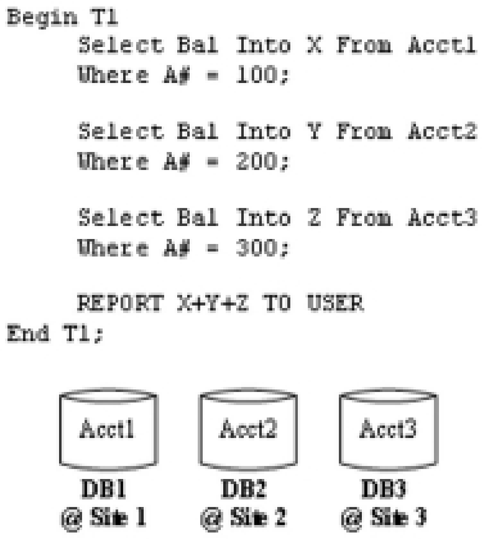


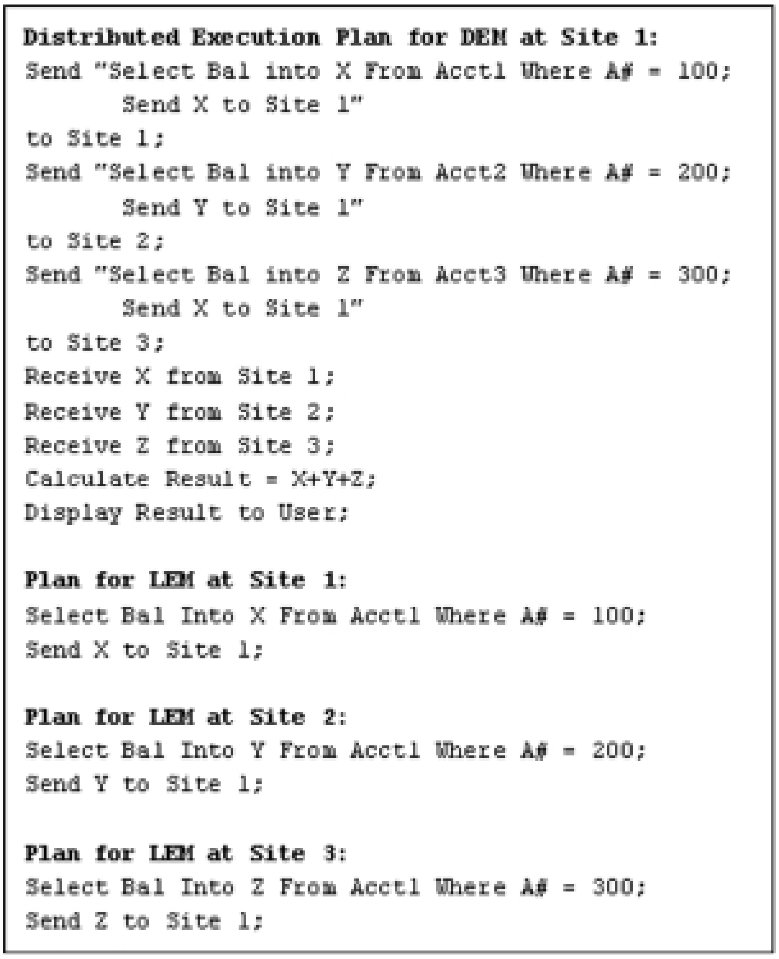
```
Begin T1
     Select Bal Into X From Acct1
     Where A# = 100;

     Select Bal Into Y From Acct2
     Where A# = 200;

     Select Bal Into Z From Acct3
     Where A# = 300;

     REPORT X+Y+Z TO USER
End T1;
```

Figure 1.14 An example of a three-site distributed transaction.

Example 1.1 Consider a three-site system as shown in Figure 1.14, where transaction T1 enters the system at Site 1. Transaction T1 needs to reads the total balance for accounts 100, 200, and 300. Let's assume that the account table is horizontally fragmented across three sites. Site 1 holds accounts 1 to 150, Site 2 holds accounts 151 to 250, and Site 3 holds accounts 251 and higher. If X, Y, and Z represent the balances for accounts 100, 200, and 300, respectively, we can execute this distributed transaction as follows:

```
Send "necessary commands" to Site 1 to read "X" from DB1;
Send "necessary commands" to Site 2 to read "Y" from DB2;
Send "necessary commands" to Site 3 to read "Z" from DB3;
Receive "X" from Site 1;
Receive "Y" from Site 2;
Receive "Z" from Site 3;
Calculate Result = X + Y + Z;
Display Result to User;
```

Note that the Decomposer must know that the account table has three horizontal fragments called Acct1, Acct2, and Acct3. Based on the fragmentation assumptions we have made, account 100 is stored at Site 1, account 200 is stored at Site 2, and account 300 is stored at Site 3. For this example, the DEM needs to send two commands to the LEM at Site 2. One command is to read the balance of account 200. This command in SQL is "Select bal into Y from Account where A# = 200." The second command is an instruction to Site 2 to send the results back to Site 1. Site 1 packages these two commands in one message and sends the message to Site 2. In response, Site 2 sends the balance of account 200 in variable Y, as part of a response message, back to Site 1. Similar commands are sent to Site 1 and Site 3.



```
Distributed Execution Plan for DEM at Site 1:
Send "Select Bal into X From Acct1 Where A# = 100;
       Send X to Site 1"
to Site 1;
Send "Select Bal into Y From Acct2 Where A# = 200;
       Send Y to Site 1"
to Site 2;
Send "Select Bal into Z From Acct3 Where A# = 300;
       Send X to Site 1"
to Site 3;
Receive X from Site 1;
Receive Y from Site 2;
Receive Z from Site 3;
Calculate Result = X+Y+Z;
Display Result to User;

Plan for LEM at Site 1:
Select Bal Into X From Acct1 Where A# = 100;
Send X to Site 1;

Plan for LEM at Site 2:
Select Bal Into Y From Acct1 Where A# = 200;
Send Y to Site 1;

Plan for LEM at Site 3:
Select Bal Into Z From Acct1 Where A# = 300;
Send Z to Site 1;
```

Figure 1.15 Detailed execution plans for DEM and its LEMs.

Since Site 1 cannot add variables X, Y, and Z until all three variables arrive at its site, Site 1 must perform a blocking receive to achieve the necessary synchronization. A blocking receive is a receive command that blocks the process that executes it until the sought data has arrived at the site. Figure 1.15 shows the detailed distributed execution plan and individual plans for each of the three LEMs. In order for the DEM and the LEMs to be able to carry out their tasks, some services are required.

At a minimum, the system must have the ability to:

- Find the server address where the agent is going to run
- Remotely activate a process—DEM must activate its agents at remote sites
- Synchronize processes across sites—DEM must wait for LEMs to send their results to continue
- Transfer commands and receive results from its agents
- Transfer temporary data across sites—a select may return more than one row as its results

- Find out and ignore any commands that might be out of order or should not be processed
- Remotely deactivate a process—DEM needs to deactivate its agents

In the approach we used for coordinating the activities of the DEM and its LEMs, the DEM acted as the coordinator (Master) and the LEMs acted as the followers (Slaves) [Larson85]. Figure 1.16 depicts the Master–Slave approach to distributed execution management.

In the Master–Slave approach, the DEM synchronizes all temporary data movements. In this approach, LEMs send temporary data only to the DEM. Sometimes it is more beneficial to send temporary data from one LEM directly to another LEM. In this case, the DEM needs to send the proper commands and actions to the LEMs so that the LEMs can synchronize their own activities. Larson [Larson85] calls this type of control triangular distributed execution control. Figure 1.17 shows this type of control across one DEM and two LEMs.

Example 1.2 For this example, we assume the system has four sites. The user enters transaction T into the system at Site 0. There are three tables in the system defined as:

```
Person (SSN, Name, Profession, sal) stored at Site 1
Car (Name, Make, Year, SSN) stored at Site 2
Home (Address, No-of-rooms, SSN) stored at Site 3
```

For this distributed system, we want to print car and home address information for every person who makes more than $100,000. Applying a triangular distributed transaction control approach to this example results in the local execution plans shown in Figure 1.18. There are four LEMs in this system. These LEMs are numbered 0 through 3, indicating the site at which each runs. Although we do not have any data stored at Site 0, we must have one LEM there to perform the final join across the temporary tables that other LEMs send to it. This LEM acts as the merger, which we discussed in Section 1.1.

The synchronization mechanism used to coordinate the activities of the LEMs is implemented via the send and receive commands. When a LEM executes a receive command, it blocks itself until the corresponding data arrives at its site (this is an example of a blocking execution call or a blocking messaging call; see Section 14.3.1

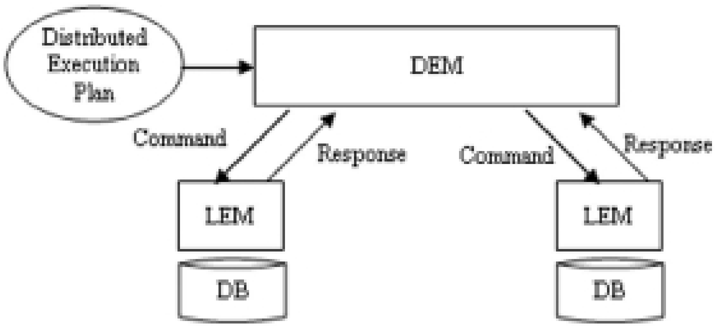


Figure 1.16 Master–Slave distributed transaction control.

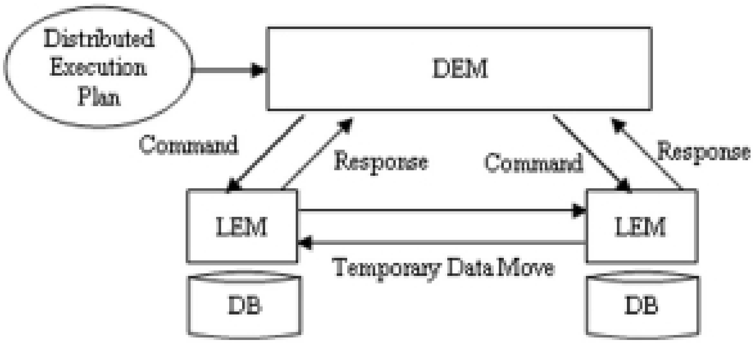
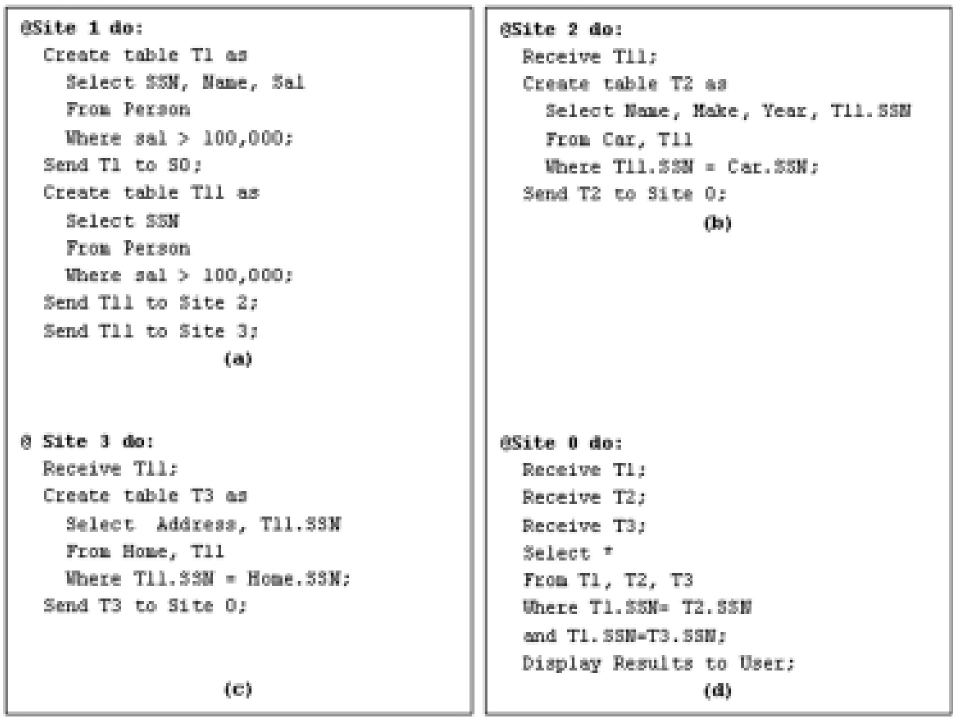


Figure 1.17 Triangular distributed transaction control.



```
@Site 1 do:
  Create table T1 as
    Select SSN, Name, Sal
    From Person
    Where sal > 100,000;
  Send T1 to S0;
  Create table T11 as
    Select SSN
    From Person
    Where sal > 100,000;
  Send T11 to Site 2;
  Send T11 to Site 3;
                (a)
```

```
@Site 2 do:
  Receive T11;
  Create table T2 as
    Select Name, Make, Year, T11.SSN
    From Car, T11
    Where T11.SSN = Car.SSN;
  Send T2 to Site 0;
                (b)
```

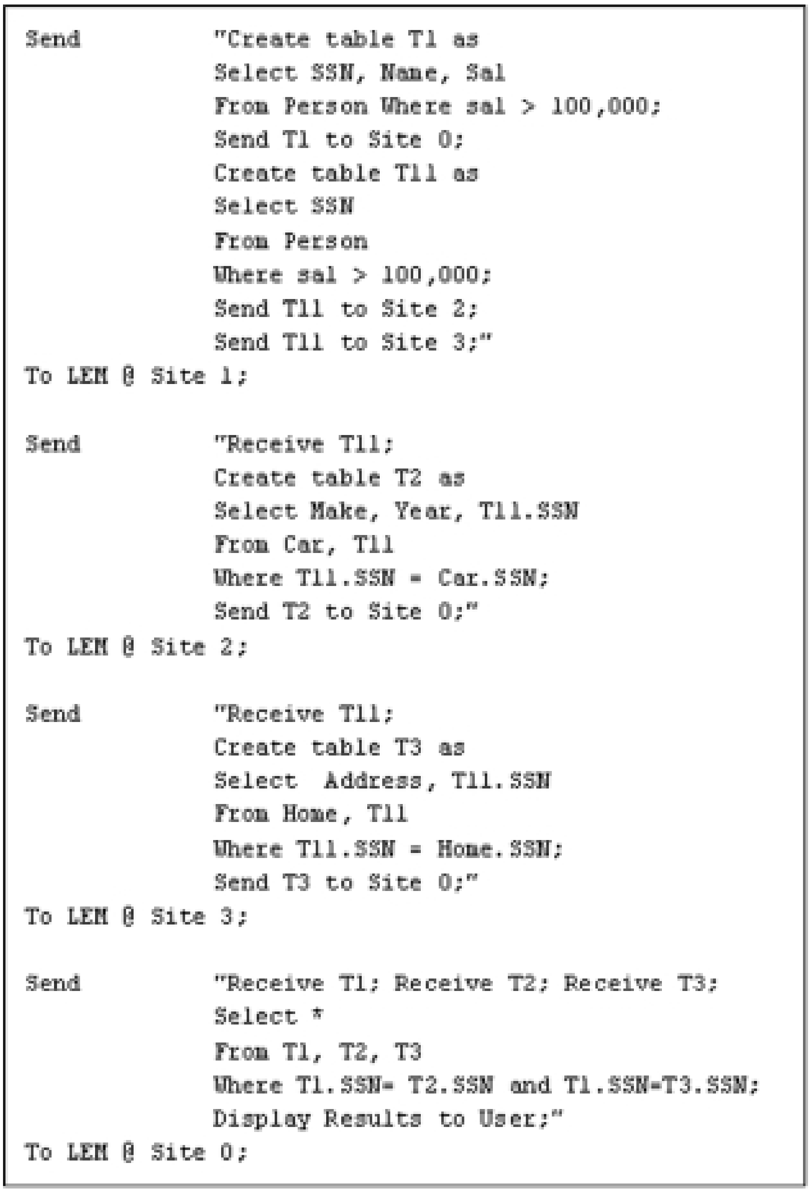
```
@ Site 3 do:
  Receive T11;
  Create table T3 as
    Select  Address, T11.SSN
    From Home, T11
    Where T11.SSN = Home.SSN;
  Send T3 to Site 0;

                (c)
```

```
@Site 0 do:
  Receive T1;
  Receive T2;
  Receive T3;
  Select *
  From T1, T2, T3
  Where T1.SSN= T2.SSN
  and T1.SSN=T3.SSN;
  Display Results to User;
                (d)
```

Figure 1.18 Local execution plans for Example 1.2.

for further details). For our example, the only LEM that can start processing immediately after activation is LEM 1 (see Figure 1.18a). That is because this LEM does not execute a receive command right away. On the other hand, all the other LEMs execute a receive command first. This action blocks these processes until the necessary data arrives at their sites.

It should be clear by now that the distributed execution plan—the plan that the DEM must run—contains the local commands that must be sent to the four LEMs.



```
Send          "Create table T1 as
              Select SSN, Name, Sal
              From Person Where sal > 100,000;
              Send T1 to Site 0;
              Create table T11 as
              Select SSN
              From Person
              Where sal > 100,000;
              Send T11 to Site 2;
              Send T11 to Site 3;"
To LEM @ Site 1;

Send          "Receive T11;
              Create table T2 as
              Select Make, Year, T11.SSN
              From Car, T11
              Where T11.SSN = Car.SSN;
              Send T2 to Site 0;"
To LEM @ Site 2;

Send          "Receive T11;
              Create table T3 as
              Select  Address, T11.SSN
              From Home, T11
              Where T11.SSN = Home.SSN;
              Send T3 to Site 0;"
To LEM @ Site 3;

Send          "Receive T1; Receive T2; Receive T3;
              Select *
              From T1, T2, T3
              Where T1.SSN= T2.SSN and T1.SSN=T3.SSN;
              Display Results to User;"
To LEM @ Site 0;
```

Figure 1.19 Distributed execution plan for Example 1.2.

Figure 1.19 depicts the distributed execution plan for this example. This execution strategy obviously uses a triangular control approach. As seen from this figure, the distributed execution plan contains only four messages that the DEM must send. Each one of these messages consists of the commands that one LEM must run. Obviously, since the DEM and LEM 0 are at the same site (Site 0), the message that the DEM sends to LEM 0 is a local message.

Example 1.3 Assume the table "EMP(EmpID, Name, Loc, Sal)" is horizontally fragmented into fragments MPLS_Farg, LA_Frag, and NY_Frag. Each horizontal fragment stores information about those employees who work at the corresponding location. LA_Frag stores information about employees who work in LA, NY_Frag contains information about employees who work in NY, and MPLS_Frag contains information about employees who work in MPLS. Transaction T enters the DEM deployed in St. Paul and needs to figure out the average salary for all employees in the company. An unoptimized query execution plan would try to reconstruct the EMP table from its fragments by sending all rows from all fragments to St. Paul and then using the SQL aggregate function AVG(sal) as

```
Select AVG(sal) From EMP;
```

If each row of the table takes one message to send from any site to any other site, this strategy would require as many messages as the total number of employees in the organization. However, we can execute this transaction using a small number of messages. The idea is not to materialize the EMP table but to figure out the average salary from the three fragments mathematically as

```
AVG(sal) =
(SAL_LA+SAL_MPLS+SAL_NY)/(count_LA+count_MPLS+count_NY)
```

In this formula, each "SAL" variable represents the total salary for the employees in its corresponding fragment, and each "count" variable indicates the total number of employees in its fragment. Once we realize this, we can use the Master–Slave control as shown Figure 1.20 to get the results. Note that in this figure, we use the LEM at St. Paul as the merger of the results from the local sites. Since this LEM does not have to perform any database operation, there is no need for a DBMS at St. Paul. In a

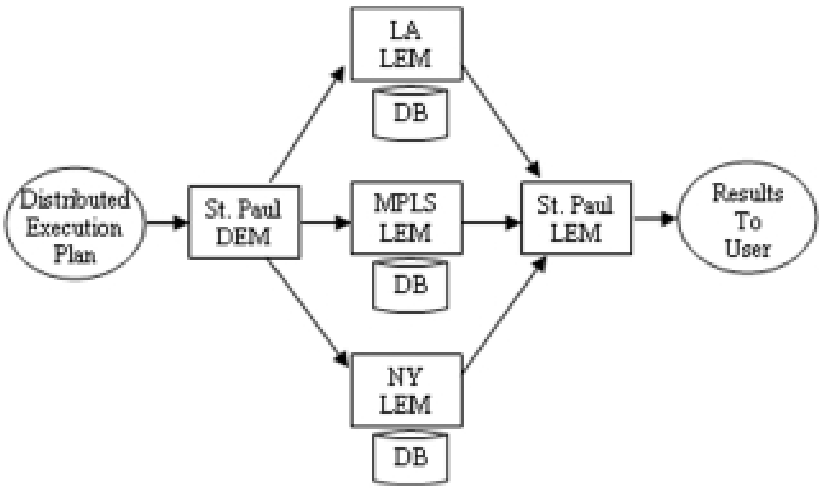


Figure 1.20 Master–Slave execution plan for query in Example 1.3.

```
LEM @ LA:
Select count(*), SUM(Salary) from LA_Frag into count_LA, SAL_LA;
Send "count_LA and SAL_LA" to LEM @ St. Paul;
Communication cost = 1 message

LEM @ MPLS:
Select count(*), SUM(Salary) from MPLS_Frag into count_MPLS, SAL_MPLS;
Send "count_MPLS and SAL_MPLS" to LEM @ St. Paul;
Communication cost = 1 message

LEM @ NY:
Select count(*), SUM(Salary) from NY_Frag into count_NY, SAL_NY;
Send "count_NY and SAL_NY" to LEM @ St. Paul;
Communication cost = 1 message

LEM @ St. Paul:
Receive count_LA and SAL_LA from LEM @ LA;
Receive count_MPLS and SAL_MPLS from LEM @ MPLS;
Receive count_NY and SAL_NY from LEM @ NY;
Calculate AVG = (SAL_LA+SAL_MPLS+SAL_NY)/(count_MPLS+count_LA+count_NY);
Display to the user;
Communication cost = 0

DEM @ NY:
Send "Select count(*), SUM(Salary) from MPLS_Frag into count_MPLS, SAL_MPLS;
      Send "count_MPLS and SAL_MPLS" to LEM @ St. Paul;" to LEM @ MPLS;
Send "Select count(*), SUM(Salary) from LA_Frag into count_LA, SAL_LA;
      Send "count_LA and SAL_LA" to LEM @ St. Paul;" to LEM @ LA;
Send "Select count(*), SUM(Salary) from NY_Frag into count_NY, SAL_NY;
      Send "count_NY and SAL_NY" to LEM @ St. Paul;" to LEM @ NY;
Communication cost = 3 messages
```

Figure 1.21 Distributed and local execution plans for query in Example 1.3.

nonoptimized approach as discussed above, the St. Paul LEM would require a DBMS to run the select statement that calculates the average salary.

Figure 1.21 shows the local execution plans for each LEM and the distributed execution plan for the DEM. As seen from this figure, the Master–Salve execution control takes six messages. A careful reader realizes that using a triangular execution control would require only four messages. We leave this as an exercise for the readers.

1.8 SUMMARY

Centralized database environments and specifically distributed database environments are complex systems composed of many subsystems, components, and services. This book discusses issues in implementing a DDBE. The emphasis of the book is on the practical aspects of implementing a DDBE. In this chapter, we have outlined the architecture of a DDBE and have discussed approaches to controlling the execution of a transaction.