Appendix K

The Hierarchical Database Model

Preview

Chapter 2, Data Models, briefly introduced the hierarchical model's history and basic structure. The focus in this appendix is on implementation issues. IBM's Information Management System (IMS) is used to show you how the hierarchical model's basic structures are implemented. Although the hierarchical model now labors on as a mere legacy system, its structure and the implementation issues examined in this appendix remain a valuable part of your database knowledge base. In fact, many hierarchical concepts survive within the modern database environment, thus illustrating that "the more things change, the more they stay the same."

You will discover that, when properly implemented, the hierarchical model creates an environment in which some very important data integrity rules are maintained automatically. If the database design conforms to the hierarchical structure, the hierarchical model yields fast access and is capable of handling large amounts of data.

Data Files and Available Formats

MS Access | Oracle | MS SQL | My SQL

MS Access | Oracle | MS SQL | My SQL

There are no data files for this appendix.

Data Files Available on cengagebrain.com

Recall from Chapter 2 that the hierarchical database model is based on a *tree* structure. You will see how each tree structure is stored in its own (physical) database and how each is defined by a detailed **database description (DBD) statement**. After the physical database has been defined through the DBD, you will see how application programs are given a subset of the physical database through a **program communication block (PCB)**. Although the hierarchical model's application programs tend to be less complex than those written for file systems, the complexity of the database-definition process makes the hierarchical model's implementation more difficult.

database description (DBD) statement
In the hierarchical model, the series of commands that define the hierarchical tree structure and how the segments are stored in the database.

program communication block (PCB)
In a hierarchical database, after the physical database has been defined through the DBD, a way through which application programs are given a subset of the physical database.

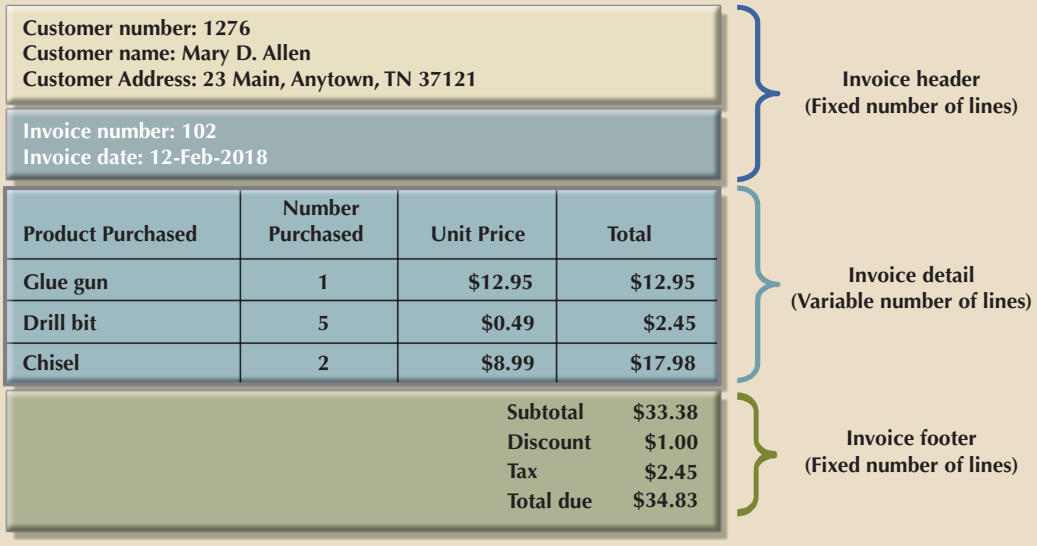
Before reading about how to implement a hierarchical database model, you should understand its basic concepts and components. To review, augment, and illustrate the hierarchical database discussion presented in Chapter 2, you will examine some of the details of the hierarchical database structure in the next two sections.

K-1 A Simple Billing System

One of the billing system's components is the invoice. A typical invoice form, shown in Figure K.1, shows that a customer named Mary D. Allen purchased three items on 12-Feb-2018. Note that the invoice in Figure K.1 contains:

- Basic customer data, such as the customer number, name, and address. The label CUSTOMER will be used to refer to such data.
- Specific invoice data, such as the invoice number and date. The label INVOICE will be used when referring to such data.
- A variable number of invoice detail lines, one for each product bought. The label INVLINE will be used when referring to the invoice detail-line data.
- Computed (derived) data such as subtotals, taxes, and totals.

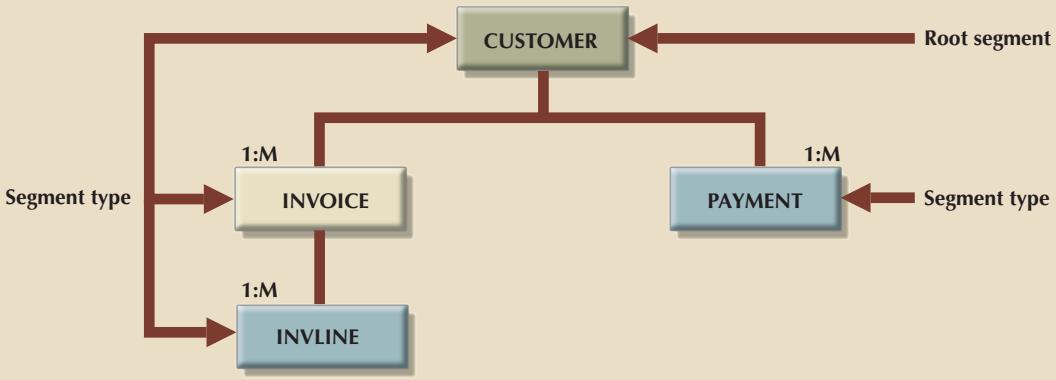
FIGURE K.1 AN INVOICE FORM



Naturally, a billing system contains additional components. Because customers may make purchases on credit, the payments made by customers must be tracked. The label PAYMENT will be used to refer to the customer payment data. To keep the billing system simple, there will be no need to track customer balances at this point.

From a hierarchical point of view, the purchase and payment information introduced thus far can be represented by a hierarchy based on four segment types, CUSTOMER, INVOICE, PAYMENT, and INVLINE, as shown in Figure K.2.

FIGURE K.2 HIERARCHICAL STRUCTURE OF A SAMPLE DATABASE



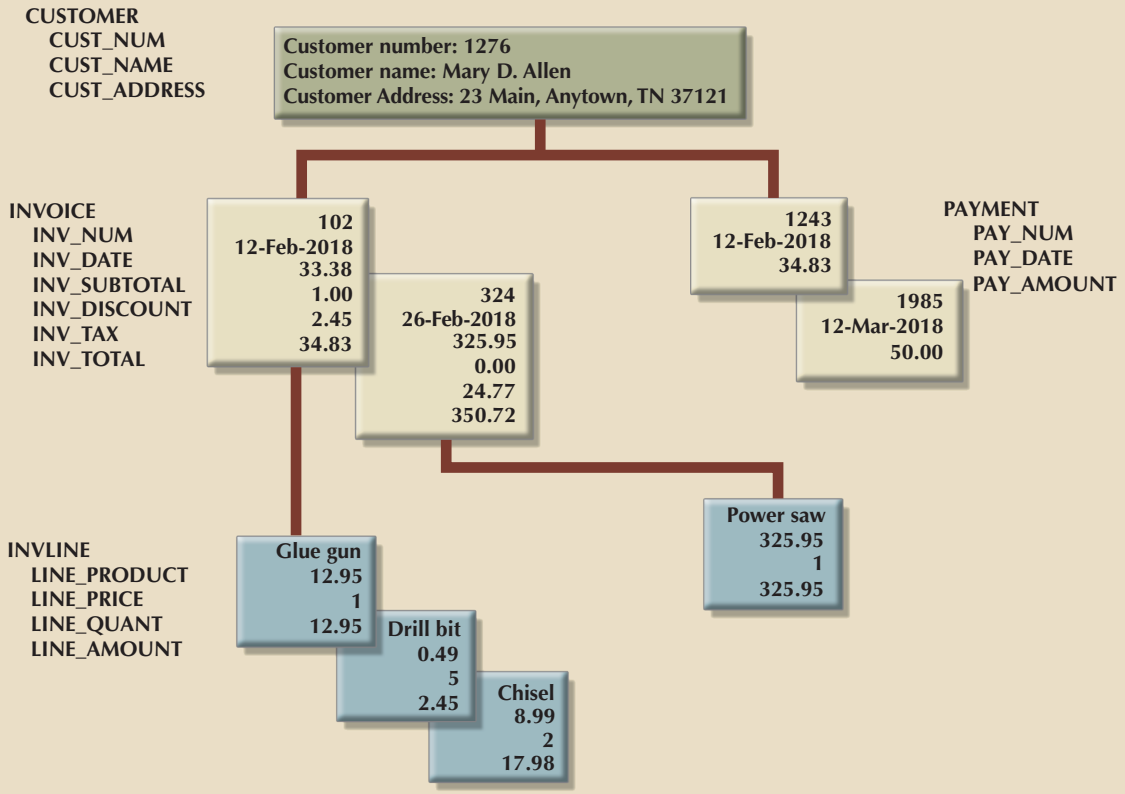
The four segment types or segments you see in Figure K.2 exist within a database named CUSREC. Each *segment type* represents a specific entity set and contains several *segment occurrences*. For example, the CUSTOMER segment type may contain segment occurrences such as Mary D. Allen, John P. Marsutto, and Jean M. Valverde. Given the structures shown in Figure K.2, you can now describe the following relationships:

1. A customer can have one or more invoices and can make one or more payments. However, each payment is made by only one customer, and each invoice belongs to only one customer. In other words, the model depicts a 1:M relationship between CUSTOMER and the two segments INVOICE and PAYMENT. The CUSTOMER segment is the *parent* of the INVOICE and PAYMENT segments.
2. Each INVOICE and each PAYMENT segment occurrence is related to only one CUSTOMER segment occurrence.
3. The INVOICE segment is the parent of the INVLINE segment.
4. Each INVLINE segment occurrence is related to only one INVOICE segment occurrence. Given the conditions in Numbers 3 and 4, you may conclude that a 1:M relationship exists between INVOICE and INVLINE. (Remember, a single INVOICE may contain many items that are entered as *detail lines*.)

Each match of a root segment occurrence with its child segment occurrences represents a hierarchical database *record occurrence*. Figure K.3 illustrates several relationships produced by the root segment occurrence of the customer named Mary D. Allen. As you examine Figure K.3, note that Mary D. Allen's record consists of two INVOICE segment occurrences and two PAYMENT segment occurrences. Invoice number 102 contains the detail lines for the items "Glue gun," "Drill bit," and "Chisel" and is related to Mary D. Allen. The item "Power saw," located in invoice number 324, is also related to Mary D. Allen.

Figure K.3 illustrates that the hierarchical database record is formed by the segment types CUSTOMER, INVOICE, INVLINE, and PAYMENT. The segment components are the equivalent of file fields. In other words, the CUSTOMER segment components CUST_NUMBER, CUST_NAME, and CUST_ADDRESS are equivalent to a file system's CUSTOMER file fields.

FIGURE K.3 A SINGLE OCCURRENCE OF A CUSREC RECORD



The tree structure depicted in Figure K.3 cannot be duplicated (as shown) on the computer's storage media. Instead, the tree is defined by the path that traces the parents to their children, beginning from the left. This ordered sequencing of segments to represent the hierarchical structure is known as the *hierarchical path*.

Given the structure depicted in Figure K.3, the hierarchical path for the record composed of the segments CUSTOMER, INVOICE, INVLINE, and PAYMENT can be traced as shown in Figure K.4. Note that the path followed in Figure K.4 traces all segments from the root, starting at the leftmost segment. The *left-list path* is known as the *preorder traversal* or the *hierarchic sequence*. Given such a path, designers must make sure that the most frequently accessed segments and their components are located closest to the tree's leftmost branches.

K-2 Contrasting File Systems with the Hierarchical Model

To help you better understand the segment concept, it might be useful to examine the relationship between file structures and the hierarchical database. For example, consider the small file system depicted in Figure K.5. Note that the three physical files are connected through the use of pointers. Thus, the *Classical* pointer in the STYLE file points to Beethoven and Tchaikovsky in the ARTIST file. In turn, the ARTIST file's pointers lead to specific music in the MUSIC file.

The file system depicted in Figure K.5 is composed of three distinct physical files: STYLE, ARTIST, and MUSIC. The records in each of the three files are physically isolated

FIGURE K.4 TRACING THE PATH OF A SINGLE HIERARCHICAL RECORD

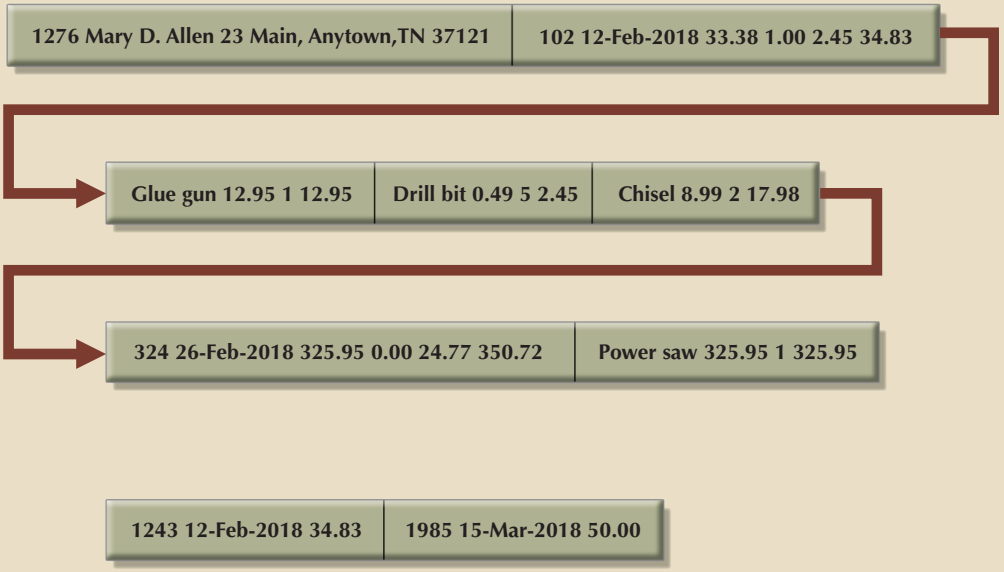
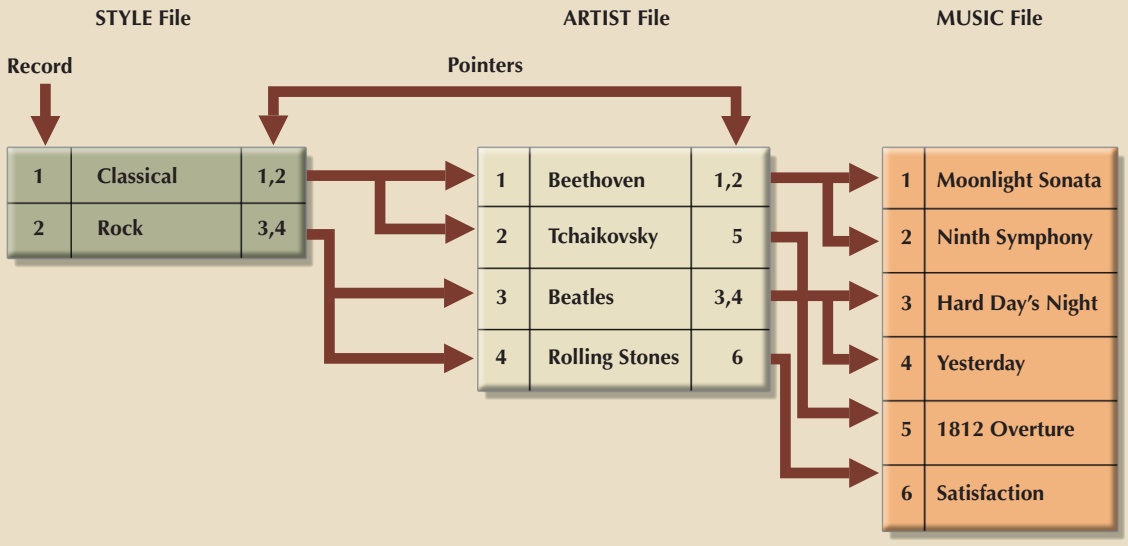


FIGURE K.5 COMPOSITION OF A SMALL FILE SYSTEM



from the records in the other files. Therefore, the first record in each of the three files may be depicted as shown in Table K.1.

In sharp contrast to the file system, the hierarchical model merges the separate physical files into a single structure known as a database. *Therefore, there is no equivalent of a file in the hierarchical model. The fields encountered in the file system are simply segment components in the hierarchical database.*

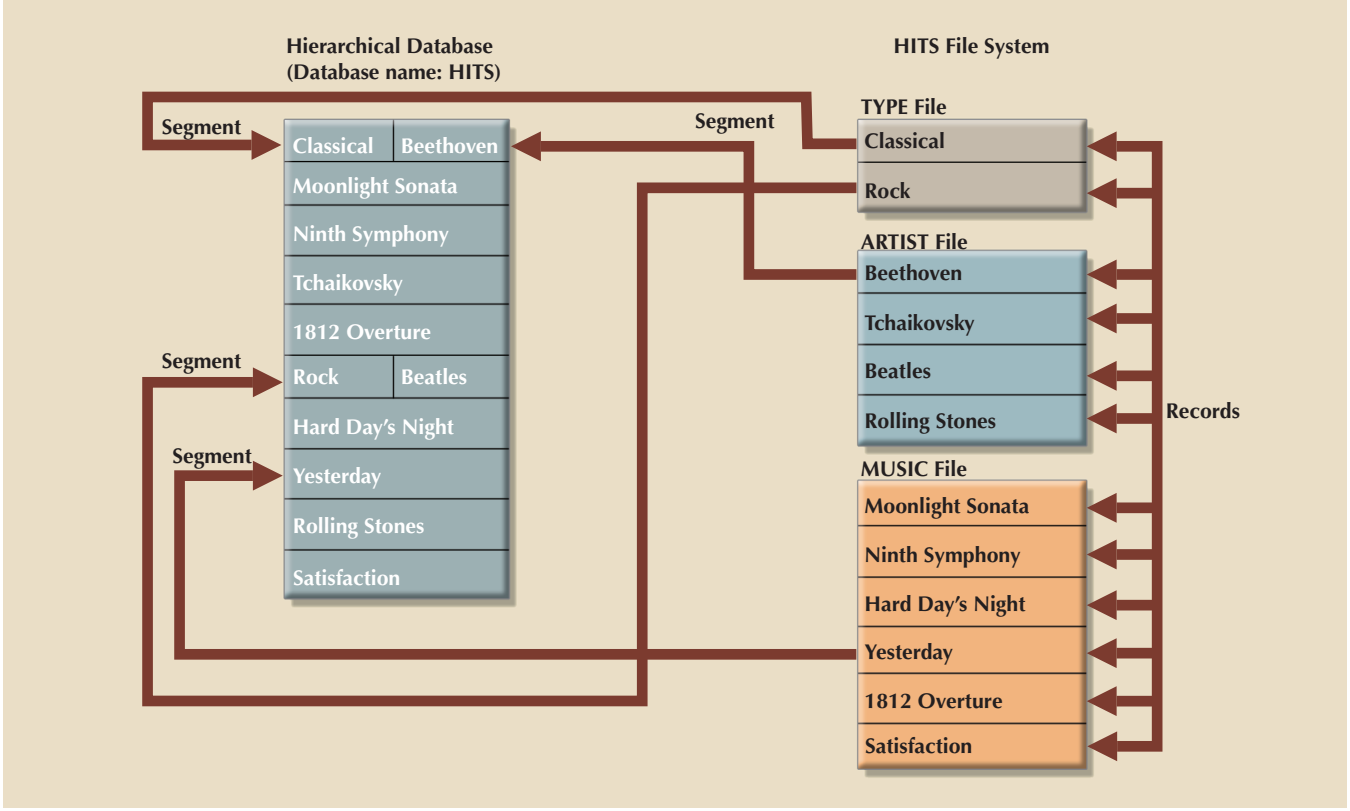
TABLE K.1

SAMPLE M_STORE FILE COMPONENTS

FILE	RECORD
STYLE	Classical
ARTIST	Beethoven
MUSIC	Moonlight Sonata

Translating the small file system into a hierarchical database (named HITS) yields a structure in which each file record becomes a database segment. Thus, the HITS database structure will be composed of three different segment types: STYLE, ARTIST, and MUSIC. Figure K.6 shows you how the file system's records are arranged within a hierarchical database.

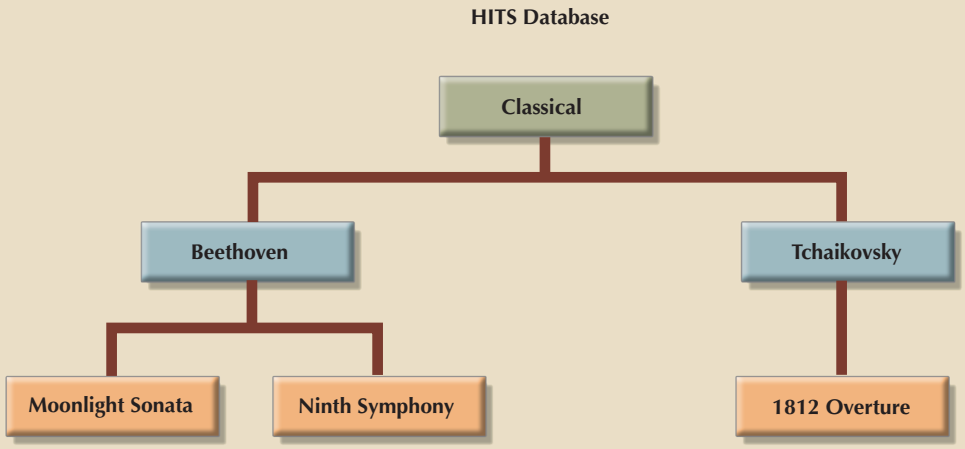
FIGURE K.6 THE HIERARCHICAL EQUIVALENT OF THE FILE SYSTEM



Given the contrasting structures shown in Figure K.6, keep in mind that the file system's user must maintain physical control of the indexes and pointers that validate data integrity. However, in the hierarchical model, the DBMS takes care of those complex chores, and the pointer movement is *transparent* to the user. (The word **transparent** indicates that the user is unaware of the system's operation.) Figure K.7 shows the hierarchical representation of the first database record for the HITS database.

transparent
Indicating that the user is unaware of the system's operations.

FIGURE K.7 THE FIRST HIERARCHICAL DATABASE RECORD



K-3 Defining a Hierarchical Database

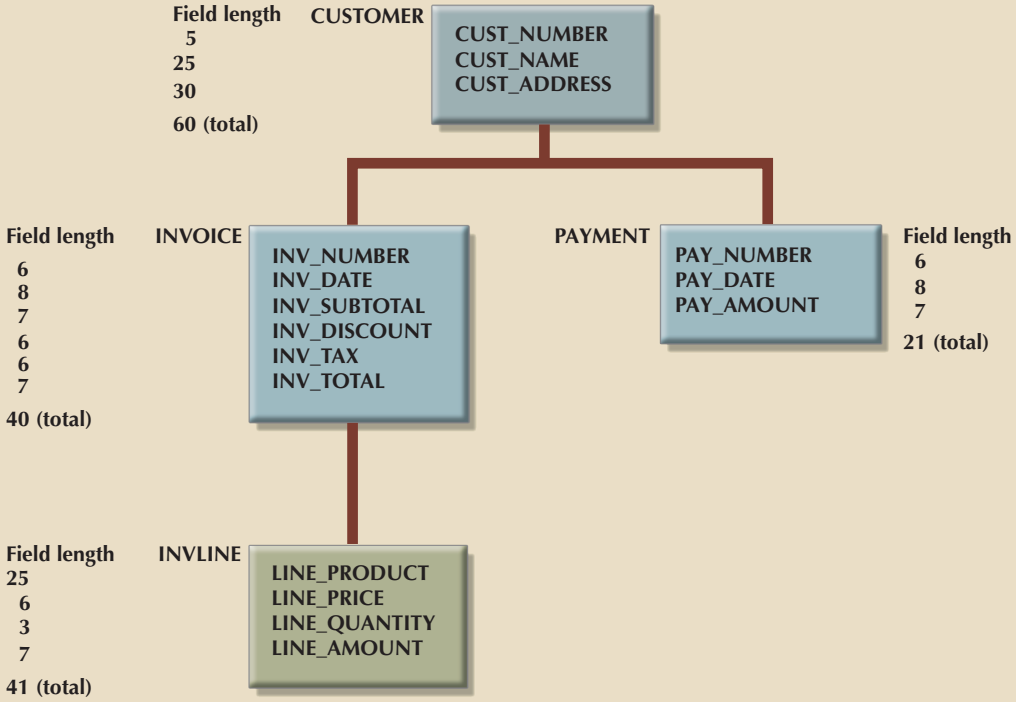
You will now learn how Figure K.2's simple billing system can be implemented through IBM's Information Management System (IMS). IMS uses a language named Data Language One (DL/1). At the conceptual level, IMS may control several databases. Each database is composed of a collection of physical records (segments) that are occurrences of a single tree structure. Therefore, each tree requires its own database. For example, the tree structure depicted in Figure K.8 may be stored in a database named CUSREC to reflect its CUStomer RECord orientation. (Incidentally, you could name the database GEORGE or SALLY; however, it is helpful to give the database a name that describes its contents.) Each of the physical databases is defined by a database description (DBD) statement when the database is created.

The hierarchical model's segment relationships are determined explicitly by the user when the database is defined, using the data definition language (DDL). Those segment relationships do *not* depend on the contents of a field in the child record (as was true in the relational model). Therefore, the relationships between the segments cannot be derived *via* each segment's components or fields. For an illustration of the use of the DDL, refer to the field names as shown in Figure K.8.

As you examine Figure K.8, note that the field lengths, measured in bytes, are shown next to each field. For example, the three fields describing the segment named CUSTOMER are CUST_NUMBER, CUST_NAME, and CUST_ADDRESS and their field lengths are 5, 25, and 30, respectively. Therefore, the segment named CUSTOMER is 5 + 25 + 30 = 60 bytes long. The INVOICE segment is 40 bytes long, the INVLINE segment is 41 bytes long, and the PAYMENT segment is 21 bytes long. Therefore, the total hierarchical database record length is 60 + 40 + 41 + 21 = 162 bytes.

To define the CUSREC database, a simplified syntax of DL/1, the data access-and-manipulation language of IMS, will be used. DL/1 is used to describe the conceptual and logical views of the database. The *conceptual view* encompasses the entire database as seen by the database administrator; the *logical view* describes the programmer's and user's perceptions of the database. Thus, the logical view is more restrictive, limiting the programmer/user to the portion of the database that is currently in use. The existence of logical views constitutes a security measure that helps avoid the unauthorized use of the database. Both the conceptual and logical views are necessary when the database administrator is working with a hierarchical database.

FIGURE K.8 SEGMENT FIELDS AND FIELD LENGTHS (BYTES) IN THE CUSREC DATABASE



K-3a The Conceptual View Definition

Remember from the discussion in Chapter 2 that the tree structure is defined starting from the left. Therefore, the sequence shown in the DDL conforms to the path:

CUSTOMER →INVOICE → INVLINE → PAYMENT

Based on that structure definition, Table K.2 shows the DL/1 statements used to define the conceptual view of the CUSREC database as seen by the database administrator.

TABLE K.2

DL/1 STATEMENTS THAT DEFINE THE CONCEPTUAL VIEW OF THE CUSREC DATABASE

STATEMENT #	CODE	STATEMENT
1	DBD	NAME=CUSREC, ACCESS=HISAM
2	SEGM	NAME=CUSTOMER,BYTES=60
3	FIELD	NAME=(CUST_NUMBER,SEQ,U),BYTES=5,START=1
4	FIELD	NAME=CUST_NAME,BYTES=25,START=6
5	FIELD	NAME=CUST_ADDRESS,BYTES=30,START=31
6	SEGM	NAME= INVOICE, PARENT=CUSTOMER,BYTES=40
7	FIELD	NAME=(INV_NUMBER,SEQ,U),BYTES= 6,START=1
8	FIELD	NAME=INV_DATE,BYTES=8,START=7
9	FIELD	NAME=INV_SUBTOTAL,BYTES=7,START=15

TABLE K.2

DL/1 STATEMENTS THAT DEFINE THE CONCEPTUAL VIEW OF THE CUSREC DATABASE (CONTINUED)

STATEMENT #	CODE	STATEMENT
10	FIELD	NAME=INV_DISCOUNT,BYTES=6,START=22
11	FIELD	NAME=INV_TAX,BYTES=6,START=28
12	FIELD	NAME=INV_TOTAL,BYTES=7,START=34
13	SEGM	NAME=INVLINE,PARENT= INVOICE,BYTES= 42
14	FIELD	NAME=LINE_PRODUCT,SEQ,M),BYTES=25,START=1
15	FIELD	NAME=LINE_PRICE,BYTES=7,START=26
16	FIELD	NAME=LINE_QUANTITY,BYTES=3,START=33
	FIELD	NAME=LINE_AMOUNT,BYTES=7,START=36
17	SEGM	NAME=PAYMENT,PARENT=CUSTOMER,BYTES=21
18	FIELD	NAME=(PAY_NUMBER,SEQ,U),BYTES= 6,START=1
19	FIELD	NAME=PAY_DATE,BYTES=8,START=7
20	FIELD	NAME=PAY_AMOUNT,BYTES=7,START=15
21	DBGEN	
22	FINISH	
23	END	

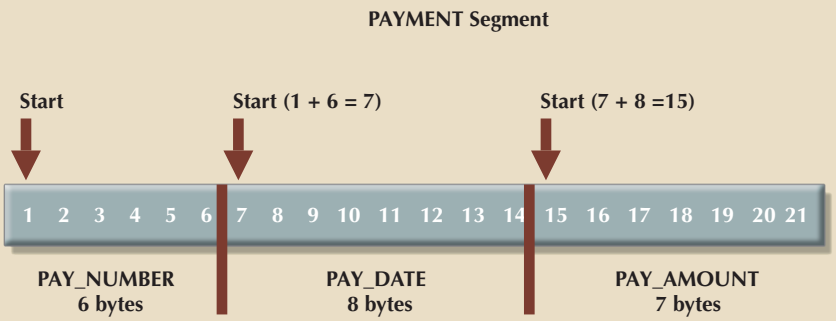
Table K.2's DL/1 lines describe the database and its contents this way:

1. The first line tells IMS that a database named CUSREC is being defined. (The acronym *DBD* stands for *database description*.) The selected access mode is HISAM, or Hierarchical Indexed Sequential Access Method.
2. Line 2 defines the root segment; IMS uses the term **segment (SEGM)** to serve as a reference to the logical records of a database. This example defines the root segment to be the CUSTOMER segment, composed of the fields CUS_NUMBER, CUS_NAME, and CUS_ADDRESS. The CUSTOMER segment is 60 bytes long.
3. Lines 3–5 define the fields that are contained in the CUSTOMER segment. The FIELD specification defines the name, the size, and the starting position for each field making up a segment.
4. Line 3 defines CUST_NUMBER as the Sequence (SEQ) field for the CUSTOMER segment. By definition, the hierarchical database contains ordered collections of records. To avoid having two customers with the same customer number, the ID values are unique (U) for this field.
5. Line 6 defines the INVOICE segment; the parameter PARENT is used to indicate the parent of a segment. The parent of INVOICE is CUSTOMER in this example.
6. Note (in line 14) that there can be multiple (M) occurrences of the product's value.
7. Similar definitions are used for the remaining segments (INVLINE and PAYMENT).

segment (SEGM)
The equivalent of a file record type in the hierarchical database model."

8. **DBGEN** generates the physical database with all of its necessary structures. (The hierarchical database creation is *not* an interactive process.)
9. Note that the hierarchical model's implementation requires that you keep track of physical details such as the number of bytes and the starting position for each field. Figure K.9 illustrates how the starting position for each of the PAYMENT fields is determined.

FIGURE K.9 FIELD STARTING POSITIONS FOR THE PAYMENT SEGMENT



As you can see, the hierarchical model's database definition must conform to its physical characteristics. Even given the simplified DL/1 syntax, the details make the hierarchical model sufficiently complex to be described as a system designed by programmers for programmers. For instance, the physical storage details may require the definition of complex storage schemes such as:

- HSAM (Hierarchical Sequential Access Method).
- SHSAM (Simple Hierarchical Sequential Access Method).
- HISAM (Hierarchical Indexed Sequential Access Method).
- SHISAM (Simple Hierarchical Indexed Sequential Access Method).
- HDAM (Hierarchical Direct Access Method).
- HIDAM (Hierarchical Indexed Direct Access Method).
- MSDB (Main Storage DataBase).
- DEDB (Data Entry DataBase).
- GSAM (Generalized Sequential Access Method).

Specific access methods are best suited to particular kinds of applications. HSAM, SHSAM, HISAM, and SHISAM are particularly well suited for storing and retrieving data in hierarchic sequence, putting parent and children records in contiguous disk locations. (GSAM is a special case of the sequential access method.) On the other hand, if direct-access pointers are required to keep track of the hierarchy of segments, HDAM, HIDAM, MSDB, and DEBD are preferred. That series of access methods is generally more valuable when many (and frequent) changes are made to the database. Generally, the IMS manuals suggest that you:

DBGEN
In the hierarchical model, the component that generates the physical database with all its necessary structures.

1. Use HSAM when relatively small databases with relatively few access requirements are used.
2. Use HISAM with databases that require direct segment access, especially when:
   a. Fixed record lengths are used.

b. All segments are the same size.
c. Few root segments and many child segments exist.
d. Few deletions are made.

3. Use HDAM with databases designed for fast direct access.
4. Use HIDAM with databases having users who require both random (direct) and sequential access.
5. Use MSDB with databases that use fixed-length segments and that require very fast processing. MSDB will reside in virtual storage during execution.
6. Use DEBD with databases that are characterized by high data volume.
7. Use SHSAM, SHISAM, and GSAM when you frequently import and export data between database and nondatabase applications.

Table K.2's database definition requires each segment to be identified by a so-called **sequence field**. The identifier is also known as a **key**. Working with sequence fields requires that you recognize these features and conditions:

- Sequence fields allow direct access to segments when you are working with HISAM, HDAM, or HIDAM access methods. Those access methods make it possible to address segments directly, without having to search the entire database. Direct access increases performance substantially.
- Sequence fields do not have to be defined for every segment.
- Sequence fields may be either unique (U) or duplicate (M).

Keep in mind that an IMS database is rather limited structurally:

- Each database can have a maximum of 255 different segment types.
- Each segment can have a maximum of 255 segment fields.
- Each database can have a maximum of 1,000 different fields.

Having defined the conceptual view of the database, now let's define the logical views for each application program that will access the database.

K-3b The Logical View Definition

The logical view depicts the application program's view. Application programs use embedded DL/1 statements to manipulate the data in the database. Each application that accesses an IMS database requires the creation of a **program specification block (PSB)**. The PSB defines the database(s), segments, and types of operations that can be performed by the application. The PSB represents a logical view of a selected portion of the database. The use of PSBs yields better data security as well as improved program efficiency by allowing access to only the portion of the database that is required to perform a given function.

The application program and the database system communicate through a common storage area in primary memory known as the program communication block (PCB). The PSB contains one or more PCBs, one for each database that is accessed by the application program.

To illustrate the use of the PCB, let's create one for an application that displays customer payments. Since the program access requirements can be defined, you need only be in the database portion defined by the CUSTOMER and the PAYMENT segments shown in Figure K.8. You may then use DL/1 to define the type of access or **processing option (PROCOPT)** granted to the program. The access types are (G)et, (I)nsert,

sequence field
An attribute that contains values that are unique and sequential (ascending or descending). Some DBMS allows the explicit definition of sequence or autonumber attributes that are generally used to uniquely identify each row.

key
An entity identifier based on the concept of functional dependence; keys may be classified in several ways. See also *superkey, candidate key, primary key (PK), secondary key, and foreign key*.

program specification block (PSB)
In a hierarchical database, this represents a logical view of a selected portion of the database and also defines the database(s), segments, and types of operations that can be performed by the application. Using PSBs yields better data security as well as improved program efficiency by allowing access only to the portion of the database that is required to perform a given function.

processing option (PROCOPT)
A type of access granted to a program.

(R)eplace, and (D)elete. Table K.3 shows the appropriate DL/1 statements used to create the PSB for the application.

TABLE K.3

THE DL/1 STATEMENTS USED TO CREATE THE PSB

BLOCK	DL/1 STATEMENT	DEFINITION
1	PCB	DBNAME = CUSREC
2	SENSEG	NAME = CUSTOMER, PROCOPT = G
3	SENSEG	NAME = PAYMENT, PARENT = CUSTOMER, PROCOFF = G
4	SENFLD	NAME = PAY_DATE, START 8
5	SENFLD	NAME = PAY_AMOUNT, START 15
6	PSBCEN	LANG = COBOL, PSBNAME ROBPROG

The **SENSEG (SENsitive SEGment)** declares the segments that will be available, starting with the root segment. The SENFLD indicates which fields are available to the program. In Table K.3's example, all of the CUSTOMER fields will be available, but only the PAY_DATE and PAY_AMOUNT will be available in the PAYMENT segment because the PAY_NUMBER field was omitted. (*Note*: The logical views may be limited to only a portion of a physical database or to parts of *several* different physical databases.)

The creation of the database structure and the PSBs is not based on interactive operations. Instead, independent utility programs that run from the operating-system prompt must be used. Therefore, the database definitions must be re-created (recompiled) and reloaded, *and all of the user views must be re-created and validated if any changes are made to the database*.

The order of the SEGM statements indicates the physical order of the records in the database. In other words, the physical order represents the hierarchical path that must be followed to access any segment. In this case, the order of the segments is shown in Table K.4.

TABLE K.4

THE HIERARCHICAL PATH FOR THE CUSREC DATABASE

HIERARCHICAL PATH	SAMPLE DATA
CUSTOMER 1	Mary D. Allen
INVOICE 1	102
INVLINE 1	Glue gun
INVLINE 2	Drill bit
INVLINE 3	Chisel
INVOICE 2	324
INVLINE 1	Power saw
PAYMENT 1	1243
PAYMENT 2	1985
CUSTOMER 2	John G. Washington

SENSEG (SENsitive SEGment)
In the IMS hierarchical database, this keyword declares the segments that will be available, starting with the root segment.

TABLE K.4

THE HIERARCHICAL PATH FOR THE CUSREC DATABASE (CONTINUED)

HIERARCHICAL PATH	SAMPLE DATA
INVOICE 1	410
INVLINE 1	Grease pencils
INVLINE 2	Masking tape
INVOICE 2	306
INVLINE 1	Computer paper
INVLINE 2	Ink-jet cartridge
…	…
…	…

Remember that IMS provides support for several different data-access methods. Some are very efficient at sequential file processing; others work well in an indexed file environment; yet others work best in a direct-access environment. The example shown in Table K.3 assumes the use of the HSAM storage structure in which the database is represented as an ordered sequence of segments and all dependent segments are located close to their parent segments for fast sequential access.

K-4 Loading IMS Databases

An IMS database must be loaded before any program can access it. You cannot load a database from an interactive application program. Instead, a batch program must be used to perform the loading, and this batch program must be run in "load" mode (PROCOPT=L in the PCB).

The database must be loaded in the proper hierarchic sequence; *the segment order is crucial*. (Load the parent segments before loading the child segments!) If you have defined sequence fields, the segment order must conform to the sequence field order. You must maintain the proper segment order, or the subsequent applications programs will fail.

K-5 Accessing the Database

Hierarchical databases are so-called *record-at-a-time* databases. The term **record at a time** indicates that the database commands affect a single record at a time. You may remember that other database types, such as the relational database, allow a command to affect several (many) records at a time.

The record-at-a-time structure implies that each record is accessed independently when database operations are performed. Therefore, to access a specific record, you must follow the tree's hierarchical path, starting at the root and following the appropriate branches of the tree, using preorder traversal. For example, if you want to access the payments of CUSTOMER Mary D. Allen, you must first access the parent segment, after which you can access the first PAYMENT child, then the next PAYMENT child, and so on, until you have accessed all PAYMENT segments in the subtree. (Remember that PAYMENT segments are ordered by the PAY_NUMBER field.) Similarly, if you want to access the INVOICE segment occurrences, you must first access the parent CUSTOMER segment; then you must access the INVOICE segment occurrences, starting with the first

record at a time
This term indicates that the database commands affect a single record at a time.

one. For each INVOICE, you can access the subtree of INVLINE segment occurrences for that INVOICE. (Remember that the segments are ordered according to the field specified as the sequence field when the database is defined.)

After the database and its characteristics have been defined, you can navigate through the database by using the data manipulation language (DML) invoked from some host language such as COBOL, PL/1, or assembler. Keep in mind that some lines of code must be written by an experienced programmer before you can access the database. Given the complexity of the hierarchical database environment, end users are not likely to have the technical expertise to generate even the simplest query output, thus putting "spur-of-the-moment" queries out of reach. For example, a query such as "list all customers who reside in the 12345 zip code" requires detailed knowledge of the hierarchical database's physical file structure and the physical storage details. (In contrast, that query is easy to generate in a relational database environment, merely requiring the execution of the brief SQL command SELECT * FROM CUSTOMER WHERE CUST_ZIP = "12345".)

IMS requires the use of a (3GL) host language such as COBOL to access the database. To communicate with the application program correctly, IMS assumes the use of certain parameters. Therefore, each application must declare:

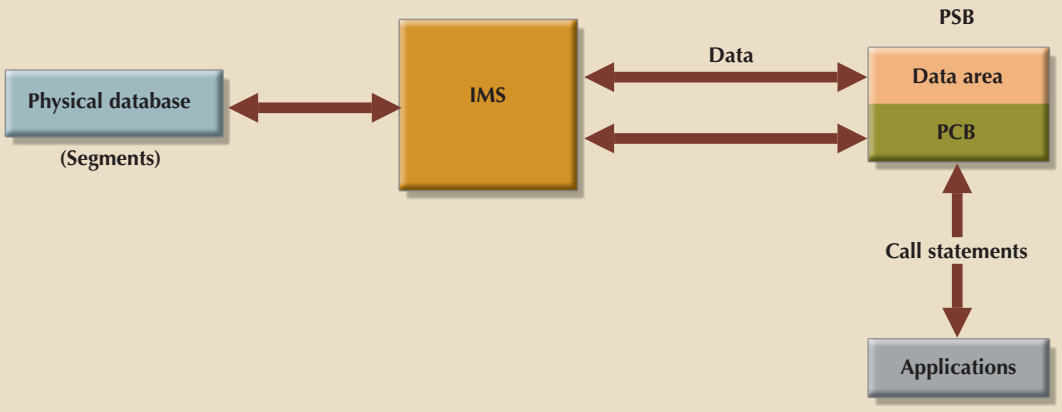
1. An *input* area (program record area) reserved to:
   a. Receive the data retrieved from the database.
   b. Temporarily store data to be written into the database.
2. A PCB to store a return code for every operation that is executed. (The program must check this area to see if the requested operation was completed successfully.)

A COBOL application communicates with the IMS DBMS through *call* statements in its procedure division. Figure K.10 illustrates the use of the PCB. When the application program calls IMS, the following *flow parameters* are needed:

- The function code, that is, the operation to be executed on the database.
- The PCB name to be used.
- The input area address.
- The (optional) segment search argument (SSA). The SSA parameter identifies the key of the segment being retrieved.

After the completion of a call to the database, the program must check the status of the return code in the PCB to ensure that the operation was executed correctly.

FIGURE K.10 HOW A PCB IS USED



K-5a Data Retrieval: Get Unique

The IMS statement **Get Unique (GU)** is used to retrieve a database segment into the application program input area or record area. The syntax for the Get Unique statement is:

CU (segment) (SSA)

Using the data shown earlier in Figure K.3, the GU statement required to retrieve the customer Mary D. Allen must read:

GU CUSTOMER (CUST_NUMBER=1276)

Similarly, the GU statement:

GU INVOICE (INV_NUMBER=102)

will retrieve the INVOICE segment whose number is 102. If HSAM is being used, the DBMS will search the database sequentially until it finds the INVOICE segment whose field value INV_NUMBER is 102. If this INV_NUMBER is the last segment in the database, the DBMS will have searched the entire database, thereby producing significant performance degradation. It is strongly recommended that the hierarchical path be specified to maximize the DBMS performance!

Retrieval by a nonkey field is also possible; for example:

GU CUSTOMER (CUST_NAME 'Mary D. Allen')

will achieve its intended purpose. If two customers have the same name, that command will retrieve the *first* segment that satisfies the condition.

Logical operators may be used to search for several customer records that meet a specified condition. For instance, the GU statements:

GU CUSTOMER (CUST_NUMBER > 1034)

and

GU CUSTOMER (CUST_NUMBER < =1167)

are both valid. If the user fails to specify the SSA, the database search automatically locates the first segment of the database. Therefore, the GU statement:

GU CUSTOMER

will yield the first CUSTOMER segment.

IMS can retrieve more than one segment at a time. For example, if you want to access the INVOICE segment with an INV_NUMBER = 102 as well as its parent CUSTOMER segment, the command:

```
GU CUSTOMER *D
    INVOICE (IN_NUMBER=102)
```

will retrieve both the parent CUSTOMER segment and the *specified* INVOICE child segment; the *D indicates that the user wants to retrieve both. In contrast:

```
GU CUSTOMER
    INVOICE
```

will retrieve only the first INVOICE segment found. IMS always retrieves the *last* referenced segment unless the *D is used.

Get Unique (GU)
In an IMS hierarchical DBMS, a statement that is used to retrieve a database segment into the application program input area or record area.

K-5b Sequential Retrieval: Get Next

The **Get Next (GN)** statement is used to retrieve segments sequentially. (Naturally, the retrieval sequence is based on the preorder traversal requirements.) The GN syntax conforms to the format:

GN (segment) SSA

For example, the statements in Table K.5 will retrieve all payments. (Note that *Pseudocode* has been used to indicate the use of some programming language to complete the request.)

TABLE K.5

RETRIEVE ALL PAYMENTS

PSEUDOCODE	COMMENTS
GU PAYMENT	Retrieve 1st PAYMENT segment.
DO WHILE PCB-CODE IS OKAY	Check the PCB return code.
PRINT (PAY_NUMBER, PAY_DATE)	Process the segment.
ENDDO	

Similarly, if you want to retrieve all payments over $1,000, you would write the pseudocode shown in Table K.6.

TABLE K.6

RETRIEVE ALL PAYMENTS OVER 1000

PSEUDOCODE	COMMENTS
GU PAYMENT (PAY_AMOUNT> 1000)	Retrieve the first PAYMENT segment.
DO WHILE PCB-CODE IS OKAY	Check the PCB return code.
PRINT (PAY_NUMBER,PAY_DATE)	Print the requested data.
GNPAYMENT (PAY_AMOUNT> 1000)	Retrieve the next PAYMENT segment.
ENDDO	

Note

The use of OKAY indicates that the return code is correct. The return code is part of the PCB.

Get Next (GN)
In hierarchical databases, a statement to retrieve sequential segments.

Get Next within Parent (GNP)
In hierarchical databases, a statement to return all segments within the current parent.

K-5c Get Next within Parent

Get Next within Parent (GNP) will return all of the segments within the current parent. The following command sequence will retrieve all INVOICE segments for the CUSTOMER whose CUST_NUMBER= 1276 in the preorder traversal sequence shown in Table K.7.

TABLE K.7

RETRIEVE INVOICES FOR SPECIFIED CUSTOMER

PSEUDOCODE	COMMENTS
GU CUSTOMER (CUSTOMER=1276)	Retrieve the first INVOICE segment for customer 1276.
INVOICE	
DO WHILE PCB-CODE IS OKAY	
…….	
…….(process segment)	
…….	
GNP INVOICE	Retrieve the next INVOICE segment for customer 1276.
ENDDO	

K-5d Data Deletion and Replacement

The **Get Hold (GH)** statement is used to hold a segment for delete or replace operations. There are three different Get Hold statements, as shown in Table K.8.

TABLE K.8

GET HOLD STATEMENTS

STATEMENT	MEANING
GHU	Get Hold Unique
GHN	Get Hold Next
GHNP	Get Hold Next Within Parent

Used in combination with the GH statement, DLET deletes a segment occurrence from the database. For example, to delete the PAYMENT segment numbered 1985 in Table K.3, you would use:

```
GHU CUSTOME R (CUSTOMER=1276)
      PAYMENT (PAY_NUMBER=1985)
DLET
```

If a root segment is deleted, all dependent segments are deleted. Therefore, the command sequence:

```
CHU CUSTOMER (CUST_NUMBER=1276)
DLET
```

will delete the occurrence Mary D. Allen in the CUSTOMER segment and all dependent segments (INVOICE, INVLINE, and PAYMENT).

The REPL statement allows you to change (update) the contents of a field within a segment. REPL also requires the GH operation before it can be invoked. Keep in mind that the REPL function cannot be used to update a key field. Instead, first delete the record, then insert the updated version. The application program should use the input

Get Hold (GH)
In an IMS hierarchical DBMS, this statement is used to hold a segment for delete or replace operations. There are three different Get Hold statements: Get Hold Next (GHN), Get Hold Next within Parent (GHNP), and Get Hold Unique (GHU).

area to store the necessary fields that are to be updated and the new values. The operation sequence thus becomes:

1. Retrieve the data and put it in the input area.
2. Make the changes in the input area.
3. Invoke REPL to move the changed values into the physical database.

For example, to change Mary Allen's address, you can use the pseudocode shown in Table K.9.

TABLE K.9

UPDATE FIELD CONTENTS FOR A SPECIFIED CUSTOMER

PSEUDOCODE	COMMENTS
GU CUSTOMER (CUST_NUMBER = 1276)	Find the CUSTOMER segment.
STORE '103 E. Main St. D-44' TO CUST_ADDRESS	Move the data to the input area.
REPL	Save the data to the disk.

K-5e Adding a New Segment to the Database

The **Insert (ISRT)** statement is used to add a segment to the database. The parent segment must already exist if a child segment is to be inserted. The segment will be inserted in the database in the sequence field order specified for the segment.

The input area in the applications program must contain the data to be stored in the segment. Therefore, if you want to insert the segment PAYMENT for customer number 1276, you write the pseudocode shown in Table K.10.

TABLE K.10

ADDING A NEW SEGMENT

PSEUDOCODE	COMMENTS
STORE 1632 TO PAY_NUMBER	Move the data into the input area.
STORE '20180315' TO PAY_DATE	
STORE 345.66 TO PAY_AMOUNT	
ISRT CUSTOMER (CUS_NUMBER= 1276)	Insert the field values. (Naturally, customer 1276 must exist in the database.)
PAYMENT	

K-6 Logical Relationships

Suppose you want to keep product information in the database system. Further suppose that the product information is to be stored in an INVENTORY database and that you want this database to be related to the CUSREC database. Because the invoice lines contain product information, the PRODUCT segment in the INVENTORY database must be related to the INVLINE segment in the CUSREC database.

Given the preceding scenario, you face the problem of having a segment with two parents, a condition that cannot be easily supported by the hierarchical model. The

Insert (ISRT)
In hierarchical databases, a statement used to add a segment to the database.

multiple-parent problem can be solved by creating a logical relationship between INVLINE and PRODUCT in which INVLINE becomes the logical child of PRODUCT and PRODUCT becomes the logical parent of INVLINE. Unfortunately, this solution has some drawbacks.

- Implementing such a solution yields an even more complex applications environment.
- Creating logical parent/child relationships is very complex and requires the services of an experienced programmer. To accomplish the task, referential rules must be defined for each of the operations (Insert, Replace, and Delete) for each logical segment involved in the two physical databases. The rules may be unidirectional or bidirectional depending on which way the database is to be accessed.

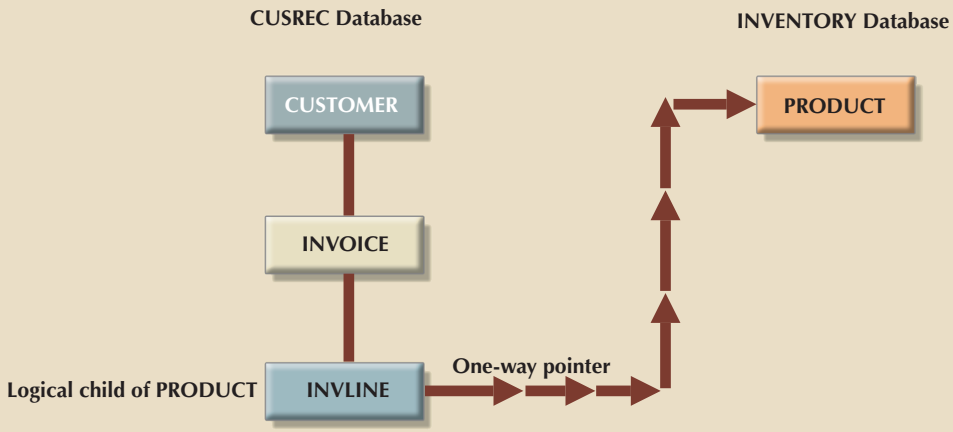
Nonetheless, using logical relationships, you can link two independent physical databases and treat them as though they were one. Thus, logical relationships allow you to reduce data redundancy. In addition, IMS can manage all of the data required to link the databases in logical relationships; it is always better to have the DBMS software do the delicate work of keeping track of such data rather than trust the applications software to do those chores.

IMS supports three different types of logical relationships, as follows:

1. **Unidirectional logical relationships** are established by linking a logical child with a logical parent in a one-way arrangement. In this case, a pointer in the logical child points to the logical parent. (See Figure K.11.)

unidirectional logical relationships
In a hierarchical database, relationships that are established by linking a logical child with a logical parent in a one-way arrangement.

FIGURE K.11 A UNIDIRECTIONAL LOGICAL RELATIONSHIP



The two segments may be in the same database, or they may be located in different databases. If the two segments of the unidirectional relationship are located in different databases, the segments are treated independently of one another. Therefore, if a parent segment is deleted, the logical children are not deleted (see Figure K.12) because the logical parent does not point to the logical child.

1. **Bidirectional physically paired logical relationships** link a logical child with its logical parent in two directions. IMS creates a duplicate of the child segment in the logical parent's database and manages all operations (Insert, Delete, Replace) applied to the segments, as shown in Figure K.13. IMS uses pointers in the logical child segments pointing to their logical parents. The segments may be in one database, or

bidirectional physically paired logical relationships
In the hierarchical model, a relationship that links a logical child with its logical parent in two directions.

FIGURE K.12 TWO UNIDIRECTIONAL LOGICAL RELATIONSHIPS

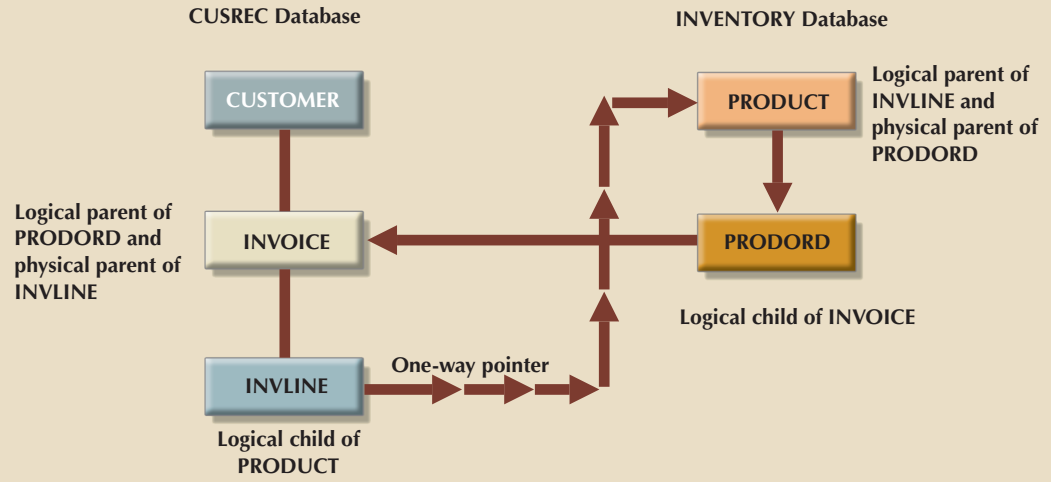
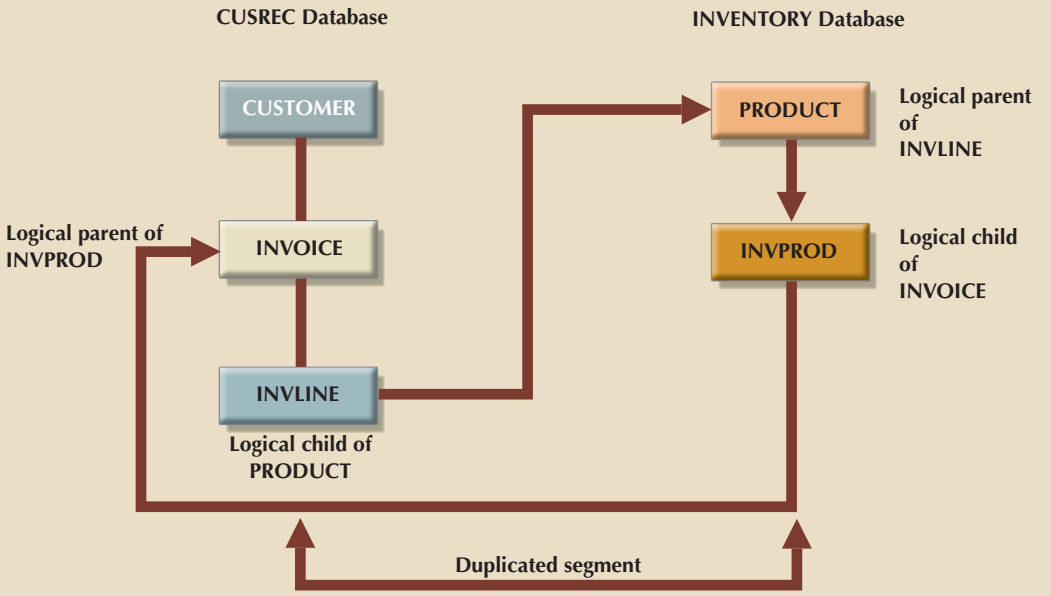


FIGURE K.13 BIDIRECTIONAL PHYSICALLY PAIRED LOGICAL RELATIONSHIPS

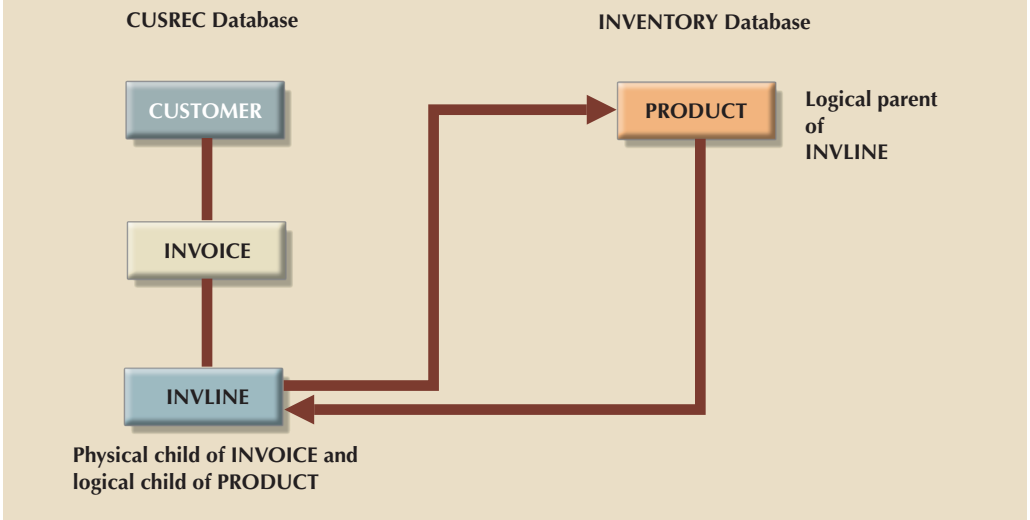


they may be in different (physical) databases. In this type of relationship, the user can navigate from the CUSREC database to the INVENTORY database, and vice versa, because a two-way link exists between the INVOICE and the PRODUCT segments through their common child INVLINE. Although the process creates data redundancy, IMS manages the redundancies transparently.

2. **Bidirectional virtually paired logical relationships** are created when a logical child segment is linked to its logical parent in two directions. The virtually paired relationship is different from the physically paired relationship in that no duplicates are created; IMS stores one pointer in the logical parent to point to the logical child's database and another pointer in the logical child to point to the logical parent. Thus, the virtually paired method reduces data duplication and overhead in the management of both hierarchical paths. (See Figure K.14.)

bidirectional virtually paired logical relationships
In the hierarchical model, a relationship created when a logical child segment is linked to its logical parent in two directions. The virtually paired relationship is different from the physically paired relationship in that no duplicates are created.

FIGURE K.14 A BIDIRECTIONAL VIRTUALLY PAIRED LOGICAL RELATIONSHIP



The creation of bidirectional virtually paired logical relationships is a delicate, cumbersome task that requires a skilled designer with extensive knowledge of the physical details this task requires. For example, if you want to implement logical relationships, IMS requires that you follow the rules listed in Table K.11.

TABLE K.11

RULES FOR DEFINING LOGICAL RELATIONSHIPS IN PHYSICAL DATABASES

RULE	LOGICAL CHILD
1	A logical child must have a physical and a logical parent.
2	A logical child can have only one physical and one logical parent.
3	A logical child is defined as a physical child in the physical database of its physical parent.
4	A logical child is always a dependent segment in a physical database and can, therefore, be defined at any level except the first level of the database.
5	In its physical database, a logical child cannot have a physical child defined at the next lower level in the database that is also a logical child.
6	A logical child can have a physical child. However, if the logical child is physically paired with another logical child, only one of the paired segments can have physical children.

TABLE K.11

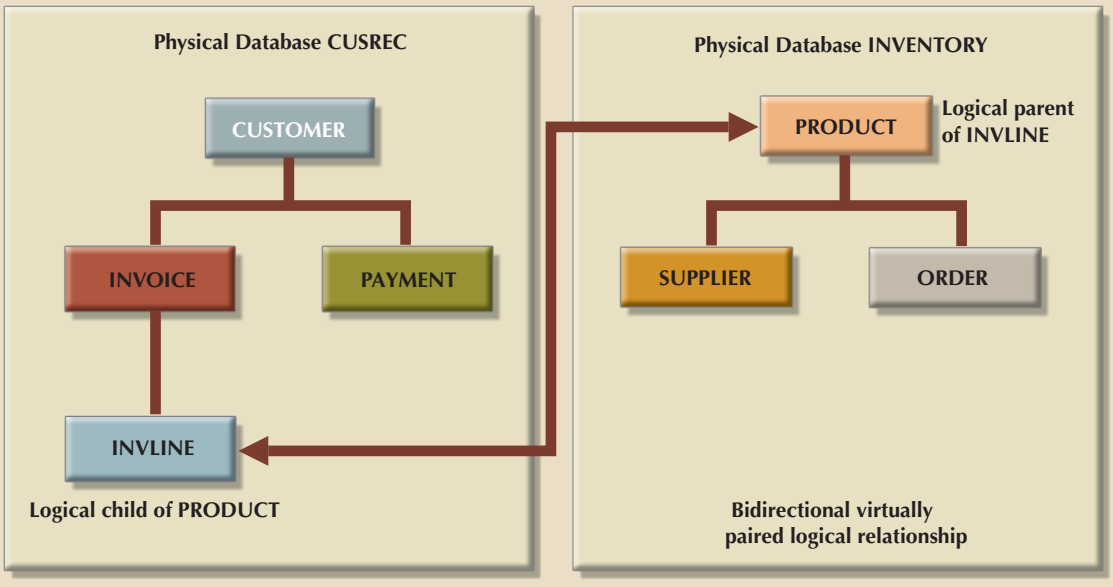
RULES FOR DEFINING LOGICAL RELATIONSHIPS IN PHYSICAL DATABASES (CONTINUED)

RULE	LOGICAL PARENT
1	A logical parent can be defined at any level in the physical database, including the root level.
2	A logical parent can have one or more logical children. Each logical child related to the same logical parent defines a logical relationship.
3	A segment in a physical database cannot be defined as both a logical parent and a logical child.
4	A logical parent can be defined in the same physical database as its logical child or in a different database.
RULE	**PHYSICAL PARENT**
1	A physical parent of a logical child cannot also be a logical child.

Source: IBM IMS Manual, IMS/ESA Version 3 Database Administration Guide, Release 1, 2nd ed., October, 1990, Purchase, NY 10577, pp. 155–56.

Assuming that the designer has the required knowledge of the implementation details, you can conclude that using logical relationships solves the problem of relating INVLINE and PRODUCT by creating a logical link between the two database segments, as shown in Figure K.15.

FIGURE K.15 A BIDIRECTIONAL VIRTUALLY PAIRED LOGICAL RELATIONSHIP BETWEEN TWO DATABASES



Based on the structure shown in Figure K.15, PRODUCT will be the logical parent of INVLINE and INVLINE will be the logical child of PRODUCT. Therefore, (I)nsert, (R)eplace, and (D)elete rules must be defined for each segment in the relationship—for CUSTOMER, INVOICE, and INVLINE in the CUSREC database and for PRODUCT in the INVENTORY database. For example, if a CUSTOMER segment is erased, all of the corresponding CUSTOMER children must be erased, too.

Similarly, if a PRODUCT segment is to be deleted, all of the corresponding INVLINE segments must also be deleted.

The use of logical parents is rather limited. One of DL/1's restrictions is that any given segment can have only one logical parent. That restriction severely limits IMS's ability to deal with complex structures. In fact, the two-parent problem is one of the reasons the network model examined in Appendix L was developed.

K-7 Altering the Hierarchical Database Structure

The hierarchical model's database structure modifications are cumbersome. For example, suppose the sales department manager asks the data processing department's database administrator to add a VENDOR field to the INVOICE segment. That is a simple request, yet even that minor alteration is not naturally supported by the hierarchical system.

Database modifications require the performance of the following tasks in sequence:

1. Unload the database.
2. Define the new database structure.
3. Load the old database into the new structure.
4. Delete the old database.

Since those four tasks are time-consuming and potentially dangerous from a database point of view, database structure modifications require very careful planning, excellent system coordination skills, and a high level of technical understanding of the DBMS.

Key Terms

bidirectional physically paired logical relationships, K-19

bidirectional virtually paired logical relationships, K-21

database description (DBD) statement, K-2

DBGEN, K-10

Get Hold (GH), K-17

Get Next (GN), K-16

Get Next within Parent (GNP), K-16

Get Unique (GU), K-15

Insert (ISRT), K-18

key, K-11

processing option (PROCOPT), K-11

program communication block (PCB), K-2

program specification block (PSB), K-11

record at a time, K-13

segment (SEGM), K-9

SENSEG (SENsitive SEGment), K-12

sequence field, K-11

transparent, K-6

unidirectional logical relationships, K-19