[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html" \l "header)

**5.1**Logical versus Physical Models

During the requirements engineering process described in [Chapter 4](javascript://), fact-finding techniques were used to investigate the current system and identify user requirements. [Chapters 5](javascript://) and [6](javascript://) explain how that information is used to develop a logical model of the proposed system and document the system requirements. A [**logical model**](javascript://) shows *what* the system must do, regardless of how it will be implemented physically. Later, in the systems design phase, a [**physical model**](javascript://) is built that describes *how* the system will be constructed.

While structured analysis tools are used to develop a logical model for a new information system, such tools also can be used to develop physical models of an information system. A physical model shows how the system’s requirements are implemented. During the systems design phase, a physical model of the new information system is created that follows from the logical model and involves operational tasks and techniques.

To understand the relationship between logical and physical models, think back to the beginning of the systems analysis phase. To understand how the current tasks were carried out in the existing system, the physical operations of the existing system were studied before the logical model. Many systems analysts create a physical model of the current system and then develop a logical model of the current system before tackling a logical model of the new system. Performing that extra step allows them to understand the current system better.

Many analysts follow a [**four-model approach**](javascript://), which means that they develop a physical model of the current system, a logical model of the current system, a logical model of the new system, and a physical model of the new system. The major benefit of the four-model approach is that it provides a clear picture of current system functions before any modifications or improvements are made. That is important because mistakes made early in systems development will affect later SDLC phases and can result in unhappy users and additional costs. Taking additional steps to avoid these potentially costly mistakes can prove to be well worth the effort. Another advantage is that the requirements of a new information system often are quite similar to those of the current information system, especially where the proposal is based on new computer technology rather than a large number of new requirements. Adapting the current system logical model to the new system logical model in these cases is a straightforward process.

The only disadvantage of the four-model approach is the added time and cost needed to develop a logical and physical model of the current system. Most projects have very tight schedules that might not allow time to create the current system models. Additionally, users and managers want to see progress on the new system—they are much less concerned about documenting the current system. The systems analyst must stress the importance of careful documentation and resist the pressure to hurry the development process at the risk of creating serious problems later.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**5.2**Data Flow Diagrams

Systems analysts use many graphical techniques to describe an information system. One popular method is to draw a set of data flow diagrams. A **data flow diagram (DFD)** uses various symbols to show how the system transforms input data into useful information. Other graphical tools include object models, which are explained in [Chapter 6](javascript://) (Object Modeling), and entity-relationship diagrams, which are described in [Chapter 9](javascript://) (Data Design).

During the systems analysis phase of the SDLC, the systems analyst creates a visual model of the information system using a set of DFDs. In his seminal book *The Visual Display of Quantitative Information*, author and renowned academic Edward Tufte provides guidance on creating effective diagrams to concisely convey complex information. A DFD is an example of this type of visual explanation of system behavior.

A DFD shows how data moves through an information system but does not show program logic or processing steps. A set of DFDs provides a logical model that shows *what* the system does, not *how* it does it. That distinction is important because focusing on implementation issues at this point would restrict the search for the most effective system design.

Change font size

[**help**](javascript://)

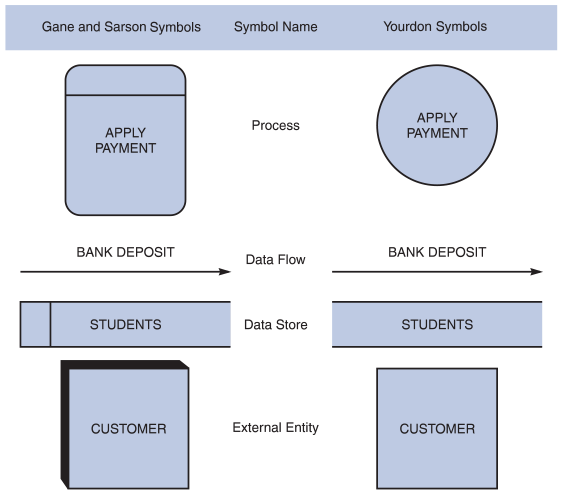
[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**5.3**Data Flow Diagram Symbols

DFDs use four basic symbols that represent processes, data flows, data stores, and entities. Several different versions of DFD symbols exist, but they all serve the same purpose. DFD examples in this textbook use the [**Gane and Sarson**](javascript://) symbol set. Another popular symbol set is the [**Yourdon**](javascript://) symbol set. [Figure 5-1](javascript://) shows examples of both versions. In this text, symbols are referenced using all capital letters for the symbol name.

**Figure 5-1**

Data flow diagram symbols, symbol names, and examples of the Gane and Sarson and Yourdon symbol sets.



Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.3.1Process Symbols

A [**process**](javascript://) receives input data and produces output that has a different content, form, or both. For instance, the process for calculating pay uses two inputs (pay rate and hours worked) to produce one output (total pay). Processes can be very simple or quite complex. In a typical company, processes might include calculating sales trends, filing online insurance claims, ordering inventory from a supplier’s system, or verifying email addresses for web customers. Processes contain the [**business logic**](javascript://), also called [**business rules**](javascript://), that transforms the data and produces the required results.

The symbol for a process is a rectangle with rounded corners. The name of the process appears inside the rectangle. The process name identifies a specific function and consists of a verb (and an adjective, if necessary) followed by a singular noun. Examples of process names are APPLY RENT PAYMENT, CALCULATE COMMISSION, ASSIGN FINAL GRADE, VERIFY ORDER, and FILL ORDER.

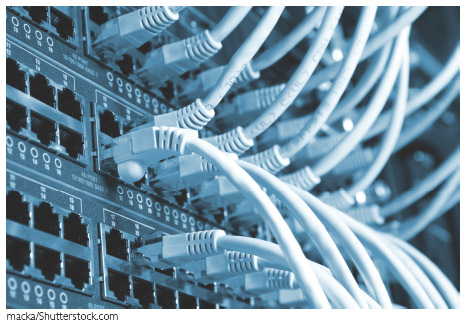
Processing details are not shown in a DFD. For example, there might be a process named DEPOSIT PAYMENT. The process symbol does not reveal the business logic for the DEPOSIT PAYMENT process. To document the logic, a process description is created, which is explained later in this chapter.

In DFDs, a process symbol can be referred to as a [**black box**](javascript://), because the inputs, outputs, and general functions of the process are known, but the underlying details and logic of the process are hidden. By showing processes as black boxes, an analyst can create DFDs that show how the system functions but avoid unnecessary detail and clutter. When the analyst wishes to show additional levels of detail, he or she can zoom in on a process symbol and create a more in-depth DFD that shows the process’s internal workings—which might reveal even more processes, data flows, and data stores. In this manner, the information system can be modeled as a series of increasingly detailed pictures.

The network router shown in [Figure 5-2](javascript://) is an example of a black box. An observer can see cables that carry data into and out of the router, but the router’s internal operations are not revealed—only the results are apparent.

**Figure 5-2**

A router acts like a black box for network data. Cables carry data in and out, but internal operations are hidden inside the case.



macka/ [Shutterstock.com](http://shutterstock.com/" \t "_blank)

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.3.2Data Flow Symbols

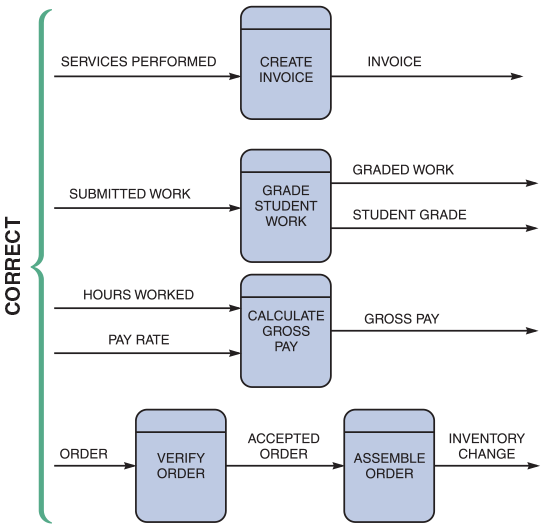
A [**data flow**](javascript://) is a path for data to move from one part of the information system to another. A data flow in a DFD represents one or more data items. For example, a data flow could consist of a single data item (such as a student ID number), or it could include a set of data (such as a class roster with student ID numbers, names, and registration dates for a specific class). Although the DFD does not show the detailed contents of a data flow, that information is included in the data dictionary, which is described later in this chapter.

The symbol for a data flow is a line with a single or double arrowhead. The data flow name appears above, below, or alongside the line. A data flow name consists of a singular noun and an adjective, if needed. Examples of data flow names are DEPOSIT, INVOICE PAYMENT, STUDENT GRADE, ORDER, and COMMISSION. Exceptions to the singular name rule are data flow names, such as GRADING PARAMETERS, where a singular name could mislead the analyst into thinking a single parameter or single item of data exists.

[Figure 5-3](javascript://) shows correct examples of data flow and process symbol connections. Because a process changes the data’s content or form, at least one data flow must enter and one data flow must exit each process symbol, as they do in the CREATE INVOICE process. A process symbol can have more than one outgoing data flow, as shown in the GRADE STUDENT WORK process, or more than one incoming data flow, as shown in the CALCULATE GROSS PAY process. A process also can connect to any other symbol, including another process symbol, as shown by the connection between VERIFY ORDER and ASSEMBLE ORDER in [Figure 5-3](javascript://). A data flow, therefore, must have a process symbol on at least one end.

**Figure 5-3**

Examples of correct combinations of data flow and process symbols.

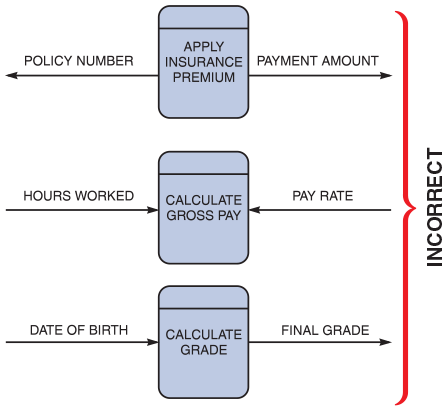


[Figure 5-4](javascript://) shows three data flow and process combinations that must be avoided:

* [**Spontaneous generation**](javascript://). The APPLY INSURANCE PREMIUM process, for instance, produces output but has no input data flow. Because it has no input, the process is called a spontaneous generation process.
* [**Black hole**](javascript://). CALCULATE GROSS PAY is called a black hole process, which is a process that has input but produces no output.
* [**Gray hole**](javascript://). A gray hole is a process that has at least one input and one output, but the input obviously is insufficient to generate the output shown. For example, a date of birth input is not sufficient to produce a final grade output in the CALCULATE GRADE process.

**Figure 5-4**

Examples of incorrect combinations of data flow and process symbols. APPLY INSURANCE PREMIUM has no input and is called a spontaneous generation process. CALCULATE GROSS PAY has no outputs and is called a black hole process. CALCULATE GRADE has an input that is obviously unable to produce the output. This process is called a gray hole.



Spontaneous generation, black holes, and gray holes are impossible logically in a DFD because a process must act on input, shown by an incoming data flow, and produce output, represented by an outgoing data flow.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.3.3Data Store Symbols

A [**data store**](javascript://) is used in a DFD to represent data that the system stores because one or more processes need to use the data at a later time. For instance, instructors need to store student scores on tests and assignments during the semester, so they can assign final grades at the end of the term. Similarly, a company stores employee salary and deduction data during the year in order to print W-2 forms with total earnings and deductions at the end of the year. A DFD does not show the detailed contents of a data store—the specific structure and data elements are defined in the data dictionary, which is discussed later in this chapter.

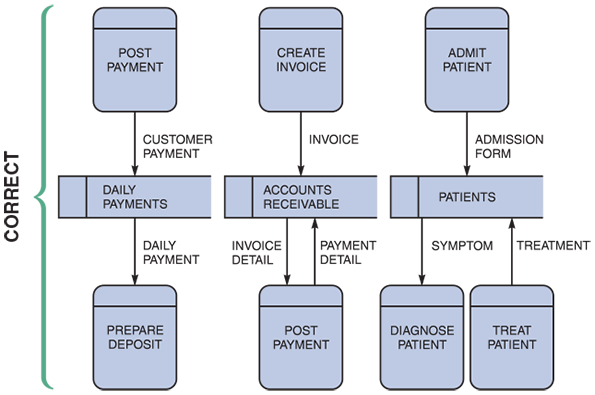
The physical characteristics of a data store are unimportant because the logical model is the only concern at this point. Also, the length of time that the data is stored is unimportant—it can be a matter of seconds while a transaction is processed or a period of months while data is accumulated for year-end processing. What is important is that a process needs access to the data at some later time.

In a DFD, the Gane and Sarson symbol for a data store is a flat rectangle that is open on the right side and closed on the left side. The name of the data store appears between the lines and identifies the data it contains. A data store name is a plural name consisting of a noun and adjectives, if needed. Examples of data store names are STUDENTS, ACCOUNTS RECEIVABLE, PRODUCTS, DAILY PAYMENTS, PURCHASE ORDERS, OUTSTANDING CHECKS, INSURANCE POLICIES, and EMPLOYEES. Exceptions to the plural name rule are collective nouns that represent multiple occurrences of objects. For example, GRADEBOOK represents a group of students and their scores.

A data store must be connected to a process with a data flow. [Figure 5-5](javascript://) illustrates typical examples of data stores. Since data stores represent data storage for use by another process in the future, in each case, the data store has at least one incoming and one outgoing data flow and is connected to a process symbol with a data flow.

**Figure 5-5**

Examples of correct use of data store symbols in a data flow diagram.

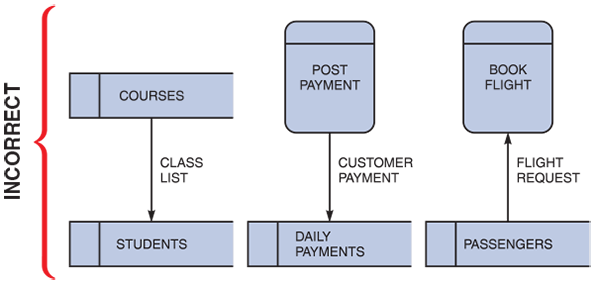


Enlarge Image

Violations of the rule that a data store must have at least one incoming and one outgoing data flow are shown in [Figure 5-6](javascript://). In the first example, two data stores are connected incorrectly because no process is between them. Also, COURSES has no incoming data flow and STUDENTS has no outgoing data flow. In the second and third examples, the data stores lack either an outgoing or incoming data flow.

**Figure 5-6**

Examples of incorrect use of data store symbols in a data flow diagram. Two data stores cannot be connected by a data flow without an intervening process, and each data store should have an outgoing and incoming data flow.



Enlarge Image

There is an exception to the requirement that a data store must have at least one incoming and one outgoing data flow. In some situations, a data store has no input data flow because it contains fixed reference data that is not updated by the system. For example, consider a data store called TAX TABLE, which contains withholding tax data that a company downloads from the Internal Revenue Service. When the company runs its payroll, the CALCULATE WITHHOLDING process accesses data from this data store. On a DFD, this would be represented as a one-way outgoing data flow from the TAX TABLE data store into the CALCULATE WITHHOLDING process.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.3.4Entity Symbols

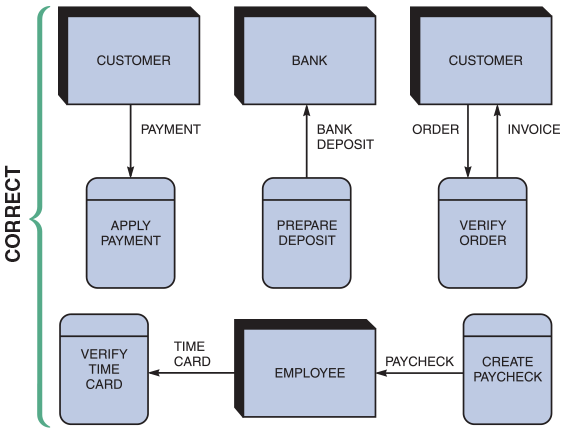
The symbol for an [**entity**](javascript://) is a rectangle, which may be shaded to make it look three-dimensional. The name of the entity appears inside the symbol.

A DFD shows only external entities that provide data to the system or receive output from the system. A DFD shows the boundaries of the system and how the system interfaces with the outside world. For example, a customer entity submits an order to an order processing system. Other examples of entities include a patient who supplies data to a medical records system, a homeowner who receives a bill from a city property tax system, or an accounts payable system that receives data from the company’s purchasing system.

DFD entities also are called [**terminators**](javascript://) because they are data origins or final destinations. Systems analysts call an entity that supplies data to the system a [**source**](javascript://) and an entity that receives data from the system a [**sink**](javascript://). An entity name is the singular form of a department, an outside organization, other information system, or a person. An external entity can be a source or a sink or both, but each entity must be connected to a process by a data flow. [Figure 5-7](javascript://) and [Figure 5-8](javascript://) show correct and incorrect examples of this rule, respectively.

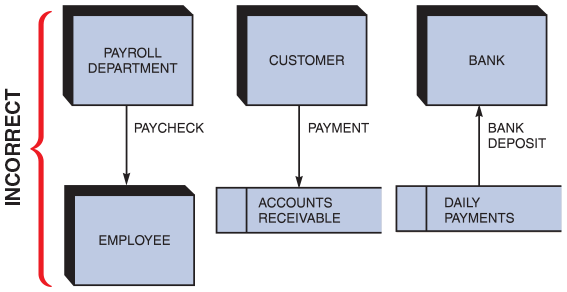
**Figure 5-7**

Examples of correct uses of external entities in a data flow diagram.



**Figure 5-8**

Examples of incorrect uses of external entities in a data flow diagram. An external entity must be connected by a data flow to a process and not directly to a data store or to another external entity.



Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.3.5Using DFD Symbols

With an understanding of the proper use of DFD symbols, the next step is to construct diagrams that use these symbols. [Figure 5-9](javascript://) shows a summary of the rules for using DFD symbols.

**Figure 5-9**

Examples of correct and incorrect uses of data flows.



Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**5.4**Drawing Data Flow Diagrams

During requirements engineering, interviews, questionnaires, and other techniques were used to gather facts about the system, and it was explained how the various people, departments, data, and processes fit together to support business operations. Now a graphical model of the information system is created based on the fact-finding results.

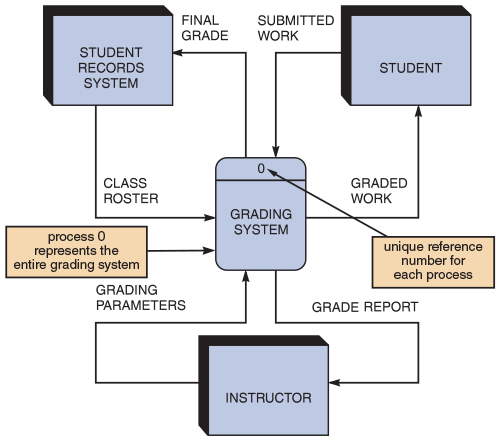
To learn how to draw DFDs, examples of two information systems will be used. The first example is a grading system that instructors use to assign final grades based on the scores that students receive during the term. The second example is an order system that a company uses to enter orders and apply payments against a customer’s balance.

When drawing a context diagram and other DFDs, these guidelines should be followed:

* Draw the context diagram so it fits on one page.
* Use the name of the information system as the process name in the context diagram. For example, the process name in [Figure 5-10](javascript://) is GRADING SYSTEM. Note that the process name is the same as the system name. This is because the context diagram shows the entire information system as if it were a single process. For processes in lower-level DFDs, use a verb followed by a descriptive noun, such as ESTABLISH GRADEBOOK, ASSIGN FINAL GRADE, or PRODUCE GRADE REPORT.
* Use unique names within each set of symbols. For instance, the diagram in [Figure 5-10](javascript://) shows only one entity named STUDENT and only one data flow named FINAL GRADE. Whenever the entity STUDENT appears on any other DFD in the grading system, it indicates that it is the same entity. Whenever the FINAL GRADE data flow appears, it indicates that it is the same data flow. The naming convention also applies to data stores.
* Do not cross lines. One way to achieve that goal is to restrict the number of symbols in any DFD. On lower-level diagrams with multiple processes, there should not be more than nine process symbols. Including more than nine symbols usually is a signal that the diagram is too complex and that the analysis should be reconsidered. Another way to avoid crossing lines is to duplicate an entity or a data store. When duplicating a symbol on a diagram, make sure to document the duplication to avoid possible confusion. A special notation, such as an asterisk, next to the symbol name and inside the duplicated symbols signifies that they are duplicated on the diagram.
* Provide a unique name and reference number for each process. Because it is the highest-level DFD, the context diagram contains process 0, which represents the entire information system, but does not show the internal workings. To describe the next level of detail inside process 0, create a DFD named diagram 0, which will reveal additional processes that must be named and numbered. As lower-level DFDs are created, assign unique names and reference numbers to all processes, until the logical model is completed.
* Obtain as much user input and feedback as possible. The main objective is to ensure that the model is accurate, is easy to understand, and meets the needs of its users.

**Figure 5-10**

Context diagram DFD for a grading system.



Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

# 5.5Drawing a Context Diagram

The first step in constructing a set of DFDs is to draw a context diagram. A [**context diagram**](javascript://) is a top-level view of an information system that shows the system’s boundaries and scope. To draw a context diagram, start by placing a single process symbol in the center of the page. The symbol represents the entire information system, and it is identified as [**process 0**](javascript://) (the numeral zero, and not the letter O). Then place the system entities around the perimeter of the page and use data flows to connect the entities to the central process. Data stores are not shown in the context diagram because they are contained within the system and remain hidden until more detailed diagrams are created.

To determine which entities and data flows to place in the context diagram, begin by reviewing the system requirements to identify all external data sources and destinations. During that process, identify the entities, the name and content of the data flows, and the direction of the data flows. If that is done carefully, and the job of fact-finding was done well in the previous stage, drawing the context diagram should be relatively easy. Now review the following context diagram examples.

### Example: Context Diagram for a Grading System

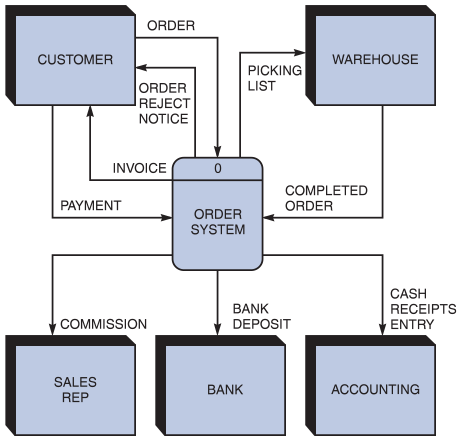
The context diagram for a grading system is shown in [Figure 5-10](javascript://). The GRADING SYSTEM process is at the center of the diagram. The three entities (STUDENT RECORDS SYSTEM, STUDENT, and INSTRUCTOR) are placed around the central process. Interaction among the central process and the entities involves six different data flows. The STUDENT RECORDS SYSTEM entity supplies data through the CLASS ROSTER data flow and receives data through the FINAL GRADE data flow. The STUDENT entity supplies data through the SUBMITTED WORK data flow and receives data through the GRADED WORK data flow. Finally, the INSTRUCTOR entity supplies data through the GRADING PARAMETERS data flow and receives data through the GRADE REPORT data flow.

### Example: Context Diagram for an Order System

The context diagram for an order system is shown in [Figure 5-11](javascript://). Note that the ORDER SYSTEM process is at the center of the diagram and five entities surround the process. Three of the entities, SALES REP, BANK, and ACCOUNTING, have single incoming data flows for COMMISSION, BANK DEPOSIT, and CASH RECEIPTS ENTRY, respectively. The WAREHOUSE entity has one incoming data flow—PICKING LIST—that is, a report that shows the items ordered and their quantity, location, and sequence to pick from the warehouse. The WAREHOUSE entity has one outgoing data flow: COMPLETED ORDER. Finally, the CUSTOMER entity has two outgoing data flows, ORDER and PAYMENT, and two incoming data flows, ORDER REJECT NOTICE and INVOICE.

**Figure 5-11**

Context diagram DFD for an order system.



The context diagram for the order system appears more complex than the grading system because it has two more entities and three more data flows. What makes one system more complex than another is the number of components, the number of levels, and the degree of interaction among its processes, entities, data stores, and data flows.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

# 5.6Drawing a Diagram 0 DFD

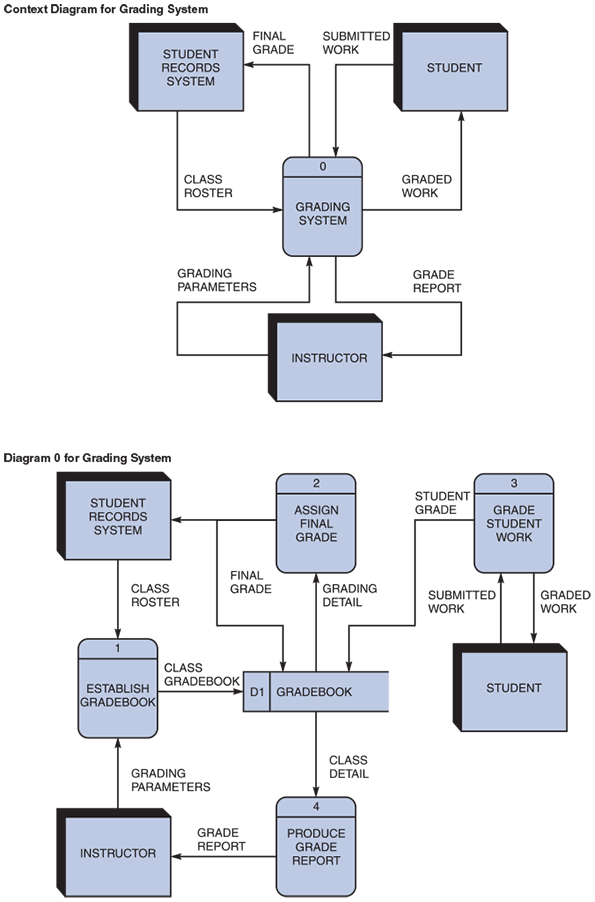
In the previous step, it was explained how a context diagram provides the most general view of an information system and contains a single process symbol, which is like a black box. To show the detail inside the black box, a DFD diagram 0 is created. [**Diagram 0**](javascript://) (the numeral zero, and not the letter O) provides an overview of all the components that interact to form the overall system. It zooms in on the system and shows major internal processes, data flows, and data stores. Diagram 0 also repeats the entities and data flows that appear in the context diagram. When the context diagram is expanded into DFD diagram 0, all the connections that flow into and out of process 0 must be retained.

### Example: Diagram 0 DFD for a Grading System

[Figure 5-12](javascript://) shows a context diagram at the top and diagram 0 beneath it. Note that diagram 0 is an expansion of process 0. Also note that the three same entities (STUDENT RECORDS SYSTEM, STUDENT, and INSTRUCTOR) and the same six data flows (FINAL GRADE, CLASS ROSTER, SUBMITTED WORK, GRADED WORK, GRADING PARAMETERS, and GRADE REPORT) appear in both diagrams. In addition, diagram 0 expands process 0 to reveal four internal processes, one data store, and five additional data flows.

**Figure 5-12**

Context diagram and diagram 0 for the grading system.



Enlarge Image

Note that each process in diagram 0 has a reference number: ESTABLISH GRADEBOOK is 1, ASSIGN FINAL GRADE is 2, GRADE STUDENT WORK is 3, and PRODUCE GRADE REPORT is 4. These reference numbers are important because they identify a series of DFDs. If more detail were needed for ESTABLISH GRADEBOOK, for example, a diagram 1 would be drawn, because ESTABLISH GRADEBOOK is process 1.

The process numbers do not suggest that the processes are accomplished in a sequential order. Each process always is considered to be available, active, and awaiting data to be processed. If processes must be performed in a specific sequence, the information should be documented in the process descriptions (discussed later in this chapter), not in the DFD.

The FINAL GRADE data flow output from the ASSIGN FINAL GRADE process is a diverging data flow that becomes an input to the STUDENT RECORDS SYSTEM entity and to the GRADEBOOK data store. A [**diverging data flow**](javascript://) is a data flow in which the same data travels to two or more different locations. In that situation, a diverging data flow is the best way to show the flow rather than showing two identical data flows, which could be misleading.

If the same data flows in both directions, a double-headed arrow can be used to connect the symbols. To identify specific data flows into and out of a symbol, however, separate data flow symbols with single arrowheads should be used. For example, in [Figure 5-12](javascript://), the separate data flows (SUBMITTED WORK and GRADED WORK) go into and out of the GRADE STUDENT WORK process.

Because diagram 0 is an exploded version of process 0, it shows considerably more detail than the context diagram. Diagram 0 can also be referred to as a partitioned or decomposed view of process 0. When a DFD is exploded, the higher-level diagram is called the [**parent diagram**](javascript://) and the lower-level diagram is referred to as the [**child diagram**](javascript://). The grading system is simple enough that no additional DFDs are needed to model the system. At that point, the four processes, the one data store, and the 10 data flows can be documented in the data dictionary.

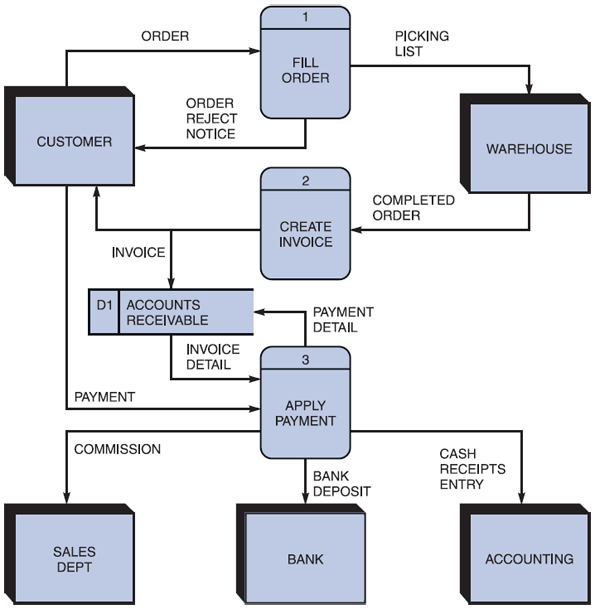
When a set of DFDs is created for a system, the processing logic is broken down into smaller units, called functional primitives, which programmers will use to develop code. A [**functional primitive**](javascript://) is a process that consists of a single function that is not exploded further. For example, each of the four processes shown in the lower portion of [Figure 5-12](javascript://) is a functional primitive. The logic for a functional primitive is documented by writing a process description in the data dictionary. Later, when the logical design is implemented as a physical system, programmers will transform each functional primitive into program code and modules that carry out the required steps. Deciding whether to explode a process further or determine that it is a functional primitive is a matter of experience, judgment, and interaction with programmers who must translate the logical design into code.

### Example: Diagram 0 Dfd for an Order System

[Figure 5-13](javascript://) is the diagram 0 for an order system. Process 0 on the order system’s context diagram is exploded to reveal three processes (FILL ORDER, CREATE INVOICE, and APPLY PAYMENT), one data store (ACCOUNTS RECEIVABLE), two additional data flows (INVOICE DETAIL and PAYMENT DETAIL), and one diverging data flow (INVOICE).

**Figure 5-13**

Diagram 0 DFD for the order system.



Enlarge Image

The following walk-through explains the DFD shown in [Figure 5-13](javascript://):

1. A CUSTOMER submits an ORDER. Depending on the processing logic, the FILL ORDER process either sends an ORDER REJECT NOTICE back to the customer or sends a PICKING LIST to the WAREHOUSE.
2. A COMPLETED ORDER from the WAREHOUSE is input to the CREATE INVOICE process, which outputs an INVOICE to both the CUSTOMER process and the ACCOUNTS RECEIVABLE data store.
3. A CUSTOMER makes a PAYMENT that is processed by APPLY PAYMENT. APPLY PAYMENT requires INVOICE DETAIL input from the ACCOUNTS RECEIVABLE data store along with the PAYMENT. APPLY PAYMENT also outputs PAYMENT DETAIL back to the ACCOUNTS RECEIVABLE data store and outputs COMMISSION to the SALES DEPT, BANK DEPOSIT to the BANK, and CASH RECEIPTS ENTRY to ACCOUNTING.

The walk-through of diagram 0 illustrates the basic requirements of the order system. Examine the detailed description of each separate process to learn more.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

# 5.7Drawing Lower-Level DFDs

This set of lower-level DFDs is based on the order system. To create lower-level diagrams, leveling and balancing techniques must be used. [**Leveling**](javascript://) is the process of drawing a series of increasingly detailed diagrams, until all functional primitives are identified. **[Balancing](javascript://)** maintains consistency among a set of DFDs by ensuring that input and output data flows align properly. Leveling and balancing are described in more detail in the following sections.

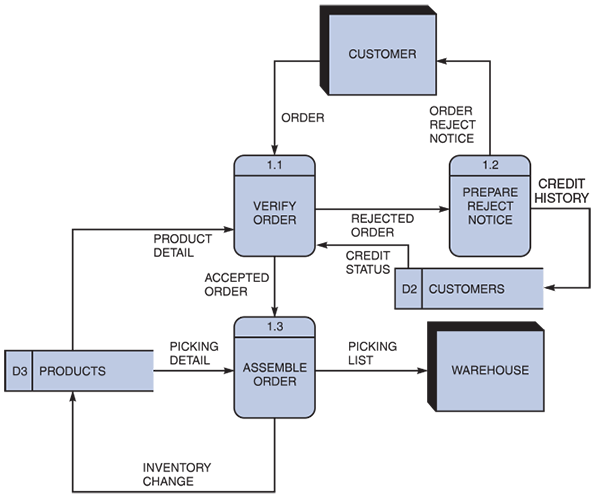
### Example: Leveling Examples

Leveling uses a series of increasingly detailed DFDs to describe an information system. For example, a system might consist of dozens, or even hundreds, of separate processes. Using leveling, an analyst starts with an overall view, which is a context diagram with a single process symbol. Next, the analyst creates diagram 0, which shows more detail. The analyst continues to create lower-level DFDs until all processes are identified as functional primitives, which represent single processing functions. More complex systems have more processes, and analysts must work through many levels to identify the functional primitives. Leveling also is called [**exploding**](javascript://), [**partitioning**](javascript://), or [**decomposing**](javascript://).

[Figures 5-13](javascript://) and [5-14](javascript://) provide an example of leveling. [Figure 5-13](javascript://) shows diagram 0 for an order system, with the FILL ORDER process labeled as process 1. Now consider [Figure 5-14](javascript://), which provides an exploded view of the FILL ORDER process. Note that FILL ORDER (process 1) actually consists of three processes: VERIFY ORDER (process 1.1), PREPARE REJECT NOTICE (process 1.2), and ASSEMBLE ORDER (process 1.3).

**Figure 5-14**

Diagram 1 DFD shows details of the FILL ORDER process in the order system.



Enlarge Image

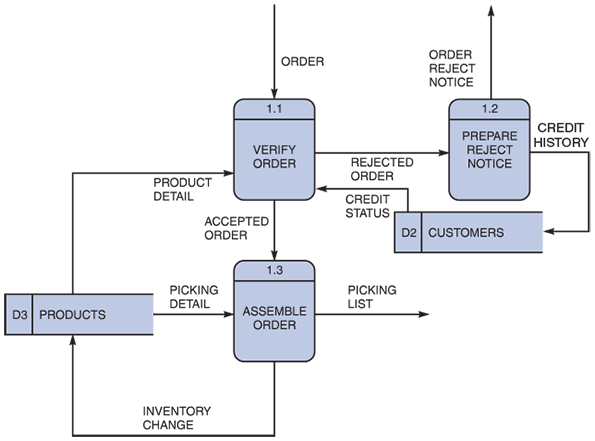
As [Figure 5-14](javascript://) shows, all processes are numbered using a decimal notation consisting of the parent’s reference number, a decimal point, and a sequence number within the new diagram. In [Figure 5-14](javascript://), the parent process of diagram 1 is process 1, so the processes in diagram 1 have reference numbers of 1.1, 1.2, and 1.3. If process 1.3, ASSEMBLE ORDER, is decomposed further, then it would appear in diagram 1.3 and the processes in diagram 1.3 would be numbered as 1.3.1, 1.3.2, 1.3.3, and so on. This numbering technique makes it easy to integrate and identify all DFDs.

When [Figure 5-13](javascript://) and [Figure 5-14](javascript://) are compared, it is apparent that [Figure 5-14](javascript://) (the exploded FILL ORDER process) shows two data stores (CUSTOMER and PRODUCTS) that do not appear on [Figure 5-13](javascript://), which is the parent DFD. Why not? The answer is based on a simple rule: When drawing DFDs, a data store is shown only when two or more processes use that data store. The CUSTOMER and PRODUCTS data stores were internal to the FILL ORDER process, so the analyst did not show them on diagram 0, which is the parent. When the FILL ORDER process is exploded into a diagram 1 DFD, however, three processes (1.1, 1.2, and 1.3) interacting with the two data stores are now shown.

Now compare [Figure 5-14](javascript://) and [Figure 5-15](javascript://). Note that [Figure 5-15](javascript://) shows the same data flows as [Figure 5-14](javascript://) but does not show the CUSTOMER and WAREHOUSE entities. Analysts often use this technique to simplify a DFD and reduce unnecessary clutter. Because the missing symbols appear on the parent DFD, that diagram can be used to identify the source or destination of the data flows.

**Figure 5-15**

This diagram does not show the symbols that connect to data flows entering or leaving FILL ORDER on the context diagram.



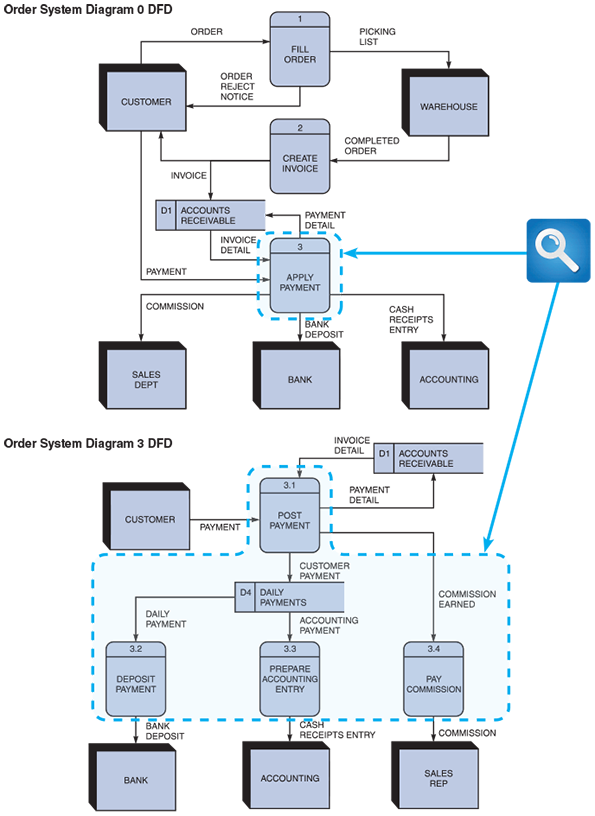
Enlarge Image

### Balancing Examples

Balancing ensures that the input and output data flows of the parent DFD are maintained on the child DFD. For example, [Figure 5-16](javascript://) shows two DFDs: The order system diagram 0 is shown at the top of the figure, and the exploded diagram 3 DFD is shown at the bottom.

**Figure 5-16**

The order system diagram 0 is shown at the top of the figure, and the exploded diagram 3 DFD (for the APPLY PAYMENT process) is shown at the bottom. The two DFDs are balanced because the child diagram at the bottom has the same input and output flows as the parent process 3 shown at top.



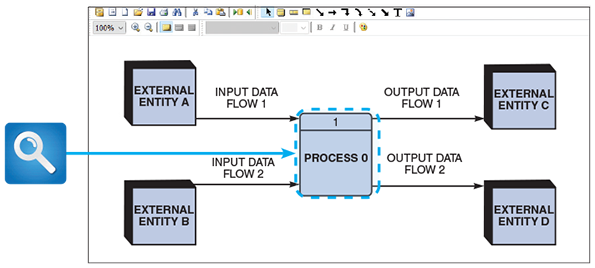
Enlarge Image

The two DFDs are balanced because the child diagram at the bottom has the same input and output flows as the parent process 3 shown at the top. To verify the balancing, note that the parent process 3, APPLY PAYMENT, has one incoming data flow from an external entity and three outgoing data flows to external entities. Examine the child DFD, which is diagram 3. Ignore the internal data flows and count the data flows to and from external entities. From the diagram, it is evident that the three processes maintain the same one incoming and three outgoing data flows as the parent process.

Another example of balancing is shown in [Figures 5-17](javascript://) and [5-18](javascript://). The DFDs in these figures were created using the Visible Analyst CASE tool.

**Figure 5-17**

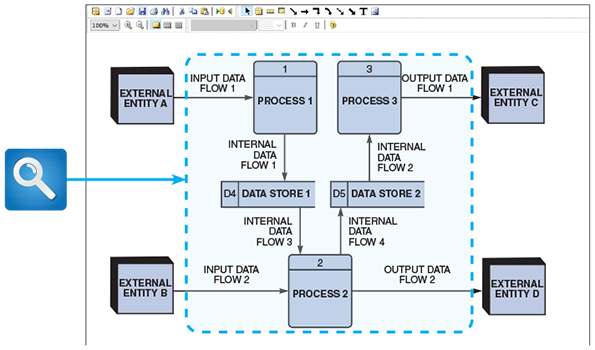
Examples of a parent DFD diagram, showing process 0 as a black box.



Enlarge Image

**Figure 5-18**

In the next level of detail, the process 0 black box reveals three processes, two data stores, and four internal data flows—all of which are shown inside the dashed line.



Enlarge Image

[Figure 5-17](javascript://) shows a sample context diagram. The process 0 symbol has two input flows and two output flows. Note that process 0 can be considered as a black box with no internal detail shown. In [Figure 5-18](javascript://), process 0 (the parent DFD) is exploded into the next level of detail. Now three processes, two data stores, and four internal data flows are visible. Note that the details of process 0 are shown inside a dashed line, just as if the inside of the process was visible.

The DFDs in [Figures 5-17](javascript://) and [5-18](javascript://) are balanced because the four data flows into and out of process 0 are maintained on the child DFD. The DFDs also are leveled because each internal process is numbered to show that it is a child of the parent process.

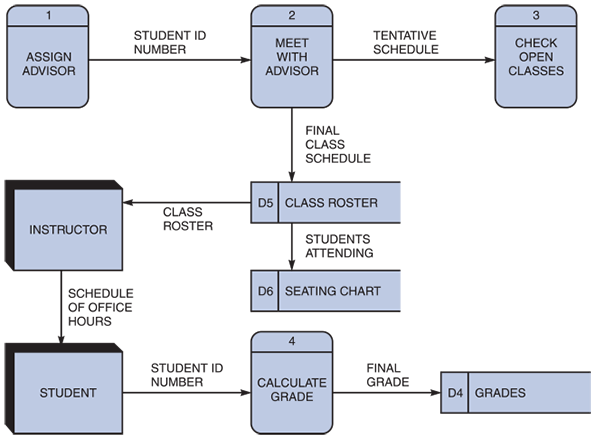
**Case in Point 5.1**

### Big Ten University

* You are the IT director at Big Ten University. As part of a training program, you decide to draw a DFD that includes some obvious mistakes to see whether your newly hired junior analysts can find them. You came up with the diagram 0 DFD shown in [Figure 5-19](javascript://). Based on the rules explained in this chapter, how many problems should the analysts find?

**Figure 5-19**

What are the mistakes in this diagram 0 DFD?



Enlarge Image

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**5.8**Data Dictionary

A set of DFDs produces a logical model of the system, but the details within those DFDs are documented separately in a data dictionary, which is the second component of structured analysis.

A [**data dictionary**](javascript://), or [**data repository**](javascript://), is a central storehouse of information about the system’s data. An analyst uses the data dictionary to collect, document, and organize specific facts about the system, including the contents of data flows, data stores, entities, and processes. The data dictionary also defines and describes all data elements and meaningful combinations of data elements. A [**data element**](javascript://), also called a [**data item**](javascript://) or [**field**](javascript://), is the smallest piece of data that has meaning within an information system. Examples of data elements are student grade, salary, Social Security number, account balance, and company name. Data elements are combined into **records**, also called [**data structures**](javascript://). A record is a meaningful combination of related data elements that is included in a data flow or retained in a data store. For example, an auto parts store inventory record might include part number, description, supplier code, minimum and maximum stock levels, cost, and list price.

Significant relationships exist among the items in a data dictionary. For example, data stores and data flows are based on data structures, which in turn are composed of data elements. Data flows are connected to data stores, entities, and processes. Accurately documenting these relationships is essential so the data dictionary is consistent with the DFDs. The more complex the system, the more difficult it is to maintain full and accurate documentation. Fortunately, modern CASE tools simplify the task by flowing documentation automatically from the modeling diagrams into the central repository, along with information entered by the user.

A CASE repository ensures data consistency, which is especially important where multiple systems require the same data. In a large company, for example, the sales, accounting, and shipping systems all might use a data element called CUSTOMER NUMBER. Once the CUSTOMER NUMBER element has been defined in the repository, other processes can access it, data flows, and data stores. The result is that all systems across the enterprise can share data that is up to date and consistent.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.8.1Documenting the Data Elements

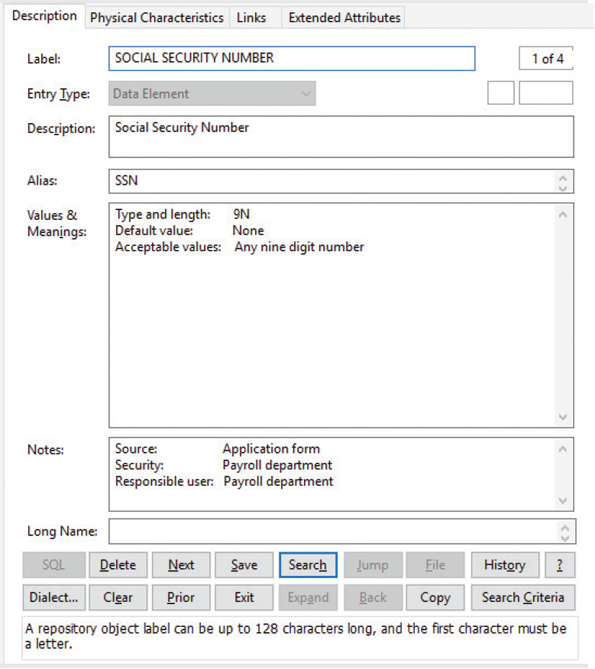
Every data element in the data dictionary must be documented. Some analysts like to record their notes in online or manual forms. Others prefer to enter the information directly into a CASE tool. Irrespective of the specific CASE tool used, the objective is the same: to provide clear, comprehensive information about the data and processes that make up the system.

[Figure 5-20](javascript://) shows a sample screen that illustrates how a data element representing a SOCIAL SECURITY NUMBER might be recorded in the Visible Analyst data dictionary. Regardless of the terminology or method, the following attributes usually are recorded and described in the data dictionary:

* Data element name or label. The data element’s standard name, which should be meaningful to users.
* Alias. Any name(s) other than the standard data element name; this alternate name is called an [**alias**](javascript://). For example, if there is a data element named CURRENT BALANCE, various users might refer to it by alternate names such as OUTSTANDING BALANCE, CUSTOMER BALANCE, RECEIVABLE BALANCE, or AMOUNT OWED.
* Type and length. Type refers to whether the data element contains numeric, alphabetic, or character values. Length is the maximum number of characters for an alphabetic or a character data element or the maximum number of digits and number of decimal positions for a numeric data element. In addition to text and numeric data, sounds and images also can be stored in digital form. In some systems, these binary data objects are managed and processed just as traditional data elements are. For example, an employee record might include a digitized photo image of the person.
* Default value. The default value is the value for the data element if a value otherwise is not entered for it. For example, all new customers might have a default value of $500 for the CREDIT LIMIT data element.
* Acceptable values. Specification of the data element’s [**domain**](javascript://), which is the set of values permitted for the data element. These values either can be specifically listed or referenced in a table or can be selected from a specified range of values. Also indicate if a value for the data element is optional. Some data elements have additional [**validity rules**](javascript://). For example, an employee’s salary must be within the range defined for the employee’s job classification.
* Source. The specification for the origination point for the data element’s values. The source could be a specific form, a department or an outside organization, another information system, or the result of a calculation.
* Security. Identification for the individual or department that has access or update privileges for each data element. For example, only a credit manager has the authority to change a credit limit, while sales reps are authorized to access data in a read-only mode.
* Responsible user(s). Identification of the user(s) responsible for entering and changing values for the data element.
* Description and comments. This part of the documentation permits the entry of additional notes.

**Figure 5-20**

A Visible Analyst screen describes the data element named SOCIAL SECURITY NUMBER.



Enlarge Image

**Source:** Screenshot used with permission from Visible Systems Corp.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.8.2Documenting the Data Flows

In addition to documenting each data element, all data flows in the data dictionary must be documented. Although terms can vary, the typical attributes are as follows:

* Data flow name or label. The data flow name as it appears on the DFDs.
* Description. Describes the data flow and its purpose.
* Alternate name(s). Aliases for the DFD data flow name(s).
* Origin. The DFD beginning, or source, for the data flow; the origin can be a process, a data store, or an entity.
* Destination. The DFD ending point(s) for the data flow; the destination can be a process, a data store, or an entity.
* Record. Each data flow represents a group of related data elements called a record or data structure. In most data dictionaries, records are defined separately from the data flows and data stores. When records are defined, more than one data flow or data store can use the same record, if necessary.
* Volume and frequency. Describes the expected number of occurrences for the data flow per unit of time. For example, if a company has 300 employees, a TIME CARD data flow would involve 300 transactions and records each week as employees submit their work hour data.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.8.3Documenting the Data Stores

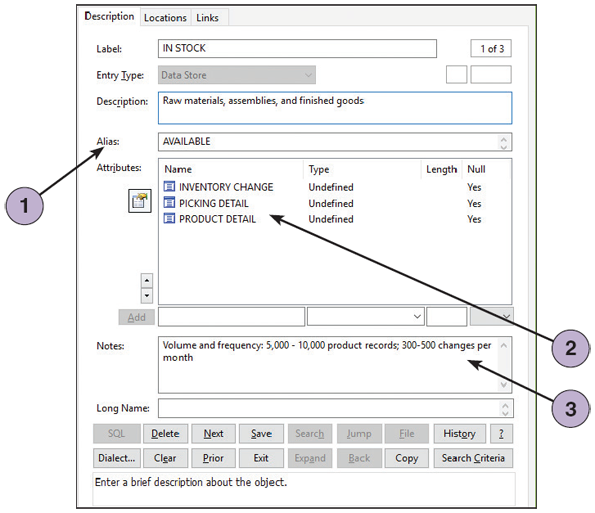
Every DFD data store in the data dictionary must be documented. [Figure 5-21](javascript://) shows the definition of a data store named IN STOCK. Typical characteristics of a data store are as follows:

* Data store name or label. The data store name as it appears on the DFDs.
* Description. Describes the data store and its purpose.
* Alternate name(s). Aliases for the DFD data store name.
* Attributes. Standard DFD names that enter or leave the data store.
* Volume and frequency. Describes the estimated number of records in the data store and how frequently they are updated.

**Figure 5-21**

A Visible Analyst screen that documents a data store named IN STOCK.

1. This data store has an alternative name, or alias.
2. For consistency, data flow names are standardized throughout the data dictionary.
3. It is important to document these estimates because they will affect design decisions in subsequent SDLC phases.



Enlarge Image

**Source:** Screenshot used with permission from Visible Systems Corp.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

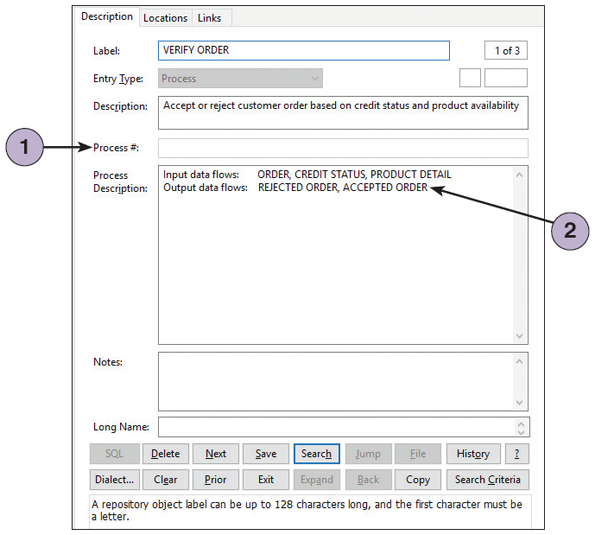
## 5.8.4Documenting the Processes

Every process must be documented, as shown in [Figure 5-22](javascript://). The documentation includes a description of the process’s characteristics and, for functional primitives, a process description, which is a model that documents the processing steps and business logic.

**Figure 5-22**

A Visible Analyst screen that describes a process named VERIFY ORDER.

1. The process number identifies this process. Any subprocesses are numbered 1.1, 1.2, 1.3, and so on.
2. These data flows will be described specifically elsewhere in the data dictionary.



Enlarge Image

**Source:** Screenshot used with permission from Visible Systems Corp.

The following are typical characteristics of a process:

* Process name or label. The process name as it appears on the DFDs.
* Description. A brief statement of the process’s purpose.
* Process number. A reference number that identifies the process and indicates relationships among various levels in the system.
* Process description. This section includes the input and output data flows. For functional primitives, the process description also documents the processing steps and business logic. The next section explains how to write process descriptions.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.8.5Documenting the Entities

By documenting all entities, the data dictionary can describe all external entities that interact with the system. Typical characteristics of an entity include the following:

* Entity name. The entity name as it appears on the DFDs.
* Description. Describe the entity and its purpose.
* Alternate name(s). Any aliases for the entity name.
* Input data flows. The standard DFD names for the input data flows to the entity.
* Output data flows. The standard DFD names for the data flows leaving the entity.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

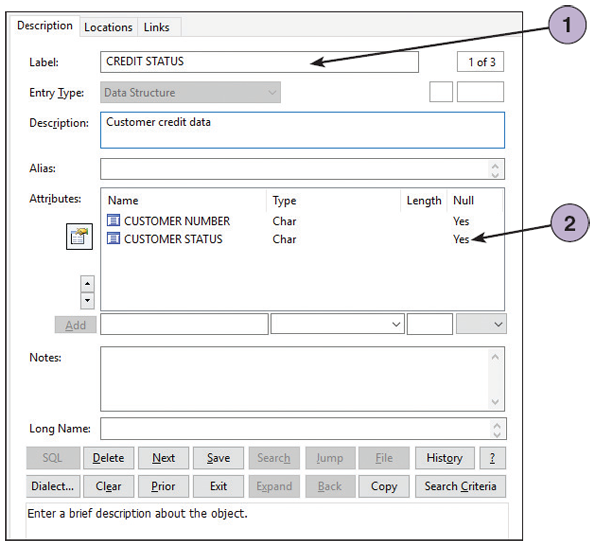
## 5.8.6Documenting the Records

A record is a data structure that contains a set of related data elements that are stored and processed together. Data flows and data stores consist of records that must be documented in the data dictionary. Characteristics of each record must also be defined, as shown in [Figure 5-23](javascript://).

**Figure 5-23**

A Visible Analyst screen that documents a record, or data structure, named CREDIT STATUS.

1. This data structure is named CREDIT STATUS.
2. The CREDIT STATUS data structure consists of two data elements: CUSTOMER NUMBER and CUSTOMER STATUS CODE.



Enlarge Image

**Source:** Screenshot used with permission from Visible Systems Corp.

Typical characteristics of a record include the following:

* Record or data structure name. The record name as it appears in the related data flow and data store entries in the data dictionary.
* Definition or description. A brief definition of the record.
* Alternate name(s). Any aliases for the record name.
* Attributes. A list of all the data elements, or fields, included in the record. The data element names must match exactly those entered in the data dictionary.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.8.7Data Dictionary Reports

The data dictionary serves as a central storehouse of documentation for an information system. A data dictionary is created when the system is developed and is updated constantly as the system is implemented, operated, and maintained. In addition to describing each data element, data flow, data store, record, entity, and process, the data dictionary documents the relationships among these components.

Many valuable reports can be obtained from a data dictionary, including the following:

* An alphabetized list of all data elements by name
* A report describing each data element and indicating the user or department that is responsible for data entry, updating, or deletion
* A report of all data flows and data stores that use a particular data element
* Detailed reports showing all characteristics of data elements, records, data flows, processes, or any other selected item stored in the data dictionary

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**5.9**Process Description Tools in Modular Design

A [**process description**](javascript://) documents the details of a functional primitive and represents a specific set of processing steps and business logic. When a functional primitive is analyzed, the processing steps are broken down into smaller units in a process called modular design. Using a set of process description tools, a model is created that is accurate, complete, and concise. Typical process description tools include structured English, decision tables, and decision trees.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.9.1Process Descriptions in Object-Oriented Development

This chapter deals with structured analysis, but the process descriptions can also be used in object-oriented (O-O) development, which is described in [Chapter 6](javascript://). As explained in [Chapter 1](javascript://), O-O analysis combines data and the processes that act on the data into things called objects, and similar objects can be grouped together into classes, and O-O processes are called methods. Although O-O programmers use different terminology, they create the same kind of modular coding structures, except that the processes, or methods, are stored inside the objects, rather than as separate components.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.9.2Modular Design

[**Modular design**](javascript://) is based on combinations of three [**logical structures**](javascript://), sometimes called [**control structures**](javascript://), which serve as building blocks for the process. Each logical structure must have a single entry and exit point. The three structures are called sequence, selection, and iteration. A rectangle represents a step or process, a diamond shape represents a condition or decision, and the logic follows the lines in the direction indicated by the arrows.

1. [**Sequence**](javascript://). The completion of steps in sequential order, one after another, as shown in [Figure 5-24](javascript://). One or more of the steps might represent a subprocess that contains additional logical structures.
2. [**Selection**](javascript://). The completion of one of two or more process steps based on the results of a test or condition. In the example shown in [Figure 5-25](javascript://), the system tests the input, and if the hours are greater than 40, it performs the CALCULATE OVERTIME PAY process.
3. [**Iteration**](javascript://). The completion of a process step that is repeated until a specific condition changes, as shown in [Figure 5-26](javascript://). An example of iteration is a process that continues to print paychecks until it reaches the end of the payroll file. Iteration also is called [**looping**](javascript://).

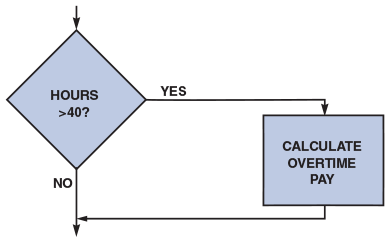
**Figure 5-24**

Sequence structure.



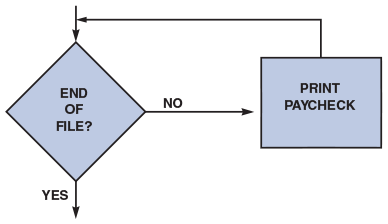
**Figure 5-25**

Selection structure.



**Figure 5-26**

Iteration structure.



Sequence, selection, and iteration structures can be combined in various ways to describe processing logic.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.9.3Structured English

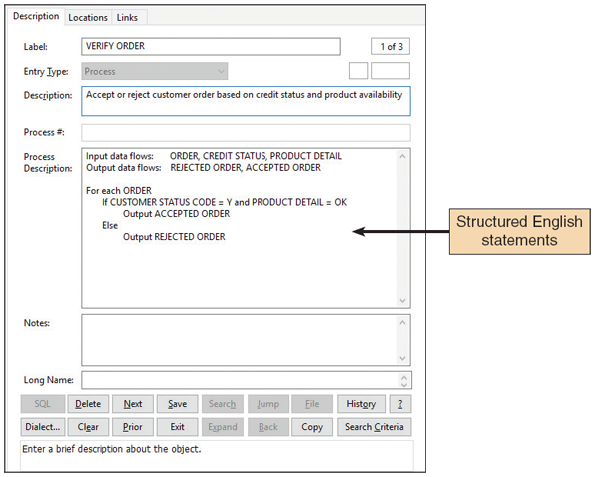
[**Structured English**](javascript://) is a subset of standard English that describes logical processes clearly and accurately. When using structured English, be mindful of the following guidelines:

* Use only the three building blocks of sequence, selection, and iteration.
* Use indentation for readability.
* Use a limited vocabulary, including standard terms used in the data dictionary and specific words that describe the processing rules.

An example of structured English appears in [Figure 5-27](javascript://), which shows the VERIFY ORDER process that was illustrated earlier. Note that the structured English version documents the actual logic that will be coded into the system. Structured English can help process descriptions accurate and understandable to users and system developers.

**Figure 5-27**

The VERIFY ORDER process description includes logical rules and a structured English version of the policy. Note the alignment and indentation of the logic statements.



Enlarge Image

**Source:** Screenshot used with permission from Visible Systems Corp.

Structured English might look familiar to programming students because it resembles [**pseudocode**](javascript://), which is used in program design. Although the techniques are similar, the primary purpose of structured English is to describe the underlying business logic, while programmers, who are concerned with coding, mainly use pseudocode as a shorthand notation for the actual code.

Following structured English rules ensures that process descriptions are understandable to users who must confirm that the process is correct, as well as to other analysts and programmers who must design the information system from the descriptions.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

## 5.9.4Decision Tables

A [**decision table**](javascript://) is a logical structure that shows every combination of conditions and outcomes. Analysts often use decision tables to describe a process and ensure that they have considered all possible situations. Decision tables can be created using tools such as Microsoft PowerPoint, Word, or Excel.

### Tables with One Condition

If a process has a single condition, there only are two possibilities—yes or no. Either the condition is present or it is not, so there are only two rules. For example, to trigger an overtime calculation, the process condition might be: Are the hours greater than 40? If so, the calculation is made. Otherwise, it is not.

### Tables with Two Conditions

Suppose there is a need to create a decision table based on the VERIFY ORDER business process shown in [Figure 5-28](javascript://). When documenting a process, it is important to ensure that every possibility is listed. In this example, the process description contains two conditions: product stock status and customer credit status. If both conditions are met, the order is accepted. Otherwise the order is rejected.

**Figure 5-28**

The VERIFY ORDER business process has two conditions. For an order to be accepted, the product must be in stock and the customer must have an acceptable credit status.

| **VERIFY ORDER Business Process with Two Conditions** |
| --- |
| * An order will be accepted only if the product is in stock and the customer’s credit status is OK. |
| * All other orders will be rejected. |

After all the conditions and outcomes have been identified, the next step is to create a decision table similar to the one shown in [Figure 5-29](javascript://). Note that each condition has two possible values, so the number of rules doubles each time another condition is added. For example, one condition creates two rules, two conditions create four rules, three conditions create eight rules, and so on. In the two-condition example in [Figure 5-29](javascript://), four possibilities exist, but Rule 1 is the only combination that will accept an order.

**Figure 5-29**

Example of a simple decision table showing the processing logic of the VERIFY ORDER process. To create the table, follow the four steps shown.

1. Place the name of the process in a heading at the top left.
2. Enter the conditions under the heading, with one condition per line, to represent the customer status and availability of products.
3. Enter all potential combinations of Y/N for the conditions. Each column represents a numbered possibility called a rule.
4. Place an X in the action entries area for each rule to indicate whether to accept or reject the order.



Enlarge Image

### Tables with Three Conditions

Suppose the company now decides that the credit manager can waive the customer credit requirement, as shown in [Figure 5-30](javascript://). That creates a third condition, so there will be eight possible rules. The new decision table might resemble the one shown in [Figure 5-31](javascript://).

**Figure 5-30**

A third condition has been added to the VERIFY ORDER business process. For an order to be accepted, the product must be in stock and the customer must have an acceptable credit status. However, the credit manager now has the authority to waive the credit status requirements.

| **VERIFY ORDER Business Process with Three Conditions** |
| --- |
| * An order will be accepted only if the product is in stock and the customer’s credit status is OK. |
| * The credit manager can waive the credit status requirement. |
| * All other orders will be rejected. |

Enlarge Table

**Figure 5-31**

This decision table is based on the VERIFY ORDER conditions shown in [Figure 5-30](javascript://). With three conditions, there are eight possible combinations or rules.

| **VERIFY ORDER Process with Credit Waiver (initial version)** | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| Credit status is OK | Y | Y | Y | Y | N | N | N | N |
| Product is in stock | Y | Y | N | N | Y | Y | N | N |
| Waiver from credit manager | Y | N | Y | N | Y | N | Y | N |
| Accept order | X | X |  |  | X |  |  |  |
| Reject order |  |  | X | X |  | X | X | X |

Enlarge Table

First, the Y-N patterns must be filled in, as shown in [Figure 5-31](javascript://). The best way to assure that all combinations appear is to use patterns like these. The first condition uses a pattern of Y-Y-Y-Y followed by N-N-N-N, the second condition uses a repeating Y-Y-N-N pattern, and the pattern in the third condition is a series of Y-Ns.

The next step is very important, regardless of the number of conditions. Each numbered column, or rule, represents a different set of conditions. The logic must be carefully analyzed and the outcome for each rule shown. Before going further, study the table in [Figure 5-31](javascript://) to be sure the logical outcome for each of the eight rules is understood.

When all the outcomes have been determined, the next step is to simplify the table. In a multi-condition table, some rules might be duplicates, redundant, or unrealistic. To simplify the table, follow these steps:

1. 1.

Study each combination of conditions and outcomes. When there are rules with three conditions, only one or two of them may control the outcome, and the other conditions simply do not matter.

1. 2.

If there are conditions that do not affect the outcome, mark them with dashes (-) as shown in the first table in [Figure 5-32](javascript://).

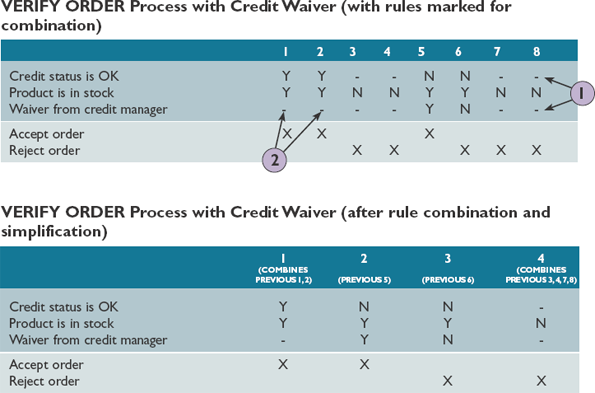
1. 3.

Now combine and renumber the rules, as shown in the second table in [Figure 5-32](javascript://).

**Figure 5-32**

In the first decision table, dashes have been added to indicate that a condition is not relevant. In the second version of the decision table, rules have been combined following the steps shown below. Note that in the final version, only four rules remain. These rules document the logic and will be transformed into program code when the system is developed.

1. Because the product is not in stock, the other conditions do not matter.
2. Because the other conditions are met, the waiver does not matter.



Enlarge Image

After following these steps, Rules 1 and 2 can be combined, because credit status is OK in both rules, so the waiver would not matter. Rules 3, 4, 7, and 8 also can be combined because the product is not in stock, so other conditions do not matter. The result is that instead of eight possibilities, only four logical rules control the Verify Order process.

### Multiple Outcomes

In addition to multiple conditions, decision tables can have more than two possible outcomes. For example, the sales promotion policy shown in [Figure 5-33](javascript://) includes three conditions: Was the customer a preferred customer, did the customer order $1,000 or more, and did the customer use our company charge card? Based on these conditions, four possible actions can occur, as shown in the decision table in [Figure 5-34](javascript://).

**Figure 5-33**

A sales promotion policy with three conditions. Note that the first statement contains two separate conditions: one for the 5% discount and another for the additional discount.

| **SALES PROMOTION POLICY – Holiday Season** |
| --- |
| * Preferred customers who order $1,000 or more are entitled to a 5% discount and an additional 5% discount if they use our charge card. |
| * Preferred customers who do not order $1,000 or more will receive a $25 bonus coupon. |
| * All other customers will receive a $5 bonus coupon. |

Enlarge Table

**Figure 5-34**

This decision table is based on the sales promotion policy in [Figure 5-33](javascript://). This is the initial version of the table before simplification.

| **Sales Promotion Policy (initial version)** | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Preferred customer | Y | Y | Y | Y | N | N | N | N |
| Ordered $1,000 or more | Y | Y | N | N | Y | Y | N | N |
| Used our charge card | Y | N | Y | N | Y | N | Y | N |
| 5% discount | X | X |  |  |  |  |  |  |
| Additional 5% discount | X |  |  |  |  |  |  |  |
| $25 bonus coupon |  |  | X | X |  |  |  |  |
| $5 bonus coupon |  |  |  |  | X | X | X | X |

Enlarge Table

As explained in the preceding section, most tables can be simplified, and this one is no exception. After studying the conditions and outcomes, the following becomes apparent:

* In Rule 1, all three conditions are met, so both 5% discounts apply.
* In Rule 2, a preferred customer orders $1,000 or more but does not use our charge card, so only one 5% discount applies.
* Rules 3 and 4 can be combined into a single rule. Why? If preferred customers do not order $1,000 or more, it does not matter whether they use our charge card—either way, they earn only a $25 bonus coupon. Therefore, Rules 3 and 4 really are a single rule.
* Rules 5, 6, 7, and 8 also can be combined into a single rule—because if the person is not a preferred customer, he or she can only receive a $5 bonus coupon, and the other conditions simply do not matter. A dash is inserted if a condition is irrelevant, as shown in [Figure 5-35](javascript://).

**Figure 5-35**

In this version of the decision table, dashes have been added to indicate that a condition is not relevant. As this point, it appears that several rules can be combined.

| **Sales Promotion Policy (final version)** | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| Preferred customer | Y | Y | Y | Y | N | N | N | N |
| Ordered $1,000 or more | Y | Y | N | N | - | - | - | - |
| Used our charge card | Y | N | - | - | - | - | - | - |
| 5% discount | X | X |  |  |  |  |  |  |
| Additional 5% discount | X |  |  |  |  |  |  |  |
| $25 bonus coupon |  |  | X | X |  |  |  |  |
| $5 bonus coupon |  |  |  |  | X | X | X | X |

Enlarge Table

If dashes are added for rules that are not relevant, the table should resemble the one shown in [Figure 5-35](javascript://). When the results are combined and simplified, only four rules remain: Rule 1, Rule 2, Rule 3 (a combination of initial Rules 3 and 4), and Rule 4 (a combination of initial Rules 5, 6, 7, and 8).

Decision tables often are the best way to describe a complex set of conditions. Many analysts use decision tables because they are easy to construct and understand, and programmers find it easy to work from a decision table when developing code.

**Case in Point 5.2**

### Rock Solid Outfitters (Part 1)

* The marketing director at Rock Solid Outfitters, a medium-sized supplier of outdoor climbing and camping gear, has asked the IT manager to develop a special web-based promotion. Rock Solid will provide free shipping for any customer who either completes an online survey form or signs up for the Rock Solid online newsletter. Additionally, if a customer completes the survey and signs up for the newsletter, Rock Solid will provide a $10 merchandise credit for orders of $100 or more. The IT manager has asked you to develop a decision table that will reflect the promotional rules that a programmer will use. She wants you to show all possibilities, and then to simplify the results to eliminate any combinations that would be unrealistic or redundant. How will you proceed?

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

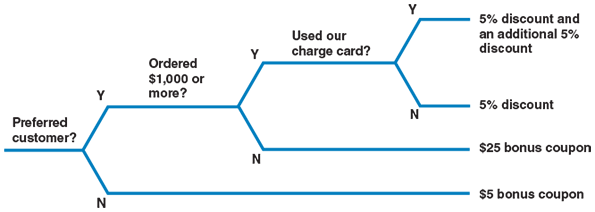
## 5.9.5Decision Trees

A [**decision tree**](javascript://) is a graphical representation of the conditions, actions, and rules found in a decision table. Decision trees show the logic structure in a horizontal form that resembles a tree with the roots at the left and the branches to the right. Like flowcharts, decision trees are useful ways to present the system to management. Decision trees and decision tables provide the same results but in different forms. In many situations, a graphic is the most effective means of communication.

[Figure 5-36](javascript://) is based on the sales promotion policy shown in [Figure 5-33](javascript://). A decision tree is read from left to right, with the conditions along the various branches and the actions at the far right. Because the example has two conditions with four resulting sets of actions, the example has four terminating branches at the right side of the tree.

**Figure 5-36**

This decision tree example is based on the same Sales Promotion Policy shown in the decision tables in [Figures 5-34](javascript://) and [5-35](javascript://). Like a decision table, a decision tree shows all combinations of conditions and outcomes. The main difference is the graphical format, which some viewers may find easier to interpret.



Enlarge Image

Whether to use a decision table or a decision tree often is a matter of personal preference. A decision table might be a better way to handle complex combinations of conditions. On the other hand, a decision tree is an effective way to describe a relatively simple process.

**Case in Point 5.3**

### Rock Solid Outfitters (Part 2)

* The IT manager at Rock Solid Outfitters thinks you did a good job on the decision table task she assigned to you. Now she wants you to use the same data to develop a decision tree that will show all the possibilities for the web-based promotion described in Part 1 of the case. She also wants you to discuss the pros and cons of decision tables versus decision trees. How shall you proceed this time?

### A Question of Ethics

* [iStockphoto.com](https://istockphoto.com/" \t "_blank)/faberfoto\_itThis is your first week in your new job at Safety Zone, a leading producer of IT modeling software. Your prior experience with a smaller competitor gave you an edge in landing the job, and you are excited about joining a larger company in the same field.

So far, all is going well, and you are getting used to the new routine. However, you are concerned about one issue. In your initial meeting with the IT manager, she seemed very interested in the details of your prior position, and some of her questions made you a little uncomfortable. She did not actually ask you to reveal any proprietary information, but she made it clear that Safety Zone likes to know as much as possible about its competitors.

Thinking about it some more, you try to draw a line between information that is OK to discuss and topics such as software specifics or strategy that should be considered private. This is the first time you have ever been in a situation like this. How will you handle it?

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

**5.10**Summary

Structured analysis tools can be used to develop a logical model during one systems analysis phase and a physical model during the systems design phase. Many analysts use a four-model approach, which involves a physical model of the current system, a logical model of the current system, a logical model of the new system, and a physical model of the new system.

During data and process modeling, a systems analyst develops graphical models to show how the system transforms data into useful information. The end product of data and process modeling is a logical model that will support business operations and meet user needs. Data and process modeling involves three main tools: DFDs, a data dictionary, and process descriptions.

DFDs graphically show the movement and transformation of data in the information system. DFDs use four symbols: The process symbol transforms data, the data flow symbol shows data movement, the data store symbol shows data at rest, and the external entity symbol represents someone or something connected to the information system. Various rules and techniques are used to name, number, arrange, and annotate the set of DFDs to make them consistent and understandable.

A set of DFDs is like a pyramid with the context diagram at the top. The context diagram represents the information system’s scope and its external connections but not its internal workings. Diagram 0 displays the information system’s major processes, data stores, and data flows and is the exploded version of the context diagram’s process symbol, which represents the entire information system. Lower-level DFDs show additional detail of the information system through the leveling technique of numbering and partitioning. Leveling continues until the functional primitive processes are reached, which are not decomposed further and are documented with process descriptions. All diagrams must be balanced to ensure their consistency and accuracy.

The data dictionary is the central documentation tool for structured analysis. All data elements, data flows, data stores, processes, entities, and records are documented in the data dictionary. Consolidating documentation in one location allows the analyst to verify the information system’s accuracy and consistency more easily and generate a variety of useful reports.

Each functional primitive process is documented using structured English, decision tables, and decision trees. Structured English uses a subset of standard English that defines each process with combinations of the basic building blocks of sequence, selection, and iteration. Using decision tables or decision trees can also document the logic.

Change font size

[**help**](javascript://)

[Main content](https://ng.cengage.com/static/nbreader/ui/apps/nbreader/nbreader.html#header)

# Chapter Review

## **Key Terms**

* [**alias**](javascript://)
* [**balancing**](javascript://)
* [**black box**](javascript://)
* [**black hole**](javascript://)
* [**business logic**](javascript://)
* [**business rules**](javascript://)
* [**child diagram**](javascript://)
* [**context diagram**](javascript://)
* [**control structures**](javascript://)
* [**data dictionary**](javascript://)
* [**data element**](javascript://)
* [**data flow**](javascript://)
* **data flow diagram (DFD)**
* [**data item**](javascript://)
* [**data repository**](javascript://)
* [**data store**](javascript://)
* [**data structures**](javascript://)
* [**decision table**](javascript://)
* [**decision tree**](javascript://)
* [**decomposing**](javascript://)
* [**diagram 0**](javascript://)
* [**diverging data flow**](javascript://)
* [**domain**](javascript://)
* [**entity**](javascript://)
* [**exploding**](javascript://)
* [**field**](javascript://)
* [**four-model approach**](javascript://)
* [**functional primitive**](javascript://)
* [**Gane and Sarson**](javascript://)
* [**gray hole**](javascript://)
* [**iteration**](javascript://)
* [**leveling**](javascript://)
* [**logical model**](javascript://)
* [**logical structures**](javascript://)
* [**looping**](javascript://)
* [**Modular design**](javascript://)
* [**parent diagram**](javascript://)
* [**partitioning**](javascript://)
* [**physical model**](javascript://)
* [**process**](javascript://)
* [**process 0**](javascript://)
* [**process description**](javascript://)
* [**pseudocode**](javascript://)
* **records**
* [**selection**](javascript://)
* [**sequence**](javascript://)
* [**sink**](javascript://)
* [**source**](javascript://)
* [**spontaneous generation**](javascript://)
* [**structured English**](javascript://)
* [**terminators**](javascript://)
* [**validity rules**](javascript://)
* [**Yourdon**](javascript://)

Change font size

[**help**](javascript://)