconds

hbase> get 'my_table', '000700', 'cf2:cq3'

COLUMN      CELL
 cf2:cq3    timestamp=1393866431669, value=data4
1 row(s) in 0.0080 seconds
```

The `get` operation can provide multiple versions by specifying the number of versions to retrieve. This example illustrates that the cells are presented in descending version order.

```
hbase> get 'my_table', '000700', {COLUMN => 'cf2:cq3', VERSIONS => 2}

COLUMN      CELL
 cf2:cq3    timestamp=1393866431669, value=data4
 cf2:cq3    timestamp=1393866138714, value=data3
2 row(s) in 1.0200 seconds
```

A similar operation to the `get` command is `scan`. A `scan` retrieves all the rows between a specified `STARTROW` and a `STOPROW`, but excluding the `STOPROW`. Note: if the `STOPROW` was set to `000700`, only row `000600` would have been returned.

```
hbase> scan 'my_table', {STARTROW => '000600', STOPROW =>'000800'}

ROW          COLUMN+CELL
 000600      column=cf1:cq2, timestamp=1393866792008, value=data5
 000700      column=cf1:cq1, timestamp=1393866105687, value=data1
 000700      column=cf1:cq2, timestamp=1393866122073, value=data2
 000700      column=cf2:cq3, timestamp=1393866431669, value=data4
2 row(s) in 0.0400 seconds
```

The next operation deletes the oldest entry for column `cf2:cq3` for row `000700` by specifying the timestamp.

```
hbase> delete 'my_table', '000700', 'cf2:cq3', 1393866138714
0 row(s) in 0.0110 seconds
```

Repeating the earlier `get` operation to obtain both versions only provides the last version for that cell. After all, the older version was deleted.

```
hbase> get 'my_table', '000700', {COLUMN => 'cf2:cq3', VERSIONS => 2}

COLUMN      CELL
 cf2:cq3    timestamp=1393866431669, value=data4
1 row(s) in 0.0130 seconds
```

However, running a `scan` operation, with the RAW option set to `true`, reveals that the deleted entry actually remains. The highlighted line illustrates the creation of a tombstone marker, which informs the default `get` and `scan` operations to ignore all older cell versions of the particular row and column.

```
hbase> scan 'my_table', {RAW => true, VERSIONS => 2,
            STARTROW => '000700'}

ROW           COLUMN+CELL
 000700       column=cf1:cq1, timestamp=1393866105687, value=data1
 000700       column=cf1:cq2, timestamp=1393866122073, value=data2
 000700       column=cf2:cq3, timestamp=1393866431669, value=data4
 000700     column=cf2:cq3, timestamp=1393866138714, type=DeleteColumn
 000700       column=cf2:cq3, timestamp=1393866138714, value=data3
1 row(s) in 0.0370 seconds
```

When will the deleted entries be permanently removed? To understand this process, it is necessary to understand how HBase processes operations and achieves the real-time read and write access. As mentioned earlier, an HBase table is split into regions based on the row. Each region is maintained by a worker node. During a `put` or `delete` operation against a particular region, the worker node first writes the command to a Write Ahead Log (WAL) file for the region. The WAL ensures that the operations are not lost if a system fails. Next, the results of the operation are stored within the worker node's RAM in a repository called MemStore [31].

Writing the entry to the MemStore provides the real-time access required. Any client can access the entries in the MemStore as soon as they are written. As the MemStore increases in size or at predetermined time intervals, the sorted MemStore is then written (flushed) to a file, known as an HFile, in HDFS on the same worker node. A typical HBase implementation flushes the MemStore when its contents are slightly less than the HDFS block size. Over time, these flushed files accumulate, and the worker node performs a ***minor compaction*** that performs a sorted merge of the various flushed files.

Meanwhile, any `get` or `scan` requests that the worker node receives examine these possible storage locations:

- MemStore
- HFiles resulting from MemStore flushes
- HFiles from minor compactions

Thus, in the case of a `delete` operation followed relatively quickly by a `get` operation on the same row, the tombstone marker is found in the MemStore and the corresponding previous versions in the smaller HFiles or previously merged HFiles. The `get` command is instantaneously processed and the appropriate data returned to the client.

Over time, as the smaller HFiles accumulate, the worker node runs a ***major compaction*** that merges the smaller HFiles into one large HFile. During the major compaction, the deleted entries and the tombstone markers are permanently removed from the files.

Use Cases for HBase

As described in Google's Bigtable paper, a common use case for a data store such as HBase is to store the results from a web crawler. Using this paper's example, the row `com.cnn.www`, for example, corresponds

to a website URL, www.cnn.com. A column family, called anchor, is defined to capture the website URLs that provide links to the row's website. What may not be an obvious implementation is that those anchoring website URLs are used as the column qualifiers. For example, if sportsillustrated.cnn.com provides a link to www.cnn.com, the column qualifier is sportsillustrated.cnn.com. Additional websites that provide links to www.cnn.com appear as additional column qualifiers. The value stored in the cell is simply the text on the website that provides the link. Here is how the CNN example may look in HBase following a get operation.

```
hbase> get 'web_table', 'com.cnn.www', {VERSIONS => 2}

COLUMN                                  CELL
 anchor:sportsillustrated.cnn.com       timestamp=1380224620597, value=cnn
 anchor:sportsillustrated.cnn.com       timestamp=1380224000001, value=cnn.com
 anchor:edition.cnn.com                 timestamp=1380224620597, value=cnn
```

Additional results are returned for each corresponding website that provides a link to www.cnn.com. Finally, an explanation is required for using com.cnn.www for the row instead of www.cnn.com. By reversing the URLs, the various suffixes (.com, .gov, or .net) that correspond to the Internet's top-level domains are stored in order. Also, the next part of the domain name (cnn) is stored in order. So, all of the cnn.com websites could be retrieved by a scan with the STARTROW of com.cnn and the appropriate STOPROW.

This simple use case illustrates several important points. First, it is possible to get to a billion rows and millions of columns in an HBase table. As of February 2014, more than 920 million websites have been identified [32]. Second, the row needs to be defined based on how the data will be accessed. An HBase table needs to be designed with a specific purpose in mind and a well-reasoned plan for how data will be read and written. Finally, it may be advantageous to use the column qualifiers to actually store the data of interest, rather than simply storing it in a cell. In the example, as new hosting websites are established, they become new column qualifiers.

A second use case is the storage and search access of messages. In 2010, Facebook implemented such a system using HBase. At the time, Facebook's system was handling more than 15 billion user-to-user messages per month and 120 billion chat messages per month [33]. The following describes Facebook's approach to building a search index for user inboxes. Using each word in each user's message, an HBase table was designed as follows:

- The row was defined to be the user ID.
- The column qualifier was set to a word that appears in the message.
- The version was the message ID.
- The cell's content was the offset of the word in the message.

This implementation allowed Facebook to provide auto-complete capability in the search box and to return the results of the query quickly, with the most recent messages at the top. As long as the message IDs increase over time, the versions, stored in descending order, ensure that the most recent e-mails are returned first to the user [34].

These two use cases help illustrate the importance of the upfront design of the HBase table based on how the data will be accessed. Also, these examples illustrate the power of being able to add new columns

by adding new column qualifiers, on demand. In a typical RDBMS implementation, new columns require the involvement of a DBA to alter the structure of the table.

Other HBase Usage Considerations

In addition to the HBase design aspects presented in the use case discussions, the following considerations are important for a successful implementation.

- **Java API:** Previously, several HBase shell commands and operations were presented. The shell commands are useful for exploring the data in an HBase environment and illustrating their use. However, in a production environment, the HBase Java API could be used to program the desired operations and the conditions in which to execute the operations.
- **Column family and column qualifier names:** It is important to keep the name lengths of the column families and column qualifiers as short as possible. Although short names tend to go against conventional wisdom about using meaningful, descriptive names, the names of column family name and the column qualifier are stored as part of the key of each key/value pair. Thus, every additional byte added to a name over each row can quickly add up. Also, by default, three copies of each HDFS block are replicated across the Hadoop cluster, which triples the storage requirement.
- **Defining rows:** The definition of the row is one of the most important aspects of the HBase table design. In general, this is the main mechanism to perform read/write operations on an HBase table. The row needs to be constructed in such a way that the requested columns can be easily and quickly retrieved.
- **Avoid creating sequential rows:** A natural tendency is to create rows sequentially. For example, if the row key is to have the customer identification number, and the customer identification numbers are created sequentially, HBase may run into a situation in which all the new users and their data are being written to just one region, which is not distributing the workload across the cluster as intended [35]. An approach to resolve such a problem is to randomly assign a prefix to the sequential number.
- **Versioning control:** HBase table options that can be defined during table creation or altered later control how long a version of a cell's contents will exist. There are options for TimeToLive (TTL) after which any older versions will be deleted. Also, there are options for the minimum and maximum number of versions to maintain.
- **Zookeeper:** HBase uses Apache Zookeeper to coordinate and manage the various regions running on the distributed cluster. In general, Zookeeper is "a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications." [36] Instead of building its own coordination service, HBase uses Zookeeper. Relative to HBase, there are some Zookeeper configuration considerations [37].

10.2.4 Mahout

The majority of this chapter has focused on processing, structuring, and storing large datasets using Apache Hadoop and various parts of its ecosystem. After a dataset is available in HDFS, the next step may be to apply an analytical technique presented in Chapters 4 through 9. Tools such as R are useful for analyzing relatively small datasets, but they may suffer from performance issues with the large datasets stored in Hadoop. To apply the analytical techniques within the Hadoop environment, an option is to use Apache

Mahout. This Apache project provides executable Java libraries to apply analytical techniques in a scalable manner to Big Data. In general, a mahout is a person who controls an elephant. Apache Mahout is the toolset that directs Hadoop, the elephant in this case, to yield meaningful analytic results.

Mahout provides Java code that implements the algorithms for several techniques in the following three categories [38]:

Classification:

- Logistic regression
- Naïve Bayes
- Random forests
- Hidden Markov models

Clustering:

- Canopy clustering
- K-means clustering
- Fuzzy k-means
- Expectation maximization (EM)

Recommenders/collaborative filtering:

- Nondistributed recommenders
- Distributed item-based collaborative filtering

Pivotal HD Enterprise with HAWQ

Users can download and install Apache Hadoop and the described ecosystem tools directly from the `www.apache.org` website. Another installation option is downloading commercially packaged distributions of the various Apache Hadoop projects. These distributions often include additional user functionality as well as cluster management utilities. Pivotal is a company that provides a distribution called Pivotal HD Enterprise, as illustrated in Figure 10-7.

Pivotal HD Enterprise includes several Apache software components that have been presented in this chapter. Additional Apache software includes the following:

- **Oozie:** Manages Apache Hadoop jobs by acting as a workflow scheduler system
- **Sqoop:** Efficiently moves data between Hadoop and relational databases
- **Flume:** Collects and aggregates streaming data (for example, log data)

Additional functionality provided by Pivotal includes [39] the following:

- **Command Center** is a robust cluster management tool that allows users to install, configure, monitor, and manage Hadoop components and services through a web graphical interface. It simplifies Hadoop cluster installation, upgrades, and expansion using a comprehensive dashboard with instant views of the health of the cluster and key performance metrics. Users can view live and historical information about the host, application,

and job-level metrics across the entire Pivotal HD cluster. Command Center also provides CLI and web services APIs for integration into enterprise monitoring services.

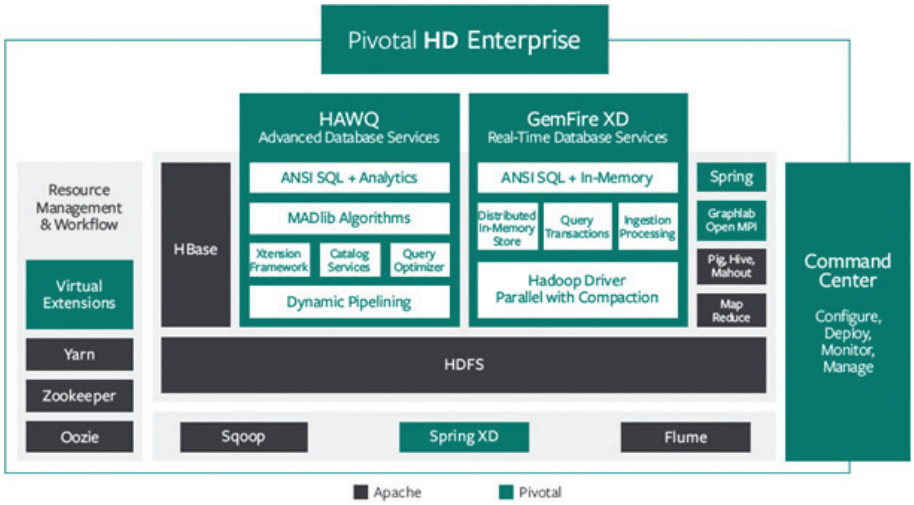

Figure 10-7 *Components of Pivotal HD Enterprise*

- **Graphlab on Open MPI (Message Passing Interface)** is a highly used and mature graph-based, high-performing, distributed computation framework that easily scales to graphs with billions of vertices and edges. It is now able to run natively within an existing Hadoop cluster, eliminating costly data movement. This allows data scientists and analysts to leverage popular algorithms such as page rank, collaborative filtering, and computer vision natively in Hadoop rather than copying the data somewhere else to run the analytics, which would lengthen data science cycles. Combined with MADlib's machine learning algorithms for relational data, Pivotal HD becomes the leading advanced analytical platform for machine learning in the world.
- **Hadoop Virtualization Extensions (HVE)** plug-ins make Hadoop aware of the virtual topology and scale Hadoop nodes dynamically in a virtual environment. Pivotal HD is the first Hadoop distribution to include HVE plug-ins, enabling easy deployment of Hadoop in an enterprise environment. With HVE, Pivotal HD can deliver truly elastic scalability in the cloud, augmenting on-premises deployment options.
- **HAWQ (HAdoop With Query)** adds SQL's expressive power to Hadoop to accelerate data analytics projects, simplify development while increasing productivity, expand Hadoop's capabilities, and cut costs. HAWQ can help render Hadoop queries faster than any Hadoop-based query interface on the market by adding rich, proven, parallel SQL processing facilities. HAWQ leverages existing business intelligence and analytics products and a workforce's existing SQL skills to bring more than 100 times performance improvement to a wide range of query types and workloads.

10.3 NoSQL

NoSQL (Not only Structured Query Language) is a term used to describe those data stores that are applied to unstructured data. As described earlier, HBase is such a tool that is ideal for storing key/values in column families. In general, the power of NoSQL data stores is that as the size of the data grows, the implemented solution can scale by simply adding additional machines to the distributed system. Four major categories of NoSQL tools and a few examples are provided next [40].

Key/value stores contain data (the value) that can be simply accessed by a given identifier (the key). As described in the MapReduce discussion, the values can be complex. In a key/value store, there is no stored structure of how to use the data; the client that reads and writes to a key/value store needs to maintain and utilize the logic of how to meaningfully extract the useful elements from the key and the value. Here are some uses for key/value stores:

- Using a customer's login ID as the key, the value contains the customer's preferences.
- Using a web session ID as the key, the value contains everything that was captured during the session.

Document stores are useful when the value of the key/value pair is a file and the file itself is self-describing (for example, JSON or XML). The underlying structure of the documents can be used to query and customize the display of the documents' content. Because the document is self-describing, the document store can provide additional functionality over a key/value store. For example, a document store may provide the ability to create indexes to speed the searching of the documents. Otherwise, every document in the data store would have to be examined. Document stores may be useful for the following:

- Content management of web pages
- Web analytics of stored log data

Column family stores are useful for sparse datasets, records with thousands of columns but only a few columns have entries. The key/value concept still applies, but in this case a key is associated with a collection of columns. In this collection, related columns are grouped into column families. For example, columns for age, gender, income, and education may be grouped into a demographic family. Column family data stores are useful in the following instances:

- To store and render blog entries, tags, and viewers' feedback
- To store and update various web page metrics and counters

Graph databases are intended for use cases such as networks, where there are items (people or web page links) and relationships between these items. While it is possible to store graphs such as trees in a relational database, it often becomes cumbersome to navigate, scale, and add new relationships. Graph databases help to overcome these possible obstacles and can be optimized to quickly traverse a graph (move from one item in the network to another item in the network). Following are examples of graph database implementations:

- Social networks such as Facebook and LinkedIn
- Geospatial applications such as delivery and traffic systems to optimize the time to reach one or more destinations

Table 10-2 provides a few examples of NoSQL data stores. As is often the case, the choice of a specific data store should be made based on the functional and performance requirements. A particular data store may provide exceptional functionality in one aspect, but that functionality may come at a loss of other functionality or performance.

Table 10-2 *Examples of NoSQL Data Stores*

Category	Data Store	Website
Key/Value	Redis	`redis.io`
	Voldemort	`www.project-voldemort.com/voldemort`
Document	CouchDB	`couchdb.apache.org`
	MongoDB	`www.mongodb.org`
Column family	Cassandra	`cassandra.apache.org`
	HBase	`hbase.apache.org/`
Graph	FlockDB	`github.com/twitter/flockdb`
	Neo4j	`www.neo4j.org`

Summary

This chapter examined the MapReduce paradigm and its application in Big Data analytics. Specifically, it examined the implementation of MapReduce in Apache Hadoop. The power of MapReduce is realized with the use of the Hadoop Distributed File System (HDFS) to store data in a distributed system. The ability to run a MapReduce job on the data stored across a cluster of machines enables the parallel processing of petabytes or exabytes of data. Furthermore, by adding additional machines to the cluster, Hadoop can scale as the data volumes grow.

This chapter examined several Apache projects within the Hadoop ecosystem. By providing a higher-level programming language, Apache Pig and Hive simplify the code development by masking the underlying MapReduce logic to perform common data processing tasks such as filtering, joining datasets, and restructuring data. Once the data is properly conditioned within the Hadoop cluster, Apache Mahout can be used to conduct data analyses such as clustering, classification, and collaborative filtering.

The strength of MapReduce in Apache Hadoop and the so far mentioned projects in the Hadoop ecosystem are in batch processing environments. When real-time processing, including read and writes, are required, Apache HBase is an option. HBase uses HDFS to store large volumes of data across the cluster, but it also maintains recent changes within memory to ensure the real-time availability of the latest data. Whereas MapReduce in Hadoop, Pig, and Hive are more general-purpose tools that can address a wide range of tasks, HBase is a somewhat more purpose-specific tool. Data will be retrieved from and written to the HBase in a well-understood manner.

HBase is one example of the NoSQL (Not only Structured Query Language) data stores that are being developed to address specific Big Data use cases. Maintaining and traversing social network graphs are examples of relational databases not being the best choice as a data store. However, relational databases and SQL remain powerful and common tools and will be examined in more detail in Chapter 11.

Exercises

1. Research and document additional use cases and actual implementations for Hadoop.
2. Compare and contrast Hadoop, Pig, Hive, and HBase. List strengths and weaknesses of each tool set. Research and summarize three published use cases for each tool set.

   Exercises 3 through 5 require some programming background and a working Hadoop environment. The text of the novel *War and Peace* can be downloaded from `http://onlinebooks.library.upenn.edu/` and used as the dataset for these exercises. However, other datasets can easily be substituted. Document all processing steps applied to the data.
3. Use MapReduce in Hadoop to perform a word count on the specified dataset.
4. Use Pig to perform a word count on the specified dataset.
5. Use Hive to perform a word count on the specified dataset.

Bibliography

[1] Apache, "Apache Hadoop," [Online]. Available: `http://hadoop.apache.org/`. [Accessed 8 May 2014].

[2] Wikipedia, "IBM Watson," [Online]. Available: `http://en.wikipedia.org/wiki/IBM_Watson`. [Accessed 11 Februry 2014].

[3] D. Davidian, "IBM.com," 14 February 2011. [Online]. Available: `https://www-304.ibm.com/connections/blogs/davidian/tags/hadoop?lang=en_us`. [Accessed 11 February 2014].

[4] IBM, "IBM.com," [Online]. Available: `http://www-03.ibm.com/innovation/us/watson/watson_in_healthcare.shtml`. [Accessed 11 February 2014].

[5] Linkedin, "LinkedIn," [Online]. Available: `http://www.linkedin.com/about-us`. [Accessed 11 February 2014].

[6] LinkedIn, "Hadoop," [Online]. Available: `http://data.linkedin.com/projects/hadoop`. [Accessed 11 February 2014].

[7] S. Singh, "http://developer.yahoo.com/," [Online]. Available: `http://developer.yahoo.com/blogs/hadoop/apache-hbase-yahoo-multi-tenancy-helm-again-171710422.html`. [Accessed 11 February 2014].

[8] E. Baldeschwieler, "http://www.slideshare.net," [Online]. Available: http://www.slideshare.net/ydn/hadoop-yahoo-internet-scale-data-processing. [Accessed 11 February 2014].

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," [Online]. Available: http://research.google.com/archive/mapreduce.html. [Accessed 11 February 2014].

[10] D. Gottfrid, "Self-Service, Prorated Supercomputing Fun!," 01 November 2007. [Online]. Available: http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/. [Accessed 11 February 2014].

[11] "apache.org," [Online]. Available: http://www.apache.org/. [Accessed 11 February 2014].

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," [Online]. Available: http://static.googleusercontent.com/media/research.google.com/en/us/archive/gfs-sosp2003.pdf. [Accessed 11 February 2014].

[13] D. Cutting, "Free Search: Rambilings About Lucene, Nutch, Hadoop and Other Stuff," [Online]. Available: http://cutting.wordpress.com. [Accessed 11 February 2014].

[14] "Hadoop Wiki Disk Setup," [Online]. Available: http://wiki.apache.org/hadoop/DiskSetup. [Accessed 20 February 2014].

[15] "wiki.apache.org/hadoop," [Online]. Available: http://wiki.apache.org/hadoop/NameNode. [Accessed 11 February 2014].

[16] "HDFS High Availability," [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithNFS.html. [Accessed 8 May 2014].

[17] Eclipse. [Online]. Available: https://www.eclipse.org/downloads/. [Accessed 27 February 2014].

[18] Apache, "Hadoop Streaming," [Online]. Available: https://wiki.apache.org/hadoop/HadoopStreaming. [Accessed 8 May 2014].

[19] "Hadoop Pipes," [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/api/org/apache/hadoop/mapred/pipes/package-summary.html. [Accessed 19 February 2014].

[20] "HDFS Design," [Online]. Available: http://hadoop.apache.org/docs/stable1/hdfs_design.html. [Accessed 19 February 2014].

[21] "BSP Tutorial," [Online]. Available: http://hama.apache.org/hama_bsp_tutorial.html. [Accessed 20 February 2014].

[22] "Hama," [Online]. Available: http://hama.apache.org/. [Accessed 20 February 2014].

[23] "PoweredByYarn," [Online]. Available: http://wiki.apache.org/hadoop/PoweredByYarn. [Accessed 20 February 2014].

[24] "pig.apache.org," [Online]. Available: http://pig.apache.org/.

[25] "Pig," [Online]. Available: http://pig.apache.org/. [Accessed 11 Feb 2014].

[26] "Piggybank," [Online]. Available: https://cwiki.apache.org/confluence/display/PIG/PiggyBank. [Accessed 28 February 2014].

[27] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber Fay Chang, "Bigtable: A Distributed Storage System for Structured Data," [Online]. Available: http://research.google.com/archive/bigtable.html. [Accessed 11 February 2014].

[28] K. Muthukkaruppan, "The Underlying Technology of Messages," 15 November 2010. [Online]. Available: http://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919. [Accessed 11 February 2014].

[29] "HBase Key Value," [Online]. Available: http://hbase.apache.org/book/regions.arch.html. [Accessed 28 February 2014].

[30] "Number of Column Families," [Online]. Available: http://hbase.apache.org/book/number.of.cfs.html.

[31] "HBase Regionserver," [Online]. Available: http://hbase.apache.org/book/regionserver.arch.html. [Accessed 3 March 2014].

[32] "Netcraft," [Online]. Available: http://news.netcraft.com/archives/2014/02/03/february-2014-web-server-survey.html. [Accessed 21 February 2014].

[33] K. Muthukkaruppan, "The Underlying Technology of Messages," 15 November 2010. [Online]. Available: http://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919. [Accessed 2011 February 2014].

[34] N. Spiegelberg. [Online]. Available: http://www.slideshare.net/brizzzdotcom/facebook-messages-hbase. [Accessed 11 February 2014].

[35] "HBase Rowkey," [Online]. Available: http://hbase.apache.org/book/rowkey.design.html. [Accessed 4 March 2014].

[36] "Zookeeper," [Online]. Available: http://zookeeper.apache.org/. [Accessed 11 Feb 2014].

[37] "Zookeeper," [Online]. Available: http://hbase.apache.org/book/zookeeper.html. [Accessed 21 February 2014].

[38] "Mahout," [Online]. Available: http://mahout.apache.org/users/basics/algorithms.html. [Accessed 19 February 2014].

[39] "Pivotal HD," [Online]. Available: http://www.gopivotal.com/big-data/pivotal-hd. [Accessed 8 May 2014].

[40] P. J. Sadalage and M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot*, Upper Saddle River, NJ: Addison Wesley, 2013.

11

Advanced Analytics—Technology and Tools: In-Database Analytics

Key Concepts

MADlib
Regular expressions
SQL
User-defined functions
Window functions

In-database analytics is a broad term that describes the processing of data within its repository. In many of the earlier R examples, data was extracted from a data source and loaded into R. One advantage of in-database analytics is that the need for movement of the data into an analytic tool is eliminated. Also, by performing the analysis within the database, it is possible to obtain almost real-time results. Applications of in-database analytics include credit card transaction fraud detection, product recommendations, and web advertisement selection tailored for a particular user.

A popular open-source database is PostgreSQL. This name references an important in-database analytic language known as ***Structured Query Language (SQL)***. This chapter examines basic as well as advanced topics in SQL. The provided examples of SQL code were tested against Greenplum database 4.1.1.1, which is based on PostgreSQL 8.2.15. However, the presented concepts are applicable to other SQL environments.

11.1 SQL Essentials

A relational database, part of a Relational Database Management System (RDBMS), organizes data in tables with established relationships between the tables. Figure 11-1 shows the relationships between five tables used to store details about orders placed at an e-commerce retailer.

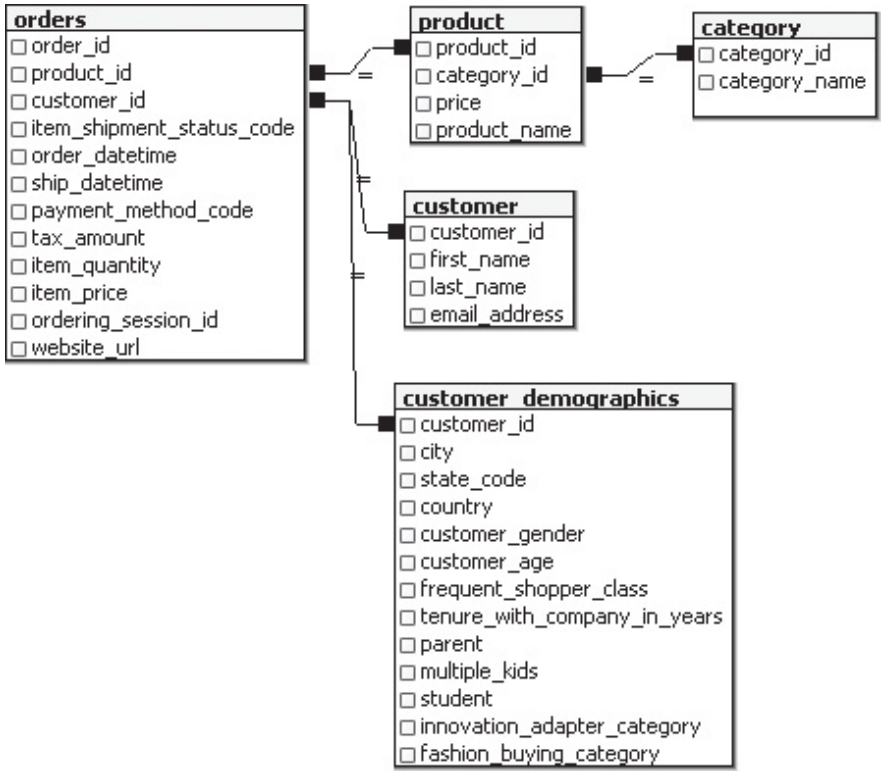

Figure 11-1 *Relationship diagram*

The table *orders* contains records for each order transaction. Each record contains data elements such as the *product_id* ordered, the *customer_id* for the customer who placed the order, the *order_datetime*, and so on. The other four tables provide additional details about the ordered items and the customer. The lines between the tables in Figure 11-1 illustrate the relationships between the tables. For example, a customer's first name, last name, and gender from the *customer* table can be associated with an *orders* record based on equality of the *customer_id* in these two tables.

Although it is possible to build one large table to hold all the order and customer details, the use of five tables has its advantages. The first advantage is disk storage savings. Instead of storing the product name, which can be several hundred characters in length, in the *orders* table, a much shorter *product_id*, of perhaps a few bytes, can be used and stored in place of the product's name.

Another advantage is that changes and corrections are easily made. In this example, the table *category* is used to categorize each product. If it is discovered that an incorrect category was assigned to a particular product item, only the *category_id* in the *product* table needs to be updated. Without the *product* and *category* tables, it may be necessary to update hundreds of thousands of records in the *orders* table.

A third advantage is that products can be added to the database prior to any orders being placed. Similarly, new categories can be created in anticipation of entirely new product lines being added to the online retailer's offerings later.

In a relational database design, the preference is not to duplicate pieces of data such as the customer's name across multiple records. The process of reducing such duplication is known as ***normalization***. It is important to recognize that a database that is designed to process transactions may not necessarily be optimally designed for analytical purposes. Transactional databases are often optimized to handle the insertion of new records or updates to existing records, but not optimally tuned to perform ad-hoc querying. Therefore, in designing analytical data warehouses, it is common to combine several of the tables and create one larger table, even though some pieces of data may be duplicated.

Regardless of a database's purpose, SQL is typically used to query the contents of the relational database tables as well as to insert, update, and delete data. A basic SQL query against the *customer* table may look like this.

```
SELECT first_name,
       last_name
FROM   customer
WHERE  customer_id = 666730

first_name   last_name
Mason        Hu
```

This query returns the customer information for the customer with a *customer_id* of 666730. This SQL query consists of three key parts:

- **SELECT:** Specifies the table columns to be displayed
- **FROM:** Specifies the name of the table to be queried
- **WHERE:** Specifies the criterion or filter to be applied

In a relational database, it is often necessary to access related data from multiple tables at once. To accomplish this task, the SQL query uses `JOIN` statements to specify the relationships between the multiple tables.

11.1.1 Joins

Joins enable a database user to appropriately select columns from two or more tables. Based on the relationship diagram in Figure 11-1, the following SQL query provides an example of the most common type of join: an inner join.

```
SELECT c.customer_id,
       o.order_id,
       o.product_id,
       o.item_quantity AS qty
FROM   orders o
       INNER JOIN customer c
               ON o.customer_id = c.customer_id
WHERE  c.first_name = 'Mason'
       AND c.last_name = 'Hu'
```

```
customer_id order_id               product_id  qty
666730      51965-1172-6384-6923   33611       5
666730      79487-2349-4233-6891   34098       1
666730      39489-4031-0789-6076   33928       1
666730      29892-1218-2722-3191   33625       1
666730      07751-7728-7969-3140   34140       4
666730      85394-8022-6681-4716   33571       1
```

This query returns details of the orders placed by customer Mason Hu. The SQL query joins the two tables in the `FROM` clause based on the equality of the *`customer_id`* values. In this query, the specific *`customer_id`* value for Mason Hu does not need to be known by the programmer; only the customer's full name needs to be known.

Some additional functionality beyond the use of the `INNER JOIN` is introduced in this SQL query. Aliases `o` and `c` are assigned to tables *`orders`* and *`customer`*, respectively. Aliases are used in place of the full table names to improve the readability of the query. By design, the column names specified in the `SELECT` clause are also provided in the output. However, the outputted column name can be modified with the `AS` keyword. In the SQL query, the values of *`item_quantity`* are displayed, but this outputted column is now called *`qty`*.

The `INNER JOIN` returns those rows from the two tables where the `ON` criterion is met. From the earlier query on the *`customer`* table, there is only one row in the table for customer Mason Hu. Because the corresponding *`customer_id`* for Mason Hu appears six times in the *`orders`* table, the `INNER JOIN` query returns six records. If the `WHERE` clause was not included, the query would have returned millions of rows for all the orders that had a matching customer.

Suppose an analyst wants to know which customers have created an online account but have not yet placed an order. The next query uses a `RIGHT OUTER JOIN` to identify the first five

customers, alphabetically, who have not placed an order. The sorting of the records is accomplished with the `ORDER BY` clause.

```
SELECT c.customer_id,
       c.first_name,
       c.last_name,
       o.order_id
FROM   orders o
       RIGHT OUTER JOIN customer c
                    ON o.customer_id = c.customer_id
WHERE  o.order_id IS NULL
ORDER  BY c.last_name,
          c.first_name
LIMIT 5
```

```
customer_id first_name  last_name   order_id
143915      Abigail     Aaron
965886      Audrey      Aaron
982042      Carter      Aaron
125302      Daniel      Aaron
103964      Emily       Aaron
```

In the SQL query, a `RIGHT OUTER JOIN` is used to specify that all rows from the table *`customer`*, on the right-hand side (RHS) of the join, should be returned, regardless of whether there is a matching *`customer_id`* in the *`orders`* table. In this query, the `WHERE` clause restricts the results to only those joined customer records where there is no matching *`order_id`*. `NULL` is a special SQL keyword that denotes an unknown value. Without the `WHERE` clause, the output also would have included all the records that had a matching *`customer_id`* in the *`orders`* table, as seen in the following SQL query.

```
SELECT c.customer_id,
       c.first_name,
       c.last_name,
       o.order_id
FROM   orders o
       RIGHT OUTER JOIN customer c
                    ON o.customer_id = c.customer_id
ORDER  BY c.last_name,
          c.first_name
LIMIT 5
```

```
customer_id  first_name    last_name    order_id
143915       Abigail       Aaron
222599       Addison       Aaron        50314-7576-3355-6960
222599       Addison       Aaron        21007-7541-1255-3531
222599       Addison       Aaron        19396-4363-4499-8582
222599       Addison       Aaron        69225-1638-2944-0264
```

In the query results, the first customer, Abigail Aaron, had not placed an order, but the next customer, Addison Aaron, has placed at least four orders.

There are several other types of join statements. The `LEFT OUTER JOIN` performs the same functionality as the `RIGHT OUTER JOIN` except that all records from the table on the left-hand side (LHS) of the join are considered. A `FULL OUTER JOIN` includes all records from both tables regardless of whether there is a matching record in the other table. A `CROSS JOIN` combines two tables by matching every row of the first table with every row of the second table. If the two tables have 100 and 1,000 rows, respectively, then the resulting `CROSS JOIN` of these tables will have 100,000 rows.

The actual records returned from any join operation depend on the criteria stated in the `WHERE` clause. Thus, careful consideration needs to be taken in using a `WHERE` clause, especially with outer joins. Otherwise, the intended use of the outer join may be undone.

11.1.2 Set Operations

SQL provides the ability to perform set operations, such as unions and intersections, on rows of data. For example, suppose all the records in the *orders* table are split into two tables. The *orders_arch* table, short for orders archived, contains the orders entered prior to January 2013. The orders transacted in or after January 2013 are stored in the *orders_recent* table. However, all the orders for *product_id* 33611 are required for an analysis. One approach would be to write and run two separate queries against the two tables. The results from the two queries could then be merged later into a separate file or table. Alternatively, one query could be written using the `UNION ALL` operator as follows:

```
SELECT customer_id,
       order_id,
       order_datetime,
       product_id,
       item_quantity AS qty
FROM   orders_arch
WHERE  product_id = 33611
UNION ALL
SELECT customer_id,
       order_id,
       order_datetime,
       product_id,
       item_quantity AS qty
FROM   orders_recent
WHERE  product_id = 33611
ORDER  BY order_datetime
```

```
customer_id  order_id              order_datetime      product_id  qty
643126       13501-6446-6326-0182  2005-01-02 19:28:08 33611       1
725940       70738-4014-1618-2531  2005-01-08 06:16:31 33611       1
742448       03107-1712-8668-9967  2005-01-08 16:11:39 33611       1
.
.
.
```

```
640847       73619-0127-0657-7016  2013-01-05 14:53:27 33611       1
660446       55160-7129-2408-9181  2013-01-07 03:59:36 33611       1
647335       75014-7339-1214-6447  2013-01-27 13:02:10 33611       1
.
.
.
```

The first three records from each table are shown in the output. Because the resulting records from both tables are appended together in the output, it is important that the columns are specified in the same order and that the data types of the columns are compatible. `UNION ALL` merges the results of the two `SELECT` statements regardless of any duplicate records appearing in both `SELECT` statements. If only `UNION` was used, any duplicate records, based on all the specified columns, would be eliminated.

The `INTERSECT` operator determines any identical records that are returned by two `SELECT` statements. For example, if one wanted to know what items were purchased prior to 2013 as well as later, the SQL query using the `INTERSECT` operator would be this.

```
SELECT product_id
FROM   orders_arch
INTERSECT
SELECT product_id
FROM   orders_recent

product_id
22
30
31
.
.
.
```

It is important to note that the intersection only returns a *`product_id`* if it appears in both tables and returns exactly one instance of such a *`product_id`*. Thus, only a list of distinct product IDs is returned by the query.

To count the number of products that were ordered prior to 2013 but not after that point in time, the `EXCEPT` operator can be used to exclude the product IDs in the *`orders_recent`* table from the product IDs in the *`orders_arch`* table, as shown in the following SQL query.

```
SELECT COUNT(e.*)
FROM   (SELECT product_id
        FROM   orders_arch
        EXCEPT
        SELECT product_id
        FROM   orders_recent) e

13569
```

The preceding query uses the `COUNT` aggregate function to determine the number of returned rows from a second SQL query that includes the `EXCEPT` operator. This SQL query within a query is sometimes

called a ***subquery*** or a ***nested query***. Subqueries enable the construction of fairly complex queries without having to first execute the pieces, dump the rows to temporary tables, and then execute another SQL query to process those temporary tables. Subqueries can be used in place of a table within the `FROM` clause or can be used in the `WHERE` clause.

11.1.3 Grouping Extensions

Previously, the `COUNT()` aggregate function was used to count the number of returned rows from a query. Such aggregate functions often summarize a dataset after applying some grouping operation to it. For example, it may be desired to know the revenue by year or shipments per week. The following SQL query uses the `SUM()` aggregate function along with the `GROUP BY` operator to provide the top three ordered items based on *`item_quantity`*.

```
SELECT i.product_id,
       SUM(i.item_quantity) AS total
FROM   orders_recent i
GROUP  BY i.product_id
ORDER  BY SUM(i.item_quantity) DESC
LIMIT 3
```

```
product_id    total
15072         6089
15066         6082
15060         6053
```

`GROUP BY` can use the `ROLLUP()` operator to calculate subtotals and grand totals. The following SQL query employs the previous query as a subquery in the `WHERE` clause to supply the number of items ordered by year for the top three items ordered overall. The `ROLLUP` operator provides the subtotals, which match the previous output for each *`product_id`*, as well as the grand total.

```
SELECT r.product_id,
       DATE_PART('year', r.order_datetime) AS year,
       SUM(r.item_quantity)                AS total
FROM   orders_recent r
WHERE  r.product_id IN (SELECT o.product_id
                        FROM   orders_recent o
                        GROUP  BY o.product_id
                        ORDER  BY SUM(o.item_quantity) DESC
                        LIMIT 3)
GROUP  BY ROLLUP( r.product_id, DATE_PART('year', r.order_datetime) )
ORDER  BY r.product_id,
          DATE_PART('year', r.order_datetime)
```

```
product_id  year    total
15060       2013    5996
15060       2014    57
15060               6053
15066       2013    6030
```

```
15066       2014    52
15066               6082
15072       2013    6023
15072       2014    66
15072               6089
                    18224
```

The CUBE operator expands on the functionality of the ROLLUP operator by providing subtotals for each column specified in the CUBE statement. Modifying the prior query by replacing the ROLLUP operator with the CUBE operator results in the same output with the addition of the subtotals for each year.

```
SELECT r.product_id,
       DATE_PART('year', r.order_datetime) AS year,
       SUM(r.item_quantity)                AS total
FROM   orders_recent r
WHERE  r.product_id IN (SELECT o.product_id
                        FROM   orders_recent o
                        GROUP  BY o.product_id
                        ORDER  BY SUM(o.item_quantity) DESC
                        LIMIT 3)
GROUP  BY CUBE( r.product_id, DATE_PART('year', r.order_datetime) )
ORDER  BY r.product_id,
          DATE_PART('year', r.order_datetime
```

```
product_id  year    total
15060       2013    5996
15060       2014    57
15060               6053
15066       2013    6030
15066       2014    52
15066               6082
15072       2013    6023
15072       2014    66
15072               6089
            2013    18049          ← additional row
            2014    175            ← additional row
                    18224
```

Because null values in the output indicate the subtotal and grand total rows, care must be taken when null values appear in the columns being grouped. For example, null values may be part of the dataset being analyzed. The GROUPING() function can identify which rows with null values are used for the subtotals or grand totals.

```
SELECT r.product_id,
       DATE_PART('year', r.order_datetime)           AS year,
       SUM(r.item_quantity)                          AS total,
       GROUPING(r.product_id)                        AS group_id,
       GROUPING(DATE_PART('year', r.order_datetime)) AS group_year
FROM   orders_recent r
```

```
WHERE  r.product_id IN (SELECT o.product_id
                        FROM   orders_recent o
                        GROUP  BY o.product_id
                        ORDER  BY SUM(o.item_quantity) DESC
                        LIMIT 3)
GROUP  BY CUBE( r.product_id, DATE_PART('year', r.order_datetime) )
ORDER  BY r.product_id,
          DATE_PART('year', r.order_datetime)
```

product_id	year	total	group_id	group_year
15060	2013	5996	0	0
15060	2014	57	0	0
15060		6053	0	1
15066	2013	6030	0	0
15066	2014	52	0	0
15066		6082	0	1
15072	2013	6023	0	0
15072	2014	66	0	0
15072		6089	0	1
	2013	18049	1	0
	2014	175	1	0
		18224	1	1

In the preceding query, *group_year* is set to 1 when a total is calculated across the values of *year*. Similarly, *group_id* is set to 1 when a total is calculated across the values of *product_id*.

The functionality of ROLLUP and CUBE can be customized via GROUPING SETS. The SQL query using the CUBE operator can be replaced with the following query that employs GROUPING SETS to provide the same results.

```
SELECT r.product_id,
       DATE_PART('year', r.order_datetime) AS year,
       SUM(r.item_quantity)                AS total
FROM   orders_recent r
WHERE  r.product_id IN (SELECT o.product_id
                        FROM   orders_recent o
                        GROUP  BY o.product_id
                        ORDER  BY SUM(o.item_quantity) DESC
                        LIMIT 3)
GROUP  BY GROUPING SETS( ( r.product_id,
                           DATE_PART('year', r.order_datetime) ),
                         ( r.product_id ),
                         ( DATE_PART('year', r.order_datetime) ),
                         ( ) )
ORDER  BY r.product_id,
          DATE_PART('year', r.order_datetime)
```

The listed grouping sets define the columns for which subtotals will be provided. The last grouping set, `( )`, specifies that the overall total is supplied in the query results. For example, if only the grand total was desired, the following SQL query using `GROUPING SETS` could be used.

```
SELECT r.product_id,
       DATE_PART('year', r.order_datetime) AS year,
       SUM(r.item_quantity)                AS total
FROM   orders_recent r
WHERE  r.product_id IN (SELECT o.product_id
                        FROM   orders_recent o
                        GROUP  BY o.product_id
                        ORDER  BY SUM(o.item_quantity) DESC
                        LIMIT 3)
GROUP  BY GROUPING SETS( ( r.product_id,
                           DATE_PART('year', r.order_datetime) ),
                         ( ) )
ORDER  BY r.product_id,
          DATE_PART('year', r.order_datetime)
```

```
product_id   year   total
15060        2013   5996
15060        2014   57
15066        2013   6030
15066        2014   52
15072        2013   6023
15072        2014   66
                    18224
```

Because the `GROUP BY` clause can contain multiple `CUBE`, `ROLLUP`, or column specifications, duplicate grouping sets might occur. The `GROUP_ID()` function identifies the unique rows with a 0 and the redundant rows with a 1, 2, …. To illustrate the function `GROUP_ID()`, both `ROLLUP` and `CUBE` are used when only one specific ***product_id*** is being examined.

```
SELECT r.product_id,
       DATE_PART('year', r.order_datetime) AS year,
       SUM(r.item_quantity)                AS total,
       GROUP_ID()                          AS group_id
FROM   orders_recent r
WHERE  r.product_id IN ( 15060 )
GROUP  BY ROLLUP( r.product_id, DATE_PART('year', r.order_datetime) ),
          CUBE( r.product_id, DATE_PART('year', r.order_datetime) )
ORDER  BY r.product_id,
          DATE_PART('year', r.order_datetime),
          GROUP_ID()
```

```
product_id  year    total   group_id
15060       2013    5996    0
15060       2013    5996    1
15060       2013    5996    3
15060       2013    5996    4
15060       2013    5996    5
15060       2013    5996    6
15060       2014    57      0
15060       2014    57      1
15060       2014    57      2
15060       2014    57      3
15060       2014    57      4
15060       2014    57      5
15060       2014    57      6
15060               6053    0
15060               6053    1
15060               6053    2
            2013    5996    0
            2014    57      0
                    6053    0
```

Filtering on the *group_id* values equal to zero yields unique records. This filtering can be accomplished with the HAVING clause, as illustrated in the next SQL query.

```
SELECT r.product_id,
       DATE_PART('year', r.order_datetime) AS year,
       SUM(r.item_quantity)                AS total,
       GROUP_ID()                          AS group_id
FROM   orders_recent r
WHERE  r.product_id IN ( 15060 )
GROUP  BY ROLLUP( r.product_id, DATE_PART('year', r.order_datetime) ),
          CUBE( r.product_id, DATE_PART('year', r.order_datetime) )
HAVING GROUP_ID() = 0
ORDER  BY r.product_id,
          DATE_PART('year', r.order_datetime),
          GROUP_ID()
```

```
product_id   year   total   group_id
15060        2013   5996    0
15060        2014   57      0
15060               6053    0
             2013   5996    0
             2014   57      0
                    6053    0
```

11.2 In-Database Text Analysis

SQL offers several basic text string functions as well as wildcard search functionality. Related SELECT statements and their results enclosed in the SQL comment delimiters, /**/, include the following:

```
SELECT SUBSTRING('1234567890', 3,2)   /* returns '34'  */
SELECT '1234567890' LIKE '%7%'        /* returns True  */
SELECT '1234567890' LIKE '7%'         /* returns False */
SELECT '1234567890' LIKE '_2%'        /* returns True  */
SELECT '1234567890' LIKE '_3%'        /* returns False */
SELECT '1234567890' LIKE '__3%'       /* returns True  */
```

This section examines more dynamic and flexible tools for text analysis, called ***regular expressions,*** and their use in SQL queries to perform pattern matching. Table 11-1 includes several forms of the comparison operator used with regular expressions and related SQL examples that produce a `True` result.

TABLE 11-1 *Regular Expression Operators*

Operator	Description	Example
~	Contains the regular expression (case sensitive)	`'123a567' ~ 'a'`
~*	Contains the regular expression (case insensitive)	`'123a567' ~* 'A'`
!~	Does not contain the regular expression (case sensitive)	`'123a567' !~ 'A'`
!~*	Does not contain the regular expression (case insensitive)	`'123a567' !~* 'b'`

More complex forms of the patterns that are specified at the RHS of the comparison operator can be constructed by using the elements in Table 11-2.

TABLE 11-2 *Regular Expression Elements*

Element	Description
|	Matches item a or b (a|b)
^	Looks for matches at the beginning of the string
$	Looks for matches at the end of the string
.	Matches any single character
*	Matches preceding item zero or more times
+	Matches preceding item one or more times
?	Makes the preceding item optional
{n}	Matches the preceding item exactly n times

(*continues*)

TABLE 11-2 *Regular Expression Elements (Continued)*

Element	Description
( )	Matches the contents exactly
[ ]	Matches any of the characters in the content, such as [0–9]
\\x	Matches a nonalphanumeric character named x
\\y	Matches an escape string \y

To illustrate the use of these elements, the following `SELECT` statements include examples in which the comparisons are `True` or `False`.

```
/* matches x or y ('x|y')*/
SELECT  '123a567' ~ '23|b'       /* returns True  */
SELECT  '123a567' ~ '32|b'       /* returns False */

/* matches the beginning of the string  */
SELECT '123a567' ~ '^123a'       /* returns True  */
SELECT '123a567' ~ '^123a7'      /* returns False */

/* matches the end of the string  */
SELECT '123a567' ~ 'a567$'       /* returns True  */
SELECT '123a567' ~ '27$'         /* returns False */

/* matches any single character */
SELECT '123a567' ~ '2.a'         /* returns True  */
SELECT '123a567' ~ '2..5'        /* returns True  */
SELECT '123a567' ~ '2...5'       /* returns False */

/* matches preceding character zero or more times */
SELECT '123a567' ~ '2*'          /* returns True  */
SELECT '123a567' ~ '2*a'         /* returns True  */
SELECT '123a567' ~ '7*a'         /* returns True  */
SELECT '123a567' ~ '37*'         /* returns True  */
SELECT '123a567' ~ '87*'         /* returns False */

/* matches preceding character one or more times */
SELECT '123a567' ~ '2+'          /* returns True  */
SELECT '123a567' ~ '2+a'         /* returns False */
SELECT '123a567' ~ '7+a'         /* returns False */
SELECT '123a567' ~ '37+'         /* returns False */
SELECT '123a567' ~ '87+'         /* returns False */

/* makes the preceding character optional */
SELECT '123a567' ~ '2?'          /* returns True  */
```

```
SELECT '123a567' ~ '2?a'         /* returns True  */
SELECT '123a567' ~ '7?a'         /* returns True  */
SELECT '123a567' ~ '37?'         /* returns True  */
SELECT '123a567' ~ '87?'         /* returns False */

/* Matches the preceding item exactly {n} times */

SELECT '123a567' ~ '5{0}'        /* returns True  */
SELECT '123a567' ~ '5{1}'        /* returns True  */
SELECT '123a567' ~ '5{2}'        /* returns False */
SELECT '1235567' ~ '5{2}'        /* returns True  */
SELECT '123a567' ~ '8{0}'        /* returns True  */
SELECT '123a567' ~ '8{1}'        /* returns False */

/* Matches the contents exactly */
SELECT '123a567' ~ '(23a5)'      /* returns True  */
SELECT '123a567' ~ '(13a5)'      /* returns False */
SELECT '123a567' ~ '(23a5)7*'    /* returns True  */
SELECT '123a567' ~ '(23a5)7+'    /* returns False */

/* Matches any of the contents */
SELECT '123a567' ~ '[23a8]'      /* returns True  */
SELECT '123a567' ~ '[8a32]'      /* returns True  */
SELECT '123a567' ~ '[(13a5)]'    /* returns True  */
SELECT '123a567' ~ '[xyz9]'      /* returns False */
SELECT '123a567' ~ '[a-z]'       /* returns True  */
SELECT '123a567' ~ '[b-z]'       /* returns False */

/* Matches a nonalphanumeric */
SELECT '$50K+' ~ '\\$'           /* returns True  */
SELECT '$50K+' ~ '\\+'           /* returns True  */
SELECT '$50K+' ~ '\\$\\+'        /* returns False */

/* Use of the backslash for escape clauses                        */
/* \\w denotes the characters 0-9, a-z, A-Z, or the underscore(_) */
SELECT '123a567'  ~ '\\w'        /* returns True  */
SELECT '123a567+' ~ '\\w'        /* returns True  */
SELECT '++++++++' ~ '\\w'        /* returns False */
SELECT '_' ~ '\\w'               /* returns True  */
SELECT '+' ~ '\\w'               /* returns False */
```

Regular expressions can be developed to identify mailing addresses, e-mail addresses, phone numbers, or currency amounts.

```
/* use of more complex regular expressions */
SELECT '$50K+' ~ '\\$[0-9]*K\\+'          /* returns True  */
SELECT '$50K+' ~ '\\$[0-9]K\\+'           /* returns False */
SELECT '$50M+' ~ '\\$[0-9]*K\\+'          /* returns False */
```

```
SELECT '$50M+' ~ '\\$[0-9]*(K|M)\\+'       /* returns True  */

/* check for ZIP code of form #####-#### */
SELECT '02038-2531' ~ '[0-9]{5}-[0-9]{4}'  /* returns True  */
SELECT '02038-253'  ~ '[0-9]{5}-[0-9]{4}'  /* returns False */
SELECT '02038'      ~ '[0-9]{5}-[0-9]{4}'  /* returns False */
```

So far, the application of regular expressions has been illustrated by including the Boolean comparison in a `SELECT` statement as if the result of the comparison was to be returned as a column. In practice, these comparisons are used in a `SELECT` statement's `WHERE` clause against a table column to identify specific records of interest. For example, the following SQL query identifies those ZIP codes in a table of customer addresses that do not match the form #####-####. Once the invalid ZIP codes are identified, corrections can be made by manual or automated means.

```
SELECT address_id,
       customer_id,
       city,
       state,
       zip,
       country
FROM   customer_addresses
WHERE  zip !~ '^[0-9]{5}-[0-9]{4}$'
```

```
address_id  customer_id   city          state    zip           country
7           13            SINAI         SD       57061-o236    USA
18          27            SHELL ROCK    IA       S0670-0480    USA
24          37            NASHVILLE     TN       37228-219     USA
.
.
.
```

SQL functions enable the use of regular expressions to extract the matching text, such as `SUBSTRING()`, as well as update the text, such as `REGEXP_REPL()`.

```
/* extract ZIP code from text string */
SELECT SUBSTRING('4321A Main Street Franklin, MA 02038-2531'
FROM '[0-9]{5}-[0-9]{4}')

02038-2531

/* replace long format zip code with short format ZIP code */
SELECT REGEXP_REPLACE('4321A Main Street Franklin, MA 02038-2531',
       '[0-9]{5}-[0-9]{4}',
       SUBSTRING(SUBSTRING('4321A Main Street Franklin, MA 02038-2531'
FROM '[0-9]{5}-[0-9]{4}'),1,5)    )

4321A Main Street Franklin, MA 02038
```

Regular expressions provide considerable flexibility in searching and modifying text strings. However, it is quite easy to build a regular expression that does not work entirely as intended. For example, a particular operation may work properly with a given dataset, but future datasets may contain new cases to be handled. Thus, it is important to thoroughly test any SQL code using regular expressions.

11.3 Advanced SQL

Building upon the foundation provided in the earlier parts of this chapter, this section presents advanced SQL techniques that can simplify in-database analytics.

11.3.1 Window Functions

In Section 11.1.3, several SQL examples using aggregate functions and grouping options to summarize a dataset were provided. A ***window function*** enables aggregation to occur but still provides the entire dataset with the summary results. For example, the `RANK()` function can be used to order a set of rows based on some attribute. Based on the SQL table, *`orders_recent`*, introduced in Section 11.1.2, the following SQL query provides a ranking of customers based on their total expenditures.

```
SELECT s.customer_id,
       s.sales,
       RANK()
         OVER (
           ORDER BY s.sales DESC ) AS sales_rank
FROM   (SELECT r.customer_id,
               SUM(r.item_quantity * r.item_price) AS sales
        FROM   orders_recent r
        GROUP  BY r.customer_id) s
```

```
customer_id   sales       sales_rank
683377        27840.00    1
238107        19983.65    2
661519        18134.11    3
628278        17965.44    4
619660        17944.20    5
.
.
.
```

The subquery in the `FROM` clause computes the total sales for each customer. In the outermost `SELECT` clause, the sales are ranked in descending order. Window functions, such as `RANK()`, are followed by an `OVER` clause that specifies how the function should be applied. Additionally, the window function can be applied to groupings of a given dataset using the `PARTITION BY` clause. The following SQL query provides the customer rankings based on sales within product categories.

```
SELECT s.category_name,
       s.customer_id,
```

```
       s.sales,
       RANK()
         OVER (
           PARTITION BY s.category_name
           ORDER BY s.sales DESC ) AS sales_rank
FROM   (SELECT c.category_name,
               r.customer_id,
               SUM(r.item_quantity * r.item_price) AS sales
        FROM   orders_recent r
               LEFT OUTER JOIN product p
                            ON r.product_id = p.product_id
               LEFT OUTER JOIN category c
                            ON p.category_id = c.category_id
        GROUP  BY c.category_name,
                  r.customer_id) s
ORDER  BY s.category_name,
          sales_rank
```

```
category_name                       customer_id sales       sales_rank
Apparel                             596396      4899.93     1
Apparel                             319036      2799.96     2
Apparel                             455683      2799.96     2
Apparel                             468209      2700.00     4
Apparel                             456107      2118.00     5
.
.
.
Apparel                             430126      2.20        78731
Automotive Parts and Accessories    362572      5706.48     1
Automotive Parts and Accessories    587564      5109.12     2
Automotive Parts and Accessories    377616      4279.86     3
Automotive Parts and Accessories    443618      4279.86     3
Automotive Parts and Accessories    590658      3668.55     5
.
.
.
```

In this case, the subquery determines each customer's sales in the respective product category. The outer `SELECT` clause then ranks the customer's sales within each category. The provided portions of the SQL query output illustrate that the ranking begins at 1 for each category and demonstrate how the rankings are affected by ties in the amount of *`sales`*.

A second use of windowing functions is to perform calculations over a sliding window in time. For example, moving averages can be used to smooth weekly sales figures that may exhibit large week-to-week variation, as shown in the plot in Figure 11-2.

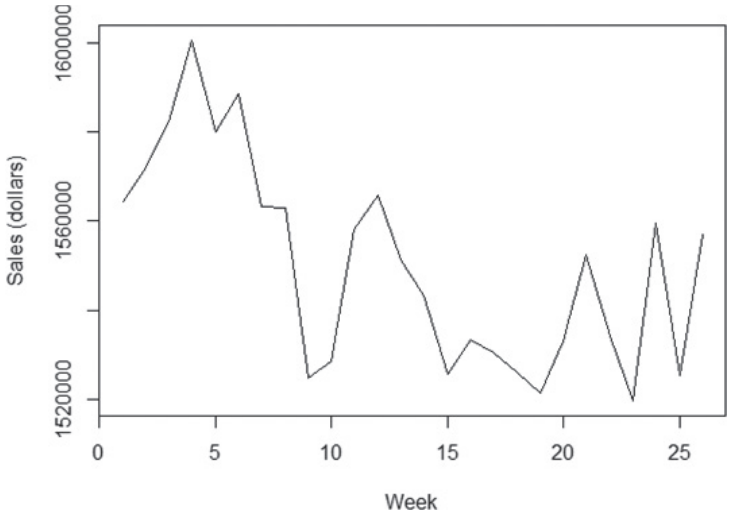

FIGURE 11-2 *Weekly sales for an online retailer*

The following SQL query illustrates how moving averages can be implemented using window functions:

```
SELECT year,
       week,
       sales,
       AVG(sales)
         OVER (
           ORDER BY year, week
           ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING) AS moving_avg
FROM   sales_by_week
WHERE  year = 2014
       AND week <= 26
ORDER  BY year,
          week
```

```
year    week    sales       moving_avg
2014    1       1564539     1572999.333 ←average of weeks 1, 2, 3
2014    2       1572128     1579941.75  ←average of weeks 1, 2, 3, 4
2014    3       1582331     1579982.6   ←average of weeks 1, 2, 3, 4, 5
2014    4       1600769     1584834.4   ←average of weeks 2, 3, 4, 5, 6
2014    5       1580146     1583037.2   ←average of weeks 3, 4, 5, 6, 7
2014    6       1588798     1579179.6
2014    7       1563142     1563975.6
2014    8       1563043     1553665
2014    9       1524749     1547534.8
```

```
2014    10      1528593     1548051.6
2014    11      1558147     1545714.2
2014    12      1565726     1549404
2014    13      1551356     1548812.6
2014    14      1543198     1543820.2
2014    15      1525636     1536767.6
2014    16      1533185     1531662.2
2014    17      1530463     1527313.6
2014    18      1525829     1528787.8
2014    19      1521455     1532649
2014    20      1533007     1533370
2014    21      1552491     1532116
2014    22      1534068     1539713.6
2014    23      1519559     1538199.6
2014    24      1559443     1539086.2  ←average of weeks 22,23,24,25,26
2014    25      1525437     1540340.75 ←average of weeks 23,24,25,26
2014    26      1556924     1547268    ←average of weeks 24,25,26
```

The windowing function uses the built-in aggregate function `AVG()`, which computes the arithmetic average of a set of values. The `ORDER BY` clause sorts the records in chronological order and specifies which rows should be included in the averaging process with the current row. In this SQL query, the moving average is based on the current row, the preceding two rows, and the following two rows. Because the dataset does not include the last two weeks of 2013, the first moving average value of 1,572,999.333 is the average of the first three weeks of 2014: the current week and the two subsequent weeks. The moving average value for the second week, 1,579,941.75, is the sales value for week 2 averaged with the prior week and the two subsequent weeks. For weeks 3 through 24, the moving average is based on the sales from 5-week periods, centered on the current week. At week 25, the window begins to include fewer weeks because the following weeks are unavailable. Figure 11-3 illustrates the applied smoothing process against the weekly sales figures.

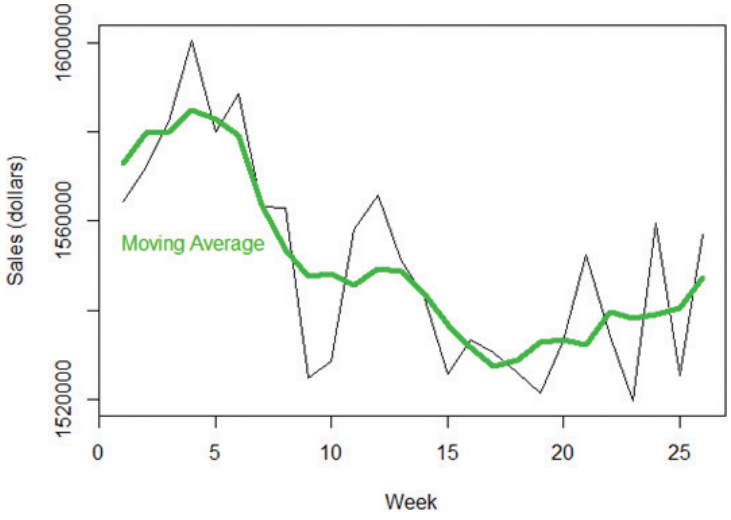

FIGURE 11-3 *Weekly sales with moving averages*

Built-in window functions may vary by SQL implementation. Table 11-3 [1] from the PostgreSQL documentation includes the list of general-purpose window functions.

TABLE 11-3 *Window Functions*

Function	Description
`row_number()`	Number of the current row within its partition, counting from 1.
`rank()`	Rank of the current row with gaps; same as `row_number` of its first peer.
`dense_rank()`	Rank of the current row without gaps; this function counts peer groups.
`percent_rank()`	Relative rank of the current row: (rank – 1) / (total rows – 1).
`cume_dist()`	Relative rank of the current row: (number of rows preceding or peer with current row) / (total rows).
`ntile(num_buckets integer)`	Integer ranging from 1 to the argument value, dividing the partition as equally as possible.
`lag(value any [, offset integer [, default any ]])`	Returns the value evaluated at the row that is offset rows before the current row within the partition; if there is no such row, instead return default. Both offset and default are evaluated with respect to the current row. If omitted, offset defaults to 1 and default to `null`.
`lead(value any [, offset integer [, default any ]])`	Returns the value evaluated at the row that is offset rows after the current row within the partition; if there is no such row, instead return default. Both offset and default are evaluated with respect to the current row. If omitted, the offset defaults to 1 and the default to `null`.
`first_value(value any)`	Returns the value evaluated at the first row of the window frame.
`last_value(value any)`	Returns the value evaluated at the last row of the window frame.
`nth_value(value any, nth integer)`	Returns the value evaluated at the nth row of the window frame (counting from 1); `null` if no such row.

`http://www.postgresql.org/docs/9.3/static/functions-window.html`

11.3.2 User-Defined Functions and Aggregates

When the built-in SQL functions are insufficient for a particular task or analysis, SQL enables the user to create user-defined functions and aggregates. This custom functionality can be incorporated into SQL queries in the same ways that the built-in functions and aggregates are used. User-defined functions can also be created to simplify processing tasks that a user may commonly encounter.

For example, a user-defined function can be written to translate text strings for female (F) and male (M) to 0 and 1, respectively. Such a function may be helpful when formatting data for use in a regression analysis. Such a function, `fm_convert()`, could be implemented as follows:

```
CREATE FUNCTION fm_convert(text) RETURNS integer AS
'SELECT CASE
         WHEN $1 = ''F'' THEN 0
         WHEN $1 = ''M'' THEN 1
         ELSE NULL
        END'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT
```

In declaring the function, the SQL query is placed within single quotes. The first and only passed value is referenced by *$1*. The SQL query is followed by a `LANGUAGE` statement that explicitly states that the preceding statement is written in SQL. Another option is to write the code in C. `IMMUTABLE` indicates that the function does not update the database and does not use the database for lookups. The `IMMUTABLE` declaration informs the database's query optimizer how best to implement the function. The `RETURNS NULL ON NULL INPUT` statement specifies how the function addresses the case when any of the inputs are `null` values.

In the online retail example, the `fm_convert()` function can be applied to the *customer_gender* column in the *customer_demographics* table as follows.

```
SELECT customer_gender,
       fm_convert(customer_gender) as male
FROM   customer_demographics
LIMIT 5
```

```
customer_gender     male
M                   1
F                   0
F                   0
M                   1
M                   1
```

Built-in and user-defined functions can be incorporated into user-defined aggregates, which can then be used as a window function. In Section 11.3.1, a window function is used to calculate moving averages to smooth a data series. In this section, a user-defined aggregate is created to calculate an ***Exponentially Weighted Moving Average (EWMA)***. For a given time series, the EWMA series is defined as shown in Equation 11-1.

$$EWMA_t = \begin{cases} y_t & for\ t = 1 \\ \alpha . y_t + (1-\alpha) . EWMA_{t-1} & for\ t \geq 2 \end{cases} \quad (11\text{-}1)$$

where $0 \leq \alpha \leq 1$

The smoothing factor, determines how much weight to place on the latest point in a given time series. By repeatedly substituting into Equation 11-1 for the prior value of the EWMA series, it can be shown that the weights against the original series are exponentially decaying backward in time.

To implement EWMA smoothing as a user-defined aggregate in SQL, the functionality in Equation 11-1 needs to be implemented first as a user-defined function.

```
CREATE FUNCTION ewma_calc(numeric, numeric, numeric) RETURNS numeric as
/* $1 = prior value of EWMA         */
/* $2 = current value of series     */
/* $3 = alpha, the smoothing factor */
'SELECT CASE
         WHEN $3 IS NULL                      /* bad alpha */
         OR $3 < 0
         OR $3 > 1 THEN NULL
         WHEN $1 IS NULL THEN $2              /* t = 1        */
         WHEN $2 IS NULL THEN $1              /* y is unknown */
         ELSE ($3 * $2) + (1-$3) *$1          /* t >= 2       */
        END'
LANGUAGE SQL
IMMUTABLE
```

Accepting three numeric inputs as defined in the comments, the `ewma_calc()` function addresses possible bad values of the smoothing factor as well as the special case in which the other inputs are `null`. The `ELSE` statement performs the usual EWMA calculation. Once this function is created, it can be referenced in the user-defined aggregate, `ewma()`.

```
CREATE AGGREGATE ewma(numeric, numeric)
       (SFUNC = ewma_calc,
        STYPE = numeric,
        PREFUNC = dummy_function)
```

In the `CREATE AGGREGATE` statement for `ewma()`, `SFUNC` assigns the state transition function (`ewma_calc` in this example) and `STYPE` assigns the data type of the variable to store the current state of the aggregate. The variable for the current state is made available to the `ewma_calc()` function as the first variable, *`$1`*. In this case, because the `ewma_calc()` function requires three inputs, the `ewma()` aggregate requires only two inputs; the state variable is always internally available to the aggregate. The `PREFUNC` assignment is required in the Greenplum database for use in a massively parallel processing (MPP) environment. For some aggregates, it is necessary to perform some preliminary functionality on the current state variables for a couple of servers in the MPP environment. In this example, the assigned `PREFUNC` function is added as a placeholder and is not utilized in the proper execution of the `ewma()` aggregate function.

As a window function, the `ewma()` aggregate, with a smoothing factor of 0.1, can be applied to the weekly sales data as follows.

```
SELECT year,
       week,
       sales,
       ewma(sales, .1)
         OVER (
           ORDER BY year, week)
FROM   sales_by_week
WHERE  year = 2014
```

```
       AND week <= 26
ORDER  BY year,
          week

year    week    sales       ewma
2014    1       1564539     1564539.00
2014    2       1572128     1565297.90
2014    3       1582331     1567001.21
2014    4       1600769     1570377.99
2014    5       1580146     1571354.79
.
.
.
2014    23      1519559     1542043.47
2014    24      1559443     1543783.42
2014    25      1525437     1541948.78
2014    26      1556924     1543446.30
```

Figure 11-4 includes the EWMA smoothed series to the plot from Figure 11-3.

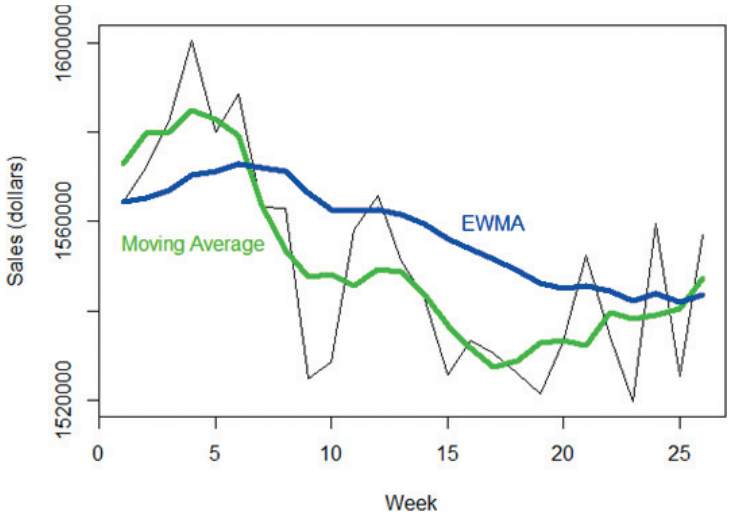

FIGURE 11-4 *Weekly sales with moving average and EWMA*

Increasing the value of the smoothing factor from 0.1 causes the EWMA to follow the actual data better, but the trade-off is that large fluctuations in the data cause larger fluctuations in the smoothed series. The user-defined aggregate, `ewma()`, is used in the SQL query in the same manner as any other window function with the specification of the `OVER` clause.

11.3.3 Ordered Aggregates

Sometimes the value of an aggregate may depend on an ordered set of values. For example, to determine the median of a set of values, it is common to first sort the values from smallest to largest and identify the median from the center of the sorted values. The sorting can be accomplished by using the function `array_agg()`. The following SQL query calculates the median of the weekly sales data.

```
SELECT (d.ord_sales[ d.n/2 + 1 ] +
                    d.ord_sales[ (d.n + 1)/2 ]) / 2.0 as median
FROM   (SELECT ARRAY_AGG(s.sales ORDER BY s.sales) AS ord_sales,
               COUNT(*)     AS n
        FROM   sales_by_week s
        WHERE  s.year = 2014
               AND s.week <= 26) d
```

```
median
1551923.5
```

In general, the function `ARRAY_AGG()` builds an array from a table column. Executing the subquery from the previous SQL query for just the first five weeks illustrates the creation of the array, denoted by the braces, and the sorted weekly sales within the array.

```
SELECT ARRAY_AGG(s.sales ORDER BY s.sales) AS ord_sales,
       COUNT(*) AS n
FROM   sales_by_week s
WHERE  s.year = 2014
       AND s.week <= 5
```

```
ord_sales                                        n
{1564539,1572128,1580146,1582331,1600769}        5
```

Besides creating an array, the values can be concatenated together into one text string using the `string_agg()` function.

```
SELECT STRING_AGG(s.sales ORDER BY s.sales) AS ord_sales,
       COUNT(*) AS n
FROM   sales_by_week s
WHERE  s.year = 2014
```

```
       AND s.week <= 5

ord_sales                                          n
1564539157212815801461582331160076 9               5
```

However, in this particular example, it may be useful to separate the values with a delimiter, such as a comma.

```
SELECT STRING_AGG(s.sales, ',' ORDER BY s.sales) AS ord_sales,
       COUNT(*) AS n
FROM   sales_by_week s
WHERE  s.year = 2014
       AND s.week <= 5

ord_sales                                          n
1564539,1572128,1580146,1582331,1600769            5
```

Although the sorted sales appear to be an array, there are no braces around the output. So the displayed ordered sales are a text string.

11.3.4 MADlib

SQL implementations include many basic analytical and statistical built-in functions, such as means and variances. As illustrated in this chapter, SQL also enables the development of user-defined functions and aggregates to provide additional functionality. Furthermore, SQL databases can utilize an external library of functions. One such library is known as ***MADlib***. The description file [2] included with the MADlib library download states the following:

> *MADlib is an open-source library for scalable in-database analytics. It offers data-parallel implementations of mathematical, statistical, and machine learning methods for structured and unstructured data.*

The concept of Magnetic/Agile/Deep (MAD) analysis skills was introduced in a 2009 paper by Cohen, et al. [3]. This paper describes the components of MAD as follows:

- **Magnetic:** Traditional Enterprise Data Warehouse (EDW) approaches "repel" new data sources, discouraging their incorporation until they are carefully cleansed and integrated. Given the ubiquity of data in modern organizations, a data warehouse can keep pace today only by being "magnetic": attracting all the data sources that crop up within an organization regardless of data quality niceties.
- **Agile:** Data Warehousing orthodoxy is based on long-range and careful design and planning. Given growing numbers of data sources and increasingly sophisticated and mission-critical data analyses, a modern warehouse must instead allow analysts to easily ingest, digest, produce, and adapt data rapidly. This requires a database whose physical and logical contents can be in continuous rapid evolution.
- **Deep:** Modern data analyses involve increasingly sophisticated statistical methods that go well beyond the rollups and drilldowns of traditional business intelligence (BI). Moreover, analysts often need to see both the forest and the trees in running these algorithms; they want to study enormous datasets without resorting to samples and extracts. The modern data warehouse should serve both as a deep data repository and as a sophisticated algorithmic runtime engine.

In response to the inability of a traditional EDW to readily accommodate new data sources, the concept of a ***data lake*** has emerged. A data lake represents an environment that collects and stores large volumes of structured and unstructured datasets, typically in their original, unaltered forms. More than a data depository, the data lake architecture enables the various users and data science teams to conduct data exploration and related analytical activities. Apache Hadoop is often considered a key component of building a data lake [4].

Because MADlib is designed and built to accommodate massive parallel processing of data, MADlib is ideal for Big Data in-database analytics. MADlib supports the open-source database PostgreSQL as well as the Pivotal Greenplum Database and Pivotal HAWQ. HAWQ is a SQL query engine for data stored in the Hadoop Distributed File System (HDFS). Apache Hadoop and the Pivotal products were described in Chapter 10, "Advanced Analytics—Technology and Tools: MapReduce and Hadoop."

MADlib version 1.6 modules [5] are described in Table 11-4.

Table 11-4 *MADlib Modules*

Module	Description
Generalized Linear Models	Includes linear regression, logistic regression, and multinomial logistic regression
Cross Validation	Evaluates the predictive power of a fitted model
Linear Systems	Solves dense and sparse linear system problems
Matrix Factorization	Performs low-rank matrix factorization and singular value decomposition
Association Rules	Implements the Apriori algorithm to identify frequent item sets
Clustering	Implements k-means clustering
Topic Modeling	Provides a Latent Dirichlet Allocation predictive model for a set of documents
Descriptive Statistics	Simplifies the computation of summary statistics and correlations
Inferential Statistics	Conducts hypothesis tests
Support Modules	Provides general array and probability functions that can also be used by other MADlib modules
Dimensionality Reduction	Enables principal component analyses and projections
Time Series Analysis	Conducts ARIMA analyses

`http://doc.madlib.net/latest/modules.html`

In the following example, MADlib is used to perform a k-means clustering analysis, as described in Chapter 4, "Advanced Analytical Theory and Methods: Clustering," on the web retailer's customers. Two

customer attributes—age and total sales since 2013—have been identified as variables of interest for the purposes of the clustering analysis. The customer's age is available in the *customer_demographics* table. The total sales for each customer can be computed from the *orders_recent* table. Because it was decided to include customers who had not purchased anything, a `LEFT OUTER JOIN` is used to include all customers. The customer's age and sales are stored in an array in the *cust_age_sales* table. The MADlib k-means function expects the coordinates to be expressed as an array.

```
/* create an empty table to store the input for the k-means analysis */
CREATE TABLE cust_age_sales (
  customer_id integer,
  coordinates float8[])

/* prepare the input for the k-means analysis */
INSERT INTO cust_age_sales (customer_id, coordinates[1], coordinates[2])
  SELECT d.customer_id,
         d.customer_age,
         CASE
           WHEN s.sales IS NULL THEN 0.0
           ELSE s.sales
         END
  FROM   customer_demographics d
         LEFT OUTER JOIN (SELECT r.customer_id,
                          SUM(r.item_quantity * r.item_price) AS sales
                         FROM   orders_recent r
                         GROUP  BY r.customer_id) s
         ON d.customer_id = s.customer_id

/* examine the first 10 rows of the input */
SELECT * from cust_age_sales
order by customer_id
LIMIT 10

customer_id     coordinates
1               {32,14.98}
2               {32,51.48}
3               {33,151.89}
4               {27,88.28}
5               {31,4.85}
6               {26,54}
7               {29,63}
8               {25,101.07}
9               {32,41.05}
10              {32,0}
```

Using the MADlib function, `kmeans_random()`, the following SQL query identifies six clusters within the provided dataset. A description of the key input values is provided with the query.

```
/*
K-means analysis

cust_age_sales - SQL table containing the input data
coordinates - the column in the SQL table that contains the data points
customer_id - the column in the SQL table that contains the
              identifier for each point
km_coord - the table to store each point and its assigned cluster
km_centers - the SQL table to store the centers of each cluster
l2norm - specifies that the Euclidean distance formula is used
25 - the maximum number of iterations
0.001 - a convergence criterion
False(twice) - ignore some options
6 - build six clusters
*/

SELECT madlib.kmeans_random('cust_age_sales', 'coordinates',
               'customer_id', 'km_coord', 'km_centers',
               'l2norm', 25 ,0.001, False, False, 6)


SELECT *
FROM   km_coord
ORDER  BY pid
LIMIT 10

pid     coords              cid
1       {1,1}:{32,14.98}    6
2       {1,1}:{32,51.48}    1
3       {1,1}:{33,151.89}   4
4       {1,1}:{27,88.28}    1
5       {1,1}:{31,4.85}     6
6       {1,1}:{26,54}       1
7       {1,1}:{29,63}       1
8       {1,1}:{25,101.07}   1
9       {1,1}:{32,41.05}    1
10      {1,1}:{32,0}        6
```

The output consists of the *km_coord* table. This table contains the coordinates for each point id (*pid*), the *customer_id*, and the assigned cluster ID (*cid*). The coordinates (*coords*) are stored as sparse vectors. Sparse vectors are useful when values in an array are repeated many times. For example, {1,200,3}:{1,0,1} represents the following vector containing 204 elements, {1,0,0,…0,1,1,1}, where the zeroes are repeated 200 times.

The coordinates for each cluster center or centroid are stored in the SQL table *km_center*.

```
SELECT *
FROM   km_centers
```

```
ORDER  BY coords

cid coords
6   {1,1}:{44.1131730722154,6.31487804161302}
1   {1,1}:{39.8000419034649,61.6213603286732}
4   {1,1}:{39.2578830823738,167.758556117954}
5   {1,1}:{40.9437092852768,409.846906145043}
3   {1,1}:{42.3521947160391,1150.68858851676}
2   {1,1}:{41.2411873840445,4458.93716141001}
```

Because the age values are similar for each centroid, it appears that the sales values dominated the distance calculations. After visualizing the clusters, it is advisable to repeat the analysis after rescaling, as discussed in Chapter 4.

Summary

This chapter presented several techniques and examples illustrating how SQL can be used to perform in-database analytics. A typical SQL query involves joining several tables, filtering the returned dataset to the desired records with a `WHERE` clause, and specifying the particular columns of interest. SQL provides the set operations of `UNION` and `UNION ALL` to merge the results of two or more `SELECT` statements or `INTERSECT` to find common record elements. Other SQL queries can summarize a dataset using aggregate functions such as `COUNT()` and `SUM()` and the `GROUP BY` clause. Grouping extensions such as the `CUBE` and `ROLLUP` operators enable the computation of subtotals and grand totals.

Although SQL is most commonly associated with structured data, SQL tables often contain unstructured data such as comments, descriptions, and other freeform text content. Regular expressions and related functions can be used in SQL to examine and restructure such unstructured data for further analysis.

More complex SQL queries can utilize window functions to supply computed values such as ranks and rolling averages along with an original dataset. In addition to built-in functions, SQL offers the ability to create user-defined functions. Although it is possible to process the data within a database and extract the results into an analytical tool such as R, external libraries such as MADlib can be utilized by SQL to conduct statistical analyses within a database.

Exercises

1. Show that EWMA smoothing is equivalent to an ARIMA(0,1,1) model with no constant, as described in Chapter 8, "Advanced Analytical Theory and Methods: Time Series Analysis."
2. Referring to Equation (11-1), demonstrate that the assigned weights decay exponentially in time.
3. Develop and test a user-defined aggregate to calculate n factorial (n!), where n is an integer.
4. From a SQL table or query, randomly select 10% of the rows. Hint: Most SQL implementations have a `random()` function that provides a uniform random number between 0 and 1. Discuss possible reasons to randomly sample records from a SQL table.

Bibliography

[1] PostgreSQl.org, "Window Functions" [Online]. Available: `http://www.postgresql.org/docs/9.3/static/functions-window.html`. [Accessed 10 April 2014].

[2] MADlib, "MADlib" [Online]. Available: `http://madlib.net/download/`. [Accessed 10 April 2014].

[3] J. Cohen, B. Dolan, M. Dunlap, J. Hellerstein, and C. Welton, "MAD Skills: New Analysis Practices for Big Data," in *Proceedings of the VLDB Endowment Volume 2 Issue 2, August 2009*.

[4] E. Dumbill, "The Data Lake Dream," ***Forbes***, 14 January 2014. [Online]. Available: `http://www.forbes.com/sites/edddumbill/2014/01/14/the-data-lake-dream/`. [Accessed 4 June 2014].

[5] MADlib, "MADlib Modules" [Online]. Available: `http://doc.madlib.net/latest/modules.html`. [Accessed 10 April 2014].

12

The Endgame, or Putting It All Together

Key Concepts

Communicating and operationalizing an analytics project
Creating the final deliverables
Using a core set of material for different audiences
Comparing main focus areas for sponsors and analysts
Understanding simple data visualization principles
Cleaning up a chart or visualization

This chapter focuses on the final phase of the Data Analytics Lifecycle: operationalize. In this phase, the project team delivers final reports, code, and technical documentation. At the conclusion of this phase, the team generally attempts to set up a pilot project and implement the developed models from Phase 4 in a production environment. As stated in Chapter 2, "Data Analytics Lifecycle," teams can perform a technically accurate analysis, but if they cannot translate the results into a language that resonates with their audience, others will not see the value, and significant effort and resources will have been wasted. This chapter focuses on showing how to construct a clear narrative summary of the work and a framework for conveying the narrative to key stakeholders.

12.1 Communicating and Operationalizing an Analytics Project

As shown in Figure 12-1, the final phase in the Data Analytics Lifecycle focuses on operationalizing the project. In this phase, teams need to assess the benefits of the project work and set up a pilot to deploy the models in a controlled way before broadening the work and sharing it with a full enterprise or ecosystem of users. In this context, a pilot project can refer to a project prior to a full-scale rollout of the new algorithms or functionality. This pilot can be a project with a more limited scope and rollout to the lines of business, products, or services affected by these new models.

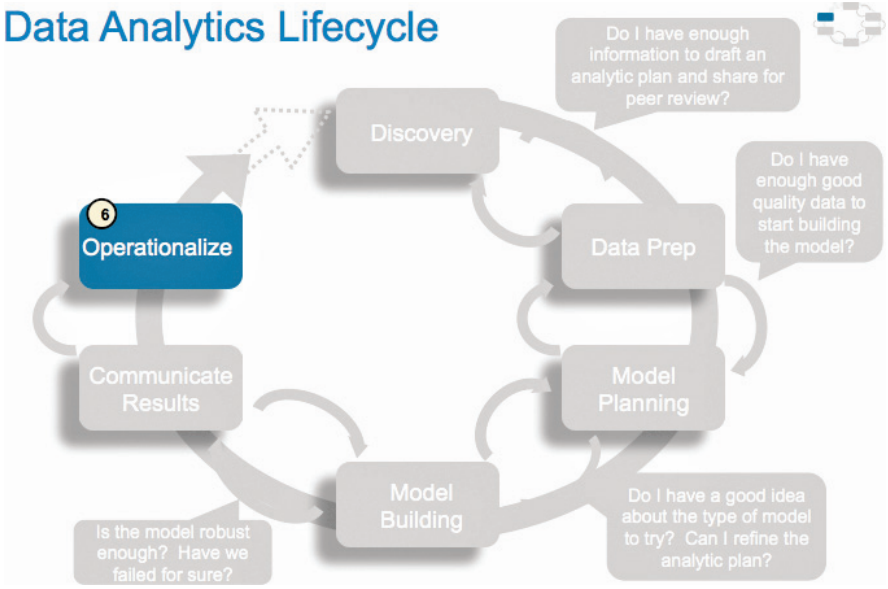

FIGURE 12-1 *Data Analytics Lifecycle, Phase 6: operationalize*

The team's ability to quantify the benefits and share them in a compelling way with the stakeholders will determine if the work will move forward into a pilot project and ultimately be run in a production environment. Therefore, it is critical to identify the benefits and state them in a clear way in the final presentations.

As the team scopes the effort involved to deploy the analytical model as a pilot project, it also needs to consider running the model in a production environment for a discrete set of products or a single line of business, which tests the model in a live setting. This allows the team to learn from the deployment and make adjustments before deploying the application or code more broadly across the enterprise. This phase can bring in a new set of team members—namely, those engineers responsible for the production environment who have a new set of issues and concerns. This group is interested in ensuring that running the model fits smoothly into the production environment and the model can be integrated into downstream processes. While executing the model in the production environment, the team should aim to detect input anomalies before they are fed to the model, assess run times, and gauge competition for resources with other processes in the production environment.

Chapter 2 included an in-depth discussion of the Data Analytics Lifecycle, including an overview of the deliverables provided in its final phase, at which time it is advisable for the team to consider the needs of each of its main stakeholders and the deliverables, illustrated in Figure 12-2, to satisfy these needs.

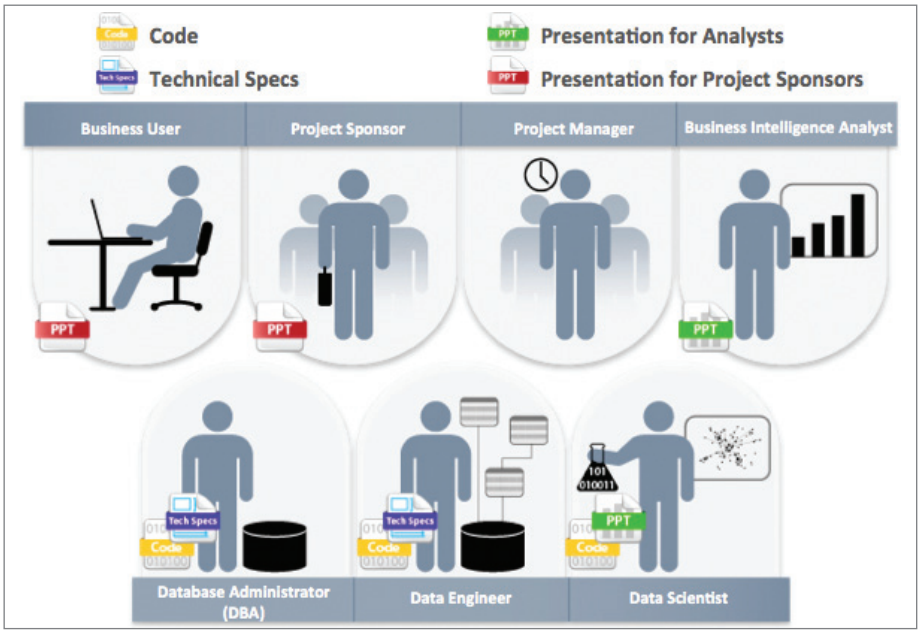

FIGURE 12-2 *Key outputs from a successful analytic project*

Following is a brief review of the key outputs for each of the main stakeholders of an analytics project and what they usually expect at the conclusion of a project:

- **Business User** typically tries to determine the benefits and implications of the findings to the business.
- **Project Sponsor** typically asks questions related to the business impact of the project, the risks and return on investment (ROI), and how the project can be evangelized within the organization and beyond.

- **Project Manager** needs to determine if the project was completed on time and within budget.
- **Business Intelligence Analyst** needs to know if the reports and dashboards he manages will be impacted and need to change.
- **Data Engineer *and* Database Administrator** (DBA) typically need to share the code from the analytical project and create technical documents that describe how to implement the code.
- **Data Scientists** need to share the code and explain the model to their peers, managers, and other stakeholders.

Although these seven roles represent many interests within a project, these interests usually overlap, and most of them can be met with four main deliverables:

- **Presentation for Project Sponsors** contains high-level takeaways for executive-level stakeholders, with a few key messages to aid their decision-making process. Focus on clean, easy visuals for the presenter to explain and for the viewer to grasp.
- **Presentation for Analysts**, which describes changes to business processes and reports. Data scientists reading this presentation are comfortable with technical graphs (such as Receiver Operating Characteristic [ROC] curves, density plots, and histograms) and will be interested in the details.
- **Code** for technical people, such as engineers and others managing the production environment
- **Technical specifications** for implementing the code

As a rule, the more executive the audience, the more succinct the presentation needs to be for project sponsors. Ensure that the presentation gets to the point quickly and frames the results in terms of value to the sponsor's organization. When presenting to other audiences with more quantitative backgrounds, focus more time on the methodology and findings. In these instances, the team can be more expansive in describing the outcomes, methodology, and analytical experiments with a peer group. This audience will be more interested in the techniques, especially if the team developed a new way of processing or analyzing data that can be reused in the future or applied to similar problems. In addition, use imagery or data visualization when possible. Although it may take more time to develop imagery, pictures are more appealing, easier to remember, and more effective to deliver key messages than long lists of bullets.

12.2 Creating the Final Deliverables

After reviewing the list of key stakeholders for data science projects and main deliverables, this section focuses on describing the deliverables in detail. To illustrate this approach, a fictional case study is used to make the examples more specific. Figure 12-3 describes a scenario of a fictional bank, YoyoDyne Bank, which would like to embark on a project to do churn prediction models of its customers. *Churn rate* in this context refers to the frequency with which customers sever their relationship as customers of YoyoDyne Bank or switch to a competing bank.

Synopsis of YoyoDyne Bank Case Study

- YoyoDyne Bank is a retail bank that wants to improve its Net Present Value (NPV) and its customer retention rate.
- It wants to establish an effective marketing campaign targeting customers to reduce the churn rate by at least five percent.
- The bank wants to determine whether those customers are worth retaining. In addition, the bank wants to analyze reasons for customer attrition and what it can do to keep customers from leaving.
- The bank wants to build a data warehouse to support marketing and other related customer care groups.

FIGURE 12-3 *Synopsis of YoyoDyne Bank case study example*

Based on this information, the data science team may create an analytics plan similar to Figure 12-4 during the project.

Components of Analytic Plan	Retail Banking: YoyoDyne Bank
Discovery **Business Problem Framed**	How can the bank identify customers with the highest likelihood for churn?
Initial Hypotheses	Transaction volume and type are key predictors of churn rates
Data and Scope	5 months of customer account history
Model Planning - Analytic Technique	Logistic regression to identify most influential factors predicting churn
Result and Key Findings	Key predictors of churn are: 1. Once customers stop using their accounts for gas and groceries, their account holdings quickly diminish and the customers churn. 2. If the customers use their debit card fewer than 5 times per month, they will leave the bank within 60 days.
Business Impact	By targeting customers who are at high risk for churn, customer attrition can be reduced by 23%. This would save $3 million in lost customer revenue and avoid $1.5 million in new customer acquisition costs each year for the bank.

FIGURE 12-4 *Analytics plan for YoyoDyne Bank case study*

In addition to guiding the model planning and methodology, the analytic plan contains components that can be used as inputs for writing about the scope, underlying assumptions, modeling techniques, initial hypotheses, and key findings in the final presentations. After spending substantial amounts of time in the modeling and performing in-depth data analysis, it is critical to reflect on the project work and consider

the context of the problems the team set out to solve. Review the work that was completed during the project, and identify observations about the model outputs, scoring, and results. Based on these observations, begin to identify the key messages and any unexpected insights.

In addition, it is important to tailor the project outputs to the audience. For a project sponsor, show that the team met the project goals. Focus on what was done, what the team accomplished, what ROI can be anticipated, and what business value can be realized. Give the project sponsor talking points to evangelize the work. Remember that the sponsor needs to relay the story to others, so make this person's job easy, and help ensure the message is accurate by providing a few talking points. Find ways to emphasize ROI and business value, and mention whether the models can be deployed within performance constraints of the sponsor's production environment.

In some organizations, the data science team may not be expected to make a full business case for future projects and implementation of the models. Instead, it needs to be able to provide guidance about the impact of the models to enable the project sponsor, or someone designated by that person, to create a business case to advocate for the pilot and subsequent deployment of this functionality. In other words, the data science team can assist in this effort by putting the results of the modeling and data science work into context to help assess the actual value and cost of implementing this work more broadly.

When presenting to a technical audience such as data scientists and analysts, focus on how the work was done. Discuss how the team accomplished the goals and the choices it made in selecting models or analyzing the data. Share analytical methods and decision-making processes so other analysts can learn from them for future projects. Describe methods, techniques, and technologies used, as this technical audience will be interested in learning about these details and considering whether the approach makes sense in this case and whether it can be extended to other, similar projects. Plan to provide specifics related to model accuracy and speed, such as how well the model will perform in a production environment.

Ideally, the team should consider starting the development of the final presentation during the project rather than at the end of the project as commonly occurs. This approach ensures that the team always has a version of the presentation with working hypotheses to show stakeholders, in case there is a need to show a work-in-process version of the project progress on short notice. In fact, many analysts write the executive summary at the outset of a project and then continually refine it over time so that at the end of the project, portions of the final presentation are already completed. This approach also reduces the chance that the team members will forget key points or insights discovered during the project. Finally, it reduces the amount of work to be done on the presentation at the conclusion of the project.

12.2.1 Developing Core Material for Multiple Audiences

Because some of the components of the projects can be used for different audiences, it can be helpful to create a core set of materials regarding the project, which can be used to create presentations for either a technical audience or an executive sponsor.

Table 12-1 depicts the main components of the final presentations for the project sponsor and an analyst audience. Notice that teams can create a core set of materials in these seven areas, which can be used for the two presentation audiences. Three areas (Project Goals, Main Findings, and Model Description), can be used as is for both presentations. Other areas need additional elaboration, such as the Approach. Still other areas, such as the Key Points, require different levels of detail for the analysts and data scientists than for the project sponsor. Each of these main components of the final presentation is discussed in subsequent sections.

Table 12-1 *Comparison of Materials for Sponsor and Analyst Presentations*

Presentation Component	Project Sponsor Presentation	Analyst Presentation
Project Goals	List top 3–5 agreed-upon goals.	
Main Findings	Emphasize key messages.	
Approach	High-level methodology	High-level methodology Relevant details on modeling techniques and technology
Model Description	Overview of the modeling technique	
Key Points Supported with Data	Support key points with simple charts and graphics (example: bar charts).	Show details to support the key points. Analyst-oriented charts and graphs, such as ROC curves and histograms Visuals of key variables and significance of each
Model Details	Omit this section, or discuss only at a high level.	Show the code or main logic of the model, and include model type, variables, and technology used to execute the model and score data. Identify key variables and impact of each. Describe expected model performance and any caveats. Detailed description of the modeling technique Discuss variables, scope, and predictive power.
Recommendations	Focus on business impact, including risks and ROI. Give the sponsor salient points to help her evangelize work within the organization.	Supplement recommendations with implications for the modeling or for deploying in a production environment.

12.2.2 Project Goals

The Project Goals portion of the final presentation is generally the same, or similar, for sponsors and for analysts. For each audience, the team needs to reiterate the goals of the project to lay the groundwork for

the solution and recommendations that are shared later in the presentation. In addition, the Goals slide serves to ensure there is a shared understanding between the project team and the sponsors and confirm they are aligned in moving forward in the project. Generally, the goals are agreed on early in the project. It is good practice to write them down and share them to ensure the goals and objectives are clearly understood by both the project team and the sponsors.

Figures 12-5 and 12-6 show two examples of slides for Project Goals. Figure 12-5 shows three goals for creating a predictive model to anticipate customer churn. The points on this version of the Goals slide emphasize what needs to be done, but not why, which will be included in the alternative.

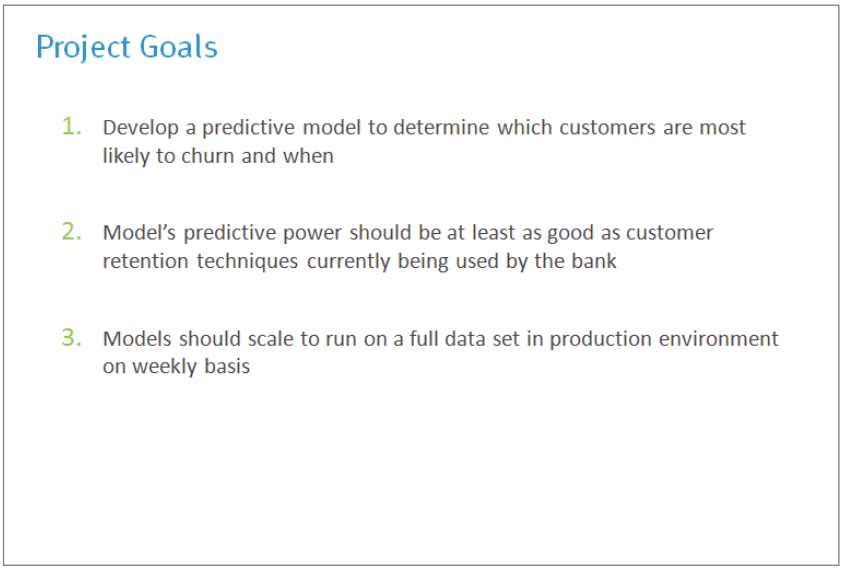

FIGURE 12-5 *Example of Project Goals slide for YoyoDyne case study*

Figure 12-6 shows a variation of the previous Project Goals slide in Figure 12-5. It is a summary of the situation prior to listing the goals. Keep in mind that when delivering final presentations, these deliverables are shared within organizations, and the original context can be lost, especially if the original sponsor leaves the group or changes roles. It is good practice to briefly recap the situation prior to showing the project goals. Keep in mind that adding a situation overview to the Goals slide does make it appear busier. The team needs to determine whether to split this into a separate slide or keep it together, depending on the audience and the team's style for delivering the final presentation.

One method for writing the situational overview in a succinct way is to summarize it in three bullets, as follows:

- **Situation:** Give a one-sentence overview of the situation that has led to the analytics project.
- **Complication:** Give a one-sentence overview of the need for addressing this now. Something has triggered the organization to decide to take action at this time. For instance, perhaps it lost 100

customers in the past two weeks and now has an executive mandate to address an issue, or perhaps it has lost five points of market share to its biggest competitor in the past three months. Usually, this sentence represents the driver for why a particular project is being initiated at this time, rather than in some vague time in the future.

- **Implication:** Give a one-sentence overview of the impact of the complication. For instance, if the bank fails to address its customer attrition problem, it stands to lose its dominant market position in three key markets. Focus on the business impact to illustrate the urgency of doing the project.

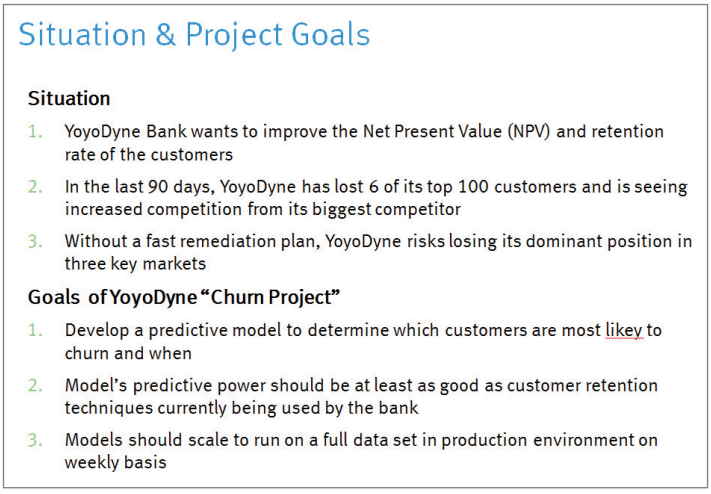

FIGURE 12-6 *Example of Situation & Project Goals slide for YoyoDyne case study*

12.2.3 Main Findings

Write a solid executive summary to portray the main findings of a project. In many cases, the summary may be the only portion of the presentation that hurried managers will read. For this reason, it is imperative to make the language clear, concise, and complete. Those reading the executive summary should be able to grasp the full story of the project and the key insights in a single slide. In addition, this is an opportunity to provide key talking points for the executive sponsor to use to evangelize the project work with others in the customer's organization. Be sure to frame the outcomes of the project in terms of both quantitative and qualitative business value. This is especially important if the presentation is for the project sponsor. The executive summary slide containing the main findings is generally the same for both sponsor and analyst audiences.

Figure 12-7 shows an example of an executive summary slide for the YoyoDyne case study. It is useful to take a closer look at the parts of the slide to make sure it is clear. Keep in mind this is not the only format for conveying the Executive Summary; it varies based on the author's style, although many of the key components are common themes in Executive Summaries.

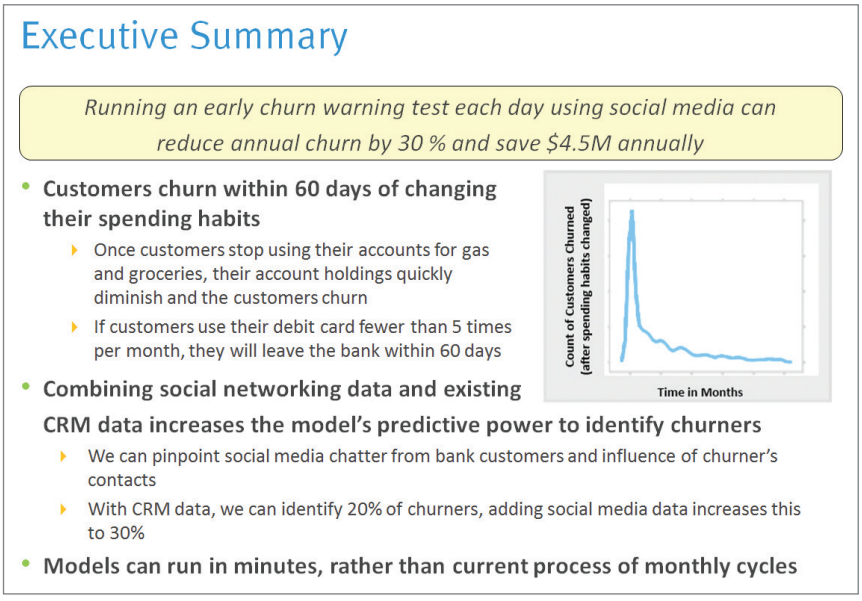

FIGURE 12-7 *Example of Executive Summary slide for YoyoDyne case study*

The key message should be clear and conspicuous at the front of the slide. It can be set apart with color or shading, as shown in Figure 12-8; other techniques can also be used to draw attention to it. The key message may become the single talking point that executives or the project sponsor take away from the project and use to support the team's recommendation for a pilot project, so it needs to be succinct and compelling. To make this message as strong as possible, measure the value of the work and quantify the cost savings, revenue, time savings, or other benefits to make the business impact concrete.

Follow the key message with three major supporting points. Although Executive Summary slides can have more than three major points, going beyond three ideas makes it difficult for people to recall the main points, so it is important to ensure that the ideas remain clear and limited to the few most impactful ideas the team wants the audience to take away from the work that was done. If the author lists ten key points, messages become diluted, and the audience may remember only one or two main points.

In addition, because this is an analytics project, be sure to make one of the key points related to if, and how well, the work will meet the sponsor's service level agreement (SLA) or expectations. Traditionally, the SLA refers to an arrangement between someone providing services, such as an information technology (IT) department or a consulting firm, and an end user or customer. In this case, the SLA refers to system performance, expected uptime of a system, and other constraints that govern an agreement. This term has become less formal and many times conveys system performance or expectations more generally related to performance or timeliness. It is in this sense that SLA is being used here. Namely, in this context, SLA

refers to the expected performance of a system and the intent that the models developed will not adversely impact the expected performance of the system into which they are integrated.

Finally, although it's not required, it is often a good idea to support the main points with a visual or graph. Visual imagery serves to make a visceral connection and helps retain the main message with the reader.

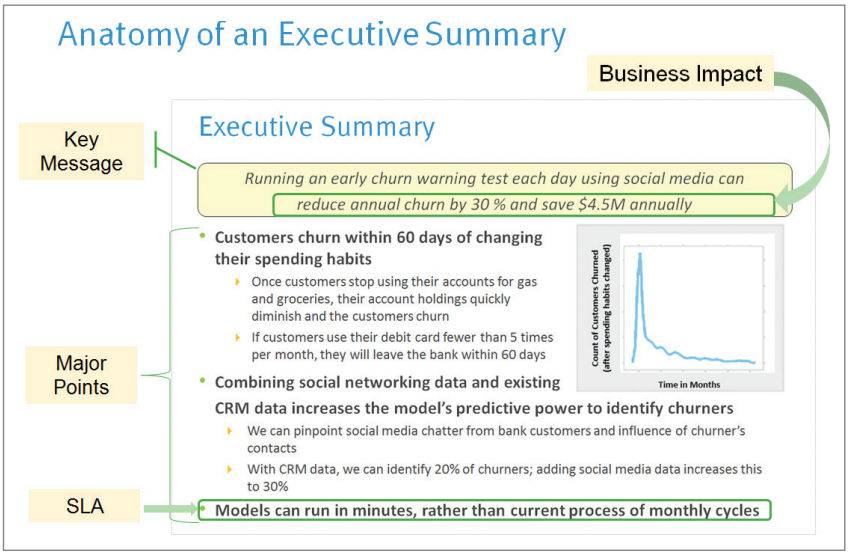

FIGURE 12-8 *Anatomy of an Executive Summary slide*

12.2.4 Approach

In the Approach portion of the presentation, the team needs to explain the methodology pursued on the project. This can include interviews with domain experts, the groups collaborating within the organization, and a few statements about the solution developed. The objective of this slide is to ensure the audience understands the course of action that was pursued well enough to explain it to others within the organization. The team should also include any additional comments related to working assumptions the team followed as it performed the work, because this can be critical in defending why they followed a specific course of action.

When explaining the solution, the discussion should remain at a high level for the project sponsors. If presenting to analysts or data scientists, provide additional detail about the type of model used, including the technology and the actual performance of the model during the tests. Finally, as part of the description of the approach, the team may want to mention constraints from systems, tools, or existing processes and any implications for how these things may need to change with this project.

Figure 12-9 shows an example of how to describe the methodology followed during a data science project to a sponsor audience.

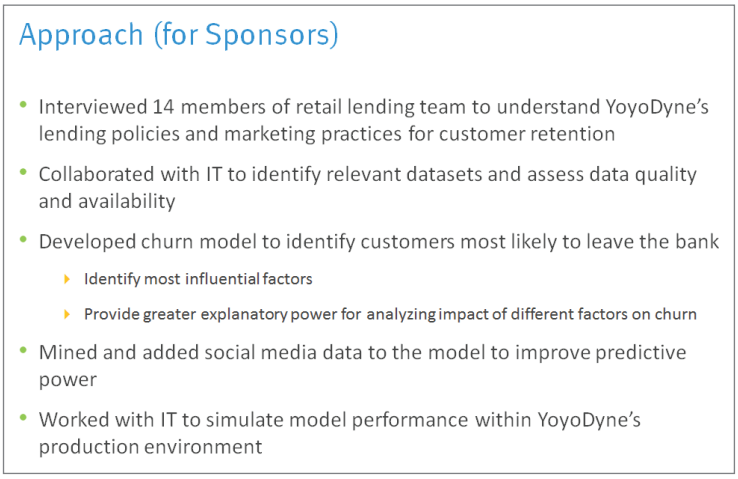

FIGURE 12-9 *Example describing the project methodology for project sponsors*

Note that the third bullet describes the churn model in general terms. Furthermore, the subbullets provide additional details in nontechnical terms. Compare this approach to the variation shown in Figure 12-10.

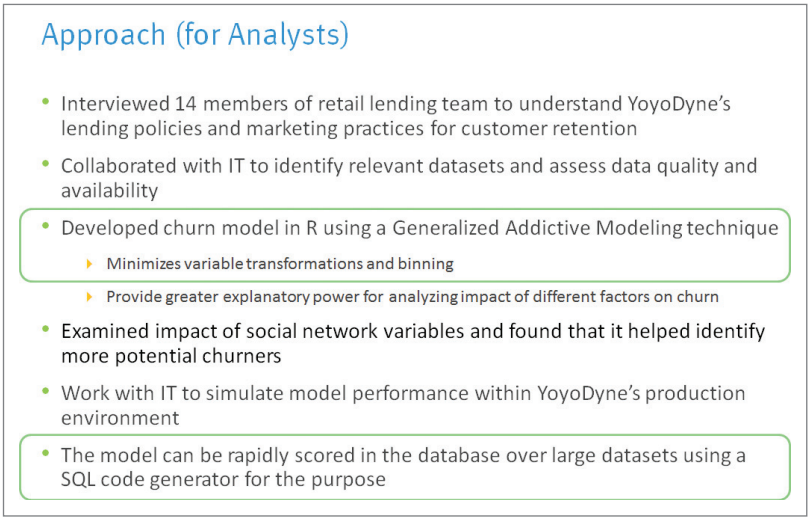

FIGURE 12-10 *Example describing the project methodology for analysts and data scientists*

Figure 12-10 shows a variation on the approach and methodology used in the data science project. In this case, most of the language and description are the same as in the example for project sponsors.

The main difference is that this version contains additional detail regarding the kind of model used and the way the model will score data quickly to meet the SLA. These differences are highlighted in the boxes shown in Figure 12-10.

12.2.5 Model Description

After describing the project approach, teams generally include a description of the model that was used. Figure 12-11 provides the model description for the Yoyodyne Bank example. Although the Model Description slide can be the same for both audiences, the interests and objectives differ for each. For the sponsor, the general methodology needs to be articulated without getting into excessive detail. Convey the basic methodology followed in the team's work to allow the sponsor to communicate this to others within the organization and provide talking points.

Mentioning the scope of the data used is critical. The purpose is to illustrate thoroughness and exude confidence that the team used an approach that accurately portrays its problem and is as free from bias as possible. A key trait of a good data scientist is the ability to be skeptical of one's own work. This is an opportunity to view the work and the deliverable critically and consider how the audience will receive the work. Try to ensure it is an unbiased view of the project and the results.

Assuming that the model will meet the agreed-upon SLAs, mention that the model will meet the SLAs based on the performance of the model within the testing or staging environment. For instance, one may want to indicate that the model processed 500,000 records in 5 minutes to give stakeholders an idea of the speed of the model during run time. Analysts will want to understand the details of the model, including the decisions made in constructing the model and the scope of the data extracts for testing and training. Be prepared to explain the team's thought process on this, as well as the speed of running the model within the test environment.

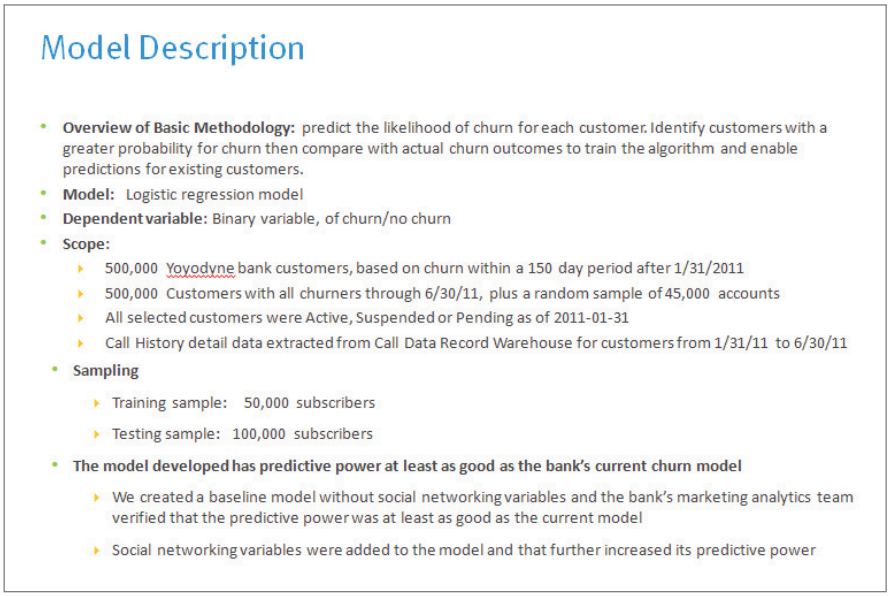

Figure 12-11 *Example of a model description for a data science project*

12.2.6 Key Points Supported with Data

The next step is to identify key points based on insights and observations resulting from the data and model scoring results. Find ways to illustrate the key points with charts and visualization techniques, using simpler charts for sponsors and more technical data visualization for analysts and data scientists.

Figure 12-12 shows an example of providing supporting detail regarding the rate of bank customers who would churn in various months. When developing the key points, consider the insights that will drive the biggest business impact and can be defended with data. For project sponsors, use simple charts such as bar charts, which illustrate data clearly and enable the audience to understand the value of the insights. This is also a good point to foreshadow some of the team's recommendations and begin tying together ideas to demonstrate what led to the recommendations and why. In other words, this section supplies the data and foundation for the recommendations that come later in the presentation. Creating clear, compelling slides to show the key points makes the recommendations more credible and more likely to be acted upon by the customer or sponsor.

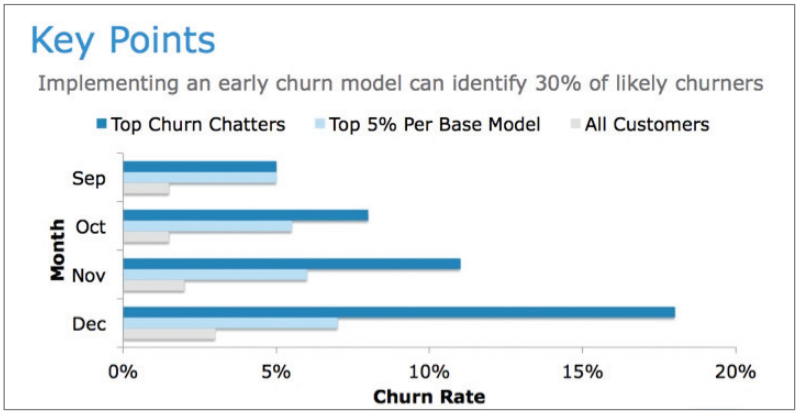

Figure 12-12 *Example of a presentation of key points of a data science project shown as a bar chart*

For analyst presentations, use more granular or technical charts and graphs. In this case, appropriate visualization techniques include dot charts, density plots, ROC curves, or histograms of a data distribution to support decisions made in the modeling techniques. Basic concepts of data visualization are discussed later in the chapter.

12.2.7 Model Details

Model details are typically needed by people who have a more technical understanding than the sponsors, such as those who will implement the code, or colleagues on the analytics team. Project sponsors are typically less interested in the model details; they are usually more focused on the business implications of the work rather than the details of the model. This portion of the presentation needs to show the code or main logic of the model, including the model type, variables, and technology used to execute

the model and score the data. The model details segment of the presentation should focus on describing expected model performance and any caveats related to the model performance. In addition, this portion of the presentation should provide a detailed description of the modeling technique, variables, scope, and expected effectiveness of the model.

This is where the team can provide discussion or written details related to the variables used in the model and explain how or why these variables were selected. In addition, the team should share the actual code (or at least an excerpt) developed to explain what was created and how it operates. This also serves to foster discussion related to any additional constraints or implications related to the main logic of the code. In addition, the team can use this section to illustrate details of the key variables and the predictive power of the model, using analyst-oriented charts and graphs, such as histograms, dot charts, density plots, and ROC curves.

Figure 12-13 provides a sample slide describing the data variables, and Figure 12-14 shows a sample slide with a technical graph to support the work.

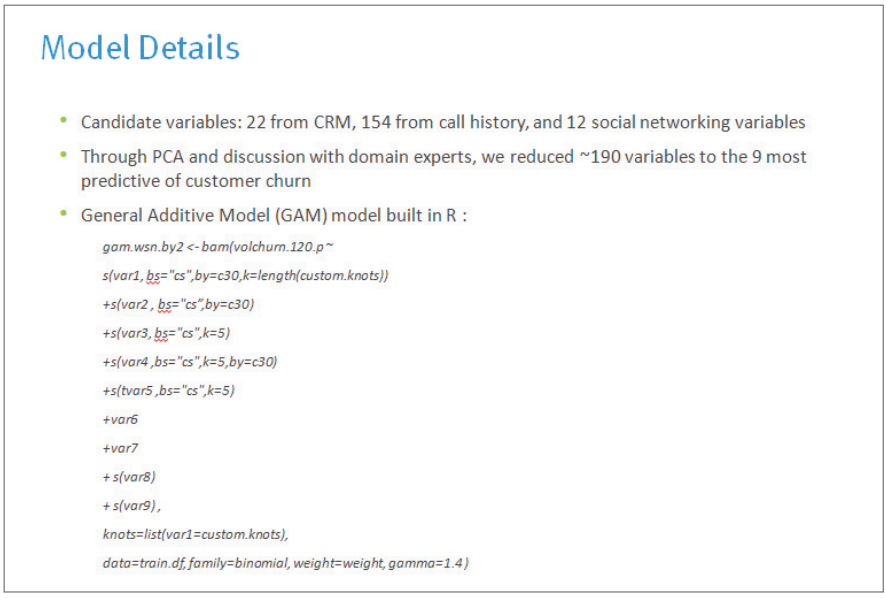

Figure 12-13 *Example of model details showing model type and variables*

As part of the model detail description, guidance should be provided regarding the speed with which the model can run in the test environment; the expected performance in a live, production environment; and the technology needed. This kind of discussion addresses how well the model can meet the organization's SLA.

This section of the presentation needs to include additional caveats, assumptions, or constraints of the model and model performance, such as systems or data the model needs to interact with, performance

issues, and ways to feed the outputs of the model into existing business processes. The author of this section needs to describe the relationships of the main variables on the project objectives, such as the effects of key variables on predicting churn, and the relationship of key variables to other variables. The team may even want to make suggestions to improve the model, highlight any risks to introducing bias into the modeling technique, or describe certain segments of the data that may skew the overall predictive power of the methodology.

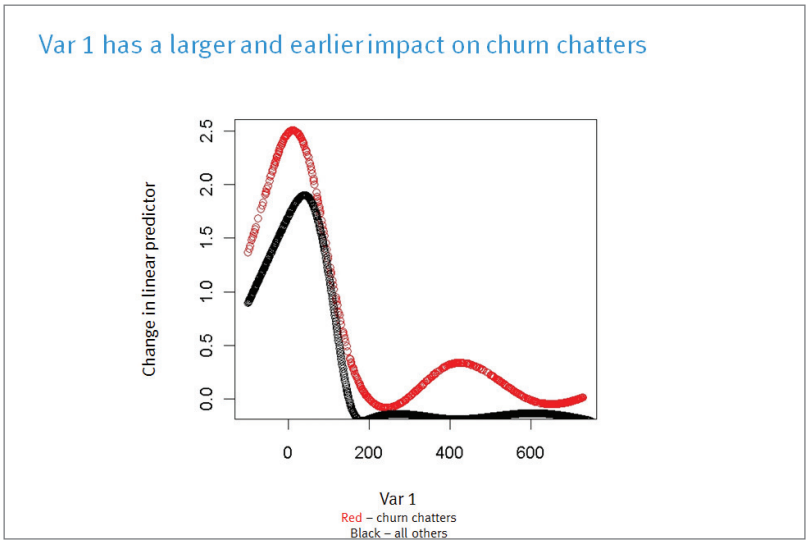

FIGURE 12-14 *Model details comparing two data variables*

12.2.8 Recommendations

The final main component of the presentation involves creating a set of recommendations that include how to deploy the model from a business perspective within the organization and any other suggestions on the rollout of the model's logic. For the Yoyodyne Bank example, Figure 12-15 provides possible recommendations from the project. In this section of the presentation, measuring the impact of the improvements and stating how to leverage that impact within the recommendations are key. For instance, the presentation might mention that every customer retained represents a time savings of six hours for one of the bank's account managers or $50,000 in savings of new account acquisitions, due to marketing costs, sales, and system-related costs.

For a presentation to a project sponsor audience, focus on the business impact of the project, including risks and ROI. Because project sponsors will be most interested in the business impact of the project, the presentation should also provide the sponsor with salient points to help evangelize the work within the organization. When preparing a presentation for analysts, supplement the main set of recommendations with any implications for the modeling or for deployment in a production environment. In either case, the

team should focus on recommending actions to operationalize the work and the benefits the customer will receive because of implementing these recommendations.

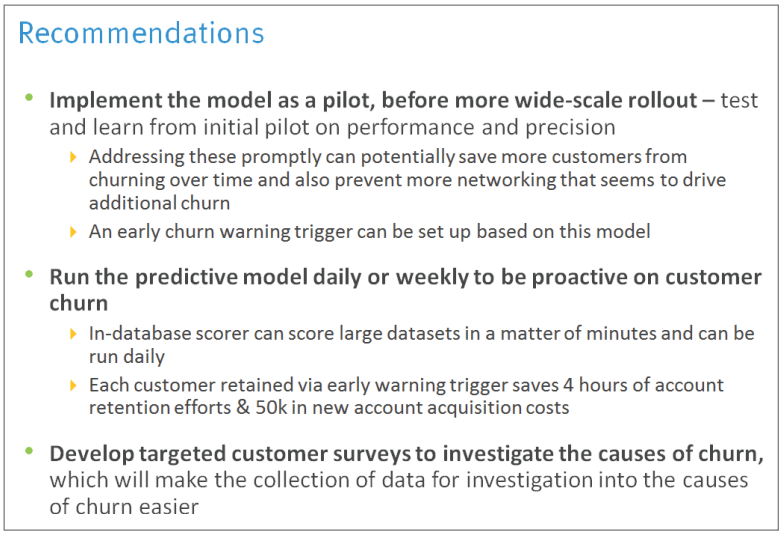

FIGURE 12-15 *Sample recommendations for a data science project*

12.2.9 Additional Tips on the Final Presentation

As a team completes a project and strives to move on to the next one, it must remember to invest adequate time in developing the final presentations. Orienting the audience to the project and providing context is important. On occasion, a team is so immersed in the project that it fails to provide sufficient context for its recommendations and the outputs of the models. A team needs to remember to spell out terminology and acronyms and avoid excessive use of jargon. It should also keep in mind that presentations may be shared extensively; therefore, recipients may not be familiar with the context and the journey the team has gone through over the course of the project.

The story may need to be told multiple times to different audiences, so the team must remain patient in repeating some of the key messages. These presentations should be viewed as opportunities to refine the key messages and evangelize the good work that was done. By this point in the process, the team has invested many hours of work and uncovered insights for the business. These presentations are an opportunity to communicate these projects and build support for future projects. As with most presentations, it is important to gauge the audience to guide shaping the message and the level of detail. Here are several more tips on developing the presentations.

- **Use imagery and visual representations:** Visuals tend to make the presentation more compelling. Also, people recall imagery better than words, because images can have a more visceral impact. These visual representations can be static and interactive data.

- **Make sure the text is mutually exclusive and collectively exhaustive (MECE):** This means having an economy of words in the presentation and making sure the key points are covered but not repeated unnecessarily.
- **Measure and quantify the benefits of the project:** This can be challenging and requires time and effort to do well. This kind of measurement should attempt to quantify benefits that have financial and other benefits in a specific way. Making the statement that a project provided "$8.5M in annual cost savings" is much more compelling than saying it has "great value."
- **Make the benefits of the project clear and conspicuous:** After calculating the benefits of the project, make sure to articulate them clearly in the presentation.

12.2.10 Providing Technical Specifications and Code

In addition to authoring the final presentations, the team needs to deliver the actual code that was developed and the technical documentation needed to support it. The team should consider how the project will affect the end users and the technical people who will need to implement the code. It is recommended that the team think through the implications of its work on the recipients of the code, the kinds of questions they will have, and their interests. For instance, indicating that the model will need to perform real-time monitoring may require extensive changes to an IT runtime environment, so the team may need to consider a compromise of nightly batch jobs to process the data. In addition, the team may need to get the technical team talking with the project sponsor to ensure the implementation and SLA will meet the business needs during the technical deployment.

The team should anticipate questions from IT related to how computationally expensive it will be to run the model in the production environment. If possible, indicate how well the model ran in the test scenarios and whether there are opportunities to tune the model or environment to optimize performance in the production environment.

Teams should approach writing technical documentation for their code as if it were an application programming interface (API). Many times, the models become encapsulated as functions that read a set of inputs in the production environment, possibly perform preprocessing on data, and create an output, including a set of post-processing results.

Consider the inputs, outputs, and other system constraints to enable a technical person to implement the analytical model, even if this person has not had a connection to the data science project up to this point. Think about the documentation as a way to introduce the data that the model needs, the logic it is using, and how other related systems need to interact with it in a production environment for it to operate well. The specifications detail the inputs the code needs and the data format and structures. For instance, it may be useful to specify whether structured data is needed or whether the expected data needs to be numeric or string formats. Describe any transformations that need to be made on the input data before the code can use it, and if scripting was created to perform these tasks. These kinds of details are important when other engineers must modify the code or utilize a different dataset or table, if and when the environment changes.

Regarding exception handling, the team must consider how the code should handle data that is outside the expected data ranges of the model parameters and how it will handle missing data values (Chapter 3, "Review of Basic Data Analytic Methods Using R"), null values, zeros, NAs, or data that is in an unexpected format or type. The technical documentation describes how to treat these exceptions and what implications may emerge on downstream processes. For the model outputs, the team must explain to what extent to post-process the output. For example, if the model returns a value representing

the probability of customer churn, additional logic may be needed to identify the scoring threshold to determine which customer accounts to flag as being at risk of churn. In addition, some provision should be made for adjusting this threshold and training the algorithm, either in an automated learning fashion or with human intervention.

Although the team must create technical documentation, many times engineers and other technical staff receive the code and may try to use it without reading through all the documentation. Therefore, it is important to add extensive comments in the code. This directs the people implementing the code on how to use it, explains what pieces of the logic are supposed to do, and guides other people through the code until they're familiar with it. If the team can do a thorough job adding comments in the code, it is much easier for someone else to maintain the code and tune it in the runtime environment. In addition, it helps the engineers edit the code when their environment changes or they need to modify processes that may be providing inputs to the code or receiving its outputs.

12.3 Data Visualization Basics

As the volume of data continues to increase, more vendors and communities are developing tools to create clear and impactful graphics for use in presentations and applications. Although not exhaustive, Table 12-2 lists some popular tools.

TABLE 12-2 *Common Tools for Data Visualization*

Open Source	Commercial Tools
R (Base package, `lattice`, `ggplot2`)	Tableau
GGobi/Rggobi	Spotfire (TIBCO)
Gnuplot	QlikView
Inkscape	Adobe Illustrator
Modest Maps OpenLayers	
Processing D3.js Weave	

As the volume and complexity of data has grown, users have become more reliant on using crisp visuals to illustrate key ideas and portray rich data in a simple way. Over time, the open source community has developed many libraries to offer more options for portraying graphics data visually. Although this book showed examples primarily using the base package of R, `ggplot2` provides additional options for creating professional-looking data visualization, as does the `lattice` library for R.

Gnuplot and GGobi have a command-line-driven approach to generating data visualization. The genesis of these tools mainly grew out of scientific computing and the need to express complex data visually. GGobi

also has a variant called Rggobi that enables users to access the GGobi functionality with the R software and programming language. There are many open source mapping tools available, including Modest Maps and OpenLayers, both designed for developers who would like to create interactive maps and embed them within their own development projects or on the web. The software programming language development environment, Processing, employs a Java-like language for developers to create professional-looking data visualization. Because it is based on a programming language rather than a GUI, Processing enables developers to create robust visualization and have precise control over the output. D3.js is a JavaScript library for manipulating data and creating web-based visualization with standards, such as Hypertext Markup Language (HTML), Scalable Vector Graphics (SVG), and Cascading Style Sheets (CSS). For more examples of using open source visualization tools, refer to Nathan Yau's website, flowingdata.com [1], or his book ***Visualize This*** [2], which discusses additional methods for creating data representations with open source tools.

Regarding the commercial tools shown in Table 12-2, Tableau, Spotfire (by TIBCO), and QlikView function as data visualization tools and as interactive business intelligence (BI) tools. Due to the growth of data in the past few years, organizations for the first time are beginning to place more emphasis on ease of use and visualization in BI over more traditional BI tools and databases. These tools make visualization easy and have user interfaces that are cleaner and simpler to navigate than their predecessors. Although not traditionally considered a data visualization tool, Adobe Illustrator is listed in Table 12-2 because some professionals use it to enhance visualization made in other tools. For example, some users develop a simple data visualization in R, save the image as a PDF or JPEG, and then use a tool such as Illustrator to enhance the quality of the graphic or stitch multiple visualization work into an infographic. Inkscape is an open source tool used for similar use cases, with much of Illustrator's functionality.

12.3.1 Key Points Supported with Data

It is more difficult to observe key insights when data is in tables instead of in charts. To underscore this point, in ***Say it with Charts,*** Gene Zelazny [3] mentions that to highlight data, it is best to create a visual representation out of it, such as a chart, graph, or other data visualization. The opposite is also true. Suppose an analyst chooses to downplay the data. Sharing it in a table draws less attention to it and makes it more difficult for people to digest.

The way one chooses to organize the visual in terms of the color scheme, labels, and sequence of information also influences how the viewer processes the information and what he perceives as the key message from the chart. The table shown in Figure 12-16 contains many data points. Given the layout of the information, it is difficult to identify the key points at a glance. Looking at 45 years of store opening data can be challenging, as shown in Figure 12-16.

Year	1962	1963	1964	1965	1966	1967	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	Total
SuperBox	1		1	1		1	5	4	4	14	13	14	20	14	17	29	24	37	33	117	42	65	79	81	90	92	82	86	106	72	62	62	40	49	22	26	33	47	78	71	67	64	91	91	33	1980
BigBox						1		1	1	1	4	5	5	5	10	10	10	6	21	33	21	22	20	29	31	50	43	45	72	91	76	94	67	80	31	34	33	33	27	35	47	32	39	27	4	1196
Total	1		1	1		2	5	5	5	15	17	19	25	19	27	39	34	43	54	150	63	87	99	110	121	142	125	131	178	163	138	156	107	129	53	60	66	80	105	106	114	96	130	118	37	3176

FIGURE 12-16 *Forty-five years of store opening data*

Even showing somewhat less data is still difficult to read through for most people. Figure 12-17 hides the first 10 years, leaving 35 years of data in the table.

Year	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	Total
SuperBox	13	14	20	14	17	29	24	37	33	117	42	65	79	81	90	92	82	86	106	72	62	62	40	49	22	26	33	47	78	71	67	64	91	91	33	1980
BigBox	4	5	5	5	10	10	10	6	21	33	21	22	20	29	31	50	43	45	72	91	76	94	67	80	31	34	33	33	27	35	47	32	39	27	4	1196
Total	17	19	25	19	27	39	34	43	54	150	63	87	99	110	121	142	125	131	178	163	138	156	107	129	53	60	66	80	105	106	114	96	130	118	37	3176

FIGURE 12-17 *Thirty-five years of store opening data*

As most readers will observe, it is challenging to make sense of data, even at relatively small scales. There are several observations in the data that one may notice, if one looks closely at the data tables:

- BigBox experienced strong growth in the 1980s and 1990s.
- By the 1980s, BigBox began adding more SuperBox stores to its mix of chain stores.
- SuperBox stores outnumber BigBox stores nearly 2 to 1 in aggregate.

Depending on the point trying to be made, the analyst must take care to organize the information in a way that intuitively enables the viewer to take away the same main point that the author intended. If the analyst fails to do this effectively, the person consuming the data must guess at the main point and may interpret something different from what was intended.

Figure 12-18 shows a map of the United States, with the points representing the geographic locations of the stores. This map is a more powerful way to depict data than a small table would be. The approach is well suited to a sponsor audience. This map shows where the BigBox store has market saturation, where the company has grown, and where it has SuperBox stores and other BigBox stores, based on the color and shading. The visualization in Figure 12-18 clearly communicates more effectively than the dense tables in Figure 12-16 and Figure 12-17. For a sponsor audience, the analytics team can also use other simple visualization techniques to portray data, such as bar charts or line charts.

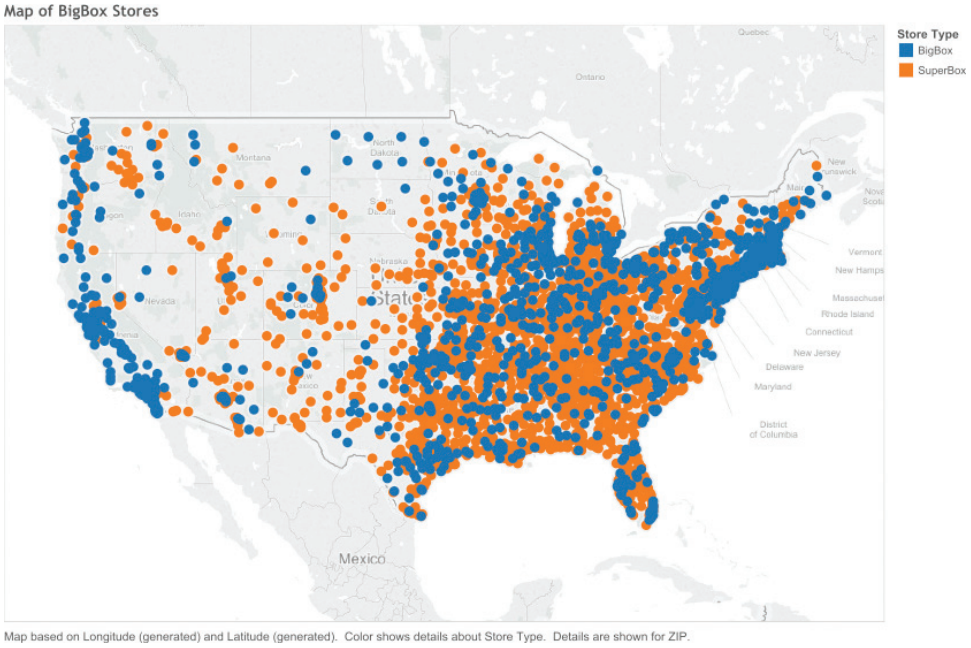

FIGURE 12-18 *Forty-five years of store opening data, shown as map*

12.3.2 Evolution of a Graph

Visualization allows people to portray data in a more compelling way than tables of data and in a way that can be understood on an intuitive, precognitive level. In addition, analysts and data scientists can use visualization to interact with and explore data. Following is an example of the steps a data scientist may go through in exploring pricing data to understand the data better, model it, and assess whether a current pricing model is working effectively. Figure 12-19 shows a distribution of pricing data as a user score reflecting price sensitivity.

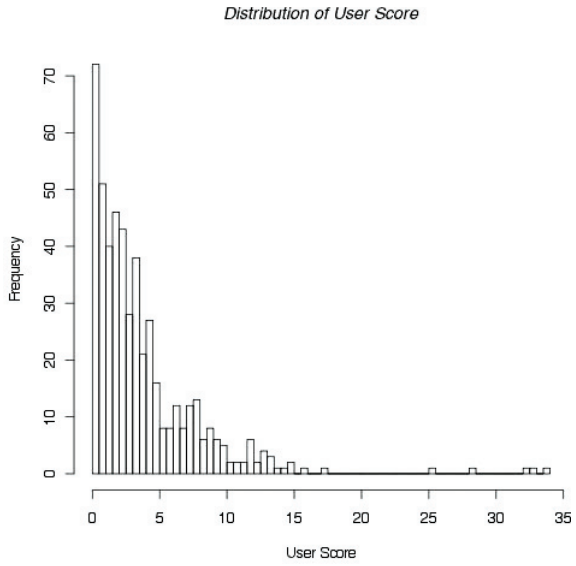

FIGURE 12-19 *Frequency distribution of user scores*

A data scientist's first step may be to view the data as a raw distribution of the pricing levels of users. Because the values have a long tail to the right, in Figure 12-19, it may be difficult to get a sense of how tightly clustered the data is between user scores of zero and five.

To understand this better, a data scientist may rerun this distribution showing a log distribution (Chapter 3) of the user score, as demonstrated in Figure 12-20.

This shows a less skewed distribution that may be easier for a data scientist to understand. Figure 12-21 illustrates a rescaled view of Figure 12-20, with the median of the distribution around 2.0. This plot provides the distribution of a new user score, or index, that may gauge the level of price sensitivity of a user when expressed in log form.