Chapter Introduction

In this chapter, you will learn:

- What Big Data is and why it is important in modern business
- The primary characteristics of Big Data and how these go beyond the traditional "3 Vs"
- How the core components of the Hadoop framework, HDFS and MapReduce, operate
- What the major components of the Hadoop ecosystem are
- The four major approaches of the NoSQL data model and how they differ from the relational model
- About data analytics, including data mining and predictive analytics

Preview

In Chapter 2, Data Models, you were introduced to the emerging NoSQL data model and the Big Data problem that has led to NoSQL's development. In this chapter, you learn about these issues in much greater detail. You will find that there is more to Big Data and the problem that it represents to modern businesses than just the volume, velocity, and variety ("3 V") characteristics introduced in Chapter 2. In fact, you will find that these characteristics themselves are more complex than previously discussed.

After learning about Big Data issues, you learn about the technologies that have developed, and continue to be developed, to address Big Data. First, you learn about the low-level technologies in the Hadoop framework. Hadoop has become a standard component in organizations' efforts to address Big Data. Next, you learn about the higher-level approaches of the NoSQL data model to develop nonrelational databases such as key-value databases, document databases, column-oriented databases, and graph databases.

Finally, you learn about the important area of data analytics and how statistical techniques are being used to help organizations turn the vast stores of data that are being collected into actionable information. Analytics are helping organizations understand not only what has happened in the business, but also to predict what is likely to happen.

The relational database model has been dominant for decades, and during that time it has faced challenges such as object-oriented databases and the development of data warehouses. The relational model, and the tools based on it, have evolved to adapt to these challenges and remain dominant in the data management arena. In each case, the challenge arose because technological advances changed business's perceptions of what is possible and created new opportunities for organizations to create value from increased data leverage. The latest of these challenges is Big Data. Big Data is an ill-defined term that describes a new wave of data storage and manipulation possibilities and requirements. Organizations' efforts to store, manipulate, and analyze this new wave of data represent one of the most urgent emerging trends in the database field. The challenges of dealing with the wave of Big Data have led to the development of NoSQL databases that reject many of the underlying assumptions of the relational model. Although the term *Big Data* lacks a consistent definition, there is a set of characteristics generally associated with it.

14-1Big Data

Big Data generally refers to a set of data that displays the characteristics of volume, velocity, and variety (the “3 Vs”) to an extent that makes the data unsuitable for management by a relational database management system. These characteristics can be defined as follows:

- **Volume**—the quantity of data to be stored
- **Velocity**—the speed at which data is entering the system
- **Variety**—the variations in the structure of the data to be stored

Notice the lack of specific values associated with these characteristics. This lack of specificity is what leads to the ambiguity in defining Big Data. What was Big Data five years ago might not be considered Big Data now. Similarly, something considered Big Data now might not be considered Big Data five years from now. The key is that the characteristics are present to an extent that the current relational database technology struggles with managing the data.

Further adding to the problem of defining Big Data is that there is some disagreement among pundits about which of the 3 Vs must be present for a data set to be considered Big Data. Originally, Big Data was conceived as shown in Figure 14.1 as a combination of the 3 Vs. Web data, a combination of text, graphics, video, and audio sources combined into complex structures, is often cited as creating the new challenges for data management that involve all three characteristics. After the dot-com bubble burst in the 1990s, many startup web-based companies failed, but the companies that survived experienced significant growth as web commerce consolidated into a smaller set of businesses. As a result, companies like Google and Amazon experienced significant growth and were among the first to feel the pressure of managing Big Data. The success of social media giant Facebook quickly followed, and these companies became pioneers in creating new technologies to address Big Data problems. Google created the BigTable data store, Amazon created Dynamo, and Facebook created Cassandra to deal with the growing need to store and manage large sets of data that had the characteristics of the 3 Vs.

Figure 14.1Original View of Big Data

Although social media and web data have been at the forefront of perceptions of Big Data issues, other organizations have Big Data issues, too. More recently, changes in technology have increased the opportunities for businesses to generate and track data so that Big Data has been redefined as involving any, but not necessarily all, of the 3 Vs, as shown in Figure 14.2. Advances in technology have led to a vast array of user-generated data and machine-generated data that can spur growth in specific areas.

Figure 14.2Current View of Big Data

For example, Disney World has introduced “Magic Bands” for park visitors to wear on their wrists. Each visitor's Magic Band is connected to much of the data that Disney stores about that individual. These bands use RFID and near-field communications (NFC) to act as tickets for rides, hotel room keys, and even credit cards within the park. The bands can be tracked so the Disney systems can track individuals as they move through the park, record with which Disney characters (who are also tracked) they interact, purchases made, wait time in lines, and more. Visitors can make reservations at a restaurant and order meals through a Disney app on their smartphones, and by tracking the Magic Bands, the restaurant staff knows when the visitor arrives for their reservation, can track at which table they are seated, and deliver their meals within minutes of the guests sitting down. With the many cameras mounted throughout the park, Disney can also capture pictures and short videos of the visitor throughout their stay in the park to produce a personalized movie of their vacation experience, which can then be sold to the visitor as a souvenir. All of this involves the capture of a constant stream of data from each band, processed in real time. Considering the thousands of visitors in Disney World each day, each with their own Magic Band, the volume, velocity, and variety of the data is enormous.

14-1a Volume

Volume, the quantity of data to be stored, is a key characteristic of Big Data. The storage capacities associated with Big Data are extremely large. Table 14.1 provides definitions for units of data storage capacity.

Table 14.1

Storage Capacity Units

Term	Capacity	Abbreviation
Bit	0 or 1 value	b
Byte	8 bits	B
Kilobyte	1024* bytes	KB
Megabyte	1024 KB	MB
Gigabyte	1024 MB	GB
Terabyte	1024 GB	TB
Petabyte	1024 TB	PB
Exabyte	1024 PB	EB
Zettabyte	1024 EB	ZB
Yottabyte	1024 ZB	YB

Naturally, as the quantity of data needing to be stored increases, the need for larger storage devices increases as well. When this occurs, systems can either scale up or scale out. **Scaling up** is keeping the same number of systems, but migrating each system to a larger system: for example, changing from a server with 16 CPU cores and a 1 TB storage system to a server with 64 CPU cores and a 100 TB storage system. Scaling up involves moving to larger and faster systems. However, there are limits to how large and fast a single system can be. Further, the costs of these high-powered systems increase at a dramatic rate. On the other hand, **scaling out** means that when the workload exceeds the capacity of a server, the workload is spread out across a number of servers. This is also referred to as *clustering*—creating a cluster of low-cost servers to share a workload. This can help to reduce the overall cost of the computing resources since it is cheaper to buy ten 100 TB storage systems than it is to buy a single 1 PB storage system. Make no mistake, organizations need storage capacities in these extreme sizes. The eBay singularity system, which collects clickstream data among other things, is over 40 PB. This is in addition to the eBay enterprise data warehouse, which is over 14 PB and spread over hundreds of thousands of nodes.*

Recall from Chapter 3 that one of the greatest advances represented by the relational model was the development of an RDBMS—a sophisticated database management system that could hide the complexity of the underlying data storage and manipulation from the user so that the data always appears to be in tables. To carry out these functions, the DBMS acts as the "brain" of the database system and must maintain control over all of the data within the database. As discussed in Chapter 12, it is possible to distribute a relational database over multiple servers using replication and fragmentation. However, because the DBMS must act as a single point of control for all of the data in the database, distributing the database across multiple systems requires a high degree of communication and coordination across the systems. There are significant limits associated with the ability to distribute the DBMS due to the increased performance costs of communication and coordination as the number of nodes grows. This limits the degree to which a relational database to be scaled out as data volume grows, and it makes RDBMSs ill-suited for clusters.

Note

Although some RDBMS products, such as SQL Server and Oracle Real-Application Clusters (RAC), legitimately claim to support clusters, these clusters are limited in scope and generally rely on a single, shared data storage subsystem, such as a storage area network (SAN).

14-1b Velocity

Velocity, another key characteristic of Big Data, refers to the rate at which new data enters the system as well as the rate at which the data must be processed. In many ways, the issues of velocity mirror those of volume. For example, consider a web retailer such as Amazon. In the past, a retail store might capture only the data about the final transaction of a customer making a purchase. A retailer like Amazon captures not only the final transaction, but every click of the mouse in the searching, browsing, comparing, and purchase process. Instead of capturing one event (the final sale) in a 20-minute shopping experience, it might capture data on 30 events during that 20-minute time frame—a 30× increase in the velocity of the data. Other advances in technology, such as RFID, GPS, and NFC, add new layers of data-gathering opportunities that often generate large amounts of data that must be stored in real-time. For example, RFID tags can be used to track items for inventory and warehouse management. The tags do not require line-of-sight between the tag and the reader, and the reader can read hundreds of tags simultaneously while the products are still in boxes. This means that instead of a single record for tracking a given quantity of a product being produced, each individual product is tracked, creating an increase of several orders of magnitude in the amount of data being delivered to the system at any one time.

In addition to the speed with which data is entering the system, for Big Data to be actionable, that data must be processed at a very rapid pace. The velocity of processing can be broken down into two categories.

- Stream processing
- Feedback loop processing

Stream processing focuses on input processing, and it requires analysis of the data stream as it enters the system. In some situations, large volumes of data can enter the system at such a rapid pace that it is not feasible to try to store all of the data. The data must be processed and filtered as it enters the system to determine which data to keepand which data to discard. For example, at the CERN Large Hadron Collider, the largest and most powerful particle accelerator in the world, experiments produce about 600 TB per second of raw data. Scientists have created **algorithms** to decide ahead of time which data will be kept. These algorithms are applied in a two-step process to filter the data down to only about 1 GB per second of data that will actually be stored.*

Feedback loop processing refers to the analysis of the data to produce actionable results. While stream processing could be thought of as focused on inputs, feedback loop processing can be thought of as focused on outputs. The process of capturing the data, processing it into usable information, and then acting on that information is a feedback loop. Figure 14.3 shows a feedback loop for providing recommendations for book purchases. Feedback loop processing to provide immediate results requires analyzing large amounts of data within just a few seconds so that the results of the analysis can become a part of the product delivered to the user in real time. Not all feedback loops are used for inclusion of results within immediate data products. Feedback loop processing is also used to help organizations sift through terabytes and petabytes of data to inform decision makers to help them make faster strategic and tactical decisions, and it is a key component in data analytics.

Figure 14.3Feedback Loop Processing

14-1cVariety

In a Big Data context, variety refers to the vast array of formats and structures in which the data may be captured. Data can be considered to be structured, unstructured, or semistructured. **Structured data** is data that has been organized to fit a predefined data model. **Unstructured data** is data that is not organized to fit into a predefined data model. Semistructured data combines elements of both—some parts of the data fit a predefined model while other parts do not. Relational databases rely on structured data. A data model is created by the database designer based on the business rules, as discussed in Chapter 4. As data enters the database, the data is decomposed and routed for storage in the corresponding tables and columns as defined in the data model. Although much of the transactional data that organizations use works well in a structured environment, most of the data in the world is semistructured or unstructured. Unstructured data includes maps, satellite images, emails, texts, tweets, videos, transcripts, and a whole host of other data forms. Over the decades that the relational model has been dominant, relational databases have evolved to address some forms of unstructured data. For example, most large-scale RDBMSs support a binary large object (BLOB) data type that allows the storage of unstructured objects like audio, video, and graphic data as a single, atomic value. One problem with BLOB data is that the semantic value of the data, the meaning that the object conveys, is inaccessible and uninterpretable by data processing.

Big Data requires that the data be captured in whatever format it naturally exists, without any attempt to impose a data model or structure to the data. This is one of the key differences between processing data in a relational database and Big Data processing. Relational databases impose a structure on the data when the data is captured and stored. Big Data processing imposes a structure on the data as needed for applications as a part of retrieval and processing. One advantage to providing structure during retrieval and processing is the flexibility of being able to structure the data in different ways for different applications.

14-1d Other Characteristics

Characterizing Big Data with the 3 Vs is fairly standard. However, as the industry matures, other characteristics have been put forward as being equally important. Keeping with the spirit of the 3 Vs, these additional characteristics are typically presented as additional *Vs*. **Variability** refers to the changes in the meaning of the data based on context. While *variety* and *variability* are similar terms, they mean distinctly different things in Big Data. Variety is about differences in structure. Variability is about differences in meaning. Variability is especially relevant in areas such as sentiment analysis that attempt to understand the meanings of words. **Sentiment analysis** is a method of text analysis that attempts to determine if a statement conveys a positive, negative, or neutral attitude about a topic. For example, the statements, “I just bought a new smartphone—I love it!” and “The screen on my new smartphone shattered the first time I dropped it—I love it!” In the first statement the presence of the phrase “I love it” might help an algorithm correctly interpret the statement as expressing a positive attitude. However, the second statement uses sarcasm to express a negative attitude so the presence of the phrase “I love it” may cause the analysis to interpret the meaning of the phrase incorrectly.

Veracity refers to the trustworthiness of the data. Can decision makers reasonably rely on the accuracy of the data and the information generated from it? This is especially pertinent given the automation of data capture and some of the analysis. Uncertainty about the data can arise from several causes, such as having to capture only selected portions of the data due to high velocity. Also, in terms of sentiment analysis, customers' opinions and preferences can change over time, so comments at one point in time might not be suitable for action at another point in time.

Increasingly, value is being touted as an important characteristic for Big Data. **Value**, also called *viability*, refers to the degree to which the data can be analyzed to provide meaningful information that can add value to the organization. Just because a set of data *can* be captured does not mean that it *should* be captured. Only data that can form the basis for analysis that has the potential to impact organizational behavior should be included in a company's Big Data efforts.

The final characteristic of Big Data is visualization. **Visualization** is the ability to graphically present the data in such a way as to make it understandable. Volumes of data can leave decision makers awash in facts but with little understanding of what the facts mean. Visualization is a way of presenting the facts so that decision makers can comprehend the meaning of the information to gain insights.

An argument could be made that these additional Vs are not necessarily characteristics of Big Data; or, perhaps more accurately, they are not characteristics of *only* Big Data. Veracity of data is an issue with even the smallest data store, which is why data management is so important in relational databases. Value of data also applies to traditional, structured data in a relational database. One of the keys to data modeling is that only the data that is of interest to the users should be included in the data model. Data that is not of value should not be recorded in any data store—Big Data or not. Visualization was discussed and illustrated at length in Chapter 13 as an important tool in working with data warehouses, which are often maintained as structured data stores in RDBMS products. The important thing to remember is that these characteristics that play an important part in working with data in the relational model are universal and also apply to Big Data.

Big Data represents a new wave in data management challenges, but it does not mean that relational database technology is going away. Structured data that depends on ACID transactions, as discussed in Chapter 10, will always be critical to business operations. Relational databases are still the best way for storing and managing this type of data. What has changed is that now, for the first time in decades, relational databases are not necessarily the best way for storing and managing *all* of an organization's data. Since the rise of the relational model, the decision for data managers when faced with new storage requirements was not whether to use a relational database, but rather which relational DBMS to use. Now, the decision of whether to use a relational database at all is a real question. This has led to **polyglot persistence**—the coexistence of a variety of data storage and management technologies within an organization's infrastructure. Scaling up, as discussed, is often considered a viable option as relational databases grow. However, it has practical limits and cost considerations that make it infeasible for many Big Data installations. Scaling out into clusters based on low-cost, commodity servers is the dominant approach that organizations are currently pursuing for Big Data management. As a result, new technologies not based on the relational model have been developed.

14-2Hadoop

Big Data requires a different approach to distributed data storage that is designed for large-scale clusters. Although other implementation technologies are possible, Hadoop has become the de facto standard for most Big Data storage and processing. Hadoop is not a database. Hadoop is a Java-based framework for distributing and processing very large data sets across clusters of computers. While the Hadoop framework includes many parts, the two most important components are the Hadoop Distributed File System (HDFS) and MapReduce. HDFS is a low-level distributed file processing system, which means that it can be used directly for data storage. MapReduce is a programming model that supports processing large data sets in a highly parallel, distributed manner. While it is possible to use HDFS and MapReduce separately, the two technologies complement each other so that they work better together as a Hadoop system. Hadoop was engineered specifically to distribute and process enormous amounts of data across vast clusters of servers.

14-2aHDFS

The **Hadoop Distributed File System (HDFS)** approach to distributing data is based on several key assumptions:

- *High volume*. The volume of data in Big Data applications is expected to be in terabytes, petabytes, or larger. Hadoop assumes that files in the HDFS will be extremely large. Data in the HDFS is organized into physical blocks, just as in other file storage. For example, on a typical personal computer, file storage is organized into blocks that are often 512 bytes in size, depending on the hardware and operating system involved. Relational databases often aggregate these into database blocks. By default, Oracle organizes data into 8-KB physical blocks. Hadoop, on the other hand, has a default block size of 64 MB (8,000 times the size of an Oracle block!), and it can be configured to even larger values. As a result, the number of blocks per file is greatly reduced, simplifying the metadata overhead of tracking the blocks in each file.

- *Write-once, read-many*. Using a write-once, read-many model simplifies concurrency issues and improves overall data throughput. Using this model, a file is created, written to the file system, and then closed. Once the file is closed, changes cannot be made to its contents. This improves overall system performance and works well for the types of tasks performed by many Big Data applications. Although existing contents of the file cannot be changed, recent advancements in the HDFS allow for files to have new data appended to the end of the file. This is a key advancement for NoSQL databases because it allows for database logs to be updated.
- *Streaming access*. Unlike transaction processing systems where queries often retrieve small pieces of data from several different tables, Big Data applications typically process entire files. Instead of optimizing the file system to randomly access individual data elements, Hadoop is optimized for batch processing of entire files as a continuous stream of data.
- *Fault tolerance*. Hadoop is designed to be distributed across thousands of low-cost, commodity computers. It is assumed that with thousands of such devices, at any point in time, some will experience hardware errors. Therefore, the HDFS is designed to replicate data across many different devices so that when one device fails, the data is still available from another device. By default, Hadoop uses a replication factor of three, meaning that each block of data is stored on three different devices. Different replication factors can be specified for each file, if desired.

Hadoop uses several types of nodes. A *node* is just a computer that performs one or more types of tasks within the system. Within the HDFS, there are three types of nodes: the client node, the name node, and one or more data nodes, as depicted in Figure 14.4.

Figure 14.4Hadoop Distributed File System (HDFS)

Data nodes store the actual file data within the HDFS. Recall that files in HDFS are broken into blocks and are replicated to ensure fault tolerance. As a result, each block is duplicated on more than one data node. Figure 14.4 shows the default replication factor of three, so each block appears on three data nodes.

The name node contains the metadata for the file system. There is typically only one name node within a HDFS cluster. The metadata is designed to be small, simple, and easily recoverable. Keeping the metadata small allows the name node to hold all of the metadata in memory to reduce disk accesses and improve system performance. This is important because there is only one name node so contention for the name node is minimized. The metadata is composed primarily of the name of each file, the block numbers that comprise each file, and the desired replication factor for each file. The client node makes requests to the file system, either to read files or to write new files, as needed to support the user application.

When a client node needs to create a new file, it communicates with the name node. The name node:

- Adds the new file name to the metadata.
- Determines a new block number for the file.
- Determines a list of which data nodes the block will be stored.
- Passes that information back to the client node.

The client node contacts the first data node specified by the name node and begins writing the file on that data node. At the same time, the client node sends the data node the list of other data nodes that will be replicating the block. As the data is received from the client node, the data node contacts the next data node in the list and begins sending the data to this node for replication. This second data node then contacts the next data node in the list and the process continues with the data being streamed across all of the data nodes that are storing the block. Once the first block is written, the client node can get another block number and list of data nodes from the name node for the next block. When the entire file has been written, the client node informs the name node that the file is closed. It is important to note that at no time was any of the data file actually transmitted to the name node. This helps to reduce the data flow to the name node to avoid congestion that could slow system performance.

Similarly, if a client node needs to read a file, it contacts the name node to request the list of blocks associated with that file and the data nodes that hold them. Given that each block may appear in many data nodes, for each block, the client attempts to retrieve the block from the data node that is closest to it on the network. Using this information, the client node reads the data directly from each of those nodes.

Periodically, each data node communicates with the name node. The data nodes send block reports and heartbeats. A **block report** is sent every 6 hours and informs the name node of which blocks are on that data node. Heartbeats are sent every 3 seconds. A **heartbeat** is used to let the name node know that the data node is still available. If a data node experiences a fault, due to hardware failure, power outage, etc., then the name node will not receive a heartbeat from that data node. As a result, the name node knows not to include that data node in lists to client nodes for reading or writing files. If the lack of a heartbeat from a data node causes a block to have fewer than the desired number of replicas, the name node can have a "live" data node initiate replicating the block on another data node.

Taken together, the components of the HDFS produce a powerful, yet highly specialized distributed file system that works well for the specialized processing requirements of Big Data applications. Next, we will consider how MapReduce provides data processing to complement data storage of HDFS.

14-2b MapReduce

MapReduce is the computing framework used to process large data sets across clusters. Conceptually, MapReduce is easy to understand and follows the principle of *divide and conquer*. MapReduce takes a complex task, breaks it down into a collection of smaller subtasks, performs the subtasks all at the same time, and then combines the result of each subtask to produce a final result for the original task. As the name implies, it is a combination of a map function and a reduce function. A **map** function takes a collection of data and sorts and filters the data into a set of key-value pairs. The map function is performed by a program called a **mapper**. A **reduce** function takes a collection of key-value pairs, all with the same key value, and summarizes them into a single result. The reduce function is performed by a program called a **reducer**. Recall that Hadoop is a Java-based platform, therefore map and reduce functions are written as detailed, procedure-oriented Java programs.

Figure 14.5 provides a simple, conceptual illustration of MapReduce that determines the total number of units of each product that has been sold. The original data in Figure 14.5 is stored as key-value pairs, with the invoice number as the key and the remainder of the invoice data as a value. Remember, the data in Hadoop data storage is not a relational database so the data is not separated into tables and there is no form of normalization that ensures that each fact is stored only once. Therefore, there is a great deal of duplication of data in the original data store. Note that even in the very small subset of data that is shown in Figure 14.5, redundant data is kept for customer 10011, Leona Dunne. In the figure, map functions parse each invoice to find data about the products sold on that invoice. The result of the map function is a new list of key-value pairs in which the product code is the key and the line units are the value. The reduce function then takes that list of key-value pairs and combines them by summing the values associated with each key (product code) to produce the summary result.

Figure 14.5MapReduce

As previously stated, the data sets used in Big Data applications are extremely large. Transferring entire files from multiple nodes to a central node for processing would require a tremendous amount of network bandwidth, and place an incredible processing burden on the central node. Therefore, instead of the computational program retrieving the data for processing in a central location, copies of the program are “pushed” to the nodes containing the data to be processed. Each copy of the program produces results that are then aggregated across nodes and sent back to the client. This mirrors the distribution of data in the HDFS. Typically, the Hadoop framework will distribute a mapper for each block on each data node that must be processed. This can lead to a very large number of mappers. For example, if 1 TB of data is to be processed and the HDFS is using 64-MB blocks, that yields over 15,000 mapper programs. The number of reducers is configurable by the user, but best practices suggest about one reducer per data node.

Note

Best practices suggest that the number of mappers on a given node should be kept to 100 or less. However, there are cases of applications with simple map functions running as many as 300 mappers on a given node with satisfactory performance. Clearly, much depends on the computing resources available at each node.

The implementation of MapReduce complements the structure of the HDFS, which is an important reason why they work so well together. Just as the HDFS structure is composed of a name node and several data nodes, MapReduce uses a **job tracker** (the actual name of the program is JobTracker) and several **task trackers** (the programs are named TaskTrackers). The job tracker acts as a central control for MapReduce processing and it normally exists on the same server that is acting as the name node. Task tracker programs reside on the data nodes. One important feature of the MapReduce framework is that the user must write the Java code for the map and reduce functions, and must specify the input and output files to be read and written for the job that is being submitted. However, the job tracker will take care of locating the data, determining which nodes to use, dividing the job into tasks for the nodes, and managing failures of the nodes. All of this is done automatically without user intervention. When a user submits a MapReduce job for processing, the general process is as follows:

1. A client node (client application) submits a MapReduce job to the job tracker.
2. The job tracker communicates with the name node to determine which data nodes contain the blocks that should be processed for this job.
3. The job tracker determines which task trackers are available for work. Each task tracker can handle a set number of tasks. Remember, many MapReduce jobs from different users can be running on the Hadoop system simultaneously, so a data node may contain data that is being processed by multiple mappers from different jobs all at the same time. Therefore, the task tracker on that node might be busy running mappers for other jobs when this new request arrives. Because the data is replicated on multiple nodes, the job tracker may be able to select from multiple nodes for the same data.
4. The job tracker then contacts the task trackers on each of those nodes to begin mappers and reducers to complete that node's portion of the task.
5. The task tracker creates a new JVM (Java virtual machine) to run the map and reduce functions. This way, if a function fails or crashes, the entire task tracker is not halted.
6. The task tracker sends heartbeat messages to the job tracker to let the job tracker know that the task tracker is still working on the job (and about the nodes availability for more jobs).
7. The job tracker monitors the heartbeat messages to determine if a task manager has failed. If so, the job tracker can reassign that portion of the task to another node.
8. When the entire job is finished, the job tracker changes status to indicate that the job is completed.
9. The client node periodically queries the job tracker until the job status is completed.

The Hadoop system uses batch processing. **Batch processing** is when a program runs from beginning to end, either completing the task or halting with an error, without any interaction with the user. Batch processing is often used when the computing task requires an extended period of time or a large portion of the system's processing capacity. Businesses often use batch processing to run year-end financial reports in the evenings when systems are often idle, and universities might use batch processing for student fee payment processing. Batch processing is not bad, but it has limitations. As a result, a number of complementary programs have been developed to improve the integration of Hadoop within the larger IT infrastructure. The next section discusses some of these programs.

14-2c Hadoop Ecosystem

Hadoop is widely used by organizations tapping into the potential of analyzing extremely large data sets. Unfortunately, because Hadoop is a very low-level tool requiring considerable effort to create, manage, and use, it presents quite a few obstacles. As a result, a host of related applications have grown up around Hadoop to attempt to make it easier to use and more accessible to users who are not skilled at complex Java programming. Figure 14.6 shows examples of some of these types of applications. Most organizations that use Hadoop also use a set of other related products that interact and complement each other to produce an entire ecosystem of applications and tools. Like any ecosystem, the interconnected pieces are constantly evolving and their relationships are changing, so it is a rather fluid situation. The following are some of the more popular components in a Hadoop ecosystem and how they relate to each other.

Figure 14.6A Sample of the Hadoop Ecosystem

MapReduce Simplification Applications

Creating MapReduce jobs requires significant programming skills. As the mapper and reducer programs become more complex, the skill requirements increase and the time to produce the programs becomes significant. These skills are beyond the capabilities of most data users. Therefore, applications to simplify the process of creating MapReduce jobs have been developed. Two of the most popular are Hive and Pig.

Hive is a data warehousing system that sits on top of HDFS. It is not a relational database, but it supports its own SQL-like language, called HiveQL, that mimics SQL commands to run ad hoc queries. HiveQL commands are processed by the Hive query engine into sets of MapReduce jobs. As a result, the underlying processing tends to be batch-oriented, producing jobs that are very scalable over extremely large sets of data. However, the batch nature of the jobs makes Hive a poor choice for jobs that only require a small subset of data to be returned very quickly.

Pig is a tool for compiling a high-level scripting language, named Pig Latin, into MapReduce jobs for executing in Hadoop. In concept it is similar to Hive in that it provides a means of producing MapReduce jobs without the burden of low-level Java programming. The primary difference is that Pig Latin is a scripting language, which means it is procedural, while HiveQL, like SQL, is declarative. Declarative languages allow the user to specify what they want, not how to get it. This is very useful for query processing. Procedural languages require the user to specify how the data is to be manipulated. This is very useful for performing data transformations. As a result, Pig is often used for producing data pipeline tasks that transform data in a series of steps. This is often seen in ETL processes as described in Chapter 13.

Data Ingestion Applications

One challenge faced by organizations that are taking advantage of Hadoop's massive data storage and data processing capabilities is the issue of actually getting data from their existing systems into the Hadoop cluster. To simplify this task, applications have been developed to “ingest” or gather this data into Hadoop.

Flume is a component for ingesting data into Hadoop. It is designed primarily for harvesting large sets of data from server log files, like clickstream data from web server logs. It can be configured to import the data on a regular schedule or based on specified events. In addition to simply bringing the data into Hadoop, Flume contains a simple query processing component so the possibility exists of performing some transformations on the data as it is being harvested. Typically, Flume would move the data into the HDFS, but it can also be configured to input the data directly into another component of the Hadoop ecosystem named HBase.

Sqoop is a more recent addition to the Hadoop ecosystem. It is a tool for converting data back and forth between a relational database and the HDFS. The name Sqoop (pronounced, "scoop," as in a scoop of ice cream) is an amalgam of "SQL-to-Hadoop." In concept, Sqoop is similar to Flume in that it provides a way of bringing data into the HDFS. However, while Flume works primarily with log files, Sqoop works with relational databases such as Oracle, MySQL, and SQL Server. Further, while Flume operates in one direction only, Sqoop can transfer data in both directions—into and out of HDFS. When transferring data from a relational database into HDFS, the data is imported one table at a time with the process reading the table row-by-row. This is done in a highly parallelized manner using MapReduce, so the contents of the table will usually be distributed into several files with the rows stored in a delimited format. Once the data has been imported into HDFS, it can be processed by MapReduce jobs or using Hive. The resulting data can then be exported from HDFS back to the relational database, most often a traditional data warehouse.

Direct Query Applications

Direct query applications attempt to provide faster query access than is possible through MapReduce. These applications interact with HDFS directly, instead of going through the MapReduce processing layer.

HBase is a column-oriented NoSQL database designed to sit on top of the HDFS. One of HBase's primary characteristics is that it is highly distributed and designed to scale out easily. It does not support SQL or SQL-like languages, relying instead on lower-level languages such as Java for interaction. The system does not rely on MapReduce jobs, so it avoids the delays caused by batch processing, making it more suitable for fast processing involving smaller subsets of the data. HBase is very good at quickly processing sparse data sets. HBase is one of the more popular components of the Hadoop ecosystem, and is used by Facebook for its messaging system. Column-oriented databases will be discussed in more detail in the next section.

Impala was the first SQL-on-Hadoop application. It was produced by Cloudera as a query engine that supports SQL queries that pull data directly from HDFS. Prior to Impala, if an organization needed to make data from Hadoop available to analysts through an SQL interface, data would be extracted from HDFS and imported into a relational database. With Impala, analysts can write SQL queries directly against the data while it is still in HDFS. Impala makes heavy use of in-memory caching on data nodes. It is generally considered an appropriate tool for processing large amounts of data into a relatively small result set.

Note

Other than Impala, each of the components of the Hadoop ecosystem described in this section are all open-source, top-level projects of the Apache Software Foundation. More information on each of these projects and many others is available at www.apache.org.

14-3 NoSQL

NoSQL is the unfortunate name given to a broad array of nonrelational database technologies that have developed to address the challenges represented by Big Data. The name is unfortunate in that it does not describe what the NoSQL technologies are, but rather what they are not. In fact, the name also does a poor job of explaining what the technologies are not! The name was chosen as a Twitter hashtag to simplify coordinating a meeting of developers to discuss ideas about the nonrelational database technologies that were being developed by organizations like Google, Amazon, and Facebook to deal with the problems they were encountering as their data sets reached enormous sizes. The term “NoSQL” was never meant to imply that products in this category should never include support for SQL. In fact, many such products support query languages that mimic SQL in important ways. Although no one has yet produced a NoSQL system that implements standard SQL, given the large base of SQL users, the appeal of creating such a product is obvious. More recently, some industry observers have tried to interject that “NoSQL” could stand for “Not Only SQL.” In fact, if the requirement to be considered a NoSQL product were simply that languages beyond SQL are supported, then all of the traditional RDBMS products such as Oracle, SQL Server, MySQL, and MS Access would all qualify. Regardless, you are better off focusing on understanding the array of technologies to which the term refers than worrying about the name itself.

There are literally hundreds of products that can be considered as being under the broadly defined term NoSQL. Most of these fit roughly into one of four categories: key-value data stores, document databases, column-oriented databases, and graph databases. Table 14.2 shows some popular NoSQL databases of each type. Although not all NoSQL databases have been produced as open-source software, most have been. As a result, NoSQL databases are generally perceived as a part of the open-source movement. Accordingly, they also tend to be associated with the Linux operating system. It makes sense from a cost standpoint that, if an organization is going to create a cluster containing tens of thousands of nodes, the organization does not want to purchase licenses for Windows or Mac OS for all of those nodes. The preference is to use a platform, like Linux, that is freely available and highly customizable. Therefore, most of the NoSQL products run only in a Linux or Unix environment. The following sections discuss each of the major NoSQL approaches.

Table 14.2

NoSQL Databases

NoSQL Category	Example Databases
Key-value database	• Dynamo • Riak • Redis • Voldemort
Document databases	• MongoDB • CouchDB • OrientDB • RavenDB
Column-oriented databases	• HBase • Cassandra • Hypertable
Graph databases	• Neo4J • ArangoDB • GraphBase

14-3a Key-Value Databases

Key-value (KV) databases are conceptually the simplest of the NoSQL data models. A KV database is a NoSQL database that stores data as a collection of key-value pairs. The key acts as an identifier for the value. The value can be anything such as text, an XML document, or an image. The database does not attempt to understand the contents of the value component or its meaning—the database simply stores whatever value is provided for the key. It is the job of the applications that use the data to understand the meaning of the data in the value component. There are no foreign keys; in fact, relationships cannot be tracked among keys at all. This greatly simplifies the work that the DBMS must perform, making KV databases extremely fast and scalable for basic processing.

Key-value pairs are typically organized into "buckets." A **bucket** can roughly be thought of as the KV database equivalent of a table. A bucket is a logical grouping of keys. Key values must be unique within a bucket, but they can be duplicated across buckets. All data operations are based on the bucket plus the key. In other words, it is not possible to query the data based on anything in the value component of the key-value pair. All queries are performed by specifying the bucket and key. Operations on KV databases are rather simple—only *get*, *store*, and *delete* operations are used. *Get* or *fetch* is used to retrieve the value component of the pair. *Store* is used to place a value in a key. If the bucket + key combination does not exist, then it is added as a new key-value pair. If the bucket + key combination does exist, then the existing value component is replaced with the new value. *Delete* is used to remove a key-value pair. Figure 14.7 shows a customer bucket with three key-value pairs. Since the KV model does not allow queries based on data in the value component, it is not possible to query for a key-value pair based on customer last name, for example. In fact, the KV DBMS does not even know that there is such a thing as a customer last name because it does not understand the content of the value component. An application could issue a *get* command to have the KV DBMS return the key-value pair for bucket customer and key 10011, but it would be up to the application to know how to parse the value component to find the customer's last name, first name, and other characteristics. (One important note about Figure 14.7: Be aware that although key-value pairs appear in tabular form in the figure, the tabular format is just a convenience to help visually distinguish the components. Actual key-value pairs are not stored in a table-like structure.)

Figure 14.7 Key-Value Database Storage

14-3b Document Databases

Document databases are conceptually similar to key-value databases, and they can almost be considered a subtype of KV databases. A document database is a NoSQL database that stores data in tagged documents in key-value pairs. Unlike a KV database where the value component can contain any type of data, a document database always stores a document in the value component. The document can be in any encoded format, such as XML, **JSON (JavaScript Object Notation)**, or **BSON (Binary JSON)**. Another important difference is that while KV databases do not attempt to understand the content of the value component, document databases do. Tags are named portions of a document. For example, a document may have tags to identify which text in the document represents the title, author, and body of the document. Within the body of the document, there may be additional tags to indicate chapters and sections. Despite the use of tags in documents, document databases are considered schema-less, that is, they do not impose a predefined structure on the data that is stored. For a document database, being schema-less means that although all documents have tags, not all documents are required to have the same tags, so each document can have its own structure. The tags in a document database are extremely important because they are the basis for most of the additional capabilities that document databases have over KV databases. Tags inside the document are accessible to the DBMS, which makes sophisticated querying possible.

Just as KV databases group key-value pairs into logical groups called *buckets*, document databases group documents into logical groups called *collections*. While a document may be retrieved by specifying the collection and key, it is also possible to query based on the contents of tags. For example, Figure 14.8 represents the same data from Figure 14.7, but in a tagged format for a document database. Because the DBMS is aware of the tags within the documents, it is possible to write queries that retrieve all of the documents where the Balance tag has the value 0. Document databases even support some aggregate functions such as summing or averaging balances in queries.

Figure 14.8Document Database Tagged Format

Document databases tend to operate on an implied assumption that a document is relatively self-contained, not a fragment of the data about a given topic. Relational databases decompose complex data in the business environment into a set of related tables. For example data about orders may be decomposed into customer, invoice, line, and product tables. A document database would expect all of the data related to an order to be in a single order document. Therefore, each order document in an *Orders* collection would contain data on the customer, the order itself, and the products purchased in that order all as a single self-contained document. Document databases do not store relationships as perceived in the relational model and generally have no support for join operations.

14-3cColumn-Oriented Databases

The term *column-oriented database* can refer to two different sets of technologies that are often confused with each other. In one sense, column-oriented database or columnar database can refer to traditional, relational database technologies that use **column-centric storage** instead of **row-centric storage**. Relational databases present data in logical tables; however, the data is actually stored in data blocks containing rows of data. All of the data for a given row is stored together in sequence with many rows in the same data block. If a table has many rows of data, the rows will be spread across many data blocks. Figure 14.9 illustrates a relational table with 10 rows of data that is physically stored across five data blocks. Row-centric storage minimizes the number of disk reads necessary to retrieve a row of data. Retrieving one row of data requires accessing just one data block, as shown in Figure 14.9. Remember, in transactional systems, normalization is used to decompose complex data into related tables to reduce redundancy and to improve the speed of rapid manipulation of small sets of data. These manipulations tend to be row-oriented, so row-oriented storage works very well. However, in queries that retrieve a small set of columns across a large set of rows, a large number of disk accesses is required. For example, a query that wants to retrieve only the city and state of every customer will have to access every data block that contains a customer row to retrieve that data. In Figure 14.9, that would mean accessing five data blocks to get the city and state of every customer. A column-oriented or columnar database stores the data in blocks by column instead of by row. A single customer's data will be spread across several blocks, but all of the data from a single column will be in just a few blocks. In Figure 14.9, all of the city data for customers will be stored together, just as all of the state data will be stored together. In that case, retrieving the city and state for every customer might require accessing only two data blocks. This type of column-centric storage works very well for databases that are primarily used to run queries over few columns but many rows, as is done in many reporting systems and data warehouses. Though Figure 14.9 shows only a few rows and data blocks, it is easy to imagine that the gains would be significant if the table size grew to millions or billions of rows across hundreds of thousands of data blocks. At the same time, column-centric storage would be very inefficient for processing transactions since insert, update, and delete activities would be very disk intensive. It is worth noting that column-centric storage can be achieved within relational database technology, meaning that it still requires structured data and has the advantage of supporting SQL for queries.

Figure 14.9 Comparison of Row-Centric and Column-Centric Storage

The other use of the term *column-oriented database*, also called column family database, is to describe a type of NoSQL database that takes the concept of column-centric storage beyond the confines of the relational model. As NoSQL databases, these products do not require the data to conform to predefined structures nor do they support SQL for queries. This database model originated with Google's BigTable product. Other column-oriented database products include HBase, described earlier, and Cassandra. Cassandra began as a project at Facebook, but Facebook released it to the open-source community, which has continued to develop Cassandra into one of the most popular column-oriented databases. A **column family database** is a NoSQL database that organizes data in key-value pairs with keys mapped to a set of columns in the value component. While column family databases use many of the same terms as relational databases, the terms don't mean quite the same things. Fortunately, the column family databases are conceptually simple and are conceptually close enough to the relational model that your understanding of the relational model can help you understand the column family model. A column is a key-value pair that is similar to a cell of data in a relational database. The key is the name of the column, and the value component is the data that is stored in that column. Therefore, “cus_lname: Ramas” is a column; *cus_lname* is the name of the column, and *Ramas* is the data value in the column. Similarly, “cus_city: Nashville” is another column, with *cus_city* as the column name and *Nashville* as the data value.

Note

Even though column family databases do not (yet) support standard SQL, Cassandra developers have created a Cassandra query language (CQL). It is similar to SQL in many respects and is one of the more compelling reasons for adopting Cassandra.

As more columns are added, it becomes clear that some columns form natural groups, such as cus_fname, cus_lname, and cus_initial which would logically group together to form a customer's name. Similarly, cus_street, cus_city, cus_state, and cus_zip would logically group together to form a customer's address. These groupings are used to create super columns. A **super column** is a group of columns that are logically related. Recall the discussion in Chapter 4 about simple and composite attributes in the entity relationship model. In many cases, super columns can be thought of as the composite attribute and the columns that compose the super column as the simple attributes. Just as all simple attributes do not have to belong to a composite attribute, not all columns have to belong to a super column. Although this analogy is helpful in many contexts, it is not perfect. It is possible to group columns into a super column that logically belongs together for application processing reasons but does not conform to the relational idea of a composite attribute.

Row keys are created to identify objects in the environment. All of the columns or super columns that describe these objects are grouped together to create a **column family**; therefore, a column family is conceptually similar to a table in the relational model. While a column family is similar in concept to a relational table, Figure 14.10shows that it is structurally very different. Notice in Figure 14.10 that each row key in the column family can have different columns.

Figure 14.10Column Family Database

Note

A column family can be composed of columns or super columns, but it cannot contain both.

14-3dGraph Databases

A **graph database** is a NoSQL database based on graph theory to store data about relationship-rich environments. Graph theory is a mathematical and computer science field that models relationships, or edges, between objects called nodes. Modeling and storing data about relationships is the focus of graph databases. Graph theory is a well-established field of study going back hundreds of years. As a result, creating a database model based on graph theory immediately provides a rich source for algorithms and applications that have helped graph databases gain in sophistication very quickly. Since it also happens that much of the data explosion over the last decade has involved data that is relationship-rich, graph databases have been poised to experience significant interest in the business environment.

Interest in graph databases originated in the area of social networks. Social networks include a wide range of applications beyond the typical Facebook, Twitter, and Instagram that immediately come to mind. Dating websites, knowledge management, logistics and routing, master data management, and identity and access management are all areas that rely heavily on tracking complex relationships among objects. Of course, relational databases support relationships too. One of the great advances of the relational model was that relationships are easy to maintain. A relationship between a customer and an agent is as easy to implement in the relational model as adding a foreign key to create a common attribute, and the customer and agent rows are related by having the same value in the common attributes. If the customer changes to a different agent, then simply changing the value in the foreign key will change the relationship between the rows to maintain the integrity of the data. The relational model does all of these things very well. However, what if we want a “like” option so customers can “like” agents on our website? This would require a structural change to the database to add a new foreign key to support this second relationship. Next, what if the company wants to allow customers on its website to “friend” each other so a customer can see which agents their friends like, or the friends of their friends? In social networking data, there can be dozens of different relationships among individuals that need to be tracked, and often the relationships are tracked many layers deep (e.g., friends, friends of friends, friends of friends of friends, etc.). This results in a situation where the relationships become just as important as the data itself. This is the area where graph databases shine.

The primary components of graph databases are nodes, edges, and properties, as shown in Figure 14.11. A node corresponds to the idea of a relational entity instance. The **node** is a specific instance of something we want to keep data about. Each node (circle) in Figure 14.10 represents a single agent. Properties are like attributes; they are the data that we need to store about the node. All agent nodes might have properties like first name and last name, but all nodes are not required to have the same properties. An **edge** is a relationship between nodes. Edges (shown as arrows in Figure 14.10) can be in one direction, or they can be bidirectional. For example, in Figure 14.11, the *friends* relationships are bidirectional, but the *likes* relationships are not. Note that edges can also have **properties**. In Figure 14.11 the date on which customer Alfred Ramas *liked* agent Alex Alby is recorded in the graph database. A query in a graph database is called a **traversal**. Instead of *querying the database*, the correct terminology would be *traversing the graph*. Graph databases excel at traversals that focus on relationships between nodes, such as shortest path and degree of connectedness.

Figure 14.11Graph Database Representation

Graph database share some characteristics with other NoSQL databases in that graph databases do not force data to fit predefined structures, do not support SQL, and are optimized to provide velocity of processing, at least for relationship-intensive data. However, other key characteristics do not apply to graph databases. Graph databases do not scale out very well to clusters. The other NoSQL database models achieve clustering efficiency by making each piece of data relatively independent. That allows a key-value pair to be stored on one node in the cluster without the DBMS needing to associate it with another key-value pair that may be on a different node on the cluster. The greater the number of nodes involved in a data operation, the greater the need for coordination and centralized control of resources. Separating independent pieces of data, often called *shards*, across nodes in the cluster is what allows NoSQL databases to scale out so effectively. Graph databases specialize in highly related data, not independent pieces of data. As a result, graph databases tend to perform best in centralized or lightly clustered environments, similar to relational databases.

14-3eNewSQL Databases

Relational databases are the mainstay of organizational data, and NoSQL databases do not attempt to replace them for supporting line-of-business transactions. These transactions that support the day-to-day operations of business rely on ACID-compliant transactions and concurrency control, as discussed in Chapter 10. NoSQL databases (except graph databases that focus on specific relationship-rich domains) are concerned with the distribution of user-generated and machine-generated data over massive clusters. NewSQL databases try to bridge the gap between RDBMS and NoSQL. **NewSQL** databases attempt to provide ACID-compliant transactions over a highly distributed infrastructure. NewSQL databases are the latest technologies to appear in the data management arena to address

Big Data problems. As a new category of data management products, NewSQL databases have not yet developed a track record of success and have been adopted by relatively few organizations.

NewSQL products, such as ClusterixDB and NuoDB, are designed from scratch as hybrid products that incorporate features of relational databases and NoSQL databases.

Like RDBMS, NewSQL databases support:

- SQL as the primary interface
- ACID-compliant transactions

Similar to NoSQL, NewSQL databases also support:

- Highly distributed clusters
- Key-value or column-oriented data stores

As expected, no technology can perfectly provide the advantages of both RDBMS and NoSQL, so NewSQL has disadvantages (the CAP theorem still applies!). Principally, the disadvantages that have been discovered center around NewSQL's heavy use of in-memory storage. Critics point to the fact that this can jeopardize the “durability” component of ACID. Further, the ability to handle vast data sets can be impacted by the reliance on in-memory structures because there are practical limits to the amount of data that can be held in memory. Although in theory NewSQL databases should be able to scale out significantly, in practice little has been done to scale beyond a few dozen data nodes. While this is a marked improvement over traditional RDBMS distribution, it is far from the hundreds of nodes used by NoSQL databases.

Capturing data, in and of itself, is not the goal of data management. As discussed earlier, the data must add value to the organization. The data must help the organization to meet the needs of customers and provide value to shareholders. Data analysis is the process of turning the data into information that adds insights that enable data-based decisions. The next section will describe the complexity of that process.

14-4Data Analytics

Data analytics is a subset of business intelligence (BI) functionality that encompasses a wide range of mathematical, statistical, and modeling techniques with the purpose of extracting knowledge from data. Data analytics is used at all levels within the BI framework, including queries and reporting, monitoring and alerting, and data visualization. Hence, data analytics is a “shared” service that is crucial to what BI adds to an organization. Data analytics represents what business managers really want from BI: the ability to extract actionable business insight from current events and foresee future problems or opportunities.

Data analytics discovers characteristics, relationships, dependencies, or trends in the organization's data, and then explains the discoveries and predicts future events based on the discoveries. In practice, data analytics is better understood as a continuous spectrum of knowledge acquisition that goes from *discovery* to *explanation* to *prediction*. The outcomes of data analytics then become part of the information framework on which decisions are built. Data analytics tools can be grouped into two separate (but closely related and often overlapping) areas:

- **Explanatory analytics** focuses on discovering and explaining data characteristics and relationships based on existing data. Explanatory analytics uses statistical tools to formulate hypotheses, test them, and answer the *how* and *why* of such relationships—for example, how do past sales relate to previous customer promotions?
- **Predictive analytics** focuses on *predicting future data outcomes* with a high degree of accuracy. Predictive analytics uses sophisticated statistical tools to help the end user create advanced models that answer questions about future data occurrences—for example, what would next month's sales be based on a given customer promotion?
  You can think of explanatory analytics as explaining the past and present, while predictive analytics forecasts the future. However, you need to understand that both sciences work together; predictive analytics uses explanatory analytics as a stepping stone to create predictive models.

Data analytics has evolved over the years from simple statistical analysis of business data to dimensional analysis with OLAP tools, and then from data mining that discovers data patterns, relationships, and trends to its current status of predictive analytics. The next sections illustrate the basic characteristics of data mining and predictive analytics.

14-4aData Mining

Data mining refers to analyzing massive amounts of data to uncover hidden trends, patterns, and relationships; to form computer models to simulate and explain the findings; and then to use such models to support business decision making. In other words, data mining focuses on the discovery and explanation stages of knowledge acquisition.

To put data mining in perspective, look at the pyramid in Figure 14.12, which represents how knowledge is extracted from data. *Data* forms the pyramid base and represents what most organizations collect in their operational databases. The second level contains *information* that represents the purified and processed data. Information forms the basis for decision making and business understanding. *Knowledge* is found at the pyramid's apex and represents highly distilled information that provides concise, actionable business insight.

Figure 14.12Extracting Knowledge From Data

Current-generation data-mining tools contain many design and application variations to fit specific business requirements. Depending on the problem domain, data-mining tools focus on market niches such as banking, insurance, marketing, retailing, finance, and health care. Within a given niche, data-mining tools can use certain algorithms that are implemented in different ways and applied over different data. Despite the lack of precise standards, data mining consists of four general phases:

- Data preparation
- Data analysis and classification
- Knowledge acquisition
- Prognosis
  In the *data preparation phase*, the main data sets to be used by the data-mining operation are identified and cleansed of any data impurities. Because the data in the data warehouse is already integrated and filtered, the data warehouse usually is the target set for data-mining operations.

The *data analysis and classification phase* studies the data to identify common data characteristics or patterns. During this phase, the data-mining tool applies specific algorithms to find:

- Data groupings, classifications, clusters, or sequences
- Data dependencies, links, or relationships
- Data patterns, trends, and deviations

The *knowledge acquisition phase* uses the results of the data analysis and classification phase. During the knowledge acquisition phase, the data-mining tool (with possible intervention by the end user) selects the appropriate modeling or knowledge acquisition algorithms. The most common algorithms used in data mining are based on neural networks, decision trees, rules induction, genetic algorithms, classification and regression trees, memory-based reasoning, and nearest neighbor. A data-mining tool may use many of these algorithms in any combination to generate a computer model that reflects the behavior of the target data set.

Although many data-mining tools focus on the knowledge–discovery phase, others continue to the *prognosis phase*. In that phase, the data-mining findings are used to predict future behavior and forecast business outcomes. Examples of data-mining findings can be:

- Sixty-five percent of customers who did not use a particular credit card in the last six months are 88 percent likely to cancel that account.
- Eighty-two percent of customers who bought a 42-inch or larger LCD TV are 90 percent likely to buy an entertainment center within the next four weeks.
- If age < 30, income <= 25,000, credit rating < 3, and credit amount > 25,000, then the minimum loan term is 10 years.

The complete set of findings can be represented in a decision tree, a neural network, a forecasting model, or a visual presentation interface that is used to project future events or results. For example, the prognosis phase might project the likely outcome of a new product rollout or a new marketing promotion. Figure 14.13 illustrates the different phases of the data-mining process.

Figure 14.13 Data-Mining Phases

Because of the nature of the data-mining process, some findings might fall outside the boundaries of what business managers expect. For example, a data-mining tool might find a close relationship between a customer's favorite brand of soda and the brand of tires on the customer's car. Clearly, that relationship might not be held in high regard among sales managers. (In regression analysis, those relationships are commonly described by the label "idiot correlation.") Fortunately, data mining usually yields more meaningful results. In fact, data mining has proven helpful in finding practical relationships among data that help define customer buying patterns, improve product development and acceptance, reduce health care fraud, analyze stock markets, and so on.

Data mining can be run in two modes:

- *Guided*. The end user guides the data-mining tool step by step to explore and explain known patterns or relationships. In this mode, the end user decides what techniques to apply to the data.

- *Automated*. In this mode, the end user sets up the data-mining tool to run automatically and uncover hidden patterns, trends, and relationships. The data-mining tool applies multiple techniques to find significant relationships.

  As you learned in this section, data-mining methodologies focus on discovering and extracting information that describes and explains the data. For example, an explanatory model could create a customer profile that describes a given customer group. However, data mining can also be used as the basis to create advanced predictive data models. For example, a predictive model could be used to predict future customer behavior, such as a customer response to a target marketing campaign. The next section explains the use of predictive analytics in more detail.

14-4b Predictive Analytics

Although the term *predictive analytics* is used by many BI vendors to indicate many different levels of functionality, the promise of predictive analytics is very attractive for businesses looking for ways to improve their bottom line. Therefore, predictive analytics is receiving a lot of marketing buzz; vendors and businesses are dedicating extensive resources to this BI area. Predictive analytics refers to the use of advanced mathematical, statistical, and modeling tools to predict future business outcomes with high degrees of accuracy.

What is the difference between data mining and predictive analytics? As you learned earlier, data mining also has predictive capabilities. In fact, data mining and predictive analytics use similar and overlapping sets of tools, but with a slightly different focus. Data mining focuses on answering the "how" and "what" of *past* data, while predictive analytics focuses on creating actionable models to predict *future* behaviors and events. In some ways, you can think of predictive analytics as the next logical step after data mining; once you understand your data, you can use the data to predict future behaviors. In fact, most BI vendors are dropping the term *data mining* and replacing it with the more alluring term *predictive analytics*.

The origins of predictive analytics can be traced back to the banking and credit card industries. The need to profile customers and predict customer buying patterns in these industries was a critical driving force for the evolution of many modeling methodologies used in BI data analytics today. For example, based on your demographic information and purchasing history, a credit card company can use data-mining models to determine what credit limit to offer, what offers you are more likely to accept, and when to send those offers.

Predictive analytics received a big stimulus with the advent of social media. Companies turned to data mining and predictive analytics as a way to harvest the mountains of data stored on social media sites. Google was one of the first companies that offered targeted ads as a way to increase and personalize search experiences. Similar initiatives were used by all types of organizations to increase customer loyalty and drive up sales. Note the example of the airline and credit card industries and their frequent flyer and affinity card programs. Today, many organizations use predictive analytics to profile customers in an attempt to get and keep the right ones, which in turn will increase loyalty and sales.

Predictive analytics employs mathematical and statistical algorithms, neural networks, artificial intelligence, and other advanced modeling tools to create actionable predictive models based on available data. The algorithms used to build the predictive model are specific to certain types of problems and work with certain types of data. Therefore, it is important that the end user, who typically is trained in statistics and understands business, applies the proper algorithms to the problem in hand. However, thanks to constant technology advances, modern BI tools automatically apply multiple algorithms to find the optimum model.

Most predictive analytics models are used in areas such as customer relationships, customer service, customer retention, fraud detection, targeted marketing, and optimized pricing. Predictive analytics can add value to an organization in many different ways. For example, it can help optimize existing processes, identify hidden problems, and anticipate future problems or opportunities. However, predictive analytics is not the “secret sauce” to fix all business problems. Managers should carefully monitor and evaluate the value of predictive analytics models to determine their return on investment.

In Chapter 13, you learned about data warehouses and star schemas to model and store decision support data. In this chapter, you have added to that by exploring the vast stores of data that organizations are collecting in unstructured formats and the technologies that make that data available to users. Data analytics is used to extract knowledge from all of these sources of data—NoSQL databases, Hadoop data stores, and data warehouses—to provide decision support to all organizational users.

Summary

- Big Data is characterized by data of such volume, velocity, and/or variety that the relational model struggles to adapt to it. Volume refers to the quantity of data that must be stored. Velocity refers to both the speed at which data is entering storage as well as the speed with which it must be processed. Variety refers to the lack of uniformity in the structure of the data being stored. As a result of Big Data, organizations are having to employ a variety of data storage solutions that include technologies in addition to relational databases, a situation referred to as polyglot persistence.
- Volume, velocity, and variety are collectively referred to as the 3 Vs of Big Data. However, these are not the only characteristics of Big Data to which data administrators must be sensitive. Additional Vs that have been suggested by the data management industry include variability, veracity, value, and visualization. Variability is the variation in the meaning of data that can occur over time. Veracity is the trustworthiness of the data. Value is concerned with whether or not the data is useful. Finally, visualization is the requirement that the data must be able to be presented in a manner that makes it comprehendible to decision makers. Most of these additional Vs are not unique to Big Data. They are also concerns for data in relational databases as well.
- The Hadoop framework has quickly emerged as a standard for the physical storage of Big Data. The primary components of the framework include the Hadoop Distributed File System (HDFS) and MapReduce. HDFS is a coordinated technology for reliably distributing data over a very large cluster of commodity servers. MapReduce is a complementary process for distributing data processing across distributed data. One of the key concepts for MapReduce is to move the computations to the data instead of moving the data to the computations. MapReduce works by combining the functions of *map*, which distributes subtasks to the cluster servers that hold data to be processed, and *reduce*, which combines the map results into a single result set. The Hadoop framework also supports an entire ecosystem of additional tools and technologies, such as Hive, Pig, and Flume that work together to produce a complex system of Big Data processing.

- NoSQL is a broad term to refer to any of several nonrelational database approaches to data management. Most NoSQL databases fall into one of four categories: key-value databases, document databases, column-oriented databases, or graph databases. Due to the wide variability of products under the NoSQL umbrella, these categories are not necessarily all-encompassing, and many products can fit into multiple categories.
- Key-value databases store data in key-value pairs. In a key-value pair, the value of the key must be known to the DBMS, but the data in the value component can be of any type, and the DBMS makes no attempt to understand the meaning of the data in it. These types of databases are very fast when the data is completely independent, and the application programs can be relied on to understand the meaning of the data.
- Document databases also store data in key-value pairs, but the data in the value component is an encoded document. The document must be encoded using tags, such as in XML or JSON. The DBMS is aware of the tags in the documents, which makes querying on tags possible. Document databases expect documents to be self-contained and relatively independent of each other.
- Column-oriented databases, also called column family databases, organize data into key-value pairs in which the value component is composed of a series of columns, which are themselves key-value pairs. Columns can be grouped into super columns, similar to a composite attribute in the relational model being composed of simple attributes. All objects of a similar type are identified as rows, given a row key, and placed within a column family. Rows within a column family are not required to have the same structure, that is, they are not required to have the same columns.
- Graph databases are based on graph theory and represent data through nodes, edges, and properties. A node is similar to an instance of an entity in the relational model. Edges are the relationships between nodes. Both nodes and edges can have properties, which are attributes that describe the corresponding node or edge. Graph databases excel at tracking data that is highly interrelated, such as social media data. Due to the many relationships among the nodes, it is difficult to distribute a graph database across a cluster in a highly-distributed manner.
- NewSQL databases attempt to integrate features of both RDBMS (providing ACID-compliant transactions) and NoSQL databases (using a highly distributed infrastructure).
- Data analytics is a subset of BI functionality that provides advanced data analysis tools to extract knowledge from business data. Data analytics can be divided into explanatory and predictive analytics. Explanatory analytics focuses on discovering and explaining data characteristics and relationships. Predictive analytics focuses on creating models to predict future outcomes or events based on the existing data.
- Data mining automates the analysis of operational data to find previously unknown data characteristics, relationships, dependencies, and trends. The data-mining process has four phases: data preparation, data analysis and classification, knowledge acquisition, and prognosis.
- Predictive analytics uses the information generated in the data-mining phase to create advanced predictive models with high degrees of accuracy.