```
CREATE TRIGGER TRANS_AfterInsertSetAskingPrice
   ON TRANS
   AFTER INSERT

AS
BEGIN
   SET NOCOUNT ON;

   DECLARE   @PriorRowCount      AS Int,
             @WorkID             AS Int,
             @TransactionID      AS Int,
             @AcquisitionPrice   AS Numeric(8,2),
             @NewAskingPrice     AS Numeric(8,2),
             @SumNetProfit       AS Numeric(8,2),
             @AvgNetProfit       AS Numeric(8,2)

   SELECT @TransactionID = TransactionID,
          @AcquisitionPrice = AcquisitionPrice,
          @WorkID = WorkID
   FROM   inserted;

   -- First find if work has been here before.

   SELECT    *
   FROM      dbo.TRANS AS T
   WHERE     T.WorkID = @WorkID;

   -- Since this is an AFTER trigger, @@Rowcount includes the new row.

   SET @PriorRowCount = (@@Rowcount - 1)

   IF (@PriorRowCount = 0)
      -- This is first time work has been in the gallery.
      -- Set @NewAskingPrice to twice the acquisition cost.
      SET @NewAskingPrice = (2 * @AcquisitionPrice)

   ELSE
      -- The work has been here before
      -- We have to determine the value of @NewAskingPrice
      BEGIN
         SELECT    @SumNetProfit = SUM(NetProfit)
         FROM      dbo.ArtistWorkNetView AWNV
         WHERE     AWNV.WorkID = @WorkID
         GROUP BY  AWNV.WorkID;

         SET @AvgNetProfit = (@SumNetProfit / @PriorRowCount);

            -- Now choose larger value for the new AskingPrice.

         IF ((@AcquisitionPrice + @AvgNetProfit)
               > (2 * @AcquisitionPrice))
            SET @NewAskingPrice = (@AcquisitionPrice + @AvgNetProfit)
         ELSE
            SET @NewAskingPrice = (2 * @AcquisitionPrice)
      END
```

Figure 10-32

The SQL Statements for the TRANS_After InsertSetAskingPrice Trigger

Amazon Ex. 1008.0422

```
    -- Update TRANS with the value of AskingPrice
    UPDATE dbo.TRANS
    SET    AskingPrice = @NewAskingPrice
    WHERE  TransactionID = @TransactionID;

    -- The INSERT is completed. Print output
    BEGIN
          PRINT '*****************************************************'
          PRINT ''
          PRINT '   INSERT complete. '
          PRINT ''
          PRINT '   TransactionID     = '+CONVERT(Char(6), @TransactionID)
          PRINT '   WorkID            = '+CONVERT(Char(6), @WorkID)
          PRINT '   Acquisition Price = '+CONVERT(Char(12), @AcquisitionPrice)
          PRINT '   Asking Price      = '+CONVERT(Char(12), @NewAskingPrice)
          PRINT ''
          PRINT '*****************************************************'
    END
END
```

Figure 10-32 (Continued)

The variable @PriorRowCount is set to @@rowCount minus one because this is an AFTER trigger, and the new row will already be in the database. The expression (@@rowCount-1) is the correct number of qualifying TRANS rows that were in the database prior to the insert.

We test to see if the work is new. If so, @NewAskingPrice is then set to twice the AcquisitionPrice. If the work has been in the gallery before, we must calculate which value of @NewAskingPrice to use. Thus, if @PriorRowCount is greater than zero, there were TRANS rows for this work in the database, and the ArtistWorkNet view (see pages 270–271) is used to obtain the sum of the WorkNetProfit for the work. The AVG built-in function cannot be used because it will compute the average using @@rowCount rather than @PriorRowCount. Next, the two possible values of @AskingNewPrice are compared, and @NewAskingPrice is set to the larger value. Finally, an update is made to TRANS to set the computed value of @newPrice for AskingPrice.

> **BY THE WAY** In the two stored procedures we discussed in the previous section, the @@rowCount function could have been used instead of counting rows using COUNT(*) to set the value of the variable @RowCount. Just remember to check it or store it immediately after the SQL statement is executed.

To test the trigger, we will begin by obtaining a new work for the View Ridge Gallery. Because Melinda Gliddens just bought the only copy of the print John Singer Sargent's *Spanish Dancer*, we will replace it:

```
INSERT INTO WORK VALUES(
     'Spanish Dancer', '635/750', 'High Quality Limited Print',
     'American Realist style - From work in Spain', 11);
-- Obtain the new WorkID
SELECT        WorkID
FROM          dbo.WORK
WHERE         ArtistID = 11
    AND       Title = 'Spanish Dancer'
    AND       Copy = '635/750';
-- Use the new WorkID value (597 in this case)
INSERT INTO TRANS (DateAcquired, AcquisitionPrice, WorkID)
VALUES ('06/8/2008', 200.00, 597);
```

Amazon Ex. 1008.0423

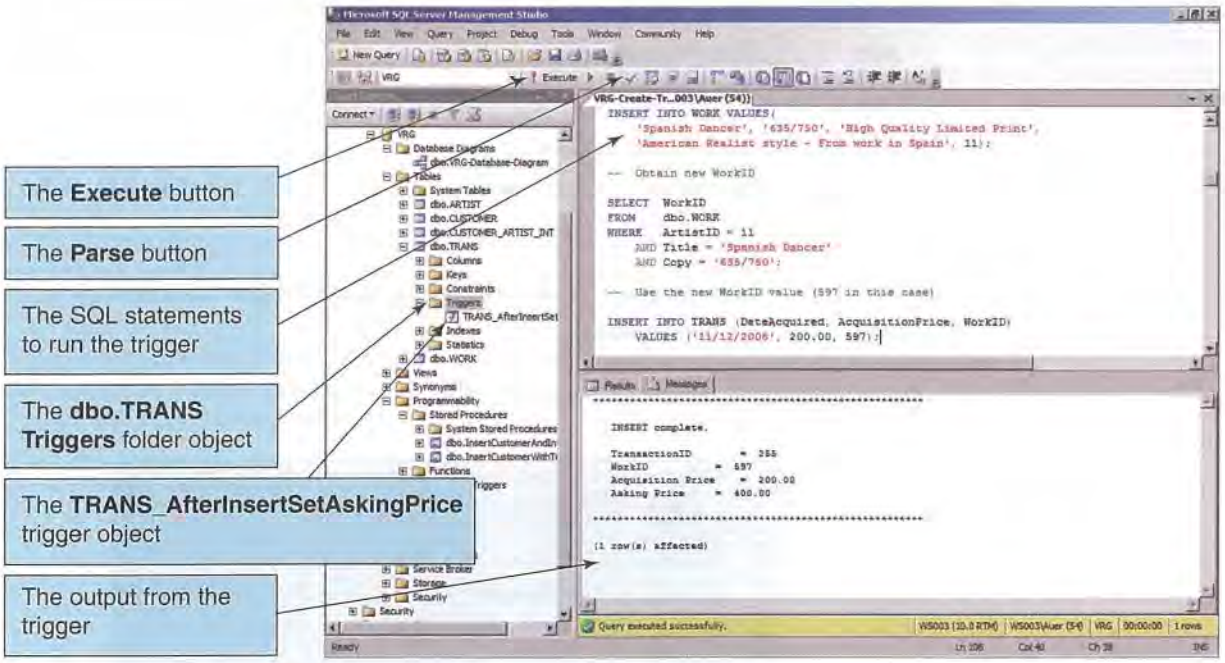


Figure 10-33

Results of the TRANS_AfterInsertSetAskingPrice Trigger

Figure 10-33 shows the results of the events triggered by the INSERT statement on TRANS. Note that the asking price for the new work (400.00) has been set to twice the acquisition cost (200.00), which is the correct value for a work that has not previously been in the gallery. This trigger provides useful functionality for the gallery. It saves the gallery personnel considerable manual work in implementing their pricing policy and likely improves the accuracy of the results as well.

> **BY THE WAY** At the beginning of the trigger code, you will see the command "SET NOCOUNT ON;" You may have noticed that when, for example, you execute INSERT statements SQL Server generates the message "(1 row(s) affected)" after each INSERT statement so that this message appears in the Microsoft SQL Server Management Studio Messages window. This is useful if you are actually watching these actions (it tells you that the intended action successfully completed), but it is meaningless when a trigger is firing based on an application action. Further, the time it takes to generate these messages can slow performance down in some cases. Therefore, we set NOCOUNT to ON so that these messages are not generated by trigger actions.

A Trigger for Enforcing a Data Constraint

The View Ridge Gallery needs to track problem-customer accounts; these are customers who have either not paid promptly or who have presented other problems to the gallery. When a customer who is on the problem list attempts to make a purchase at the gallery, the gallery wants the transaction to be rolled back and a message displayed. Note that this feature requires an intertable CHECK constraint between the TRANS table and the CUSTOMER table, which, as we discussed in Chapter 7, requires a trigger to implement.

To enforce this policy and the corresponding constraint, we need to add a column to the CUSTOMER table named isProblemAccount. This column will use the **SQL Server Bit data type**, which can have the values NULL, 0, and 1. Zero indicates a good account, whereas a 1 indicates a problem account. And it looks like our new customer Melinda Gliddens had trouble with her previous payment, so we set her isProblemAccount value to 1:

```
ALTER TABLE dbo.CUSTOMER
      ADD isProblemAccount    Bit NULL DEFAULT '0';
```

Amazon Ex. 1008.0424

```
UPDATE      dbo.CUSTOMER
   SET      isProblemAccount = 0;

UPDATE      dbo.CUSTOMER
   SET      isProblemAccount = 1
   WHERE    LastName = 'Gliddens'
      AND   FirstName = 'Melinda';

SELECT      CustomerID, LastName, FirstName, isProblemAccount
FROM        dbo.CUSTOMER;
```

The results of the SELECT statement are:

	CustomerID	LastName	FirstName	isProblemAccount
1	1000	Janes	Jeffrey	0
2	1001	Smith	David	0
3	1015	Twilight	Tiffany	0
4	1033	Smathers	Fred	0
5	1034	Frederic...	Mary Beth	0
6	1036	Warning	Selma	0
7	1037	Wu	Susan	0
8	1040	Gray	Donald	0
9	1041	Johnson	Lynda	0
10	1051	Wilkens	Chris	0
11	1052	Bench	Michael	0
12	1053	Gliddens	Melinda	1

Now we will create a trigger on TRANS named TRANS_CheckIsProblemAccount. With this trigger, when a customer makes a purchase the trigger determines whether the customer is flagged by the value of the isProblemAccount data in the CUSTOMER table. If so, the transaction is rolled back and a message is displayed. The trigger code in Figure 10-34 enforces this policy.

Note one interesting feature of the trigger code in Figure 10-34. As noted there, this trigger will fire for every update on TRANS, including updates fired by another trigger such as TRANS_AfterInsertSetAskingPrice. But in that trigger, no customer is involved. Therefore, before completing the rest of this trigger, we have to be sure that there is actually a customer participating in a transaction whose account status needs to be checked. This is done by the line:

```
IF (@CustomerID IS NULL) RETURN
```

Note that we only want to exit the TRANS_CheckIsProblemAccount trigger if there is no customer, not rollback the transaction that fired the trigger. When writing multiple triggers, remember that they may be run from other actions besides the one that you originally created them to handle.

OK, here comes Melissa to make another purchase—let's see what happens.

```
UPDATE      TRANS
SET         DateSold = '11/18/2008',
            SalesPrice = 475.00,
            CustomerID = 1056
WHERE       TransactionID = 229;
```

Amazon Ex. 1008.0425

```
CREATE TRIGGER TRANS_CheckIsProblemAccount
   ON dbo.TRANS
   FOR UPDATE

AS
BEGIN
   SET NOCOUNT ON;

   DECLARE   @TransactionID      AS Int,
             @CustomerID         AS Int,
             @isProblemAccount   AS Bit

   SELECT @TransactionID = TransactionID,
          @CustomerID   = CustomerID
   FROM   inserted;

   /* This trigger will fire for every update of TRANS.
    * This includes updates without a Customer participating,
    * such as an update of AskingPrice using the
    * TRANS_AfterInsertSetAskingPrice tigger.
    * Therefore, make sure there is a Customer particpating
    * in the Update of TRANS/
    */

   -- Check if Customer ID is NULL and if so RETURN.
   -- Do not ROLLBACK the transaction, just don't complete this trigger.

   IF (@CustomerID IS NULL) RETURN

   -- Valid CustomerID.
   -- Obtain value of @isProblemAcocunt.

   SELECT    @isProblemAccount = isProblemAccount
   FROM      dbo.CUSTOMER AS C
   WHERE     C.CustomerID = @CustomerID;

   IF (@isProblemAccount = 1)
      -- This is a problem account.
      -- Rollback the transaction and send message.
      BEGIN
         ROLLBACK TRANSACTION
         PRINT '*****************************************************'
         PRINT ''
         PRINT '   Transaction canceled.'
         PRINT ''
         PRINT '   CustomerID      = '+CONVERT(Char(6), @CustomerID)
         PRINT ''
         PRINT '   Refer customer to the manager immediately.'
         PRINT ''
         PRINT '*****************************************************'
         RETURN
```

Figure 10-34

The SQL Statements for the TRANS_CheckIsProblem Account Trigger

Amazon Ex. 1008.0426

```
    ELSE
            -- This is a good account
            -- Let the transaction stand.
            BEGIN
                PRINT '*************************************************'
                PRINT ''
                PRINT '   Transaction complete.'
                PRINT '   TransactionID    = '+CONVERT(Char(6), @TransactionID)
                PRINT '   CustomerID       = '+CONVERT(Char(6), @CustomerID)
                PRINT ''
                PRINT '   Thank the customer for their business.'
                PRINT ''
                PRINT '*************************************************'
            END
```

Figure 10-34

(Continued)

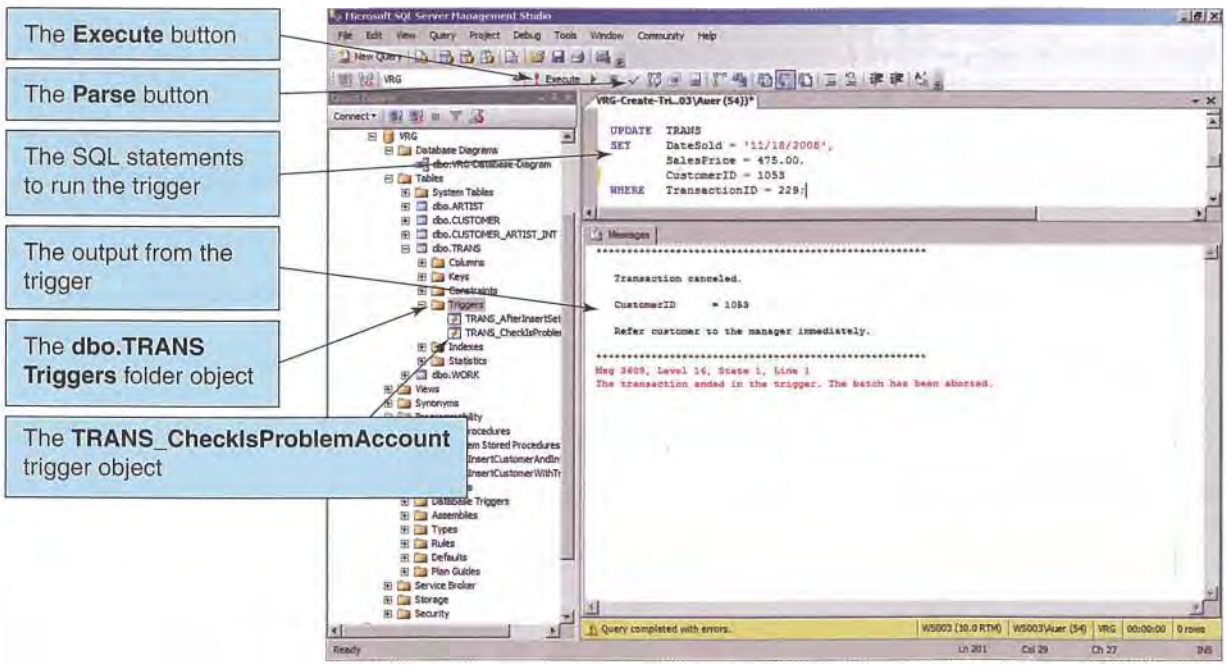
The resulting output is shown in Figure 10-35. Looks like Melinda is off to talk to the manager about her account!

BY THE WAY Using a table of valid or invalid values is more flexible and dynamic than placing such values in a CHECK constraint. For example, consider the CHECK constraint on Nationality values in the ARTIST table. If the gallery manager wants to expand the nationality of allowed artists, the manager will have to change the CHECK constraint using the ALTER TABLE statement. In reality, the gallery manager will have to hire a consultant to change this constraint.

A better approach is to place the allowed values of Nationality in a table, say, ALLOWED_NATIONALITY. Then, write a trigger like that shown in Figure 10-34 to enforce the constraint that new values of Nationality exist in ALLOWED_NATIONALITY. When the gallery owner wants to change the allowed artists, the manager would simply add or remove values in the ALLOWED_NATIONALITY table.

Figure 10-35

Results of the TRANS_CheckIsProblem Account



Amazon Ex. 1008.0427

A Trigger for Updating a View

In Chapter 7, we discussed the problem of updating views. One such problem concerns updating views created via joins; it is normally not possible for the DBMS to know how to update tables that underlie the join. However, sometimes application-specific knowledge can be used to determine how to interpret a request to update a joined view.

Consider the view CustomerInterestsView shown in Figure 10-25. It contains rows of CUSTOMER and ARTIST joined over their intersection table. CUSTOMER.LastName is given the alias CustomerLastName, CUSTOMER.FirstLastName is given the alias CustomerFirstName, and ARTIST.LastName is given the alias ArtistName. A request to change the last name of a customer in CustomerInterests can be interpreted as a request to change the last name of the underlying CUSTOMER table. Such a request, however, can be processed only if the value of CUSTOMER.LastName is unique. If not, the request cannot be processed.

The INSTEAD OF trigger shown in Figure 10-36 implements this logic. First, the new and old values of the CustomerLastName column in CustomerInterestsView are obtained. Then, a correlated subquery is used to determine whether the old value of CUSTOMER.LastName is unique. If so, the name can be changed, but otherwise, no update can be made.

This trigger needs to be tested against cases in which the last name is unique and cases in which the last name is not unique. Figure 10-37 shows the case in which the customer name was unique. View Ridge Gallery's two newest customers, Michael Bench and Melissa Gliddens, just got married after meeting at a View Ridge Gallery opening, and Melissa wants to change her last name. To do this, we use the SQL statement:

```
UPDATE    dbo.CustomerInterestsView
SET       CustomerLastName = 'Bench'
WHERE     CustomerLastName = 'Gliddens';
```

Note that the UPDATE command was issued against the view. As indicated in the Messages pane, Melissa is now Melissa Bench.

A Trigger for Enforcing a Required Child Constraint

The VRG database design includes an M-M relationship between WORK and TRANS. Every WORK must have a TRANS to store the price of the work and the date the work was acquired, and every TRANS must relate to a WORK parent. Figure 10-38 shows the tasks that must be accomplished to enforce this constraint; it is based on the boilerplate shown in Figure 6-28(b).

Because the CREATE TABLE statement for TRANS in Figure 10-13 defines TRANS.WorkID as NOT NULL and defines the FOREIGN KEY constraint without cascading deletions, the DBMS will ensure that every TRANS has a WORK parent. So, we need not be concerned with enforcing the insert on TRANS or the deletion on WORK. As stated in Figure 10-38, the DBMS will do that for us. Also, we need not be concerned with updates to WORK.WorkID because it is a surrogate key.

Three constraints remain that must be enforced by triggers: (1) ensuring that a TRANS row is created when a new WORK is created; (2) ensuring that TRANS.WorkID never changes; and (3) ensuring that the last TRANS child for a WORK is never deleted.

We can enforce the second constraint by writing a trigger on the update of TRANS that checks for a change in WorkID. If there is such a change, the trigger can roll back the change.

With regards to the third constraint, View Ridge Gallery has a business policy that no TRANS data ever be deleted. Thus, we need not only to disallow the deletion of the last child, we need to disallow the deletion of any child. We can do this by writing a trigger on the deletion of TRANS that rolls back any attempted deletion. (If the gallery allowed TRANS deletions, we could enforce the deletion constraint using views, as shown in Chapter 7, Figures 7-29 and 7-30.) The triggers for enforcing the second and third constraints are simple, and we leave them as exercises 11.28 and 11.29.

However, the first constraint is a problem. Now, we could write a trigger on the WORK INSERT to create a default TRANS row, but this trigger will be called before the application has

Amazon Ex. 1008.0428

```
CREATE TRIGGER CIV_ChangeCustomerLastName
   ON dbo.CustomerInterestsView
   INSTEAD OF UPDATE

AS
BEGIN
   SET NOCOUNT ON;

   DECLARE   @NewCustomerLastName    AS Char(25),
             @OldCustomerLastName    AS Char(25)

   -- Get values of new and old names.
   SELECT    @NewCustomerLastName = CustomerLastName
   FROM      inserted;

   SELECT    @OldCustomerLastName = CustomerLastName
   FROM      deleted;

   -- Count number of synonyms is CUSTOMER.
   SELECT    *
   FROM      dbo.CUSTOMER AS C1
   WHERE     C1.LastName = @OldCustomerLastName
      AND    EXISTS
             (SELECT   *
              FROM     dbo.CUSTOMER AS C2
              WHERE    C1.LastName = C2.LastName
                 AND   C1.CustomerID <> C2.CustomerID);

   IF (@@rowCount = 0)
      -- The Customer last name is unique.
      -- Update the Customer record.
      BEGIN
         UPDATE    dbo.CUSTOMER
         SET       LastName = @NewCustomerLastName
         WHERE     LastName = @OldCustomerLastName;

         -- Print update message.
         PRINT '*******************************************************'
         PRINT ''
         PRINT '   The Customer last name has been changed.'
         PRINT ''
         PRINT '   Former Customer Last Name    = '+@OldCustomerLastName
         PRINT ''
         PRINT '   Upddated Customer Last Name  = '+@NewCustomerLastName
         PRINT ''
         PRINT '*******************************************************'
      END
```

Figure 10-36

The SQL Statements for the CIV_ChangeCustomer LastName Trigger

Amazon Ex. 1008.0429

```
    ELSE
        -- The Customer name is not unique.
        -- Rollback the transaction and send message.
        BEGIN
            ROLLBACK
            PRINT '*****************************************************'
            PRINT ''
            PRINT '    Transaction canceled.'
            PRINT ''
            PRINT '    Customer Last Name  = '+@NewCustomerLastName
            PRINT ''
            PRINT '    The customer last name is not unique.'
            PRINT ''
            PRINT '*****************************************************'
        END
    END
```

Figure 10-36

(Continued)

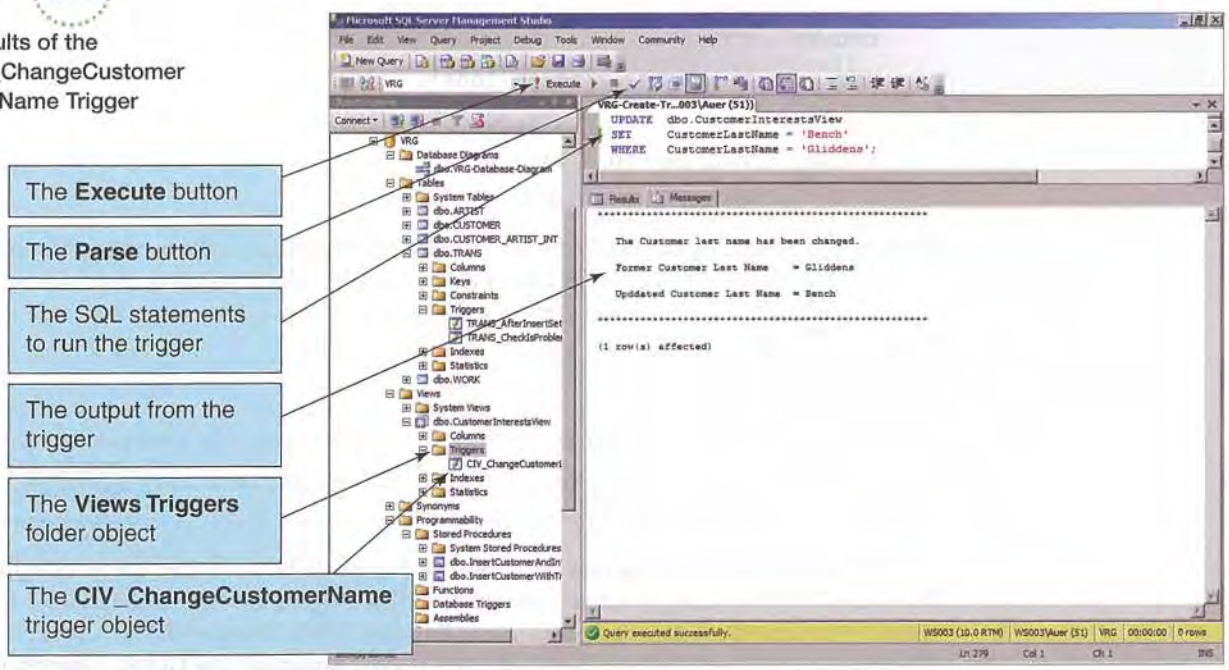
a chance to create the TRANS row itself. The trigger would create a TRANS row and then the application may create a second one. To guard against the duplicate, we could then write a trigger on TRANS to remove the row the WORK trigger created in those cases when the application creates its own trigger. However, this solution is awkward, at best.

A better design is to require the applications to create the WORK and TRANS combination via a view. For example, we can create a view named WorkAndTransView (we will add the code into our VRG-Create-Views.sql script and execute it from there):

```
CREATE VIEW WorkAndTransView AS
        SELECT      Title, Copy, Medium, [Description], ArtistID,
                    DateAcquired, AcquisitionPrice
        FROM        WORK AS W JOIN TRANS AS T
            ON      W.WorkID = T.WorkID;
```

Figure 10-37

Results of the CIV_ChangeCustomer LastName Trigger



Amazon Ex. 1008.0430

WORK Is Required Parent TRANS Is Required Child	Action on WORK (Parent)	Action on TRANS (Child)
Insert	Create a TRANS row.	New TRANS must have a valid WorkID (enforced by DBMS).
Modify key or foreign key	Prohibit—WORK uses a surrogate key.	Prohibit—TRANS uses a surrogate key, and TRANS cannot change to a different WORK.
Delete	Prohibit—Cannot delete a WORK with and TRANS children (enforced by DBMS by lack of CASCADE DELETE).	Cannot delete the last child. [Actually, data related to a transaction is never deleted (business rule)].

Figure 10-38

Actions to Enforce Minimum Cardinality for the WORK-to-TRANS Relationship

The DBMS will not be able to process an insert on this view. We can, however, define an INSTEAD OF trigger to process the insert. Our trigger, named WATV_InsertTransactionWithWork, will create both a new row in WORK and the new required child in TRANS. The code for this trigger is shown in Figure 10-39. Note that with this solution applications must not be allowed to insert WORK rows directly. They must always insert them via the view WorkAndTransView.

To test our trigger, we will add a new work to the VRG database. Melissa, now Mrs. Michael Bench, has worked out her account problems with the View Ridge Gallery and completed her purchase of the print of Horiuchi's *Color Floating in Time*.

```
UPDATE    dbo.CUSTOMER
SET       isProblemAccount = 0
WHERE     LastName       = 'Bench'
  AND     FirstName      = 'Melinda';

UPDATE    TRANS
SET       DateSold = '11/18/2008',
          SalesPrice = 475.00,
          CustomerID = 1053
          WHERE    TransactionID = 229;
```

Note that the preceding UPDATE statement fired the TRAN_CheckIsProblemAccount trigger, but since Melinda is now a good customer again, the transaction was accepted. Now we will restock a copy of *Color Floating in Time* into the gallery.

```
INSERT INTO WorkAndTransView
    VALUES(
    'Color Floating in Time', '493/750', 'High Quality Limited Print',
    'Northwest School Abstract Expressionist style', 18,
    '02/05/2009', 250.00);
```

The results of the transaction are shown in Figure 10-39. Note that the WATV_InsertTransactionWithWork trigger actually sets off two other triggers. First it fires the INSERT trigger TRANS_AfterInsertSetAskingPrice, which then fires the UPDATE trigger TRANS_CheckProblemAccount trigger. Because no customer is involved in this transaction, the TRANS_CheckProblem

Amazon Ex. 1008.0431

```
CREATE TRIGGER WATV_InsertTransactionWithWork
   ON dbo.WorkAndTransView
   INSTEAD OF INSERT

AS
BEGIN
   SET NOCOUNT ON;

   DECLARE   @TransactionID      AS Int,
             @WorkID             AS Int,
             @Title              AS Char(35),
             @Copy               AS Char(12),
             @Medium             AS Char(35),
             @Description        AS Varchar(1000),
             @ArtistID           AS Int,
             @DateAcquired       AS DateTime,
             @AcquisitionPrice   AS Numeric(8,2),
             @AskingPrice        AS Numeric(8,2)

   -- Get available values from Insert on the view.
   SELECT    @Title = Title, @Copy = Copy, @Medium = Medium,
             @Description = [Description],
             @ArtistID = ArtistID, @DateAcquired =DateAcquired,
             @AcquisitionPrice = AcquisitionPrice
   FROM      inserted;

   -- Insert new row into WORK.
   INSERT INTO WORK VALUES(
             @Title, @Copy, @Medium, @Description, @ArtistID);

   -- Get new WorkID surrogate key value using @@Identity funcion.
   SET @WorkID = @@Identity;

   -- Insert new row into TRANS.
   -- Note that INSERT will trigger TRANS_AfterInsertSetAskingPrice.
   INSERT INTO TRANS (DateAcquired, AcquisitionPrice, WorkID)
             VALUES(
             @DateAcquired, @AcquisitionPrice, @WorkID);

   -- Get new TransactionID surrogate key value.
   SET @TransactionID = @@Identity;

   -- Get new AskingPrice set by TRANS_AfterInsertSetAskingPrice.
   SELECT    @AskingPrice = AskingPrice
   FROM      TRANS
   WHERE     TransactionID = @TransactionID;

   -- Print results message.
   PRINT '*****************************************************'
   PRINT ''
   PRINT '   The new work has been inserted into WORK and TRANS.'
   PRINT ''
   PRINT '   TransactionID     = '+CONVERT(Char(6), @TransactionID)
   PRINT '   WorkID            = '+CONVERT(Char(6), @WorkID)
   PRINT '   ArtistID          = '+CONVERT(Char(6), @ArtistID)
   PRINT '   Title             = '+@Title
   PRINT '   Copy              = '+@Copy
   PRINT '   Medium            = '+@Medium
   PRINT '   Description       = '+@Description
   PRINT '   DateAcquired      = '+CONVERT(Char(12), @DateAcquired)
   PRINT '   Acquisition Price = '+CONVERT(Char(12), @AcquisitionPrice)
   PRINT '   Asking Price      = '+CONVERT(Char(12), @AskingPrice)
   PRINT '*****************************************************'
   END
```

Figure 10-39

The SQL Statements for the WATV_Insert TransactionWithWork Trigger

Amazon Ex. 1008.0432

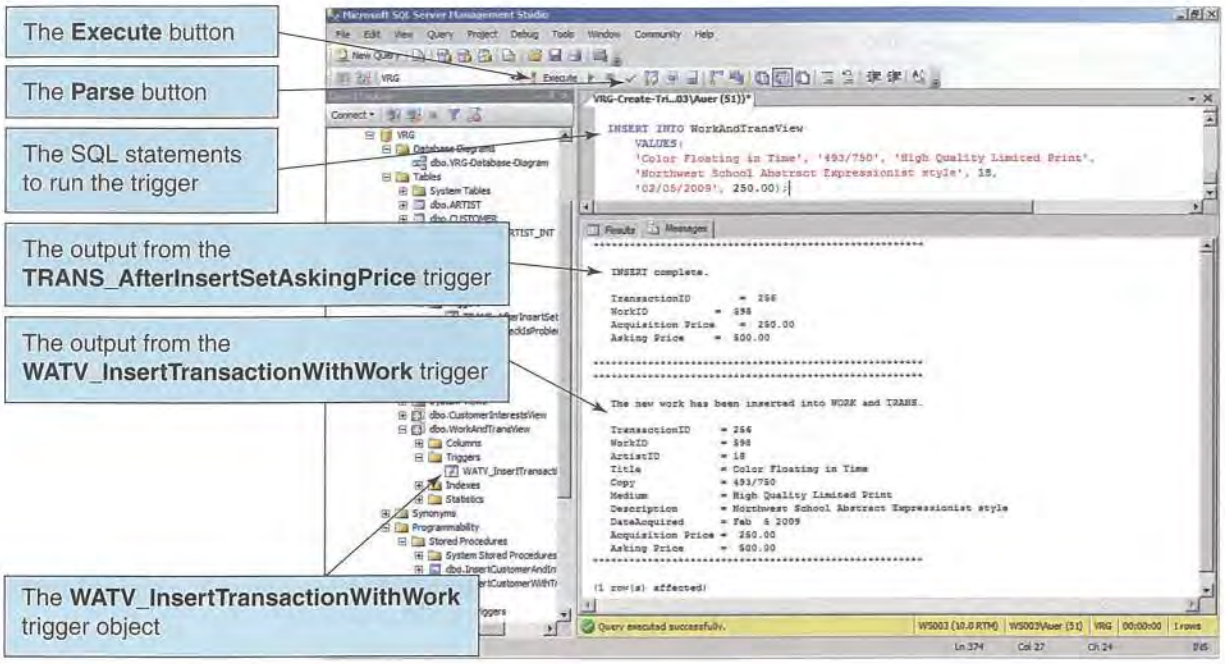


Figure 10-40
Results of the WATV_InsertTransaction WithWork Trigger

Account returns control without completing the trigger (see the previous discussion of the code for this trigger). However, the TRANS_AfterInsertSetAskingPrice trigger does run and sets the new asking price for the new work. Therefore, the results of two triggers show up in the output in Figure 10-40 and the new work does now have an asking price.

BY THE WAY If we look at the SQL code that has actually been stored for these triggers, we will again find that SQL Server has added some lines before the code we wrote:

```
USE [VRG]
GO
/****** Object: Trigger [ {StoredProcedureName} ]
    Script Date: {Date and Time created or altered} ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

This is essentially the same code used for stored procedures. However, we should also note that there is an important use of the GO command when running scripts. Triggers are run only *once* for a set of operations, such as typically found in a script, not for each individual command.

For example, suppose that in our last trigger example we had decided to use an SQL script to input the data for three copies of *Color Floating in Time* at once as follows (note the change in the value of Copy):

```
INSERT INTO WorkAndTransView
    VALUES(
    'Color Floating in Time', '493/750', 'High Quality Limited Print',
    'Northwest School Abstract Expressionist style', 18,
    '02/05/2009', 250.00);
```

Amazon Ex. 1008.0433

```
INSERT INTO WorkAndTransView
    VALUES(
    'Color Floating in Time', '494/750', 'High Quality Limited Print',
    'Northwest School Abstract Expressionist style', 18,
    '02/05/2009', 250.00);
INSERT INTO WorkAndTransView
    VALUES(
    'Color Floating in Time', '495/750', 'High Quality Limited Print',
    'Northwest School Abstract Expressionist style', 18,
    '02/05/2009', 250.00);
```

In this case, the WATV_InsertTransactionWithWork trigger only fires *once*! But this is not what we want—we want it to fire once per INSERT. This is where the GO command comes in handy, and we will use the following code:

```
INSERT INTO WorkAndTransView
    VALUES(
    'Color Floating in Time', '493/750', 'High Quality Limited Print',
    'Northwest School Abstract Expressionist style', 18,
    '02/05/2009', 250.00);
GO
    INSERT INTO WorkAndTransView
    VALUES(
    'Color Floating in Time', '494/750', 'High Quality Limited Print',
    'Northwest School Abstract Expressionist style', 18,
    '02/05/2009', 250.00);
GO
    INSERT INTO WorkAndTransView
    VALUES(
    'Color Floating in Time', '495/750', 'High Quality Limited Print',
    'Northwest School Abstract Expressionist style', 18,
    '02/05/2009', 250.00);
GO
```

Now the each statement is considered its own "batch of commands," and the trigger fires three times, as we intended.

Concurrency Control

SQL Server 2008 provides a comprehensive set of capabilities to control concurrent processing. Many choices and options are available, and the resulting behavior is determined by the interaction of three factors: the transaction isolation level, the cursor concurrency setting, and locking hints provided in the SELECT clause. Locking behavior also depends on whether the cursor is processed as part of a transaction, whether the SELECT statement is part of a cursor, and whether INSERT, UPDATE, or DELETE commands occur inside of transactions or independently.

Figure 10-41 summarizes the concurrency control options. In this section, we will just describe the basics. For more information, see the article "Set Transaction Isolation Level" at *http://msdn.microsoft.com/en-us/library/ms173763(SQL.90).aspx*.

With SQL Server, developers do not place explicit locks. Instead, developers declare the concurrency control behavior they want, and SQL Server determines where to place the locks.

Amazon Ex. 1008.0434

Type	Scope	Options
Tranaction isolation level	Connection—all transactions	READ UNCOMMITTED READ COMMITTED REPEATABLE READ SNAPSHOT SERIALIZABLE
Cursor concurrency	CURSOR statements	READ_ONLY SCROLL_LOCKS OPTIMSTIC
Locking hints	SELECT statements	READCOMMITTED READUNCOMMITTED REPEATABLERAD SERIALIZABLE NOLOCK HOLDLOCK And many more...

Figure 10-41
Concurrency Options for SQL Server 2008

Locks are applied on rows, pages, keys, indexes, tables, and even on the entire database. SQL Server determines what level of lock to use and may promote or demote a lock level while processing. It also determines when to place the lock and when to release it, depending on the declarations made by the developer.

Transaction Isolation Level

The broadest level of concurrency settings is the **transaction isolation level**. As shown in Figure 10-41, there are five transaction isolation level options, listed in ascending level of restriction. Four of these options are the four you studied in Chapter 9, and they are the SQL-92 standard levels. A new level, SNAPSHOT, is unique to SQL Server. With the **SNAPSHOT transaction isolation level**, the data read by an SQL statement will be the same as the data that existed and were committed at the start of the transaction. SNAPSHOT does not request locks during reads (except during database recovery) and does not prevent other transactions from writes. With SQL Server, READ COMMITED is the default isolation level, although it is possible to make dirty reads by setting the isolation level to READ UNCOMMITTED. However, the exact behavior of READ UNCOMMITTED now depends on the setting of the READ_COMMITTED_SNAPSHOT option. See the article "Set Transaction Isolation Level" at *http://msdn.microsoft.com/en-us/library/ms173763(SQL.90).aspx* for a complete discussion.

An SQL to set the isolation level can be issued anyplace Transact-SQL is allowed, prior to any other database activity. An example Transact-SQL statement to set the isolation level of, say, REPEATABLE READ is:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Cursor Concurrency

The second way in which the developer can declare locking characteristics is with cursor concurrency. Possibilities are read only, optimistic, and pessimistic, here called **SCROLL_LOCK**. As described in Chapter 9, with optimistic locking, no lock is obtained until the user updates the data. At that point, if the data have been changed since they were read, the update is refused. Of course, the application program must specify what to do when such a refusal occurs. See the SQL Server 2008 documentation on "Declare Cursor" at *http://msdn.microsoft.com/en-us/library/ms180169(SQL.90).aspx* for a complete discussion.

SCROLL_LOCK is a version of pessimistic locking. With it, an update lock is placed on a row when the row is read. If the cursor is opened within a transaction, the lock is held until the transaction commits or rolls back. If the cursor is outside of a transaction, the lock is

Amazon Ex. 1008.0435

dropped when the next row is read. Recall from Chapter 9 that an update lock blocks another update lock, but it does not block a shared lock. Thus, other connections can read the row with shared locks.

As described in Chapter 9, the default cursor concurrency setting depends on the cursor type. It is read only for static and forward only cursors, and it is optimistic for dynamic and key-set cursors.

Cursor concurrency is set with the DECLARE CURSOR statement. An example to declare a dynamic SCROLL_LOCK cursor on all rows of the TRANS table is as follows:

```
DECLARE MyCursor CURSOR DYNAMIC SCROLL_LOCKS
FOR
    SELECT    *
    FROM      dbo.TRANS;
```

Locking Hints

Locking behavior can be further modified by providing locking hints in the WITH parameter of the FROM clause in SELECT statements. Figure 11-41 lists several of the locking hints available with SQL Server. The first four hints override the transaction isolation level; the next two influence the type of lock issued. For a full list of locking hints, see the documentation at *http://msdn.microsoft.com/en-us/library/ms187373.aspx*.

Consider the following statements:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
DECLARE MyCursor CURSOR DYNAMIC SCROLL_LOCKS
FOR
    SELECT    *
    FROM      dbo.TRANS WITH READUNCOMMITTED NOLOCK;
```

Without the locking hints, the cursor MyCursor would have REPEATABLE READ isolation and would issue update locks on all rows read. The locks would be held until the transaction committed. With the locking hints, the isolation level for this cursor becomes READ UNCOMMITTED. Furthermore, the specification of NOLOCK changes this cursor from DYNAMIC to READ_ONLY.

Consider another example:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
DECLARE MyCursor CURSOR DYNAMIC SCROLL_LOCKS
FOR
    SELECT    *
    FROM      dbo.TRANS WITH HOLDLOCK;
```

Here, the locking hint will cause SQL Server to hold update locks on all rows read until the transaction commits. The effect is to change the transaction isolation level for this cursor from REPEATABLE READ to SERIALIZABLE.

In general, the beginner is advised not to provide locking hints. Rather, until you have become an expert, set the isolation level and cursor concurrency to appropriate values for your transactions and cursors and leave it at that. In fact, the SQL Server 2008 documentation specifically suggests relying on the SQL Server query optimizer and only using locking hints when absolutely necessary.

Amazon Ex. 1008.0436

SQL Server 2008 Security

We discussed security in general terms in Chapter 9. Here, we will summarize how those general ideas pertain to SQL Server security. Starting with SQL Server 2005 and continuing with SQL Server 2008, Microsoft introduced a more complex version of the model of DBMS security shown in Figure 9-14. The SQL Server 2008 model is shown in Figure 10-42.

As usual, security consists of authentication and authorization. As shown in Figure 10-1, we have to log into SQL Server itself first, and we log in using one of the server logins illustrated in Figure 10-14. This login will be a member of one or more server roles, and each of these roles has specific permission at the server level.

Figure 10-42
SQL Server 2008 Security Model

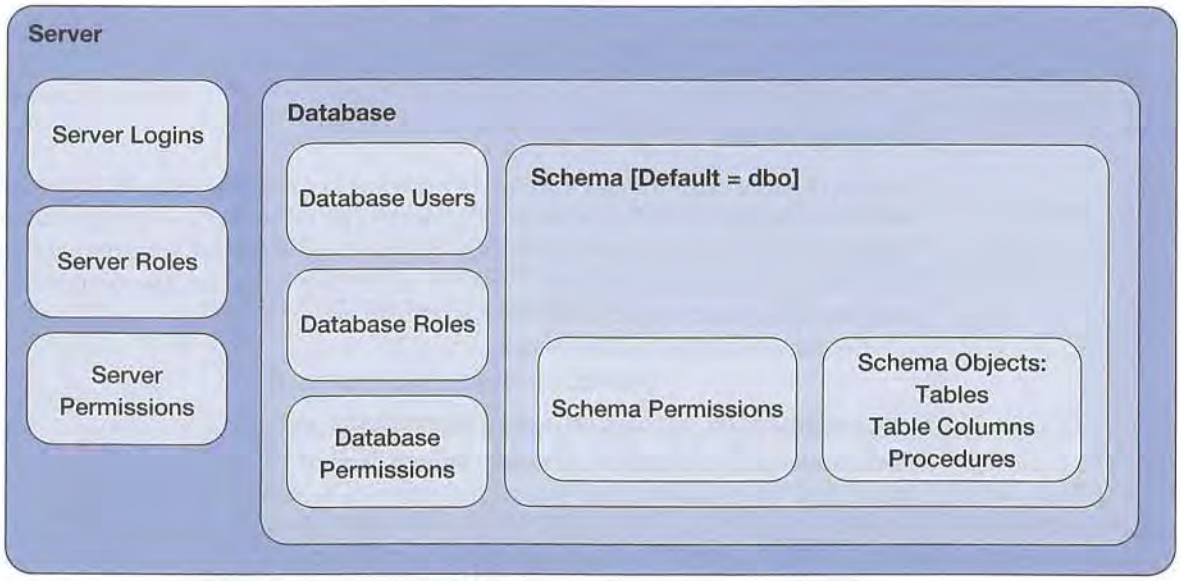
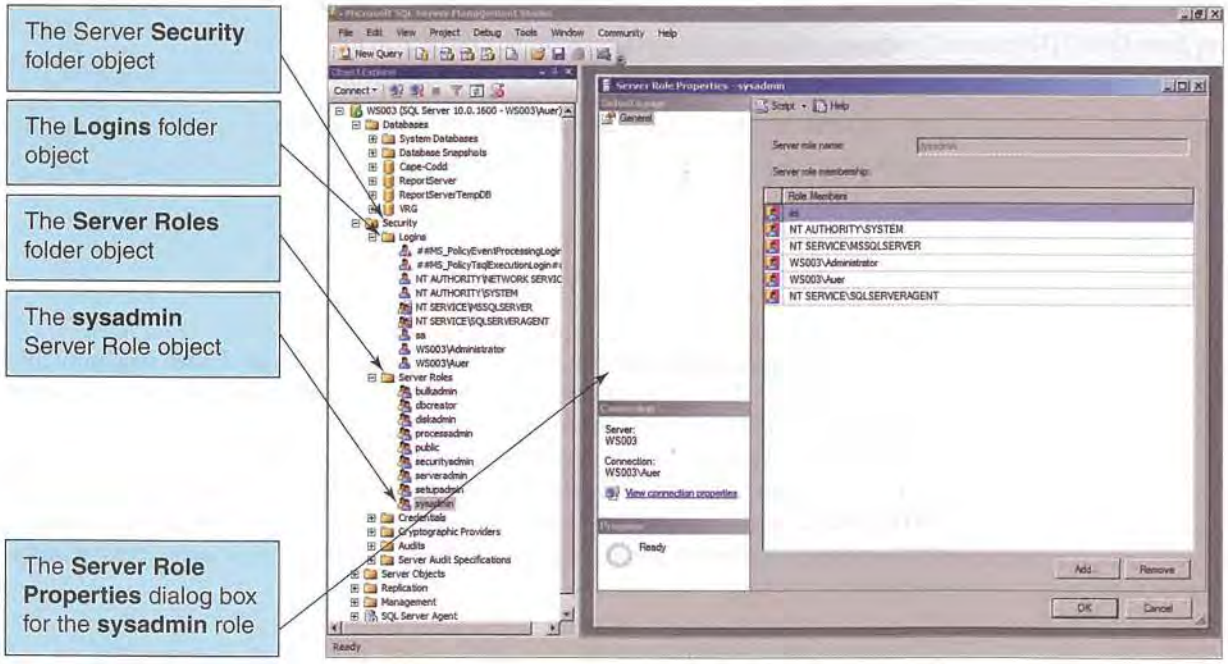


Figure 10-43
Server Logins and Server Roles



Amazon Ex. 1008.0437

We are currently logged into SQL Server 2008 (on a computer running the Windows Server 2008 operating system and named WS003) as the login WS003\Auer (see Figure 10-1), which means that we logged in based on the local computer user account *Auer*, and, as shown in Figure 10-43, that login is assigned to the sysadmin role. This means that the user auer has all the privileges of an SQL Server 2008 systems administrator (which is everything!). SQL Server 2008 logins can either be based on Windows operating system security (recommended by Microsoft), SQL-Server-specific logins (see the *sa* login in Figure 10-43—*sa* is the SQL Server system administrator account and if you use it be sure to set a secure password for it), or both (which is how our SQL Server 2008 instance is set up).

To see the server permissions that the login *Auer* has, we look at the Permissions page of the Server Properties dialog box (right-click the SQL Server object at the top of the Object Explorer, and then click the Properties command). Figure 10-44 shows the effective properties for the login *Auer*.

This gives us an overview of the SQL Server authorization model. **SQL Server principals** (either **server principals**, such as logins and server roles, or **database principals**, such users and database roles) are granted permissions to **SQL Server securable objects** (databases, schemas, tables, procedures, etc.). The new concept here is the schema. An **SQL Server schema** is a named container for SQL Server. In other words, SQL objects such as tables and stored procedures are now held in a container called a schema. A schema can be owned by any SQL server principal. The main advantage is that schema ownership can be transferred, and thus a database principal can be deleted without the loss of the objects owned by that principal. For more information, see "Owner–Schema Separation" at *http://msdn.microsoft.com/en-us/library/ms190387.aspx*. The default schema, the one an object will automatically be assigned to, is the **dbo schema**. In earlier versions of SQL Server, dbo (database owner) was only a *user*, now dbo is a also a *schema name*.

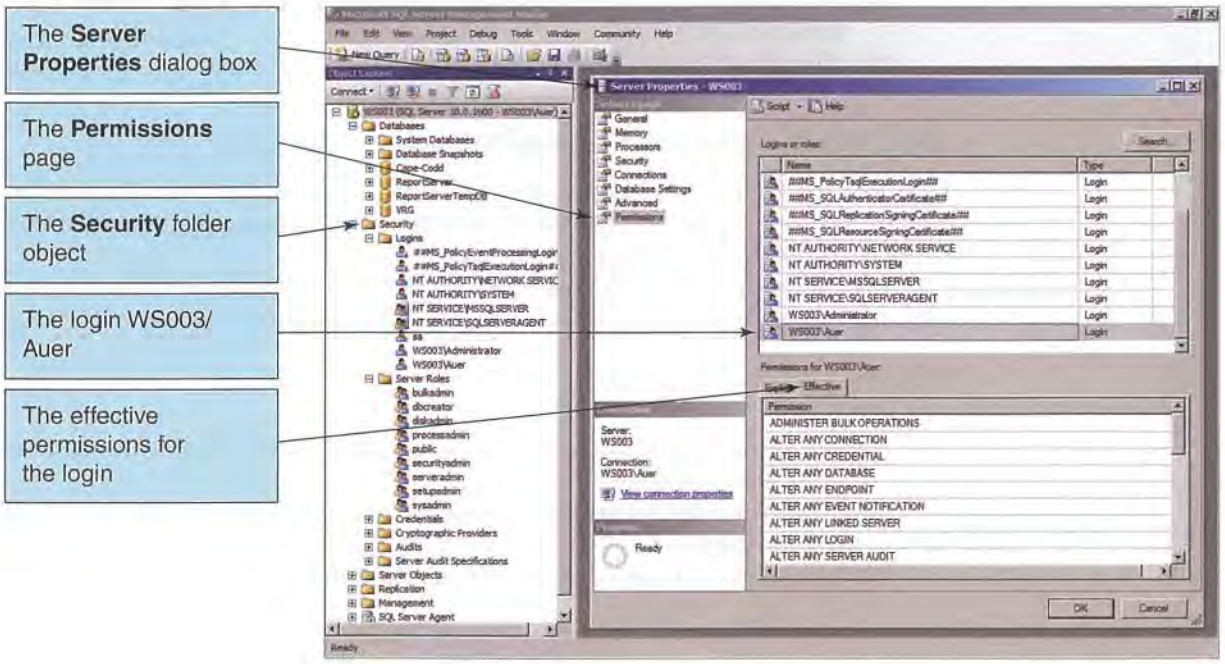
As an example, let's create a login for use by the View Ridge Gallery.

Creating a New Login

1. Expand the **Security** folder object in the Object Explorer so that the Logins folder object is displayed.
2. Right-click the **Logins** folder object to display the shortcut menu.

Figure 10-44

Effective Server Permissions for the Login *WS003/Auer*



Amazon Ex. 1008.0438

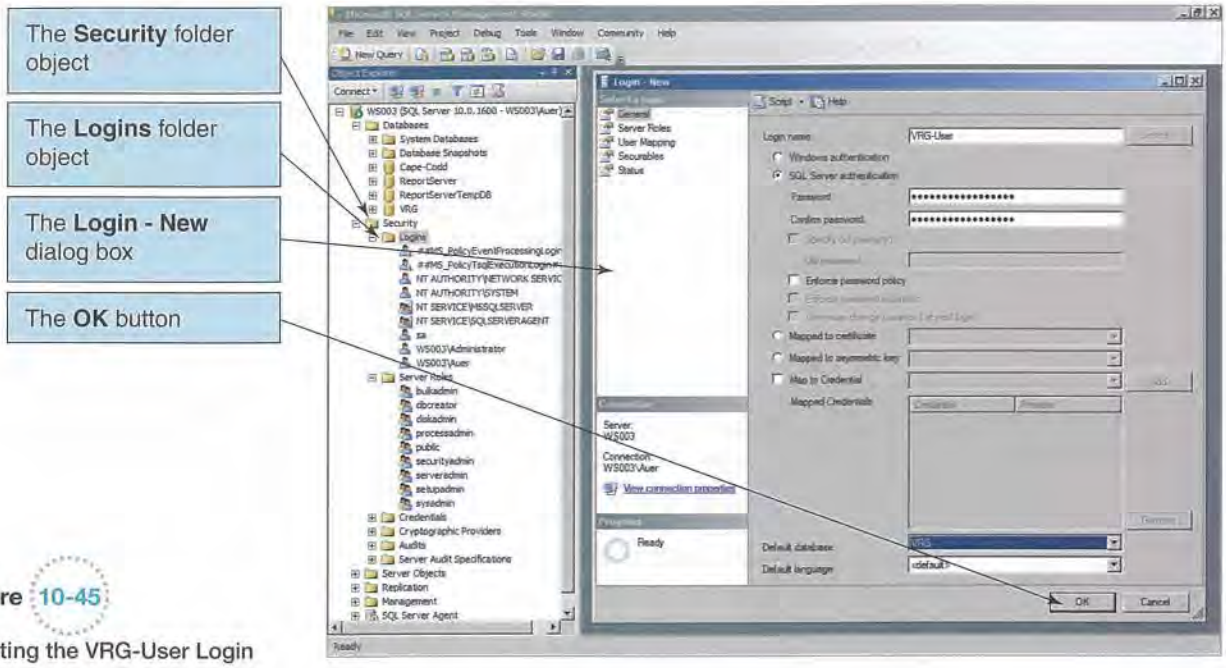


Figure 10-45
Creating the VRG-User Login

3. In the shortcut menu, click the **New Login** command. The Login – New dialog box is displayed.
4. The completed new login data is shown in Figure 10-45. Use that figure for reference in the following steps.
5. In the Login name text box, type in the login name **VRG-User**.
6. Click the **SQL Server authentication** radio button to enable the use of SQL Server authentication mode.
7. In the Password text box, type the password **VRG-User+password**.
8. In the Confirm password text box, type the password **VRG-User+password**.
9. Uncheck the **Enforce password policy** check box to disable enforcing password policy, disable enforcing password expiration, and disable forcing a password change. **NOTE:** This step is to simplify the use of this login in our test environment only. Do *not* do this in a production environment.
10. Select **VRG** as the Default database. The Login – New dialog box now appears, as shown in Figure 10-45.
11. Click the **OK** button. The new login is created.

SQL Server Database Security Settings

Now we need to move to the database level of security. Here we create specific database users and assign database roles (and associated permissions) to those users. SQL Server 2008 database roles and permissions are shown in Figure 10-46.

To illustrate the use of a database specific user, we will create a database user based on the VRG-User login we have created for use by the View Ridge Gallery.

Creating a New Database User

1. Expand the **VRG** database object in the Object Explorer so that the database Security folder object is displayed.
2. Expand the **VRG Security** folder object in the Object Explorer so that the Users folder object is displayed.
3. Right-click the **Users** folder object to display a shortcut menu.

Amazon Ex. 1008.0439

Fixed Database Role	Database-Specific Permissions	DBMS Server Permissions
db_accessadmin	Permissions granted: ALTER ANY USER, CREATE SCHEMA Permissions granted with GRANT option: CONNECT	Permissions granted: VIEW ANY DATABASE
db_backupoperator	Permissions granted: BACKUP DATABASE, BACKUP LOG, CHECKPOINT	Permissions granted: VIEW ANY DATABASE
db_datareader	Permissions granted: SELECT	Permissions granted: VIEW ANY DATABASE
db_datawriter	Permissions granted: DELETE. INSERT, UPDATE	Permissions granted: VIEW ANY DATABASE
db_ddladmin	Permissions granted: See SQL Server 2005 documentation	Permissions granted: VIEW ANY DATABASE
db_denydatareader	Permissions denied: SELECT	Permissions granted: VIEW ANY DATABASE
db_denydatawriter	Permissions denied: DELETE, INSERT, UPDATE	Permissions granted: VIEW ANY DATABASE
db_owner	Permissions granted with GRANT option: CONTROL	Permissions granted: VIEW ANY DATABASE
db_securityadmin	Permissions granted: ALTER ANY APPLICATION ROLE, ALTER ANY ROLE, CREATE SCHEMA, VIEW DEFINITION	Permissions granted: VIEW ANY DATABASE

Note: For the definitions of each of the SQL Server permissions shown in the table, consult the SQL Server documentation.

Figure 10-46

SQL Server 2008 Fixed Database Roles

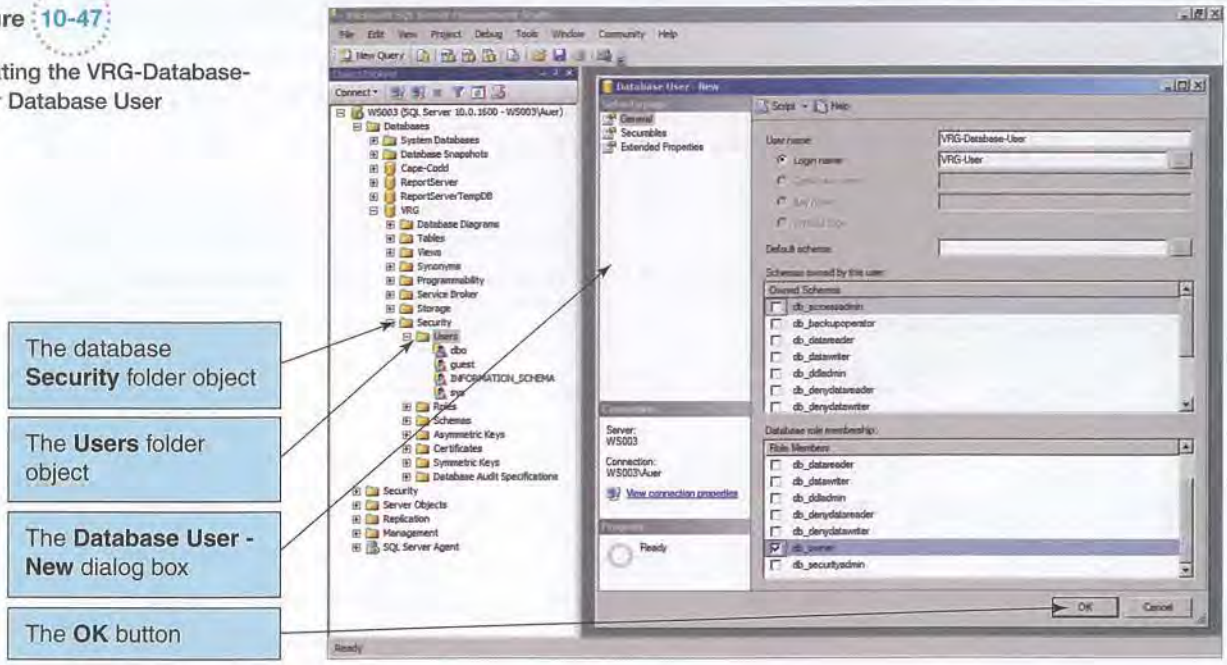
4. In the shortcut menu, click the **New User** command. The Database User – New dialog box is displayed.
5. The completed new login data is shown in Figure 10-47. Use that figure for reference in the following steps.
6. In the User name text box, type in the user name **VRG-Database-User**.
7. In the Login name text box, type in the login name **VRG-User**. **NOTE:** You can also browse to the correct login name using the Browse button shown in Figure 10-47.
8. In the Database role membership list, check **db_owner**.
9. The Database User – New dialog box now appears, as shown in Figure 10-47.
10. Click the **OK** button. The new login is created.

Now we have completed creating the needed logins and database users for the VRG database. We will use these in other chapters when we develop applications based on this database.

Amazon Ex. 1008.0440

Figure 10-47

Creating the VRG-Database-User Database User



SQL Server 2008 Backup and Recovery

When you create an SQL Server 2008 database, both data and log files are created. As explained in Chapter 9, these files should be backed up periodically. When backups are available, it is possible to recover a failed database by restoring it from a prior database save and applying changes in the log.

To recover a database with SQL Server, the database is restored from a prior database backup and log after images are applied to the restored database. When the end of the log is reached, changes from any transaction that failed to commit are then rolled back.

It is also possible to process the log to a particular point in time or to a transaction mark. For example, the following statement causes a mark labeled NewCust to be placed into the log every time this transaction is run:

```
BEGIN TRANSACTION NewCust WITH MARK;
```

If this is done, the log can be restored to a point either just before or just after the first NewCust mark or the first NewCust mark after a particular point in time. The restored log can then be used to restore the database. Such marks consume log space, however, so they should not be used without good reason.

Backing Up a Database

SQL Server 2008 supports several types of backup. We can, for example, choose to do a full backup, a differential backup, or a transaction log backup. A **full backup** backs up the complete database and the transaction logs. A **differential backup** backs up up only the changes since the last full backup, which is useful if a full backup takes a long time. This allows us to make periodic full backups (for example, once a week) and still capture daily changes to the database using differential backups. A **transaction log backup** backs up only the current transaction log. This is useful in keeping a series of transaction log files backed up more currently than even our differential backups. For example, we may do a transaction log backup every 6 hours.

Amazon Ex. 1008.0441

Here, we will make a complete backup of the VRG database. Other backup options may require the use of backup device objects and management policies. A full discussion of these topics is beyond the scope of this book. For more information, see the Microsoft SQL Server 2008 Books Online at *http://msdn.microsoft.com/en-us/library/ms130214.aspx*.

Backing Up the VRG Database

1. Right-click the **VRG** database object in the Object Explorer to display a shortcut menu.
2. In the shortcut menu, click the **Tasks** command, then click the **Backup** command. The Back Up Database - VRG dialog box is displayed, as shown in Figure 10-48.
3. SQL Server 2008 has already set up all the backup options correctly for a simple full backup of the database.
4. Click the **OK** button. The database backup is created.

We could use the same backup task we just ran to make differential backups if we wanted—we would simply choose Differential instead of Full as the backup type.

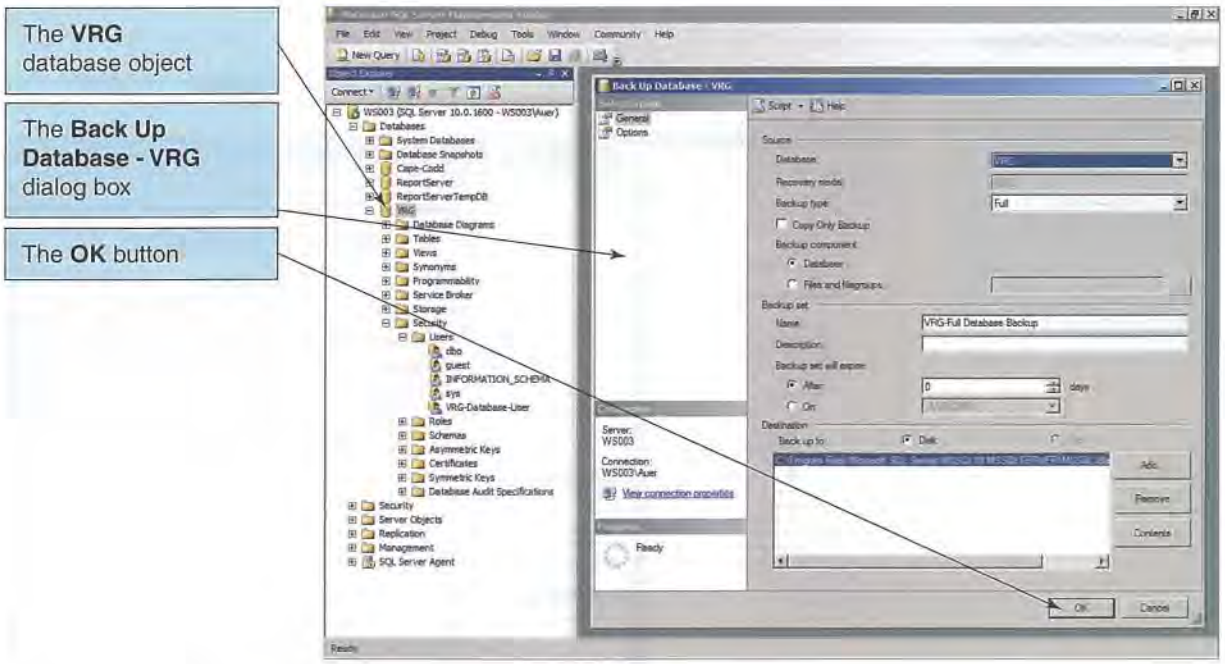
In a more production-oriented backup system, the transaction log needs to be periodically backed up to ensure that its contents are preserved. Further, the transaction log must be backed up before it can be used to recover a database. Backups can be made either to disk or to tape—our backup was just made to disk. When possible, the backups should be made to devices other than those that store the operational database and log. Backing up to removable devices allows the backups to be stored in a location physically removed from the database data center. This is important for recovery in the event of disasters caused by floods, hurricanes, earthquakes, and the like.

SQL Server Recovery Models

SQL Server supports three recovery models: simple, full, and bulk logged. With the simple recovery model, no logging is done. The only way to recover a database is to restore the database to the last backup. Changes made since that last backup are lost. The simple recovery model can be used for a database that is never changed—one having the names and locations of the occupants of a full graveyard, for example—or for one that is used for read-only analysis of data that are copied from some other transactional database.

Figure 10-48

Backing Up the VRG Database



Amazon Ex. 1008.0442

With full recovery, all database changes are logged. With bulk-logged database recovery, all changes are logged except those that cause large log entries. With bulk-logged recovery, changes to large text and graphic data items are not recorded to the log, actions such as CREATE INDEX are not logged, and some other bulk-oriented actions are not logged. An organization uses bulk-logged recovery if conserving log space is important and if the data used in the bulk operations are saved in some other way.

Restoring a Database

If the database and log files have been properly backed up, restoring the database is straightforward.

Restoring the VRG Database

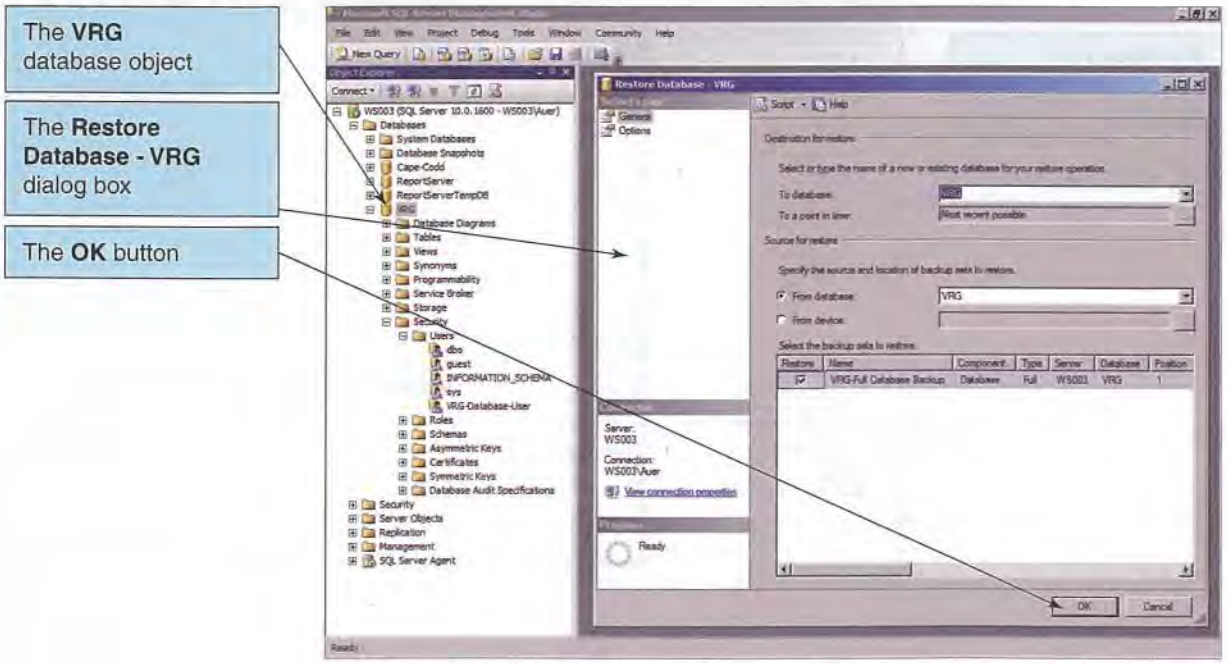
1. Right-click the **VRG** database object in the Object Explorer to display a shortcut menu.
2. In the shortcut menu, click the **Tasks** command, then click the **Restore** command. The Restore Database – VRG dialog box is displayed, as shown in Figure 10-49.
3. SQL Server 2008 has already set up all the backup options correctly for a simple full backup of the database.
4. If we were going to actually restore the VRG database, we would click the OK button at this point, and the database backup would be restored. However there is no need to do so at this point, so click the **Cancel** button.

You can use backup and restore to transfer a database to another computer or user (for example, to your professor!). Just do a full backup of the database you want to share to a file, say, the file *MyBackup*. Then, create a new database on another computer and name it whatever you want. Then, restore the database using the file MyBackup.

Database Maintenance Plans

You can create a database maintenance plan to facilitate the making of database and log backups, among other tasks, using SQL Server policy management. This topic, however, is beyond the scope of this book. For more information, see the Microsoft SQL Server 2008 Books Online at *http://msdn.microsoft.com/en-us/library/ms130214.aspx*.

Figure 10-49
Restoring the VRG Database



Amazon Ex. 1008.0443

Topics Not Discussed in This Chapter

Several important SQL Server topics are beyond the scope of this discussion. For one, SQL Server provides utilities to measure database activity and performance. The DBA can use these utilities when tuning the database. Another facility not described here is connecting Access to SQL Server. You can check the Access documentation for more information about this topic.

SQL Server 2008 provides facilities to support distributed database processing (called **replication** in SQL Server). Although very important in its own right, distributed database processing is beyond the scope of this text. Finally, SQL Server has facilities for processing database views in the form of XML documents. We will discuss those facilities in Chapter 14.

Summary

SQL Server 2008 can be installed on computers running Windows XP, Windows Vista, Windows Server 2003, or Windows Server 2008. Tables, views, indexes, and other database structures can be created in two ways. One is to use the graphical design tools, which are similar to those in Microsoft Access. The other is to write SQL statements to create the structures and then submit them to SQL Server via the SQL Server Management Studio utility. SQL Server supports all of the SQL DDL that you have learned in this text, including the IDENTITY keyword for defining surrogate keys. The only change required for the View Ridge schema was to change the name of the TRANSACTION table to TRANS.

Indexes are special data structures used to improve performance. SQL Server automatically creates an index on all primary and foreign keys. Additional views can be created using CREATE INDEX or the Manage Index graphical tool. SQL Server supports clustered and nonclustered indexes.

SQL Server supports a language called Transact-SQL, which surrounds basic SQL statements with programming constructs such as parameters, variables, and logic structures, such as IF, WHILE, and so forth.

SQL Server databases can be processed from application programs coded in standard programming languages, such as Visual Basic .NET, Visual C# .NET, or Visual C++ .NET or application logic can be placed in stored procedures and triggers. Stored procedures can be invoked from standard languages or from VBScript and JScript in Web pages. In this chapter, stored procedures were invoked from the SQL Server Query Manager. This technique should be used only during development and testing.

For security reasons, no one should process an SQL Server operational database in interactive mode. This chapter demonstrated SQL Server triggers for computing default values, for enforcing a data constraint, for updating a view, and for enforcing a mandatory child referential integrity constraint.

Three factors determine the concurrency control behavior of SQL Server: the transaction isolation level, the cursor concurrency setting, and locking hints provided in the SELECT clause. These factors are summarized in Figure 10-41. Behavior also changes depending on whether actions occur in the context of transactions or cursors or independently. Given these behavior declarations, SQL Server places locks on behalf of the developer. Locks may be placed at many levels of granularity and may be promoted or demoted as work progresses.

SQL Server supports log backups and both complete and differential database backups. Three recovery models are available: simple, full, and bulk logged. With simple recovery, no logging is done nor are log records applied. Full recovery logs all database operations and applies them for restoration. Bulk-logged recovery omits certain transactions that would otherwise consume large amounts of space in the log.

Key Terms

/* and */ signs
BEGIN . . . END keywords
CLOSE CURSOR keywords
clustered index
cmdlets
command-line utility
Connect to Server dialog box
database principals
dbo schema
DEALLOCATE CURSOR keywords
DECLARE CURSOR keywords
delimited identifier
differential backup
FETCH keyword
full backup
GO command
IF . . . ELSE keywords
index
Integrated Development Environment (IDE)
Microsoft SQL Server

Amazon Ex. 1008.0444

Microsoft SQL Server 2008 Management Studio
Microsoft Windows PowerShell
nonclustered index
OPEN CURSOR keywords
parameter
PowerShell sqlps utility
replication
reserved word
RETURN keyword
SCROLL_LOCK
server principals
SNAPSHOT transaction isolation level
SQL ALTER PROCEDURE statement
SQL CMD utility
SQL Common Language Runtime (CLR)
SQL CREATE PROCEDURE statement
SQL script
SQL script comment
SQL Server 2008 Express
SQL Server 2008 Express Advanced
SQL Server bit data type
SQL Server function @@Identity
SQL Server principals
SQL Server schema
SQL Server securable objects
transaction isolation level
Transact-SQL (T-SQL)
Transact-SQL @@FETCH_STATUS function
Transact-SQL BEGIN TRANSACTION command
Transact-SQL COMMIT TRANSACTION command
Transact-SQL control-of-flow language
Transact-SQL CONVERT function
Transact-SQL cursor
Transact-SQL GETDATE( ) function
Transact-SQL IDENTITY property
Transact-SQL IDENTITY_INSERT property
Transact-SQL PRINT command
Transact-SQL ROLLBACK TRANSACTION command
Transact-SQL SET ANSI_NULLS ON command
Transact-SQL SET QUOTED_IDENTIFIER ON command
Transact-SQL USE [{DatabaseName}] command
transaction log backup
two dashes (- -)
variable
WHILE keyword

Review Questions

If you have not already installed SQL Server 2008 (or do not otherwise have it available to you), you need to install a version of it at this point.

Review Questions 10.1–10.9 are based on a database named MEDIA that is used to record data about photographs that are stored in the database.

10.1 Create a database named MEDIA in SQL Server 2008. Use the default settings for file sizes, names, and locations.

10.2 Write an SQL statement to create a table named PICTURE with columns Name, Description, DateTaken, and FileName. Assume that Name is char(20), Description is varchar(200), DateTaken is smalldate, and FileName is char(45). Also assume that Name and DateTaken are required. Use Name as the primary key. Set the default value of Description to '(None)'.

10.3 Use the Microsoft SQL Server Management Studio to submit the SQL statement in Review Question 10.2 to create the PICTURE table in the MEDIA database.

10.4 Write a CREATE TABLE statement to create the table SLIDE_SHOW (ShowID, Name, Description, Purpose). Assume that ShowID is a surrogate key. Set the data type of Name and Description however you deem appropriate. Set the data type of Purpose to char (15) and limit it to the set of values ('Home', 'Office', 'Family', 'Recreation', 'Sports', 'Pets'). Execute your CREATE TABLE statement using Query Analyzer.

10.5 Use the Microsoft SQL Server Management Studio to create the table SHOW_PICTURE_INT as an intersection table between PICTURE and SLIDE_SHOW. Create appropriate relationships between PICTURE and SHOW_PICTURE_INT and between SLIDE_SHOW and SHOW_PICTURE_INT. Set the referential integrity properties to

Amazon Ex. 1008.0445

disallow any deletion of a SLIDE_SHOW row that has any SHOW_PICTURE_INT rows related to it. Set the referential integrity properties to cascade deletions when a PICTURE is deleted. Cascade updates to PICTURE.Name.

10.6 Write an SQL statement to create a view name PopularShows that has SLIDE_SHOW.Name and PICTURE.Name for all slide shows that have a Purpose of either 'Home' or 'Pets'. Execute this statement using the Microsoft SQL Server Management Studio.

10.7 Use the Microsoft SQL Server Management Studio GUI tools to determine that PopularShows was constructed correctly. Modify this view to include PICTURE.Description and FileName.

10.8 Can the SQL DELETE statement be used with the PopularShows view? Why or why not?

10.9 Under what circumstances can PopularShows be used for inserts and modifications?

Review Questions 10.10–10.25 are about terms and concepts discussed in this chapter.

10.10 In Figure 10-28, what is the purpose of the @RowCount variable?

10.11 In Figure 10-28, why is the SELECT statement that begins SELECT @CustomerID necessary?

10.12 Explain how you would change the stored procedure in Figure 10-28 to connect the customer to all artists who either (a) were born before 1900 or (b) had a null value for DateOfBirth.

10.13 Explain the purpose of the transaction shown in Figure 10-30.

10.14 What happens if an incorrect value of Copy is input to the stored procedure in Figure 10-30?

10.15 In Figure 10-30, what happens if the ROLLBACK statement is executed?

10.16 In Figure 10-30, why is SUM used instead of AVG?

10.17 What are the three primary factors that influence SQL Server locking behavior?

10.18 Explain why the strategy for storing CHECK constraint values in a separate table is better than implementing them in a table-based constraint. How can this strategy be used to implement the constraint on ARTIST.Nationality?

10.19 Explain why the view CustomerInterestsView in Figure 10-25 is not updatable. Describe the logic of the INSTEAD OF UPDATE trigger in Figure 10-36.

10.20 Explain what limitation must be enforced for the trigger in Figure 10-36 to be effective.

10.21 Explain the meaning of each of the transaction isolation levels under Options shown in Figure 10-41.

10.22 Explain the meaning of each of the cursor concurrency settings listed in Figure 10-41.

10.23 What is the purpose of locking hints?

10.24 What is the difference between complete and differential backups? Under what conditions are complete backups preferred? Under what conditions are differential backups preferred?

10.25 Explain the differences between simple, full, and bulk-logged recovery models. Under what conditions would you choose each one?

Amazon Ex. 1008.0446

Project Questions

In the Chapter 7 Review Questions, we introduced the Wedgewood Pacific Corporation (WPC) and developed the WPC database. Two of the tables that are used in the WPC database are:

DEPARTMENT (DepartmentName, BudgetCode, OfficeNumber, Phone)
EMPLOYEE (EmployeeNumber, FirstName, LastName, *Department*, Phone, Email)

Assume that the relationship between these tables is M-M, and use them as the basis for your answers to Review Questions 10.26 through 10.28.

10.26 Code an SQL Server trigger to enforce the constraint that an employee can never change his or her department.

10.27 Code an SQL Server trigger to allow the deletion of a department if it only has one employee. Assign the last employee to the Human Resources department.

10.28 Design a system of triggers to enforce the M-M relationship. Use Figure 10-38 as an example, but assume that departments with only one employee can be deleted. Assign the last employee in a department to Human Resources.

Review Question 10.29 is based on the View Ridge Gallery database discussed in this chapter.

10.29 If you have not already installed SQL Server 2008 (or do not otherwise have it available to you), you need to install a version of it at this point. Write SQL statements to accomplish the following tasks and submit them to SQL Server 2008 via the Microsoft SQL Server Management Studio.

A. Create an SQL Server 2008 database named VRG.

B. Create the VRG database tables shown in Figure 10-13, except do not create the NationalityValues constraint.

C. Populate your database with the sample data from Figure 10-24.

D. Create all the VRG views discussed in the Chapter 7 section on SQL views.

E. Write a stored procedure to read the ARTIST table and display the artist data using the Transact-SQL PRINT command.

F. Write a stored procedure that reads the ARTIST and WORK tables, and that accepts the name of the artist to display as an input parameter. Your procedure should then display the data for that artist, followed by a display of all the works for that artist stored in WORK.

G. Write a stored procedure to update customer phone data. Assume that your stored procedure receives LastName, FirstName, PriorAreaCode, NewAreaCode, Prior PhoneNumber, and NewPhoneNumber. Your procedure should first ensure that there is only one customer with the values of (LastName, FirstName, PriorAreaCode, PriorPhoneNumber). If not, produce an error message and quit. Otherwise, update the customer data with the new phone number data, and print a results message.

H. Create a table named ALLOWED_NATIONALITY with one column, called Nation. Place the values of all nationalities currently in the VRG database into the table. Write a trigger that will check to determine whether a new or

Amazon Ex. 1008.0447

updated value of Nationality resides in this table. If not, write an error message and roll back the insert or change. Use the Microsoft SQL Server Management Studio to demonstrate that your trigger works.

I. Create a view having all of the data from the WORK and TRANS tables except for the surrogate keys. Write and insert an INSTEAD OF trigger on this view that will create a new row in both WORK and TRANS. Use the Microsoft SQL Server Management Studio to demonstrate that your trigger works. Hint: Recall that you can issue an INSERT command on WORK and TRANS without specifying a value for the surrogate key. SQL Server will provide it.



Answer questions A through M for Marcia's Dry Cleaning at the end of Chapter 7, page 293 if you have not already done so. Use the SQL Server data types in your answer.

A. Using SQL Server 2008 and the Microsoft SQL Server Management Studio, execute all of the SQL statements that you created in your answers to A through M for Marcia's Dry Cleaning at the end of Chapter 7.

B. Assume that the relationship between ORDER and ORDER_ITEM is M-M. Design triggers to enforce this relationship. Use Figure 10-38 and the discussion of that figure as an example, but assume that Marcia does allow ORDERs and their related ORDER_ITEM rows to be deleted. Use the deletion strategy shown in Figures 7-29 and 7-30 for this case.

C. Write and test the triggers you designed in part B.



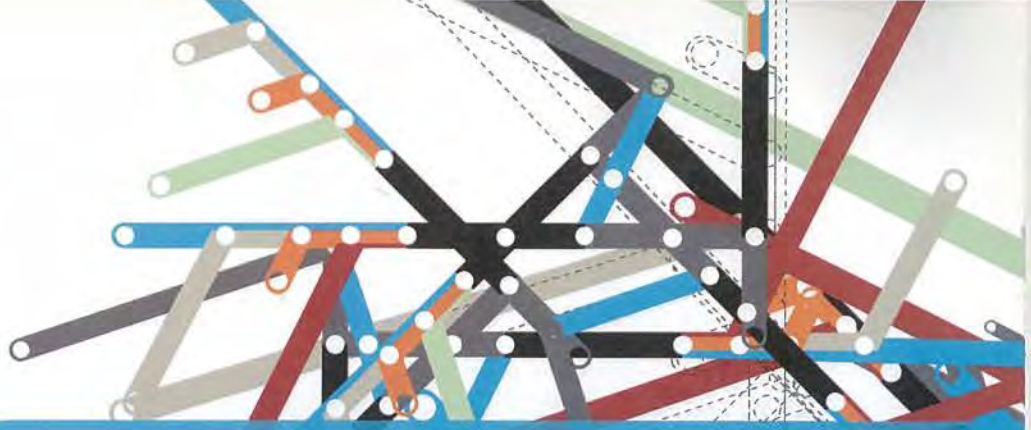
Answer questions A through N for Morgan Importing at the end of Chapter 7, page 294, if you have not already done so. Use the SQL Server data types in your answer.

A. Using SQL Server 2008 and the Microsoft SQL Server Management Studio, execute all of the SQL statements that you created in your answers to A through N for Morgan Importing at the end of Chapter 7.

B. Assume that the relationship between SHIPMENT and SHIPMENT_ITEM is M-M. Design triggers to enforce this relationship. Use Figure 10-38 and the discussion of that figure as an example, but assume that Morgan does allow SHIPMENTs and their related SHIPMENT_ITEM rows to be deleted. Use the deletion strategy shown in Figures 7-29 and 7-30 for this case.

C. Write and test the triggers you designed in part B.

Amazon Ex. 1008.0448



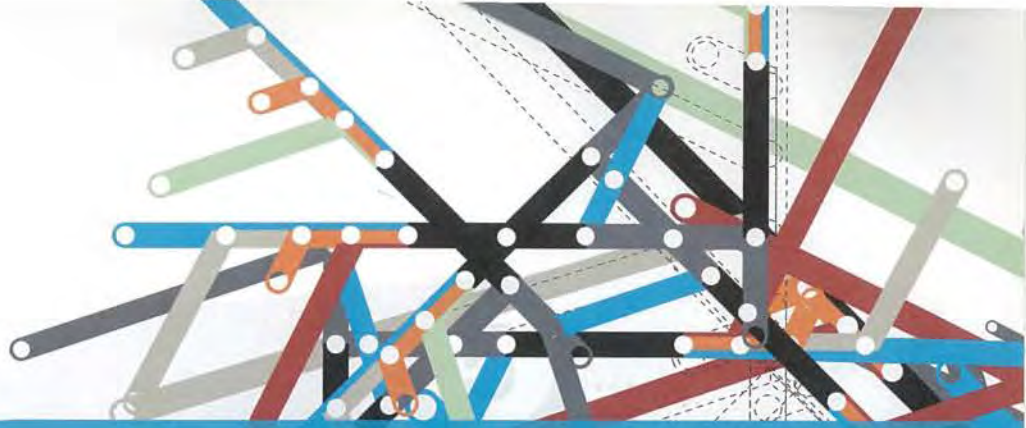
10A Managing Databases with Oracle Database 11g

Chapter Objectives

- To be able to install Oracle Database 11g and create a database.
- To be able to use Oracle Database 11g's Web-based Enterprise Manager Database Control utility.
- To be able to use Oracle Database 11g's graphical utilities.
- To be able to create and use Oracle Database 11g namespaces.
- To be able to submit both SQL DDL and DML via the SQL Developer utility.
- To understand the purpose and role of stored procedures and learn how to create simple stored procedures.
- To understand the purpose and role of triggers and learn how to create simple stored procedures.
- To understand how Oracle Database 11g implements concurrency control.
- To understand the fundamental features of Oracle Database 11g backup and recovery facilities.

A complete version of this chapter is available on the textbook's Web site:
www.pearsonhighered.com/kroenke

Amazon Ex. 1008.0449



10B Managing Databases with MySQL 5.1

Chapter Objectives

- To be able to install MySQL and create a database.
- To be able to use MySQL's graphical utilities.
- To be able to submit both SQL DDL and DML via the MySQL Query Browser.
- To understand the purpose and role of stored procedures and know how to create simple stored procedures.
- To understand the purpose and role of triggers and know how to create simple stored procedures.
- To understand how MySQL implements concurrency control.
- To understand the fundamental features of MySQL backup and recovery facilities.

A complete version of this chapter is available on the textbook's Web site:
www.pearsonhighered.com/kroenke

Amazon Ex. 1008.0450

Part 5

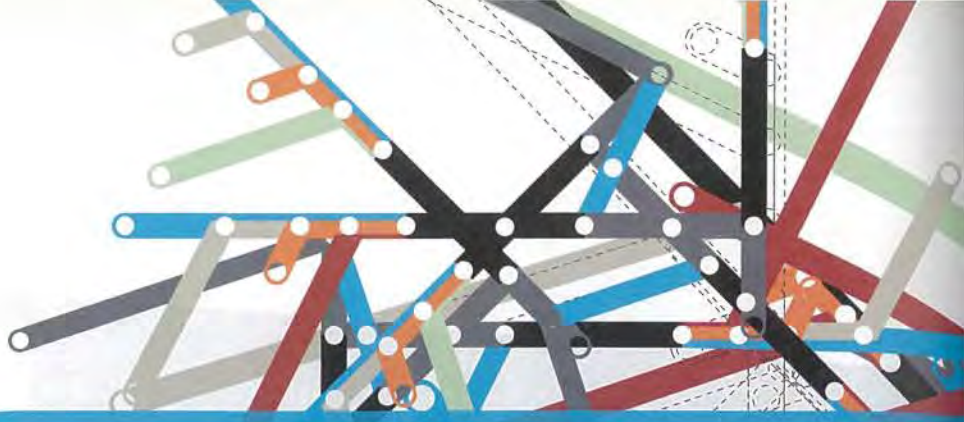
Database Access Standards

The three chapters in this section examine standards for database application processing. We begin in Chapter 11 by discussing database access standards, including ODBC, ADO.NET and ASP.NET in Microsoft's .NET Framework, and the Java-based JDBC and Java Server Pages (JSP) technologies. Even though some of these standards are no longer on the leading edge of database processing, many applications still use them, and you will likely encounter them in your career. Chapter 11 then describes the use of the popular PHP scripting language to create Web pages that access the View Ridge Gallery database.

Chapter 12 discusses one of the most important developments in information technology today, the confluence of database processing and document processing. This chapter introduces you to XML and XML Schema. Chapter 13 discusses business intelligence (BI) systems and the data warehouse and data mart databases that support them.



Amazon Ex. 1008.0451



11 The Web Server Environment

Chapter Objectives

- To understand the nature and characteristics of the data environment that surrounds Internet technology database applications.
- To learn the purpose, features, and facilities of ODBC.
- To understand the characteristics of the Microsoft .NET Framework
- To understand the nature and goals of OLE DB.
- To learn the characteristics and object model of ADO.NET.
- To understand the characteristics of JDBC and the four types of JDBC drivers.
- To understand the nature of JSP and know the differences between JSP and ASP.NET.
- To understand HTML and PHP.
- To be able to construct Web database applications pages using PHP.

This chapter discusses some traditional standard interfaces and current tools for accessing database servers. ODBC, or the Open Database Connectivity standard, was developed in the early 1990s to provide a product-independent interface to relational and other tabular data. In the mid-1990s, Microsoft announced OLE DB, which is an object-oriented interface that encapsulates data-server functionality. Microsoft then developed Active Data Objects (ADO), which is a set of objects for utilizing OLE DB that is designed for use by any language, including VBScript, and JScript. This technology was used in Active Server Pages (ASP), which were the basis of Web database applications. In 2002, Microsoft introduced the .NET Framework, which included ADO.NET

Amazon Ex. 1008.0452

(the successor to ADO) and ASP.NET (the successor to ASP) components. Today, the .NET Framework is the basis for all application development using Microsoft technology.

As an alternative to the Microsoft technologies, Sun Microsystems developed the Java platform, which includes the Java programming language, Java Database Connectivity (JDBC) and Java Server Pages (JSP).

While the .NET and Java technologies are important development platforms, there are additional technologies that have been developed by other companies and open-source projects. We will use two of these independently developed tools in this chapter—the Eclipse integrated development environment (IDE) and the PHP scripting language.

However, before considering these standards, we need to gain some perspective on the data environment that surrounds the Web server in Internet technology database applications.

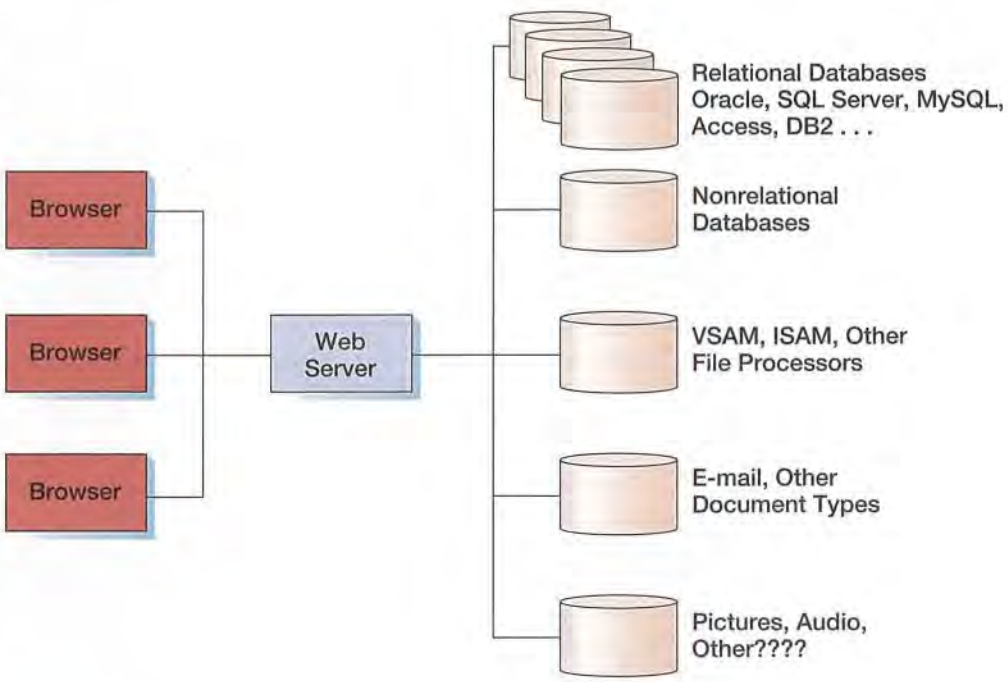
The Web Database Processing Environment

The environment in which today's database applications reside is rich and complicated. As shown in Figure 11-1, a typical Web server needs to publish applications that involve data of dozens of different data types. So far in this text, we have considered only relational databases, but as you can see from this figure, there are many other data types as well.

Consider the problems that the developer of Web server applications has when integrating these data. The developer may need to connect to an Oracle database; a DB2 mainframe database; a nonrelational database, such as IMS, IBM's older DBMS product; file-processing

Figure 11-1

The Variety of Data Types in Web Database Applications



Amazon Ex. 1008.0453

data, such as VSAM and ISAM; e-mail directories; and so forth. Each one of these products has a different programming interface that the developer must learn. Further, these products evolve, thus new features and functions will be added over time that will increase the developer's challenge.

Several standard interfaces have been developed for accessing database servers. Every DBMS product has an **application program interface (API)**. An API is a collection of objects, methods, and properties for executing DBMS functions from program code. Unfortunately, each DBMS has its own API, and APIs vary from one DBMS product to another. To save programmers from having to learn to use many different interfaces, the computer industry has developed standards for database access.

The **Open Database Connectivity (ODBC)** standard was developed in the early 1990s to provide a DBMS-independent means for processing relational database data. In the mid-1990s, Microsoft announced **OLE DB**, which is an object-oriented interface that encapsulates data-server functionality. OLE DB was designed not just for access to relational databases, but also for accessing many other types of data as well. As a **Component Object Model (COM)** interface, OLE DB is readily accessible to programmers, using such programming languages as C, C#, and Java. However, OLE DB is not as accessible to users of Visual Basic (VB) and scripting languages. Therefore, Microsoft developed **Active Data Objects (ADO)**, which is a set of objects for utilizing OLE DB that is designed for use by any language, including Visual Basic (VB), VBScript, and JScript. ADO has now been followed by **ADO.NET** (pronounced "A-D-O-dot-NET"), which is an improved version of ADO developed as part of Microsoft's .NET (pronounced "dot-NET") initiative and a component of the .NET Framework.

ADO technology is used in **Active Server Pages (ASP)** Web pages to create Web-based database applications. ASP is a combination of Hypertext Markup Language (HTML) and VBScript or JScript that can read and write database data and transmit it over public and private networks, using Internet protocols. ASP runs on Microsoft's Web server product, **Internet Information Services (IIS)**. When ADO.NET was introduced, Microsoft also introduced **ASP.NET**. ASP.NET is the successor to ASP, and is the preferred Web page technology in the .NET Framework.

Of course, there are other connectivity methods and standards besides those propagated by Microsoft. The main alternatives to ADO.NET technology are based on or associated with Sun Microsystems' **Java platform**, and include the **Java programming language**, **Java Database Connectivity (JBDC)**, **Java Data Objects (JDO)** and **JavaServer Pages (JSP)**.

JavaServer Pages (JSP) technology is a combination of HTML and Java that accomplishes the same function as ASP.NET by compiling pages into Java servlets. JSP may connect to databases using Java Database Connectivity (JDBC). JSP is often used on the open-source **Apache Web server**.

However, the defining characteristic of the Java-related technology is that you must use Java as the programming language. You cannot even use JavaScript, Java's somewhat related scripting language cousin. If you know (or want to learn) Java, this if fine.

While the Microsoft .NET Framework and the Sun Microsystems Java platform are two major players in Web database application development, there are other options. One such product is PHP, which is an open-source Web page programming language, and another favorite combination of Web developers is the Apache Web server with the MySQL DBMS and the PHP language. This combination is called **AMP** (Apache-MySQL-PHP). When running on the Linux operating system, it's it is referred to as **LAMP**; when running on the Windows operating system, it's it is referred to as **WAMP**. And because PHP works with all DBMS products, we will use it in this book. Other possibilities include the Perl and Python languages (both of which can be the "P" in AMP, LAMP and WAMP), and the Ruby language with its Web development framework called Ruby on Rails.

In a Web-based database processing environment, if the Web server and the DBMS can run on the same computer, the system has **two-tier architecture**. (One tier is for the browsers, and one is for the Web server/DBMS computer.) Alternatively, the Web server and DBMS can run on different computers, in which case the system has **three-tier architecture**. High-performance applications might use many Web server computers, and in some systems several computers can run the DBMS, as well. In the latter case, if the DBMS computers are processing the same databases, the system is referred to as a *distributed database*.

Amazon Ex. 1008.0454

The ODBC Standard

The ODBC standard was created to address the data access problem that concerns relational databases and data sources that are table-like, such as spreadsheets. As shown in Figure 11-2, ODBC is an interface between the Web server (or other database application) and the DBMS. It consists of a set of standards by which SQL statements can be issued and results and error messages can be returned. As shown in Figure 11-2, developers can call the DBMS using native DBMS interfaces (which are APIs) if they want to (sometimes they do this to improve performance), but the developer who does not have the time or desire to learn many different DBMS native libraries can use the ODBC instead.

The ODBC standard is an interface by which application programs can access and process databases and tabular data in a DBMS-independent manner. This means, for example, that an application that uses the ODBC interface could process an Oracle database, an SQL Server database, a spreadsheet, or any other ODBC-compliant database without making any coding changes. The goal is to allow a developer to create a single application that can access databases supported by different DBMS products without needing to be changed, or even recompiled.

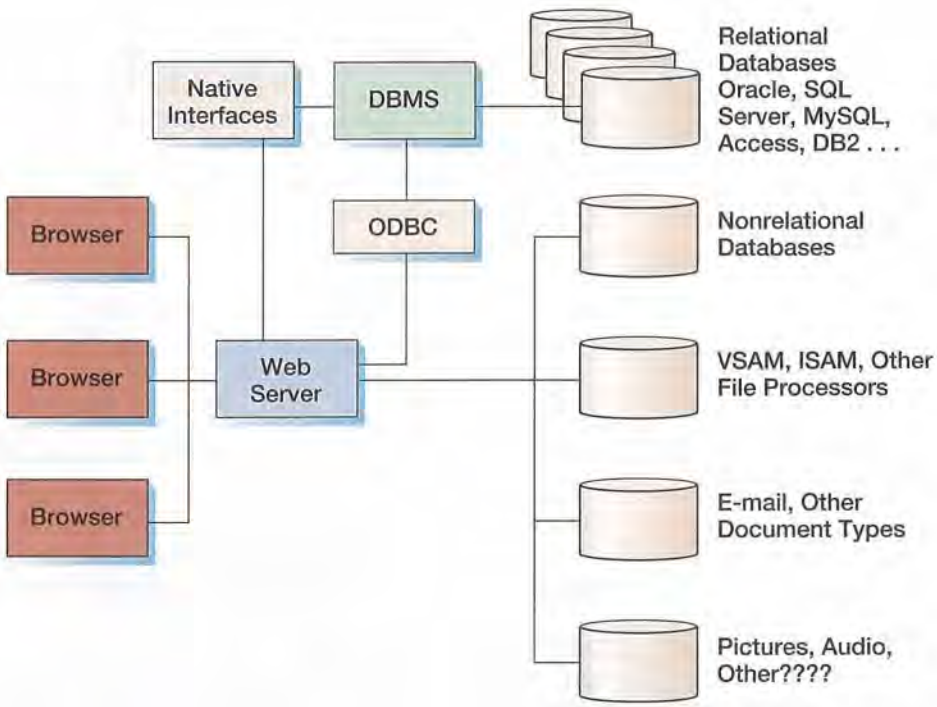
ODBC was developed by a committee of industry experts from the X/Open and SQL Access Group committees. Several such standards were proposed, but ODBC emerged as the winner, primarily because it had been implemented by Microsoft and is an important part of Windows. Microsoft's initial interest in support of such a standard was to allow products such as Microsoft Excel to access database data from a variety of DBMS products without having to be recompiled. Of course, Microsoft's interests have changed since the introduction of OLE DB and ADO.NET.

ODBC Architecture

Figure 11-3 shows the components of the ODBC standard. The application program, driver manager, and DBMS drivers all reside on the application server computer. The drivers send requests to data sources, which reside on the database server. According to the standard, an **ODBC data source** is the database and its associated DBMS, operating system, and network platform. An ODBC data source can be a relational database; it can also be a file server, such as BTrieve, or even a spreadsheet.

The application issues requests to create a connection with a data source; to issue SQL statements and receive results; to process errors; and to start, commit, and roll back transactions.

Figure 11-2
Role of the ODBC Standard



Amazon Ex. 1008.0455

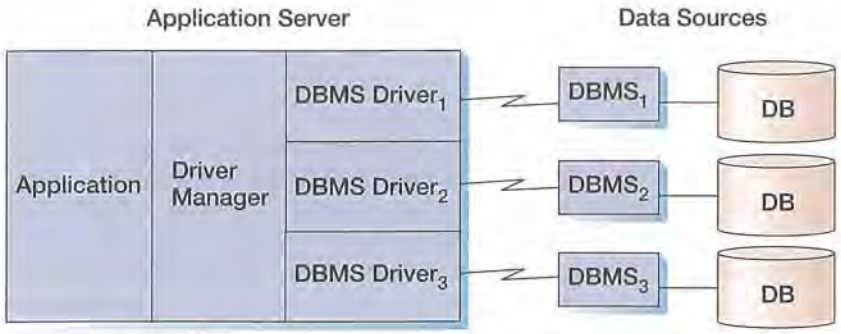


Figure 11-3
ODBC Architecture

Application can process a database using any of the three DBMS products.

ODBC provides a standard means for each of these requests, and it defines a standard set of error codes and messages.

The **ODBC driver manager** serves as an intermediary between the application and the DBMS drivers. When the application requests a connection, the driver manager determines the type of DBMS that processes a given ODBC data source and loads that driver into memory (if it is not already loaded). The driver manager also processes certain initialization requests and validates the format and order of ODBC requests that it receives from the application. For Windows, the driver manager is provided by Microsoft.

An **ODBC driver** processes ODBC requests and submits specific SQL statements to a given type of data source. Each data source type has a different driver. For example, there are drivers for SQL Server, for Oracle, for MySQL, for Access, and for all of the other products whose vendors have chosen to participate in the ODBC standard. Drivers are supplied by DBMS vendors and by independent software companies.

It is the responsibility of the driver to ensure that standard ODBC commands execute correctly. In some cases, if the data source is itself not SQL compliant, the driver may need to perform considerable processing to fill in for a lack of capability at the data source. In other cases, when the data source supports full SQL, the driver need only pass the request through for processing by the data source. The driver also converts data source error codes and messages into the ODBC standard codes and messages.

ODBC identifies two types of drivers: single tier and multiple tier. An **ODBC single-tier driver** processes both ODBC calls and SQL statements. An example of a single-tier driver is shown in Figure 11-4(a). In this example, the data are stored in Xbase files (the format used by FoxPro, dBase, and others). Because Xbase file managers do not process SQL, it is the job of the driver to translate the SQL request into Xbase file-manipulation commands and to transform the results back into SQL form.

An **ODBC multiple-tier driver** processes ODBC calls, but passes the SQL requests directly to the database server. Although it may reformat an SQL request to conform to the dialect of a particular data source, it does not process the SQL. An example of the use of a multiple-tier driver is shown in Figure 11-4(b).

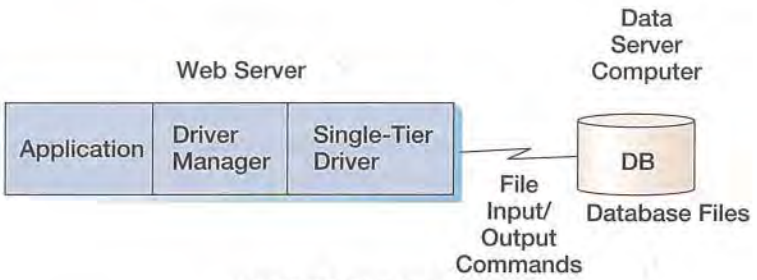
Conformance Levels

The creators of the ODBC standard faced a dilemma. If they chose to describe a standard for a minimal level of capability, many vendors would be able to comply. But if they did so, the standard would represent only a small portion of the complete power and expressiveness of ODBC and SQL. However, if the standard addressed a very high level of capability, only a few vendors would be able to comply with the standard, and it would become unimportant. To deal with this dilemma, the committee wisely chose to define levels of conformance to the standard. The committee defined two types of conformance: ODBC conformance and SQL conformance.

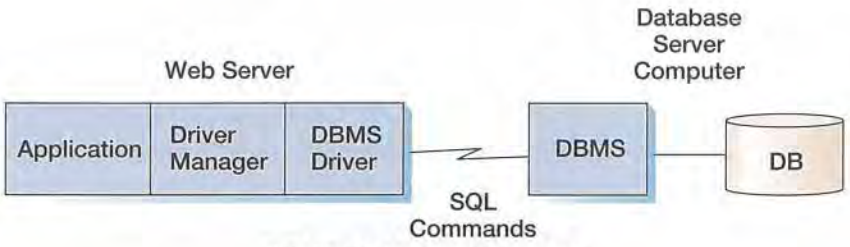
ODBC Conformance Level

ODBC conformance levels are concerned with the features and functions that are made available through the driver's application program interface (API). A driver API is a set of functions

Amazon Ex. 1008.0456



(a) ODBC Single-tier Driver



(b) ODBC Multiple-tier Driver

Figure 11-4
ODBC Driver Types

that the application can call to receive services. Figure 11-5 summarizes the three levels of ODBC conformance that are addressed in the standard. In practice, almost all drivers provide at least Level 1 API conformance, so the core API level is not too important.

An application can call a driver to determine which level of ODBC conformance it provides. If the application requires a level of conformance that is not present, it can terminate the session in an orderly fashion and generate appropriate messages to the user. Or, the application can be written to use higher-level conformance features if they are available and to work around the missing functions if a higher level is not available.

Figure 11-5
Summary of ODBC Conformance Levels

Core API
- Connect to data sources
- Prepare and execute SQL statements
- Retrieve data from a result set
- Commit or roll back transactions
- Retrieve error information

Level 1 API
- Core API
- Connect to data sources with driver-specific information
- Send and receive partial results
- Retrieve catalog information
- Retrieve information about driver options, capabilities, and functions

Level 2 API
- Core and Level 1 API
- Browse possible connections and data sources
- Retrieve native form of SQL
- Call a translation library
- Process a scrollable cursor

Amazon Ex. 1008.0457

For example, drivers at the Level 2 API must provide a scrollable cursor. Using conformance levels, an application could be written to use cursors if they are available; but if they are not, to work around the missing feature, it would select needed data using very restrictive WHERE clauses. Doing this would ensure that only a few rows were returned at a time to the application, and it would process those rows using a cursor that it maintained itself. Performance would likely be slower in the second case, but at least the application would be able to successfully execute.

SQL Conformance Level

ODBC SQL conformance levels specify which SQL statements, expressions, and data types a driver can process. Three SQL conformance levels are defined, as summarized in Figure 11-6. The capability of the minimum SQL grammar is very limited, and most drivers support at least the core SQL grammar.

As with ODBC conformance levels, an application can call the driver to determine what level of SQL conformance it supports. With that information, the application can then determine which SQL statements can be issued. If necessary, the application can then terminate the session or use alternative, less-powerful means of obtaining the data.

Creating an ODBC Data Source Name

An **ODBC data source** is an ODBC data structure that identifies a database and the DBMS that processes it. Data sources identify other types of data, such as spreadsheets and other non-database tabular data stores, but we are not concerned with that use here.

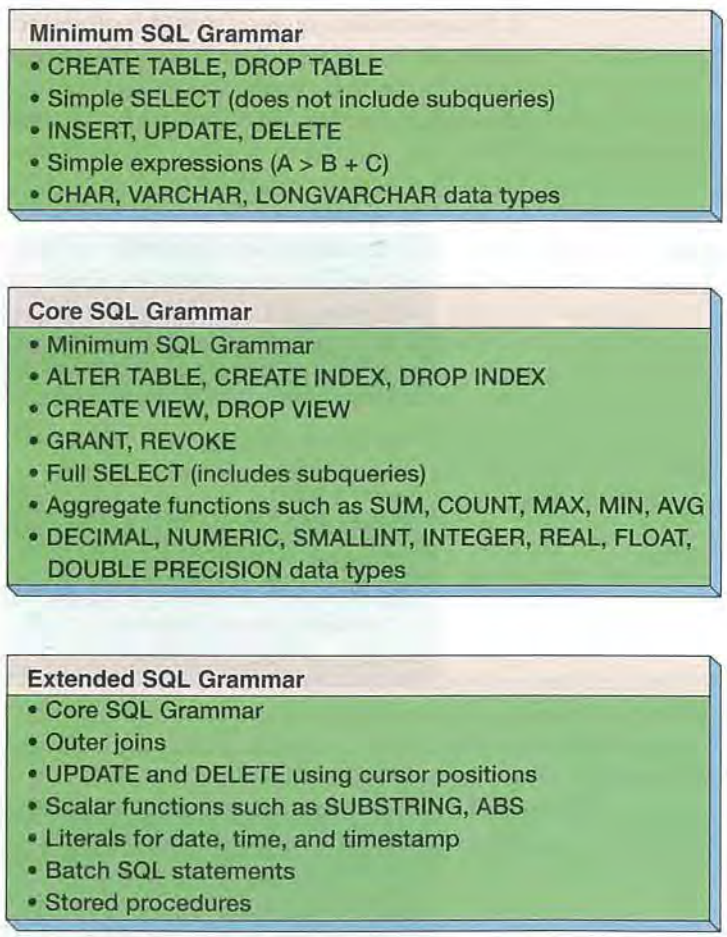


Figure 11-6
Summary of SQL Conformance Levels

Amazon Ex. 1008.0458

The three types of data sources are file, system, and user. A **file data source** is a file that can be shared among database users. The only requirement is that the users have the same DBMS driver and privilege to access the database. The data source file can be e-mailed or otherwise distributed to possible users. A **system data source** is one that is local to a single computer. The operating system and any user on that system (with proper privileges) can use a system data source. A **user data source** is available only to the user who created it.

In general, the best choice for Internet applications is to create a system data source on the Web server. Browser users then access the Web server, which in turn uses the system data source to set up a connection with the DBMS and the database.

We need a system data source for the View Ridge Gallery VRG database so that we can use it in a Web database processing application. To create a system data source in a Windows operating system, you use the **ODBC Data Source Administrator**.[1]

Opening the ODBC Data Source Administrator in Windows XP or Windows Vista

1. Open the Windows Control Panel by clicking the **Start** button and then **Control Panel**.
2. Switch the Control Panel to **Classic View** if necessary.
3. In the Control Panel window, double-click **Administrative Tools**.
4. In the Administrative Tools window, double-click the **Data Sources (ODBC)** shortcut icon.

Opening the ODBC Data Source Administrator in Windows Server 2008

1. Click the **Start** button and then click the **All Programs** button.
2. Click the **Administrative Tools** folder to open it.
3. Click the **Data Sources (ODBC)** program.

We can now use the ODBC Data Source Administrator to create a system data source named VRG for use with SQL Server 2008:

Creating the VRG System Data Source

1. In the ODBC Data Source Administrator, click the **System DSN** tab and then click the **Add** button.
2. In the Create New Data Source dialog box, click the **Data Sources (ODBC)** program.
3. Select the **SQL Server Native Client 10**, as shown in Figure 11-7.
4. Click the **Finish** button. The Create New Data Source to SQL Server dialog box appears.
5. In the Create New Data Source to SQL Server dialog box, enter the information shown in Figure 11-8 (note that the database server is selected from the Server drop-down list) and then click the **Finish** button.
6. The completed VRG system data source is shown in Figure 11-9. Click the **OK** button to close the ODBC Data Source Administrator.

We will use the VRG DSN later in this chapter to process the SQL Server database created in Chapter 10. Similarly, if you are using either the Oracle or MySQL DBMS, you should create an appropriate system data source for use with your Oracle or MySQL version of the View Ridge Gallery database.

[1] If you are using a 64-bit Windows operating system, be aware that there are two different ODBC Data Source Administrator programs provided—one for 32-bit applications and one for 64-bit applications. The ODBC Data Source Administrator used if you follow the steps in the text is the 64-bit version. However, if you are running a 32-bit DBMS (for example, the 32-bit version of SQL Server 2008 Express Advanced), then you must use the 32-bit version of the ODBC Data Source Administrator. In the 64-bit of Vista, this is the odbcad32.exe program located at C:\Windows\sysWOW64\odbcad32.exe.

Amazon Ex. 1008.0459

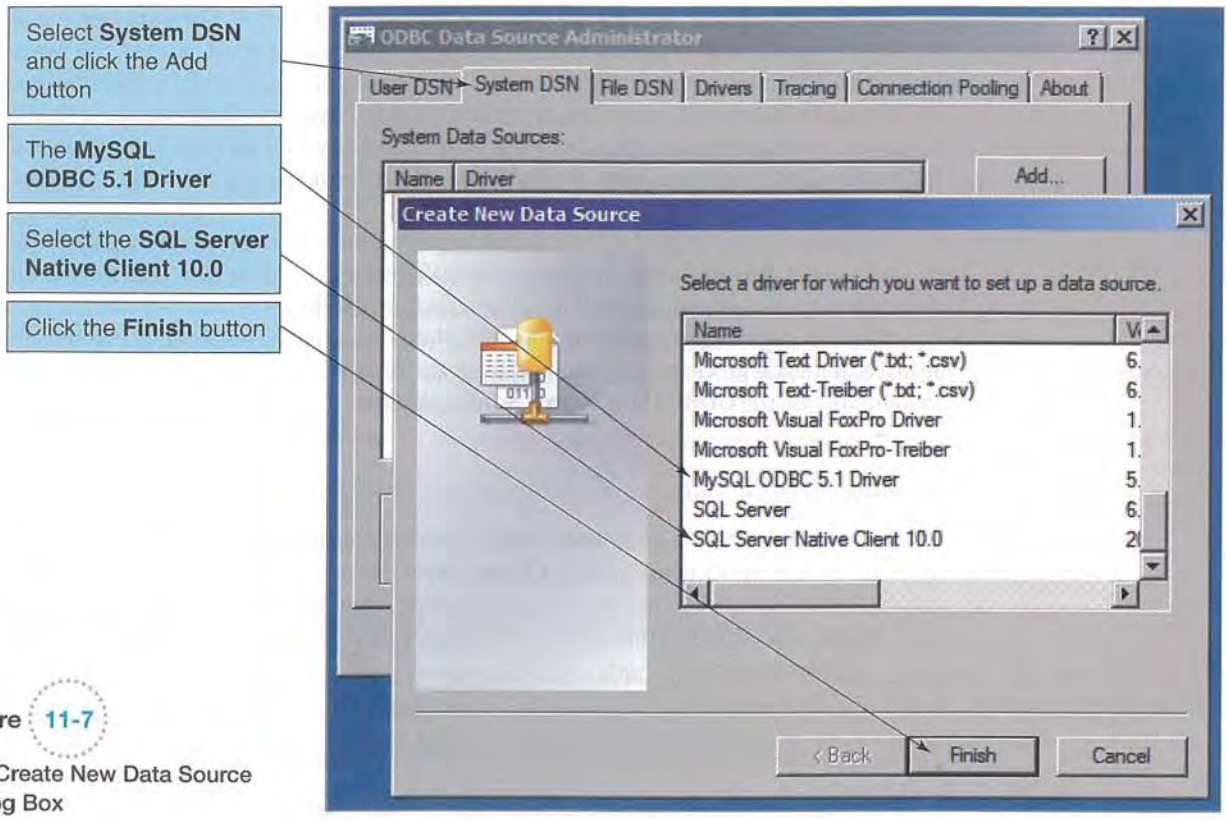
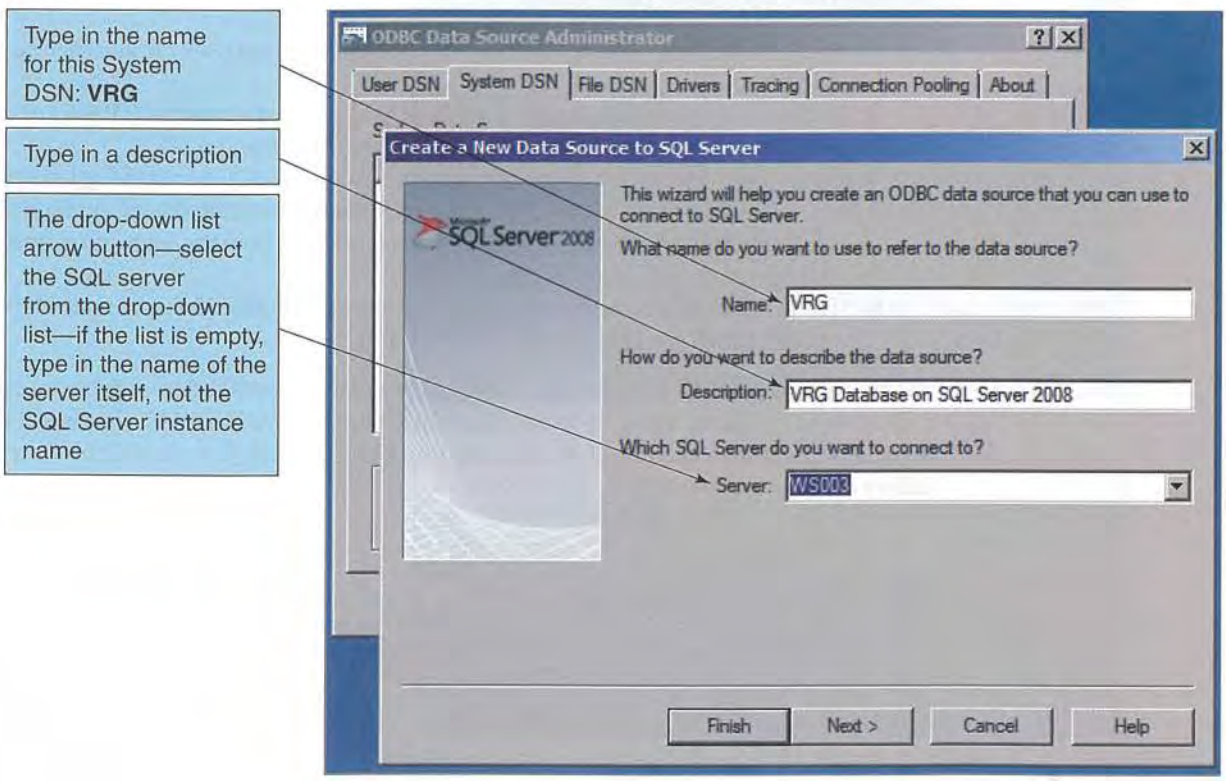


Figure 11-7

The Create New Data Source Dialog Box

Figure 11-8

The Create New Data Source to SQL Server Dialog Box



Amazon Ex. 1008.0460

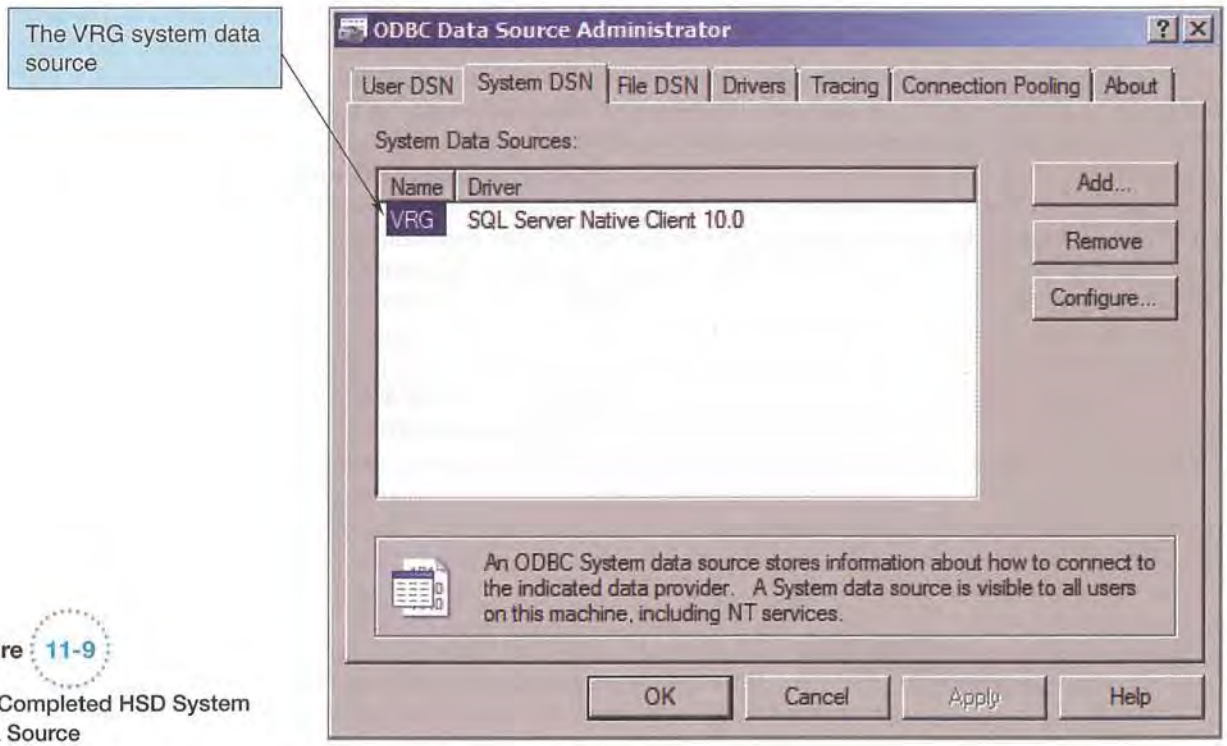


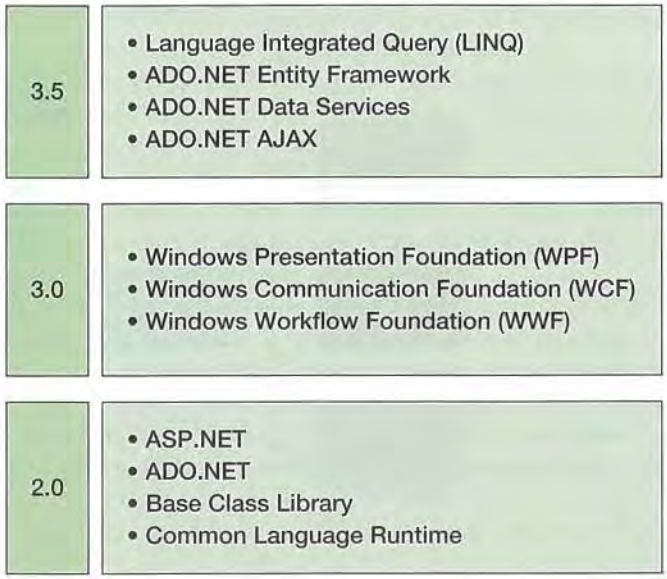
Figure 11-9
The Completed HSD System Data Source

The Mircosoft .NET Framework and ADO.NET

The **NET Framework** is Microsoft's comprehensive application development platform. Web database applications tools are included in the .NET Framework. Originally released as the .NET Framework 1.0 in January 2002, the current version is the .NET Framework 3.5 SP1 (and this is the version needed for use with SQL Server 2008).

As shown in Figure 11-10, the .NET Framework can best be visualized as a set of building blocks stacked on top of each other. Each additional block adds additional functionality to the

Figure 11-10
The Microsoft .NET Framework Structure



Amazon Ex. 1008.0461

components already existing in previous blocks, and if earlier components need to be updated this is done by service packs to the older blocks. Thus, .NET Framework 2.0 SP2 and .NET Framework SP2 are included as part of .NET Framework 3.5 SP1.

While Figure 11-10 does not show every feature of the .NET Framework 3.5 SP1, the basic structure is easy to see. The .NET Framework 2.0 is now the basic layer and contains the most basic features. These include the **Common Language Runtime** and the **Base Class Library**, which support all of the programming languages (for example, VB.NET and Visual C#.NET) used with the .NET Framework. This layer also includes the ADO.NET and ASP.NET components, which are needed for Web database applications.

The .NET Framework 3.0 added a set of components that are not of interest to us here. We are more concerned with the features added in .NET Framework 3.5 and 3.5 SP1. Note that several extensions to ADO.NET have been included, such as the **ADO.NET Entity Framework**, which supports Microsoft's emerging **Entity Data Model (EDM)** data modeling technology, as well as the **Language Integrated Query (LINQ)** component, which allows SQL queries to be programmed directly into application programs in a simple manner.

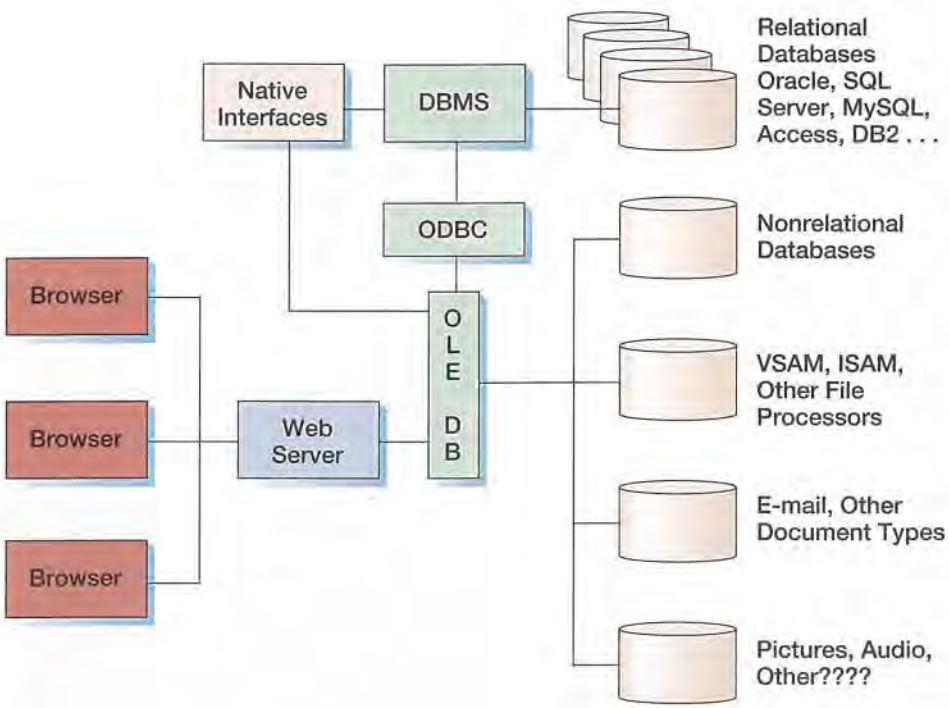
Now that we understand the basic structure of the .NET Framework, we can look at some of the pieces in detail.

BY THE WAY The Microsoft Entity Data Model (EDM) is similar in concept to the Semantic Object Model discussed in Appendix E of this book. A discussion of the EDM can be found at msdn.microsoft.com/en-us/library/aa697428(VS.80).aspx.

OLE DB

ODBC has been a tremendous success and has greatly simplified some database development tasks. However, it does have some disadvantages, and in particular one substantial disadvantage that Microsoft addressed by creating OLE DB. Figure 11-11 shows the relationship among OLE DB, ODBC, and other data types. OLE DB is one of the foundations of data access in the Microsoft world. As such, it is important to understand the fundamental ideas of OLE DB, even if you will

Figure 11-11
The Role of OLE DB



Amazon Ex. 1008.0462

only work with the ADO.NET interface that lies on top of it because, as you will see, OLE DB remains as a data provider to ADO.NET. In this section, we present essential OLE DB concepts, and use them to introduce some important object-oriented programming topics.

OLE DB provides an object-oriented interface to data of almost any type. DBMS vendors can wrap portions of their native libraries in OLE DB objects to expose their product's functionality through this interface. OLE DB can also be used as an interface to ODBC data sources. Finally, OLE DB was developed to support the processing of nonrelational data as well.

OLE DB is an implementation of the Microsoft **Object Linking and Embedding (OLE)** object standard. OLE DB objects are Component Object Model (COM) objects and support all required interfaces for such objects. Fundamentally, OLE DB breaks the features and functions of a DBMS up into COM objects. Some objects support query operations; others perform updates; others support the creation of database schema constructs, such as tables, indexes, and views; and still others perform transaction management, such as optimistic locking.

This characteristic overcomes a major disadvantage of ODBC. With ODBC, a vendor must create an ODBC driver for almost all DBMS features and functions in order to participate in ODBC at all. This is a large task that requires a substantial investment. With OLE DB, however, a DBMS vendor can implement portions of a product. One could, for example, implement only the query processor, participate in OLE DB, and hence be accessible to customers using ADO.NET. Later, the vendor could add more objects and interfaces to increase OLE DB functionality.

This text does not assume that you are an object-oriented programmer, so we need to develop a few concepts. In particular, you need to understand abstractions, methods, properties, and collections. An **abstraction** is a generalization of something. ODBC interfaces are abstractions of native DBMS access methods. When we abstract something, we lose detail, but we gain the ability to work with a broader range of types.

For example, a **recordset** is an abstraction of a relation. In this abstraction, a recordset is defined to have certain characteristics that will be common to all recordsets. Every recordset, for instance, has a set of columns, which in this abstraction is called Fields. Now, the goal of abstraction is to capture everything important but to omit details that are not needed by users of the abstraction. Thus, Oracle relations may have some characteristics that are not represented in a recordset; the same might be true for relations in SQL Server, DB2, and in other DBMS products. These unique characteristics will be lost in the abstraction, but if the abstraction is a good one, no one will care.

Moving up a level, a **rowset** is the OLE DB abstraction of a recordset. Now, why does OLE DB need to define another abstraction? Because OLE DB addresses data sources that are not tables but that do have *some of* the characteristics of tables. Consider all of the e-mail addresses in your personal e-mail file. Are those addresses the same as a relation? No, but they do share some of the characteristics that relations have. Each address is a semantically related group of data items. Like rows of a table, it is sensible to go to the first one, move to the next one, and so forth. But, unlike relations, they are not all of the same type. Some addresses are for individuals, others are for mailing lists. Thus, any action on a recordset that depends on everything in the recordset being the same kind of thing cannot be used on a rowset.

Working from the top down, OLE DB defines a set of data properties and behaviors for rowsets. Every rowset has those properties and behaviors. Furthermore, OLE DB defines a recordset as a subtype of a rowset. Recordsets have all of the properties and behaviors that rowsets have, plus they have some that are uniquely characteristic of recordsets.

Abstraction is both common and useful. You will hear of abstractions of transaction management or abstractions of querying or abstractions of interfaces. This simply means that certain characteristics of a set of things are formally defined as a type.

An object-oriented programming **object** is an abstraction that is defined by its properties and methods. For example, a recordset object has an AllowEdits property and a RecordsetType property and an EOF property. These **properties** represent characteristics of the recordset abstraction. An object also has actions that it can perform that are called **methods**. A recordset has methods such as Open, MoveFirst, MoveNext, and Close. Strictly speaking, the definition of an object abstraction is called an **object class**, or just a *class*. An instance of an object class, such as a particular recordset, is called an *object*. All objects of a class have the same methods and the same properties, but the values of the properties vary from object to object.

Amazon Ex. 1008.0463

The last term we need to address is *collection*. A **collection** is an object that contains a group of other objects. A recordset has a collection of other objects called Fields. The collection has properties and methods. One of the properties of all collections is Count, which is the number of objects in the collection. Thus, recordset.Fields.Count is the number of fields in the collection. In OLE DB, collections are named as the plural of the objects they collect. Thus, there is a Fields collection of Field objects, an Errors collection of Error objects, a Parameters collection of Parameters, and so forth. An important method of a collection is an iterator, which is a method that can be used to pass through or otherwise identify the items in the collection.

Goals of OLE DB

The major goals for OLE DB are listed in Figure 11-12. First, as mentioned, OLE DB breaks DBMS functionality and services into object pieces. This partitioning means great flexibility for both **data consumers** (users of OLE DB functionality) and **data providers** (vendors of products that deliver OLE DB functionality). Data consumers take only the objects and functionality they need; a wireless device for reading a database can have a very slim footprint. Unlike with ODBC, data providers need only implement a portion of DBMS functionality. This partitioning also means that data providers can deliver capabilities in multiple interfaces.

This last point needs expansion. An object interface is a packaging of objects. An **interface** is specified by a set of objects and the properties and methods that they expose. An object need not expose all of its properties and methods in a given interface. Thus, a recordset object would expose only read methods in a query interface, but would expose create, update, and delete methods in a modification interface.

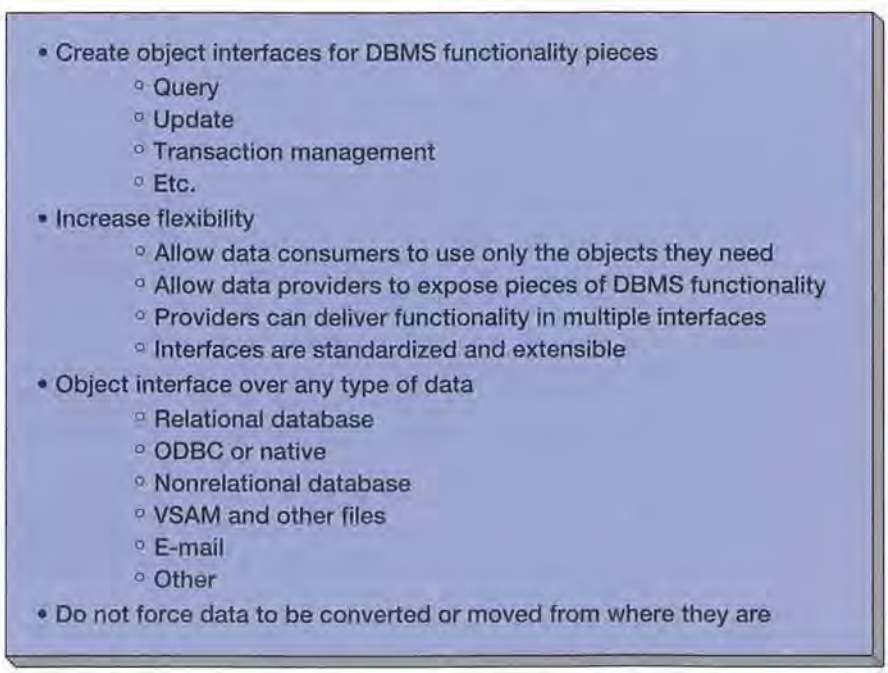
How the object supports the interface, or the **implementation**, is completely hidden from the user. In fact, the developers of an object are free to change the implementation whenever they want. Who will know? But they may not ever change the interface without incurring the justifiable disdain of their users!

OLE DB defines standardized interfaces. Data providers, however, are free to add interfaces on top of the basic standards. Such extensibility is essential for the next goal, which is to provide an object interface to any data type. Relational databases can be processed through OLE DB objects that use ODBC or that use the native DBMS drivers. OLE DB includes support for the other types as indicated.

The net result of these design goals is that data need not be converted from one form to another, nor need they be moved from one data source to another. The Web server shown in

Figure 11-12

The Goals of OLE DB



Amazon Ex. 1008.0464

Figure 11-11 can utilize OLE DB to process data in any of the formats, right where the data reside. This means that transactions may span multiple data sources and may be distributed on different computers. The OLE DB provision for this is the **Microsoft Transaction Manager (MTS)**; however, discussion of the MTS is beyond the scope of this text.

OLE DB Terminology

As shown in Figure 11-13, OLE DB has two types of data providers. **Tabular data providers** present their data via rowsets. Examples are DBMS products, spreadsheets, and ISAM file processors, such as dBase and FoxPro. Additionally, other types of data, such as e-mail, can also be presented in rowsets. Tabular data providers bring data of some type into the OLE DB world.

A **service provider**, in contrast, is a transformer of data. Service providers accept OLE DB data from an OLE DB tabular data provider and transform it in some way. Service providers are both consumers and providers of transformed data. An example of a service provider is one that obtains data from a relational DBMS and then transforms them into XML documents. Both data and service providers process rowset objects. A rowset is equivalent to what we called a **cursor** in Chapter 9, and in fact the two terms are frequently used synonymously.

For database applications, rowsets are created by processing SQL statements. The results of a query, for example, are stored in a rowset. OLE DB rowsets have dozens of different methods, which are exposed via the interfaces listed in Figure 11-14.

IRowSet provides object methods for forward-only sequential movement through a rowset. When you declare a forward-only cursor in OLE DB, you are invoking the IRowSet interface. The IAccessor interface is used to bind program variables to rowset fields.

The IColumnsInfo interface has methods for obtaining information about the columns in a rowset. IRowSet, IAccessor, and IColumnsInfo are the basic rowset interfaces. Other interfaces are defined for more advanced operations such as scrollable cursors, update operations, direct access to particular rows, explicit locks, and so forth.

Figure 11-13

Two Types of OLE DB Data Providers

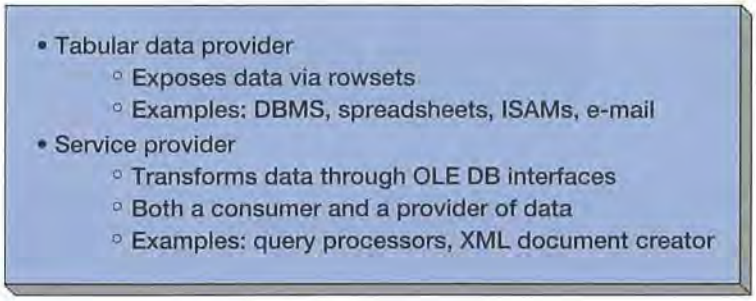
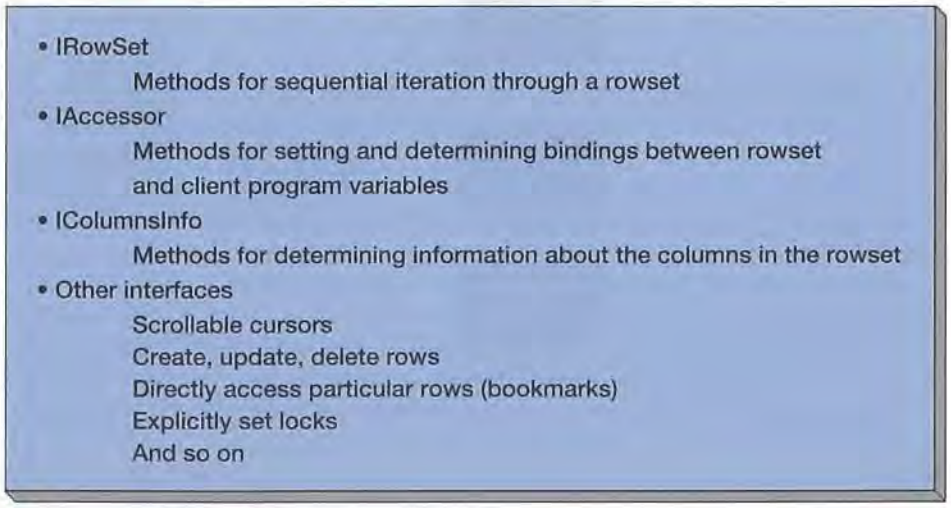


Figure 11-14

Rowset Interfaces



Amazon Ex. 1008.0465

ADO and ADO.NET

Because OLE DB is an object-oriented interface, it is particularly suited to object-oriented languages such as VB.NET and Visual C#.NET. Many database application developers, however, program in scripting languages such as VBScript or JScript (Microsoft's version of JavaScript). To meet the needs of these programmers, Microsoft developed **Active Data Objects (ADO)** as a cover over OLE DB objects, as shown in Figure 11-15. ADO enabled programmers to use almost any language to access OLE DB functionality.

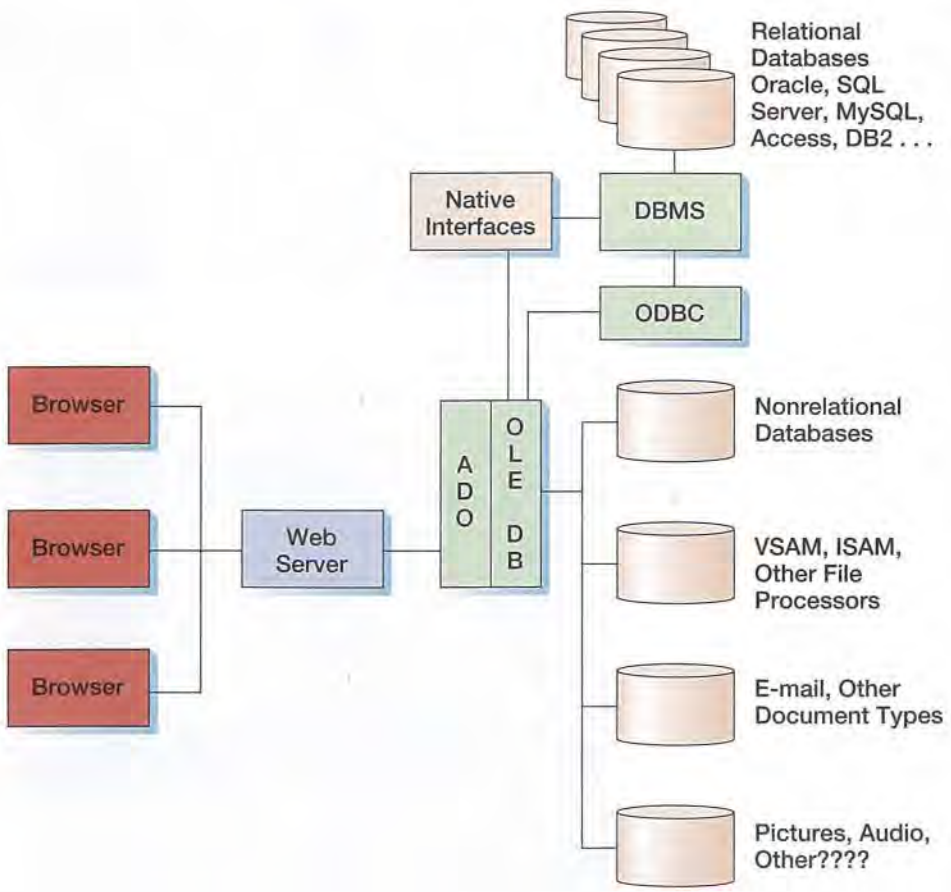
ADO is a simple object model that overlies the more complex OLE DB object model. ADO can be called from scripting languages, such as JScript and VBScript, and it can also be called from more powerful languages, such as Visual Basic .NET, Visual C#.NET, Visual C++.NET, and even Java. Because ADO is easier to understand and use than OLE DB, ADO was (and still is) often used for database applications.

ADO.NET is a new, improved, and greatly expanded version of ADO that was developed as part of Microsoft's .NET initiative. It incorporates the functionality of ADO and OLE DB, but adds much more. In particular, ADO.NET facilitates the transformation of XML documents (discussed in Chapter 12) to and from relational database constructs. ADO.NET also provides the ability to create and process in-memory databases called *datasets*. Figure 11-16 shows the role of ADO.NET.

The ADO.NET Object Model

Now we need to look at ADO.NET in more detail. As shown in Figure 11-17, an **ADO.NET Data Provider** is a class library that provides ADO.NET services. There are .NET Framework Data Providers for ODBC, OLE DB, SQL Server, and EDM applications, which means that ADO.NET works with not only the ODBC and OLE DB data access methods we've discussed in this chapter, but directly with SQL Server and .NET language applications that use EDM as well.

Figure 11-15
The Role of ADO



Amazon Ex. 1008.0466

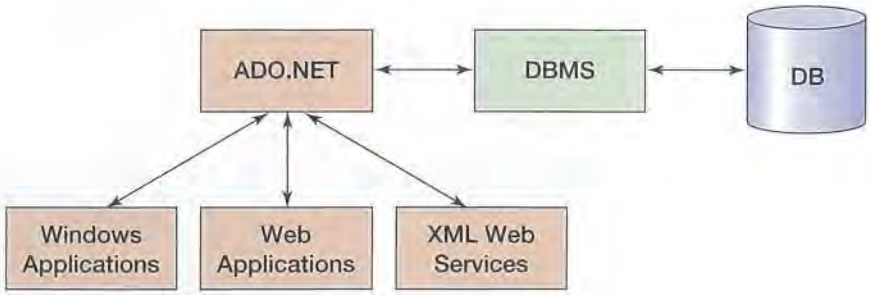


Figure 11-16
The Role of ADO.NET

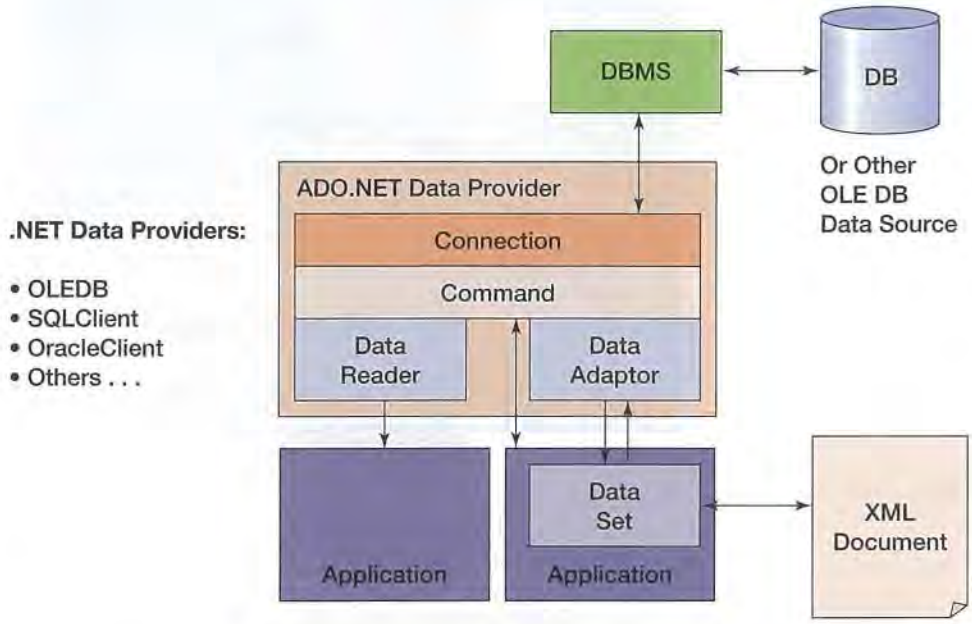


Figure 11-17
Components of an ADO.NET Data Provider

A simplified version of the ADO.NET object model is shown in Figure 11-18. The ADO.NET object classes are grouped into Data Providers and Data Sets.

The **ADO.NET Connection object** is responsible for connecting to the data source. It is basically the same as the ADO Connection object, except that ODBC is not used as a data source.

The **ADO.NET DataSet** is a representation of the data stored in the computer memory as a set of data separate from the one in the DBMS. The DataSet is distinct and disconnected from the DBMS data. This allows commands to be run against the DataSet instead of the actual data. DataSet data can be constructed from data in multiple databases, and they can be managed by different DBMS products. The DataSet contains the DataTableCollection and the DataRelationCollection. A more detailed version of the ADO.NET dataset object model is shown in Figure 11-19.

The **DataTableCollection** mimics DBMS tables with **DataTable objects**. DataTable objects include a **DataColumnsCollection**, a **DataRowCollection**, and **Constraints**. Data values are stored in DataRow collections in three forms: **original values**, **current values**, and **proposed values**. Each DataTable object has a **PrimaryKey property** to enforce row uniqueness. The Constraints collection uses two constraints. The **ForeignKeyConstraint** supports referential integrity, and the **UniqueConstraint** supports data integrity.

The **DataRelationCollection** stores **DataRelations**, which act as the relational links between tables. Note again that referential integrity is maintained by the ForeignKeyConstraint in the Constraints collection. Relationships among DataSet tables can be processed just as relationships in a database can be processed. A relationship can be used to compute the values of a column, and DataSet tables can also have views.

Amazon Ex. 1008.0467

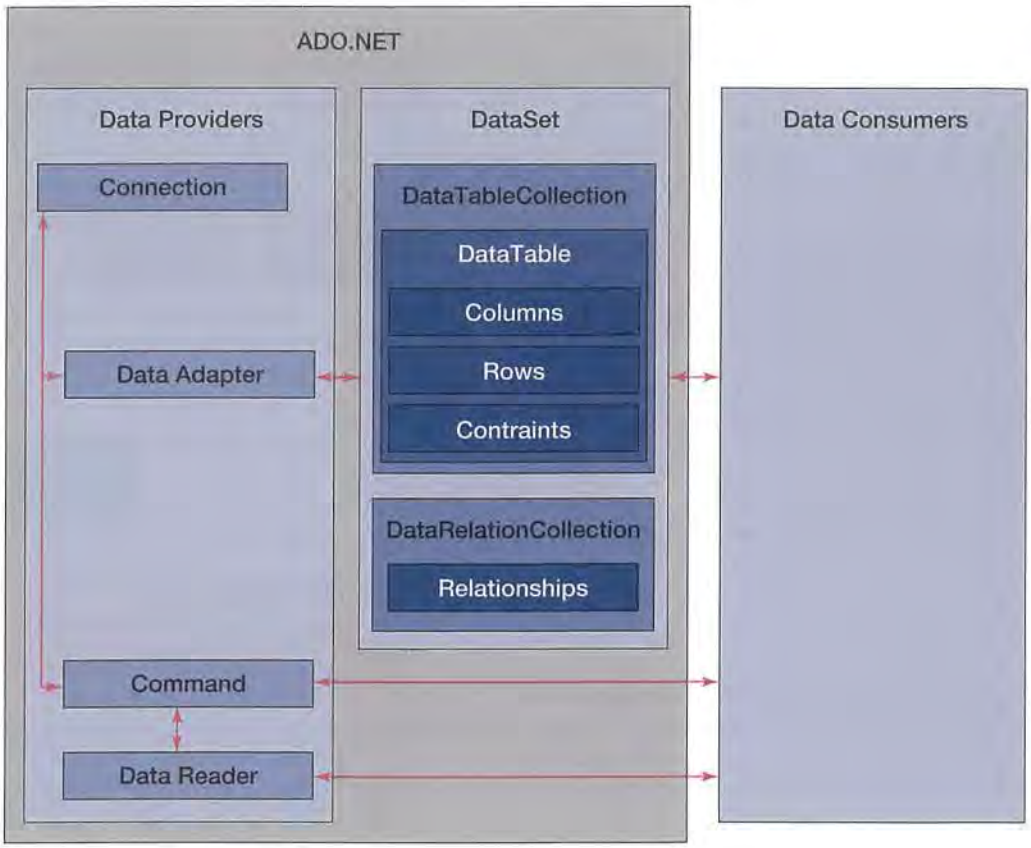


Figure 11-18
The ADO.NET Object Model

The **ADO.NET Command object** shown in Figures 11-17 and 11-18 is used as an SQL statement or stored procedure and is run on data in the DataSet. The **ADO.NET DataAdapter object** is the link between a Connection object and a DataSet object. The DataAdapter uses four Command objects: the **SelectCommand object**, the **InsertCommand object**, the **UpdateCommand object**, and the **DeleteCommand object**. The SelectCommand object gets data from a DBMS and places it in a DataSet. The other commands send changes in the DataSet back to the DBMS data.

The **ADO.NET DataReader** is similar to a cursor that provides read-only, forward-only data transfers from a data source, and can only be used through an Execute method of a Command.

Looking ahead to Chapter 12 on XML, we see some of the advantages of ADO.NET over ADO. Once a DataSet is constructed, its contents can be formatted as an XML document with a single command. Similarly, an XML Schema document for the DataSet can also be produced with a single command. This process works in reverse as well. An XML Schema document can be used to create the structure of a DataSet, and the DataSet data can then be filled by reading an XML document.

BY THE WAY You may be wondering, "Why is all of this necessary? Why do we need an in-memory database?" The answer lies in database views like that shown in Chapter 12 in Figure 12-14. There is no standardized way to describe and process such data structures. Because it involves two multivalue paths through the data, SQL cannot be used to describe the data. Instead, we must execute two SQL statements and somehow patch the results to obtain the view.

Views like that shown in Figure 12-14 have been processed for many years, but only by private, proprietary means. Every time such a structure needs to be processed, a developer designs programs for creating and manipulating the data in memory and for

Amazon Ex. 1008.0468

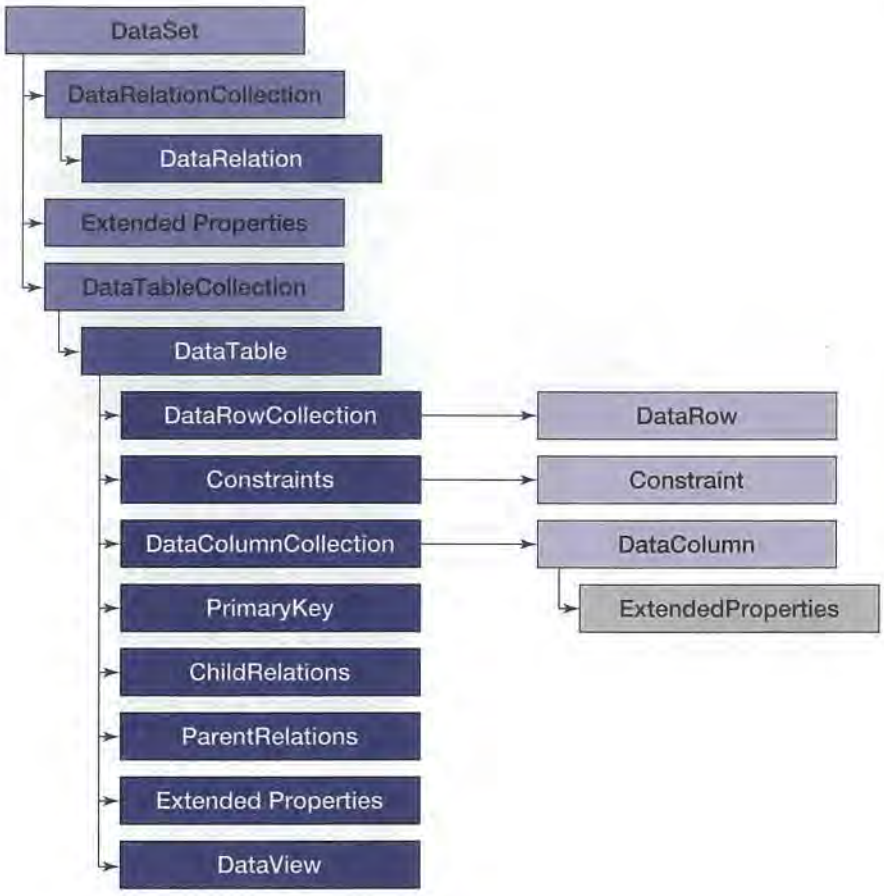


Figure 11-19

The ADO.NET DataSet Object Model

saving them to the database. Object-oriented programmers define a class for this data structure and create methods to serialize objects of this class into the database. Other programmers use other means. The problem is that every time a different view is designed, a different scheme must be designed and developed to process the new view.

As Microsoft developed .NET technology, it became clear that a generalized means was needed to define and process database views and related structures. Microsoft could have defined a new proprietary technology for this purpose, but thankfully it did not. Instead, it recognized that the concepts, techniques, and facilities used to manage regular databases can be used to manage in-memory databases as well. The benefit to you is that all of the concepts and techniques that you have learned to this point for processing regular databases can also be used to process datasets.

DataSets do have a downside, and a serious one for some applications. Because DataSet data are disconnected from the regular database, only optimistic locking can be used. The data are read from the database, placed into the DataSet, and processed there. No attempt is made to propagate changes in the DataSet back to the database. If, after processing, the application later wants to save all of the DataSet data into a regular database, it needs to use optimistic locking. If some other application has changed the data, either the DataSet will need to be reprocessed or the data change will be forced onto the database, causing the lost update problem.

Thus, DataSets cannot be used for applications in which optimistic locking is problematic. For such applications, the ADO.NET Command object should be used instead. But for applications in which conflict is rare or for those in which reprocessing after conflict can be accommodated, DataSets provide significant value.

Amazon Ex. 1008.0469

BY THE WAY ASP.NET connects to the DBMS using the default user name ASP.NET. Oracle requires any user name that has operating system authentication to begin with the letters OPS$. Thus, for this application to run, there must be an account named OPS$ASPNET in the Oracle VRG database, and that account must have privileges necessary to read and update the View Ridge tables in VRG. If you do not want to use the default name ASPNET, you can cause ASP.NET to pass the user's credentials through to Oracle. That technique, however, is beyond the scope of this discussion.

BY THE WAY The only way to use Oracle's XML facilities is to write in Java, an object-oriented programming language. Further, the only way to process ADO.NET is from one of the .NET languages, all of which, like Visual Basic .NET, are object-oriented languages. Thus, if you do not yet know object-oriented design and programming, and if you want to work in the emerging world of database processing, you should run, not walk, to your nearest object-oriented design and programming class!

The Java Platform

Having looked at the Microsoft .NET Framework in some detail, we will now turn our attention to the Java platform and look at its components.

JDBC

Originally, and contrary to many sources, JDBC originally did *not* stand for Java Database Connectivity. According to Sun Microsystems—the inventor of Java and the source of many Java-oriented products—JDBC was not an acronym; it just stood for JDBC. At this point in time, however, we can even find the name Java Database Connectivity (JBDC) on Sun's Web site (see *http://java.sun.com/javase/technologies/database/*)! Still, because we use acronyms in this book after introducing the full term, we will use *JDBC*.

A JDBC driver is available for almost every conceivable DBMS product. Sun maintains a directory of them available through *http://java.sun.com/products/jdbc/overview.html*). Some of the drivers are free, and almost all of them have an evaluation edition that can be used for free for a limited period of time. The JDBC driver for MySQL is the MySQL Connector/J, which is available at *http://dev.mysql.com/downloads/connector/j/5.1.html*.

Driver Types

As summarized in Figure 11-20, Sun defines four JDBC driver types. Type 1 drivers are JDBC–ODBC bridge drivers, which provide an interface between Java and regular ODBC drivers. Most ODBC drivers are written in C or C++. For reasons unimportant to us here, there are incompatibilities between Java and C/C++. Bridge drivers resolve these incompatibilities and allow access to ODBC data sources from Java. Because we use ODBC in the chapter, if you are using MySQL you will want to download the MySQL Connector/ODBC driver from *http://dev.mysql.com/downloads/connector/odbc/5.1.html*.

Drivers of Types 2 through 4 are written entirely in Java; they differ only in how they connect to the DBMS. Type 2 drivers connect to the native API of the DBMS. For example, they call Oracle using the standard (non-ODBC) programming interface to Oracle. Drivers of Types 3 and 4 are intended for use over communications networks. A Type 3 driver translates JDBC calls into a DBMS-independent network protocol. This protocol is then translated into the network protocol used by a particular DBMS. Finally, Type 4 drivers translate JDBC calls into DBMS-specific network protocols.

To understand how drivers Types 2 through 4 differ, you must first understand the difference between a *servlet* and an *applet*. As you probably know, Java was designed to be portable. To accomplish portability, Java programs are not compiled into a particular machine language,

Amazon Ex. 1008.0470

Driver Type	Characteristics
1	JDBC–ODBC bridge. Provides a Java API that interfaces to an ODBC driver. Enables processing of ODBC data sources from Java.
2	A Java API that connects to the native-library of a DBMS product. The Java program and the DBMS must reside on the same machine, or the DBMS must handle the intermachine communication, if not.
3	A Java API that connects to a DBMS-independent network protocol. Can be used for servlets and applets.
4	A Java API that connects to a DBMS-dependent network protocol. Can be used for servlets and applets.

Figure 11-20
Summary of JDBC Driver Types

but instead are compiled into machine-independent bytecode. Sun, Microsoft, and others have written **bytecode interpreters** for each machine environment (Intel 386, Alpha, and so on). These interpreters are referred to as **Java virtual machines**.

To run a compiled Java program, the machine-independent bytecode is interpreted by the virtual machine at run time. The cost of this, of course, is that bytecode interpretation constitutes an extra step, so such programs can never be as fast as programs that are compiled directly into machine code. This may or may not be a problem, depending on the application's workload.

An **applet** is a Java bytecode program that runs on the application user's computer. Applet bytecode is sent to the user via HTTP and is invoked using the HTTP protocol on the user's computer. The bytecode is interpreted by a virtual machine, which is usually part of the browser. Because of portability, the same bytecode can be sent to a Windows, a UNIX, or an Apple computer.

A **servlet** is a Java program that is invoked via HTTP on the Web server computer. It responds to requests from browsers. Servlets are interpreted and executed by a Java virtual machine running on the server.

Because they have a connection to a communications protocol, Type 3 and Type 4 drivers can be used in either applet or servlet code. Type 2 drivers can be used only in situations where the Java program and the DBMS reside on the same machine or where the Type 2 driver connects to a DBMS program that handles the communications between the computer running the Java program and the computer running the DBMS.

Thus, if you write code that connects to a database from an applet (two-tier architecture), only a Type 3 or Type 4 driver can be used. In these situations, if your DBMS product has a Type 4 driver, use it; it will be faster than a Type 3 driver.

In three-tier or *n*-tier architecture, if the Web server and the DBMS are running on the same machine, you can use any of the four types of drivers. If the Web server and the DBMS are running on different machines, Type 3 and Type 4 drivers can be used without a problem. Type 2 drivers can also be used if the DBMS vendor handles the communications between the Web server and the DBMS. The MySQL Connector/J, the Java connector that you can download from *http://dev.mysql.com/downloads/connector/j/5.1.html* is a Type 4 driver.

Using JDBC

Unlike ODBC, JDBC does not have a separate utility for creating a JDBC data source. Instead, all of the work to define a connection is done in Java code via the JDBC driver. The coding pattern for using a JDBC driver is as follows:

1. Load the driver.
2. Establish a connection to the database.
3. Create a statement.
4. Do something with the statement.

Amazon Ex. 1008.0471

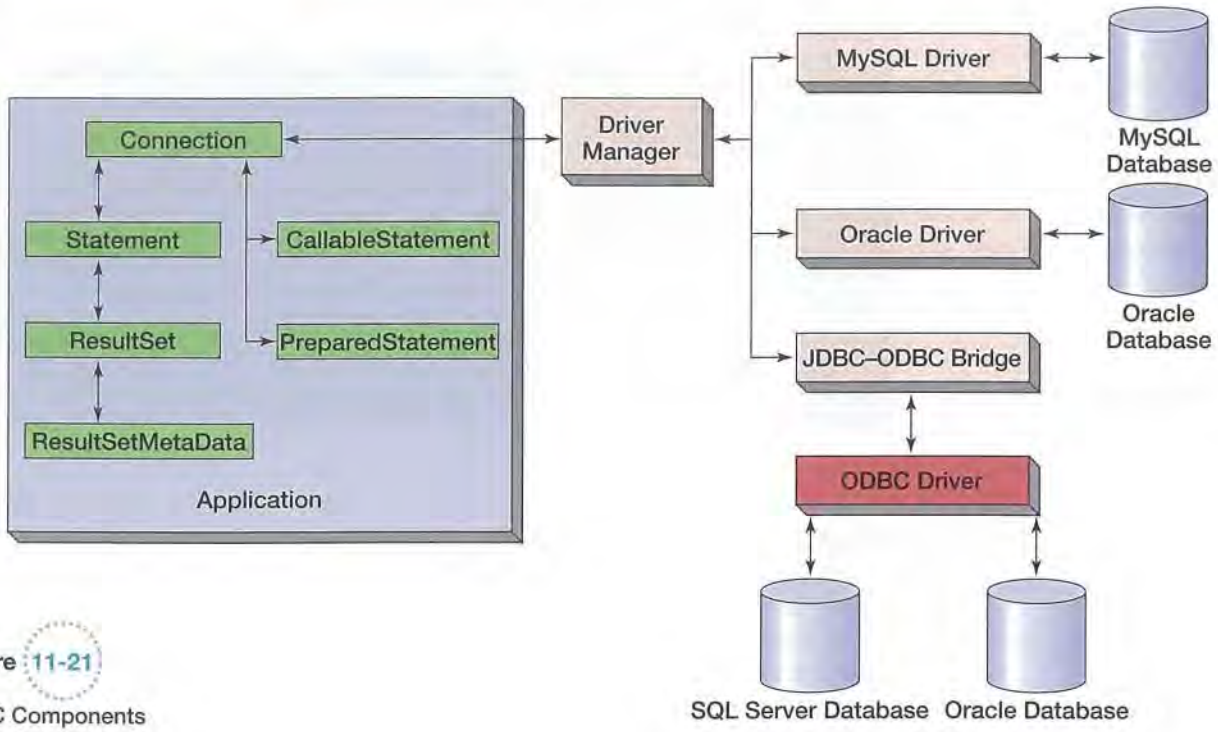


Figure 11-21
JDBC Components

To load the driver, you must first obtain the driver library and install it in a directory. You need to ensure that the directory is named in the CLASSPATH for both the Java compiler and for the Java virtual machine. The name of the DBMS product to be used and the name of the database are provided at step 2. Figure 11-21 summarizes the JDBC components.

Note that Java is used to create the application shown in the figure, and because Java is an object-oriented programming language, we see a set of objects in the application that are similar to those we have discussed for ADO.NET. The application creates a **JDBC Connection object**, **JDBC Statement objects**, a **JDBC ResultSet object**, and a **JDBC ResultSetMetaData object**. Calls from these objects are routed via the **JDBC Driver Manager** to the proper driver. Drivers then process their databases. Notice that the Oracle database in this figure could be processed via either a JDBC–ODBC bridge or via a pure JDBC driver.

BY THE WAY By the way, most of this technology arose in the UNIX world. UNIX is case sensitive, and almost everything you enter here also is case sensitive. Thus, *jdbc* and *JDBC* are *not* the same.

Prepared Statement objects and **Callable Statement objects** can be used to invoke compiled queries and stored procedures in the database. Their use is similar to the use of ADO.NET Command objects discussed previously in this chapter. It is possible to receive values back from procedures as well. Start at *http://java.sun.com/javase/reference/index.jsp* for more information.

Java Server Pages (JSP) and Servlets

Java Server Pages (JSP) technology provides a means to create dynamic Web pages using HTML (and XML) and the Java programming language. With Java, the capabilities of a complete

Amazon Ex. 1008.0472

object-oriented language are directly available to the Web page developer. This is similar to what can be done using ASP.NET using the Microsoft .NET languages.

Because Java is machine independent, JSP is also machine independent. With JSP, you are not locked into using Windows and IIS. You can run the same JSP page on a Linux server, on a Windows server, and on others as well. The official specification for JSP can be found at *http://java.sun.com/products/jsp/*.

JSP pages are transformed into standard Java language and then compiled just like a regular program. In particular, they are transformed into Java servlets, which means that JSP pages are transformed into subclasses of the HTTPServlet class behind the scenes. JSP code thus has access to the HTTP request and response objects and also to their methods and to other HTTP functionality.

Apache Tomcat

The Apache Web server does not support servlets. However, the Apache Foundation and Sun cosponsored the Jakarta Project that developed a servlet processor named **Apache Tomcat** (now in version 6). You can obtain the source and binary code of Tomcat from the Apache Tomcat Web site at *http://tomcat.apache.org/*.

Tomcat is a servlet processor that can work in conjunction with Apache or as a stand-alone Web server. Tomcat has limited Web server facilities, however; so it is normally used in stand-alone mode only for testing servlets and JSP pages. For commercial production applications, Tomcat should be used in conjunction with Apache. If you are running Tomcat and Apache separately on the same Web server, they need to use different ports. The default port for a Web server is 80, and Apache normally uses it. When used in stand-alone mode, Tomcat is usually configured to listen to port 8080, though this, of course, can be changed.

Figure 11-22 shows the process by which JSP pages are compiled. When a request for a JSP page is received, a Tomcat (or other) servlet processor finds the compiled version of the page and checks to determine whether it is current. It does this by looking for an uncompiled version of the page having a creation date and time later than the compiled page's creation date and time. If the page is not current, the new page is parsed and transformed into a Java source file, and that source file is then compiled. The servlet is then loaded and executed. If the compiled JSP page is current, then it is loaded into memory, if not already there, and then executed. If it is in memory, it is simply executed.



The downside of such automatic compilation is that if you make syntax errors and forget to test your pages, the first user to access your page will receive the compiler errors!

Unlike CGI files and some other Web server programs, only one copy of a JSP page can be in memory at a time. Further, pages are executed by one of Tomcat's threads, not by an independent process. This means that much less memory and processor time are required to execute a JSP page than to execute a comparable CGI script.

Web Database Processing with PHP

At this point in our discussion, it is time to build an actual Web database application and apply both some of the knowledge from this chapter and some new techniques yet to be discussed. We have already created an ODBC data source for the View Ridge Gallery database, and now we will use it to look at Web database processing. Although we have introduced technologies such as ADO.NET, ASP.NET, Java, and JSP, these technologies are complex subjects and beyond the scope of this book. Further, these technologies tend to become vendor specific—you're either working

Amazon Ex. 1008.0473

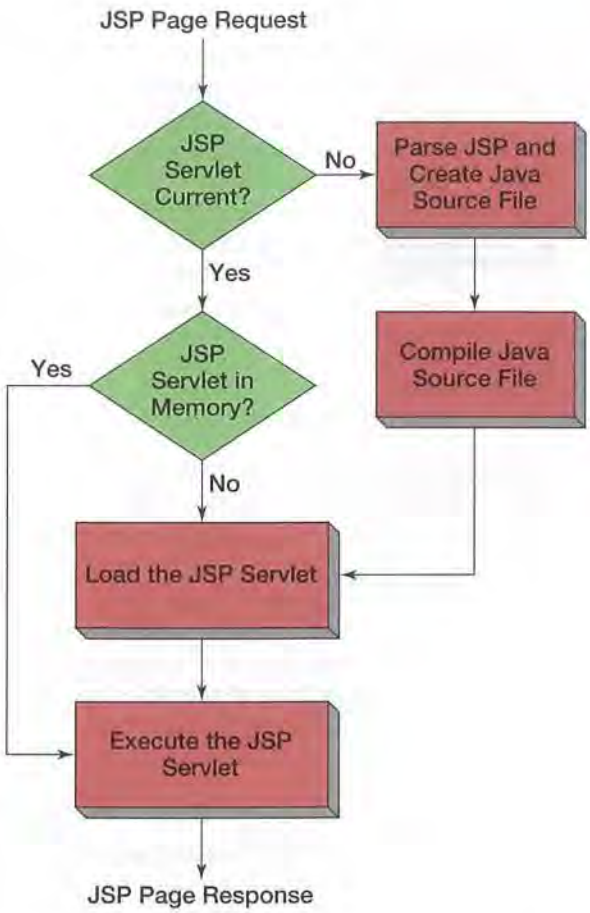


Figure 11-22
JSP Compilation Process

in a Microsoft-centric world with .NET technologies and ASPs , or a Sun Microsystems–centric world with Java and JSPs.

In this book, we will take a vendor-neutral approach and use technologies that can be used with any operating system or DBMS. We will use the **PHP** language. PHP, which is an abbreviation for **PHP: Hypertext Processor** (and that was previously known as the *Personal Hypertext Processor*), is a scripting language that can be embedded in Web pages. Although PHP started as purely a scripting language, it now also has object-oriented programming elements, but we will not cover those in this book.

PHP is extremely popular. In the summer of 2007, more than 2 million Internet domains had servers running PHP,[2] and the March 2008 TIOBE Programming Community Index ranked PHP as the fourth most popular programming language (following, in order, Java, C, and Visual Basic).[3] PHP is easy to learn and can be used in most Web server environments and with most databases. As an added bonus, it is an open-source product available for free download from the PHP Web site (*www.php.net*).

While Microsoft would probably prefer that you use ASP.NET for Web applications, there is still good information on using PHP in a Microsoft environment on the Microsoft Web site (for example, see *PHP on IIS7* at php.iis.net). Oracle and MySQL enthusiastically support PHP. Oracle publishes the *Oracle Database 2 Day + PHP Developer's Guide* (available in both HTML and PDF format at *www.oracle.com/pls/db111/homepage*), which is an excellent reference for using PHP with Oracle Database 11g. Since PHP is often the P in AMP, LAMP and WAMP, many books are available discussing the combination of PHP and MySQL, and the MySQL Web site contains basic documentation on using PHP with MySQL (for example, see *dev.mysql.com/doc/refman/5.1/en/apis-php.html*).

[2] See *www.php.net/usage.php*.
[3] See *www.tiobe.com/index.php/content/paperinfo/tpci/index.html*.

Amazon Ex. 1008.0474

Web Database Processing with PHP and Eclipse

To start, we need a Web server to store the Web pages that we will build and use. We could use the Apache HTTP Server (available from the Apache Software Foundation at *www.apache.org*). This is the most widely used Web server, and there is a version that will run on just about every operating system in existence. However, because we have been using the Windows operating system for the DBMS products shown in this book, we will build a Web site using the Microsoft IIS Web server. One advantage of using this Web server for users of the Windows XP Professional and the Windows Vista operating systems is that IIS is included with the operating system: IIS version 5.1 is included with Windows XP, and IIS version 7.0 is included with Vista. IIS is not installed by default but can easily be installed at any time. This means that any user can practice creating and using Web pages on his or her own workstation.

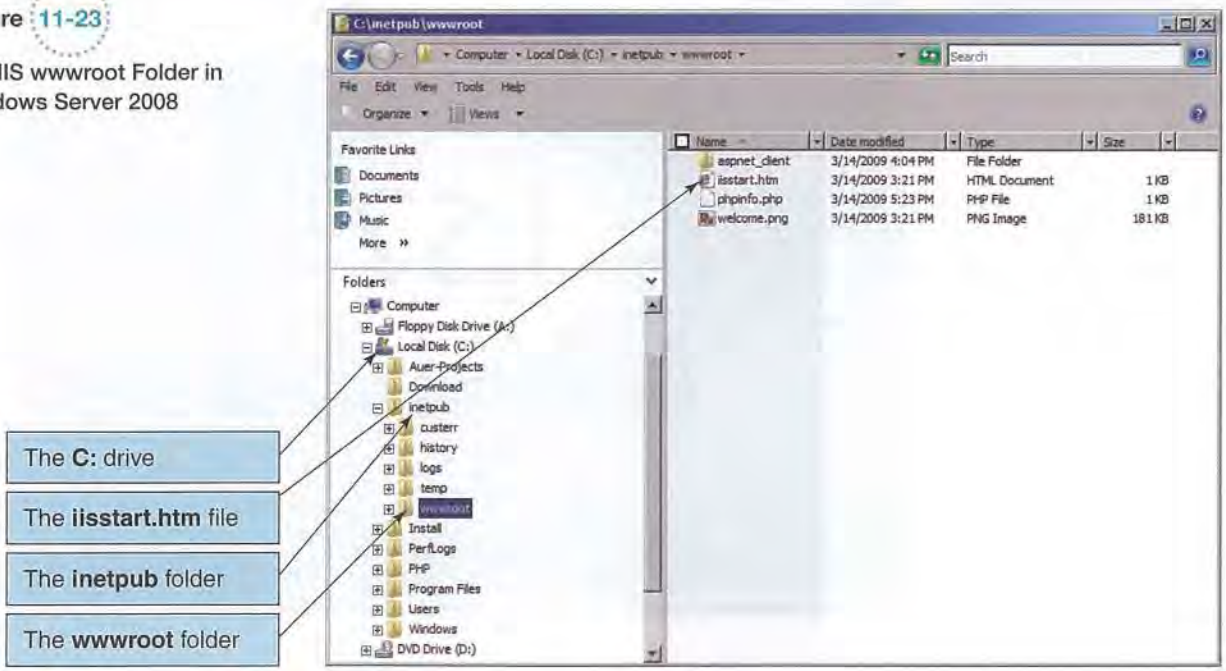
> **BY THE WAY** This discussion of Web database processing has been written to be as widely applicable as possible. With minor adjustments to the following steps, you should be able to use the Apache Web server if you have it available. Whenever possible, we have chosen to use products and technologies that are available for many operating systems.

When IIS is installed, it creates an **inetpub folder** on the C: drive as C:\inetpub. Within the inetpub folder is the **wwwroot folder**, which is where IIS stores the most basic Web pages used by the Web server. Figure 11-22 shows this directory structure in Windows Server 2008 after IIS has been installed, with the files in the wwwroot folder displayed in the file pane.

IIS is managed using a program called **Internet Information Services Manager** in Windows Vista and Server 2008, as shown in Figure 11-24. The location of the program icon varies depending on the operating system. In Windows Vista, open **Control Panel** and then open **Administrative Tools**. The shortcut icon for Internet Information Services Manager is located in Administrative Tools. For Windows Server 2008, use **Start | Administrative Tools | Internet Information Services (IIS) Manager**.

Figure 11-23

The IIS wwwroot Folder in Windows Server 2008



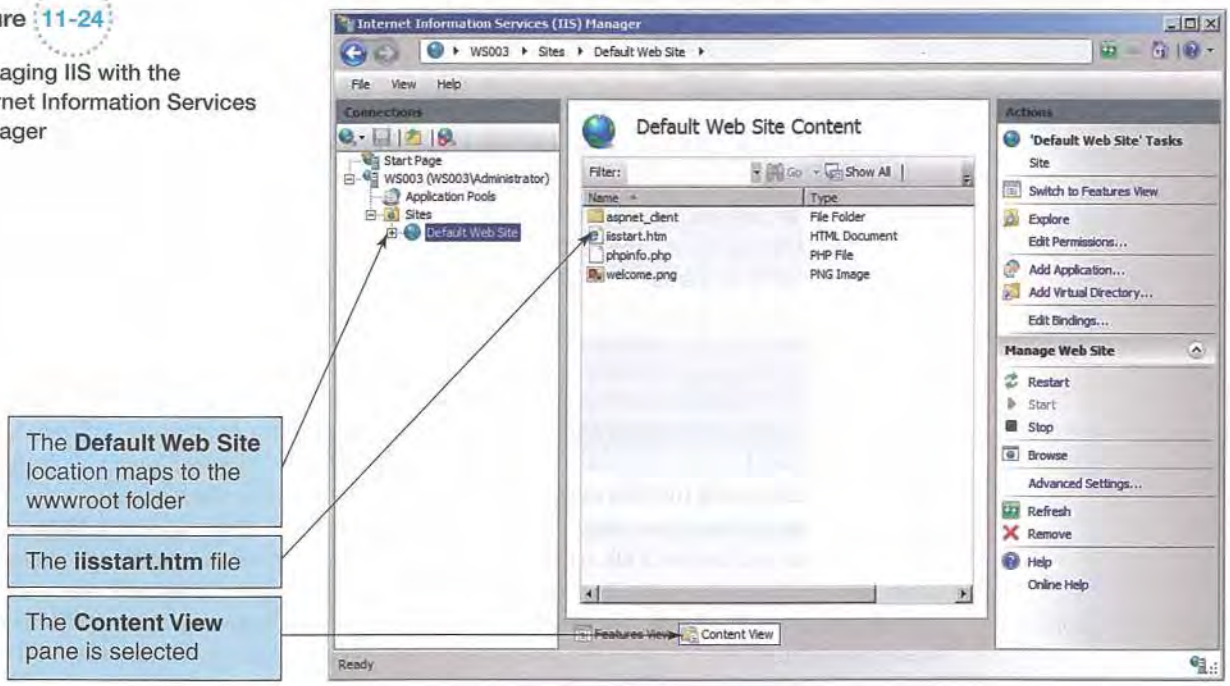
Amazon Ex. 1008.0475

Note that the files shown in the **Default Web Site folder** in Figure 11-23 are the same files that are in the wwwroot folder in Figure 11-24—they are the default files created by IIS when it is installed. In Windows Vista and Windows Server 2008, the file **iisstart.htm** generates the Web page that Internet Explorer (or any other Web browser) contacting this Web server over the Internet will see displayed.

To test the Web server installation, open your Web browser, type in the URL **http://localhost**, and press the **Enter** key. For Windows Server 2008, the Web page shown in Figure 11-25 appears. If the appropriate Web page isn't displayed in your Web browser, your Web server is not properly installed.

Figure 11-24

Managing IIS with the Internet Information Services Manager



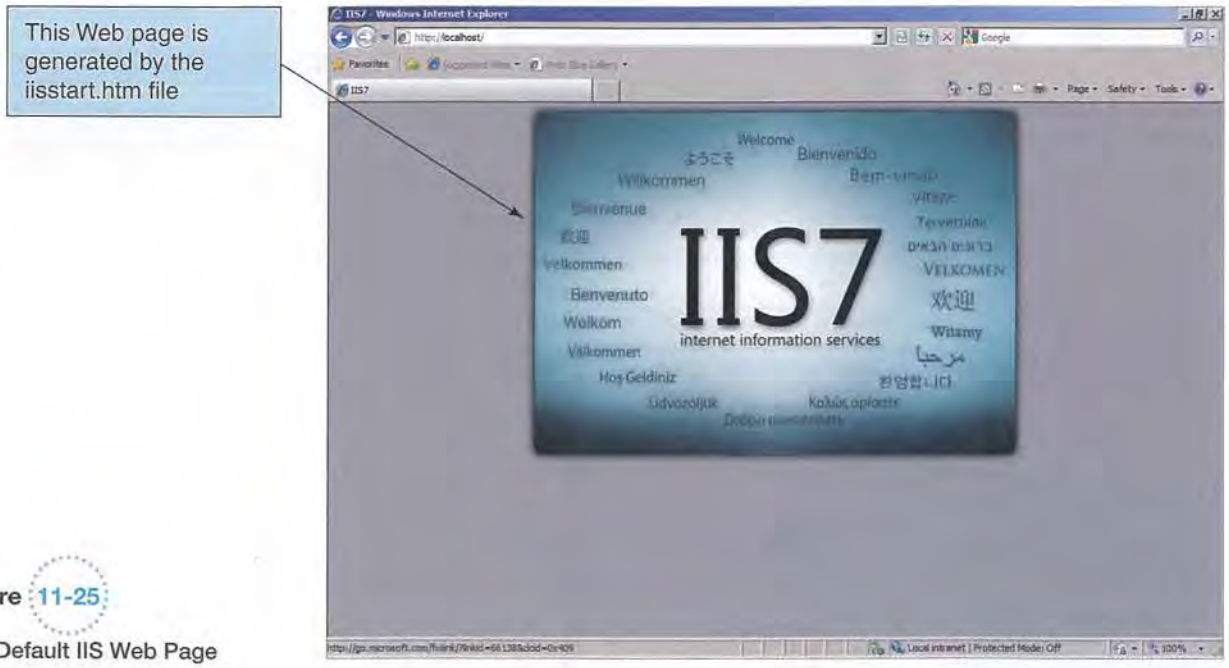


Figure 11-25

The Default IIS Web Page

Amazon Ex. 1008.0476

Now we will set up a small Web site that can be used for Web database processing of the View Ridge Gallery VRG database. First, we will create a new folder named DBP (Database Processing) under the wwwroot folder. This new folder will be used to hold all the Web pages developed in discussions and exercises in this book. Second, we will create a subfolder of DBC named VRG. This folder will hold the VRG Web site. You create these folders using Windows Explorer.

Getting Started with HTML Web Pages

The most basic Web pages are created using **Hypertext Markup Language (HTML)**. The term *hypertext* refers to the fact that you can include links to other objects, such as Web pages, maps, pictures, and even audio and video files in a Web page, and when you click the link, you are immediately taken to that other object, and it is displayed in your Web browser. HTML itself is a standard set of **HTML syntax rules** and **HTML document tags** that can be interpreted by Web browsers to create specific onscreen displays.

Tags are usually paired, with a specific beginning tag and a matching ending tag that includes the backslash character (/). Thus, a paragraph of text is tagged as {*paragraph text here*} , and a main heading is tagged as <h1>{*heading text here*}</h1>. Some tags do not need a separate end tag because they are essentially self-contained. For example, to insert a horizontal line on a Web page, you use the horizontal rule tag, <hr />. Note that such single, self-contained tags must include the backslash character as part of the tag.

The rules of HTML are defined as standards by the **World Wide Web Consortium (W3C)**, and the details of current and proposed standards can be found at *www.w3c.org* (this site also has several excellent tutorials on HTML[4]). The W3C Web site has current standards for HTML; **Extensible Markup Language (XML)**, which we will discuss in Chapter 14; and a hybrid of the two called **XHTML**. A full discussion of these standards is beyond the scope of this text; this chapter uses the current HTML 4.01 standard (usually in what is called the "strict" form).

In this chapter, we will create a simple HTML homepage for the View Ridge Gallery Web site and place it in the VRG folder. We will discuss some of the numerous available Web page editors shortly, but all you really need to create Web pages is a simple text editor. For this first Web page, we will use the Microsoft Notepad ASCII text editor, which has the advantage of being supplied with every version of the Windows operating system.

The index.html Web Page

The name for the file we are going to create is **index.html**. We need to use the name *index.html* because it is a special name as far as Web servers are concerned. The file name index.html is one of only a few file names that *most* Web servers automatically display when a URL request is made without a specific file reference, and thus it will become the new default display page for our Web database application. However, note the phrase "*most* Web servers" in the last sentence. Although Apache and IIS 7.0 (as shown in Figure 11-26) are configured to recognize index.html, IIS 5.1 is not. If you are using Windows XP and IIS 5.1, you need to add index.html to the list of recognized files using the Internet Information Services management program.

Creating the index.html Web Page

Now we can create the index.html Web page, which consists of the basic HTML statements shown in Figure 11-27. Figure 11-28 shows the HTML code in Notepad.

[4] To learn more about HTML, go to the Web site of the World Wide Web Consortium (W3C) at *www.w3.org*. For good HTML tutorials, see David Raggett's "Getting Started with HTML" tutorial at *www.w3.org/MarkUp/Guide*, his "More Advanced Features" tutorial at *www.w3.org/MarkUp/Guide/Advanced.html*, and his "Adding a Touch of Style" tutorial at *www.w3.org/MarkUp/Guide/Style.html*.

Amazon Ex. 1008.0477

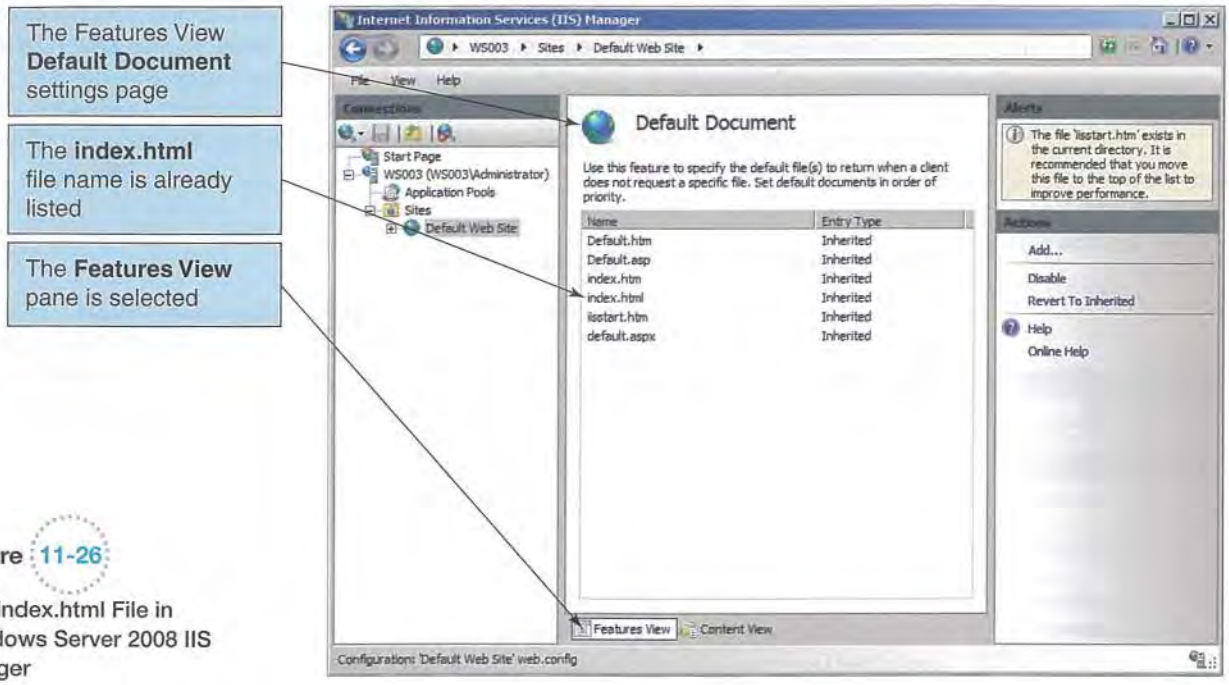


Figure 11-26

The index.html File in Windows Server 2008 IIS Manger

Figure 11-27

The HTML Code for the index.html File in the VRG Folder

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
            <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
            <title>View Ridge Gallery Demonstration Pages Home Page</title>
      </head>
      <body>
            <h1 style="text-align: center; color: blue">
                  Database Processing (11th Edition)
            </h1>
            <h2 style="text-align: center; font-weight: bold">
                  David M. Kroenke
            </h2>
            <h2 style="text-align: center; font-weight: bold">
                  David J. Auer
            </h2>
            <hr />
            <h2 style="text-align: center; color: blue">
                  Welcome to the View Ridge Gallery Home Page
            </h2>
            <hr />
            <p>Chapter 11 Demonstration Pages From Figures in the Text:</p>
            <p>Example 1:   
                  <a href="ReadArtist.php">
                        Display the ARTIST Table (LastName, FirstName, Nationality)
                  </a>
            </p>
            <hr />
      </body>
</html>
```

Amazon Ex. 1008.0478

The **index.html** HTML code—note how indentation is used to keep the code organized and readable

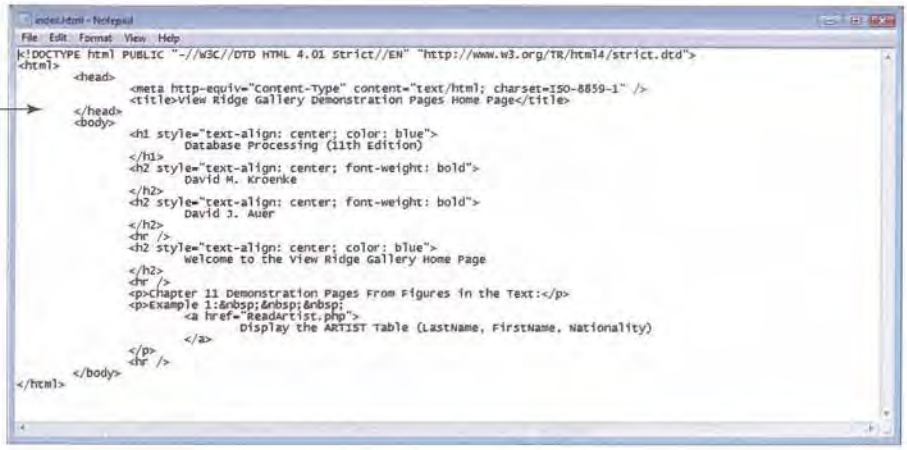


Figure 11-28

The HTML Code for the index.html File in Notepad

BY THE WAY In the HTML code for index.html, the HTML code segment:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

is an HTML/XML **document type declaration (DTD)**, which is used to check and validate the contents of the code that you write. DTDs are discussed later in this chapter. For now, just include the code as it is written.

If we now use either the URL *http://localhost/DBC/VRG* (if the Web server is on the same computer we're working on) or the URL http://{*Web server DNS Name or IP Number*}/DBC/WPC (if the Web server is on another computer), we get the Web page shown in Figure 11-29.

This IP number indicates that the Web server is on a separate computer—if the Web server is on your computer itself, use the URL *http://localhost/DBP/VRG*

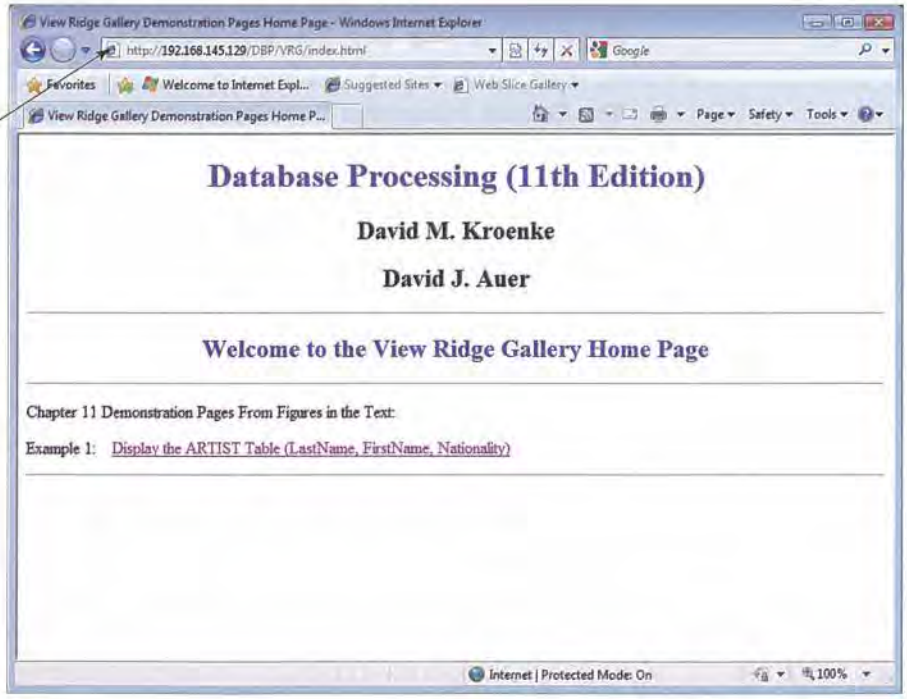


Figure 11-29

The index.html Web Page in VRG

Amazon Ex. 1008.0479

BY THE WAY If you are working on a single computer, with the DBMS, Web server, and development tools all installed together, you will see a consistent user interface. It may be Windows XP, Windows Vista, or a version of Linux. This is, in fact, typical of small development platforms, and allows you to easily test each application component as you create it.

In a larger production environment, however, the Web server and database server (which may or may not be the same physical server) are separate from the developer's workstation. In this case you, as the developer, will see different user interfaces depending on which computer you're using.

We are illustrating this latter setup in this chapter. Our Web server (IIS) and DBMS server (SQL Server 2008) are on one server running Windows Server 2008. Our development tools (the IE 8 Web browser and the Eclipse IDE) are on a separate workstation running Windows Vista. Thus, you will see the differences in the user interface depending on whether the work is being done on the server (for example, in Figures 11-23 through 11-26) or on the workstation (for example, Figure 11-29).

Using PHP

Now that we have our basic Web site set up, we will expand its capabilities with a Web development environment that allows us to connect Web pages to our database. Several technologies allow us to do this. Developers using Microsoft products usually work with the .NET framework and use ASP.NET technology. Developers who use the Apache Web server may prefer creating JSP files in the JavaScript scripting language or using the Java programming language in the Java Enterprise Edition (Java EE) environment.

The PHP Scripting Language

In this chapter, we will use PHP, available as this is being written in version 5.2.9, and available for free download from the PHP Web site (*www.php.net*). You should download the latest version for you operating system and install PHP on your computer. There is documentation on the PHP Web site, and good discussion can also be found by searching the Web for "PHP installation." Setting up PHP usually requires several steps (not just running an installation routine), so take some time and be sure you have PHP running correctly. Also be sure to enable PHP Data Objects (PDO)—this is not done automatically.

The Eclipse Integrated Development Environment (IDE)

Although a simple text editor such as Notepad is fine for simple Web pages, as we start creating more complex pages we will move to an **Integrated Development Environment (IDE)**. An IDE is intended to be a complete development framework, with all the tools you need in one place. An IDE gives you the most robust and user-friendly means of creating and maintaining your Web pages.

If you are working with Microsoft products, you will most likely use Visual Studio (or the Visual Studio 2008 Express Editions, available at for free from *www.microsoft.com/express/*). In fact, if you've installed SQL Server 2008 Express Advanced or any non-Express version of the product, you have already installed some Visual Studio 2008 components. These are installed to support SQL Server Reporting Services, and are sufficient for creating basic Web pages. If you are working with JavaScript or Java, you might prefer the NetBeans IDE (downloadable from *www.netbeans.org*).

For this chapter, we will again turn to the open-source development community and use the **Eclipse IDE**. Eclipse provides a framework that can be modified by add-in modules for many purposes. For PHP, we can use Eclipse as modified for the **Eclipse PDT (PHP Development Tools) Project**, which is specifically intended to provide a PHP development environment in

Amazon Ex. 1008.0480

The **DBP-e11-VRG** project—Eclipse organizes work into projects

The **index.html** HTML code—note how color coding has been added to indentation to keep the code organized and readable

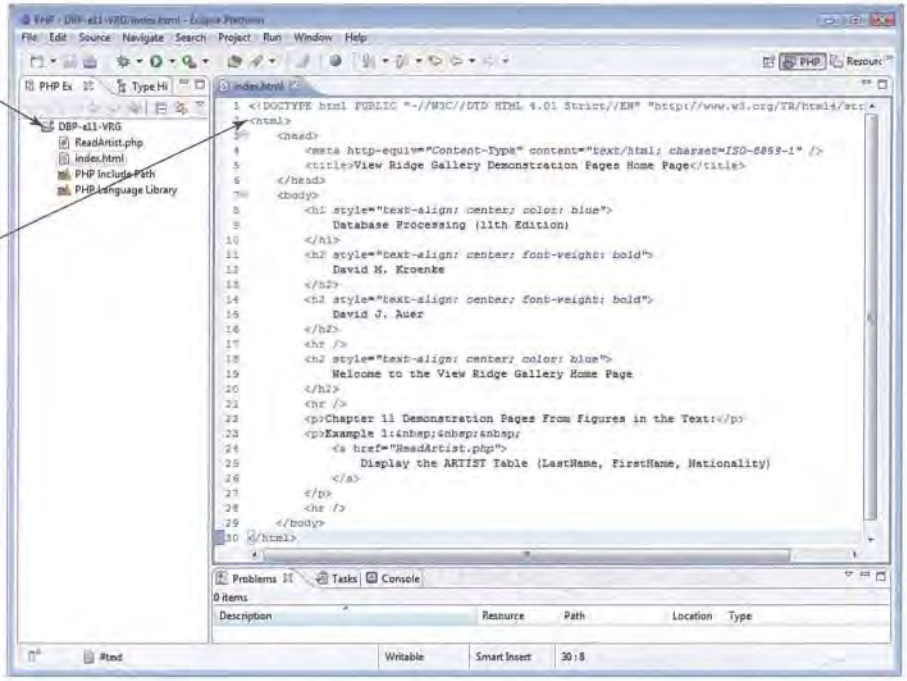


Figure 11-30

The HTML Code for the index.html File in the Eclipse IDE

Eclipse (see *www.eclipse.org/pdt/* for general information and *http://download.eclipse.org/tools/pdt/downloads/* for downloadable files—download the current stable build for your operating system).[5]

Figure 11-29 shows the index.html file as created in the Eclipse IDE. Compare this version with the Notepad version in Figure 11-30.

The ReadArtist.php File

Now that we have our basic Web site set up, we will start to integrate PHP into the Web pages. First, we will create a page to read data from a database table and display the results in a Web page. Specifically, we will create a Web page in the VRG folder named ReadArtist.php to run the SQL query:

```
SELECT LastName, FirstName, Nationality FROM ARTIST;
```

This page displays the result of the query, without the table's surrogate key of ArtistID, in a Web page. The HTML and PHP code for ReadArtist.php is shown in Figure 11-31, and the same code is shown in Eclipse in Figure 11-32.

Now if you use the URL *http://localhost/DBP/VRG* in your Web browser and then click the **Example 1: Display the ARTIST Table (No surrogate key)** link on that page, the Web page shown in Figure 11-33 is displayed.

The ReadArtist.php code blends HTML (executed on the user's workstation) and PHP statements (executed on the Web server). In Figure 11-30, the statements included between the **<?php** and **?>** tags are program code that is to be executed on the Web server computer. All

[5] As noted on the PDT download page, you also need to install the Java Runtime Environment (as of this writing, it's JRE 6 Update 11 from Sun Microsystems, at *http://java.sun.com/javase/downloads/index.jsp*). Also note that the Windows version of Eclipse does not install like most Windows program: You need to create a folder named Eclipse in the Program Files folder on your C: drive (the full path name will be C:\Program Files\Eclipse), unzip the Eclipse PDT files in that directory, and create a desktop shortcut to Eclipse.exe.

Amazon Ex. 1008.0481

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
      <head>
            <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
            <title>ReadArtist PHP Page</title>
            <style type="text/css">
                  h1 {text-align: center; color: blue}
                  h2 {font-family: Ariel, sans-serif; text-align: left; color: blue}
                  p.footer {text-align: center}
                  table.output {font-family: Ariel, sans-serif}
            </style>
      </head>
      <body>
      <?php
            // Get connection
            $DSN = "VRG";
            $User = "VRG-User";
            $Password = "VRG-User+password";

            $Conn = odbc_connect($DSN, $User, $Password);

            // Test connection
            if (!$Conn)
            {
                  exit ("ODBC Connection Failed: " . $Conn);
            }
            // Create SQL statement
            $SQL = "SELECT LastName, FirstName, Nationality FROM ARTIST";

            // Execute SQL statement
            $RecordSet = odbc_exec($Conn,$SQL);

            // Test existence of recordset
            if (!$RecordSet)
                  {
                        exit ("SQL Statement Error: " . $SQL);
                  }
      ?>
            <!--  Page Headers -->
            <h1>
                  The View Ridge Gallery ARTIST Table
            </h1>
            <hr />
            <h2>
                  ARTIST
            </h2>
      <?php
```

Figure 11-31

The HTML and PHP Code for ReadArtist.php

Amazon Ex. 1008.0482

```
            // Table headers
            echo "<table class='output' border='1'>
                  <tr>
                        <th>LastName</th>
                        <th>FirstName</th>
                        <th>Nationality</th>
                  </tr>";

            // Table data
            while($RecordSetRow = odbc_fetch_array($RecordSet))
                  {
                  echo "<tr>";
                  echo "<td>" . $RecordSetRow['LastName'] . "</td>";
                  echo "<td>" . $RecordSetRow['FirstName'] . "</td>";
                  echo "<td>" . $RecordSetRow['Nationality'] . "</td>";
                  echo "</tr>";
                  }
            echo "</table>";

            // Close connection
            odbc_close($Conn);
      ?>
            <br />
            <hr />
            <p class="footer">
                  <a href="../VRG/index.html">
                        Return to View Ridge Gallery Home Page
                  </a>
            </p>
            <hr />
      </body>
</html>
```

Figure 11-31 (Continued)

The **ReadArtist.php** code—PHP code is enclosed in the **<?php** and **?>** symbols, which are displayed in red in Eclipse

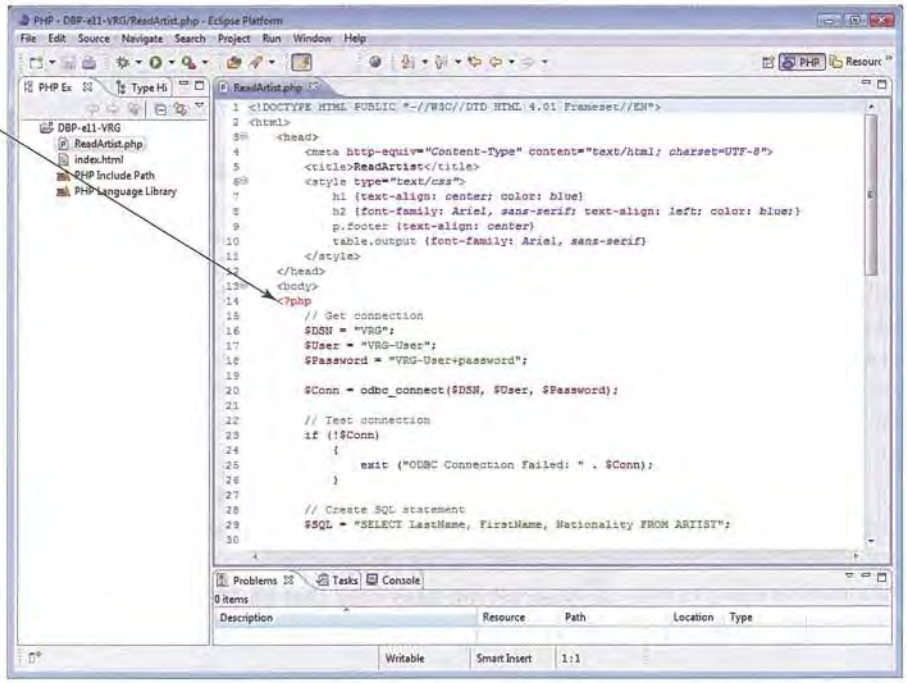


Figure 11-32
The HTML and PHP Code for ReadArtist.php in Eclipse

Amazon Ex. 1008.0483

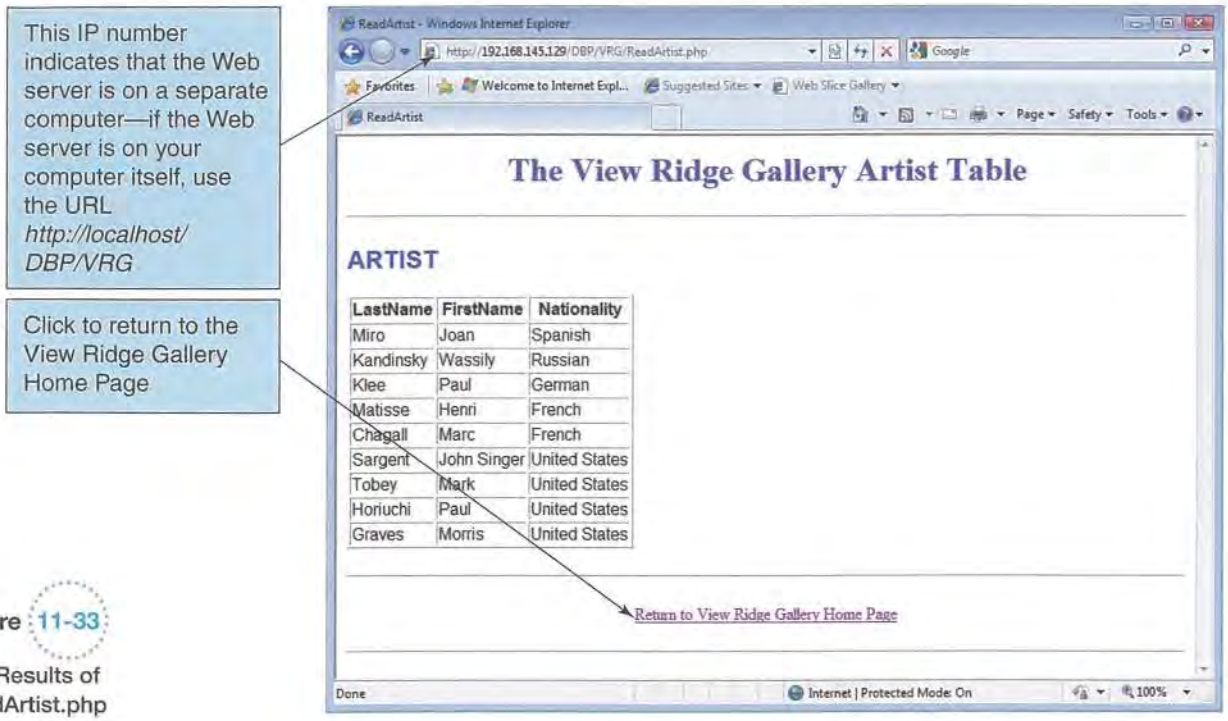


Figure 11-33
The Results of ReadArtist.php

the rest of the code is HTML that is generated and sent to the browser client. In Figure 11-30, the statements:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
      <head>
            <meta http-equiv="Content-Type" content="text/html;
            charset=UTF-8">
            <title>ReadArtist PHP Page</title>
            <style type="text/css">
                  h1 {text-align: center; color: blue}
                  h2 {font-family: Ariel, sans-serif; text-align:
                  left; color: blue}
                  p.footer {text-align: center}
                  table.output {font-family: Ariel, sans-serif}
            </style>
      </head>
      <body>
```

are normal HTML code. When sent to the browser, these statements set the title of the browser window to ReadSeminar PHP Page; define styles to be used by the headings,[6] the results table, and the footer; and cause other HTML-related actions. The next group of statements are included between <?php and ?> and, thus, are PHP code that will be executed on the Web server. Also note that all PHP statements, like SQL statements, must end with a semicolon (;).

[6] Styles are used to control the visual presentation of the Web page and are defined in the HTML section between the <style> and </style> tags. For more information about styles, see David Raggett's "Adding a Touch of Style" tutorial at *www.w3.org/MarkUp/Guide/Style.html*.

Amazon Ex. 1008.0484

Creating a Connection to the Database

In the HTML and PHP code in Figure 11-31, the following PHP code is embedded in the HTML code to create and test a connection to the database:

```
<?php
    // Get connection
    $DSN = "VRG";
    $User = "VRG-User";
    $Password = "VRG-User+password";
    $Conn = odbc_connect($DSN, $User, $Password);
    // Test connection
    if (!$Conn)
    {
        exit ("ODBC Connection Failed: " . $Conn);
    }
```

After it runs, the variable $Conn can be used to connect to the ODBC data source VRG. Note that all PHP variables start with the dollar sign symbol ($).

BY THE WAY Be sure to use comments to document your Web pages. PHP code segments with two forward slashes (//) in front of them are comments. This symbol is used to define single-line comments. In PHP, comments can also be inserted in blocks between the symbols /* and */, whereas in HTML comments *must* be inserted between the symbols <!-- and -->.

The connection is used to open the VRG ODBC data source. Here, the user ID of VRG-User and the password of VRG-User+password that we created in Chapters 10 for SQL Server 2008 are being used to authenticate to the DBMS. If you are using Oracle or MySQL, use the ODBC data source name, user name, and user password as you created it for your database.

The test of the connection is contained in the code segment:

```
// Test connection
if (!$Conn)
{
    exit ("ODBC Connection Failed: " . $Conn);
}
```

In English, this statement says, "IF the connection Conn does not exist, THEN print the error message 'ODBC Connection Failed' followed by the contents of the variable $Conn." Note that the code (!$Conn) means NOT $Conn—in PHP the exclamation point symbol (!) means NOT.

At this point, a connection has been established to the DBMS via the ODBC data source, and the database is open. The $Conn variable can be used whenever a connection to the database is needed.

Creating a RecordSet

Given the connection with an open database, the following code segment from Figure 11-31 will store an SQL statement in the variable $SQL and then use the PHP odbc_exec command to run that SQL statement against the database to retrieve the query results and store them in the variable $RecordSet:

```
// Create SQL statement
$SQL = "SELECT LastName, FirstName, Nationality FROM ARTIST";
// Execute SQL statement
$RecordSet = odbc_exec($objConn, $SQL);
```

Amazon Ex. 1008.0485

```
    // Test existence of recordset
    if (!$RecordSet)
        {
            exit ("SQL Statement Error: " . $SQL);
        }
?>
```

Note that you need to test the results to be sure the PHP command executed correctly.

Displaying the Results

Now that the record set name $RecordSet has been created and populated, we can process the $RecordSet collection with the following code:

```
    <!-- Page Headers -->
    <h1>
        The View Ridge Gallery ARTIST Table
    </h1>
    <hr />
    <h2>
        ARTIST
    </h2>
<?php
    // Table headers
    echo "<table class='output' border='1'>
        <tr>
            <th>LastName</th>
            <th>FirstName</th>
            <th>Nationality</th>
        </tr>";
    // Table data
    while($RecordSetRow = odbc_fetch_array($RecordSet))
          {
          echo "<tr>";
          echo "<td>" . $RecordSetRow['LastName'] . "</td>";
          echo "<td>" . $RecordSetRow['FirstName'] . "</td>";
          echo "<td>" . $RecordSetRow['Nationality'] . "</td>";
          echo "</tr>";
          }
    echo "</table>";
```

The HTML section defines the page headers, and the PHP section defines how to display the SQL results in a table format. Note the use of the PHP command *echo* to allow PHP to use HTML syntax within the PHP code section. Also note that a loop is executed to iterate through the rows of the RecordSet using the PHP variable $RecordSetRow.

Disconnecting from the Database

Now that we have finished running the SQL statement and displaying the results, we can end our ODBC connection to the database with the code:

```
    // Close connection
    odbc_close($Conn);
?>
```

Amazon Ex. 1008.0486

The basic page we have created here illustrates the basic concepts of using ODBC and PHP to connect to a database and process data from that database in a Web database processing application. We can now build on this foundation by studying PHP command syntax and incorporating additional PHP features into our Web pages.[7]

Challenges for Web Database Processing

Web database application processing is complicated by an important characteristic of HTTP. Specifically, HTTP is stateless; it has no provision for maintaining sessions between requests. Using HTTP, a client at a browser makes a request of a Web server. The server services the client request, sends results back to the browser, and forgets about the interaction with that client. A second request from that same client is treated as a new request from a new client. No data are kept to maintain a session or connection with the client.

This characteristic poses no problem for serving content, either static Web pages or responses to queries of a database. However, it is not acceptable for applications that require multiple database actions in an atomic transaction. Recall from Chapter 6 that in some cases, a group of database actions needs to be grouped into a transaction, with all of them committed to the database or none of them committed to the database. In this case, the Web server or other program must augment the base capabilities of HTTP.

For example, IIS provides features and functions for maintaining data about sessions between multiple HTTP requests and responses. Using these features and functions, the application program on the Web server can save data to and from the browser. A particular session will be associated with a particular set of data. In this way, the application program can start a transaction, conduct multiple interactions with the user at the browser, make intermediate changes to the database, and commit or roll back all changes when ending the transaction. Other means are used to provide for sessions and session data with Apache.

In some cases, the application programs must create their own methods for tracking session data. PHP does include support for sessions—see the PHP documentation for more information.

The particulars of session management are beyond the scope of this chapter. However, you should be aware that HTTP is stateless, and, regardless of the Web server, additional code must be added to database applications to enable transaction processing.

Web Page Examples with PHP

The following three examples extend our discussion of using PHP Web pages in Web database applications. These examples focus mainly on the use of PHP and not as much on the graphics, presentation, or workflow. If you want a flashy, better-behaving application, you should be able to modify these examples to obtain that result. Here, just learn how PHP is used.

All of these examples process the View Ridge Gallery database. In all of them we use the VRG database in each DBMS as we constructed it for SQL Server, Oracle, and MySQL in Chapters 10, 10A, and 10B, respectively. For simplicity, we connect to each using an ODBC system data source—VRG for SQL Server, VRG-Oracle for Oracle, and VRG-MySQL for MySQL. And if we use the same user name and password in each DBMS, we need to only change the ODBC data source name to switch between DBMSs! That is amazing, and exactly what the originators of ODBC hoped for when they created the ODBC specification.

Note, however, that although we are using ODBC functions, PHP actually provides a specific set for most DBMS products. These sets are generally more efficient than ODBC, and if you

[7] For more information on PHP, see the PHP documentation at *www.php.net/docs.php*.

Amazon Ex. 1008.0487

are working with a specific DBMS you will want to explore the PHP function set for it.[8] As an example of this, note that we connected to the database using

```
// Get connection
$DSN = "VRG";
$User = "VRG-User";
$Password = "VRG-User+password";
$Conn = odbc_connect($DSN, $User, $Password);
```

If we are using MySQL, however, we can use:

```
// Get connection
$Host = "localhost";
$User = "VRG-User";
$Password = "VRG-User+password";
$Database = "VRG";
$Conn = mysqli_connect($Host, $User, $Password, $Database);
```

Similarly, SQL Server uses the *sqlsrv_connect* function (using the Microsoft PHP driver described in footnote 8), and Oracle uses the *oci_connect* function.

PHP 5.2 also supports object-oriented programming and a new data abstraction layer called **PHP Data Objects (PDO)** that provides a common syntax for accessing DBMS products. There is a lot of power in PHP, and we will barely scratch the surface here.

However, before proceeding with our examples, we need to add some links to our VRG homepage. The necessary code is shown in Figure 11-34. If you are working through these examples (and you should be), be sure to make these changes.

Example 1: Updating a Table

The previous example of a PHP Web page just read data. This next example shows how to update table data by adding a row to a table with PHP. Figure 11-35 shows a data entry form that will capture artist name and nationality and create a new row. This form has three data entry fields: the First Name and Last Name fields are text boxes where the user types in the artist's name and the Nationality field has been implemented as a drop-down list to control the possible values and to make sure they are spelled correctly. When the user clicks Save New Artist, the artist is added to the database; and if the results are successful, the acknowledgment Web page in Figure 11-36 is displayed. The See Revised Artist List link will invoke the ReadArtist.php page, which will display the ARTIST table with the new row, as shown in Figure 11-37. We have tested these pages by adding the American artist Guy Anderson (born 1906, deceased 1998), who is a member of the Northwest School.

This processing necessitates two PHP pages. The first, shown in Figure 11-38, is the data entry form with three fields: artist last name, artist first name, and artist nationality.

It also contains the form tag:

```
<form action="InsertNewArtist.php" method="POST">
```

This tag defines a form section on the page, and the section will be set up to obtain data entry values. This form has only one data entry value: the table name. The **POST method** refers to a process that causes the data in the form (here, the last name, the first name, and the selected nationality) to be delivered to the PHP server so it can be used in an array variable named $_POST. Note that $_POST is an array, and can thus have multiple values. An alternative method is GET, but POST can carry more data, and this distinction is not too important to us here. The

[8] Microsoft has created an updated set of functions for SQL Server If you are going to use the SQL Server specific functions, you should download the SQL Server Driver for PHP. As this book goes to press, although this driver works with both SQL Server 2005 and SQL Server 2008, it is still referred to as the SQL Server 2005 Driver for PHP on the Microsoft SQL Feature Pack Web page at *www.microsoft.com/downloads/details.aspx?FamilyId=228DE03F-3B5A-428A-923F-58A033D316E1&displaylang=en*. Documentation is available at *msdn.microsoft.com/en-us/library/cc296189.aspx*.

Amazon Ex. 1008.0488

```
<p>Chapter 11 Demonstration Pages From Figures in the Text:</p>
      <p>Example 1:   
            <a href="ReadArtist.php">
                  Display the ARTIST Table (LastName, FirstName, Nationality)
            </a>
      </p>
<!-- ************ New text starts here ************  -->
      <p>Example 2:   
            <a href="NewArtistForm.html">
                  Add a New Artist to the ARTIST Table
            </a>
      </p>
      <p>Example 3:   
            <a href="ReadArtistPDO.php">
                  Display the ARTIST Table Using PHP PDO
            </a>
      </p>
      <p>Example 4:   
            <a href="NewCustomerWithInterestsForm.html">
                  Add a New Customer to the CUSTOMER Table Using PHP PDO
            </a>
      </p>

<!-- ************ New text ends here ************  -->
      <hr />
```

Figure 11-34

Modifications to the VRG index.html Homepage

second parameter of the form tag is *action*, which is set to InsertNewArtist.php. This parameter tells the Web server that when it receives the response from this form it should store the data values in the $_POST array and pass control to the InsertNewArtist.php page.

The rest of the page is standard HTML, with the addition of the <select><option> . . . </option></select> structure for creating a drop-down list in the form. Note that the name of the select is Nationality.



Figure 11-35

The Add New Artist Form

Amazon Ex. 1008.0489

The New Artist Added message is displayed along with the artist data

Click this link to see the ARTIST table with the new artist data



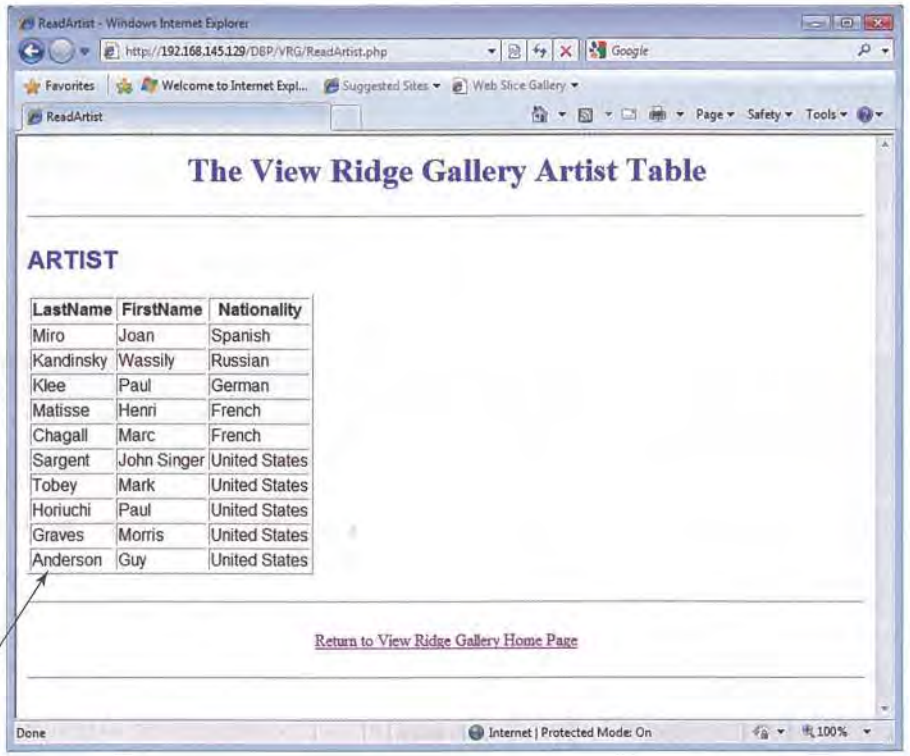
Figure 11-36

The New Artist Acknowledgment Page

Figure 11-37

The ARTIST Table with the New Artist

The **Guy Anderson** data



When the user clicks the Add New Artist button, these data are to be processed by the InsertNewArtist.php page.

Figure 11-39 shows the InsertNewArtist.php, the page that will be invoked when the response is received from the form. Note that the variable values for the INSERT statement are obtained from the $_POST[] array. First, we create short variable names for the $_POST

Amazon Ex. 1008.0490

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>NewArtistForm HTML Page</title>
      <style type="text/css">
         h1 {text-align: center; color: blue}
         h2 {font-family: Ariel, sans-serif; text-align: left; color: blue}
         p.footer {text-align: center}
         table.output {font-family: Ariel, sans-serif}
      </style>
   </head>
   <body>
      <form action="InsertNewArtist.php" method="POST">
         <!--  Page Headers -->
         <h1>
            The View Ridge Gallery New Artist Form
         </h1>
         <hr />
         <br />
         <p>
            <b>Enter artist name:</b>
         </p>
         <table>
            <tr>
               <td> Last Name:  </td>
               <td>
                  <input type="text" name="LastName" size="25" />
               </td>
            </tr>
            <tr>
               <td> First Name:  </td>
               <td>
                  <input type="text" name="FirstName" size="25" />
               </td>
            </tr>
         </table>
         <p>
            <b>Select artist nationality:</b>
         </p>
         <select name="Nationality">
            <option value="Canadian">Canadian</option>
            <option value="English">English</option>
            <option value="French">French</option>
            <option value="German">German</option>
            <option value="Mexican">Mexican</option>
            <option value="Russian">Russian</option>
            <option value="Spanish">Spanish</option>
            <option value="United States">United States</option>
         </select>
         <br />
         <p>
            <input type="submit" value="Add New Artist" />
            <input type="reset" value="Reset Values" />
         </p>
      </form>
      <br />
      <hr />
      <p class="footer">
         <a href="../VRG/index.html">
            Return to View Ridge Gallery Home Page
         </a>
      </p>
      <hr />
   </body>
</html>
```

Figure 11-38

The HTML Code for NewArtistForm.html

Amazon Ex. 1008.0491

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>InsertNewArtist PHP Page</title>
      <style type="text/css">
         h1 {text-align: center; color: blue}
         h2 {font-family: Ariel, sans-serif; text-align: left; color: blue}
         p.footer {text-align: center}
         table.output {font-family: Ariel, sans-serif}
      </style>
   </head>
   <body>
   <?php
      // Get connection
         $DSN = "VRG";
         $User = "VRG-User";
         $Password = "VRG-User+password";

         $Conn = odbc_connect($DSN, $User, $Password);

      // Test connection
      if (!$Conn)
         {
            exit ("ODBC Connection Failed: " . $Conn);
         }
      // Create short variable names
      $LastName = $_POST["LastName"];
      $FirstName = $_POST["FirstName"];
      $Nationality = $_POST["Nationality"];

      // Create SQL statement
      $SQL = "INSERT INTO ARTIST(LastName, FirstName, Nationality) ";
      $SQL .= "VALUES('$LastName', '$FirstName', '$Nationality')";

      // Execute SQL statement
      $Result = odbc_exec($Conn, $SQL);

      // Test existence of result
      echo "<h1>
            The View Ridge Gallery ARTIST Table
         </h1>
         <hr />";
      if ($Result){
         echo "<h2>
            New Artist Added:
         </h2>
```

Figure 11-39

The HTML and PHP Code for Insert NewArtist.php

Amazon Ex. 1008.0492

```
            <table>
               <tr>";
               echo "<td>Last Name:</td>";
               echo "<td>" . $LastName . "</td>";
               echo "</tr>";
               echo "<tr>";
               echo "<td>First Name:</td>";
               echo "<td>" . $FirstName . "</td>";
               echo "</tr>";
               echo "<tr>";
               echo "<td>Nationality:</td>";
               echo "<td>" . $Nationality . "</td>";
               echo "</tr>";
            echo "</table><br />";
            }
            else {
               exit ("SQL Statement Error: " . $SQL);
            }

         // Close connection
         odbc_close($Conn);
      ?>
         <br />
         <hr />
         <p class="footer">
            <a href="../VRG/ReadArtist.php">
               Display the ARTIST Table (LastName, FirstName, Nationality)
            </a>
         </p>
         <p class="footer">
            <a href="../VRG/index.html">
               Return to View Ridge Gallery Home Page
            </a>
         </p>
         <hr />
      </body>
   </html>
```

Figure 11-39 (Continued)

version of the name, and then we use these short variable names to create the SQL INSERT statement. Thus:

```
// Create short variable names
$LastName = $_POST["LastName"];
$FirstName = $_POST["FirstName"];
$Nationality = $_POST["Nationality"];

// Create SQL statement
$SQL = "INSERT INTO ARTIST(LastName, FirstName, Nationality) ";
$SQL .= "VALUES('$LastName', '$FirstName', '$Nationality')";
```

Note the use of the **PHP concatenation operator (.=)** (a combination of a period and an equals sign) to combine the two sections of the SQL INSERT statement. As another example, to create a variable named $AllOfUs with the value *me, myself, and I* we would use:

```
$AllOfUs = "me, ";
$AllOfUs .= "myself, ";
$AllOfUs .= "and I";
```

Most of the code is self-explanatory, but make sure you understand how it works.

Amazon Ex. 1008.0493

Example 2: Using PHP Data Objects (PDO)

Our next example is an exercise in using PHP Data Objects (PDO). Here we are re-creating the ReadArtist.php page, but using PDO to do it. We call the new Web page ReadArtistPDO.php, and it is shown in Figure 11-40. The PHP code to create the page is shown in Figure 11-41, and you should compare this PHP code to the PHP code for ReadArtist.php in Figure 11-31.

PHP PDO will become important as newer versions of PHP are released. The power of PHP PDO is that the only line of PHP code that needs to be changed when using a different DBMS product is the one that establishes the connection to the database. In Figure 11-41 this is the line:

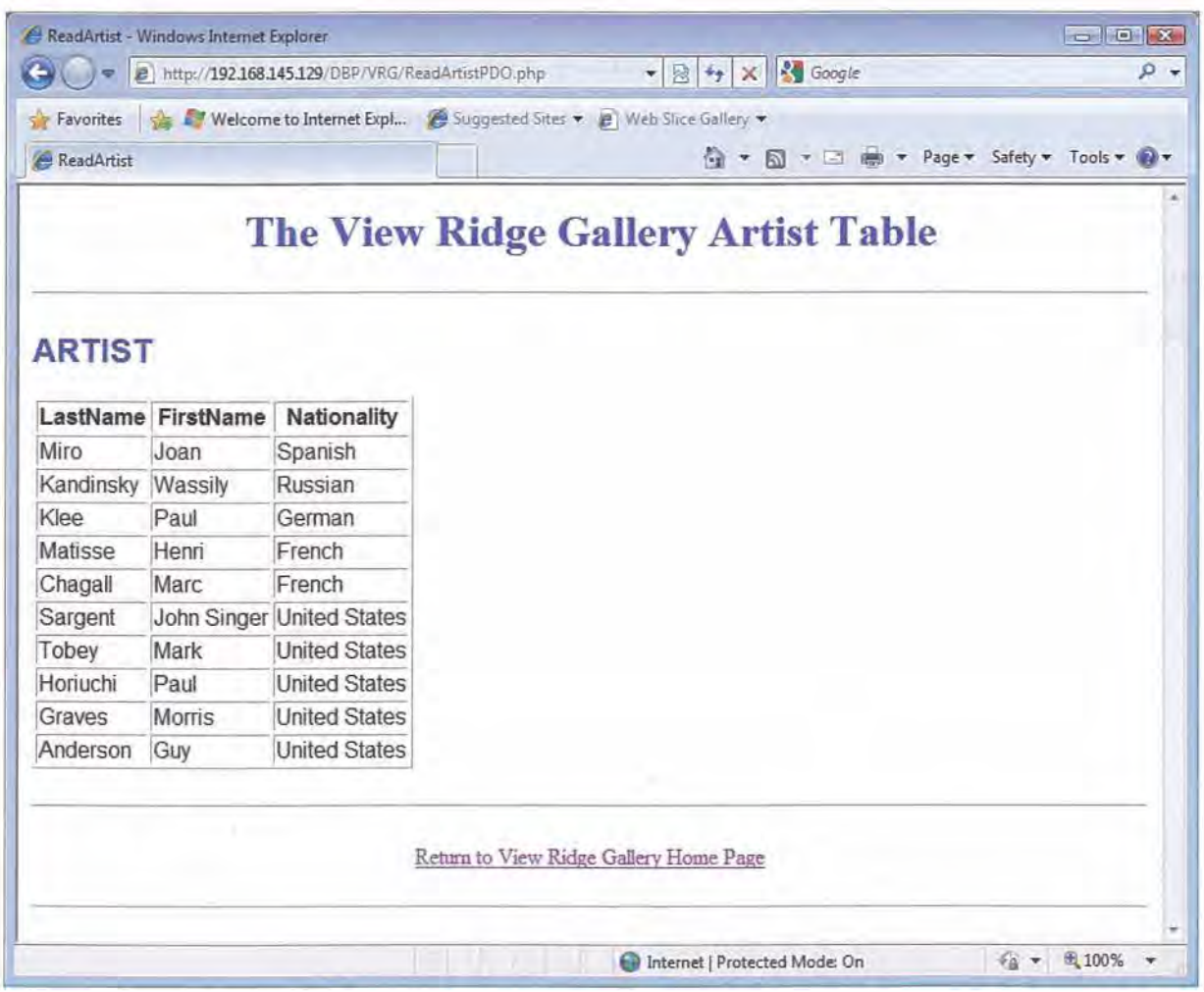
```
$PDOconnection = new PDO("odbc:$DSN", $User, $Password);
```

Example 3: Invoking a Stored Procedure

We created a stored procedure named InsertCustomerAndInterest for the SQL Server, Oracle, and MySQL versions of the VRG database in Chapters 10, 10A, and 10B, respectively. In all cases, the stored procedure accepts a new customer's last name, first name, area code, local number, e-mail, and the nationality of all artists in whom the customer is interested. It then creates a new row in CUSTOMER and adds appropriate rows to the CUSTOMER_ARTIST_INT table.

To invoke the stored procedure using a PHP page using PDO, we create a Web form page to collect the necessary data, as shown in Figure 11-42. Then, when the user clicks Add New Customer, we want to invoke a PHP page that uses PDO to call the stored procedure with the form data. So that the user can verify that the new data have been entered correctly, the PHP

Figure 11-40
The Results of ReadArtistPDO.php



Amazon Ex. 1008.0494

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>ReadArtist</title>
      <style type="text/css">
         h1 {text-align: center; color: blue}
         h2 {font-family: Ariel, sans-serif; text-align: left; color: blue;}
         p.footer {text-align: center}
         table.output {font-family: Ariel, sans-serif}
      </style>
   </head>
   <body>
   <?php
      // Get connection
      $DSN = "VRG";
      $User = "VRG-User";
      $Password = "VRG-User+password";

      $PDOconnection = new PDO("odbc:$DSN", $User, $Password);

      // Test connection
      if (!$PDOconnection)
         {
            exit ("ODBC Connection Failed: " . $PDOconnection);
         }

      // Create SQL statement
      $SQL = "SELECT LastName, FirstName, Nationality FROM ARTIST";

      // Execute SQL statement
      $RecordSet = $PDOconnection->query($SQL);

      // Test existence of recordset
      if (!$RecordSet)
         {
            exit ("SQL Statement Error: " . $SQL);
         }
   ?>
      <!--  Page Headers -->
      <h1>
         The View Ridge Gallery Artist Table
      </h1>
      <hr />
      <h2>
         ARTIST
      </h2>
   <?php

      // Table headers
      echo "<table class='output' border='1'
            <tr>
               <th>LastName</th>
               <th>FirstName</th>
               <th>Nationality</th>
            </tr>";
```

Figure 11-41
The HTML and PHP Code for ReadArtistPDO.php

Amazon Ex. 1008.0495

```
        //Table data
        while($RecordSetRow = $RecordSet->fetch())
            {
                echo "<tr>";
                    echo "<td>" . $RecordSetRow['LastName'] . "</td>";
                    echo "<td>" . $RecordSetRow['FirstName'] . "</td>";
                    echo "<td>" . $RecordSetRow['Nationality'] . "</td>";
                echo "</tr>";
            }
        echo "</table>";

        // Close connection
        $PDOconnection = null;
    ?>
        <br />
        <hr />
        <p class="footer">
            <a href="../VRG/index.html">
                Return to View Ridge Gallery Home Page
            </a>
        </p>
        <hr />
    </body>
</html>
```

Figure 11-41
(Continued)

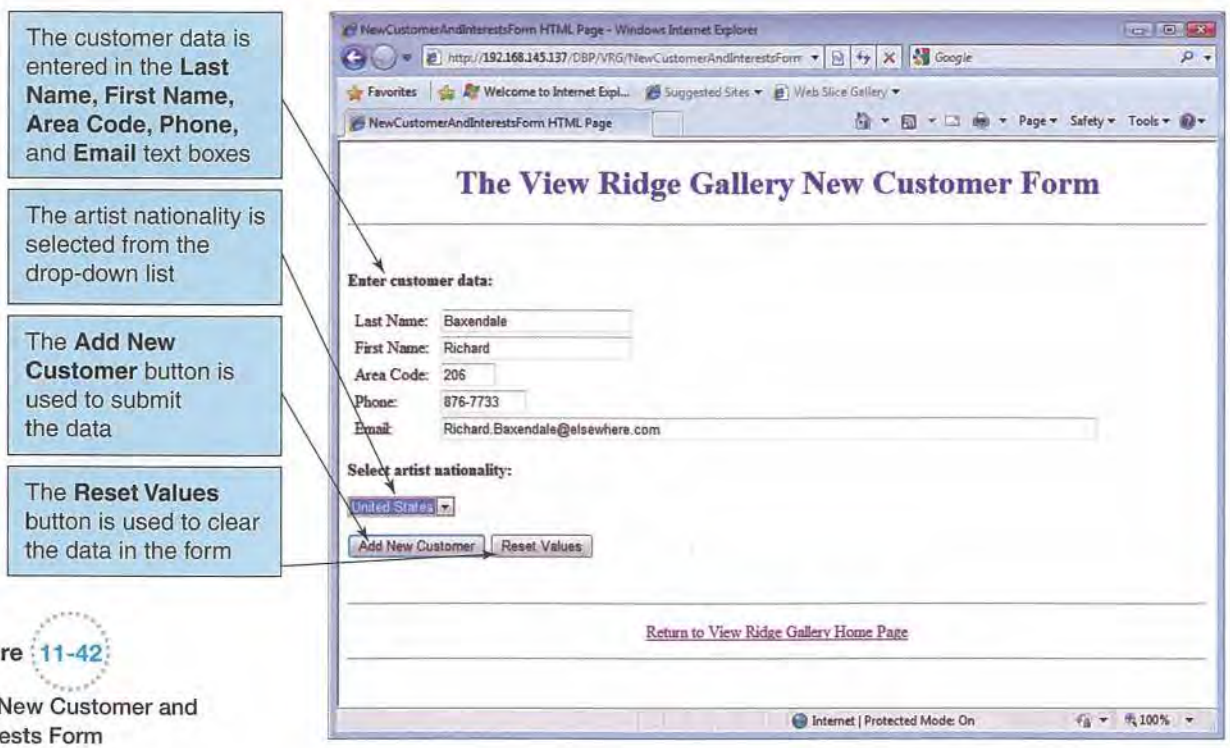


Figure 11-42
The New Customer and Interests Form

then queries a view that joins customer names with artist names and nationalities. The result is shown in Figure 11-43. In this case, we are adding Richard Baxendale, with phone number 206-876-7733 and e-mail address Richard.Baxendale@elsewhere.com. Richard is interested in United States artists.

Figure 11-44 shows the code for the NewCustomerAndInterestsForm.html page used to generate the data-gathering form. The form invokes the InsertNewCustomerAndInterestsPDO.php page code shown in Figure 11-45.

Amazon Ex. 1008.0496

The New Customer and Artist Interests added message is displayed along with the customer and artist interest data



Figure 11-43

The New Customer Acknowledgment Page

In Figure 11-45, note that the PDO statements take the form of *$Variable01 = $Variable02 ->* {PDO command}(*$Variable03*). For example, in the PDO statement $RecordSet = $PDO-connection-> query($SQL), we are using the PDO command query to send the contents of the variable $SQL to the database through the connection named $PDOconnection, and the storing the results in the variable $RecordSet. Note that while PDO standardizes the PDO command set itself, the exact SQL statements used various DBMS products will vary, and even PHP code using PDO has to be modified for those differences. For example, SQL Server uses EXEC to call a stored procedure, while MySQL uses CALL.

This PHP page is very straightforward, but it is interesting because it includes two SQL statements. First, we use an SQL CALL statement to invoke the stored procedure and pass the necessary parameters to it. Then we use an SQL SELECT statement to retrieve the values we need for the construction of our Web page acknowledging the addition of a new customer. The rest of the page reuses the same elements we have used in the previous examples.

It is also interesting that in this page we have made use of both an SQL view (CustomerInterestsView) and an SQL stored procedure (InsertCustomerAndInterests). This page illustrates the power of both these SQL structures and how we can use them in a Web database processing environment.

These examples give you an idea of the uses of PHP. The best way to learn more is to write some pages yourself. This chapter has shown all the basic techniques that you will need. You have worked hard to get to this point, and if you are able to understand enough to create some of your own pages, you have come very far indeed since Chapter 1!

Amazon Ex. 1008.0497

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>NewCustomerAndInterestsForm HTML Page</title>
      <style type="text/css">
         h1 {text-align: center; color: blue}
         h2 {font-family: Ariel, sans-serif; text-align: left; color: blue}
         p.footer {text-align: center}
         table.output {font-family: Ariel, sans-serif}
      </style>
   </head>
   <body>
      <form action="InsertNewCustomerAndInterestsPDO.php" method="POST">
         <!--  Page Headers -->
         <h1>
            The View Ridge Gallery New Customer Form
         </h1>
         <hr />
         <br />
         <p>
            <b>Enter customer data:</b>
         </p>
         <table>
            <tr>
               <td> Last Name:  </td>
               <td>
                  <input type="text" name="LastName" size="25" />
               </td>
            </tr>
            <tr>
               <td> First Name:  </td>
               <td>
                  <input type="text" name="FirstName" size="25" />
               </td>
            </tr>
            <tr>
               <td> Area Code:  </td>
               <td>
                  <input type="text" name="AreaCode" size="3" />
               </td>
            </tr>
            <tr>
               <td> Phone:  </td>
               <td>
                  <input type="text" name="PhoneNumber" size="8" />
               </td>
            </tr>
            <tr>
               <td> Email:  </td>
               <td>
                  <input type="text" name="Email" size="100" />
               </td>
            </tr>
```

Figure 11-44

The HTML Code for NewCustomerAndInterestsForm.html

Amazon Ex. 1008.0498

```
            </table>
            <p>
               <b>Select artist nationality:</b>
            </p>
            <select name="Nationality">
               <option value="Canadian">Canadian</option>
               <option value="English">English</option>
               <option value="French">French</option>
               <option value="German">German</option>
               <option value="Mexican">Mexican</option>
               <option value="Russian">Russian</option>
               <option value="Spanish">Spanish</option>
               <option value="United States">United States</option>
            </select>
            <br />
            <p>
               <input type="submit" value="Add New Customer" />
               <input type="reset" value="Reset Values" />
            </p>
         </form>
         <br />
         <hr />
         <p class="footer">
            <a href="../VRG/index.html">
               Return to View Ridge Gallery Home Page
            </a>
         </p>
         <hr />
      </body>
   </html>
```

Figure 11-44 (Continued)

Amazon Ex. 1008.0499

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>NewCustomerAndInterests PHP Page</title>
      <style type="text/css">
         h1 {text-align: center; color: blue}
         h2 {font-family: Ariel, sans-serif; text-align: left; color: blue}
         p.footer {text-align: center}
         table.output {font-family: Ariel, sans-serif}
      </style>
   </head>
   <body>
   <?php
      // Get connection
      $DSN = "VRG";
      $User = "VRG-User";
      $Password = "VRG-User+password";

      $PDOconnection = new PDO("odbc:$DSN", $User, $Password);

      // Test connection
      if (!$PDOconnection)
      {
         exit ("ODBC Connection Failed: " . $PDOconnection);
      }
      // Create short variable names
      $LastName = $_POST["LastName"];
      $FirstName = $_POST["FirstName"];
      $AreaCode = $_POST["AreaCode"];
      $PhoneNumber = $_POST["PhoneNumber"];
      $Email = $_POST["Email"];
      $Nationality = $_POST["Nationality"];

      // Create SQL statement to call the Stored Procedure
      $SQLSP = "EXEC InsertCustomerAndInterests ";
      $SQLSP .= "'$LastName', '$FirstName', '$AreaCode','$PhoneNumber', ";
      $SQLSP .= "'$Email', '$Nationality'";


      // Execute SQL statement
      $Result = $PDOconnection->exec($SQLSP);

      // Test existence of $Result
      if (!$Result)
         {
            exit ("SQL Statement Error: " . $SQLSP);
         }
```

Figure 11-45

The HTML and PHP Code for InsertNewCustomer AndInterestsPDO.php

Amazon Ex. 1008.0500

```
// Create SQL statement to retrieve additions to
// CUSTOMER_ARTIST_INT table
$SQL = "SELECT * FROM CustomerInterestsView ";
$SQL .= "WHERE CustomerLastName = '$LastName' ";
$SQL .= "AND CustomerFirstName = '$FirstName'";


// Execute SQL statement
$RecordSet = $PDOconnection->query($SQL);

// Test existence of $RecordSet
if (!$RecordSet)
   {
      exit ("SQL Statement Error: " . $SQL);
   }

echo "<h1>
      The View Ridge Gallery CUSTOMER Table
   </h1>
   <hr />";

   echo "<h2>
      New Customer and Artist Interests Added:
   </h2>
   <table>
      <tr>";
      echo "<td>Last Name:</td>";
      echo "<td>" . $LastName . "</td>";
      echo "</tr>";
      echo "<tr>";
      echo "<td>First Name:</td>";
      echo "<td>" . $FirstName . "</td>";
      echo "</tr>";
      echo "<tr>";
      echo "<td>Area Code:</td>";
      echo "<td>" . $AreaCode . "</td>";
      echo "</tr>";
      echo "<tr>";
      echo "<td>Phone Number:</td>";
      echo "<td>" . $PhoneNumber . "</td>";
      echo "</tr>";
      echo "<tr>";
      echo "<td>Email Address:</td>";
      echo "<td>" . $Email . "</td>";
      echo "</tr>";
      echo "<tr>";
      echo "<td>Artist Nationality:</td>";
      echo "<td>" . $Nationality . "</td>";
      echo "</tr>";
   echo "</table><br /><hr />";
```

Figure 11-45
(Continued)

Amazon Ex. 1008.0501

```
            // Table headers
            echo "<table class='output' border='1'>
               <tr>
                  <th>CustomerLastName</th>
                  <th>CustomerFirstName</th>
                  <th>ArtistName</th>
               </tr>";

            // Table data
            while($RecordSetRow = $RecordSet->fetch())
               {
               echo "<tr>";
               echo "<td>" . $RecordSetRow['CustomerLastName'] . "</td>";
               echo "<td>" . $RecordSetRow['CustomerFirstName'] . "</td>";
               echo "<td>" . $RecordSetRow['ArtistName'] . "</td>";
               echo "</tr>";
               }
            echo "</table>";

            // Close connection
            $PDOconnection = null;
         ?>
            <br />
            <hr />
            <p class="footer">
               <a href="../VRG-MYSQL/index.html">
                  Return to View Ridge Gallery Home Page
               </a>
            </p>
            <hr />
         </body>
      </html>
```

Figure 11-45 (Continued)

Summary

Today, database applications reside in rich and complicated environments. In addition to relational databases, there are nonrelational databases, VSAM and other file-processing data, e-mail, and other types of data. To ease the job of the application programmer, various standards have been developed. The ODBC standard is for relational databases; the OLE DB standard is for relational databases and other data sources. ADO was developed to provide easier access to OLE DB data for the non-object-oriented programmer.

ODBC, or the Open Database Connectivity standard, provides an interface by which database applications can access and process relational data sources in a DBMS-independent manner. ODBC was developed by an industry committee and has been implemented by Microsoft and many other vendors. ODBC consists of an applications program, a driver manager, DBMS drivers, and data source components. Single- and multiple-tier drivers are defined. The three data source names are file, system, and user. System data sources are recommended for Web servers. The process of defining a system data source name involves specifying the type of driver and the identity of the database to be processed.

The Microsoft .NET Framework is Microsoft's comprehensive application development framework. the current version is .NET Framework 3.5 SP1, which is built on top of the .NET Framework 2.0 and .NET Framework 3.0 (and their service pack updates). It includes ADO.NET, ASP.NET, CLR, and the Base Class Library. Enhancements specific to .NET Framework 3.5 include the ADO.NET Entity Framework, which supports the EDM, and LINQ.

OLE DB is one of the foundations of the Microsoft data access world. It implements the Microsoft OLE and COM standards, and it is accessible to object-oriented programs through those interfaces. OLE DB breaks the features and functions of a DBMS into objects, thus making it easier for vendors to implement portions of functionality. Key object terms are *abstraction, methods, properties,* and *collections.* A rowset is an abstraction of.a recordset, which, in turn, is an abstraction of a relation. Objects are defined by properties that specify their characteristics and by methods, which are the actions they can perform. A collection is an object that contains a group of other objects. An interface is a set of objects and the properties and methods they expose in that

Amazon Ex. 1008.0502

interface. Objects may expose different properties and methods in different interfaces. An implementation is how an object accomplishes its tasks. Implementations are hidden from the outside world and may be changed without impacting the users of the objects. An interface ought not to be changed, ever.

Tabular data providers present data in the form of rowsets. Service providers transform data into another form; such providers are both consumers and providers of data. A rowset is equivalent to a cursor. Basic rowset interfaces are IRowSet, IAccessor, and IColumnsInfo. Other interfaces are defined for more advanced capabilities.

ADO.NET is a new, improved, and greatly expanded version of ADO that was developed for the Microsoft .NET initiative. ADO.NET incorporates all of the functionality of ADO, but adds much more. In particular, ADO.NET facilitates the transformation of XML documents to and from database data. Most important, ADO.NET introduces the concept of DataSets, which are in-memory, fully functioned, independent databases.

A .NET data provider is a library of classes that provides ADO.NET services. A data provider data reader provides fast, forward-only access to data. A Command object can be processed to execute SQL and also to invoke stored procedures in a manner similar to but improved from that in ADO. The major new concept of ADO.NET is the DataSet. A DataSet is an in-memory database that is disconnected from any regular database but that has all the important characteristics of a regular database. DataSets can have multiple tables, relationships, referential integrity rules, referential integrity actions, views, and the equivalent of triggers. DataSet tables may have surrogate key columns (called auto-increment columns) and primary keys and may be declared unique.

DataSets are disconnected from the database(s) from which they are constructed, and they may be constructed from several different databases and possibly managed by different DBMS products. After a DataSet is constructed, an XML document of its contents and an XML Schema of its structure are easily produced. Further, the process works in reverse as well. XML Schema documents can be read to create the structure of the DataSet, and XML documents can be read to fill the DataSet.

DataSets are needed to provide a standardized, nonproprietary means to process database views. They are especially important for the processing of views with multiple multivalue paths. The potential downside of DataSets is that because they are disconnected, any updates against the databases they access must be performed using optimistic locking. In the case of conflict, either the DataSet must be reprocessed or the data change must be forced onto the database, causing the lost update problem.

JDBC is an alternative to ODBC and ADO that provides database access to programs written in Java. A JDBC driver is available for almost every conceivable DBMS product. Sun defines four driver types. Type 1 drivers provide a bridge between Java and ODBC. Types 2, 3, and 4 are written entirely in Java. Type 2 drivers rely on the DBMS product for intermachine communication, if any. Type 3 drivers translate JDBC calls into a DBMS-independent network protocol. Type 4 drivers translate JDBC calls into a DBMS-dependent network protocol.

An applet is a compiled Java bytecode program that is transmitted to a browser via HTTP and is invoked using the HTTP protocol. A servlet is a Java program that is invoked on the server to respond to HTTP requests. Type 3 and Type 4 drivers can be used for both applets and servlets. Type 2 drivers can be used only in servlets, and only then if the DBMS and Web server are on the same machine or if the DBMS vendor handles the intermachine communication between the Web server and the database server.

There are four steps when using JDBC: (1) load the driver; (2) establish a connection to the database; (3) create a statement; and (4) execute the statement.

Java Server Pages (JSP) technology provides a means to create dynamic Web pages using HTML (and XML) and Java. JSP pages provide the capabilities of a full object-oriented language to the page developer. Neither VBScript nor JavaScript can be used in a JSP page. JSP pages are compiled into machine-independent bytecode.

JSP pages are compiled as subclasses of the HTTPServlet class. Consequently, small snippets of code can be placed in a JSP page, as well as complete Java programs. To use JSP, the Web server must implement the Java Servlet 2.1+ and JSP 1.0+ specifications. Apache Tomcat, an open-source product from the Jakarta Project, implements these specifications. Tomcat can work in conjunction with Apache or as a stand-alone Web server for testing purposes.

When using Tomcat (or any other JSP processor), the JDBC drivers and JSP pages must be located in specified directories. When a JSP page is requested, Tomcat ensures that the most recent page is used. If an uncompiled newer version is available, Tomcat will automatically cause it to be parsed and compiled. Only one JSP page can be in memory at a time, and JSP requests are executed as a thread of the servlet processor, not as a separate process. The Java code in a JSP page can invoke a compiled Java bean, if desired.

PHP (PHP: Hypertext Processor) is a scripting language that can be embedded in Web pages. PHP is extremely popular and easy to learn, and it can be used in most Web server environments and with most databases.

For creating complex pages, you need an Integrated Development Environment (IDE). An IDE gives you the most robust and user-friendly means of creating and maintaining Web pages. Microsoft Visual Studio, NetBeans for Java users, and the open-source Eclipse IDE are all good IDEs. Eclipse provides a framework that can be modified by add-in modules. For PHP, there is a modification of Eclipse called the Eclipse PDT Project that is specifically intended to provide a PHP development environment.

PHP now includes object-oriented features and PHP Data Objects (PDO), which simplify connecting Web pages to databases.

Amazon Ex. 1008.0503

Key Terms

?>
<?php
abstraction
Active Data Objects (ADO)
ADO.NET
ADO.NET Command object
ADO.NET Connection object
ADO.NET Data Provider
ADO.NET DataAdapter object
ADO.NET DataReader
ADO.NET DataSet
ADO.NET Entity Framework
AMP
Apache Tomcat
Apache Web server
applet
application program interface (API)
ASP.NET
Base Class Library
bytecode interpreter
Callable Statement object
collection
Common Language Runtime (CLT)
Component Object Model (COM)
Constraints
current values
cursor
data consumer
data provider
DataRelations
DataRelationCollection
DataTable object
DataTableCollection
Default Web Site folder
DeleteCommand object
document type declaration (DTD)
DataColumns Collection
DataRow Collection
Eclipse IDE
Eclipse PDT (PHP Development Tools) Project
Entity Data Model (EDM)
Extensible Markup Language (XML)
File data source
ForeignKeyConstraint
HTML document tags
HTML syntax rules
http://localhost
Hypertext Markup Language (HTML)
iisstart.htm
implementation
index.html
inetpub folder
InsertCommand object
Integrated Development Environment (IDE)
interface
Internet Information Services (IIS)
Internet Information Services Manager
Java Data Objects (JDO)
Java Database Connectivity (JBDC)
Java platform
Java programming language
Java Server Page (JSP)
Java virtual machine
JDBC Connection object
JDBC DriverManager
JDBC ResultSet object
JDBC ResultSetMetaData object
JDBC Statement object
LAMP
Language Integrate Query (LINQ)
localstart.asp
method
Microsoft Transaction Manager (MTS)
.NET Framework
object
object class
ODBC conformance levels
ODBC data source
ODBC Data Source Administrator
ODBC driver
ODBC driver manager
ODBC multiple-tier driver
ODBC single-tier driver
ODBC SQL conformance levels
Object Linking and Embedding (OLE)
OLE DB
Open Database Connectivity (ODBC)
original values
PHP
PHP concatenation operator (.=)
PHP Data Objects (PDO)
PHP: Hypertext Processor
POST method
Prepared Statement objects
PrimaryKey property
properties
proposed values
recordset
rowset
SelectCommand object
service provider
servlet
system data source
Tabular data providers

Amazon Ex. 1008.0504

three-tier architecture
two-tier architecture
UniqueConstraint
UpdateCommand object
user data source
WAMP
World Wide Web Consortium (W3C)
wwwroot folder

Review Questions

11.1 Describe why the data environment is complicated.

11.2 Explain how ODBC, OLE DB, and ADO are related.

11.3 Explain the author's justification for describing Microsoft standards. Do you agree?

11.4 Name the components of the ODBC standard.

11.5 What role does the driver manager serve? Who supplies it?

11.6 What role does the DBMS driver serve? Who supplies it?

11.7 What is a single-tier driver?

11.8 What is a multiple-tier driver?

11.9 Do the uses of the term *tier* in the three-tier architecture and its use in ODBC have anything to do with each other?

11.10 Why are conformance levels important?

11.11 Summarize the three ODBC API conformance levels.

11.12 Summarize the three SQL grammar conformance levels.

11.13 Explain how the three types of data sources differ.

11.14 Which data source type is recommended for Web servers?

11.15 What are the two tasks to be accomplished when setting up an ODBC data source name?

11.16 What is the Microsoft .NET Framework. What basic elements does it include?

11.17 What is the current version of the .NET Framework, and what new features does it include?

11.18 Why is OLE DB important?

11.19 What disadvantage of ODBC does OLE DB overcome?

11.20 Define abstraction and explain how it relates to OLE DB.

11.21 Give an example of abstraction involving rowset.

11.22 Define object properties and methods.

11.23 What is the difference between an object class and an object?

11.24 Explain the role of data consumers and data providers.

11.25 What is an interface?

11.26 What is the difference between an interface and an implementation?

11.27 Explain why an implementation can be changed but an interface should not be changed.

11.28 Summarize the goals of OLE DB.

11.29 Explain the difference between a tabular data provider and a service provider. Which transforms OLE DB data into XML documents?

11.30 In the context of OLE DB, what is the difference between a rowset and a cursor?

Amazon Ex. 1008.0505

11.31 What is ADO.NET?

11.32 What is a data provider?

11.33 What is a data reader?

11.34 How can ADO.NET be used to process a database without using DataReaders or DataSets?

11.35 What is an ADO.NET DataSet?

11.36 How do ADO.Net DataSets differ conceptually from databases?

11.37 List the primary structures of an ADO.NET DataSet, as described in this chapter.

11.38 How do ADO.NET DataSets solve the problem of views with multivalue paths?

11.39 What is the chief disadvantage of ADO.NET DataSets? When is this likely to be a problem?

11.40 Why, in database processing, is it important to become an object-oriented programmer?

11.41 What is an ADO.NET Connection?

11.42 What is a DataAdapter?

11.43 What is the purpose of the SelectCommand property of a DataAdapter?

11.44 How is a data table relationship constructed in ADO.NET?

11.45 How is referential integrity defined in ADO.NET? What referential integrity actions are possible?

11.46 Explain how original, current, and proposed values differ.

11.47 How does an ADO.NET DataSet allow for trigger processing?

11.48 What is the purpose of the UpdateCommand property of a DataAdapter?

11.49 What are the purposes of the InsertCommand and DeleteCommand of a DataAdapter?

11.50 Explain the flexibility inherent in the use of the InsertCommand, UpdateCommand, and DeleteCommand properties.

11.51 What is the one major requirement for using JDBC?

11.52 What does JDBC stand for?

11.53 What are the four JDBC driver types?

11.54 Explain the purpose of Type 1 JDBC drivers.

11.55 Explain the purpose of Types 2, 3, and 4 JDBC drivers.

11.56 Define applet and servlet.

11.57 Explain how Java accomplishes portability.

11.58 List the four steps of using a JDBC driver.

11.59 What is the purpose of Java Server Pages?

11.60 Describe the differences between ASP and JSP.

11.61 Explain how JSP pages are portable.

11.62 What is the purpose of Tomcat?

11.63 Describe the process by which JSP pages are compiled and executed. Can a user ever access an obsolete page? Why or why not?

11.64 Why are JSP programs preferable to CGI programs?

11.65 What is Hypertext Markup Language (HTML), and what function does it serve?

11.66 What are HTML document tags, and how are they used?

Amazon Ex. 1008.0506

11.67 What is the World Wide Web Consortium (W3C)?

11.68 Why is index.hmtl a significant file name?

11.69 What is PHP, and what function does it serve?

11.70 How is PHP code designated in a Web page?

11.71 How are comments designated in PHP code?

11.72 How are comments designated in HMTL code?

11.73 What is an Integrated Development Environment (IDE), and how is it used?

11.74 What is the Eclipse IDE?

11.75 What is the Eclipse PDT Project?

11.76 Show a snippet of PHP code for creating a connection to a database. Explain the meaning of the code.

11.77 Show a snippet of PHP code for creating a RecordSet. Explain the meaning of the code.

11.78 Show a snippet of PHP code for displaying the contents of a RecordSet. Explain the meaning of the code.

11.79 Show a snippet of PHP code for disconnecting from the database. Explain the meaning of the code.

11.80 With respect to HTTP, what does *stateless* mean?

11.81 Under what circumstances does statelessness pose a problem for database processing?

11.82 In general terms, how are sessions managed by database applications when using HTTP?

11.83 What are PHP Data Objects (PDO)?

11.84 What is the significance of PDOs?

11.85 Show two snippets of PHP Code that compare creating a connection to a database in standard PHP and in PDO. Discuss the similarities and differences in the code.

Project Questions

11.86 In this exercise you will create a Web page in the DBP folder and link it to the VRG Web page in the VRG folder.

A. Figure 11-46 shows the HTML code for a Web page for the DBP folder. Note that the page is called index.html, the same name as the Web page in the VRG folder. This is not a problem because the files are in different folders. Create the index.html Web page in the DBP folder.

B. Figure 11-47 shows some additional HTML to be added near the end of the code for the VRG Web page in the file index.html in the VRG folder. Update the VRG index.html file with the code.

C. Try out the pages. Type **http://localhost/DBP** into your Web browser to display the DBP homepage. From there, you should be able to move back and forth between the two pages by using the hyperlinks on each page. *Note:* You may need to click the Refresh button on your Web browser when using the VRG home page to get the hyperlink back to the DBP home page to work properly.

11.87 Microsoft expends much effort to promote the .NET Framework. It does not directly receive revenue from these standards. IIS is free with Windows XP and 2000. Microsoft's Web sites have numerous articles to help developers learn more about these standards, and all of the information is free. Why do you think Microsoft does this? What goal is served?

Amazon Ex. 1008.0507

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
            <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
            <title>DBP-e11 Home Page</title>
      </head>
      <body>
            <h1 style="text-align: center; color: blue">
                  Database Processing (11th Edition) Home Page
            </h1>
            <hr />
            <h3 style="text-align: center">
                  Use this page to access Web-based materials from Chapter 11 of:
            </h3>
            <h2 style="text-align: center; color: blue">
                  Database Processing (11th Edition)
            </h2>
            <p style="text-align: center; font-weight: bold">
                  David M. Kroenke
            </p>
            <p style="text-align: center; font-weight: bold">
                  David J. Auer
            </p>
            <hr />
            <h3>Chapter 11 Demonstration Pages From Figures in the Text:</h3>
            <p>
                  <a href="VRG/index.html">
                        View Ridge Gallery Demonstration Pages
                  </a>
            </p>
            <hr />
      </body>
</html>
```

Figure 11-46

The HTML Code for the index.html File in the DBP Folder

```
<p>Example 4:   
   <a href="NewCustomerWithInterestsForm.html">
      Add a New Customer to the CUSTOMER Table Using PHP PDO
   </a>
</p>
<hr />
<!-- ************ NEW CODE STARTS HERE ***********  -->
<p style="text-align: center">
   <a href="../index.html">
      Return to the Database Processing Home Page
   </a>
</p>
<hr />
<!-- ************ NEW CODE ENDS HERE *************  -->
</body>
</html>
```

Figure 11-47

HTML Modifications for the index.html File in the VRG Folder

Amazon Ex. 1008.0508



If you have not already done so, answer questions A through G for Marcia's Dry Cleaning at the end of Chapter 7 (page 293). Use SQL Server, Oracle, or MySQL to implement your database.

A. Add a new folder to the DBP Web site named MDC. Create a Web page for Marcia's Dry Cleaning in this folder, using the file name index.html. Link this page to the DBP Web page.

B. Create an appropriate ODBC data source for your database.

C. Add a new column Status to ORDER. Assume that Status can have the values ['Waiting', 'In-process', 'Finished', 'Pending'].

D. Create a view called CustomerOrder that has the columns LastName, FirstName, Phone, InvoiceNumber, Date, Total, and Status.

E. Code a PHP page to display CustomerOrder. Using your sample database, demonstrate that your page works.

F. Code two HTML/PHP pages to receive a date value AsOfDate and to display rows of CustomerOrder for orders having DateIn greater than or equal to AsOfDate. Using your sample database, demonstrate that your pages work.

G. Code two HTML/PHP pages to receive customer Phone, LastName, and FirstName and to display rows for customers having that Phone, LastName, and FirstName. Using your sample database, demonstrate that your pages work.

H. Write a stored procedure that receives values for InvoiceNumber and NewStatus and that sets the value of Status to NewStatus for the row having the given value of Invoice Number. Generate an error message if no row has the given value of InvoiceNumber. Using your sample database, demonstrate that your stored procedure works.

I. Code two HTML/PHP pages to invoke the stored procedure created in task H. Using your sample database, demonstrate that your page works.



If you have not already done so, answer questions A through H for Morgan Importing at the end of Chapter 7 (page 294). Use SQL Server, Oracle, or MySQL to implement your database.

A. Add a new folder to the DBP Web site named MI. Create a Web page for Morgan Importing in this folder, using the file name index.html. Link this page to the DBP Web page.

B. Create an appropriate ODBC data source for your database.

C. Create a view called StorePurchases that has the columns StoreName, City, Country, Email, Contact, Date, Description, Category, and PriceUSD.

D. Code a PHP page to display StorePurchases. Using your sample database, demonstrate that your page works.

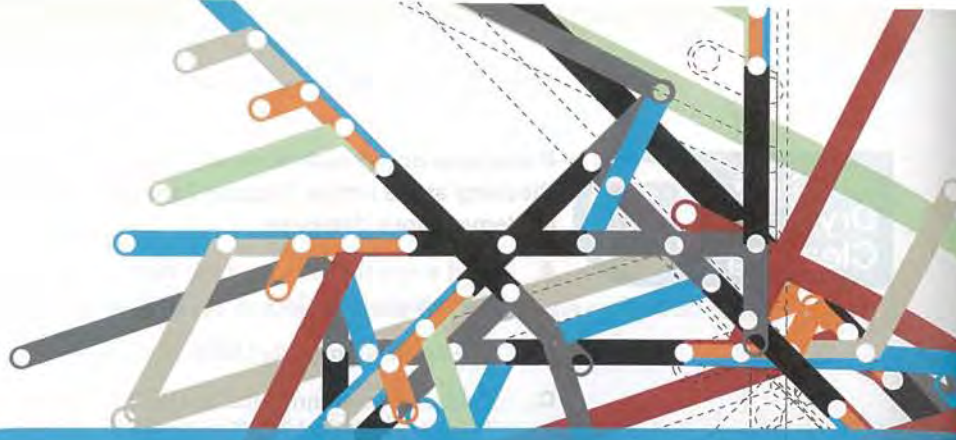
E. Code two HTML/PHP pages to receive a date value AsOfDate and display rows of StorePurchases for purchases having Date greater than or equal to AsOfDate. Using your sample database, demonstrate that your pages work.

F. Code two HTML/PHP pages to receive values of Country and Category and display rows of StorePurchases having values for input Country and Category values. Using your sample database, demonstrate that your pages work.

G. Write a stored procedure that receives values for PurchaseID and NewPriceUSD and sets the value of PriceUSD to NewPriceUSD for the row having the given value of PurchaseID. Generate an error message if no row has the given value of PurchaseID. Using your sample database, demonstrate that your stored procedure works.

H. Code two HTML/PHP pages to invoke the stored procedure created in task G. Using your sample database, demonstrate that your page works.

Amazon Ex. 1008.0509



12 Database Processing with XML

Chapter Objectives

- Understand the importance of XML.
- Learn the elements of XML, including XML documents, document type declarations (DTDs), and XML stylesheets.
- Understand the role of XSLT in materializing XML documents.
- Learn the basic concepts of XML Schema and understand their importance to database processing.
- Understand XML documents.
- Learn the basic concepts of using the SQL SELECT . . . FOR XML statement.

This chapter considers one of the most important, if not *the* most important, developments in information systems technology today. It discusses the confluence of two information technology subject areas: database processing and document processing. For more than 20 years, these two subject areas developed independently of one another. With the advent of the Internet, however, they crashed together in what some industry pundits called a technology train wreck. The result is still being sorted out, with new products, product features, technology standards, and development practices emerging every month.

Amazon Ex. 1008.0510

The Importance of XML

Database processing and document processing need each other. Database processing needs document processing for transmitting database views; document processing needs database processing for storing and manipulating data. However, even though these technologies need each other, it took the popularity of the Internet to make that need obvious. As Web sites evolved, organizations wanted to use Internet technology to display and update data from organizational databases. Web developers began to take a serious interest in SQL, database performance, database security, and other aspects of database processing.

As the Web developers invaded the database community, database practitioners wondered, "Who are these people and what do they want?" Database practitioners began to learn about HTML, the language used to mark up documents for display by Web browsers. At first, the database community scoffed at HTML because of its limitations, but they soon learned that HTML was the output of a more robust document markup language called **Standard Generalized Markup Language (SGML)**. SGML was clearly important, just as important to document processing as the relational model was to database processing. Obviously, this powerful language had some role to play in the display of database data, but what role?

In the early 1990s, the two communities began to meet, and the result of their work is a series of standards that concerns a language called **Extensible Markup Language (XML)**. XML is a subset of SGML, but additional standards and capabilities have been added to XML, and today XML technology is a hybrid of document processing and database processing. In fact, as XML standards evolved, it became clear that the communities had been working on different aspects of the same problem for many years. They even used the same terms, but with different meanings. You will see later in this chapter how the term *schema* is used in XML for a concept that is completely different from the use of *schema* in the database world.

XML provides a standardized yet customizable way to describe the content of documents. As such, it can be used to describe any database view, but in a standardized way. As you will learn, unlike SQL views, XML views are not limited to one multivalued path.

In addition, when used with the XML Schema standard, XML documents can automatically be generated from database data. Further, database data can automatically be extracted from XML documents. Even more, there are standardized ways of defining how document components are mapped to database schema components, and vice versa.

Meanwhile, the rest of the computing community began to take notice of XML. **SOAP**, which originally meant **Simple Object Access Protocol**, was defined as an XML-based standard for providing remote procedure calls over the Internet. Initially, SOAP assumed the use of HTTP as a transport mechanism. When Microsoft, IBM, Oracle, and other large companies joined forces in support of the SOAP standard, this assumption was removed, and SOAP was generalized to become a standard protocol for sending messages of any type, using any protocol. With this change, SOAP no longer meant Simple Object Access Protocol, so now SOAP is just a name, and not an acronym.

Today, XML is used for many purposes. One of the most important is its use as a standardized means to define and communicate documents for processing over the Internet. XML plays a key role in Microsoft's .NET initiative; and in 2001 Bill Gates called XML the "*lingua franca* of the Internet age."

We will begin the discussion of XML by describing its use for displaying Web pages. As you will learn, however, XML uses go far beyond Web page display. In fact, Web page display is one of the least important applications of XML. We begin with page display only because it is an easy way to introduce XML documents. After that, we will explain the XML Schema standard and discuss its use for database processing.

As you read this chapter, keep in mind that this area is at the leading edge of database processing. Standards, products, and product capabilities are changing frequently. You can keep abreast of these changes by checking the following Web sites: *www.w3c.org*, *www.xml.org*, *http://msdn.microsoft.com*, *www.oracle.com*, *www.ibm.com*, and *www.mysql.com*. Learning as much as you can about XML and database processing is one of the best ways you can prepare yourself for a successful career in database processing.

Amazon Ex. 1008.0511

XML as a Markup Language

As a markup language, XML is significantly better than HTML in several ways. For one, XML provides a clean separation between document structure, content, and materialization. XML has facilities for dealing with each, and they cannot be confounded, as they are with HTML.

Additionally, XML is standardized, but as its name implies, the standards allow for extension by developers. With XML, you are not limited to a fixed set of elements such as <title>, <h1>, and <p>; you can create your own.

Third, XML eliminates the inconsistent tag use that is possible (and popular) with HTML. For example, consider the following HTML:

```
<h2>Hello World</h2>
```

Although the <h2> tag can be used to mark a level-two heading in an outline, it can be used for other purposes, too, such as causing "Hello World" to be displayed in a particular font size, weight, and color. Because a tag has potentially many uses, we cannot rely on tags to discern the structure of an HTML page. Tag use is too arbitrary; <h2> may mean a heading, or it may mean nothing at all.

As you will see, the structure of an XML document can be formally defined. Tags are defined in relationship to one another. In XML, if we find the tag <street>, we know exactly what data we have, where that data belongs in the document, and how that tag relates to other tags.

XML Document Type Declarations

Figure 12-1 shows a sample XML document. Notice that the document has two sections. The first section defines the structure of the document; it is referred to as the **document type declaration (DTD)**. The second part is the document data.

The DTD begins with the word DOCTYPE and specifies the name of this type of document, which is customer. Then, it specifies the content of the customer document. The customer document consists of two groups: name and address. The name group consists of two elements: firstname and lastname. Firstname and lastname are defined as #PCDATA, which means that they are strings of character data. Next, the address element is defined to have four elements: street, city, state, and zip. Each of these is also defined as character data. The plus sign after street indicates that one value is required and that multiple values are possible.

Figure 12-1
XML Document with Internal DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Customer [
    <!ELEMENT Customer (CustomerName, Address)>
    <!ELEMENT CustomerName (FirstName, LastName)>
    <!ELEMENT FirstName (#PCDATA)>
    <!ELEMENT LastName (#PCDATA)>
    <!ELEMENT Address (Street+, City, State, Zip)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT State (#PCDATA)>
    <!ELEMENT Zip (#PCDATA)>
]>
<Customer>
    <CustomerName>
        <FirstName>Jeffery</FirstName>
        <LastName>Janes</LastName>
    </CustomerName>
    <Address>
        <Street>123 W. Elm St.</Street>
        <City>Renton</City>
        <State>WA</State>
        <Zip>98055</Zip>
    </Address>
</Customer>
```

Amazon Ex. 1008.0512

Figure 12-2
XML Document with External DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CustomerList SYSTEM "C:\inetpub\wwwroot\DBP\VRG\CustomerList.dtd">
<CustomerList>
    <Customer>
        <CustomerName>
            <FirstName>David</FirstName>/>
            <LastName>Smith</LastName>/>
        </CustomerName>
        <Address>
            <Street>813 Tumbleweed Lane</Street>/>
            <City>Durango</City>/>
            <State>CO</State>/>
            <Zip>81201</Zip>/>
        </Address>
    </Customer>
</CustomerList>
```

The data instance of customer shown in Figure 12-1 conforms to the DTD; hence, this document is said to be a **type-valid** XML document. If it did not conform to the DTD, it would be a **not-type-valid** document. Documents that are not-type-valid can still be perfectly good XML; they just do not conform to their type description. For example, if the document in Figure 12-1 had two city elements, it would be valid XML, but it would be not-type-valid.

Although DTDs are almost always desirable, they are not required in XML documents. Documents that have no DTD are by definition not-type-valid, because there is no type to validate them against.

The DTD does not need to be contained inside the document. Figure 12-2 shows a customer document in which the DTD is obtained from the file C:\inetpub\wwwroot\DBP\VRG\CustomerList.dtd. In this case, the DTD is located on the computer that stores this document. DTDs can also be referenced by URL Web addresses. The advantage of storing the DTD externally is that many documents can be validated against the same DTD.

The creator of a DTD is free to choose any elements he or she wants. Hence, XML documents can be extended, but in a standardized and controlled way.

Materializing XML Documents with XSLT

The XML document shown in Figure 12-1 shows both the document's structure and content. Nothing in the document, however, indicates how it is to be materialized. The designers of XML created a clean separation among structure, content, and format. The most popular way to materialize XML documents is to use XSLT, or Extensible Style Language: Transformations. XSLT is a powerful and robust transformation language. It can be used to materialize XML documents into HTML, and it can be used for many other purposes as well.

One common application of XSLT is to transform an XML document in one format into a second XML document in another format. A company can, for example, use XSLT to transform an XML order document in its own format into an equivalent XML order document in its customer's format. We will be unable to discuss many of the features and functions of XSLT here. See *www.w3.org* for more information.

XSLT is a declarative transformation language. It is declarative because you create a set of rules that govern how the document is to be materialized instead of specifying a procedure for materializing document elements. It is transformational because it transforms the input document into another document.

Figure 12-3(a) shows the CustomerList.dtd, which is a DTD for a document that has a list of customers, and Figure 12-3(b) shows the CustomerListDoucment.xml,which is an XML document that is type-valid on that DTD. The DOCTYPE statement in Figure 12-3(b) points to a file that contains the DTD shown in Figure 12-3(a). The next statement in the XML document indicates the location of another document, called a stylesheet. Shown in the

Amazon Ex. 1008.0513

Figure 12-3

CustomerList DTD and Example Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT CustomerList (Customer+)>
<!ELEMENT Customer (CustomerName, Address)>
<!ELEMENT CustomerName (FirstName, LastName)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT Address (Street+, City, State, Zip)>
<!ELEMENT Street (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT Zip (#PCDATA)>
```

(a) CustomerList.dtd DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CustomerList SYSTEM "C:\inetpub\wwwroot\DBP\VRG\CustomerList.dtd">
<?xml-stylesheet type="text/xsl" href="C:\inetpub\wwwroot\DBP\VRG\CustomerListStylesheet.xls"?>
<CustomerList>
    <Customer>
        <CustomerName>
            <FirstName>Jeffery</FirstName>
            <LastName>Janes</LastName>
        </CustomerName>
        <Address>
            <Street>123 W. Elm St.</Street>
            <City>Renton</City>
            <State>WA</State>
            <Zip>98055</Zip>
        </Address>
    </Customer>
    <Customer>
        <CustomerName>
            <FirstName>David</FirstName>
            <LastName>Smith</LastName>
        </CustomerName>
        <Address>
            <Street>813 Tumbleweed Lane</Street>
            <City>Durango</City>
            <State>CO</State>
            <Zip>81201</Zip>
        </Address>
    </Customer>
</CustomerList>
```

(b) CustomerListDocument.xml with Two Customers

CustomerListStylesheet.xsl in Figure 12-4, a **stylesheet** is used by XSLT to indicate how to transform the elements of the XML document into another format. Here, those elements will transform it into an HTML document that will be acceptable to a browser.

The XSLT processor copies the elements of the stylesheet until it finds a command in the format {*item, action*}. When it finds such a command, it searches for an instance of the indicated item; when it finds one, it takes the indicated action. For example, when the XSLT processor encounters

```
<xsl:for-each select = "CustomerList/Customer">
```

it starts a search in the document for an element named CustomerList. When it finds such an element, it looks further within the customerlist element for an element named Customer. If a match is found, it takes the actions indicated in the loop that ends with </xsl:for-each> (third from the bottom of the stylesheet). Within the loop, styles are set for each element in the CustomerList document.

Amazon Ex. 1008.0514

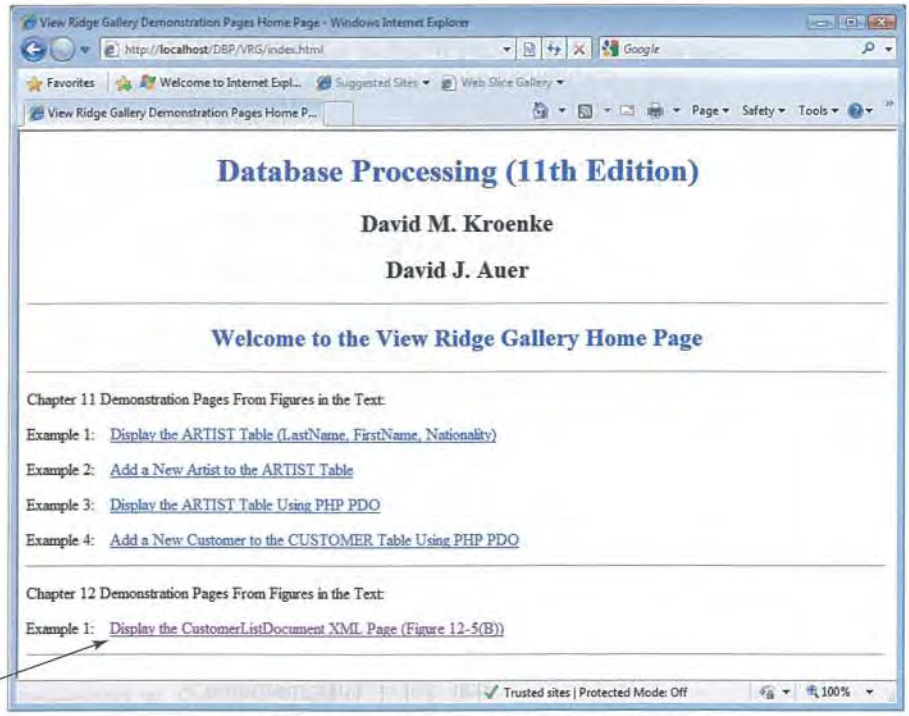
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <head>
                <title>CustomerListStylesheet</title>
                <style type="text/css">
                    h1 {text-align:center; color:blue}
                    body {font-family:ariel, sans-serif; font-size:12pt; background-color:#FFFFFF}
                    div.customername {font-weight:bold; background-color:#3399FF;
                        color:#FFFFFF; padding:4px}
                    div.customerdata {font-size:10pt; font-weight:bold; background-color:#87CEFA;
                        color:#000000; margin-left:20px; margin-bottom:20px}
                    p.footer {text-align:center}
                </style>
            </head>
            <body>
                <h1>
                    View Ridge Gallery Customer List
                </h1>
                <hr/>
                <xsl:for-each select="CustomerList/Customer">
                    <div class="customername">
                        <xsl:value-of select="CustomerName/LastName"/>,
                        <xsl:value-of select="CustomerName/FirstName"/>
                    </div>
                    <br />
                    <div class="customerdata">
                        <xsl:for-each select="Address/Street">
                            <xsl:value-of select="node()"/>
                        </xsl:for-each>
                        <br />
                        <xsl:value-of select="Address/City"/>,
                        <xsl:value-of select="Address/State"/>
                        <br />
                        <xsl:value-of select="Address/Zip"/>
                    </div>
                </xsl:for-each>
                <hr />
                <p class="footer">
                    <a href="../VRG/index.html">
                        Return to View Ridge Gallery Home Page
                    </a>
                </p>
                <hr />
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Figure 12-4

The CustomerListStylesheet.xsl XSL Stylesheet

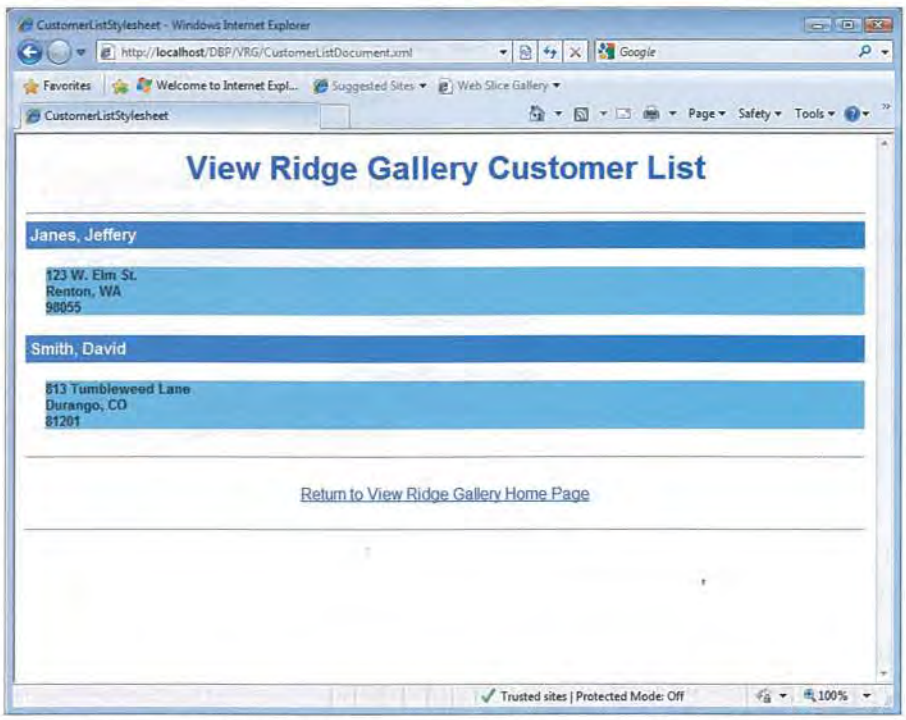
The examples we have created are based on the View Ridge Gallery database that we have used in previous chapters. Here, we are creating an XML document that can be viewed using a Web browser to display the list of customers at the View Ridge Gallery. As shown in Figure 12-5(a), we have revised the View Ridge Gallery home page we created in Chapter 11 to include a link to the XML document. Clicking the link displays the Web page shown in Figure 12-5(b), which is the result of applying the stylesheet in Figure 12-4 to the document in Figure 12-3(b). The actual code generated by the XSL stylesheet for display on the Web page is shown in Figure 12-5(c). Read through the document and the stylesheet and see how the results in Figure 12-5(c) are generated.

Amazon Ex. 1008.0515



Use this link to display the XML document in a Web browser

(a) The View Ridge Gallery Home Page



(b) The CustomerList Web Page as Displayed in Web Browser

Figure 12-5 HTML Result from Application of an XSL Stylesheet

Amazon Ex. 1008.0516

XSLT processors are context oriented; each statement is evaluated in the context of matches that have already been made. Thus, the following statement:

```
<xsl:value-of select = "name/lastname"/>
```

operates in the context of the CustomerList/Customer match that was made above. There is no need to code

```
<xsl:select = "CustomerList/Customer/CustomerName/LastName"/>
```

because the context has already been set to customerlist/customer. In fact, if the select were coded in this second way, nothing would be found. Similarly,

```
<xsl:select "LastName"/>
```

results in no match, because lastname occurs only in the context CustomerList/Customer/CustomerName, and not in the context CustomerList/Customer.



The nature of XSLT processing is: "When you find one of these, do this." Thus, the document in Figure 12-4 says, for each customer that you find under the tag customerlist, do the following: output an HTML . . . section and then some HTML with the value that you find in the document for CustomerName/LastName. Then, output more HTML and the value that you find for CustomerName/FirstName. Then, for each Address/Street you find, output some HTML along with the value of the Address/Street you just found, and so forth.

XSL can output anything. Instead of outputting HTML, it could be writing Russian or Chinese or algebraic equations. XSL is simply a transformation facility for structured documents such as XML documents.

This context orientation explains the need for the following statement (in the center of the style sheet):

```
xsl:value-of select = "node()"/>
```

The context at the location of this statement has been set to CustomerList/Customer/Address/Street. Hence, the current node is a street element, and this expression indicates that the value of that node is to be produced.

Observe, too, that a small transformation has been made by the stylesheet. The original document has FirstName followed by LastName, but the output stream has LastName followed by FirstName.

The XML document in Figure 12-3(b) is transformed into HTML using the XSL stylesheet in Figure 12-4. Figure 12-5(a) shows the revised VRG Home page, which now has a link to display the XML document. When this transformed document is input to a browser, the browser will materialize it, as shown in Figure 12-5(b).

Browsers have built-in XSLT processors. You need only supply the document to the browser; it will locate the stylesheet and apply it to the document for you. The results will be like those shown in Figure 12-5(b).

Amazon Ex. 1008.0517

XML Schema

DTDs were the XML community's first attempt at developing a document structure specification language. DTDs work, but they have some limitations, and embarrassingly, DTD documents are not XML documents. To correct these problems, the W3C Committee defined another specification language called **XML Schema**. Today, XML Schema is the preferred method for defining document structure.

XML Schemas are XML documents. This means that you use the same language to define an XML Schema as you would use to define any other XML document. It also means that you can validate an XML Schema document against its schema, just as you would any other XML document.

If you are following this discussion, then you realize that there is a chicken-and-the-egg problem here. If XML Schema documents are themselves XML documents, what document is used to validate them? What is the schema of all of the schemas? There is such a document; the mother of all schemas is located at *www.w3.org*. All XML Schema documents are validated against this document.

XML Schema is a broad and complex topic. Dozens of sizable books have been written just on XML Schema alone. Clearly, we will not be able to discuss even the major topics of XML Schema in this chapter. Instead, we will focus on a few basic terms and concepts and show how those terms and concepts are used with database processing. Given this introduction, you will then be able to learn more on your own.

BY THE WAY XML Schema validation requires thinking at two meta levels. To understand why, recall that metadata is data about data. The statement *CUSTOMER contains column CustomerLastName Char(25)* is metadata. Extending this idea, the statement *SQL has a data type Char(n) for defining character data of length n* is data about metadata, or meta-metadata.

XML has the same meta levels. An XML document has a structure that is defined by an XML Schema document. The XML Schema document contains metadata, because it is data about the structure of other XML documents. But an XML Schema document has its own structure that is defined by another XML Schema. That XML Schema document is data about metadata, or meta-metadata.

The XML case is elegant. You can write a program to validate an XML document (but don't—use one of the hundreds that already exist). Once you have such a program, you can validate any XML document against its XML Schema document. The process is exactly the same, regardless of whether you are validating an XML document, an XML Schema document, or a document at any other level.

XML Schema Validation

Figure 12-6(a) shows a simple XML Schema document that can be used to represent a single row from the ARTIST table at View Ridge Gallery. The second line indicates what schema is to be used to validate this document. Because this is an XML Schema document, it is to be validated against the mother of all schemas, the one at *www.w3.org*. This same reference will be used in all XML Schemas, in every company, worldwide. (By the way, this reference address is used only for identification purposes. Because this schema is so widely used, most schema validation programs have their own built-in copy of it.)

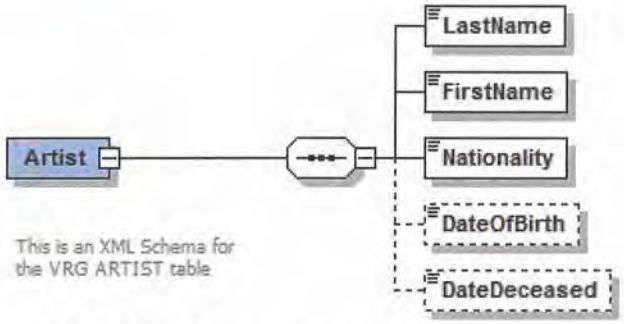
This first statement not only specifies the document that is to be used for validation, it also establishes a labeled namespace. Namespaces are a complicated topic in their own right, and we will not discuss them in this chapter other than to explain the use of labels. In this first statement, the label *xs* is defined by the expression: xmlns:xs. The first part of that expression stands for *xml n*amespace, and the second part defines the label *xs*. Notice that all of the other lines in the document use the label xs. The expression *xs:complexType* simply tells the validating program to look into the namespace called xsd (here, the one specified as *www.w3.org/2001/XMLSchema*) to find the definition of the term *complexType*.

Amazon Ex. 1008.0518

Figure 12-6
Use of an XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="Artist">
        <xs:annotation>
            <xs:documentation>
                This is the XML Schema document the VRG ARTIST table
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="LastName"/>
                <xs:element name="FirstName"/>
                <xs:element name="Nationality"/>
                <xs:element name="DateOfBirth" minOccurs="0"/>
                <xs:element name="DateDeceased" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="ArtStyle"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

(a) XML Schema Document



(b) Graphical Representation of the XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<Artist xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\inetpub\wwwroot\DBP\VRG\VRG-ARTIST-001.xsd" ArtStyle="Modern" >
    <LastName>Miro</LastName>
    <FirstName>Joan</FirstName>
    <Nationality>Spanish</Nationality>
    <DateOfBirth>1893</DateOfBirth>
    <DateDeceased>1983</DateDeceased>
</Artist>
```

(c) Schema-Valid XML Document

The name of the label is up to the designer of the document. You could change xmlns:xs to xmlns:xsd or to xmlns:mylabel, and you would set xsd or mylabel to point to the w3 document. Documents can have multiple namespaces, but that topic is beyond the scope of this discussion.

Elements and Attributes

As shown in Figure 12-6(a), schemas consist of elements and attributes. Elements are either simple or complex. Simple elements have a single data item. In Figure 12-6(a), the elements LastName, FirstName, Nationality, DateOfBirth, and DateDeceased are all simple elements.

Complex elements contain other elements that can be either simple or complex. In Figure 12-6(a), the Artist element is complexType. It contains a sequence of five simple

Amazon Ex. 1008.0519

elements: LastName, FirstName, Nationality, DateOfBirth, and DateDeceased. Later you will see examples of complex types that contain other complex types.

Complex types can have attributes. Figure 12-6(a) defines the attribute ArtStyle. The creator of an XML document uses this attribute to specify a characteristic about an artist; in this case, his or her style. The example document in Figure 12-6(c) specifies the ArtStyle for this artist (Miro) as Modern.

BY THE WAY Elements and attributes both carry data, and you may be wondering when to use one or the other. As a general rule, for database/XML applications, use elements to store data and attributes to store metadata. For example, define an ItemPrice element to store the value of a price, and define a Currency attribute to store the currency type of that price, such as USD, Aus$, or Euros.

The XML standards do not require that elements and attributes be used in this way. It is a matter of style, and in subsequent sections we will show how it is possible to cause SQL Server to place all of the column values in attributes, to place all of them in elements, or to mix them up so that some columns are placed in attributes and others are placed in elements. Thus, these decisions are a matter of design choice rather than XML standard.

By default, the cardinality of both simple and complex elements is 1.1, meaning that a single value is required and no more than a single value can be specified. For the schema in Figure 12-6(a), the minOccurs = "0" expressions indicate that the defaults are being overridden for Birthdate and DeceasedDate so that they need not have a value. This is similar to the NULL constraint in SQL schema definitions.

Figure 12-6(b) shows the XML Schema in a diagram format as drawn by Altova's XMLSpy XML editing tool (see http://www.altova.com/xml-editor/). Being able to see an XML Schema as a diagram often makes it easier to interpret exactly what the XML Schema is specifying. In this diagram, note that required elements (NOT NULL in SQL terms) are indicated with solid lines and box outlines, while optional elements (NULL in SQL terms) are indicated by dashed line and box outlines.

BY THE WAY You can find out more about this excellent product from a small company, which is not yet owned by Microsoft or some other behemoth corporation, from the Web site *www.altova.com*. A 30-day trial version is available. You can process all of the XML code in this chapter using XMLSpy, and the XML code for this chapter is available on this text's Web site at *www.pearsonhighered.com/kroenke*.

Figure 12-6(c) shows an XML document that is valid on the schema shown in Figure 12-6(a). Observe that the value of the ArtStyle attribute is given with the heading of the Artist element. Also note that a namespace of *xsi* is defined. This namespace is used just once—for the noNamespaceSchemaLocation attribute. Do not be concerned about the name of this attribute; it is simply a means of telling the XML parser where to find the XML Schema for this document. Focus your attention on the relationship of the structure of the document and the description in the XML Schema.

Flat Versus Structured Schemas

Figure 12-7 shows an XML Schema and an XML document that represent the columns of the CUSTOMER table in the View Ridge database. As shown in Figure 12-7(a), CustomerID, LastName, and FirstName are required, but all of the other elements are required.

XML Schemas like the one shown in Figure 12-7(a) are sometimes called *flat*, because all of the elements reside at the same level. Figure 12-7(b) shows a diagram of the XML Schema

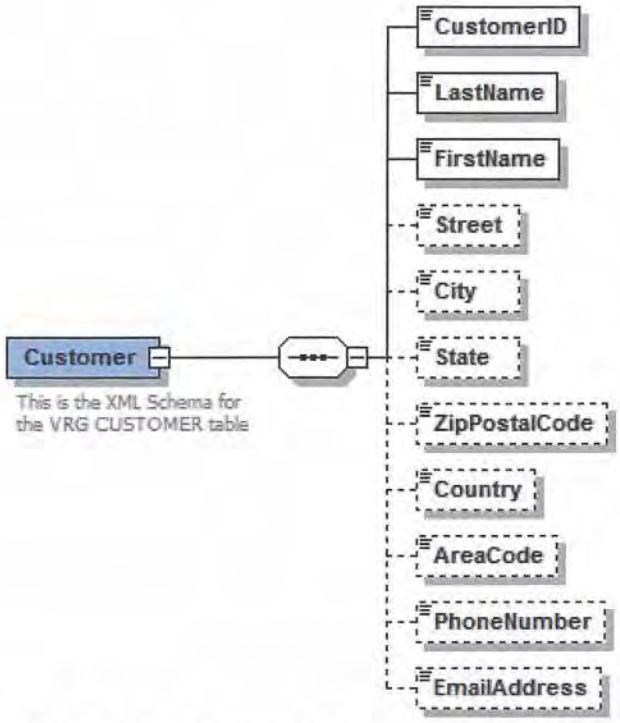
Amazon Ex. 1008.0520

as drawn by XMLSpy, and this diagram graphically depicts why this schema is called flat. The XML document in Figure 12-7(c) contains one of the rows of the CUSTOMER table.

If you think about the elements in Figure 12-7(a) for a moment, you will realize that something about the semantics of them has been left out. In particular, the group {Street, City, State, ZipPostalCode, Country} is part of an Address theme. Also, the group {AreaCode, PhoneNumber} is part of a Phone theme. As you know, in the relational model, all columns are considered equal, and there is no way to represent these themes.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="Customer">
        <xs:annotation>
            <xs:documentation>This is the XML Schema for the VRG CUSTOMER table</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="CustomerID" type="xs:int"/>
                <xs:element name="LastName" type="xs:string"/>
                <xs:element name="FirstName" type="xs:string"/>
                <xs:element name="Street" type="xs:string" minOccurs="0"/>
                <xs:element name="City" type="xs:string" minOccurs="0"/>
                <xs:element name="State" type="xs:string" minOccurs="0"/>
                <xs:element name="ZipPostalCode" type="xs:string" minOccurs="0"/>
                <xs:element name="Country" type="xs:string" minOccurs="0"/>
                <xs:element name="AreaCode" type="xs:string" minOccurs="0"/>
                <xs:element name="PhoneNumber" type="xs:string" minOccurs="0"/>
                <xs:element name="EmailAddress" type="xs:string" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

(a) XML Schema with Flat Structure



(b) Graphical Representation of the XML Schema

Figure 12-7
Example of a Flat Structure

Amazon Ex. 1008.0521

```
<?xml version="1.0" encoding="UTF-8"?>
<Customer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\inetpub\wwwroot\DBP\VRG\DBP-e11-Figure-12-07-A.xsd">
        <CustomerID>1000</CustomerID>
        <LastName>Janes</LastName>
        <FirstName>Jeffery</FirstName>
        <Street>123 W. Elm St.</Street>
        <City>Renton</City>
        <State>WA</State>
        <ZipPostalCode>98055</ZipPostalCode>
        <Country>USA</Country>
        <AreaCode>425</AreaCode>
        <PhoneNumber>543-2345</PhoneNumber>
        <EmailAddress>Jeffery.Janes@somewhere.com</EmailAddress>
</Customer>
```

(c) Schema-Valid XML Document

Figure 12-7 (Continued)

With XML, however, such groups can be modeled. The schema shown in Figure 12-8(a) structures the appropriate columns into an Address complexType and other columns into a Phone complexType. A graphical display of this schema is shown in Figure 12-8(b). An XML document for one of the rows of CUSTOMER expressed in this format is shown in Figure 12-8(c).

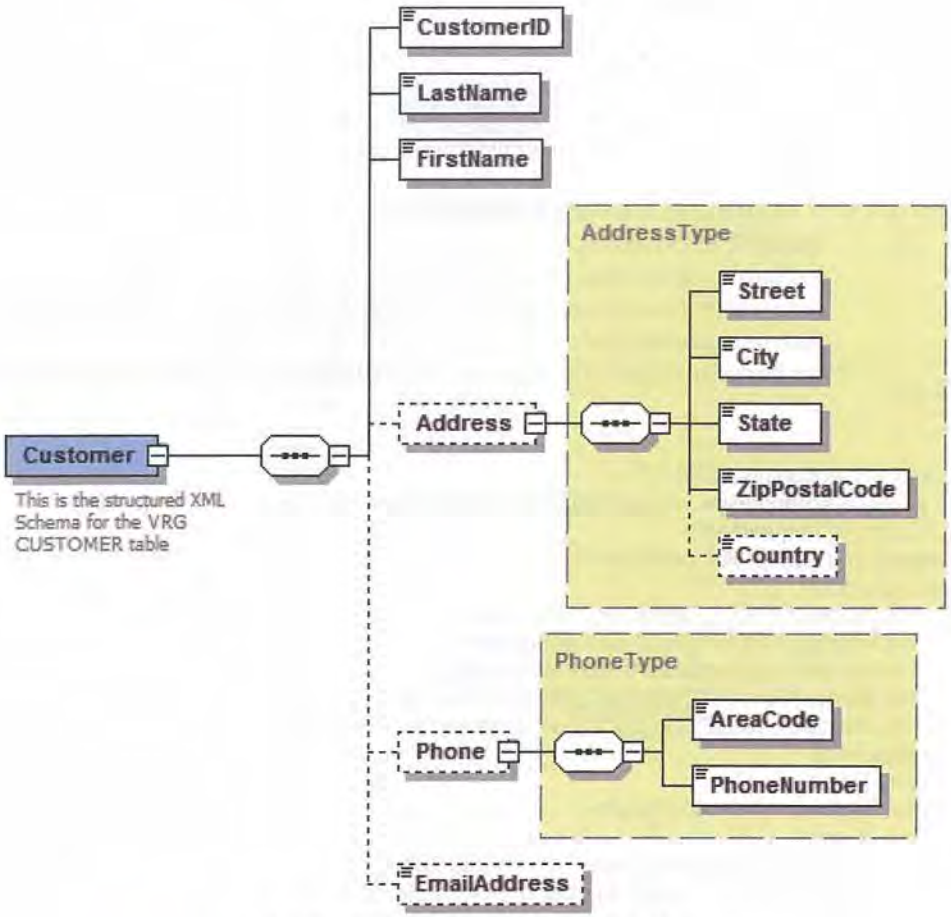
Such schemas are sometimes called **structured schemas** because they add structure to table columns. Such a model captures additional user meaning, so it is superior to the relational model from a descriptive standpoint.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:complexType name="AddressType">
        <xs:sequence>
            <xs:element name="Street" type="xs:string"/>
            <xs:element name="City" type="xs:string"/>
            <xs:element name="State" type="xs:string"/>
            <xs:element name="ZipPostalCode" type="xs:string"/>
            <xs:element name="Country" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PhoneType">
        <xs:sequence>
            <xs:element name="AreaCode" type="xs:string"/>
            <xs:element name="PhoneNumber" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Customer">
        <xs:annotation>
            <xs:documentation>
                This is the structured XML Schema for the VRG CUSTOMER table
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="CustomerID" type="xs:int"/>
                <xs:element name="LastName" type="xs:string"/>
                <xs:element name="FirstName" type="xs:string"/>
                <xs:element name="Address" type="AddressType" minOccurs="0"/>
                <xs:element name="Phone" type="PhoneType" minOccurs="0"/>
                <xs:element name="EmailAddress" type="xs:string" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

(a) Structured XML Schema

Figure 12-8 Example of a Structured Schema

Amazon Ex. 1008.0522



(b) Graphical Representation of the Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<Customer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\inetpub\wwwroot\DBP\VRG\DBP-e11-Figure-12-08-A.xsd">
    <CustomerID>1000</CustomerID>
    <LastName>Janes</LastName>
    <FirstName>Jeffery</FirstName>
    <Address>
        <Street>123 W. Elm St.</Street>
        <City>Renton</City>
        <State>WA</State>
        <ZipPostalCode>98055</ZipPostalCode>
        <Country>USA</Country>
    </Address>
    <Phone>
        <AreaCode>425</AreaCode>
        <PhoneNumber>543-2345</PhoneNumber>
    </Phone>
    <EmailAddress>Jeffery.Janes@somewhere.com</EmailAddress>
</Customer>
```

Figure 12-8 (Continued)

(c) Schema-Valid XML Document

Note that in this structured XML Schema, Address and Phone remain as optional (NULL values allowed) in Customer. This maintains the optional status of these columns in the CUSTOMER table. However, in the Address complexType {Street, City, State, ZipPostalCode) are required (NOT NULL). Similarly, in the Phone complexType, {AreaCode, Phone Number} are required. These conditions can be read as "IF there is Address data for a customer, it MUST include the street address, city, state and zip / postal code," and "IF there is Phone data for a customer, it MUST include both the area code and phone number."

Amazon Ex. 1008.0523

Global Elements

Suppose that we want to use XML Schema to represent a document that extends the customer data in Figure 12-8 to include the salesperson assigned to that customer. Further, suppose that both customers and salespeople have address and phone data. We can use the techniques shown thus far to represent this new customer structure, but, if we do so, we will duplicate the definition of Phone and Address.

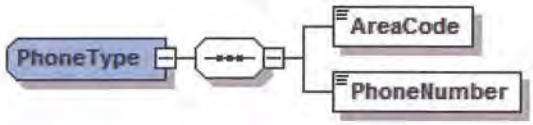
In the relational world, we worry about duplication of data, not so much because of wasted file space, but more because there is always the chance of inconsistent data when one copy of the data is changed and the other copy is not changed. Similarly, in the document-processing world, people worry about the duplicate definition of elements because there is always the chance that they will become inconsistent when one is changed and the other is not. As shown in Figure 12-9, elements can be declared globally and then reused to eliminate the definition duplication.

Figure 12-9
Example of a Schema with Global Elements

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:complexType name="AddressType">
        <xs:sequence>
            <xs:element name="Street" type="xs:string"/>
            <xs:element name="City" type="xs:string"/>
            <xs:element name="State" type="xs:string"/>
            <xs:element name="ZipPostalCode" type="xs:string"/>
            <xs:element name="Country" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PhoneType">
        <xs:sequence>
            <xs:element name="AreaCode" type="xs:string"/>
            <xs:element name="PhoneNumber" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Customer">
        <xs:annotation>
            <xs:documentation>
                This is the structured XML Schema for the VRG database with SALESPERSON added
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="CustomerID" type="xs:int"/>
                <xs:element name="LastName" type="xs:string"/>
                <xs:element name="FirstName" type="xs:string"/>
                <xs:element name="Address" type="AddressType" minOccurs="0"/>
                <xs:element name="Phone" type="PhoneType" minOccurs="0" maxOccurs="3"/>
                <xs:element name="EmailAddress" type="xs:string" minOccurs="0"/>
                <xs:element name="Salesperson">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="SalespersonID" type="xs:int"/>
                            <xs:element name="LastName" type="xs:string"/>
                            <xs:element name="FirstName" type="xs:string"/>
                            <xs:element name="Address" type="AddressType"/>
                            <xs:element name="Phone" type="PhoneType"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

(a) XML Schema with Global PhoneType Element

Amazon Ex. 1008.0524



(b) Graphical Representation of the PhoneType Global Element

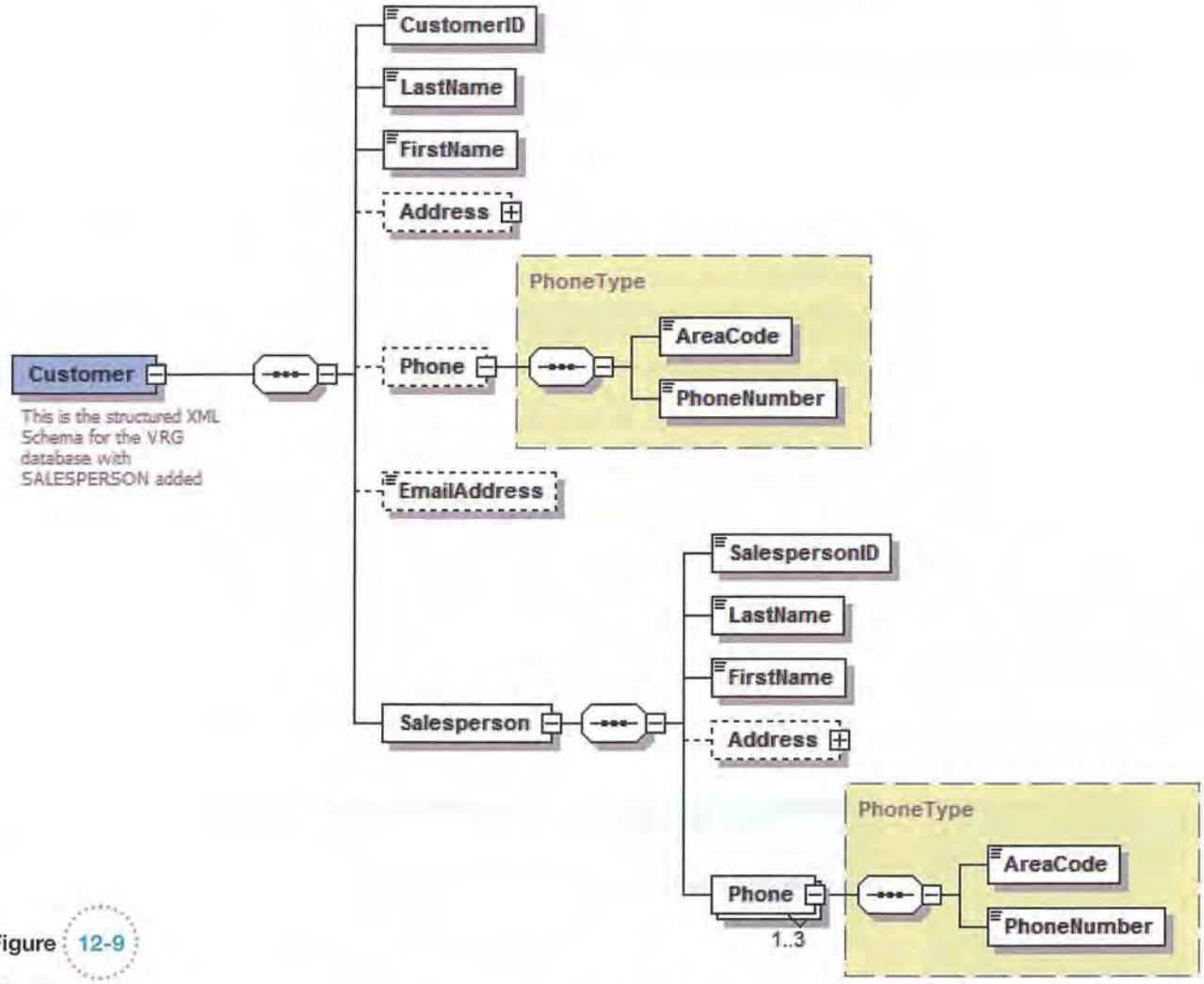


Figure 12-9 (Continued)

(c) Graphical Representation of the XML Schema

In Figure 12-9(a), for example, the Address group is defined as the global element AddressType, and the Phone group is defined as the global element PhoneType. According to the XML Schema standard, these are global elements because they reside at the top level of the schema.

If you examine Figure 12-9(a) further, you will see that both Customer and Salesperson within Customer use the AddressType and PhoneType global definitions. They are referenced by the notations *type = "AddressType"* and *type = "PhoneType."* By using these global definitions, if either PhoneType or AddressType is changed, the definition of Customer and Salesperson will inherit the change.

One other change in this figure is that the cardinality of the Phone group of Salesperson has been set to 1..3. This notation means that at least one Phone group is required and as many as three are allowed. As you learned in Chapter 5, representing such multivalued attributes in the entity-relationship model requires the definition of an ID-dependent entity. That entity will later be transformed into a table in the relational model. We will ignore this issue here. This notation is shown only so that you can see how multivalued elements are documented in an XML Schema.

Amazon Ex. 1008.0525

Figure 12-9(b) shows how XMLSpy graphically represents the PhoneType global element, and Figure 12-9(c) illustrates the way that the PhoneType reference is shown for Customer and Salesperson.

> **BY THE WAY** In the VRG table structure we specified in Chapter 7 and used in Chapter 10, Chapter 10A, and Chapter 10B, there is no SALESPERSON table defined. The XML Schema in Figure 12-9, however, suggests how we could add a SALESPERSON table to the VRG database.

Creating XML Documents from Database Data

Oracle, SQL Server, and MySQL have facilities for generating XML documents from database data. The Oracle XML features require the use of Java. Because we do not assume that you are a Java programmer, we will not discuss those features further in this chapter. If you are a Java programmer, you can learn more about Oracle's XML features at *www.oracle.com*.

The facilities in Oracle, SQL Server, and MySQL are undergoing rapid development. In the case of SQL Server, version 7.0 added the expression FOR XML to SQL SELECT syntax. That expression was carried forward to SQL Server 2000. In 2002, the SQL Server group extended the SQL Server capabilities with the SQLXML class library. SQLXML, which was produced by the SQL Server group, is different from ADO.NET. All of these features and functions were merged together in SQL Server 2005 and are carried forward in SQL Server 2008.

Using the SQL SELECT . . . FOR XML Statement

SQL Server 2008 uses the **SQL SELECT . . . FOR XML statement** to work with XML. Consider the following SQL statement:

```
SELECT    *
FROM      ARTIST
    FOR   XML RAW;
```

The expression FOR XML RAW tells SQL Server to place the values of the columns as attributes in the resulting XML document. Figure 12-10(a) shows an example of as FOR XML RAW query in the Microsoft SQL Server Management Studio. The results of the query are displayed in a single cell.

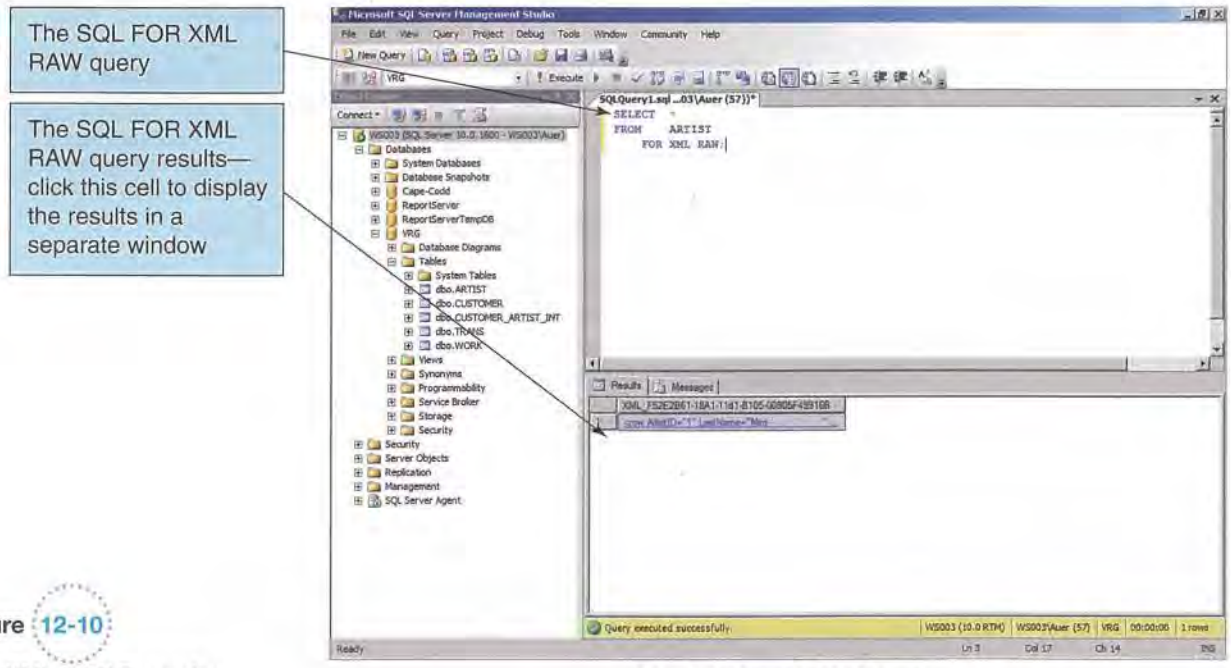
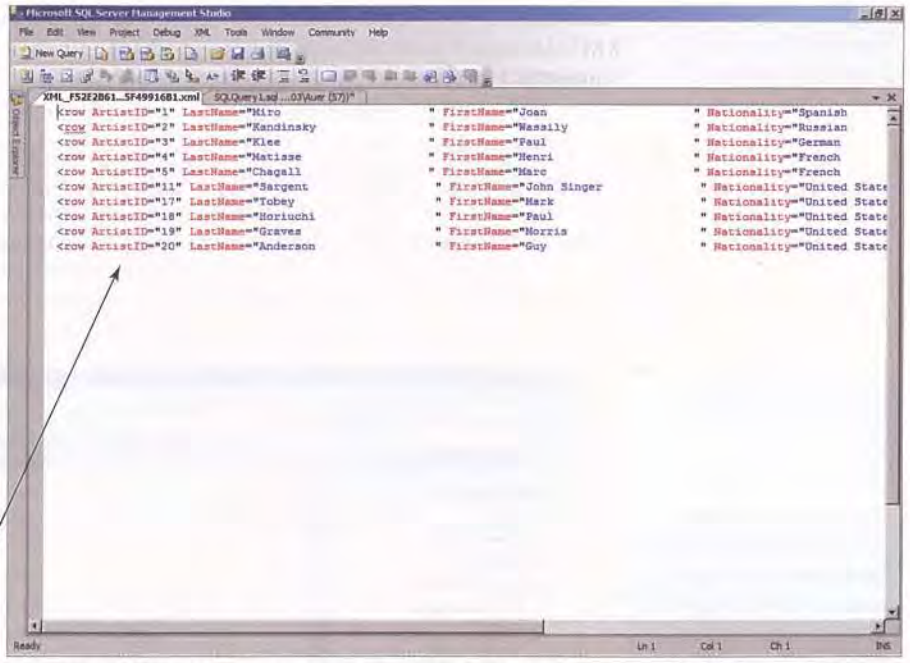


Figure 12-10 FOR XML RAW Examples

(a) FOR XML RAW Query

Amazon Ex. 1008.0526



The SQL FOR XML RAW query results

(b) FOR XML RAW Results in the Microsoft SQL Server Management Studio

```
<row ArtistID="1" LastName="Miro" FirstName="Joan"
        Nationality="Spanish"   DateOfBirth="1893" DateDeceased="1983" />
<row ArtistID="2" LastName="Kandinsky" FirstName="Wassily"
        Nationality="Russian " DateOfBirth="1866" DateDeceased="1944" />
<row ArtistID="3" LastName="Klee" FirstName="Paul"
        Nationality="German" DateOfBirth="1879" DateDeceased="1940" />
<row ArtistID="4" LastName="Matisse" FirstName="Henri"
        Nationality="French" DateOfBirth="1869" DateDeceased="1954" />
<row ArtistID="5" LastName="Chagall" FirstName="Marc"
        Nationality="French" DateOfBirth="1887" DateDeceased="1985" />
<row ArtistID="11" LastName="Sargent" FirstName="John Singer"
        Nationality="United States" DateOfBirth="1856" DateDeceased="1925" />
<row ArtistID="17" LastName="Tobey" FirstName="Mark"
        Nationality="United States" DateOfBirth="1890" DateDeceased="1976" />
<row ArtistID="18" LastName="Horiuchi" FirstName="Paul"
        Nationality="United States" DateOfBirth="1906" DateDeceased="1999" />
<row ArtistID="19" LastName="Graves" FirstName="Morris"
        Nationality="United States" DateOfBirth="1920" DateDeceased="2001" />
<row ArtistID="20" LastName="Anderson" FirstName="Guy"
        Nationality="United States" />
```

(c) FOR XML RAW Results in XML Document

Figure 12-10 (Continued)

Clicking this cell displays the results as shown in Figure 12-10(b). As expected, each column is placed as an attribute of the element named *row*. The complete output, edited as it would appear in an XML document (and with extra spaces in the attribute values removed) is shown in Figure 12-10(c).

It is also possible to cause SQL Server to place the values of the columns into elements rather than attributes by using FOR XML AUTO, ELEMENTS. For example, we can display the data in the ARTIST table using the SQL query:

```
SELECT    *
FROM      ARTIST
    FOR   XML AUTO, ELEMENTS;
```

Figure 12-11(a) shows the query in the Microsoft SQL Server Management Studio, and Figure 12-11(b) shows the full results in a separate tabbed window. Here, each attribute

Amazon Ex. 1008.0527

value appears as a separate element. The complete output, edited as it would appear in an XML document (and with MyData tags added to contain the entire data set) is shown in Figure 12-11(c). The XML Schema for this document is shown in Figure 12-11(d), and the graphical representation of the XML Schema is shown in Figure 12-11(e).

Using another option, FOR XML EXPLICIT, you can cause SQL Server to place some columns into elements and others into attributes. For example, you might decide to place all column values except surrogate key values into elements and all surrogate key values into attributes. The justification for this design is that surrogate key values have no meaning to the users, so they are more



(a) FOR XML AUTO, ELEMENTS Query

(b) FOR XML AUTO, ELEMENTS Results in the Microsoft SQL Server Management Studio

Figure 12-11

FOR XML AUTO, ELEMENTS Examples

Amazon Ex. 1008.0528

```
<?xml version="1.0" encoding="UTF-8"?>
<MyData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\inetpub\wwwroot\DBP\VRG\DBP-e11-Figure-12-11-D.xsd">
    <ARTIST>
        <ArtistID>1</ArtistID>
        <LastName>Miro                </LastName>
        <FirstName>Joan               </FirstName>
        <Nationality>Spanish              </Nationality>
        <DateOfBirth>1893</DateOfBirth>
        <DateDeceased>1983</DateDeceased>
    </ARTIST>
    <ARTIST>
        <ArtistID>2</ArtistID>
        <LastName>Kandinsky           </LastName>
        <FirstName>Wassily            </FirstName>
        <Nationality>Russian              </Nationality>
        <DateOfBirth>1866</DateOfBirth>
        <DateDeceased>1944</DateDeceased>
    </ARTIST>
    <ARTIST>
        <ArtistID>3</ArtistID>
        <LastName>Klee                </LastName>
        <FirstName>Paul               </FirstName>
        <Nationality>German               </Nationality>
        <DateOfBirth>1879</DateOfBirth>
        <DateDeceased>1940</DateDeceased>
    </ARTIST>
    <ARTIST>
        <ArtistID>4</ArtistID>
        <LastName>Matisse             </LastName>
        <FirstName>Henri              </FirstName>
        <Nationality>French               </Nationality>
        <DateOfBirth>1869</DateOfBirth>
        <DateDeceased>1954</DateDeceased>
    </ARTIST>
    <ARTIST>
        <ArtistID>5</ArtistID>
        <LastName>Chagall             </LastName>
        <FirstName>Marc               </FirstName>
        <Nationality>French               </Nationality>
        <DateOfBirth>1887</DateOfBirth>
        <DateDeceased>1985</DateDeceased>
    </ARTIST>
    <ARTIST>
    <ArtistID>11</ArtistID>
        <LastName>Sargent             </LastName>
        <FirstName>John Singer          </FirstName>
        <Nationality>United States            </Nationality>
        <DateOfBirth>1856</DateOfBirth>
        <DateDeceased>1925</DateDeceased>
    </ARTIST>
    <ARTIST>
        <ArtistID>17</ArtistID>
        <LastName>Tobey               </LastName>
        <FirstName>Mark               </FirstName>
        <Nationality>United States            </Nationality>
        <DateOfBirth>1890</DateOfBirth>
        <DateDeceased>1976</DateDeceased>
    </ARTIST>
```

Figure 12-11

(Continued)

Amazon Ex. 1008.0529

```
    <ARTIST>
        <ArtistID>18</ArtistID>
        <LastName>Horiuchi          </LastName>
        <FirstName>Paul             </FirstName>
        <Nationality>United States          </Nationality>
        <DateOfBirth>1906</DateOfBirth>
        <DateDeceased>1999</DateDeceased>
    </ARTIST>
    <ARTIST>
    <ArtistID>19</ArtistID>
        <LastName>Graves            </LastName>
        <FirstName>Morris           </FirstName>
        <Nationality>United States          </Nationality>
        <DateOfBirth>1920</DateOfBirth>
        <DateDeceased>2001</DateDeceased>
    </ARTIST>
    <ARTIST>
        <ArtistID>20</ArtistID>
        <LastName>Anderson          </LastName>
        <FirstName>Guy              </FirstName>
        <Nationality>United States          </Nationality>
    </ARTIST>
</MyData>
```

(c) FOR XML AUTO, ELEMENTS Results in XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="MyData">
        <xs:annotation>
            <xs:documentation>
                This is the XML Schema document for the VRG ARTIST table
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ARTIST" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="ArtistID" type="xs:int"/>
                            <xs:element name="LastName"/>
                            <xs:element name="FirstName"/>
                            <xs:element name="Nationality"/>
                            <xs:element name="DateOfBirth" minOccurs="0"/>
                            <xs:element name="DateDeceased" minOccurs="0"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

(d) XML Schema

Figure 12-11
(Continued)

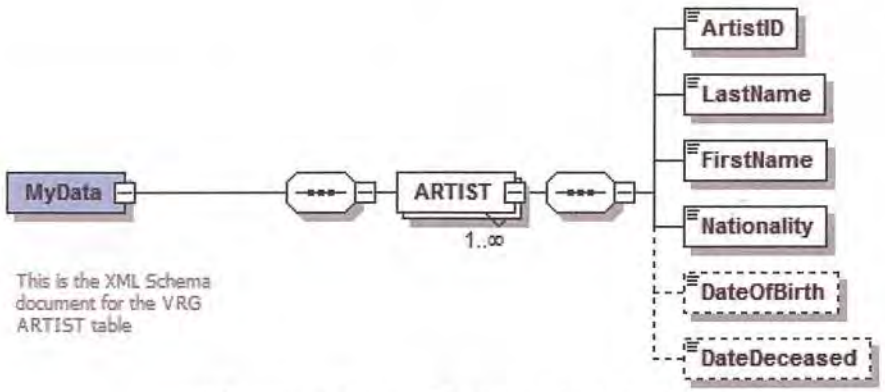
like metadata than data. The means by which this is done is beyond the scope of this discussion. See FOR XML EXPLICIT in the SQL Server documentation for more information.

Multitable SELECT with FOR XML

FOR XML SELECT statements are not limited to single-table SELECTs—they can be applied to joins as well. For example, we can use the following SQL join query to produce a list of VRG customers and the artists they are interesting in:

Amazon Ex. 1008.0530

Figure 12-11
(Continued)



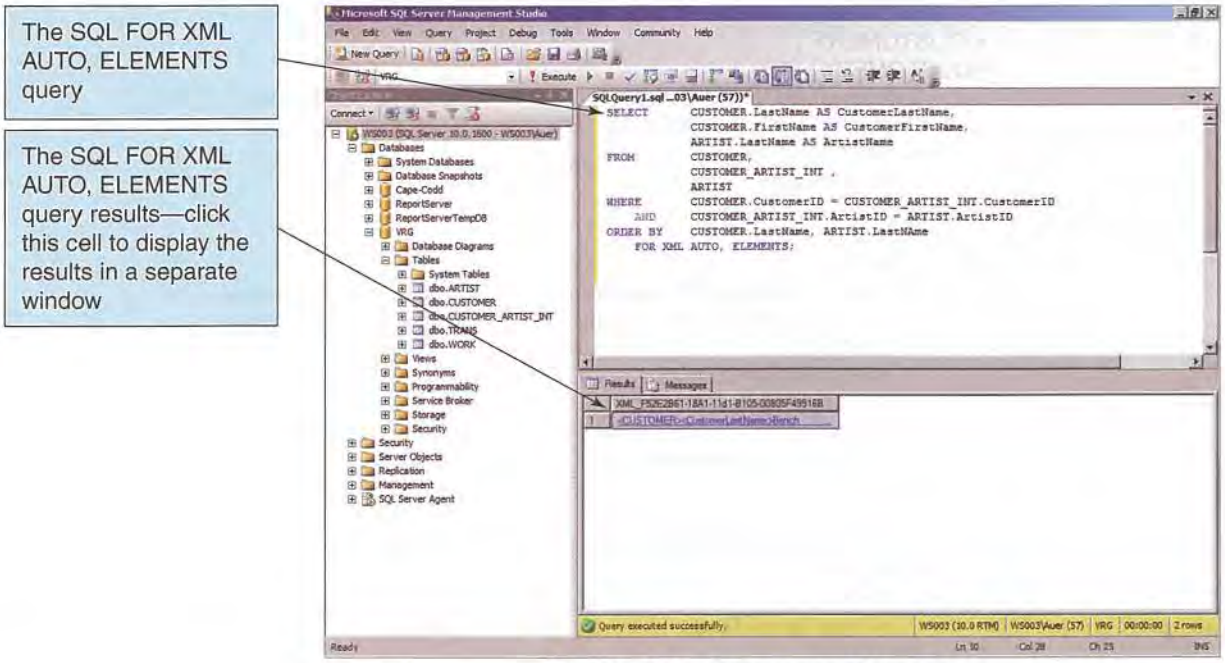
(e) Graphical Representation of the XML Schema

```
SELECT      CUSTOMER.LastName AS CustomerLastName,
            CUSTOMER.FirstName AS CustomerFirstName,
            ARTIST.LastName AS ArtistName
FROM        CUSTOMER,
            CUSTOMER_ARTIST_INT ,
            ARTIST
WHERE       CUSTOMER.CustomerID = CUSTOMER_ARTIST_INT.CustomerID
    AND     CUSTOMER_ARTIST_INT.ArtistID = ARTIST.ArtistID
ORDER BY    CUSTOMER.LastName, ARTIST.LastNAme
    FOR     XML AUTO, ELEMENTS;
```

Figure 12-12
FOR XML AUTO, ELEMENTS Example Displaying Customer and Artist Interests

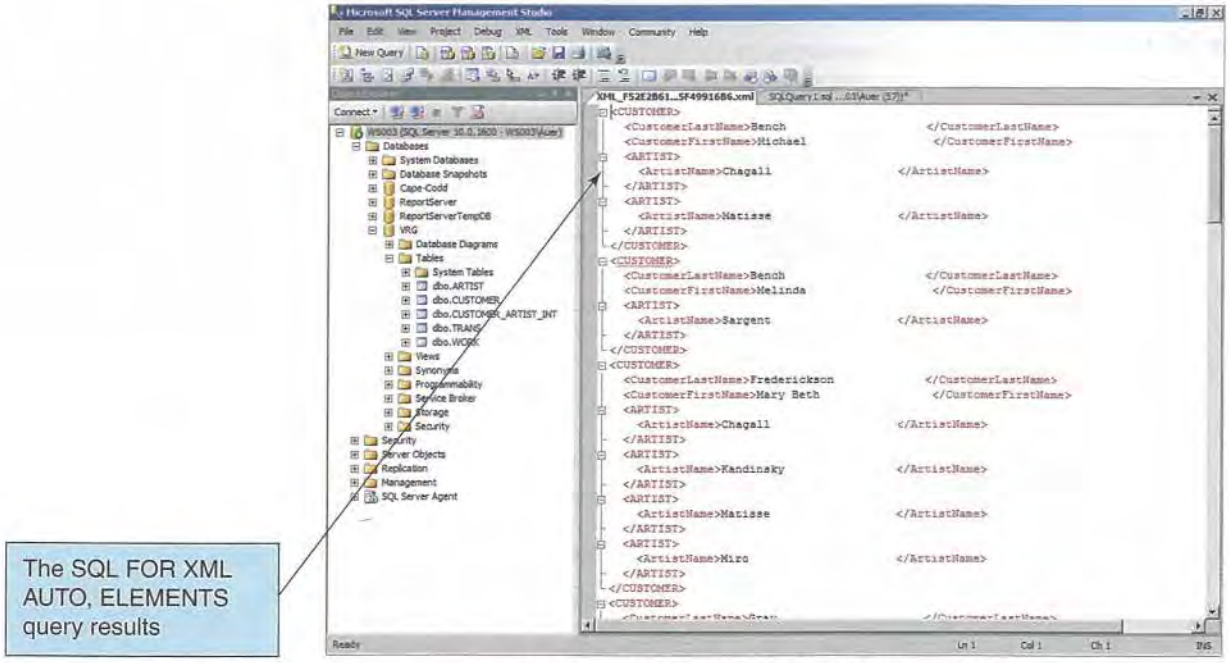
Figure 12-12(a) shows the query in the Microsoft SQL Server Management Studio, and Figure 12-12(b) shows the full results in a tabbed window. Figure 12-12(c) shows results in an XML document.

SQL Server uses the order of the tables in the FROM clause to determine the hierarchical placement of the elements in the generated XML document. Here, the top-level element is



(a) FOR XML AUTO, ELEMENTS Query

Amazon Ex. 1008.0531



(b) FOR XML AUTO, ELEMENTS Results in the Microsoft SQL Server Management Studio

```
<?xml version="1.0" encoding="UTF-8"?>
<MyData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\inetpub\wwwroot\DBP\VRG\DBP-e11-Figure-12-13-A.xsd">
    <CUSTOMER>
        <CustomerLastName>Bench          </CustomerLastName>
        <CustomerFirstName>Michael       </CustomerFirstName>
        <ARTIST>
            <ArtistName>Chagall          </ArtistName>
        </ARTIST>
        <ARTIST>
            <ArtistName>Matisse          </ArtistName>
        </ARTIST>
    </CUSTOMER>
    <CUSTOMER>
        <CustomerLastName>Bench          </CustomerLastName>
        <CustomerFirstName>Melinda       </CustomerFirstName>
        <ARTIST>
            <ArtistName>Sargent          </ArtistName>
        </ARTIST>
    </CUSTOMER>
    <CUSTOMER>
        <CustomerLastName>Frederickson   </CustomerLastName>
        <CustomerFirstName>Mary Beth     </CustomerFirstName>
        <ARTIST>
            <ArtistName>Chagall          </ArtistName>
        </ARTIST>
        <ARTIST>
            <ArtistName>Kandinsky        </ArtistName>
        </ARTIST>
        <ARTIST>
            <ArtistName>Matisse          </ArtistName>
        </ARTIST>
        <ARTIST>
            <ArtistName>Miro             </ArtistName>
        </ARTIST>
    </CUSTOMER>
```

Figure 12-12 (Continued)

Amazon Ex. 1008.0532

```
  <CUSTOMER>
      <CustomerLastName>Gray          </CustomerLastName>
      <CustomerFirstName>Donald        </CustomerFirstName>
      <ARTIST>
          <ArtistName>Graves          </ArtistName>
      </ARTIST>
      <ARTIST>
          <ArtistName>Horiuchi        </ArtistName>
      </ARTIST>
      <ARTIST>
          <ArtistName>Tobey           </ArtistName>
      </ARTIST>
  </CUSTOMER>
<CUSTOMER>
    <CustomerLastName>Janes           </CustomerLastName>
    <CustomerFirstName>Jeffrey        </CustomerFirstName>
    <ARTIST>
        <ArtistName>Graves            </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Horiuchi          </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Tobey             </ArtistName>
    </ARTIST>
</CUSTOMER>
<CUSTOMER>
    <CustomerLastName>Smathers        </CustomerLastName>
    <CustomerFirstName>Fred           </CustomerFirstName>
    <ARTIST>
        <ArtistName>Graves            </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Horiuchi          </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Tobey             </ArtistName>
    </ARTIST>
</CUSTOMER>
<CUSTOMER>
    <CustomerLastName>Smith           </CustomerLastName>
    <CustomerFirstName>David          </CustomerFirstName>
    <ARTIST>
        <ArtistName>Chagall           </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Kandinsky         </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Matisse           </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Miro              </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Sargent           </ArtistName>
    </ARTIST>
</CUSTOMER>
```

Figure 12-12
(Continued)

Amazon Ex. 1008.0533

```
<CUSTOMER>
    <CustomerLastName>Twilight          </CustomerLastName>
    <CustomerFirstName>Tiffany          </CustomerFirstName>
    <ARTIST>
        <ArtistName>Graves          </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Horiuchi        </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Sargent         </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Tobey           </ArtistName>
    </ARTIST>
</CUSTOMER>
<CUSTOMER>
    <CustomerLastName>Warning           </CustomerLastName>
    <CustomerFirstName>Selma            </CustomerFirstName>
    <ARTIST>
        <ArtistName>Chagall         </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Graves          </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Sargent         </ArtistName>
    </ARTIST>
</CUSTOMER>
<CUSTOMER>
    <CustomerLastName>Wilkens           </CustomerLastName>
    <CustomerFirstName>Chris            </CustomerFirstName>
    <ARTIST>
        <ArtistName>Graves          </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Horiuchi        </ArtistName>
    </ARTIST>
    <ARTIST>
        <ArtistName>Tobey           </ArtistName>
    </ARTIST>
</CUSTOMER>
</MyData>
```

(c) FOR XML AUTO, ELEMENTS Results in XML Document

Figure 12-12 (Continued)

CUSTOMER, and the next element is ARTIST. The CUSTOMER_ARTIST_INT table does not appear in the generated document because no column from that table appeared in the SQL SELECT statement.

You can write the expression FOR XML AUTO, XMLDATA to cause SQL Server to produce an XML Schema statement in front of the XML document that it writes. The schema produced, however, involves topics that we will not cover in this chapter, so we will not do that.

An XML Schema for the XML document shown in Figure 12-12(c) is shown in Figure 12-13(a), and Figure 12-13(b) shows the graphical display of this same schema. Observe that the MyData element can have an unbounded number of CUSTOMER elements, and each CUSTOMER can have an unbounded number of ARTIST elements, one for each artist interest. In this figure, the notation 1..∞ means that at least one CUSTOMER is required and an unlimited number will be allowed.

An XML Schema for All CUSTOMER Purchases

Suppose now that we want to produce a document that has all of the customer purchase data. To do that, we need to join CUSTOMER to TRANS to WORK to ARTIST and select the appropriate data. The following SQL statement produces the required data:

Amazon Ex. 1008.0534

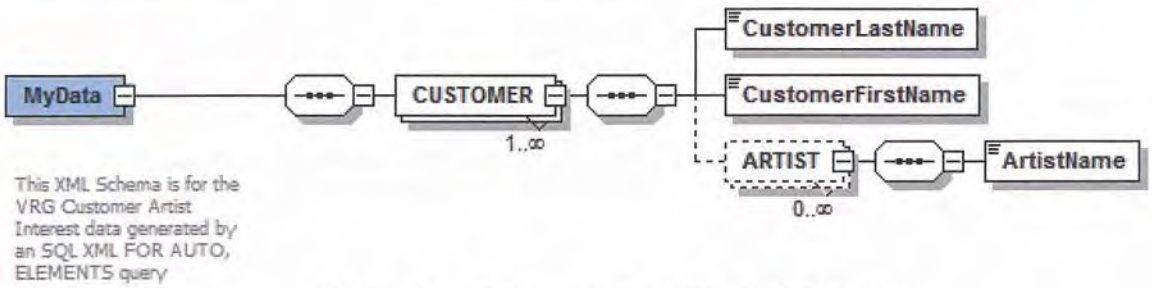
```
SELECT      CUSTOMER.CustomerID,
            CUSTOMER.LastName AS CustomerLastName,
            CUSTOMER.FirstName AS CustomerFirstName,
            TRANS.TransactionID, SalesPrice,
            WORK.WorkID, Title, Copy,
            ARTIST.ArtistID, ARTIST.LastName AS ArtistName
FROM        CUSTOMER, TRANS, [WORK], ARTIST
WHERE       CUSTOMER.CustomerID = TRANS.CustomerID
    AND     TRANS.WorkID = WORK.WorkID
    AND     WORK.ArtistID = ARTIST.ArtistID
ORDER BY    CUSTOMER.LastName, ARTIST.LastName
    FOR     XML AUTO, ELEMENTS;
```

Figure 12-13

Customer and Artist Interests

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="MyData">
        <xs:annotation>
            <xs:documentation>
                This XML Schema is for the VRG Customer Artist Interest data
                generated by an SQL XML FOR AUTO, ELEMENTS query
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="CUSTOMER" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="CustomerLastName"/>
                            <xs:element name="CustomerFirstName"/>
                            <xs:element name="ARTIST" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="ArtistName"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

(a) XML Schema



(b) Graphical Representation of the XML Schema

Amazon Ex. 1008.0535

Figure 12-14(a) shows the query in the Microsoft SQL Server Management Studio, and Figure 12-14(b) shows the full results in a tabbed window. Figure 12-14(c) shows *partial* results in an XML document (this is a very long xml document).

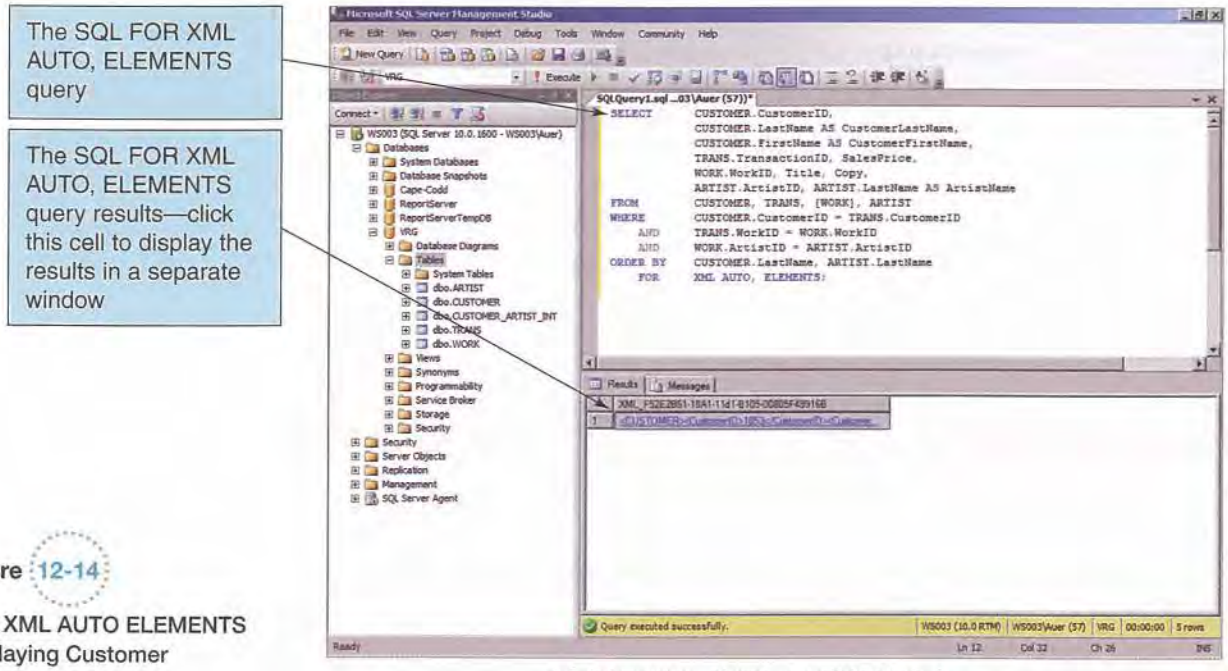
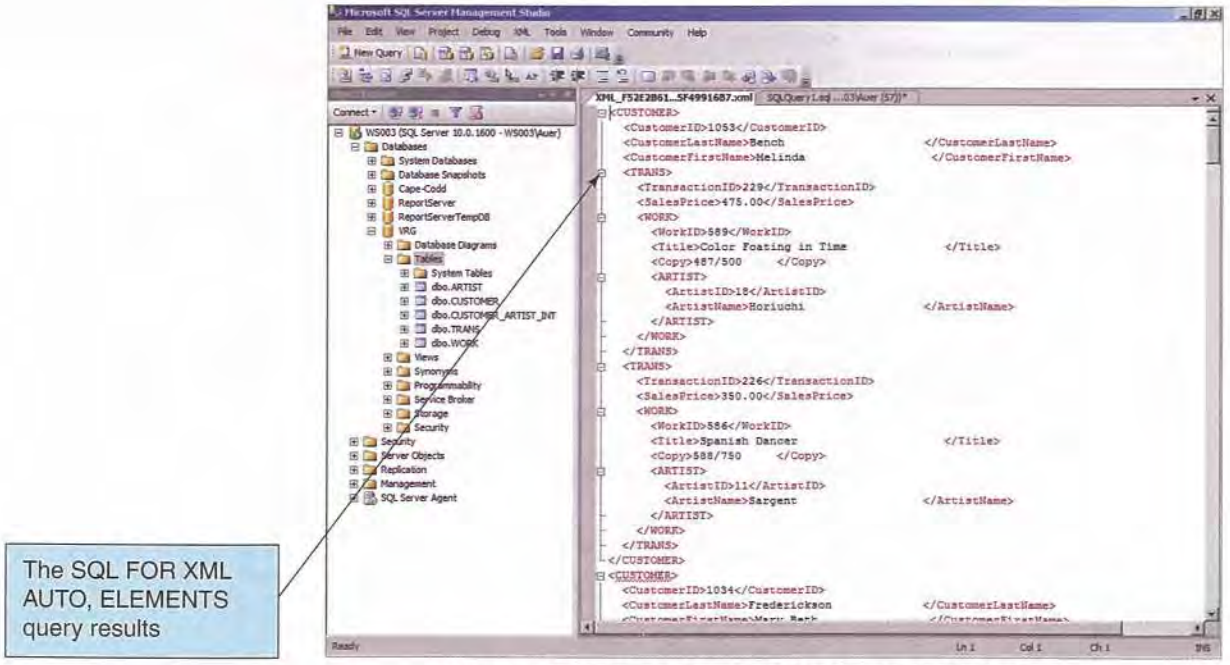


Figure 12-14
FOR XML AUTO ELEMENTS Displaying Customer Purchases

(a) FOR XML AUTO, ELEMENTS Query



(b) FOR XML AUTO, ELEMENTS Results in the Microsoft SQL Server Management Studio

Amazon Ex. 1008.0536

```
<?xml version="1.0" encoding="UTF-8"?>
<MyData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\inetpub\wwwroot\DBP\VRG\DBP-e11-Figure-12-15-A.xsd">
    <CUSTOMER>
        <CustomerID>1053</CustomerID>
        <CustomerLastName>Bench          </CustomerLastName>
        <CustomerFirstName>Melinda       </CustomerFirstName>
        <TRANS>
            <TransactionID>229</TransactionID>
            <SalesPrice>475.00</SalesPrice>
            <WORK>
                <WorkID>589</WorkID>
                <Title>Color Foating in Time      </Title>
                <Copy>487/500   </Copy>
                <ARTIST>
                    <ArtistID>18</ArtistID>
                    <ArtistName>Horiuchi          </ArtistName>
                </ARTIST>
            </WORK>
        </TRANS>
        <TRANS>
            <TransactionID>226</TransactionID>
            <SalesPrice>350.00</SalesPrice>
            <WORK>
                <WorkID>586</WorkID>
                <Title>Spanish Dancer          </Title>
                <Copy>588/750   </Copy>
                <ARTIST>
                    <ArtistID>11</ArtistID>
                    <ArtistName>Sargent           </ArtistName>
                </ARTIST>
            </WORK>
        </TRANS>
    </CUSTOMER>
    <CUSTOMER>
        <CustomerID>1034</CustomerID>
        <CustomerLastName>Frederickson    </CustomerLastName>
        <CustomerFirstName>Mary Beth      </CustomerFirstName>
        <TRANS>
            <TransactionID>105</TransactionID>
            <SalesPrice>400.00</SalesPrice>
            <WORK>
                <WorkID>524</WorkID>
                <Title>Woman with a Hat       </Title>
                <Copy>596/750   </Copy>
                <ARTIST>
                    <ArtistID>4</ArtistID>
                    <ArtistName>Matisse           </ArtistName>
                </ARTIST>
            </WORK>
        </TRANS>
        <TRANS>
        <TransactionID>127</TransactionID>
        <SalesPrice>400.00</SalesPrice>
        <WORK>
            <WorkID>553</WorkID>
            <Title>The Dance          </Title>
            <Copy>734/1000   </Copy>
            <ARTIST>
                <ArtistID>4</ArtistID>
                <ArtistName>Matisse           </ArtistName>
            </ARTIST>
        </WORK>
    </TRANS>
```

Figure 12-14 (Continued)

Amazon Ex. 1008.0537

```
<TRANS>
    <TransactionID>103</TransactionID>
    <SalesPrice>400.00</SalesPrice>
    <WORK>
        <WorkID>522</WorkID>
        <Title>La Lecon de Ski          </Title>
        <Copy>353/500    </Copy>
        <ARTIST>
            <ArtistID>1</ArtistID>
            <ArtistName>Miro            </ArtistName>
        </ARTIST>
    </WORK>
</TRANS>
</CUSTOMER>
```

Figure 12-14 (Continued)

(c) FOR XML AUTO, ELEMENTS Partial Results in XML Document

Figure 12-15(a) shows an XML Schema document for this SQL statement, and a graphical view of it is shown in Figure 12-15(b). According to the XML Schema in Figure 12-15(b), a CUSTOMER has from zero to unlimited TRANS elements, a TRANS element has exactly one WORK element, and a WORK element has exactly one ARTIST element.

A Schema with Two Multivalue Paths

Suppose now that we want to construct an XML document that has all of the View Ridge Gallery customer data. We cannot construct such a view from a single SQL statement because it has two multivalued paths. We need one SQL statement to obtain all of the customer purchase data and a second SQL statement to obtain all of the customer/artist interests.

XML Schema does not have this limitation, however. An XML document may have as many multivalued paths as the application requires. In our case, all we need to do is to combine the schemas in Figure 12-13(a) and Figure 12-15(a). While we're at it, we can also add the surrogate keys for each of the underlying tables.

The result of combining these results (using cut and paste in XMLSpy!) is shown in Figure 12-16. Observe that in Figure 12-16(b) MyData may have from one to an unlimited number of CUSTOMER elements, and that each such element may have from zero to many TRANS and from zero to many ArtistInterests elements. All of the simple elements in this schema are required.

Why Is XML Important?

At this point, you should have some idea of the nature of XML and the XML standards. You know that XML makes a clear separation between structure, content, and materialization. Structure is defined by either a DTD or an XML Schema document. Content is expressed in an XML document, and the materializations of a document are expressed in an XSL document. You also understand that SQL statements can be used to create XML documents, but only as long as those documents involve at most one multivalue path. If more than one such path exists in the document, multiple SQL statements need to be issued to fill the document in some fashion.

You may be asking, "These are interesting ideas, but why do they matter? What's so important about all of this?" The answer to these questions is that XML processing provides a standardized facility to describe, validate, and materialize any database view.

Consider the View Ridge Gallery. Suppose that the gallery wants to share all of its customer data with another gallery, maybe because of a joint sales program. If both galleries agree on an XML Schema like the one shown in Figure 12-17, they can prepare customer data documents in accordance with that schema. Before sending a document, they can run an automated process to validate the document against the schema. In this way, only correct data are transmitted. Of course, this process works in both directions. Not only can View Ridge ensure that it is sending only valid documents; by validating the documents it

Amazon Ex. 1008.0538

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="MyData">
        <xs:annotation>
            <xs:documentation>
                This XML Schema is for the VRG Customer Purchase data generated
                by an SQL XML FOR AUTO, ELEMENTS query
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="CUSTOMER" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="CustomerID" type="xs:int"/>
                            <xs:element name="CustomerLastName"/>
                            <xs:element name="CustomerFirstName"/>
                            <xs:element name="TRANS" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="TransactionID" type="xs:int"/>
                                        <xs:element name="SalesPrice"/>
                                        <xs:element name="WORK">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="WorkID" type="xs:int"/>
                                                    <xs:element name="Title"/>
                                                    <xs:element name="Copy"/>
                                                    <xs:element name="ARTIST">
                                                        <xs:complexType>
                                                            <xs:sequence>
                                                                <xs:element name="ArtistID" type="xs:int"/>
                                                                <xs:element name="ArtistName"/>
                                                            </xs:sequence>
                                                        </xs:complexType>
                                                    </xs:element>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

(a) XML Schema

Figure 12-15
Customer Purchases

receives it can ensure that it is receiving only valid documents. Best of all, the programs for document validation are publicly available and free to the galleries. The galleries do not need to write program code for validation.

Additionally, each gallery can develop its own set of XSL documents to materialize the customer data documents in whatever ways they want. View Ridge can develop one XSL document to display the data on a customer's computer, another to display it on salespersons' computers, another to display it on mobile devices when art buyers are on the road, and so forth. Given these XSLs, customer data can be displayed regardless of whether it came from one gallery or the other.

Now, broaden this idea from two small businesses to an industry. Suppose, for example, that the real estate industry agrees on an XML Schema document for property listings. Every real

Amazon Ex. 1008.0539

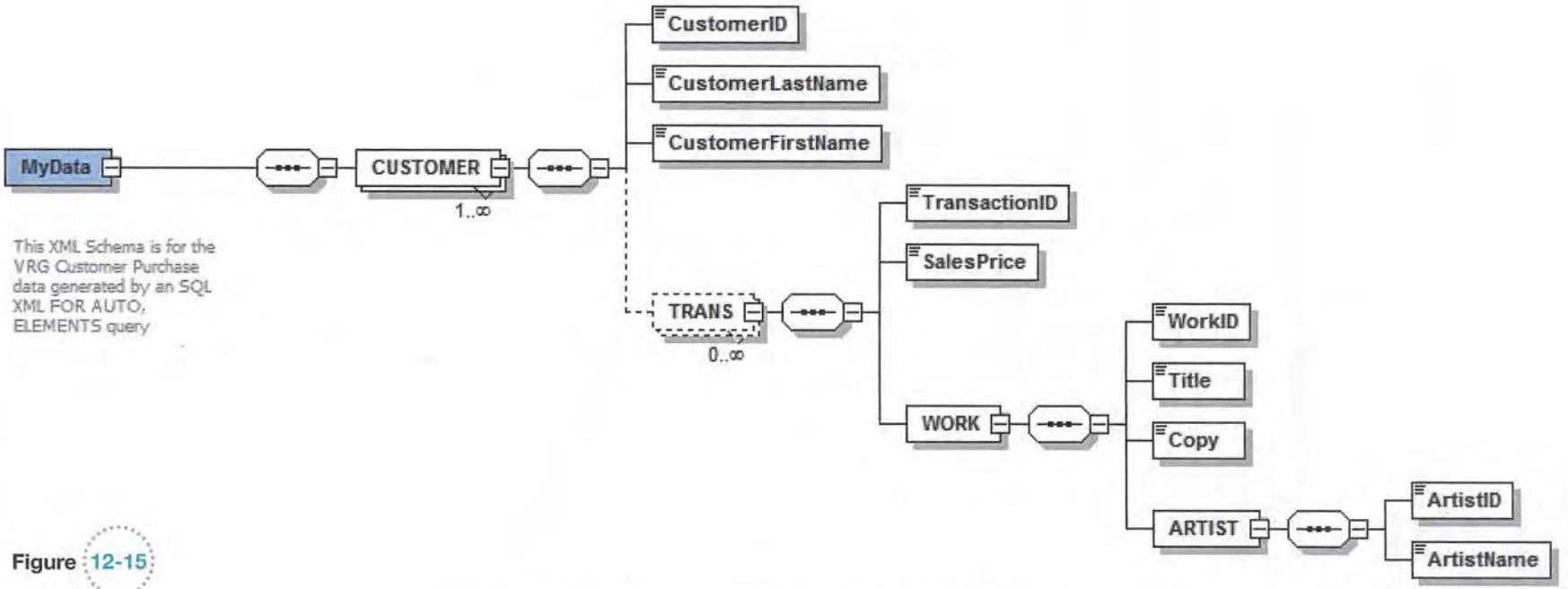


Figure 12-15 (Continued)

(b) Graphical Representation of the XML Schema

Amazon Ex. 1008.0540

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 rel. 3 sp2 (http://www.altova.com) by David Auer (David J. Auer Consultation and
Training) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="MyData">
        <xs:annotation>
            <xs:documentation>
                This XML Schema is for VRG Customer data with two multivalue paths
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="CUSTOMER" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="CustomerID" type="xs:int"/>
                            <xs:element name="LastName"/>
                            <xs:element name="FirstName"/>
                            <xs:element name="TRANS" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="TransactionID" type="xs:int"/>
                                        <xs:element name="SalesPrice"/>
                                        <xs:element name="WORK">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="WorkID" type="xs:int"/>
                                                    <xs:element name="Title"/>
                                                    <xs:element name="Copy"/>
                                                    <xs:element name="ARTIST">
                                                        <xs:complexType>
                                                            <xs:sequence>
                                                                <xs:element name="ArtistID" type="xs:int"/>
                                                                <xs:element name="LastName"/>
                                                                <xs:element name="FirstName"/>
                                                            </xs:sequence>
                                                        </xs:complexType>
                                                    </xs:element>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="ArtistInterests" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="ArtistID" type="xs:int"/>
                                        <xs:element name="LastName"/>
                                        <xs:element name="LastName"/>
                                        <xs:element name="Nationality"/>
                                        <xs:element name="DateOfBirth"" minOccurs="0"/>
                                        <xs:element name="DateDeceased"" minOccurs="0"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

(a) XML Schema

Figure 12-16

View Ridge Gallery Customer View with Two Multivalue Paths

Amazon Ex. 1008.0541

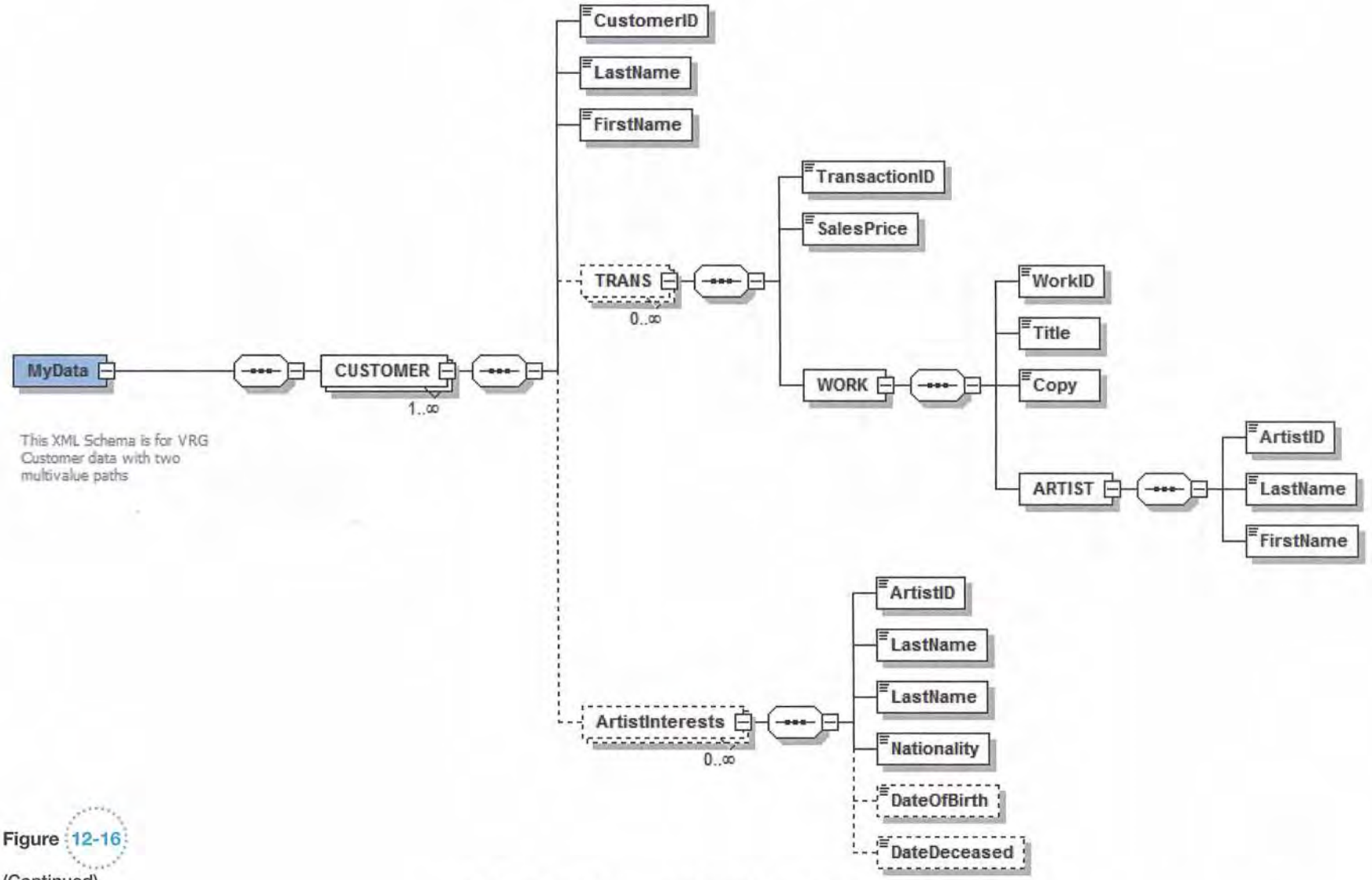


Figure 12-16
(Continued)

(b) Graphical Representation of the XML Schema

Amazon Ex. 1008.0542

Industry Type	Example Standards
Accounting	• American Institute of Certified Public Accountants (AICPA): Extensible Financial Reporting Markup Language (XFRML) [OASIS Cover page] • Open Applications Group, Inc (OAG)
Architecture and Construction	• Architecture, Engineering, and Construction XML Working Group (aecXML Working Group) • ConSource.com: Construction Manufacturing and Distribution Extensible Markup Language (cmdXML)
Automotive	• Automotive Industry Action Group (AIAG) • Global Automedia • MSR: Standards for Information Exchange in the Engineering Process (MEDOC) • The Society of Automotive Engineers (SAE): XML for the Automotive Industry–SAE J2008 [OASIS Cover page] • Open Applications Group, Inc (OAG)
Banking	• Banking Industry Technology Secretariat (BITS): [OASIS Cover page] • Financial Services Technology Consortium (FSTC): Bank Internet Payment System (BIPS) [OASIS Cover page] • Open Applications Group, Inc (OAG)
Electronic Data Interchange	• Data Interchange Standards Association (DISA): [OASIS Cover page] • EEMA EDI/EC Work Group[OASIS Cover page] • European Committee for Standardization/Information Society Standardization System (CEN/ISSS; The European XML/EDI Pilot Project) [OASIS Cover page] • XML/EDI Group [OASIS Cover page]
Human Resources	• DataMain: Human Resources Markup Language (hrml) • HR-XML Consortium [OASIS Cover page]: JobPosting, CandidateProfile, Resume • Open Applications Group (OAG): Open Applications Group Interface Specification (OASIS) [OASIS Cover page] • Tapestry.Net: JOB Markup Language (JOB) • Open Applications Group, Inc (OAG)
Insurance	• ACORD: Property and Casualty [OASIS Cover page], Life (XMLife) [OASIS Cover page] • Lexica: iLingo

Figure 12-17

Example XML Industry Standards

estate company that can produce data in the format of the schema can then exchange listings with every other real estate company. Given the schema, each company can ensure that it is transmitting valid documents, and it can also ensure that it is receiving valid documents. Further, each company can develop its own set of XSL documents to materialize property listings in whatever way it wants. Once the XSL documents have been prepared, any listing from any participating agent can be displayed in the local agency's materializations. Figure 12-17 lists some XML standards work that is under way in various industries.

For another example, consider business-to-business e-commerce. Suppose that Wal-Mart wants to send orders to its vendors in a particular standardized format and that it wants to receive shipment responses to those orders in another particular standardized format. To do this, Wal-Mart can develop an XML Schema for Order documents and another for Shipment documents. It can then publish those XML Schemas on a Web site accessible to its vendors. In this way, all vendors can determine how they will receive orders from Wal-Mart and how they should send their Shipment notifications back.

Amazon Ex. 1008.0543

Industry Type	Example Standards
Real Estate	• OpenMLS: Real Estate Listing Management System (OpenMLS) [OASIS Cover page] • Real Estate Transaction Standard working group (RETS): Real Estate Transaction Standard (RETS) [OASIS Cover page]
Software	• IBM: [OASIS Cover page] • Flashline.com: Software Component Documentation DTD • Flashline.com • INRIA: Koala Bean Markup Language (KBML) [OASIS Cover page] • Marimba and Microsoft: Open Software Description Format (OSD) [OASIS Cover page] • Object Management Group (OMG) [OASIS Cover page]
Workflow	• Internet Engineering Task Force (IETF): Simple Workflow Access Protocol (SWAP) [OASIS Cover page] • Workflow Management Coalition (MfMC): Wf-XML [OASIS Cover page]

Figure 12-17
(Continued)

The schemas can be used by Wal-Mart and all of its vendors to ensure that they are sending and receiving only valid XML documents. Further, Wal-Mart can develop XSL documents to cause the Order and Shipment documents to be transformed into the specific formats needed by its accounting, operations, marketing, and general management departments.

These XSL documents work for any Order or Shipment from any of its vendors. In all of these cases, once the XML Schema documents have been prepared and the XSL documents have been written, all validation and materialization is done via automated processes. Thus, there is no need for any human to touch the Order document between its origination at Wal-Mart and the picking of the inventory at the supplier.

So, the only challenge that remains is to populate the XML documents with database data in accordance with the relevant XML Schema. SQL can be used to populate schemas that have only one multivalue path, but this is too restrictive, and newer technologies like ADO.NET ease the transformation of database data into XML documents. SQL can also be used to ease the transformation of XML documents into database data.

Additional XML Standards

As you know, XML was developed as a series of standards. So far, we have mentioned XML, XSL, XSLT, and XML Schema. There are a number of other XML standards that you may encounter, and some of these are listed in Figure 12-18. You can find the standards, their documentation, and some tutorials on the *www.w3.org* and *www.xml.org* Web sites.

In addition to the four standards discussed in this chapter, XPath is a standard for addressing elements within documents. In Figure 12-4, expressions like

```
<xsl: value-of-select = "name/lastname"/>
```

use XPath to locate a particular element in the document. XPath includes concepts from another standard, XPointer, which was developed to provide a sophisticated means for documents to reference elements in other documents.

SAX and *DOM* refer to different methods of parsing XML documents. The process of parsing consists of reading a document, breaking it into components, and responding to those components in some way—perhaps storing them into a database. XML parsers also validate documents against the DTDs and XML Schemas.

To use the SAX API, a program that is working on an XML document—an XSLT processor, for example—invokes the SAX-compliant parser and passes it the name of the document to

Amazon Ex. 1008.0544

Standard	Description
XML	Extensible Markup Language. A document markup language that started the following:
XSL	XSLT Stylesheet. The document that provides the {match, action} pairs and other data for XSLT to use when transforming an XML document.
XSLT	A program (or process) that applies XSLT Stylesheets to an XML document to produce a transformed XML document.
XML Schema	An XML-compliant language for constraining the structure of an XML document. Extends and replaces DTDs. Under development and *very* important to database processing.
XPath	A sublanguage within XSLT that is used to identify parts of an XML document to be transformed. Can also be used for calculations and string manipulation. Comingled with XSLT.
XPointer	A standard for linking one document to another. XPath has many elements from XPointer.
SAX	Simple API (application program interface) for XML. An event-based parser that notifies a program when the elements of an XML document have been encountered during document parsing.
DOM	Document Object Model. An API that represents an XML document as a tree. Each node of the tree represents a piece of the XML document. A program can directly access and manipulate a node of the DOM representation.
XQuery	A standard for expressing database queries as XML documents. The structure of the query uses XPath facilities, and the result of the query is represented in an XML format. Under development and likely to be important in the future.
XML Namespaces	A standard for allocating terminology to defined collections. X:Name is interpreted as the element Name as defined in namespace X. Y:Name is interpreted as the element Name as defined in namespace Y. Useful for disambiguating terms.

Figure 12-18
Important XML Standards

parse. The SAX parser processes the document and calls back objects within the XSLT processor whenever particular structures are encountered. A SAX parser, for example, calls the XSLT parser when it encounters a new element, passing the name of the element, its content, and other relevant items.

The DOM API works from a different paradigm. A DOM-compliant parser processes the entire XML document and then creates a tree representation of it. Each element of the document is a node on the tree. The XSLT processor can then call the DOM parser to obtain particular elements using XPath or a similar addressing scheme. DOM requires the entire document to be processed at one time and may require an unreasonable amount of storage for very large documents. If the document is large, SAX is the better choice. However, if all of the document contents need to be available for use at once, DOM is the only choice.

XQuery is an emerging standard for expressing generalized queries on XML documents. You can think of XQuery as SQL for XML documents. When it becomes available, this standard will be very important to the database/XML world. Visit *www.w3.org/XML/Query/* for more on XQuery.

Amazon Ex. 1008.0545

The last XML standard we will mention, XML Namespaces, is very important because it is used to combine different vocabularies into the same XML Schema. It can be used to define and support domains and to disambiguate terms. The need for the latter occurs when a document contains synonyms. For example, consider a document that has two different uses for the term *Instrument*. Suppose that one use of this term refers to musical instruments and has subelements {Manufacturer, Model, Material}, as in {Horner, Bflat Clarinet, Wood}, and a second use of this term refers to electronic instruments and has subelements {Manufacturer, Model, Voltage}, as in {RadioShack, Ohm-meter, 12-volt}. The author of the XML Schema for such a document can define two different namespaces that each contain one of these definition. Then, the complexType definition for each of these definitions of Instrument can be prefixed by the label of the namespace, as was done in our schema documents when we used the label xsd. There is more to XML Namespaces, and you will undoubtedly learn more as you work with XML.

The XML Standards Committee continues its important work, and more standards will be developed as the needs arise. At present, work is underway for developing security standards. Keep checking *www.w3.org* for more information.

Summary

The confluence of database processing and document processing is one of the most important developments in information systems technology today. Database processing and document processing need each other. Database processing needs document processing for the representation and materialization of database views. Document processing needs database processing for the permanent storage of data.

SGML is as important to document processing as the relational model is to database processing. XML is a series of standards that were developed jointly by the database processing and document processing communities. XML provides a standardized yet customizable way to describe the contents of documents. XML documents can automatically be generated from database data, and database data can be automatically extracted from XML documents.

Although XML can be used to materialize Web pages, this is one of its least important uses. More important is its use for describing, representing, and materializing database views. XML is on the leading edge of database processing; see *www.w3.org* and *www.xml.org* for the latest developments.

XML is a better markup language than HTML, primarily because XML provides a clear separation between document structure, content, and materialization. Also, XML tags are not ambiguous.

The content of XML documents can be described by Document Type Declarations (DTDs) and by XML Schemas. An XML document that conforms to its DTD is called type-valid. A document can be well formed and not be type-valid, either because it violates the structure of its DTD or because it has no DTD.

XML documents are transformed when an XSLT processor applies an XSL document to the XML document. A common transformation is to convert the XML document into HTML format. In the future, other transformations will be more important. For example, XSL documents can be written to transform the same Order document into different formats needed by different departments, say for sales, accounting, or production. XSLT processing is context oriented; given a particular context, an action is taken when a particular item is located. Today, most browsers have built-in XSLT processors.

XML Schema is a standard for describing the content of an XML document. XML Schema can be used to define custom vocabularies. Documents that conform to an XML Schema are called schema-valid. Unlike DTDs, XML Schema documents are themselves XML documents and can be validated against their schema, which is maintained by W3C.

Schemas consist of elements and attributes. There are two types of elements: simple and complex. Simple elements have one data value. ComplexType elements can have multiple elements nested within them. ComplexTypes may also have attributes. The elements contained in a ComplexType may be simple or other ComplexTypes. ComplexTypes may also define element sequences. A good rule of thumb is that elements represent data and attributes represent metadata, although this is not part of any XML standard.

XML Schemas (and documents) may have more structure than the columns of a table. Groups, such as Phone and Address, can be defined. An XML Schema that has all elements at the same level is a flat schema. Structured schemas are those that have defined subgroups, such as Phone and Address. To avoid definition duplication, elements can be defined globally. Duplication is undesirable because there is the risk that definitions will become inconsistent if a change is made to one definition and not the other.

Oracle, SQL Server, and MySQL can produce XML documents from database data. The Oracle facilities require the use of Java; see *www.oracle.com* for more information. SQL Server supports an add-on expression to the SQL SELECT

Amazon Ex. 1008.0546

statement, the FOR XML expression. FOR XML can be used to produce XML documents in which all data are expressed as attributes or as elements. FOR XML can also write an XML Schema description as well as the XML document. Using FOR XML EXPLICIT, the developer can place some columns into elements and others into attributes.

When interpreting multitable selects, the FOR XML processor uses the order of the tables to determine the hierarchical order of elements in the document. FOR XML can be used to produce XML documents with one multivalue path. Documents with more than one multivalue path must be patched together in the application by some means.

XML is important because it facilitates the sharing of XML documents (and hence database data) among organizations. After an XML Schema has been defined, organizations can ensure that they are receiving and sending only schema-valid documents. Additionally, XSL documents can be coded to transform any schema-valid XML document, from any source, into other standardized formats. These advantages become even more important as industry groups standardize their own XML Schemas. XML also facilitates business-to-business processing. This chapter concludes with a brief description of additional XML standards: XPath, SAX, DOM, XQuery, and XML Namespaces.

Key Terms

document type declaration (DTD)
Extensible Markup Language (XML)
not-type-valid
Simple Object Access Protocol
SOAP
SQL SELECT . . . FOR XML statement
Standard Generalized Markup Language (SGML)
structured schemas
stylesheet
type-valid
XML Schema

Review Questions

12.1 Why do database processing and document processing need each other?

12.2 How are HTML, SGML, and XML related?

12.3 Explain the phrase "standardized but customizable."

12.4 What is SOAP? What did it stand for originally? What does it stand for today?

12.5 What are the problems in interpreting a tag such as <h2> in HTML?

12.6 What is a DTD, and what purpose does it serve?

12.7 What is the difference between a well-formed XML document and a type-valid XML document?

12.8 Why is it too limiting to say that XML is just the next version of HTML?

12.9 How are XML, XSL, and XSLT related?

12.10 Explain the use of the pattern {item, action} in the processing of an XSL document.

12.11 What is the purpose of XML Schema?

12.12 How does XML Schema differ from DTD?

12.13 What is a schema-valid document?

12.14 Explain the chicken-and-egg problem concerning the validation of XML Schema documents.

12.15 Explain the difference between simple and complex elements.

12.16 Explain the difference between elements and attributes.

12.17 What is a good basic rule for using elements and attributes to represent database data?

Amazon Ex. 1008.0547

12.18 Give an example, other than one in this text, of a flat XML Schema.

12.19 Give an example, other than one in this text, of a structured XML Schema.

12.20 What is the purpose of global elements?

12.21 What requirement is necessary for processing XML documents with Oracle?

12.22 Explain the difference between FOR XML RAW and FOR XML AUTO, ELEMENTS.

12.23 When would you use FOR XML EXPLICIT?

12.24 What is the importance of the order of tables in a SQL statement that uses FOR XML?

12.25 Explain, in your own words, why SQL with FOR XML cannot be used to construct an XML document having two multivalue paths.

12.26 Why is the limitation in Review Question 12.25 important?

12.27 Explain, in your own words, why XML is important to database processing.

12.28 Why is XML Schema important for interorganizational document sharing?

12.29 What is XPath?

12.30 How does DOM differ from SAX?

12.31 What is XQuery? What is it used for?

12.32 What is XML Namespaces? What is its purpose?

Project Questions

12.33 Create a DTD and XML document like that shown in Figure 12-1 to represent the first row in the ARTIST table (see Figure 7-12).

12.34 Using Figure 12-4 as an example, create an XSL document to materialize the document in your answer to Question 12.1. Materialize your document using a browser.

12.35 Create an XML Schema document for a row of TRANS. Place TransactionID as an attribute. Group acquisition data into a complexType, and group sales data into a second complexType. Use Figure 12-7 as an example.

12.36 Create an XML Schema for artists and the customers who are interested in them. Use Figure 12-13 as an example.

12.37 Create an XML Schema for artist, work, transaction, and customer data. Use Figure 12-15 as an example and include your answer to Project Question 12.35 in the schema.

12.38 Create an XML Schema for all artist data. Use Figure 12-16 and your answer from Project Question 12.35.

Amazon Ex. 1008.0548



If you have not already done so, answer parts A through G for Marcia's Dry Cleaning at the end of Chapter 7 (page 293). Use SQL Server, Oracle, or MySQL.

A. Using Figure 12-1 as an example, create an XML document with a DTD for a row of the CUSTOMER table.

B. Using Figure 12-4 as an example, create an XSL document to materialize the document you created in part A. Show your document in a browser.

C. Create an XML Schema document for a join of CUSTOMER and ORDER data. Assume that the document has one customer and from zero to many orders for that customer. Use Figure 12-13 as an example.

D. Write an SQL statement with FOR XML that will produce the document you created in part C.

E. Create an XML Schema document that has all of the data for a given customer. How many multivalue paths does this schema have?

F. Explain how the XML Schema document you created in part E can be used to advantage by Marcia's Dry Cleaning.



If you have not already done so, answer parts A through H for Morgan Importing at the end of Chapter 7 (page 294). Use SQL Server, Oracle, or MySQL.

A. Using Figure 12-1 as an example, create an XML document with a DTD for a row of the PURCHASE table.

B. Using Figure 12-4 as an example, create an XSL document to materialize the document you created in part A. Show your document in a browser.

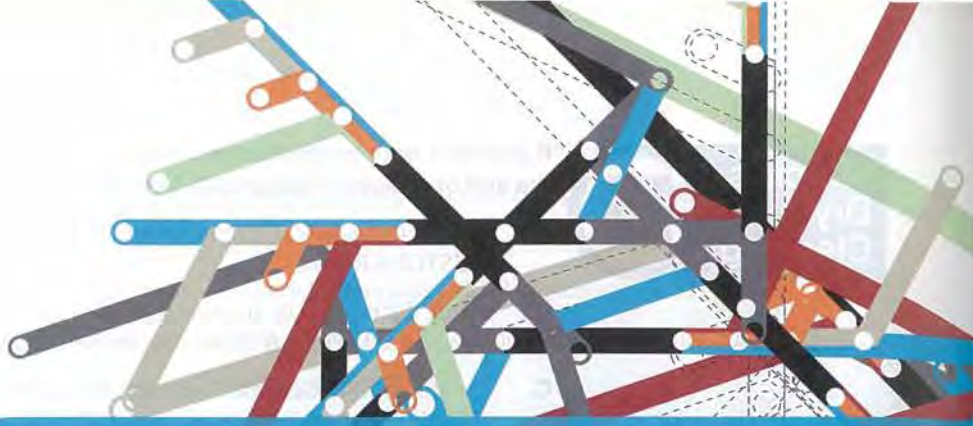
C. Create an XML Schema document for a join of STORE and PURCHASE data. Assume that the document has one store and from zero to many purchases for that store. Use Figure 12-13 as an example.

D. Write an SQL statement with FOR XML that will produce the document you created in part C.

E. Create an XML Schema document that has all of the data for a given purchase. How many multivalue paths does this schema have?

F. Explain how the XML Schema document you created in part E can be used to advantage by Morgan Importing.

Amazon Ex. 1008.0549



13 Database Processing for Business Intelligence Systems

Chapter Objectives

- To learn the basic concepts of data warehouses and data marts.
- To learn the basic concepts of dimensional databases.
- To learn the basic concepts of business intelligence (BI) systems.
- To learn the basic concepts of OnLine Analytical Processing (OLAP) and data mining.

Business Intelligence Systems

Business intelligence (BI) systems (also referred to as **decision support systems (DSSs)**) are information systems that assist managers and other professionals in the analysis of current and past activities and in the prediction of future events. Unlike transaction processing systems, they do not support operational activities, such as the recording and processing of orders. Instead, BI systems are used to support management assessment, analysis, planning, control, and, ultimately, decision making.

The Relationship Between Operational and BI Systems

Figure 13-1 summarizes the relationship among operational and business intelligence systems. **Operational systems**—such as sales, purchasing, and inventory control systems—support primary business activities. They use a DBMS to both read data from and store data in the operational database. They are also known as **transactional systems** or **online transaction processing (OLTP) systems**, because they record the ongoing stream of business transactions.

Amazon Ex. 1008.0550

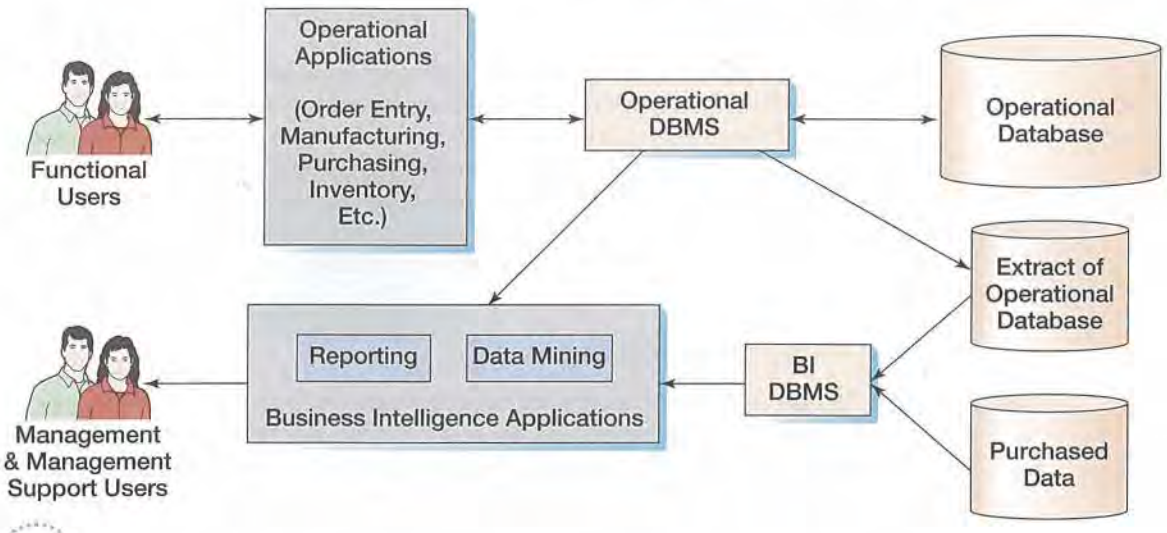


Figure 13-1

The Relationship Between Operational and BI Systems

Instead of supporting the primary business activities, BI systems support management's analysis and decision-making activities. BI systems obtain data from three possible sources. First, they read and process data existing in the operational database—they use the operational DBMS to obtain such data, but they do not insert, modify, or delete operational data. Second, BI systems process data that are extracted from operational databases. In this situation, they manage the extracted database using a BI DBMS, which may be the same as or different from the operational DBMS. Finally, BI systems read data purchased from data vendors.

Reporting Systems and Data Mining Applications

BI systems fall into two broad categories: reporting systems and data mining applications. **Reporting systems** sort, filter, group, and make elementary calculations on operational data. **Data mining applications**, in contrast, perform sophisticated analyses on data, analyses that usually involve complex statistical and mathematical processing. The characteristics of BI applications are summarized in Figure 13-2.

Figure 13-2

Characteristics of Business Intelligence Applications

- Reporting
  - Filter, sort, group, and make simple calculations
  - Summarize current status
  - Compare current status to past or predicted status
  - Classify entities (customers, products, employees, etc.)
  - Report delivery crucial
- Data Mining
  - Often employ sophisticated statistical and mathematical techniques
  - Used for:
    - What-if analyses
    - Predictions
    - Decisions
  - Results often incorporated into some other report or system

Amazon Ex. 1008.0551

Reporting Systems

Reporting systems filter, sort, group, and make simple calculations. All reporting analyses can be performed using standard SQL, although extensions to SQL, such as those used for **OnLine Analytical Processing (OLAP)**, are sometimes used to ease the task of report production.

Reporting systems summarize the current status of business activities and compare that status with past or predicted future activities. Report delivery is crucial. Reports must be delivered to the proper users on a timely basis, in the appropriate format. For example, reports may be delivered on paper, via a Web browser, or in some other format.

Data Mining Applications

Data mining applications use sophisticated statistical and mathematical techniques to perform what-if analyses, to make predictions, and to facilitate decision making. For example, data mining techniques can analyze past cell phone usage and predict which customers are likely to switch to a competing phone company. Or, data mining can be used to analyze past loan behavior to determine which customers are most (or least) likely to default on a loan.

Report delivery is not as important for data mining systems as it is for reporting systems. First, most data mining applications have only a few users, and those users have sophisticated computer skills. Second, the results of a data mining analysis are usually incorporated into some other report, analysis, or information system. In the case of cell phone usage, the characteristics of customers who are in danger of switching to another company may be given to the sales department for action. Or, the parameters of an equation for determining the likelihood of a loan default may be incorporated into a loan approval application.

Data Warehouses and Data Marts

According to Figure 13-1, some BI systems read and process operational data directly from the operational database. Although this is possible for simple reporting systems and small databases, such direct reading of operational data is not feasible for more complex applications or larger databases. Those larger applications usually process a separate database constructed from an extract of the operational database.

Operational data are difficult to read for several reasons. For one, querying data for BI applications can place a substantial burden on the DBMS and unacceptably slow the performance of operational applications. Additionally, operational data have problems that limit their use for BI applications. Further, the creation and maintenance of BI systems require programs, facilities, and expertise that are not normally available from operations.

Because of these problems, many organizations have chosen to develop data warehouses and data marts to support BI applications. Before we consider the components of data warehouses and data marts, let's first consider some of the problems of using operational data for BI processing.

Components of a Data Warehouse

To overcome the problems just described, many organizations have created **data warehouses**, which are database systems that have data, programs, and personnel that specialize in the preparation of data for BI processing. Data warehouse databases differ from operational databases because the data warehouse data are frequently denormalized. Data warehouses vary in scale and scope. They can be as simple as a sole employee processing a data extract on a part-time basis or as complex as a department with dozens of employees maintaining libraries of data and programs.

Figure 13-3 shows the components of a data warehouse. Data are read from operational databases by the **Extract, Transform, and Load (ETL) system**. The ETL system then cleans and prepares the data for BI processing. This can be a complex process.

Amazon Ex. 1008.0552

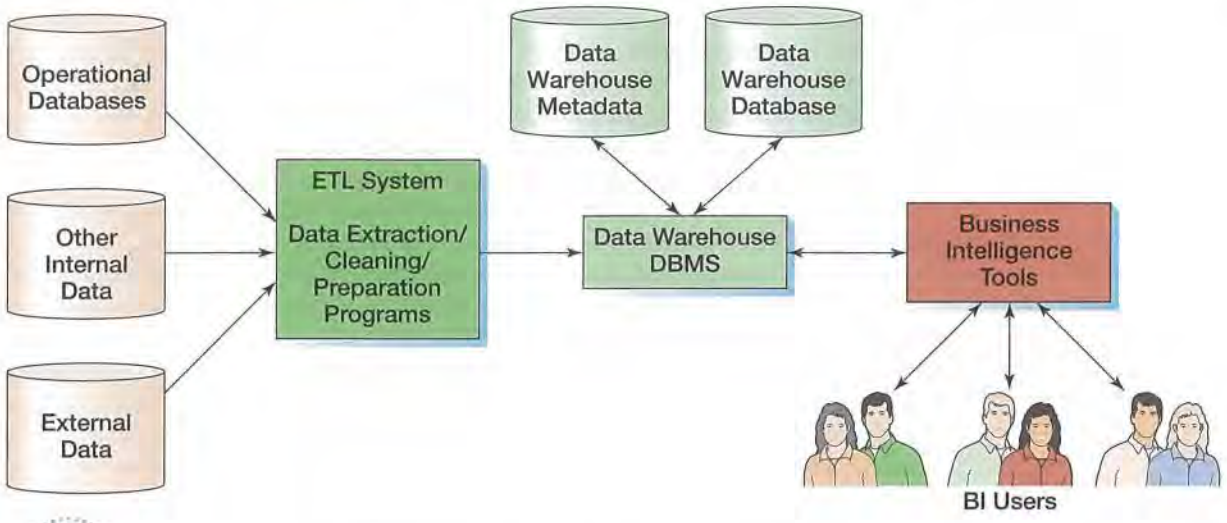


Figure 13-3
Components of a Data Warehouse

First, the data may be problematic, which we will discuss in the next section. Second, data may need to be changed or transformed for use in a data warehouse. For example, the operational systems may store data about countries using standard two-letter country codes, such as US (United States) and CA (Canada). However, applications using the data warehouse may need to use the country names in full. Thus, the data transformation **{CountryCode⟶CountryName}** will be needed before the data can be loaded into the data warehouse.

The ETL stores the extracted data in a data warehouse database using a data warehouse DBMS, which can be different from the organization's operational DBMS. For example, an organization might use Oracle for its operational processing, but use SQL Server for its data warehouse. Other organizations might use SQL Server for operational processing and data management programs from statistical package vendors such as SAS or SPSS in the data warehouse.

Metadata concerning the data's source, format, assumptions and constraints, and other facts are kept in a **data warehouse metadata database**. The data warehouse DBMS extracts and provides data to BI tools, such as data mining programs.

BY THE WAY Problematic operational data that have been cleaned in the ETL system can also be used to update the operational system to fix the original data problems.

Problems with Operational Data

Most operational databases have problems that limit their usefulness to all but the simplest BI applications. Figure 13-4 lists the major problem categories.

Figure 13-4
Problems of Using Transaction Data for Business Intelligence

- Dirty data
- Missing values
- Inconsistent data
- Data not integrated
- Wrong format
  - Too fine
  - Not fine enough
- Too much data
  - Too many attributes
  - Too much volume

Amazon Ex. 1008.0553

First, although data that are critical for successful operations must be complete and accurate, data that are only marginally necessary need not be. For example, some operational systems gather customer demographic data during the ordering process. But, because such data are not needed to fill, ship, or bill orders, the quality of the demographic data suffers.

Problematic data are termed **dirty data**. Examples are a value of "G" for customer sex and a value of "213" for customer age. Other examples are a value of "999-999-9999" for a U.S. phone number, a part color of "gren," and an e-mail address of "WhyMe@somewhereelseintheuniverse.who." All of these values pose problems for reporting and data mining purposes.

Purchased data often contain missing elements. In fact, most data vendors state the percentage of missing values for each attribute in the data they sell. An organization buys such data because, for some uses, some data are better than no data at all. This is especially true for data items whose values are difficult to obtain, such as the number of adults in a household, household income, dwelling type, and the education of primary income earner. Some missing data are not too much of a problem for reporting applications. For data mining applications, however, a few missing or erroneous data points can actually be worse than no data at all, because they bias the analysis.

Inconsistent data, the third problem in Figure 13-4, is particularly common for data that have been gathered over time. When an area code changes, for example, the phone number for a given customer before the change will differ from the customer's phone number after the change. Part codes can change, as can sales territories. Before such data can be used, it must be recoded for consistency over the period of the study.

Some data inconsistencies occur because of the nature of the business activity. Consider a Web-based order entry system used by customers around the world. When the Web server records the time of order, which time zone does it use? The server's system clock time is irrelevant to an analysis of customer behavior. Any standard time such as Universal Time Coordinate (UTC) time is also meaningless. Somehow, Web server time must be adjusted to the time zone of the customer.

Another problem is **nonintegrated data**. Suppose, for example, that an organization wants to report on customer order and payment behavior. Unfortunately, order data are stored in a CRM system developed by Siebel, whereas payment data are recorded in a PeopleSoft financial management database. To perform the analysis, the data must somehow be integrated.

Data can also be too fine or too coarse. With regards to the former, suppose that we want to analyze the placement of graphics and controls on an order entry Web page. It is possible to capture the customers' clicking behavior in what is termed **click-stream data**. However, click-stream data include *everything* the customer does. In the middle of the order stream there may be data for clicks on the news, e-mail, instant chat, and the weather. Although all of this data might be useful for a study of consumer computer behavior, it will be overwhelming if all we want to know is how customers respond to an ad located on the screen. Because the data are too fine, the data analysts must throw millions and millions of clicks away before they can proceed.

Data can also be too coarse. A file of order totals cannot be used for a market basket analysis, which identifies items that are commonly purchased together. Market basket analyses require item-level data; we need to know which items were purchased with which others. This doesn't mean the order total data are useless; it can be adequate for other analyses, it just won't do for a market basket analysis.

If the data are too fine, they can be made coarser by summing and combining. An analyst and a computer can sum and combine such data. If the data are too coarse, however, they cannot be separated into their constituent parts.

The final problem listed in Figure 13-4 concerns data volume. We can have an excess of columns, rows, or both. Suppose that we want to know the attributes that influence customers' responses to a promotion. Between customer data stored within the organization and customer data that can be purchased, we might have a hundred or more different attributes, or columns, to consider. How do we select among them? Because of a phenomenon called the **curse of dimensionality**, the more attributes there are, the easier it is to build a model that fits the sample data but that is worthless as a predictor. For this and other reasons, the number of attributes should be reduced and one of the major activities in data mining concerns the efficient and effective selection of variables.

Amazon Ex. 1008.0554

Finally, we may have too many instances, or rows, of data. Suppose that we want to analyze click-stream data on CNN.com. How many clicks does this site receive per month? Millions upon millions! To meaningfully analyze such data, we need to reduce the number of instances. A good solution to this problem is statistical sampling. However, developing a reliable sample requires specialized expertise and information system tools.

Purchasing Data for Vendors

Data warehouses often include data that are purchased from outside sources. A typical example is customer credit data. Figure 13-5 lists some of the consumer data than can be purchased from the KnowledgeBase Marketing company in their AmeriLINK database of consumer data (*www2.kbm1.com/Documents/AmeriLINK_brochure.pdf*). An amazing, and from a privacy standpoint frightening, amount of data is available just from this one vendor.

Data Warehouses Versus Data Marts

You can think of a data warehouse as a distributor in a supply chain. The data warehouse takes data from the data manufacturers (operational systems and purchased data), cleans and processes them, and locates the data on the shelves, so to speak, of the data warehouse. The people who work in a data warehouse are experts at data management, data cleaning, data transformation, and the like. However, they are not usually experts in a given business function.

A **data mart** is a collection of data that is smaller than that in the data warehouse and that addresses a particular component or functional area of the business. A data mart is like a retail store in a supply chain. Users in the data mart obtain data that pertain to a particular business function from the data warehouse. Such users do not have the data management expertise that data warehouse employees have, but they are knowledgeable analysts for a given business function. Figure 13-6 illustrates these relationships.

The data warehouse takes data from the data producers and distributes the data to three data marts. One data mart analyzes click-stream data for the purpose of designing Web pages.

The second analyzes store sales data and determines which products tend to be purchased together. This information is used to train salespeople on the best way to up-sell customers. The third data mart analyzes customer order data for the purpose of reducing labor for item picking from the warehouse. A company such as Amazon.com, for example, goes to great lengths to organize its warehouses to reduce picking expenses.

When the data mart structure shown in Figure 13-6 is combined with the data warehouse architecture shown in Figure 13-3, the combined system is known as an **enterprise data warehouse (EDW) architecture**. In this configuration, the data warehouse maintains all

Figure 13-5

The AmeriLINK Database Contains Data on 230 Million Americans



Amazon Ex. 1008.0555

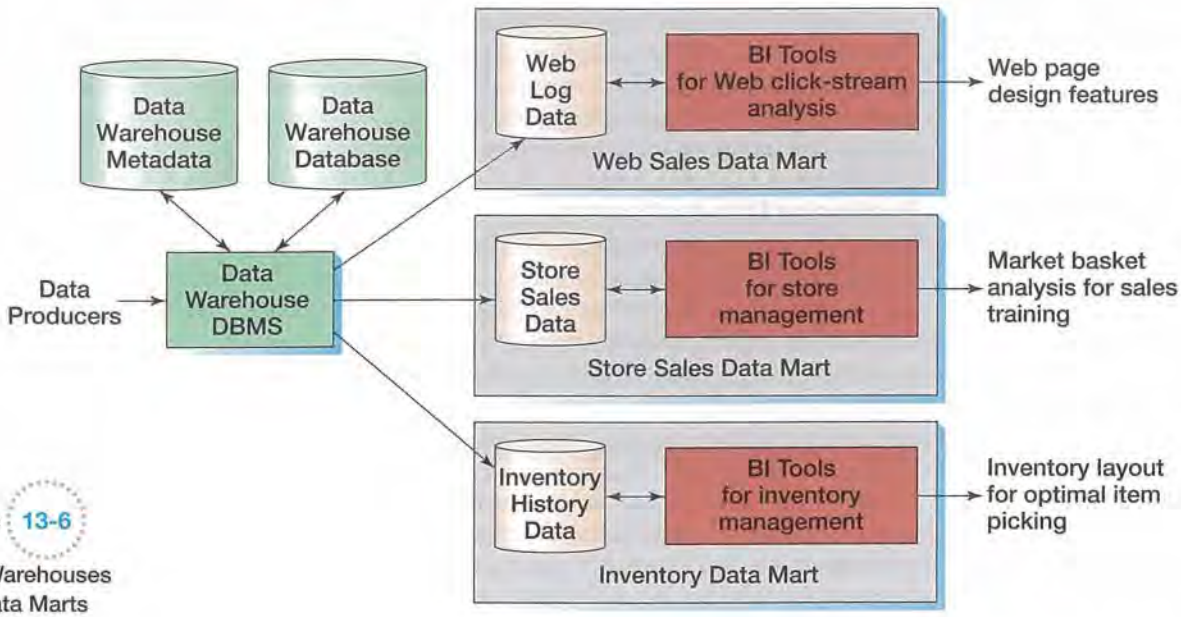


Figure 13-6
Data Warehouses and Data Marts

enterprise BI data and acts as the authoritative source for data extracts provided to the data marts. The data marts receive all their data from the data warehouse—they do not add or maintain any additional data.

Of course, it is expensive to create, staff, and operate data warehouses and data marts, and only large organizations with deep pockets can afford to operate a system such as an EDW. Smaller organizations operate subsets of this system. For example, they may have just a single data mart for analyzing marketing and promotion data.

Dimensional Databases

The databases in a data warehouse or data mart are built to a different type of database design than the normalized relational databases used for operational systems. The data warehouse databases are built in a design called a **dimensional database** that is designed for efficient data queries and analysis. A dimensional database is used to store historical data rather than just the current data stored in an operational database. Figure 13-7 compares operational databases and dimensional databases.

Because dimensional databases are used for the analysis of historical data, they must be designed to handle data that change over time. For example, a customer may have moved from one residence to another in the same city or may have moved to a completely different city and state. This type of data arrangement is called a **slowly changing dimension**, and in order to track such changes a dimensional database must have a **date dimension** or **time dimension** as well.

Figure 13-7
Characteristics of Operational and Dimensional Databases

Operational Database	Dimensional Database
Used for structured transaction data processing	Used for unstructured analytical data processing
Current data are used	Current and historical data are used
Data are inserted, updated, and deleted by users	Data are loaded and updated systematically, not by users

Amazon Ex. 1008.0556

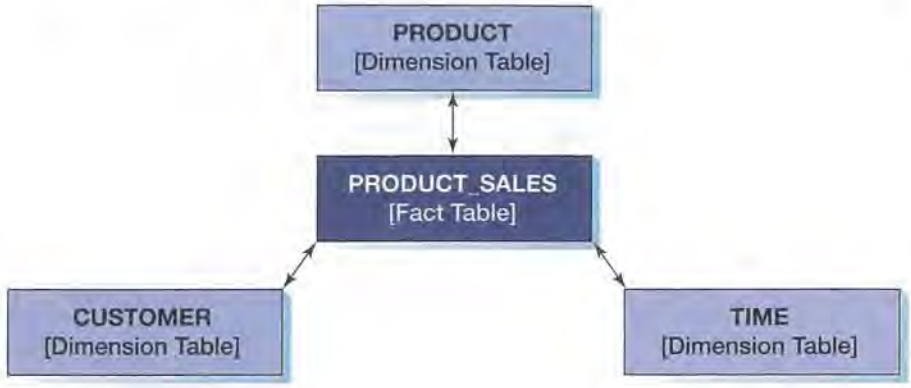


Figure 13-8
The Star Schema

The Star Schema

Rather than using the normalized database designs used in operational databases, a dimensional database uses a star schema. A **star schema**, so named because, as shown in Figure 13-8, it visually resembles a star, with a **fact table** at the center of the star and **dimension tables** radiating out from the center. The fact table is always fully normalized, but dimension tables may be non-normalized.

BY THE WAY There is a more complex version of the star schema called the *snowflake schema*. In the snowflake schema, each dimension table is normalized, which may create additional tables attached to the dimension tables.

To illustrate a star schema for a dimensional database, we will build a small (very small) data warehouse for Heather Sweeney Designs (HSD), a Texas company specializing in products for kitchen-remodeling services. HSD puts on seminars to attract customers and sell books and videos in addition to doing actual design work. A database design for HSD is shown in Figure 13-9, and an SQL Server database diagram for the HSD database is shown in Figure 13-10. The actual dimensional database for BI use at HSD is named *HSD-DW*, and it is shown in Figure 13-11. The SQL statements needed to create the tables in the HSD-DW database are shown in Figure 13-12, and the data for the HSD-DW database are shown in Figure 13-13. Compare the HSD-DW dimensional database model in Figure 13-11 to the HSD database diagram shown in Figure 13-10 and note how data in the HSD database have been used in the HSD-DW schema.

BY THE WAY Note that in the HSD-DW database the CUSTOMER table has a surrogate primary key named CustomerID, which has an integer value, whereas in the HSD database the primary key EmailAddress was used. There are two reasons for this. First, the primary key EmailAddress used in the HSD database is simply too cumbersome for a data warehouse, so we switched to the preferable small and numeric surrogate key. Second, we do not use individual EmailAddress values in the HSD-DW database, only values of EmailDomain, which is not unique and cannot be used as a primary key.

A fact table is used to store **measures** of business activity, which are quantitative or factual data about the entity represented by the fact table. For example, in the HSD-DW database, the fact table is PRODUCT_SALES:

PRODUCT_SALES (TimeID, CustomerID, ProductNumber, Quantity, UnitPrice, Total)

Amazon Ex. 1008.0557

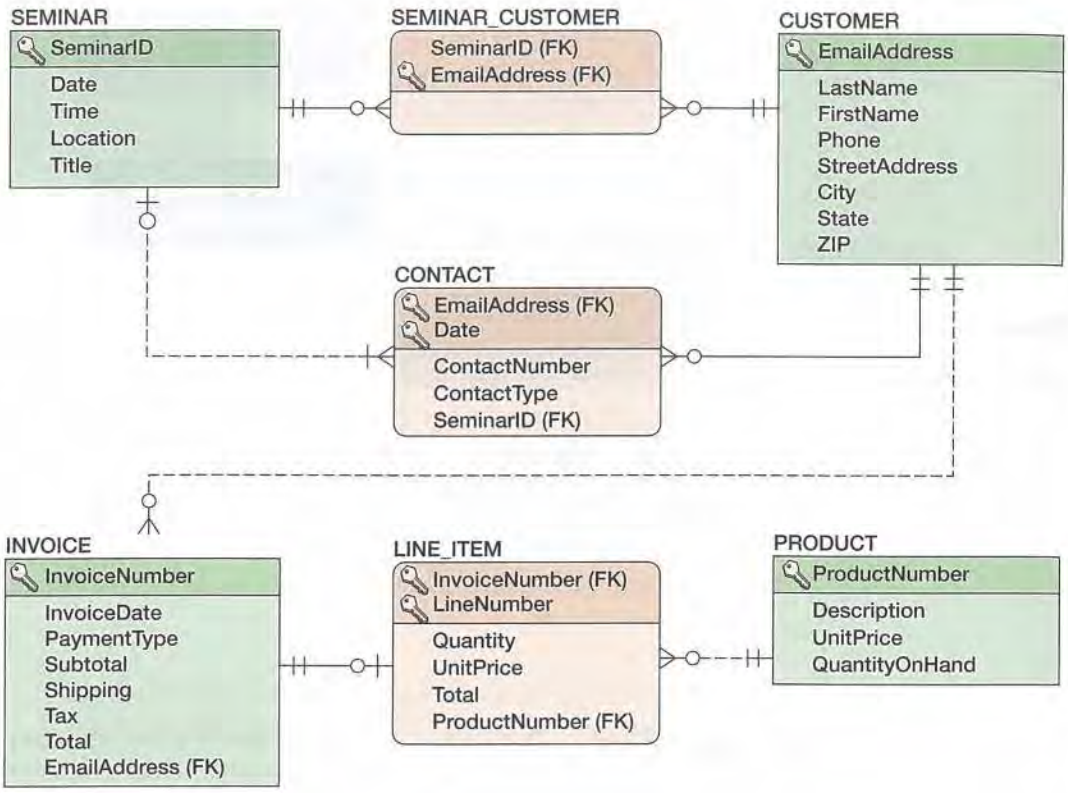


Figure 13-9

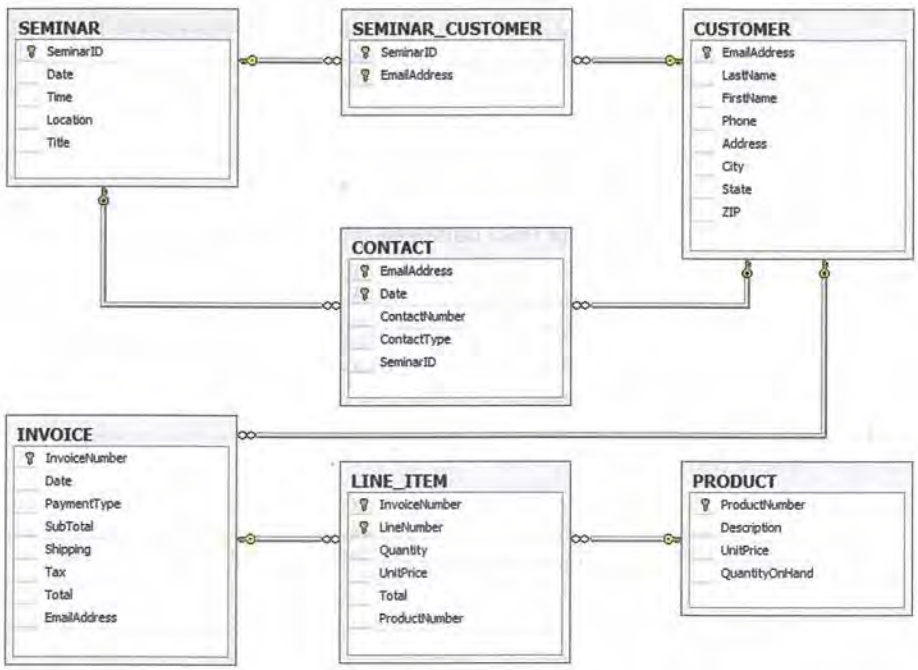
The HSD Database Design

In this table:

- Quantity is quantitative data that record how many of the item were sold.
- UnitPrice is quantitative data that record the dollar price of each item sold.
- Total (= Quantity * UnitPrice) is quantitative data that record the total dollar value of the sale of this item.

Figure 13-10

The HSD Database Diagram



Amazon Ex. 1008.0558

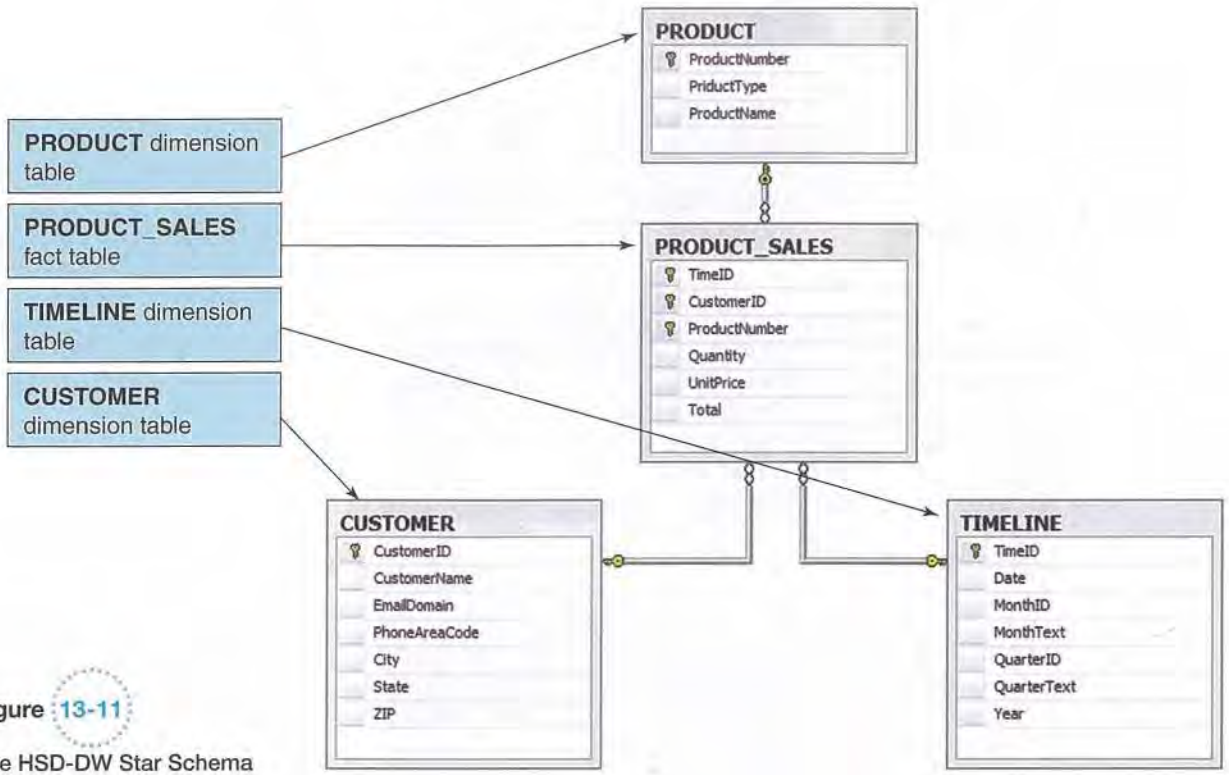


Figure 13-11
The HSD-DW Star Schema

The measures in the PRODUCT_SALES table are for *units of product per day*. We do not use individual sale data (which would be based on InvoiceNumber), but rather data summed for each customer for each day. For example, if you could compare the HSD database INVOICE data for Ralph Able for 6/5/08, you would see that Ralph made two purchases on that date (InvoiceNumber 35013 and InvoiceNumber 35016). In the HSD-DW database, however, these two purchases are summed into the PRODUCT_SALES data for Ralph (CustomerID = 3) for 6/5/08 (TimeID = 39604).

BY THE WAY The TimeID values are the sequential *serial values* used in Microsoft Excel to represent dates. Starting with 01-JAN-1900 as date value 1, the date value is increased by 1 for each calendar day. Thus, 05-JUN-2008 = 39604. For more information, search "Date formats" in the Excel help system.

A dimension table is used to record values of attributes that describe the fact measures in the fact table, and these attributes are used in queries to select and group the measures in the fact table. Thus, CUSTOMER records data about the customers referenced by CustomerID in the SALES table, TIMELINE provides data that can be used to interpret the SALES event in time (which month? which quarter?), and so on. A query to summarize product units sold by Customer (CustomerName) and Product (ProductName) would be:

```
SELECT    C.CustomerNumber, C.CustomerName,
          C.ProductNumber, P.ProductName,
          SUM(PS.Quantity) AS TotalQuantity
FROM      CUSTOMER C, PRODUCT_SALES PS, PRODUCT P
WHERE     C.CustomerID = PS.CustomerID
  AND     P.ProductNumber = PS.ProductNumber
GROUP BY  C.CustomerNumber, C.CustomerName,
          P.ProductNumber, P.ProductName
ORDER BY  C.CustomerNumber, P.ProductNumber;
```

Amazon Ex. 1008.0559

```
CREATE TABLE TIMELINE(
      TimeID            Int              NOT NULL,
      [Date]            DateTime         NOT NULL,
      MonthID           Int              NOT NULL,
      Month             Char(15)         NOT NULL,
      QuarterID         Int              NOT NULL,
      Quarter           Int              NOT NULL,
      [Year]            Char(10)         NOT NULL,
      CONSTRAINT        TIMELINE_PK      PRIMARY KEY(TimeID)
      );

CREATE TABLE CUSTOMER(
      CustomerID        Int              NOT NULL,
      CustomerName      Char(75)         NOT NULL,
      EmailDomain       VarChar(100)     NOT NULL,
      PhoneAreaCode     Char(6)          NOT NULL,
      City              Char(35)         NULL,
      [State]           Char(2)          NULL,
      ZIP               Char(10)         NULL,
      CONSTRAINT        CUSTOMER_PK      PRIMARY KEY(CustomerID)
      );

CREATE TABLE PRODUCT(
      ProductNumber     Char(35)         NOT NULL,
      ProductType       Char(25)         NOT NULL,
      ProductName       VarChar(75)      NOT NULL,
      CONSTRAINT        PRODUCT_PK       PRIMARY KEY(ProductNumber)
      );

CREATE TABLE PRODUCT_SALES(
      TimeID            Int              NOT NULL,
      CustomerID        Int              NOT NULL,
      ProductNumber     Char(35)         NOT NULL,
      Quantity          Int              NOT NULL,
      UnitPrice         Numeric(9,2)     NOT NULL,
      Total             Numeric(9,2)     NULL,
      CONSTRAINT        PRODUCT_SALES_PK
                            PRIMARY KEY (TimeID,CustomerID,ProductNumber),
      CONSTRAINT        PS_TIMELINE_FK FOREIGN KEY(TimeID)
                            REFERENCES TIMELINE(TimeID)
                               ON UPDATE NO ACTION
                               ON DELETE NO ACTION,
      CONSTRAINT        PS_CUSTOMER_FK FOREIGN KEY(CustomerID)
                            REFERENCES CUSTOMER(CustomerID)
                               ON UPDATE NO ACTION
                               ON DELETE NO ACTION,
      CONSTRAINT        PS_PRODUCT_FK FOREIGN KEY(ProductNumber)
                            REFERENCES PRODUCT(ProductNumber)
                               ON UPDATE NO ACTION
                               ON DELETE NO ACTION
      );
```

Figure 13-12
The HSD-DW SQL Statements

The results of this query are shown in Figure 13-14.

In Chapter 6, we discussed how an N:M relationship is created in a database as two 1:N relationships by use of an intersection table. We also discussed how additional attributes can be added to the intersection table in an association relationship.

Amazon Ex. 1008.0560

(a) TIMELINE Dimension Table

	TimeID	Date	MonthID	MonthText	QuarterID	QuarterText	Year
1	39370	2007-10-15 00:00:00.000	10	October	3	Qtr3	2007
2	39380	2007-10-25 00:00:00.000	10	October	3	Qtr3	2007
3	39436	2007-12-20 00:00:00.000	12	December	3	Qtr3	2007
4	39532	2008-03-25 00:00:00.000	3	March	1	Qtr1	2008
5	39534	2008-03-27 00:00:00.000	3	March	1	Qtr1	2008
6	39538	2008-03-31 00:00:00.000	3	March	1	Qtr1	2008
7	39541	2008-04-03 00:00:00.000	4	April	2	Qtr2	2008
8	39546	2008-04-08 00:00:00.000	4	April	2	Qtr2	2008
9	39561	2008-04-23 00:00:00.000	4	April	2	Qtr2	2008
10	39575	2008-05-07 00:00:00.000	5	May	2	Qtr2	2008
11	39589	2008-05-21 00:00:00.000	5	May	2	Qtr2	2008
12	39604	2008-06-05 00:00:00.000	6	June	2	Qtr2	2008

(b) CUSTOMER Dimension Table

	CustomerID	CustomerName	EmailDomain	PhoneAreaCode	City	State	ZIP
1	1	Jacobs, Nancy	somewhere.com	817	Fort Worth	TX	76110
2	2	Jacobs, Chantel	somewhere.com	817	Fort Worth	TX	76112
3	3	Able, Ralph	somewhere.com	210	San Antonio	TX	78214
4	4	Baker, Susan	elsewhere.com	210	San Antonio	TX	78216
5	5	Eagleton, Sam	elsewhere.com	210	San Antonio	TX	78218
6	6	Fortist, Kathy	somewhere.com	972	Dallas	TX	75220
7	7	George, Sally	somewhere.com	972	Dallas	TX	75223
8	8	Hulett, Shawn	elsewhere.com	972	Dallas	TX	75224
9	9	Pearson, Bobbi	elsewhere.com	512	Austin	TX	78710
10	10	Ranger, Terry	somewhere.com	512	Austin	TX	78712
11	11	Tyler, Jenny	somewhere.com	972	Dallas	TX	75225
12	12	Wayne, Joan	elsewhere.com	817	Fort Worth	TX	76115

(c) PRODUCT Dimension Table

	ProductNumber	ProductType	ProductName
1	BK001	Book	Kitchen Remodeling Basics For Everyone
2	BK002	Book	Advanced Kitchen Remodeling For Everyone
3	VB001	Video Companion	Kitchen Remodeling Basics Video Companion
4	VB002	Video Companion	Advanced Kitchen Remodeling Video Companion
5	VB003	Video Companion	Kitchen Remodeling Dallas Style Video Companion
6	VK001	DVD Video	Kitchen Remodeling Basics
7	VK002	DVD Video	Advanced Kitchen Remodeling
8	VK003	DVD Video	Kitchen Remodeling Dallas Style
9	VK004	DVD Video	Heather Sweeny Seminar Live in Dallas on 25-OCT-07

(d) PRODUCT_SALES Fact Table

	TimeID	CustomerID	ProductNumber	Quantity	UnitPrice	Total
1	39370	3	VB001	1	7.99	7.99
2	39370	3	VK001	1	14.95	14.95
3	39380	4	BK001	1	24.95	24.95
4	39380	4	VB001	1	7.99	7.99
5	39380	4	VK001	1	14.95	14.95
6	39436	7	VK004	1	24.95	24.95
7	39532	4	BK002	1	24.95	24.95
8	39532	4	VK002	1	14.95	14.95
9	39532	4	VK004	1	24.95	24.95
10	39534	6	BK002	1	24.95	24.95
11	39534	6	VB003	1	9.99	9.99
12	39534	6	VK002	1	14.95	14.95
13	39534	6	VK003	1	19.95	19.95
14	39534	6	VK004	1	24.95	24.95
15	39534	7	BK001	1	24.95	24.95
16	39534	7	BK002	1	24.95	24.95
17	39534	7	VK003	1	19.95	19.95
18	39534	7	VK004	1	24.95	24.95
19	39538	9	BK001	1	24.95	24.95
20	39538	9	VB001	1	7.99	7.99
21	39538	9	VK001	1	14.95	14.95
22	39541	11	VB003	2	9.99	19.98
23	39541	11	VK003	2	19.95	39.90
24	39541	11	VK004	2	24.95	49.90
25	39546	1	BK001	1	24.95	24.95
26	39546	1	VB001	1	7.99	7.99
27	39546	1	VK001	1	14.95	14.95
28	39546	5	BK001	1	24.95	24.95
29	39546	5	VB001	1	7.99	7.99
30	39546	5	VK001	1	14.95	14.95
31	39561	3	BK001	1	24.95	24.95
32	39575	9	VB002	1	7.99	7.99
33	39575	9	VK002	1	14.95	14.95
34	39589	8	VB003	1	9.99	9.99
35	39589	8	VK003	1	19.95	19.95
36	39589	8	VK004	1	24.95	24.95
37	39604	3	BK002	1	24.95	24.95
38	39604	3	VB001	1	7.99	7.99
39	39604	3	VB002	2	7.99	15.98
40	39604	3	VK001	1	14.95	14.95
41	39604	3	VK002	2	14.95	29.90
42	39604	11	VB002	2	7.99	15.98
43	39604	11	VK002	2	14.95	29.90
44	39604	12	BK002	1	24.95	24.95
45	39604	12	VB003	1	9.99	9.99
46	39604	12	VK002	1	14.95	14.95
47	39604	12	VK003	1	19.95	19.95
48	39604	12	VK004	1	24.95	24.95

Figure 13-13
The HSD-DW Table Data

In a star schema, the fact table is an association table—it is an intersection table for the relationships between the dimension tables with additional measures also stored in it. And, as with all other intersection tables, the key of the fact table is a composite key made up of all the foreign keys to the dimension tables.

Illustrating the Dimensional Model

When you think of the word *dimension*, you might think of "two dimensional" or "three dimensional." And the dimensional models can be illustrated by using a two-dimensional matrix and a three-dimensional cube. Figure 13-15 shows the SQL query results from Figure 13-14, displayed as a two-dimensional matrix of Product (using ProductNumber) and Customer (using CustomerID), with each cell showing the number of units of each product purchased by each customer. Note how ProductNumber and CustomerID define the two dimensions of the matrix: CustomerID labels what would be the *x*-axis, and ProductNumber labels the *y*-axis.

Figure 13-16 shows a three-dimensional cube with the same ProductNumber and CustomerID dimensions, but now with the added Time dimension on the *z*-axis. Now instead of occupying a two-dimensional box, the total quantity of products purchased by each customer on each day occupies a small three-dimensional cube, and all these small cubes combine to form a large cube.

As human beings, we can visualize two-dimensional matrices and three-dimensional cubes. Although we cannot visualize models with four, five, and more dimensions, BI systems and dimensional databases can handle such models.

Multiple Fact Tables and Conformed Dimensions

Data warehouse systems build dimensional models, as needed, to analyze BI questions, and the HSD-DW star schema in Figure 13-11 would be just one schema in a set of schemas. Figure 13-17 shows an extended HSD-DW schema.

Amazon Ex. 1008.0561

	CustomerID	CustomerName	ProductNumber	ProductName	TotalQuantity
1	1	Jacobs, Nancy	BK001	Kitchen Remodeling Basics For Everyone	1
2	1	Jacobs, Nancy	VB001	Kitchen Remodeling Basics Video Companion	1
3	1	Jacobs, Nancy	VK001	Kitchen Remodeling Basics	1
4	3	Able, Ralph	BK001	Kitchen Remodeling Basics For Everyone	1
5	3	Able, Ralph	BK002	Advanced Kitchen Remodeling For Everyone	1
6	3	Able, Ralph	VB001	Kitchen Remodeling Basics Video Companion	2
7	3	Able, Ralph	VB002	Advanced Kitchen Remodeling Video Companion	2
8	3	Able, Ralph	VK001	Kitchen Remodeling Basics	2
9	3	Able, Ralph	VK002	Advanced Kitchen Remodeling	2
10	4	Baker, Susan	BK001	Kitchen Remodeling Basics For Everyone	1
11	4	Baker, Susan	BK002	Advanced Kitchen Remodeling For Everyone	1
12	4	Baker, Susan	VB001	Kitchen Remodeling Basics Video Companion	1
13	4	Baker, Susan	VK001	Kitchen Remodeling Basics	1
14	4	Baker, Susan	VK002	Advanced Kitchen Remodeling	1
15	4	Baker, Susan	VK004	Heather Sweeny Seminar Live in Dallas on 25-OCT-07	1
16	5	Eagleton, Sam	BK001	Kitchen Remodeling Basics For Everyone	1
17	5	Eagleton, Sam	VB001	Kitchen Remodeling Basics Video Companion	1
18	5	Eagleton, Sam	VK001	Kitchen Remodeling Basics	1
19	6	Foxtrot, Kathy	BK002	Advanced Kitchen Remodeling For Everyone	1
20	6	Foxtrot, Kathy	VB003	Kitchen Remodeling Dallas Style Video Companion	1
21	6	Foxtrot, Kathy	VK002	Advanced Kitchen Remodeling	1
22	6	Foxtrot, Kathy	VK003	Kitchen Remodeling Dallas Style	1
23	6	Foxtrot, Kathy	VK004	Heather Sweeny Seminar Live in Dallas on 25-OCT-07	1
24	7	George, Sally	BK001	Kitchen Remodeling Basics For Everyone	1
25	7	George, Sally	BK002	Advanced Kitchen Remodeling For Everyone	1
26	7	George, Sally	VK003	Kitchen Remodeling Dallas Style	1
27	7	George, Sally	VK004	Heather Sweeny Seminar Live in Dallas on 25-OCT-07	2
28	8	Hullett, Shawn	VB003	Kitchen Remodeling Dallas Style Video Companion	1
29	8	Hullett, Shawn	VK003	Kitchen Remodeling Dallas Style	1
30	8	Hullett, Shawn	VK004	Heather Sweeny Seminar Live in Dallas on 25-OCT-07	1
31	9	Pearson, Bobbi	BK001	Kitchen Remodeling Basics For Everyone	1
32	9	Pearson, Bobbi	VB001	Kitchen Remodeling Basics Video Companion	1
33	9	Pearson, Bobbi	VB002	Advanced Kitchen Remodeling Video Companion	1
34	9	Pearson, Bobbi	VK001	Kitchen Remodeling Basics	1
35	9	Pearson, Bobbi	VK002	Advanced Kitchen Remodeling	1
36	11	Tyler, Jenny	VB002	Advanced Kitchen Remodeling Video Companion	2
37	11	Tyler, Jenny	VB003	Kitchen Remodeling Dallas Style Video Companion	2
38	11	Tyler, Jenny	VK002	Advanced Kitchen Remodeling	2
39	11	Tyler, Jenny	VK003	Kitchen Remodeling Dallas Style	2
40	11	Tyler, Jenny	VK004	Heather Sweeny Seminar Live in Dallas on 25-OCT-07	2
41	12	Wayne, Joan	BK002	Advanced Kitchen Remodeling For Everyone	1
42	12	Wayne, Joan	VB003	Kitchen Remodeling Dallas Style Video Companion	1
43	12	Wayne, Joan	VK002	Advanced Kitchen Remodeling	1
44	12	Wayne, Joan	VK003	Kitchen Remodeling Dallas Style	1
45	12	Wayne, Joan	VK004	Heather Sweeny Seminar Live in Dallas on 25-OCT-07	1

Figure 13-14

The HSD-DW Query Results

In Figure 13-17, a second fact table named SALES_FOR_RFM has been added:

SALES_FOR_RFM (TimeID, CustomerID, InvoiceNumber, PreTaxTotalSale)

This table shows that fact table primary keys do not need to be composed solely of foreign keys that link to dimension tables. In SALES_FOR_RFM, the primary key includes the InvoiceNumber attribute. This attribute is necessary because the composite key (TimeID, CustomerID) will not be unique and thus cannot be the primary key. Note that SALES_FOR_RFM links to the same CUSTOMER and TIMELINE dimension tables as PRODUCT_SALES. This is done to maintain consistency within the data warehouse, and when a dimension table links to two or more fact tables it is called a **conformed dimension**.

Why would we add a fact table named SALES_FOR_RFM? To explain that, we need to discuss reporting systems.

Figure 13-15

The Two-Dimensional ProductNumber–CustomerID Matrix

Each cell shows the total quantity of each product that has been purchased by each customer

	CustomerID											
ProductNumber	1	2	3	4	5	6	7	8	9	10	11	12
BK001	1		1	1			1		1			
BK002			1	1		1	1					1
VB001	1		2	1	1				1			
VB002			2						1		2	
VB003						1		1			2	1
VK001	1		2	1	1				1			
VK002			2	1		1			1		2	1
VK003						1	1	1			2	1
VK004				1		1	2	1			2	1

Amazon Ex. 1008.0562

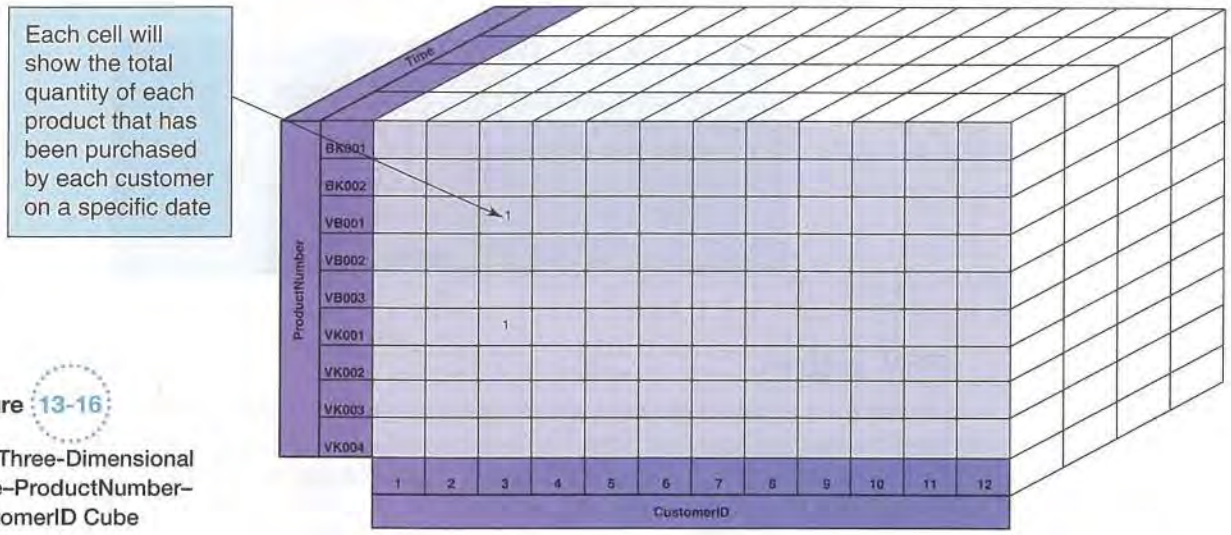


Figure 13-16

The Three-Dimensional Time–ProductNumber–CustomerID Cube

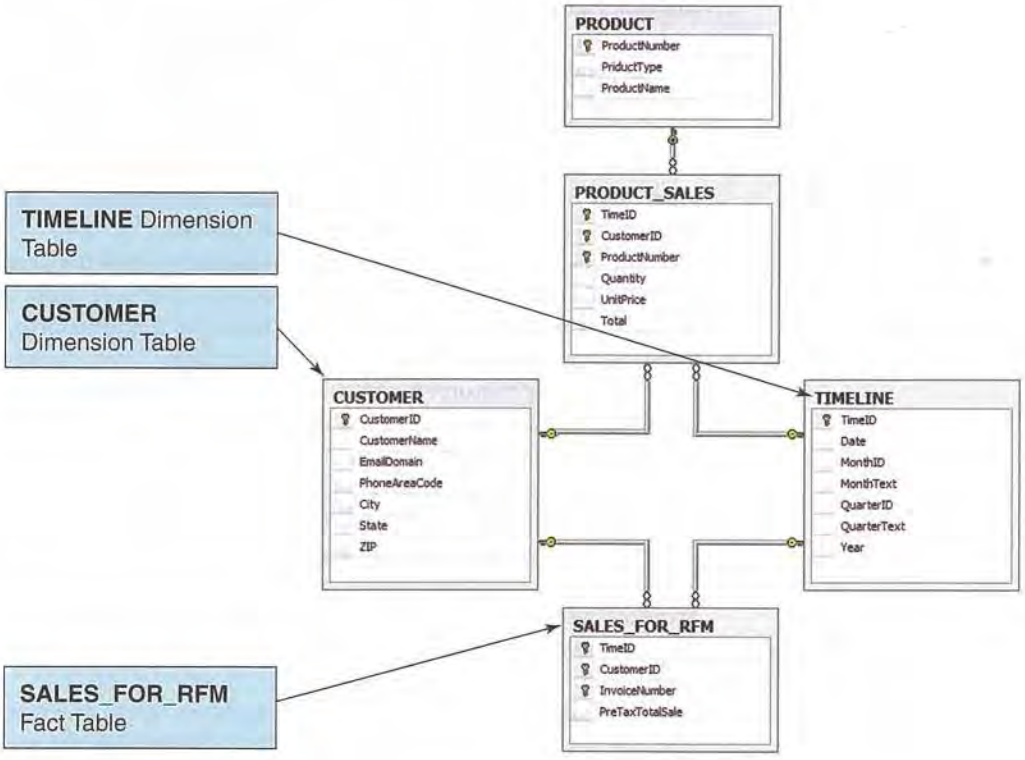


Figure 13-17

The Extended HSD-DW Schema

Reporting Systems

The purpose of a reporting system is to create meaningful information from disparate data sources and to deliver that information to the proper users on a timely basis. As stated earlier, reporting systems differ from data mining because they create information using the simple operations of sorting, filtering, grouping, and making simple calculations. We begin this section with a description of a typical reporting problem: RFM analysis.

Amazon Ex. 1008.0563

Figure 13-18
RFM Analysis

- Simple report-based customer classification scheme
- Score customers on recentness, frequency, and monetary size of orders
- Typically, divide each criterion into 5 groups and score from 1 to 5

RFM Analysis

As discussed earlier, **RFM analysis** is a way of analyzing and ranking customers according to their purchasing patterns. It is a simple technique that considers how *recently* (**R**) a customer ordered, how *frequently* (**F**) a customer orders, and how much *money* (**M**) the customer spends per order. RFM is summarized in Figure 13-18.

To produce an RFM score, we need only two things: customer data and sales data for each purchase (the date of the sale and the total amount of the sale) made by each customer. If you look at the SALES_FOR_RFM table and its associated CUSTOMER and TIMELINE dimension tables in Figure 13-17, you see that we have exactly those data: The SALE_FOR_RFM table is the starting point for RFM analysis in the HSD-DW BI system.

To produce an RFM score, customer purchase records are first sorted by the date of their most recent (R) purchase. In a common form of this analysis, the customers are divided into five groups, and a score of 1 to 5 is given to customers in each group. Thus, the 20 percent of the customers having the most recent orders are given an R score of 1, the 20 percent of the customers having the next most recent orders are given an R score of 2, and so forth, down to the last 20 percent, who are given an R score of 5.

The customers are then resorted on the basis of how frequently they order. The 20 percent of the customers who order most frequently are given an F score of 1, the next 20 percent most frequently ordering customers are given a score of 2, and so forth, down to the least frequently ordering customers, who are given an F score of 5.

Finally, the customers are sorted again according to the amount of their orders. The 20 percent who have ordered the most expensive items are given an M score of 1, the next 20 percent are given an M score of 2, and so forth, down to the 20 percent who spend the least, who are given an M score of 5.

Figure 13-19 shows sample RFM data for Heather Sweeney Designs. (Note that these data have *not* been calculated and are for illustrative purposes only.) The first customer, Ralph Able, has a score of {1 1 2}, which means that he has ordered recently and orders frequently. His M score of 2 indicates, however, that he does not order the most expensive goods. From these scores, the salespeople can surmise that Ralph is a good customer but that they should attempt to up-sell Ralph to more expensive goods.

Susan Baker (RFM score of {2 2 3}) is above average in terms of how frequently she shops and how recently she has shopped, but her purchases are average in value. Sally George (RFM score of {3 3 3}) is truly in the middle. Jenny Tyler (RFM score of {5 1 1}) is a problem. Jenny has not ordered in some time, but, in the past, when she did order, she ordered frequently, and her orders were of the highest monetary value. These data suggest that Jenny might be going to another vendor. Someone from the sales team should contact her immediately. However, no one on the sales team should be talking to Chantel Jacobs (RFM score of {5 5 5}). She has not

Figure 13-19
The RFM Score Report

Each customer is ranked for **R** (recent), **F** (frequent), and **M** (money) characteristics—1 is highest (best) and 5 is lowest (worst) score

	RFM Score		
Customer	R	F	M
Able, Ralph	1	1	2
Baker, Susan	2	2	3
George, Sally	3	3	3
Tyler, Jenny	5	1	1
Jacobs, Chantel	5	5	5

Amazon Ex. 1008.0564

ordered for some time, she doesn't order frequently, and when she does order, she only buys inexpensive items and not many of them.

Producing the RFM Report

Like most reports, an RFM report can be created using a series of SQL expressions. This section presents two SQL Server stored procedures that produce RFM scores. Figure 13-20 shows the five tables that are used.

The CUSTOMER_SALES table contains the raw data that is used in the RFM calculations. CUSTOMER_RFM contains CustomerID and the final R, F, and M scores. The remaining three tables, CUSTOMER_R, CUSTOMER_F, and CUSTOMER_M are used to store intermediate results. Note that all CustomerID columns are NOT NULL.

The stored procedure shown in Figure 13-21 is used to calculate and store the R, F, and M scores. It begins by deleting the results from any prior analysis from the CUSTOMER_R, CUSTOMER_F, and CUSTOMER_M tables. It then calls three procedures for computing the R, F, and M scores. Finally, it stores the R, F, and M scores in the CUSTOMER_RFM table. Only this table is needed for reporting purposes.

Figure 13-20
SQL Server Tables for RFM Analysis

```
CREATE TABLE CUSTOMER_SALES (
   CustomerID           Int          NOT NULL,
   TransactionID        Int          NOT NULL,
   TransactionDate      DateTime     NOT NULL,
   OrderAmt             Money        NOT NULL,
   CONSTRAINT     Customer_Sales_PK
                        PRIMARY KEY(TransactionID)
   );

CREATE TABLE CUSTOMER_RFM (
   CustomerID           Int          NOT NULL,
   R                    SmallInt     NULL,
   F                    SmallInt     NULL,
   M                    SmallInt     NULL,
   CONSTRAINT     Customer_RFM_PK
                        PRIMARY KEY(CustomerID)
   );

CREATE TABLE CUSTOMER_R (
   CustomerID           Int          NOT NULL,
   MostRecentOrderDate  DateTime     NULL,
   R                    SmallInt     NULL,
   CONSTRAINT     Customer_R_PK
                        PRIMARY KEY(CustomerID)
   );

CREATE TABLE CUSTOMER_F (
   CustomerID           Int          NOT NULL,
   OrderCount           Int          NULL,
   R                    SmallInt     NULL,
   CONSTRAINT     Customer_F_PK
                        PRIMARY KEY(CustomerID)
   );

CREATE TABLE CUSTOMER_M (
   CustomerID           Int          NOT NULL,
   AverageOrderAmount   Money        NULL,
   R                    SmallInt     NULL,
   CONSTRAINT     Customer_M_PK
                        PRIMARY KEY(CustomerID)
   );
```

Amazon Ex. 1008.0565

```
CREATE PROCEDURE RFM_Analysis

AS

/*** Delete any existing RFM data   ***/
DELETE FROM CUSTOMER_R;
DELETE FROM CUSTOMER_F;
DELETE FROM CUSTOMER_M;

/*** Compute R, F, M Scores  **********/
Exec Calculate_R;
Exec Calculate_F;
Exec Calculate_M;

/*** Display Results  *****************/
SELECT      R_Score, Count(*) AS R_Count
FROM        CUSTOMER_R
GROUP BY    R_Score;

SELECT      F_Score, Count(*) AS F_Count
FROM        CUSTOMER_F
GROUP BY    F_Score;

SELECT      M_Score, Count(*) AS M_Count
FROM        CUSTOMER_M
GROUP BY    M_Score;

/***  Store Results  ******************/
INSERT INTO CUSTOMER_RFM (CustomerID)
      (SELECT  CustomerID
       FROM CUSTOMER_SALES);

UPDATE CUSTOMER_RFM
SET   R =
      (SELECT R_Score
       FROM CUSTOMER_R
       WHERE CUSTOMER_RFM.CustomerID = CUSTOMER_R.CustomerID);

UPDATE CUSTOMER_RFM
SET   F =
      (SELECT F_Score
       FROM CUSTOMER_F
       WHERE CUSTOMER_RFM.CustomerID = CUSTOMER_F.CustomerID);

UPDATE CUSTOMER_RFM
SET   M =
      (SELECT M_Score
       FROM CUSTOMER_M
       WHERE CUSTOMER_RFM.CustomerID = CUSTOMER_M.CustomerID);
```

Figure 13-21
The RFM_Analysis Stored Procedure

The Calculate_R stored procedure shown in Figure 13-22 illustrates how the R score is calculated. This procedure first places the date of each customer's most recent order into the MostRecentOrderDate column. Then, it uses the **SQL Top ... Percent option** of SQL SELECT statements to set the R_Score values. The first UPDATE statement sets the value of R_Score to 1 for the top 20 percent of customers (after they have been sorted in descending order according to MostRecentOrderDate). Then, it sets the R_Score to 2 for the top 25 percent of

Amazon Ex. 1008.0566

```
CREATE PROCEDURE Calculate_R

AS

/*** Compute R_Score  ********************************/

INSERT INTO CUSTOMER_R (CustomerID, MostRecentOrderDate)
      (SELECT  CustomerID, Max (TransactionDate)
       FROM CUSTOMER_SALES
       GROUP BY CustomerID);

UPDATE   CUSTOMER_R
SET      R_Score = 1
WHERE    CustomerID IN
         (Select Top 20 Percent CustomerID
          FROM CUSTOMER_R
          ORDER BY MostRecentOrderDate DESC);

UPDATE   CUSTOMER_R
SET      R_Score = 2
WHERE    CustomerID IN
         (Select Top 25 PERCENT CustomerID
          FROM CUSTOMER_R
          WHERE R_Score IS NULL
          ORDER BY MostRecentOrderDate DESC);

UPDATE   CUSTOMER_R
SET      R_Score = 3
WHERE    CustomerID IN
         (Select Top 33 PERCENT CustomerID
          FROM CUSTOMER_R
          WHERE R_Score IS NULL
          ORDER BY MostRecentOrderDate DESC);

UPDATE   CUSTOMER_R
SET      R_Score = 4
WHERE    CustomerID IN
         (Select Top 50 PERCENT CustomerID
          FROM CUSTOMER_R
          WHERE R_Score IS NULL
          ORDER BY MostRecentOrderDate DESC);

UPDATE   CUSTOMER_R
SET      R_Score = 5
WHERE    CustomerID IN
         (Select CustomerID
          FROM CUSTOMER_R
          WHERE R_Score IS NULL);
```

Figure 13-22

The Calculate_R Stored Procedure

customers who have a null value for R_Score in descending order of MostRecentOrderDate. The procedure continues to set the R values for all customers. The Calculate_F and Calculate_M procedures are similar and will be left to you as exercise 13.57.

Figure 13-23 shows how the CUSTOMER_RFM table is used for reporting purposes. Figure 13-23 shows a SELECT on CUSTOMER_RFM that was prepared using a database of more than 5,000 customers and over 1 million transactions. The preparation of the CUSTOMER_RFM table required less than 30 seconds on a moderately powered personal computer.

The results in Figure 13-23 are interesting, but unless this information is delivered to the correct users, it will be of no ultimate value to the organization. For example, the 12th row in

Amazon Ex. 1008.0567

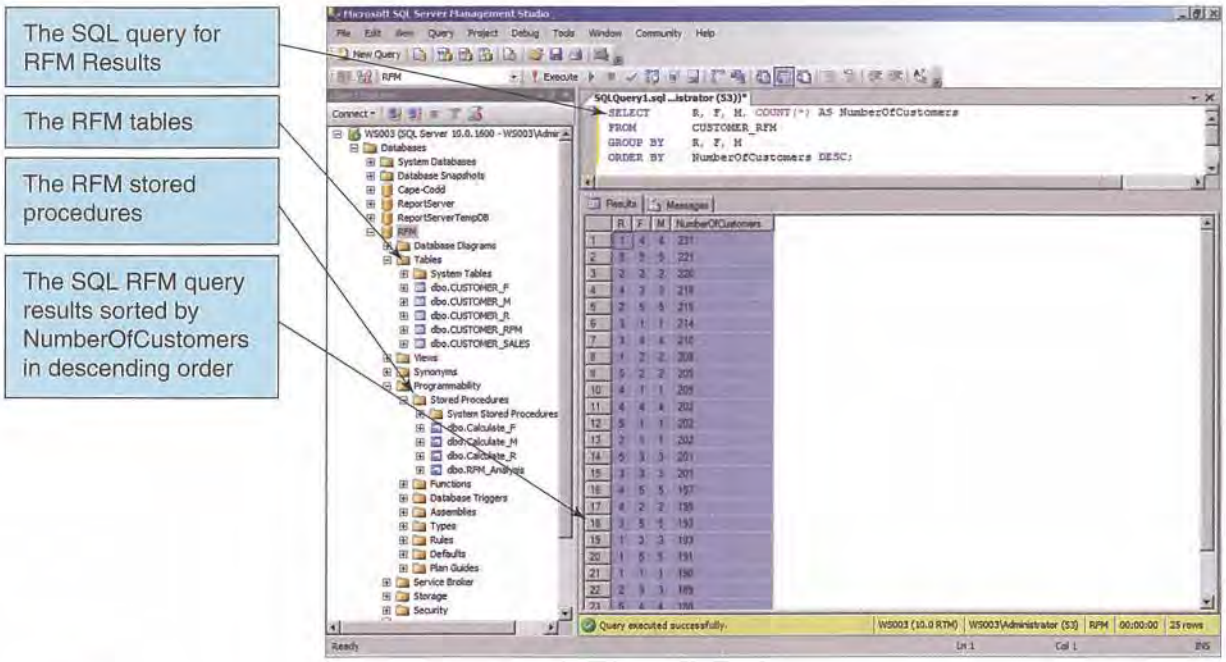


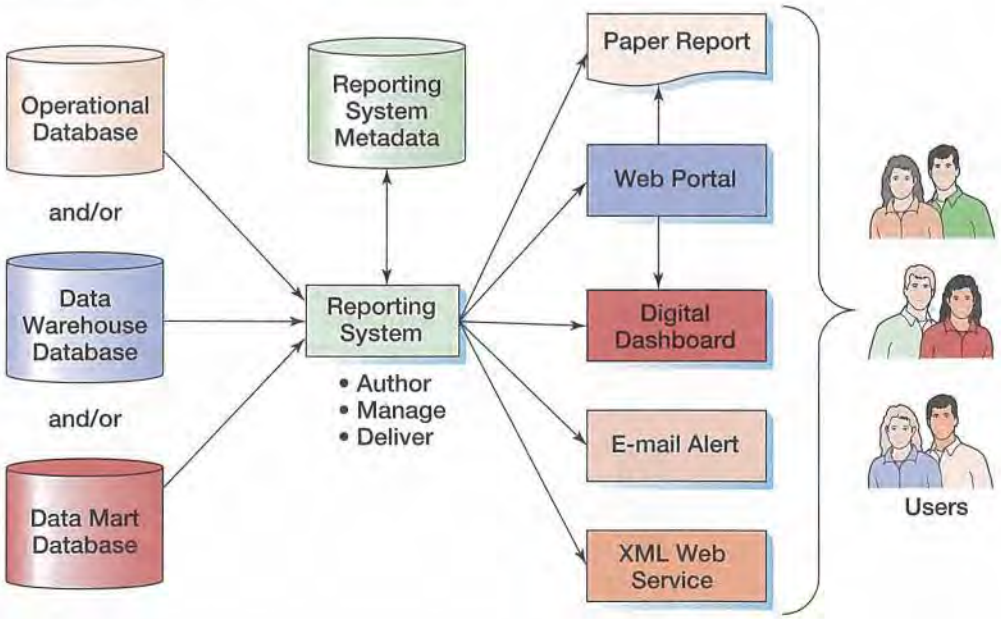
Figure 13-23
Example RFM Results

this figure shows that 202 customers have an RFM score of {5 1 1}. These customers order frequently, they order items of high monetary value, but they have not ordered recently. The company may be in danger of losing them. Somehow this report and the customers who have these scores (see Review Question 13.32) need to be made available to the appropriate sales personnel. To understand the modern means for accomplishing this, we will consider the components of a reporting system.

Reporting System Components

Figure 13-24 shows the major components of a reporting system. Data from disparate data sources are read and processed. As shown, reporting systems can obtain data from operational databases, from data warehouses, and from data marts. Some data are generated within the organization, other data are obtained from public sources, and still other data may be purchased from data utilities.

Figure 13-24
Components of a Reporting System



Amazon Ex. 1008.0568

Figure 13-25
Report Characteristics

Type	Media	Mode
Static	Paper	Push
Dynamic	Web portal	Pull
Query	Digital dashboard	
OnLine Analytical Processing (OLAP)	E-mail/alert	
	XML Web service and application specific	

A reporting system maintains a database of reporting metadata. The metadata describes reports, users, groups, roles, events, and other entities involved in the reporting activity. The reporting system uses the metadata to prepare and deliver appropriate reports to the proper users in the correct format on a timely basis.

As shown in Figure 13-24, reports can be prepared in a variety of formats. Figure 13-25 lists report characteristics.

Report Types

Some reports are **static**. They are prepared once from the underlying data, and they do not change. A report of last year's sales, for example, is a static report. Other reports are **dynamic**; at the time of their creation, the reporting system reads the latest, most current data and generates the report using those fresh data. Reports on today's sales or on current stock prices are dynamic reports.

Query reports are prepared in response to information entered by users. Google is an example of a query report. You enter the keywords you want to search for, and the reporting system within Google searches its database and generates a response that is particular to your query. Within an organization, a query report could be generated to show current inventory levels. The user enters item numbers, and the reporting system responds with inventory levels of those items at various stores and warehouses.

Another type of report is an OLAP, or OnLine Analytical Processing, report. OLAP reports enable the user to dynamically change the report-grouping structures. An OLAP reporting application is illustrated later in this chapter.

Report Media

Today, reports are delivered via many different channels. Some reports are printed on paper, others are created in formats such as PDF that can be printed or viewed electronically. Other reports are delivered via Web portals. An organization might place sales reports on the sales department's Web portal and a report on customers serviced on the customer service department's Web portal.

A **digital dashboard** is an electronic display that is customized for a particular user. Companies such as Yahoo! and MSN offer digital dashboard services. Users of these services can define the content they want to see—say, a local weather forecast, a list of stock prices, and a list of news sources—and the vendor constructs the customized display for each user. Such pages are called My Yahoo! or My MSN, or some similar title. Other dashboards are particular to an organization. Executives at a manufacturing organization, for example, might have a dashboard that shows up-to-the-minute production and sales activities.

Reports can also be delivered via **alerts**. Users can declare that they wish to be notified of news and events by e-mail or on their cell phone. Of course, some cell phones are capable of displaying Web pages and can use digital dashboards as well.

Amazon Ex. 1008.0569

Finally, reports can be delivered to other information systems. The modern way to do this is to publish reports as an XML Web service. This style of reporting is particularly useful for interorganizational information systems, such as supply chain management.

Report Modes

The final report characteristic in Figure 13-25 is report mode. A **push report** is sent to users based on a predetermined schedule. Users receive the report without any activity on their part. A **pull report** is one that users must request. To obtain a pull report, a user goes to a Web portal or digital dashboard and clicks a link or button to cause the reporting system to produce and deliver the report.

Report System Functions

As shown in Figure 13-25, the **reporting system** serves three functions: authoring, management, and delivery.

Report Authoring

Report authoring involves connecting to the required data sources, creating the report structure, and formatting the report. Figures 13-26 and 13-27 show the use of SQL Server Business Intelligence Development Studio to author a report that publishes the results of an RFM analysis done in SQL Server. The SQL Server Business Intelligence Development Studio is a version of Microsoft Visual Studio .NET that is included with SQL Server 2008 (it is one of the tools in the SQL Server 2008 Express Advanced as well as being included in all major versions of SQL Server 2008).

In Figure 13-26, the developer has specified a database that contains the CUSTOMER_RFM table and has just entered the SQL statement shown in Figure 13-23. You can see the SQL statement in the lower-center portion of this display.

In Figure 13-27, the report author creates the format of the report by specifying the headings and selecting the format for the data items. In a more complicated report, the author would specify the sorting and grouping of data items, as well as page headers and footers. The developer uses the property list in the right-hand side of the display in Figure 13-27 to set the values for item

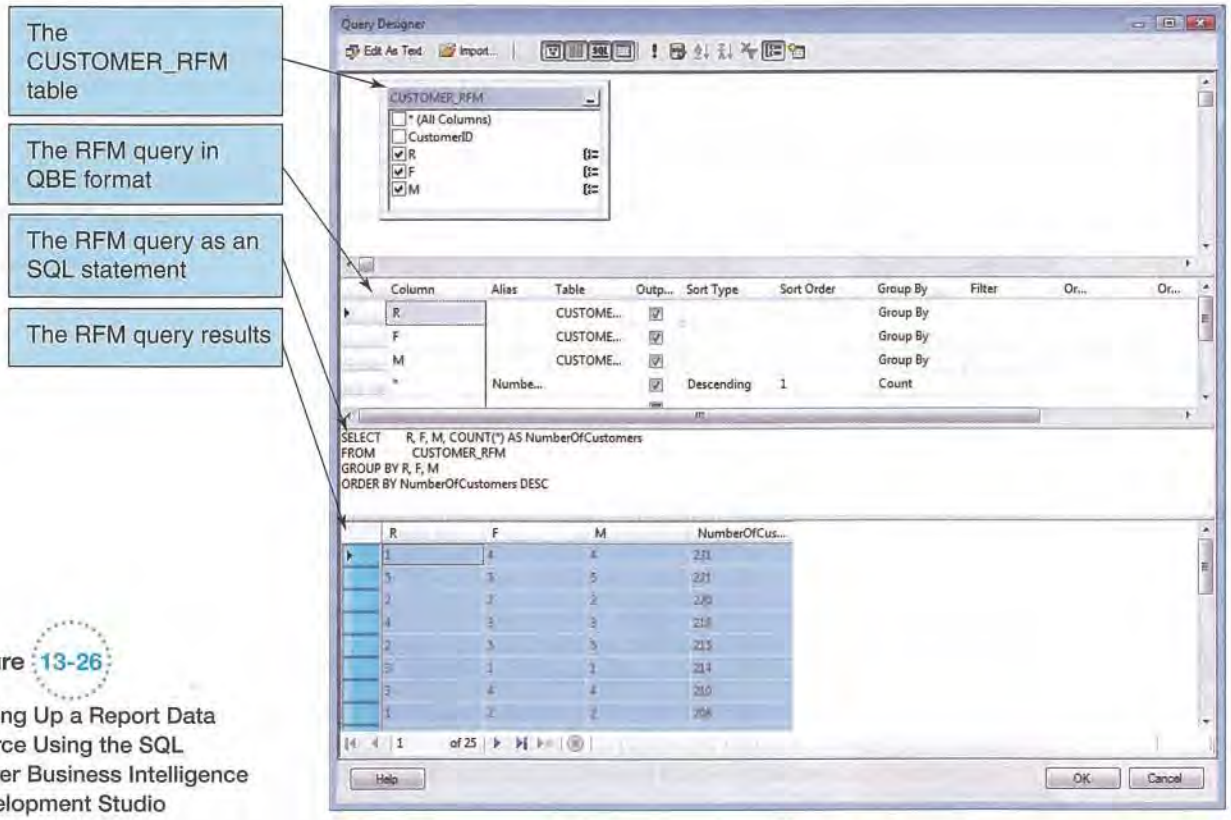


Figure 13-26

Setting Up a Report Data Source Using the SQL Server Business Intelligence Development Studio

Amazon Ex. 1008.0570

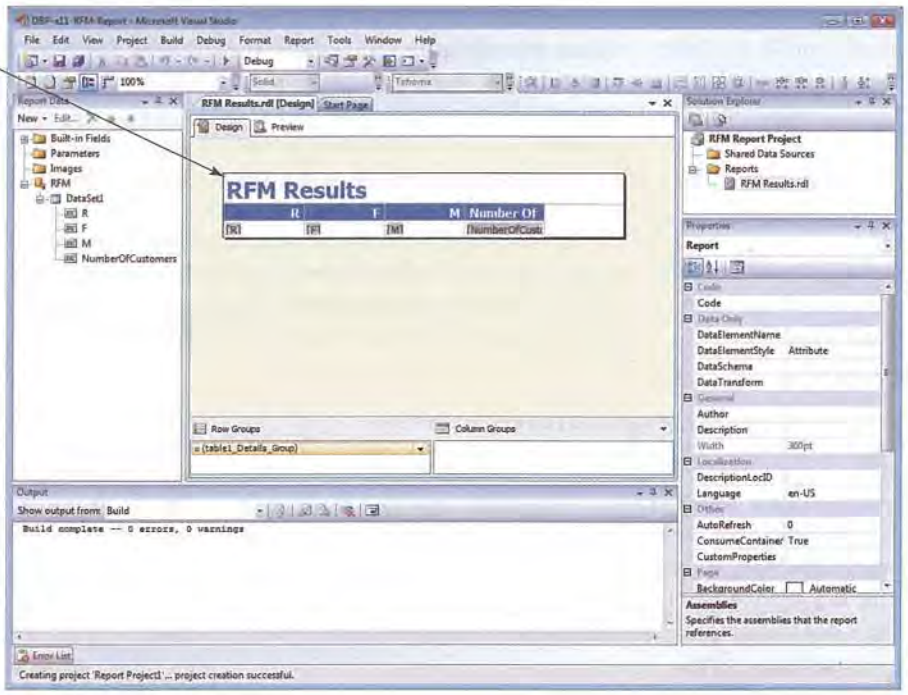


Figure 13-27

Formatting a Report Using the SQL Server Business Intelligence Developement Studio

properties. The final report, as it appears in a browser window, is shown in Figure 13-28. To learn more about this application, search for "reporting services" at *www.microsoft.com*.

Report Management

The purpose of **report management** is to define who receives what reports, when, and by what means. Most report management systems enable the report administrator to define user accounts and user groups and to assign particular users to particular groups. For example, all of the salespeople would be assigned to the sales group, all of the executives would be assigned to the executive group, and so forth. All of these data are stored in the reporting system metadata shown in Figure 13-24.

Reports created using the report authoring system are assigned to groups and users. Assigning reports to groups saves the administrator work; when a report is created, changed,

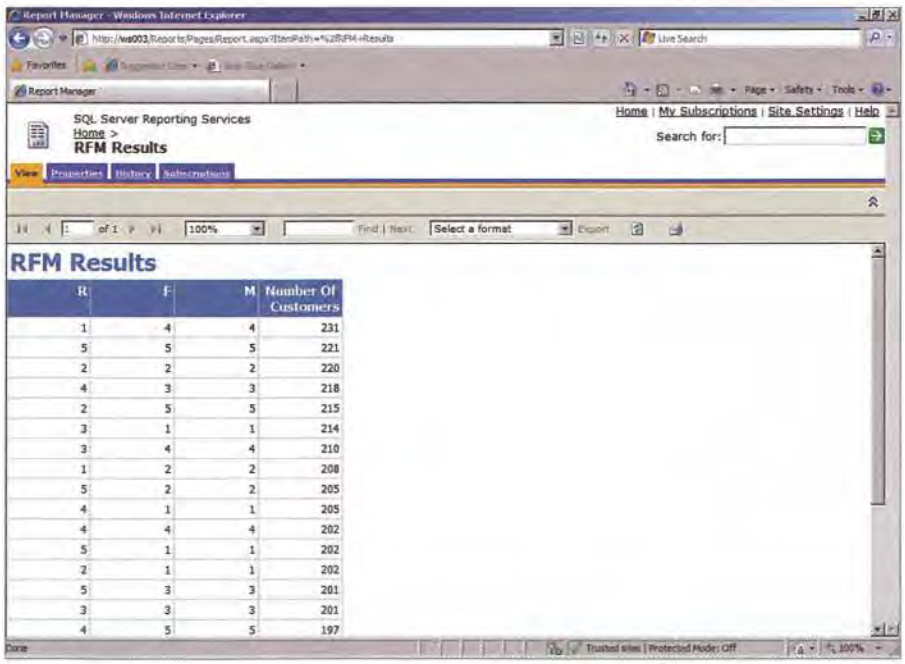


Figure 13-28

The RFM Report in a Web Browser

Amazon Ex. 1008.0571

or removed, the administrator need only change the report assignments to the group. All of the users in the group will inherit the changes. The report assignment metadata includes not only the user or group and the reports assigned, but also indicates the format of the report that should be sent to that user and the channel by which the report will be delivered. For example, Figure 13-29 shows part of the RFM report materialized in XML. The XML file can then be input into any program that consumes XML and manipulated via XSL, as described in Chapter 12.

As stated earlier, the report management metadata indicates which format of the report should be sent to which user. It also indicates what channel is to be used and whether the report is to be pushed or pulled. If pushed, the administrator declares whether the report is to be generated on a regular schedule or as an alert.

Figure 13-29

The RFM Report in XML Format

```
<?xml version="1.0" encoding="UTF-8"?>
<MyData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="C:\inetpub\wwwroot\DBP\VRG\DBP-e11-Figure-13-29-B.xsd">
    <CUSTOMER_RFM>
        <R>1</R>
        <F>4</F>
        <M>4</M>
        <NumberOfCustomers>231</NumberOfCustomers>
    </CUSTOMER_RFM>
    <CUSTOMER_RFM>
        <R>5</R>
        <F>5</F>
        <M>5</M>
        <NumberOfCustomers>221</NumberOfCustomers>
    </CUSTOMER_RFM>
    <CUSTOMER_RFM>
        <R>2</R>
        <F>2</F>
        <M>2</M>
        <NumberOfCustomers>220</NumberOfCustomers>
    </CUSTOMER_RFM>
    <CUSTOMER_RFM>
        <R>4</R>
        <F>3</F>
        <M>3</M>
        <NumberOfCustomers>218</NumberOfCustomers>
    </CUSTOMER_RFM>
    <CUSTOMER_RFM>
        <R>2</R>
        <F>5</F>
        <M>5</M>
        <NumberOfCustomers>215</NumberOfCustomers>
    </CUSTOMER_RFM>
    <CUSTOMER_RFM>
        <R>3</R>
        <F>1</F>
        <M>1</M>
        <NumberOfCustomers>214</NumberOfCustomers>
    </CUSTOMER_RFM>
    <CUSTOMER_RFM>
        <R>3</R>
        <F>4</F>
        <M>4</M>
        <NumberOfCustomers>210</NumberOfCustomers>
    </CUSTOMER_RFM>
    <CUSTOMER_RFM>
        <R>1</R>
        <F>2</F>
        <M>2</M>
        <NumberOfCustomers>208</NumberOfCustomers>
    </CUSTOMER_RFM>
</MyData>
```

Amazon Ex. 1008.0572

Report Delivery

The **report delivery** function of a reporting system pushes reports or allows them to be pulled based on the report management metadata. Reports can be delivered via an e-mail server, a Web portal, XML Web services, or by other program-specific means. The report delivery system uses the operating system and other program security components to ensure that only authorized users receive authorized reports. It also ensures that push reports are produced at appropriate times.

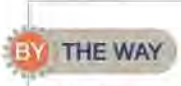
For query reports, the report delivery system serves as an intermediary between the user and the report generator. It receives user query data, such as the item numbers in an inventory query, passes the query data to the report generator, receives the resulting report, and delivers the report to the user.

OLAP

OLAP provides the ability to sum, count, average, and perform other simple arithmetic operations on groups of data. OLAP systems produce **OLAP reports**. An OLAP report is also called an **OLAP cube**. This is a reference to the dimensional data model, and some OLAP products show OLAP displays using three axes, like a geometric cube. The remarkable characteristic of an OLAP report is that it is dynamic: The format of an OLAP report can be changed by the viewer, hence the term *online* in the name OnLine Analytical Processing.

OLAP uses the dimensional database model discussed earlier in this chapter, so it is not surprising to learn that an OLAP report has measures and dimensions. A measure is a dimensional model *fact*—the data item of interest that is to be summed or averaged or otherwise processed in the OLAP report. For example, sales data may be summed to produce Total Sales or averaged to produce Average Sales. The term *measure* is used because you are dealing with quantities that have been or can be measured and recorded. A dimension, as you have already learned, is an attribute or a characteristic of a measure. Purchase date (TimeID), customer location (City), and sales region (ZIP or State) are all examples of dimensions, and in the HSD-DW database, you saw how the time dimension is important.

In this section we will generate an OLAP report by using an SQL query from the HSD-DW database and a Microsoft Excel **PivotTable**.



We use Microsoft SQL Server and Microsoft Excel to illustrate this discussion of OLAP reports and PivotTables. For other DBMS products, such as MySQL, you can use the DataPilot feature of the Calc spreadsheet application in the OpenOffice.org product suite (see *www.openoffice.org*).

Now we can either create an Excel-formatted table in an Excel worksheet:

- Copy the SQL query results into an Excel worksheet.
- Add column names to the results.
- Format the query results as an Excel table (optional).
- Select the Excel range containing the results with column names.

Or connect to a DBMS data source:

- Click the PivotTable button in the Tables group of the Insert ribbon.
- Specify that the PivotTable should be in a new worksheet.
- Select the column variables (Column Labels), row variables (Row Labels), and the measure to be displayed (Values).

Amazon Ex. 1008.0573

We can use an SQL query if we copy the data into an Excel worksheet. The SQL query, as used in SQL Server, is:

```
SELECT      C.CustomerID, CustomerName, C.City,
            P.ProductNumber, P.ProductName,
            T.[Year], T.QuarterText,
            SUM(PS.Quantity) AS TotalQuantity
FROM        CUSTOMER C, PRODUCT_SALES PS, PRODUCT P, TIMELINE T
WHERE       C.CustomerID = PS.CustomerID
   AND      P.ProductNumber = PS.ProductNumber
   AND      T.TimeID = PS.TimeID
GROUP BY    C.CustomerID, C.CustomerName, C.City,
            P. ProductNumber, P.ProductName,
            T.QuarterText, T.[Year]
ORDER BY    C.CustomerName, T.[Year], T.QuarterText;
```

However, because SQL Server (and other SQL-based DBMS products, such as Oracle and MySQL) can store views but not queries, we need to create and use an SQL view if we are going to use an Excel-data connection. The SQL query to create the HSDDWProductSalesView, as used in SQL Server, is:

```
CREATE VIEW HSDDWProductSalesView AS
  SELECT      C.CustomerID, C.CustomerName, C.City,
              P.ProductNumber, P.ProductName,
              T.[Year], T.QuarterText,
              SUM(PS.Quantity) AS TotalQuantity
  FROM        CUSTOMER C, PRODUCT_SALES PS, PRODUCT P, TIMELINE T
  WHERE       C.CustomerID = PS.CustomerID
    AND       P.ProductNumber = PS.ProductNumber
    AND       T.TimeID = PS.TimeID
  GROUP BY    C.CustomerID, C.CustomerName, C.City,
              P. ProductNumber, P.ProductName,
              T.QuarterText, T.[Year];
```

Figure 13-30 shows the OLAP report as an Excel PivotTable. Here, the measure is quantity sold, and the dimensions are ProductNumber and City. This report shows how quantity varies by product and city. For example, four copies of VB003 (Kitchen Remodeling Dallas Style Video Companion) were sold in Dallas, but none were sold in Austin.

We generated the OLAP report in Figure 13-30 by using a simple SQL query and Microsoft Excel, but many DBMS and BI products include more powerful and sophisticated tool. For example, SQL Server includes SQL Server Analysis Services.[1] It is possible to display OLAP cubes in many ways besides with Excel. Some third-party vendors provide more sophisticated graphical displays, and OLAP reports can be delivered just like any of the other reports described for report management systems.

[1] Up to this point in this book, we have been able to do everything in this book using SQL Server 2008 Express. Unfortunately, SQL Server Express Edition does not include SQL Server Analysis Services, so you will have to use the SQL Server Standard Edition or better if you want to use the SQL Server Analysis Services. Although OLAP reports *can* be done without SQL Server Analysis Services, Server Analysis Services adds a lot of functionality, and the Microsoft SQL Server 2008 Data Mining Add-ins for Microsoft Office 2007 (used in this text) will not function without it.

Amazon Ex. 1008.0574

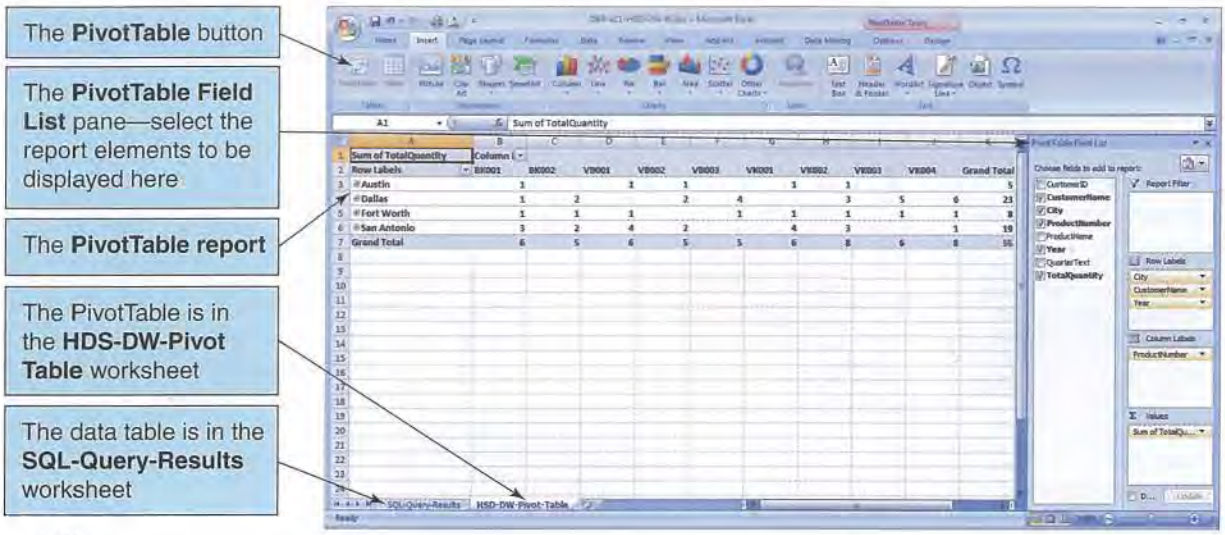


Figure 13-30

OLAP ProductNumber by City Report

The distinguishing characteristic of an OLAP report is that the user can alter the format of the report. Figure 13-30 shows an alteration in which the user added two additional dimensions, customer and year, to the horizontal display. Quantity sold is now broken out by customer and, in one case, by year. With an OLAP report, it is possible to **drill down** into the data—that is, to further divide the data into more detail. In Figure 13-31, for example, the user has drilled down into the San Antonio data to display all customer data for that city and to display year sales data for Ralph Able.

In an OLAP report, it is also possible to change the order of the dimensions. Figure 13-32 shows city quantities as vertical data, and ProductID quantities as horizontal data. This OLAP report shows quantity sold by city, by product, customer, and year.

Both displays are valid and useful, depending on the user's perspective. A product manager might like to see product families first (ProductID) and then location data (city). A sales manager might like to see location data first and then product data. OLAP reports provide both perspectives, and the user can switch between them while viewing a report.

Unfortunately, all of this flexibility comes at a cost. If the database is large, doing the necessary calculating, grouping, and sorting for such dynamic displays will require substantial computing power. Although standard, commercial DBMS products do have the features and functions required to create OLAP reports, they are not designed for such work. They are designed to provide rapid response to transaction processing applications such as order entry or manufacturing planning.

Accordingly, special-purpose products called **OLAP servers** have been developed to perform OLAP analyses. As shown in Figure 13-33, an OLAP server reads data from an operational database, performs preliminary calculations, and stores the results of those calculations in an OLAP database. For performance and security reasons, the OLAP server and the DBMS usually run on separate computers. The OLAP server would normally be located in the data warehouse or a data mart.

Figure 13-31

OLAP ProductNumber by City, Customer, and Year Report

The City = San Antonio data are also showing customer data

The Customer = Able, Ralph data are also showing year data

Sum of TotalQuantity	Column									
Row Labels	BK001	BK002	VB001	VB002	VB003	VK001	VK002	VK003	VK004	Grand Total
Austin		1		1	1		1	1		5
Pearson, Bobbi		1		1	1		1	1		5
Dallas		1	2		2	4		3	5	6	23
Foxtrot, Kathy			1			1		1	1	1	5
George, Sally		1	1						1	2	5
Hullett, Shawn						1			1	1	3
Tyler, Jenny					2	2		2	2	2	10
Fort Worth		1	1	1		1	1	1	1	1	8
Jacobs, Nancy		1		1			1				3
Wayne, Joan			1			1		1	1	1	5
San Antonio		3	2	4	2		4	3		1	19
Able, Ralph		1	1	2	2		2	2			10
2007				1			1				2
2008		1	1	1	2		1	2			8
Baker, Susan		1	1	1			1	1		1	6
Eagleton, Sam		1		1			1				3
Grand Total		6	5	6	5	5	6	8	6	8	55

Amazon Ex. 1008.0575

The city variable is on the column designator

The ProductID variable is on the primary row designator

The ProductID = VB001 data are also showing **Customer** data

The Customer = Able, Ralph data are also showing year data

Sum of TotalQuantity	Column Labels				
Row Labels	Austin	Dallas	Fort Worth	San Antonio	Grand Total
⊟BK001		1	1	1	3
⊟Able, Ralph					1
2008					1
⊞Baker, Susan					1
⊞Eagleton, Sam					1
⊞George, Sally			1		
⊞Jacobs, Nancy				1	
⊞Pearson, Bobbi		1			
⊟BK002			2	1	2
⊟Able, Ralph					1
2008					1
⊞Baker, Susan					1
⊞Foxtrot, Kathy			1		
⊞George, Sally			1		
⊞Wayne, Joan				1	
⊟VB001		1		1	4
⊟Able, Ralph					2
2007					1
2008					1
⊞Baker, Susan					1
⊞Eagleton, Sam					1
⊞Jacobs, Nancy				1	
⊞Pearson, Bobbi		1			
⊞VB002		1	2		2
⊞VB003			4	1	
⊞VK001		1		1	4
⊞VK002		1	3	1	3
⊞VK003			5	1	
⊞VK004			6	1	1
Grand Total		5	23	8	19

Figure 13-32

OLAP City by ProductNumber, Customer, and Year Report

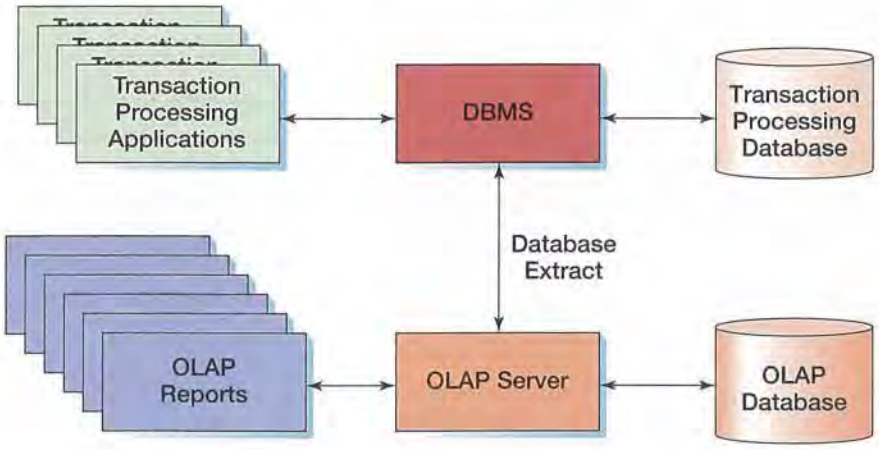


Figure 13-33

The Role of the OLAP Server and OLAP Database

Data Mining

Instead of the basic calculations filtering, sorting, and grouping used in reporting applications, data mining involves the application of sophisticated mathematical and statistical techniques to find patterns and relationships that can be used to classify data and predict future outcomes. As shown in Figure 13-34, data mining represents the convergence of several phenomena. Data mining techniques have emerged from the statistical and mathematics disciplines and from the artificial intelligence and machine-learning communities. In fact, data mining terminology is an odd combination of terms used by these different disciplines.

Data mining techniques take advantage of developments for processing enormous databases that have emerged in the past dozen or so years. Of course, all these data would not have been generated were it not for fast and inexpensive computers, and, without such computers, the new techniques would be impossible to compute.

Most data mining techniques are sophisticated and difficult to use. However, such techniques are valuable to organizations, and some business professionals, especially those in finance and marketing, have developed expertise in their use. Almost all data mining techniques require specialized software. Popular data mining products are Enterprise Miner from SAS Corporation, Clementine from SPSS, and Insightful Miner from Insightful Corporation.

Amazon Ex. 1008.0576

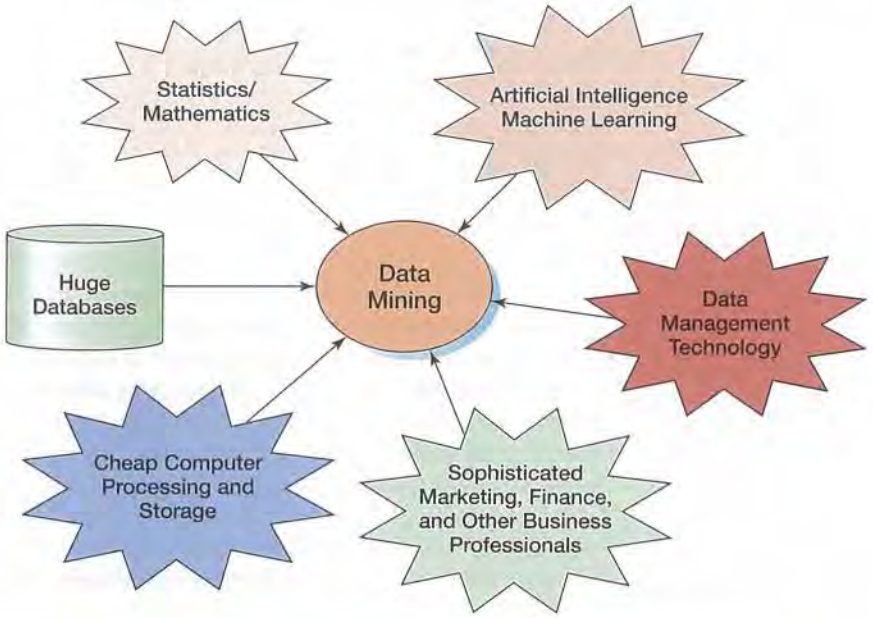


Figure 13-34

Convergence of Disciplines for Data Mining

However, there is a movement to make data mining available to more users. For example, Microsoft has created the Microsoft SQL Server 2008 Data Mining Add-ins for Office 2007.[2] Figure 13-35 shows Excel 2007 with the Data Mining command tab and command groups. With this add-in, data stored in Excel are sent to SQL Server Analysis Services for processing, and the results are returned to Excel for display.

Data mining techniques fall into two broad categories: unsupervised and supervised.

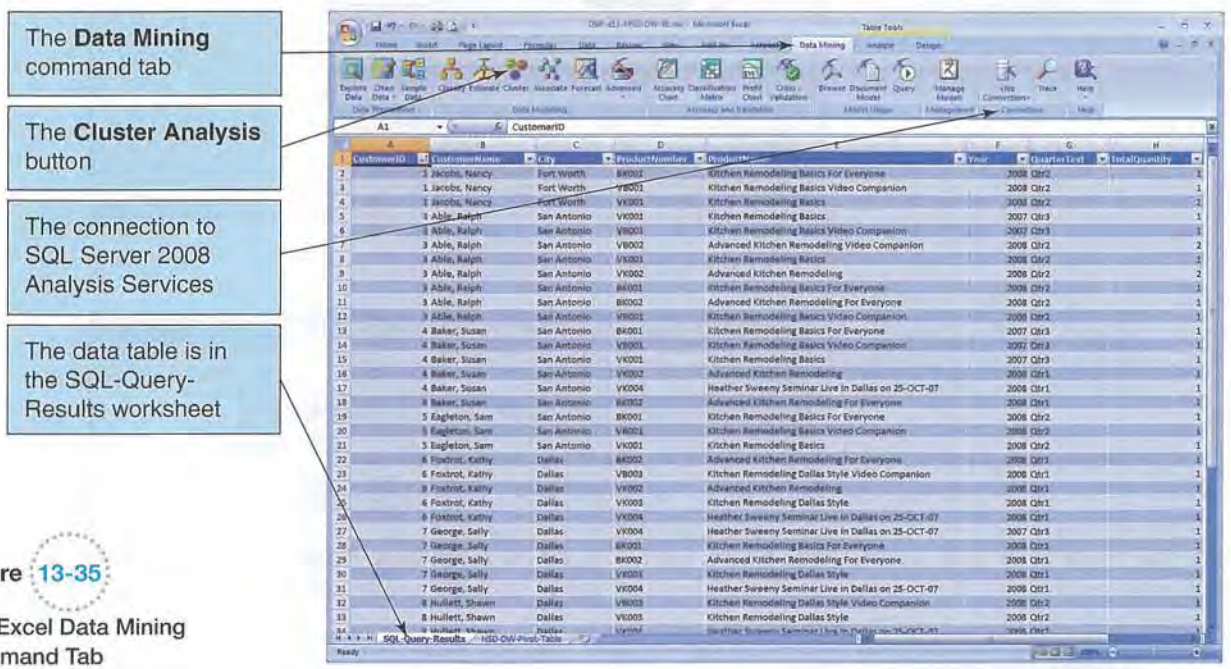


Figure 13-35

The Excel Data Mining Command Tab

[2] For more information, and to download the Microsoft SQL Server 2005 Data Mining Add-ins for Office 2007 package, go to *www.microsoft.com/sql/technologies/dm/addins.mspx* or directly to the download Web page at *www.microsoft.com/downloads/details.aspx?FamilyId=7c76e8df-8674-4c3b-a99b-55b17f3c4c51&displaylang=en*. The Microsoft SQL Server 2008 Data Mining Add-ins for Office 2007 are available at *www.microsoft.com/sqlserver/2008/en/us/data-mining.aspx* . Note, however, that these add-ins will not work with SQL Server Express Edition. You have to have a version of SQL Server with SQL Server Analysis Services.

Amazon Ex. 1008.0577

Unsupervised Data Mining

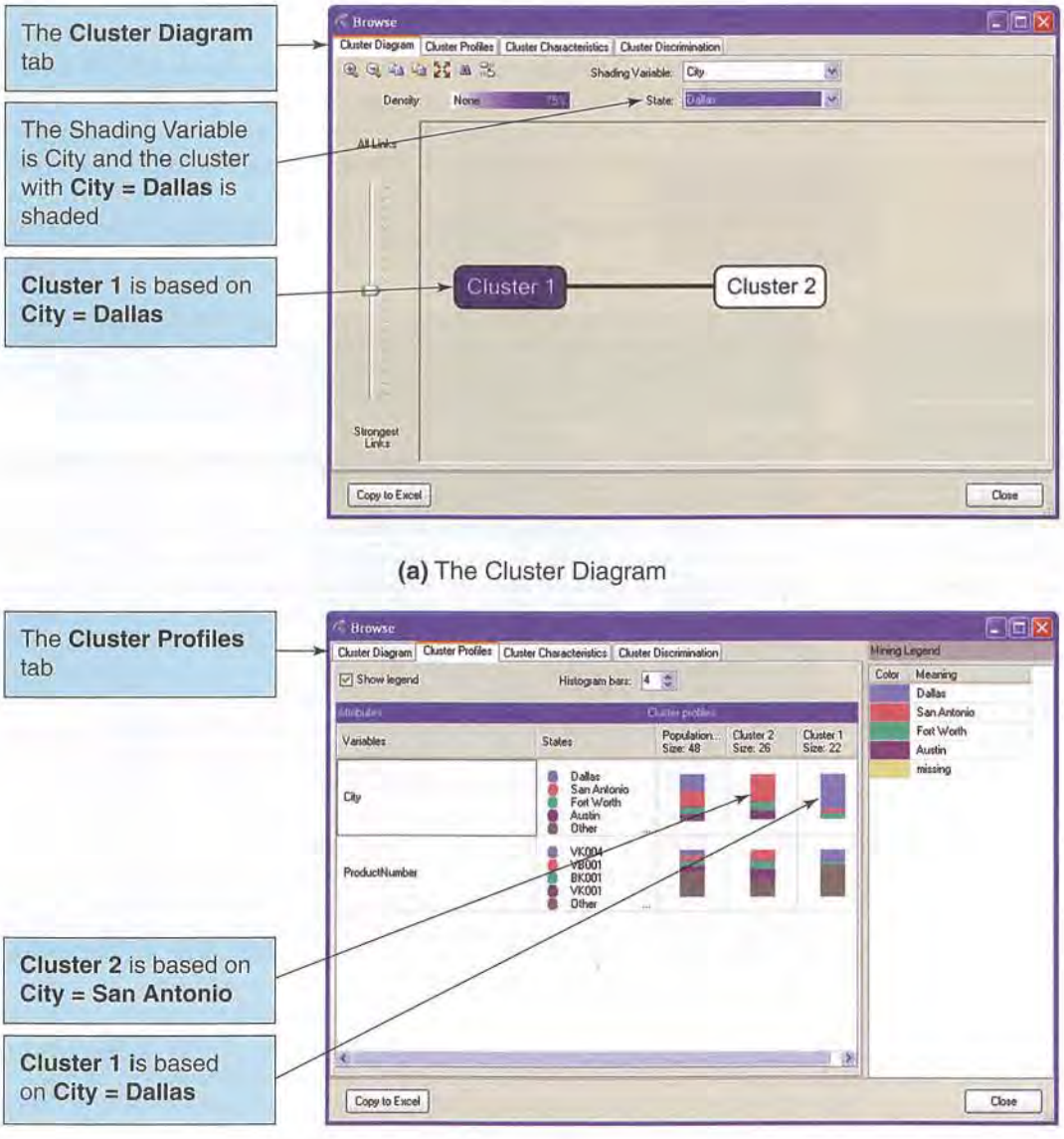
When using **unsupervised data mining** techniques, analysts do not create a model or hypothesis prior to beginning the analysis. Instead, the data mining technique is applied to the data, and results are observed. After the analysis, explanations and hypotheses are created to explain the patterns found.

One commonly used unsupervised technique is **cluster analysis**. With cluster analysis, statistical techniques are used to identify groups of entities that have similar characteristics. A common use for cluster analysis is to find customer groups in order and customer demographic data. For example, Heather Sweeney Designs could use cluster analysis to determine which groups of customers are associated with the purchase of specific products.

Figure 13-36 shows part of the results of a cluster analysis using the same HSD-DW data table in Microsoft Excel that we used to create the OLAP reports. In this case, the cluster analysis tool was asked to generate exactly two clusters based on the ProductID and City variables. Based on the analysis, it is clear that there are different sales patterns for the Dallas area and the non-Dallas area (primarily San Antonio).

These findings were obtained solely by data analysis. No model was used to find these patterns and relationships. The analysis speaks for itself. It is up to the analyst to form hypotheses, after the fact, to explain why these results were obtained.

Figure 13-36
The Cluster Analysis Results



Amazon Ex. 1008.0578

Supervised Data Mining

With **supervised data mining**, data miners develop a model prior to the analysis and then apply statistical techniques to the data to estimate parameters of the model. For example, suppose that marketing experts at a communications company believe that the use of cell phone weekend minutes is determined by the age of the customer and the number of months the customer has had the cell phone account. A data mining analyst would then run an analysis called a **regression analysis** to determine the coefficients of the equation of that model. A possible result is:

CellPhoneWeekendMinutes = 12 + (17.5 * CustomerAge) + (23.7 * NumberMonthsOfAccount)

As you will learn in your statistics classes, considerable skill is required to interpret the quality of such a model. The regression tool will create an equation; whether the equation is a good predictor of future cell phone usage depends on *t* values, confidence intervals, and related statistical techniques.

Three Popular Data Mining Techniques

Three popular data mining techniques are decision trees, logistic regression, and neural networks. **Decision tree analysis** classifies customers or other entities of interest into two or more groups according to past history. **Logistic regression** produces equations that offer probabilities that particular events will occur. Common applications of logistic regression are using donor characteristics to predict the likelihood of a donation in a given period and using customer characteristics to predict the likelihood that customers will switch to another vendor.

Neural networks are complex statistical prediction techniques. The name is a misnomer. Although there is some loose similarity between the structure of a neural network and a network of biological neurons, the similarity is only superficial. Data mining neural networks are just a technique for creating very complex mathematical functions for making predictions.

These three techniques, like almost all data mining techniques, require specialized software. Popular data mining products are Enterprise Miner from the SAS Corporation, Clementine from SPSS, and Insightful Miner from the Insightful Corporation. All of these products have facilities for importing data from relational databases, and, as a database professional, you may be asked to prepare data for input to a data mining product. Typically, this work involves joining relations together into a large flat file and then filtering the data for particular data cases. Simple SQL is used to create such files.

Market Basket Analysis

However, **market basket analysis** is a data mining technique that can be readily implemented with pure SQL. What is market basket analysis?

Suppose that you run a dive shop and one day you realize that one of your salespeople is much better at up-selling your customers. Any of your sales associates can fill a customer's order, but this one salesperson is especially able to sell customers items in addition to those for which they ask. One day you ask him how he does it.

"It's simple," he says, "I just ask myself what is the next product they'd want to buy. If someone buys a dive computer, I don't try to sell her fins. If she's buying a dive computer, she's already a diver, and she already has fins. But, look, these dive computer displays are hard to read. A better mask makes it easier to read the display and get the full benefits from the dive computer."

A market basket analysis is a data mining technique for determining such patterns. A market basket analysis shows the products that customers tend to purchase at the same time. Several different statistical techniques can be used to generate a market basket analysis. Here we will discuss a technique that involves conditional probabilities.

Figure 13-37 shows hypothetical data from 1,000 transactions at a dive shop. The first row of numbers under each column is the total number of transactions that include the product in that column. For example, the 270 in the first row of Mask means that 270 of the 1,000 transactions include the purchase of a mask. The 120 under Dive Computer means that 120 of the 1,000 purchased transactions included a dive computer.

We can use the numbers in the first row to estimate the probability that a customer will purchase an item. Because 270 out of 1,000 transactions included a mask, we can estimate the

Amazon Ex. 1008.0579

1,000 Transactions	Mask	Tank	Fins	Weights	Dive Computer
	270	200	280	130	120
Mask	20	20	150	20	50
Tank	20	80	40	30	30
Fins	150	40	10	60	20
Weights	20	30	60	10	10
Dive Computer	50	30	20	10	5
No Additional Product	10	–	–	–	5

Support = P (A & B) Example: P (Fins & Mask) = 150 / 1000 = .15
Confidence = P (A | B) Example: P (Fins | Mask) = 150 / 270 = .55556
Lift = P (A | B)/ P (A) Example: P (Fins | Mask) / P (Fins) = .55556 / .28 = 1.98
Note: P (Mask | Fins) / P (Mask) = 150 / 280 / .27 = 1.98

Figure 13-37
Market Basket Example

likelihood that a customer will buy a mask to be 270/1,000, or .27. Similarly, the likelihood of a tank purchase is 200/1,000, or .2, and that for fins is 280/1,000, or .28.

The remaining rows in this table show the occurrences of transactions that involve two items. For example, the last column indicates that 50 transactions included both a dive computer and a mask, 30 transactions included a dive computer and a tank, 20 included a dive computer and fins, 10 included a dive computer and weights, 5 included a dive computer with another dive computer (meaning the customer bought two dive computers), and 5 transactions had a dive computer and no other product.

These data are interesting, but we can refine the analysis by computing additional factors. Marketing professionals define **support** as the probability that two items will be purchased together. From these data, the support for fins and mask is 150 out of 1,000, or .15.

Confidence is defined as the probability of buying one product given that a customer purchased another product. The confidence of fins, given that the customer has already purchased a mask, is the number of purchases of fins and masks out of the number of purchases of masks. Thus, in this example, the confidence is 150 out of 270, or .55556. The confidence that a customer purchases a tank, given that the customer has purchased fins, is 40 out of 280, or .14286.

Lift is defined as the ratio of confidence divided by the base probability of an item purchase. The lift for fins given a mask is the probability that a customer buys fins, given the customer has purchased a mask, divided by the overall probability that the customer buys fins. If the lift is greater than one, then the probability of buying fins goes up when a customer buys a mask; if the lift is less than one, the probability of buying fins goes down when a customer buys a mask.

For the data in Figure 13-37, the lift for fins given a mask purchase is .55556/.28, or 1.98. This means that when someone purchases a mask, the likelihood he or she will also purchase fins almost doubles. The lift for fins given a dive computer purchase is 20/120 (the confidence of fins, given a dive computer) divided by .28. The probability that someone buys fins (280 of the 1,000 transaction involved fins) is 20/120 (or .16667), and .16667/.28 is .59524. Therefore, the lift for fins given a dive computer is just under .6, meaning that when a customer buys a dive computer, the likelihood that he or she will buy fins decreases.

Surprisingly, as shown in the last line of Figure 13-37, lift is symmetric. If the lift of fins given mask is 1.98, then the lift of mask given fins is also 1.98.

Using SQL for Market Basket Analysis

All of the major data mining products have features and functions to perform market basket analysis. These products, however, are expensive; you can perform a market basket analysis with basic SQL, if necessary.

Amazon Ex. 1008.0580

```
CREATE VIEW TwoItemBasket AS
      SELECT      T1.Item AS FirstItem,
                  T2.Item AS SecondItem
      FROM        TRANS_DATA T1, TRANS_DATA T2
      WHERE       T1.TransactionID = T1.TransactionID
            AND   T1.ItemID <> T2.ItemID;
```

Figure 13-38
SQL Statement for Creating a Two-item Market Basket

The key SQL statement is shown in Figure 13-38. That SQL statement processes a relation named TRANS_DATA that stores line-item data. Here, suppose that TRANS_DATA has a column TransactionID that stores an identifier of a transaction, and ItemID that stores the identifier of an item in that transaction. A given transaction may have multiple items, so the key of TRANS_DATA is (TransactionID, ItemID). TRANS_DATA has other data, such as ItemPrice, Qty, and ExtendedPrice, but those data are unnecessary for a market basket analysis, and we ignore them here.

The SQL statement in Figure 13-38 creates a view of all items that have appeared together in two or more transactions. You can then compute support in a view using the following statement:

```
CREATE VIEW ItemSupport AS
   SELECT      FirstItem, SecondItem, COUNT(*) as SupportCount
   FROM        TwoItemBasket
   GROUP BY    FirstItem, SecondItem;
```

This view produces the count of transactions in which each pair of items appears. You can divide the SupportCount in each row by the total number of transactions to obtain support for the two items. You can then use standard SQL to compute confidence and lift for each pair of items. See Project Questions 13.58 and 13.60.

Summary

Business intelligence (BI) systems assist managers and other professionals in the analysis of current and past activities and in the prediction of future events. BI applications are of two major types: reporting applications and data mining applications. Reporting applications make elementary calculations on data; data mining applications use sophisticated mathematical and statistical techniques.

BI applications obtain data from three sources: operational databases, extracts of operational databases, and purchased data. BI systems sometimes have their own DBMS, which may or may not be the operational DBMS. Characteristics of reporting and data mining applications are listed in Figure 13-2.

Direct reading of operational databases is not feasible for all but the smallest and simplest BI applications and databases for several reasons. Querying operational data can unacceptably slow the performance of operational systems, operational data have problems that limit their usefulness for BI applications, and BI system creation and maintenance requires programs, facilities, and expertise that are normally not available for an operational database.

Problems with operational data are listed in Figure 13-4. Because of these problems with operational data, many organizations have chosen to create and staff data warehouses and data marts. Data warehouses extract and clean operational data and store the revised data in data warehouse databases. Organizations may also purchase and manage data obtained from data vendors. Data warehouses maintain metadata that describes the source, format, assumptions, and constraints about the data they contain. A data mart is a collection of data that is smaller than that held in a data warehouse and that addresses a particular component or functional area of the business. In Figure 13-6, the data warehouse distributes data to three smaller data marts. Each data mart services the needs of a different aspect of the business.

Operational databases and dimensional databases have different characteristics, as shown in Figure 13-7. Dimensional

Amazon Ex. 1008.0581

databases use a star schema with a fully normalized fact table that connects to dimension tables that may be non-normalized. Dimensional databases must deal with slowly changing dimensions, and therefore a time dimension is important in a dimensional database. Fact tables hold measures of interest, and dimension tables hold attribute values used in queries. The star schema can be extended with additional fact tables, dimension tables, and conformed dimensions.

The purpose of a reporting system is to create meaningful information from disparate data sources and to deliver that information to the proper users on a timely basis. Reports are produced by sorting, filtering, grouping, and making simple calculations on the data. RFM analysis is a typical reporting application. Customers are grouped and classified according to how recently they have placed an order (R), how frequently they order (F), and how much money (M) they spend on orders. The result of an RFM analysis is three scores. In a typical analysis, the scores range from 1 to 5. An RFM score of {1 1 4} indicates that the customer has purchased recently, purchases frequently, but does not purchase large-dollar items. An RFM report can be produced using simple SQL. Figures 13-21 and 13-22 show stored procedures for computing these scores.

For the RFM data to add value to the organization, an RFM report must be prepared and delivered to the appropriate users. The components of a modern reporting system are shown in Figure 13-24. Reporting systems maintain metadata that supports the three basic report functions: authoring, managing, and delivering reports. The metadata includes information about users, user groups, and reports and data about which users are to receive which reports, in what medium, and when. As shown in Figure 13-24, reports vary by type, media, and mode.

OnLine Analytical Processing (OLAP) is a generic category of reporting applications that enable users to dynamically restructure reports. A measure is the data item of interest. A dimension is a characteristic of a measure. An OLAP cube is an arrangement of measures and dimensions. With OLAP, users can drill down and exchange the order of dimensions. Because of the high processing requirements, some organizations designate separate computers to function as OLAP servers.

Data mining is the application of mathematical and statistical techniques to find patterns and relationships and to classify and predict. Data mining has arisen in recent years because of the confluence of factors shown in Figure 13-34.

With unsupervised data mining, analysts do not create models or hypotheses prior to the analysis. Results are explained after the analysis has been performed. With supervised techniques, hypotheses are formed and tested before the analysis. Three popular data mining techniques are decision trees, logistic regression, and neural networks.

Although most data mining techniques require special-purpose software, one data mining technique, market basket analysis, can be performed by using only SQL. According to market basket analysis terminology, the *support* for two products is the frequency that they appear together in a transaction. The *confidence* is the conditional probability that one item will be purchased given that another item has already been purchased. *Lift* is confidence divided by the base probability that an item will be purchased.

An SQL join statement can be written to create a view showing products that have appeared together in a transaction. That view can then be processed to compute support, and the support view can then be processed to compute confidence and lift.

Key Terms

alert
business intelligence (BI) system
click-stream data
cluster analysis
confidence
conformed dimension
curse of dimensionality
data mart
data mining application
data warehouse
data warehouse metadata database
date dimension
decision support system (DSS)
decision tree analysis
digital dashboard
dimension table
dimensional database
dirty data
drill down
dynamic report
enterprise data warehouse (EDW) architecture
Extract, Transform, and Load (ETL) System
F score
fact table
lift
logistic regression
M score
market basket analysis
measure
neural network
nonintegrated data
OLAP cube
OLAP report
OLAP server

Amazon Ex. 1008.0582

OnLine Analytical Processing (OLAP)
online transaction processing (OLTP) system
operational system
PivotTable
pull report
push report
query report
R score
regression analysis
report authoring
report delivery
report management
reporting system
RFM analysis
slowly changing dimension
SQL Top . . . Percent option
star schema
static report
supervised data mining
support
time dimension
transactional system
unsupervised data mining
Web portal

Review Questions

13.1 What are BI systems?

13.2 How do BI systems differ from transaction processing systems?

13.3 Name and describe the two main categories of BI systems.

13.4 What are the three sources of data for BI systems?

13.5 Explain the difference in processing between reporting and data mining applications.

13.6 Describe three reasons why direct reading of operational data is not feasible for BI applications.

13.7 Summarize the problems with operational databases that limit their usefulness for BI applications.

13.8 What are dirty data? How do dirty data arise?

13.9 Why is server time not useful for Web-based order entry BI applications?

13.10 What is click-stream data? How is it used in BI applications?

13.11 Why are data warehouses necessary?

13.12 Why do the authors describe the data in Figure 13-5 as "frightening"?

13.13 Give examples of data warehouse metadata.

13.14 Explain the difference between a data warehouse and a data mart. Use the analogy of a supply chain.

13.15 What is the enterprise data warehouse (EDW) architecture?

13.16 Describe the differences between operational databases and dimensional databases.

13.17 What is a star schema?

13.18 What is a fact table? What type of data are stored in fact tables?

13.19 What is a measure?

13.20 What is a dimension table? What type of data are stored in dimension tables?

13.21 What is a slowly changing dimension?

13.22 Why is the time dimension important in a dimensional model?

13.23 What is a conformed dimension?

13.24 State the purpose of a reporting system.

13.25 What do the letters *RFM* stand for in RFM analysis?

Amazon Ex. 1008.0583

13.26 Describe, in general terms, how to perform an RFM analysis.

13.27 Explain the characteristics of customers having the following RFM scores: {1 1 5}, {1 5 1}, {5 5 5}, {2 5 5}, {5 1 2}, {1 1 3}.

13.28 In the RFM analysis in Figures 13-20 through 13-21, what role does the CUSTOMER_RFM table serve? What role does the CUSTOMER_R table serve?

13.29 Explain the purpose of the following SQL statement from Figure 13-21:

```
INSERT INTO CUSTOMER_R (CustomerID, MostRecentOrderDate)
   (SELECT   CustomerID, MAX(TransactionDate)
    FROM     CUSTOMER_SALES
    GROUP BY CustomerID);
```

13.30 Explain the purpose and operation of the following SQL statement from Figure 13-22:

```
UPDATE    CUSTOMER_R
SET       R_Score = 1
WHERE     CustomerID IN
          (SELECT TOP 20 PERCENT CustomerID
           FROM       CUSTOMER_R
           ORDER BY   MostRecentOrderDate DESC
           GROUP BY   CustomerID);
```

13.31 Explain the purpose and operation of the following SQL statement from Figure 13-22:

```
UPDATE    CUSTOMER_R
SET       R_Score = 2
WHERE     CustomerID IN
          (SELECT TOP 25 PERCENT CustomerID
           FROM       CUSTOMER_R
           WHERE      R_Score IS NULL
           ORDER BY   MostRecentOrderDate DESC);
```

13.32 Write an SQL statement to query the CUSTOMER_RFM table and display the CustomerID values for all customers having an RFM score of {5 1 1} or {4 1 1}. Why are these customers important?

13.33 Name and describe the purpose of the major components of a reporting system.

13.34 What are the major functions of a reporting system?

13.35 Summarize the types of reports described in this chapter.

13.36 Describe the various media used to deliver reports.

13.37 Summarize the modes of reports described in this chapter.

13.38 Name three tasks of report authoring.

13.39 Describe the major tasks in report management. Explain the role of report metadata in report management.

13.40 Describe the major tasks in report delivery.

13.41 What does OLAP stand for?

13.42 What is the distinguishing characteristic of OLAP reports?

13.43 Define *measure*, *dimension*, and *cube*.

Amazon Ex. 1008.0584

13.44 Give an example, other than one in this text, of a measure, two dimensions related to your measure, and a cube.

13.45 What is drill down?

13.46 Explain two ways that the OLAP report in Figure 13-31 differs from that in Figure 13-30.

13.47 What is the purpose of an OLAP server?

13.48 Define *data mining*.

13.49 Explain the difference between unsupervised and supervised data mining. Give examples, other than one in this text, of unsupervised and supervised data mining.

13.50 For the HSD-DW cluster analysis results shown in Figure 13-36, give an explanation for the two clusters. Use the report data and data descriptions for the HSD-DW data in the text to describe which cities and products are primarily associated with each cluster. [Hint: Study the City and ProductNumber results in Figure 13-36(b).]

13.51 Name three popular data mining techniques. What is the purpose of logistic regression? What is the purpose of a neural network?

Use the data in Figure 13-37 to answer Review Questions 13.52 through 13.57.

13.52 What is the probability that someone will buy a tank?

13.53 What is the support for buying a tank and fins? What is the support for buying two tanks?

13.54 What is the confidence for fins given that a tank has been purchased?

13.55 What is the confidence for a second tank given that a tank has been purchased?

13.56 What is the lift for fins given that a tank has been purchased?

13.57 What is the lift for a second tank given that a tank has been purchased?

Project Questions

13.58 Using the code in Figure 13-22 as an example, write the procedures Calculate_F and Calculate_M that are called from the Calculate_RFM stored procedure in Figure 13-21.

For questions 13.59 through 13.61, use Oracle, SQL Server, or MySQL.

13.59 Write a stored procedure to calculate support. Use the TRANS_DATA table described on page 557 and the TwoItemBasket view shown in Figure 13-38. Place your results in a view name SupportView.

13.60 Write a stored procedure to calculate confidence. Use the TRANS_DATA table described on page 557 and the SupportView. Place your results in a view named ConfidenceView.

13.61 Write a stored procedure to compute lift. Use the TRANS_DATA table described on page 557 to compute unconditional probabilities and use the ConfidenceView for Confidence.

13.62 Based on the discussion of the Heather Sweeney Designs operational database (HSD) and dimensional database (HSD-DW) in the text, answer the following questions.

A. Using the SQL statements shown in Figure 13-12, create the HSD-DW database in a DBMS.

B. What possible transformations of data were made before HSD-DW was loaded with data? List some possible transformations, showing the original format of the HSD data and how they appear in the HSD-DW database.

Amazon Ex. 1008.0585

C. Write the complete set of SQL statements necessary to load the transformed data into the HSD-DW database.

D. Populate the HSD-DW database, using the SQL statements you wrote to answer part C.

E. Figure 13-39 shows the SQL code to create the SALES_FOR_RFM fact table shown in Figure 13-17. Using those statements, add the SALES_FOR_RFM table to your HSD-DW database.

F. What possible transformations of data are necessary to load the SALES_FOR_RFM table? List any possible transformations, showing the original format of the HSD data and how they appear in the HSD-DW database.

G. Write an SQL query similar to the one shown on page 535 that uses the total dollar amount of each day's product sales as the measure (instead of the number of products sold each day).

H. Write the SQL view equivalent of the SQL query you wrote to answer part G.

I. Create the SQL view you wrote to answer part H in your HSD-DW database.

J. Create an Excel 2007 workbook named *HSD-DW-BI-Exercises.xlsx*.

K. Using either the results of your SQL query from part H (copy the results of the query into a worksheet in the HSD-DW-BI.xlsx workbook and then format this range as a worksheet table) or your SQL view from part K (create an Excel data connection to the view), create an OLAP report similar to the OLAP report shown in Figure 13-32. (Hint: If you need help with the needed Excel actions, search in the Excel help system for more information.)

L. Heather Sweeney is interested in the effects of payment type on sales in dollars.

1. Modify the design of the HSD-DW dimensional database to include a PAYMENT_TYPE dimension table.
2. Modify the HSD-DW database to include the PAYMENT_TYPE dimension table.
3. What data will be used to load the PAYMENT_TYPE dimension table? What data will be used to load foreign key data into the PRODUCT_SALES fact table? Write the complete set of SQL statements necessary to load these data.
4. Populate the PAYMENT_TYPE and PRODUCT_SALES tables, using the SQL statements you wrote to answer part 3.

Figure 13-39

The HSD-DW SALES_FOR_RFM SQL Statements

```
CREATE TABLE SALES_FOR_RFM(
      TimeID             Int               NOT NULL,
      CustomerID         Int               NOT NULL,
      InvoiceNumber      Int               NOT NULL,
      PreTaxTotalSale    Numeric(9,2)      NOT NULL,
      CONSTRAINT         SALES_FOR_RFM_PK
                         PRIMARY KEY (TimeID, CustomerID, InvoiceNumber),
      CONSTRAINT         SRFM_TIMELINE_FK FOREIGN KEY(TimeID)
                            REFERENCES TIMELINE(TimeID)
                               ON UPDATE NO ACTION
                               ON DELETE NO ACTION,
      CONSTRAINT         SRFM_CUSTOMER_FK FOREIGN KEY(CustomerID)
                            REFERENCES CUSTOMER(CustomerID)
                               ON UPDATE NO ACTION
                               ON DELETE NO ACTION

      );
```

Amazon Ex. 1008.0586

5. Create the SQL queries or SQL views needed to incorporate the PaymentType attribute.
6. Create an Excel 2007 OLAP report to show the effect of payment type on product sales in dollars.

13.63 The following questions require that you have completed question 13.61 and that you have a version of SQL Server 2005 or SQL Server 2008 that includes SQL Server Analysis Services.

A. Download and install the correct version of the Microsoft SQL Server Data Mining add-ins for Microsoft Office 2007 for your version of SQL Server.

B. Use the data mining add-in in Excel 2007 to do a cluster analysis of the HSD-DW data results from your SQL query results from question 13.61(I) or SQL view from questions 13.62(J) and 13.62(K). Use only the City and ProductNumber attributes in your analysis. Interpret you results.



Assume that Marcia uses a database that includes the following three tables:

CUSTOMER (**CustomerID**, **FirstName**, **LastName**, **Phone**, **Email**)

ORDER (**InvoiceNumber**, **CustomerID**, **DateIn**, **DateOut**, **Subtotal**, **Tax**, **TotalAmount**)

ORDER_ITEM (*InvoiceNumber*, **ItemNumber**, *Service*, **Quantity**, **UnitPrice**, **ExtendedPrice**)

(ORDER_ITEM.Service is a foreign key to the SERVICE table, but the SERVICE table is not needed for these exercises.)

A. Describe how an RFM analysis could be useful in Marcia's business.

B. Using the four tables in Figure 13-20, write a set of stored procedures to compute an RFM analysis on Marcia's data.

C. Show SQL to process the table generated in your answer to B to display the names and e-mail data for all customers having an RFM score of {5 1 1} or {4 1 1}.

D. Describe in general terms how a market basket analysis can be used on the items in a dry cleaning order.

E. Using the instructions in questions 13.58 through 13.60, write stored procedures to perform a market basket analysis on the items in a dry cleaning order.

Amazon Ex. 1008.0587



The tables we have used for Morgan Importing have no natural application for either RFM or market basket analysis, at least not to Morgan. However, consider the following three tables from the standpoint of a shipper:

SHIPMENT (ShipmentID, ShipDate, *ShipperName*, ShipperInvoiceNumber, Origin, Destination)

SHIPMENT_ITEM (*ShipmentID*, *PurchaseID*, InsuredValue)

SHIPPER (ShipperName, Phone, Fax, Email, Contact)

If we substitute CUSTOMER for SHIPPER, we can create the structure of a database that would record customers and shipments and items they've shipped with us. The modified tables are:

SHIPMENT (ShipperInvoiceNumber, ShipDate, *CustomerID*, Origin, Destination, Subtotal, Tax, Total)

SHIPMENT_ITEM (*ShipperInvoiceNumber*, ItemNumber, ShippingCost)

CUSTOMER (CustomerID, CustomerName, Phone, Fax, Email, Contact)

where CustomerID is a surrogate key for the customer data. Use these revised tables to answer the following questions.

A. Describe how an RFM analysis could be useful to the shipper.

B. Using the four tables in Figure 13-20, write a set of stored procedures to compute an RFM analysis for shipments.

C. Show SQL to process the table generated in your answer to B to display the names and e-mail data for all customers having an RFM score of {5 1 1} or {4 1 1}.

D. Describe in general terms how a market basket analysis can be used on the items in a shipment.

E. Using the instructions in questions 13.58 through 13.60, write stored procedures to perform a market basket analysis on the items in a shipment.

Amazon Ex. 1008.0588

Online Appendices

Complete versions of these appendices are available on the textbook's Web site:
www.pearsonhighered.com/kroenke

Appendix A

Getting Started with Microsoft Access 2007

Appendix B

The IDEF1X Standard

Appendix C

UML-Style Entity-Relationship Diagrams

Appendix D

Data Structures for Database Processing

Appendix E

The Semantic Object Model

Amazon Ex. 1008.0589

Bibliography

Web Links

News

CNET News.com: *www.news.com*
Advisor: *www.advisor.com*
Intelligent Enterprise: *www.intelligententerprise.com*
ZDNet: *www.zdnet.com*

Data Mining

KDnuggets: *www.kdnuggets.com*
Google Dating Mining Directory: *directory.google.com/Top/Computers/Software/Databases/Data_Warehousing/Decision_Support_Tools*
Insightful Miner 3: *www.insightful.com/products/iminer*
SAS Enterprise Miner: *www.sas.com/technologies/analytics/datamining/miner*
SPSS Clementine: *www.spss.com/Clementine*
Microsoft SQL Server 2008 Data Mining Add-Ins for Office 2007: *www.microsoft.com/sqlserver/2008/en/us/data-mining-addins.aspx*

DBMS and Other Vendors

Oracle Database 11g: *www.oracle.com/database/index.html*
SQL Server 2008: *www.microsoft.com/sqlserver/2008/en/us/default.aspx*
MySQL: *www.mysql.com*
Eclipse IDE: *www.eclipse.org*
PHP: *http://us.php.net*
NetBeans: *www.netbeans.org/index.html*
Microsoft Express Editions: *www.microsoft.com/Express*

Standards

JDBC: *http://java.sun.com/products/jdbc* and *http://en.wikipedia.org/wiki/JDBC*
ODBC: *www.liv.ac.uk/middleware/html/odbc.html* and *http://en.wikipedia.org/wiki/Open_Database_Connectivity*
Worldwide Web Consortium (W3C): *www.w3.org*
XML: *www.w3.org/XML*, *www.xml.org*, and *http://en.wikipedia.org/wiki/XML*

Classic Articles and References

ANSI X3. *American National Standard for Information Systems—Database Language SQL.* ANSI, 1992.

Bruce, T. *Designing Quality Databases with IDEF1X Information Models.* New York: Dorset House, 1992.

Chamberlin, D. D., et al. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control." *IBM Journal of Research and Development* 20 (November 1976).

Chen, P. "The Entity-Relationship Model: Toward a Unified Model of Data." *ACM Transactions on Database Systems* 1 (March 1976).

Amazon Ex. 1008.0590

Chen, P. *Entity-Relationship Approach to Information Modeling.* E-R Institute, 1981.
Coar, K. A. L. *Apache Server for Dummies.* Foster City, CA: IDG Books, 1997.
Codd, E. F. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM* 25 (February 1970).
Codd, E. F. "Extending the Relational Model to Capture More Meaning." *Transactions on Database Systems* 4 (December 1979).
Date, C. J. *An Introduction to Database Systems,* 8th ed. Upper Saddle River, NJ: Pearson Education, 2003.
Embley, D. W. "NFQL: The Natural Forms Query Language." *ACM Transactions on Database Systems* 14 (June 1989).
Eswaran, K. P., J. N. Gray, R. A. Lorie, and I. L. Traiger. "The Notion of Consistency and Predicate Locks in a Database System." *Communications of the ACM* 19 (November 1976).
Fagin, R. "A Normal Form for Relational Databases That Is Based on Domains and Keys." *Transactions on Database Systems* 6 (September 1981).
Fagin, R. "Multivalued Dependencies and a New Normal Form for Relational Databases." *Transactions on Database Systems* 2 (September 1977).
Hammer, M., and D. McLeod. "Database Description with SDM: A Semantic Database Model." *Transactions on Database Systems* 6 (September 1981).
Keuffel, W. "Battle of the Modeling Techniques." *DBMS Magazine* (August 1996).
Kroenke, D. "Waxing Semantic: An Interview." *DBMS Magazine* (September 1994).
Moriarty, T. "Business Rule Analysis." *Database Programming and Design* (April 1993).
Muller, R. J. *Database Design for Smarties: Using UML for Data Modeling.* San Francisco: Morgan Kaufmann, 1999.
Nijssen, G., and T. Halpin. *Conceptual Schema and Relational Database Design: A Fact-Oriented Approach.* Upper Saddle River, NJ: Prentice Hall, 1989.
Nolan, R. *Managing the Data Resource Function.* St. Paul: West Publishing, 1974.
Ratliff, C. Wayne, "dStory: How I Really Developed dBASE." *Data Based Advisor* (March 1991).
Rogers, D. "Manage Data with Modeling Tools." *VB Tech Journal* (December 1996).
Ross, R. *Principles of the Business Rule Approach.* Boston: Addison-Wesley, 2003.
Zloof, M. M. "Query by Example." *Proceedings of the National Computer Conference, AFIPS* 44 (May 1975).

Useful Books

Berry, M., and G. Linoff. *Data Mining Techniques for Marketing, Sales, and Customer Support.* New York: Wiley, 1997.
Celko, J. *SQL for Smarties,* 2d ed. San Francisco: Morgan Kaufmann, 2000.
Celko, J. *SQL Puzzles and Answers.* San Francisco: Morgan Kaufmann, 1997.
Fields, D. K., and M. A. Kolb. *Web Development with Java Server Pages.* Greenwich, CT: Manning Press, 2000.
Harold, E. R. *XML: Extensible Markup Language.* New York: IDG Books Worldwide, 1998.
Kay, M. *XSLT: Programmer's Reference.* Birmingham, UK: WROX Press, 2000.
Loney, K. *Oracle Database 11g: The Complete Reference.* Berkeley, CA: Osborne/McGraw-Hill, 2008.
Muench, S. *Building Oracle XML Applications.* Sebastopol, CA: O'Reilly, 2000.
Muller, R. J. *Database Design for Smarties: Using UML for Data Modeling.* San Francisco: Morgan Kaufmann, 1999.
Mundy, J., W. Thornthwaite, and R. Kimball, *The Microsoft Data Warehouse Toolkit.* Indianapolis, IN: Wiley, 2006.
Pyle, D. *Data Preparation for Data Mining.* San Francisco: Morgan Kaufmann, 1999.

Amazon Ex. 1008.0591

Glossary

While this section defines many of the key terms in the book, it is not meant to be exhaustive. Terms related to a specific DBMS product, for example, should be referenced in the chapter dedicated to that product. These references can be found in the index. Similarly, SQL concepts are included, but details of SQL commands and syntax should be referenced in the chapter that discusses those details.

.NET Framework. Microsoft's comprehensive application development platform. It includes such components as ADO.NET and ASP.NET.

/* and */. The symbols used to indicate a comment line in an SQL script in SQL Server 2008, Oracle Database 11g, and MySQL 5.1.

<?php and ?>. The symbols used to indicate blocks of PHP code in Web pages.

Abstraction. A generalization of something that hides some unimportant details but enables work with a wider class of types. A recordset is an abstraction of a relation. A rowset is an abstraction of a recordset.

ACID transaction. ACID stands for "*a*tomic, *c*onsistent, *i*solated, and *d*urable." An *atomic* transaction is one in which all of the database changes are committed as a unit; either all are done, or none are. A *consistent* transaction is one in which all actions are taken against rows in the same logical state. An *isolated* transaction is one that is protected from changes by other users. A *durable* transaction is one that is permanent after it is committed to the database, regardless of subsequent failures. There are different levels of consistency and isolation. *See also* transaction-level consistency, statement-level consistency, and transaction isolation level.

Active Data Objects (ADO). An implementation of OLE DB that is accessible via object- and non-object-oriented languages. It is used primarily as a scripting-language (JScript, VBScript) interface to OLE DB.

Active repository. Parts of the systems development processes where metadata is created automatically as the system components are created. *See* data repository.

Active Server Pages (ASP). A file containing markup language, server script, and client script that is processed by the Active Server Processor in Microsoft Internet Information Server (IIS).

Ad-hoc query A query created by a user as and when needed, as compared to a predefined and stored query.

ADO.NET. A data access technology that is part of Microsoft's .NET initiative. ADO.NET provides the capabilities of ADO, but with a different object structure. ADO.NET also includes new capabilities for the processing of datasets. *See also* ADO.NET DataSet.

ADO.NET Command object. The ADO.NET object that mimics an SQL statement or stored procedure. It is run against the data in the DataSet.

ADO.NET Connection object. The ADO.NET object responsible for connecting to a data source.

ADO.NET Data Provider. A class library that provides ADO.NET services. There are Data Providers for ODBC, OLE.DB, SQL Server, and EDM applications.

ADO.NET DataAdapter object. The ADO.NET object that is the connector between a Connection object and a DataSet object. It uses four command objects: SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand.

ADO.NET DataReader. An ADO.NET object that is similar to a read-only, forward-only cursor, and which can only be used by an ADO.NET Command object 's Execute method.

ADO.NET DataSet. A representation of data from a database that is stored in computer memory for immediate use. It is distinct and disconnected from the data in the database.

Amazon Ex. 1008.0592

ADO.NET Entity Framework. An extension to ADO.NET that supports the Microsoft EDM. *See* Entity Data Model (EDM).

After image. A record of a database entity (normally a row or a page) after a change. Used in recovery to perform rollforwards.

Alert. In reporting systems, a type of report that is triggered by an event.

Alternate key. In entity-relationship models, a synonym for candidate key.

American National Standards Institute (ANSI). The American standards organization that creates and publishes the SQL standards. *See* Structured Query Language (SQL).

AMP. An abbreviation for Apache, MySQL, and PHP/Pearl/Python. *See* Apache Web Server, PHP.

Anomaly. An undesirable consequence of a data modification. The term is used in normalization discussions. With an insertion anomaly, facts about two or more different themes must be added to a single row of a relation. With a deletion anomaly, facts about two or more themes are lost when a single row is deleted.

API. *See* application program interface (API).

Apache Tomcat. An application server that works in conjunction with the Apache Web server. *See* Apache Web server.

Apache Web server. A popular Web server that runs on most operating systems, particularly Windows and Linux.

Applet. A compiled, machine-independent Java bytecode program that is run by the Java virtual machine embedded in a browser.

Application. A business computer system that processes a portion of a database to meet a user's information needs. It consists of menus, forms, reports, queries, Web pages, and application programs.

Application program. A custom-developed program for processing a database. It can be written in a standard procedural language, such as Java, C#, Visual Basic .NET, or C++, or in a language unique to the DBMS, such as PL/SQL or T-SQL.

Application program interface (API). A set of program procedures or functions that can be called to invoke a set of services. The API includes the names of the procedures and functions and a description of the name, purpose, and data type of parameters to be provided. For example, a DBMS product can provide a library of functions to call for database services. The names of procedures and their parameters constitute the API for that library.

Archetype/version object. A two-object structure that represents multiple versions of a standardized item; for example, a SOFTWARE-PRODUCT (the archetype) and PRODUCT-RELEASE (the version of the archetype). The identifier of the version always includes the identifier of the archetype object.

ASP. *See* Active Server Pages (ASP).

ASP.NET. The updated version of ASP for the .NET Framework. *See* Active Server Pages (ASP), .NET Framework.

Association object. An object that represents the combination of at least two other objects and that contains data about that combination. It is often used in contracting and assignment applications.

Association pattern. In database design, a table pattern where an intersection table contains additional attributes beyond the attributes that make up the composite primary key.

Asterisk (*) wildcard character. A character used in Access 2007 queries to represent one or more unspecified characters. *See* SQL percent sign (%) wildcard character.

Atomic. A set of actions that is completed as a unit. Either all of the actions are completed, or none of them are.

Atomic transaction. A group of logically related database operations that is performed as a unit. Either all of the operations are performed, or none of them are.

Attribute. (1) A column of a relation; also called a *column*, *field*, or *data item*. (2) A property in an entity.

Authorization rules. A set of processing permissions that describes which users or user groups can take particular actions against particular portions of the database.

AUTO_INCREMENT attribute. In MySQL, the data attribute used to create surrogate keys.

AutoNumber. In Access 2007, the data type used to create surrogate keys.

AVG. In SQL, a function that averages up a set of numbers. *See* SQL built-in functions.

Amazon Ex. 1008.0593

Base Class Library. A component of the Microsoft .NET Framework that provides support for the programming languages used with the .NET Framework.

Base domain. In IDEF1X, a domain definition that stands alone. Other domains may be defined as subsets of a base domain.

Before image. A record of a database entity (normally a row or a page) before a change. Used in recovery to perform rollback.

BI. *See* business intelligence (BI) systems.

Binary relationship. A relationship between exactly two entities or tables.

Boyce-Codd normal form (BCNF). A relation in which every determinant is a candidate key.

Branch. A subelement of a tree that may consist of one or many nodes.

Buffer. An area of memory used to hold data. For a read, data are read from a storage device into a buffer; for a write, data are written from the buffer to storage.

Business intelligence (BI) systems. Information systems that assist managers and other professionals in the analysis of current and past activities and in the prediction of future events. Two major categories of BI systems are reporting systems and data mining systems.

Bytecode interpreter. For an applications written in Java, the program used by a specific operating system to execute the application. Bytecode interpreters are known as Java virtual machines. *See* Java virtual machine.

Candidate key. An attribute or group of attributes that identifies a unique row in a relation. One of the candidate keys is chosen to be the primary key.

Cardinality. In a binary relationship, the maximum or minimum number of elements allowed on each side of the relationship. The maximum cardinality can be 1:1, 1:N, N:1, or N:M. The minimum cardinality may be optional-optional, optional-mandatory, mandatory-optional, or mandatory-mandatory.

Cascading deletion. A referential integrity action specifying that when a parent row is deleted, related child rows should be deleted as well.

Cascading update. A referential integrity action specifying that when the key of a parent row is updated, the foreign keys of matching child rows should be updated as well.

Casual relationship. A relationship that is created without a foreign key constraint. This is useful if the tables are missing data values.

Categorization cluster. In IDEF1X, a group of mutually exclusive category entities. *See also* complete category cluster.

Category entity. In IDEF1X, a subtype that belongs to a category cluster.

CHECK constraint. In SQL, a constraint that specifies what data values are allowed in a particular column.

Checkpoint. The point of synchronization between a database and a transaction log. All buffers are force-written to external storage. The term is sometimes used in other ways by DBMS vendors.

Child. An entity or row on the many side of a one-to-many relationship.

Class attributes. In the uniform modeling language (UML), attributes that pertain to the class of all entities of a given type.

Click-stream data. Data about a customer's clicking behavior on a Web page; such data are often analyzed by e-commerce companies.

Cluster analysis. A form of unsupervised data mining in which statistical techniques identify groups of entities that have similar characteristics.

Collection. An object that contains a group of other objects. Examples are the ADO Names, Errors, and Parameters collections.

Column. A logical group of bytes in a row of a relation or a table. The meaning of a column is the same for every row of the relation.

COM. *See* Component Object Model (COM).

Command line utility. A character user interface program that presents a command prompt to the user. The user then types a command and presses the Enter key for execution. Each major DBMS product has a command line utility.

Commit. A command issued to the DBMS that makes database modifications permanent. After the command has been processed, database changes are written to the database and to a log so that they will survive system crashes and other failures. A commit is usually used at the end of an atomic transaction. Contrast this with rollback.

Amazon Ex. 1008.0594

Common Language Runtime (CLT). A component of the Microsoft .NET Framework that provides support for the programming languages used with the .NET Framework.

Complete category cluster. A category cluster in which all possible category entities are defined. The generic entity must also be one of the category entities.

Component Object Model (COM). A Microsoft specification for the development of object-oriented programs.

Composite determinant. In functional dependencies, a determinant consisting of two or more attributes.

Composite identifier. In data modeling, an identifier consisting of two or more attributes.

Composite key. In database design, a key with two or more attributes.

Composite primary key. In database design and actual databases, a primary key with two or more attributes.

Computed value. A column of a table that is computed from other column values. Values are not stored, but are computed when they are to be displayed.

Concurrency. A condition in which two or more transactions are processed against the database at the same time. In a single CPU system, the changes are interleaved; in a multi-CPU system, the transactions may be processed simultaneously, and the changes on the database server are interleaved.

Concurrent processing. The sharing of the CPU among several transactions. The CPU is allocated to each transaction in a round robin or in some other fashion for a certain period of time. Operations are performed so quickly that they appear to users to be simultaneous. In local area networks (LANs) and other distributed applications, concurrent processing is used to refer to the (possibly simultaneous) processing of applications on multiple computers.

Concurrent transactions. Two transactions that are being processed at the same time.

Concurrent update problem. An error condition in which one user's data changes are overwritten by another user's data changes. Same as lost update problem.

Confidence. In market basket analysis, the probability of a customer's buying one product, given that the customer has purchased another product.

Conformed dimension. In a dimensional database design, a dimension table that has relationships to two or more fact tables.

Connection relationship. In IDEF1X, a HAS-A relationship.

Consistency. Two or more concurrent transactions are consistent if the result of their processing is the same as it would have been if they had been processed in some serial order.

Consistent. In an ACID transaction, either statement-level or transaction-level consistency. *See* ACID transaction, consistency, statement-level consistency, and transaction-level consistency.

Consistent backup. A backup file from which all uncommitted changes have been removed.

Correlated subquery. A type of subquery in which an element in the subquery refers to an element in the containing query. A subquery that requires nested processing.

Crow's foot model. Formally known as the Information Engineering (IE) Crow's Foot model, it is a system of symbology used to construct E-R diagrams in data modeling and database design.

Crow's foot symbol. A symbol in the IE Crow's Foot E-R model that indicates a many side of the relationship. It visually resembles a bird's foot, and thus the name *crow's foot*.

Control-of-flow statements. Procedural program statements that direct execution of the program depending upon an existing condition. Control-of-flow statements include, for example, IF..THEN..ELSE logic and DO WHILE logic.

COUNT. In SQL, a function that counts the number of rows in a query result. *See* SQL built-in functions.

Curse of dimensionality. In data mining applications, the phenomenon that the more attributes there are, the easier it is to build a model that fits the sample data but that is worthless as a predictor.

Cursor. An indicator of the current position in a pseudofile for an SQL SELECT that has been embedded in a program; it shows the identity of the current row.

Cursor type. A declaration on a cursor that determines how the DBMS places implicit locks. Four types of cursor discussed in this text are forward only, snapshot, keyset, and dynamic.

Amazon Ex. 1008.0595

Data. The values stored in database tables.

Data administration. The enterprise-wide function that concerns the effective use and control of the organization's data assets. Data administration may be handled by an individual, but it is usually handled by a group. Specific functions include setting data standards and policies and providing a forum for conflict resolution. *See also* database administrator (DBA).

Data consumer. A user of OLE DB functionality.

Data definition language (DDL). A language used to describe the structure of a database. SQL DDL is that portion of SQL that is used to create, modify, and drop database structures.

Data dictionary. A user-accessible catalog of database and application metadata. The contents of an *active* data dictionary are automatically updated by the DBMS whenever changes are made in the database or application structure. The contents of a *passive* data dictionary must be updated manually when changes are made.

Data integrity. The state of a database in which all constraints are fulfilled. Usually refers to interrelation constraints in which the value of a foreign key is required to be present in the table having that foreign key as its primary key.

Data integrity problems. A table that has inconsistencies that create insert, update, or deletion anomalies is said to have *data integrity problems.*

Data manipulation language (DML). A language used to describe the processing of a database. SQL DML is that portion of SQL that is used to query, insert, update, and modify data.

Data mart. A facility similar to a data warehouse, but with a restricted domain. Often, the data are restricted to particular types, business functions, or business units.

Data mining application. Business intelligence systems that use sophisticated statistical and mathematical techniques to perform what-if analyses, to make predictions, and to facilitate decisions. Contrast with reporting systems.

Data model. A model of the users' data requirements usually expressed in terms of the entity-relationship model.

Data provider. A provider of OLE DB functionality. Examples are tabular data providers and service data providers.

Data repository. Collections of metadata about databases, database applications, Web pages, users, and other application components.

Data sublanguage. A language for defining and processing a database to be embedded in programs written in another language, in most cases a procedural language such as Java, C#, Visual Basic, or C++. A data sublanguage is an incomplete programming language because it contains only constructs for data access.

Data warehouse. A store of enterprise data that is designed to facilitate management decision making. A data warehouse includes not only data, but also metadata, tools, procedures, training, personnel information, and other resources that make access to the data easier and more relevant to decision makers.

Data warehouse metadata. In a data warehouse, metadata concerning the data, its source, its format, its assumptions and constraints, and other facts about the data.

Data warehouse metadata database. The database used to store the data warehouse metadata.

Database. A self-describing collection of integrated records.

Database administration. The function that concerns the effective use and control of a particular database and its related applications.

Database administrator (DBA). The person or group responsible for establishing policies and procedures to control and protect a database. The database administrator works within guidelines set by data administration to control the database structure, manage data changes, and maintain DBMS programs.

Database application. An application that uses a database to store the data needed by the application.

Database data. The portion of a database that contains data of interest and use to the application end users. *See* data.

Database design. A diagram that represents that database as it will be implemented in a DBMS product.

Database management system (DBMS). A set of programs used to define, administer, and process the database and its applications.

Amazon Ex. 1008.0596

Database redesign. The process of changing the structure of a database to adapt the database to changing requirements or to fix it so that it has the structure it should have had in the first place.

Database save. A copy of database files that can be used to restore the database to some previous consistent state.

Database schema. In MySQL, the functional equivalent of a database in Microsoft Access or SQL Server.

Dataset. In ADO.NET, an in-memory collection of tables that is not connected to any database. Datasets have relationships, referential integrity constraints, referential integrity actions, and other important database characteristics. They are processed by ADO.NET objects. A single dataset may be materialized as tables, as an XML document, or as an XML Schema.

DBA. *See* database administrator (DBA).

DBMS. *See* database management system (DBMS).

DBMS reserved word. A word that has a special meaning in the DBMS and should not be used as a table, column, or other name in a database.

DDL. *See* data definition language (DDL).

Deadlock. A condition that can occur during concurrent processing in which each of two (or more) transactions is waiting to access data that the other transaction has locked. Also called a deadly embrace.

Deadly embrace. *See* deadlock.

Decision support system (DSS). One or more applications designed to help managers make decisions. An earlier name for business intelligence (BI).

Decision tree analysis. A form of unsupervised data mining that classifies entities of interest into two or more groups according to values of attributes that measure the entities' past history.

DEFAULT keyword. In SQL, the work used to specify a default value for an attribute.

Default value. A value assigned to an attribute if there is no other value assigned to it when a new row is created in a table.

Default namespace. In an XML Schema document, the namespace that is used for all unlabeled elements.

Degree. For relationships in the entity-relationship model, the number of entities participating in the relationship. In almost all cases, such relationships are of degree two.

Deletion anomaly. In a relation, the situation in which the removal of one row of a table deletes facts about two or more themes.

Delimited identifier. A reserved word placed in special symbols to distinguish it from the DBMS reserved word so that it can be used as a table, column, or other name in a database.

Denormalize. To intentionally create a set of database tables that are not normalized to BCNF and 4NF.

Dependency graph. A network of nodes and lines that represents the logical dependencies among tables, views, triggers, stored procedures, indexes, and other database constructs.

Determinant. One or more attributes that functionally determine another attribute or attributes. In the functional dependency (A, B) → C, the attributes (A, B) are the determinant.

Differential backup. A backup file that contains only changes made since a prior backup.

Digital dashboard. In reporting systems, a display that is customized for a particular user. Typically, a digital dashboard has links to many different reports.

Dimension table. In a star schema dimensional database, the tables that connect to the central fact table. Dimension tables hold attributes used in the organizing queries in analyses such as those of OLAP cubes.

Dimensional database. A database design that is used for data warehouses and is designed for efficient queries and analysis. It contains a central fact table connected to one or more dimension tables.

Dirty data. In a business intelligence system, data with errors. Examples are a value of "G" for customer sex and a value of "213" for customer age. Other examples are a value of "999-999-9999" for a U.S. phone number, a part color of "gren," and an e-mail address of "WhyMe@somewhereelseintheuniverse.who." Dirty data pose problems for reporting and data mining applications.

Amazon Ex. 1008.0597

Dirty read. Reading data that has been changed but not yet committed to the database. Such changes may later be rolled back and removed from the database.

Discriminator. In the entity-relationship model, an attribute of a supertype entity that determines which subtype pertains to the supertype.

DK/NF. *See* domain/key normal form.

DML. *See* data manipulation language (DML).

Document Object Model (DOM). An API that represents an XML document as a tree. Each node of the tree represents a piece of the XML document. A program can directly access and manipulate a node of the DOM representation.

Document type declaration (DTD). A set of markup elements that defines the structure of an XML document.

DOM. *See* Document Object Model.

Domain. A named set of all possible values that an attribute can have. Domains can be defined by listing allowed values or by defining a rule for determining allowed values.

Domain/key normal form (DK/NF). A relation in which all constraints are logical consequences of domains and keys.

Drill down. User-directed disaggregation of data used to break higher-level totals into components.

DTD. *See* document type declaration.

Durable. In an ACID transaction, the database changes are permanent. *See* ACID transaction.

Dynamic cursor. A fully featured cursor. All inserts, updates, deletions, and changes in row order are visible to a dynamic cursor.

Dynamic report. In reporting systems, a report that reads the most current data at the time of the report's creation. Contrast with static report.

Eclipse IDE. A popular open-source integrated development environment.

Eclipse PDT (PHP Development Tools) Project. A version of the Eclipse IDE customized for use with PHP. *See* Eclipse IDE, PHP.

Enterprise-class database system. A DBMS product capable of supporting the operating requirement of large organizations.

Enterprise data warehouse (EDW) architecture. A data warehouse architecture that links specialized data marts to a central data warehouse for data consistency and efficient operations.

Entity. (1) In the entity-relationship model, a representation of something that users want to track. *See also* entity class and entity instance. (2) In a generic sense, something that users want to track. In the relational model, an entity is stored in one row of a table.

Entity class. In the entity-relationship model, a collection of entities of a given type; for example, EMPLOYEE and DEPARTMENT. The class is described by its attributes.

Entity Data Model (EDM). An emerging Microsoft data modeling technology that is part of the .NET Framework.

Entity instance. A particular occurrence of an entity; for example, Employee 100 and the Accounting Department. An entity instance is described by values of its attributes.

Entity-relationship (E-R) data modeling. Creating a data model using E-R diagrams. *See* entity-relationship (E-R) diagram

Entity-relationship (E-R) diagram. A graphic used to represent entities and their relationships. In the traditional E-R model, entities are shown as squares or rectangles, and relationships are shown as diamonds. The cardinality of the relationship is shown inside the diamond. In the crow's foot model, entities are shown in rectangles, and relationships are shown by lines between the rectangles. Attributes are generally listed within the rectangle. The many side of many relationships is represented by a crow's foot.

Entity-relationship (E-R) model. A set of constructs and conventions used to create data models. The things in the users' world are represented by entities, and the associations among those things are represented by relationships. The results are usually documented in an entity-relationship (E-R) diagram.

Enumerated list. A list of allowed values for a domain, attribute, or column.

Equijoin. The process of joining relation A containing attribute A1 with B containing attribute B1 to form relation C, so that for each row in C, A1 = B1. Both A1 and B1 are represented in C.

Amazon Ex. 1008.0598

E-R diagram. *See* entity-relationship diagram.

Exclusive lock. A lock on a data resource such that no other transaction can either read or update that resource.

Existence-dependent entity. Same as a weak entity. An entity that cannot appear in the database unless an instance of one or more other entities also appears in the database. A subclass of existence-dependent entities is ID-dependent entities.

Explicit lock. A lock requested by command from an application program.

Extensible Markup Language. *See* XML.

Extensible Style Language. *See* XSLT.

Extract. A portion of an operational database downloaded to a local area network (LAN) or personal computer for local processing. Extracts are created to reduce communication cost and time when querying and creating reports from data created by transaction processing.

Extract, Transform, and Load (ETL) system. The portion of a data warehouse that converts operation data to data warehouse data.

F score. In RFM analysis, the "how frequently" score, which reflects how often a customer makes a purchase. *See* RFM Analysis.

Fact table. In a dimensional database, the central table that contains numerical values.

Field. (1) A logical group of bytes in a record such as Name or PhoneNumber. (2) In the relational model, a synonym for attribute.

Fifth normal form (5NF). A normal form necessary to eliminate an anomaly where a table can be split apart but not correctly joined back together. Also know as Project-Join Normal Form (PJ/NF)

File data source. An ODBC data source stored in a file that can be e-mailed or otherwise distributed among users.

First normal form (1NF). Any table that fits the definition of a relation.

Flat file. A file that has only a single value in each field. The meaning of the columns is the same in every row.

Foreign key. An attribute that is a key of one or more relations other than the one in which it appears. Used to represent relationships.

FOREIGN KEY constraint. In SQL, the constraint used to create relationships and referential integrity between tables.

Fourth normal form (4NF). A relation in Boyce-Codd normal form in which there are no multivalued dependencies or in which all attributes participate in a single multivalued dependency.

Functional dependency. A relationship between attributes in which one attribute or group of attributes determines the value of another. The expression $X \rightarrow Y$ means that given a value of X, we can determine the value of Y. A given value of X may appear in a relation more than once, but if so, it is always paired with the same value of Y. Also, if $X \rightarrow (Y, Z)$, then $X \rightarrow Y$ and $X \rightarrow Z$. However, if $(X, Y) \rightarrow Z$, then, in general X Not $\rightarrow$ Z and Y Not $\rightarrow$ Z.

Generic entity. In IDEF1X, an entity that has one or more category clusters. The generic entity takes the role of a supertype for the category entities in the category cluster.

Graphical User Interface (GUI). A user interface that uses graphical elements for interaction with a user.

Granularity. The size of the database resource that is locked. Locking the entire database is large granularity; locking a column of a particular row is small granularity.

Growing phase. The first stage in two-phase locking in which locks are acquired but not released.

HAS-A relationship. A relationship between two entities or objects that are of different logical types; for example, EMPLOYEE HAS-A(n) AUTO. Contrast this with an IS-A relationship.

HTML. *See* Hypertext Markup Language.

HTML document tags. The tags in HTML documents that indicate the structure of the document.

HTML syntax rules. The standards that are used to create HTML documents.

HTTP. *See* Hypertext Transfer Protocol.

Http://localhost. For a Web server, a reference to the user's computer.

Amazon Ex. 1008.0599

Hypertext Markup Language (HTML). A standardized set of text tags for formatting text, locating images and other nontext files, and placing links or references to other documents.

Hypertext Transfer Protocol (HTTP). A standardized means for using TCP/IP to communicate over the Internet.

ID-dependent entity. An entity whose identifier contains the identifier of a second entity. For example, APPOINTMENT is ID-dependent on CLIENT, where the identifier of APPOINTMENT is (Date, Time, ClientNumber) and the identifier of CLIENT is ClientNumber. An ID-dependent entity is weak, meaning that it cannot logically exist without the existence of that second entity. Not all weak entities are ID-dependent, however.

IDEF1X (Integrated Definition 1, Extended). A version of the entity-relationship model, adopted as a national standard, but difficult to understand and use. Most organizations use a simpler E-R version like the crow's foot model.

Identifier. An attribute that names, or identifies, and entity.

Identifying connection relationship. In IDEF1X, a 1:1 or 1:N HAS-A relationship in which the child entity is ID-dependent on the parent.

Identifying relationship. A relationship that is used when the child entity is ID-dependent upon the parent entity.

IDENTITY ({StartValue}, {Increment}) property [MSSQL]. For SQL Server 2008, the attribute that is used to create a surrogate key.

IIS. *See* Internet Information Server.

Implementation. In object-oriented programming, a set of objects that instantiates a particular object-oriented interface.

Implicit lock. A lock that is automatically placed by the DBMS.

Import. A function of the DBMS; to read a file of data in bulk.

Inclusive subtype. In data modeling and database design, a subtype that allows a supertype entity to be associated with more than one subtype.

Inconsistent backup. A backup file that contains uncommitted changes.

Inconsistent read problem. In a transaction, a series of reads of a set of rows in which some of the rows have been updated by a second transaction and some of the rows have not been updated by that second transaction. Can be prevented by two-phase locking and other strategies.

Index. Data created by the DBMS to improve access and sorting performance. Indexes can be constructed for a single column or groups of columns. They are especially useful for columns used by WHERE clauses, for conditions in joins, and for sorting.

Index.html. A default Web page name provided by most Web servers.

Inetpub folder. In Windows operating systems, the root folder for the IIS Web server.

Information. (1) Knowledge derived from data, (2) data presented in a meaningful context, or (3) data processed by summing, ordering, averaging, grouping, comparing, or other similar operations.

Information Engineering (IE) model. An E-R model developed by James Martin.

Inner join. Synonym for join. Contrast with outer join.

Insertion anomaly. In a relation, the condition that exists when, to add a complete row to a table, one must add facts about two or more logically different themes.

Instance failure. A failure in the operating system or hardware that causes the DBMS to fail.

Integrated Development Environment (IDE). An application that provideds a programmer or application developer with a complete set of development tools in one package.

Integrated tables. Database tables that store both data and the relationships among the data.

Interface. (1) The means by which two or more programs call each other; the definition of the procedural calls between two or more programs. (2) In object-oriented programming, the design of a set of objects that includes the objects' names, methods, and attributes.

International Organization for Standardization (ISO). The international standards organization that works of SQL standards, among others.

Internet Information Server (IIS). A Microsoft product that operates as an HTTP server.

Internet Information Services Manger. The application used to manage Microsoft's IIS Web server.

Amazon Ex. 1008.0600

Intersection table. A table (relation) used to represent a many-to-many relationship. It contains the keys of the tables (relations) in the relationship. The relationships from the parent tables to the intersection tables must have a minimum cardinality of either mandatory-optional or mandatory-mandatory.

Interrelation constraint. A data constraint between two tables.

Intrarelation constraint. A data constraint within one table.

IS-A relationship. A relationship between a supertype and a subtype. For example, EMPLOYEE and ENGINEER have an IS-A relationship.

Isolation level. *See* transaction isolation level.

Java. An object-oriented programming language that has better memory management and bounds checking than C++. It is used primarily for Internet applications, but it also can be used as a general-purpose programming language. Java compilers generate Java bytecode that is interpreted on client computers. Many believe that Microsoft C# is a near-copy of Java.

Java Database Connectivity (JDBC). A standard interface by which application programs written in Java can access and process SQL databases (or table structures such as spreadsheets and text tables) in a DBMS-independent manner. While originally it did not stand for Java Database Connectivity, it does now and is an acronym.

Java platform. The complete set of Java tools provided by Sun Microsystems.

Java Server Pages (JSP). A combination of HTML and Java that is compiled into a Java servlet that is a subclass of the HttpServlet class. Java code embedded in a JSP has access to HTTP objects and methods. JSPs are used similarly to ASPs, but they are compiled rather than interpreted, as ASP pages are.

Java servlet. *See* servlet.

Java virtual machine. A Java bytecode interpreter that runs on a particular machine environment; for example, Intel 386 or Alpha. Such interpreters are usually embedded in browsers, included with the operating system, or included as part of a Java development environment.

JavaScript. A proprietary scripting language owned by Netscape. The Microsoft version is called JScript; the standard version is called ECMAScript-262. These are easily learned interpreted languages that are used for both Web server and Web client application processing. Sometimes written as *Java Script*.

JDBC. *See* Java Database Connectivity (JDBC).

Join. A relational algebra operation on two relations, A and B, which produces a third relation, C. A row of A is concatenated with a row of B to form a new row in C if the rows in A and B meet a restriction concerning their values. Normally, the restriction is that one or more columns of A equal one or more columns of B. For example, suppose that A1 is an attribute in A, and B1 is an attribute in B. The join of A with B in which A1 = B1 will result in a relation, C, having the concatenation of rows in A and B in which the value of A1 equals the value of B1. In theory, restrictions other than equality are allowed; a join could be made in which A1 < B1. Such nonequal joins are not used in practice, however.

Join operation. In SQL, the process of combining data rows from two tables. *See* join.

JScript. A proprietary scripting language owned by Microsoft. The Netscape version is called JavaScript; the standard version is called ECMAScript-262. These are easily learned interpreted languages used for both Web server and Web client application processing.

JSP. *See* Java Server Page.

Key. (1) A group of one or more attributes identifying a unique row in a relation. Because relations may not have duplicate rows, every relation must have at least one key, which is the composite of all of the attributes in the relation. A key is sometimes called a logical key. (2) With some relational DBMS products, an index on a column used to improve access and sorting speed. It is sometimes called a physical key.

Labeled namespace. In an XML Schema document, a namespace that is given a name (label) within the document. All elements preceded by the name of the labeled namespace are assumed to be defined in that labeled namespace.

LAMP. A version of AMP that runs on Linux. *See* AMP.

Language Integrated Query (LINQ). A Microsoft .NET Framework component that allows SQL queries to be run directly from application programs.

Amazon Ex. 1008.0601

Logical unit of work (LUW). An equivalent term for transaction. *See* transaction.

LEFT OUTER join. A join that includes all the rows of the first table listed in the SQL statement (the "left" table) regardless of whether or not they have a matching row in the other table.

Lift. In market basket analysis, confidence divided by the base probability of an item purchase.

Lock. The process of allocating a database resource to a particular transaction in a concurrent-processing system. The size of the resource locked is known as the lock granularity. With an exclusive lock, no other transaction may read or write the resource. With a shared lock, other transactions may read the resource, but no other transaction may write it.

Lock granularity. The size of a locked data element. The lock of a column value of a particular row is a small granularity lock, and the lock of an entire table is a large granularity lock.

Log. A file containing a record of database changes. The log contains before-images and after-images.

Logistic regression. A form of supervised data mining that estimates the parameters of an equation to calculate the odds that a given event will occur.

Lost update problem. Same as concurrent update problem.

M score. In RFM analysis, the "how much money" score, which reflects how much a customer spends per purchase. *See* RFM Analysis.

Many-to-many (N:M) relationship. A relationship in which one parent entity instance (or row in the parent table) can be associated with many child entity instances (or rows in the child table). At the same time, one child entity instance (or row in the child table) can be associated with many parent entity instances (or rows in the parent table). In an actual database, these relationships are tansformed into two one-to-many relationships between the original entities (tables) and an intersection table.

Market basket analysis. A type of data mining that estimates the correlations of items that are purchased together. *See also* confidence and lift.

MAX. In SQL, a function that determines the largest value in a set of numbers. *See* SQL built-in functions.

Maximum cardinality. (1) In a binary relationship in the entity-relationship model, the maximum number of entities on each side of the relationship. Common values are 1:1, 1:N, and N:M. (2) In a relationship in the relational model, the maximum number of rows on each side of the relationship. Common values are 1:1 and 1:N. An N:M relationship is not possible in the relational model.

Measure. In OLAP, the source data for the cube—data that are displayed in the cells. It may be raw data or it may be functions of raw data, such as SUM, AVG, or other computations.

Media failure. A failure that occurs when the DBMS is unable to write to a disk. Usually caused by a disk head crash or other disk failure.

Metadata. Data concerning the structure of data that are used to describe tables, columns, constraints, indexes, and so forth. Metadata is data about data.

Method. A program attached to an object-oriented programming (OOP) object. A method can be inherited by lower-level OOP objects.

Microsoft SQL Server 2008 Management Studio. The GUI utility that is used with Microsoft SQL Server 2008.

Microsoft Windows PowerShell. A Microsoft command line utility.

MIN. In SQL, a function that determines the smallest value in a set of numbers. *See* SQL built-in functions.

Minimum cardinality. (1) In a binary relationship in the entity-relationship model, the minimum number of entities required on each side of a relationship. (2) In a binary relationship in the relational model, the minimum number of rows required on each side of a relationship. Common values of minimum cardinality for both definitions are optional to optional (O-O), mandatory to optional (M-O), optional to mandatory (O-M), and mandatory to mandatory (M-M).

Minimum cardinality enforcement actions. Activities that must be taken to preserve minimum cardinality restrictions. Summarized in Figure 6-28. *See also* referential integrity (RI) actions.

Modification anomaly. In a relation, the situation that exists when the storage of one row records facts about two or more entities or when the deletion of one row removes facts about two or more entities.

Amazon Ex. 1008.0602

Multivalued dependency. A condition in a relation with three or more attributes in which independent attributes appear to have relationships they do not have. Formally, in a relation R (A, B, C), having key (A, B, C) where A is matched with multiple values of B (or of C or both), B does not determine C, and C does not determine B. An example is the relation EMPLOYEE (EmpNumber, EmpSkill, DependentName), where an employee can have multiple values of EmpSkill and DependentName. EmpSkill and DependentName do not have any relationship, but they do appear to in the relation.

MUST constraint. A constraint that requires one entity to be combined with another entity.

MUST COVER constraint. The binary relationship indicates all combinations that must appear in the ternary relationship.

MUST NOT constraint. The binary relationship indicates combinations that are not allowed to occur in the ternary relationship.

MySQL Query Browser. The GUI utility used with MySQL 5.1.

Natural join. A join of a relation A having attribute A1 with relation B having attribute B1, where A1 equals B1. The joined relation, C, contains either column A1 or B1, but not both. Contrast this with equijoin.

Neural networks. A form of supervised data mining that estimates complex mathematical functions for making predictions. The name is a misnomer. Although there is some loose similarity between the structure of a neural network and a network of biological neurons, the similarity is only superficial.

N:M. The abbreviation for a many-to-many relationship between two entities or relations.

Nonidentifying connection relationships. In IDEF1X, 1:1 and 1:N HAS-A relationships that do not involve ID-dependent entities.

Nonidentifying relationship. In data modeling, a relationship between two entities such that one is *not* ID-dependent on the other. *See* identifying relationship.

Nonintegrated data. Data that are stored in two incompatible information systems.

Non-prime attribute. In normalization, an attribute that is not contained in any candidate key.

Nonrepeatable reads. The situation that occurs when a transaction reads data it has previously read and finds modifications or deletions caused by a committed transaction.

Nonspecific IDEF1X relationships. In IDEF1X, an N:M relationship.

Normal form. A rule or set of rules governing the allowed structure of relations. The rules apply to attributes, functional dependencies, multivalue dependencies, domains, and constraints. The most important normal forms are first normal form, second normal form, third normal form, Boyce-Codd normal form, fourth normal form, fifth normal form, and domain/key normal form.

Normalization. (1) The process of constructing one or more relations such that in every relation the determinant of every functional dependency is a candidate key (BCNF). (2) The process of removing multivalued dependencies (4NF). (3) In general, the process of evaluating a relation to determine whether it is in a specified normal form and of converting it to relations in that specified normal form, if necessary.

NOT NULL constraint. In SQL, a constraint that specifies that a column must contain a value in every row.

Not-type-valid document. An XML document that either does not conform to its Document Type Declaration (DTD) or does not have a DTD. *See also* type-valid document and schema-valid document.

NULL constraint. In SQL, a constraint that specifies that a column may have empty cells in some or all rows.

Null status. Whether the column has a NULL constraint or a NOT NULL constraint. *See* NOT NULL constraint, and NULL constraint.

Null value. An attribute value that has never been supplied. Such values are ambiguous and can mean that (a) the value is unknown, (b) the value is not appropriate, or (c) the value is known to be blank.

Object class. In object-oriented programming, a set of objects with a common structure. *See* object-oriented programming (OOP).

Amazon Ex. 1008.0603

Object Linking and Embedding (OLE). Microsoft's object standard. OLE objects are Component Object Model (COM) objects and support all required interfaces for such objects.

Object persistence. In object-oriented programming, the characteristic that an object can be saved to nonvolatile memory, such as a disk. Persistent objects exist between executions of a program.

Object-oriented DBMS (OODBMS or ODBMS). A DBMS that can store the objects similar to those used in OOP. *See* object-oriented programming (OOP).

Object-oriented programming (OOP). A programming methodology that defines objects and the interactions between them to create application programs.

Object-relational DBMS. DBMS products that support both relational and object-oriented programming data structures, such as Oracle.

ODBC. *See* Open Database Connectivity standard.

ODBC conformance level. In ODBC, definitions of the features and functions that are made available through the driver's application program interface (API). A driver API is a set of functions that the application can call to receive services. There are three conformance levels: Core API, Level 1 API, and Level 2 API.

ODBC data source. In the ODBC standard, a database and its associated DBMS, operating system, and network platform.

ODBC Data Source Administrator. The application used to create ODBC data sources.

ODBC Driver. In ODBC, a program that serves as an interface between the ODBC driver manager and a particular DBMS product. Runs on the client machines in a client-server architecture.

ODBC Driver Manager. In ODBC, a program that serves as an interface between an application program and an ODBC driver. It determines the required driver, loads it into memory, and coordinates activity between the application and the driver. On Windows systems, it is provided by Microsoft.

ODBC multiple-tier driver. In ODBC, a two-part driver, usually for a client-server database system. One part of the driver resides on the client and interfaces with the application; the second part resides on the server and interfaces with the DBMS.

ODBC single-tier driver. In ODBC, a database driver that accepts SQL statements from the driver manager and processes them without invoking another program or DBMS. A single-tier driver is both an ODBC driver and a DBMS. It is used in file-processing systems.

ODBC SQL conformance levels. ODBC SQL conformance levels specify which SQL statements, expressions, and data types an OBDC driver can process. Three SQL conformance levels are defined: Minimum SQL Grammar, Core SQL Grammar, Extended SQL Grammar.

OLAP. *See* OnLine Analytical Processing.

OLAP cube. In OLAP, a presentation structure having axes upon which data dimensions are placed. Measures of the data are shown in the cells of the cube. Also called a hypercube.

OLAP report. The output of an OLAP analysis in tabular format. For example, this can be an Excel Pivot Table. *See* OLAP cube.

OLAP server. A server specifically developed to perform OLAP analyses.

OLE DB. The COM-based foundation of data access in the Microsoft world. OLE DB objects support the OLE object standard. ADO is based on OLE DB.

1:N. The abbreviation for a one-to-many relationship between two entities or relations.

OnLine Analytical Processing (OLAP). A form of dynamic data presentation in which data are summarized, aggregated, deaggregated, and viewed in the frame of a table or a cube.

Online transaction processing (OLTP) system. An operational database system available for, and dedicated to, transaction processing.

One-to-many (1:N) relationship. A relationship in which one parent entity instance (or row in the parent table) can be associated with many child entity instances (or rows in the child table). At the same time, one child entity instance (or row in the child table) can be associated with only one parent entity instance (or row in the parent table).

One-to-one (1:1) relationship. A relationship in which one parent entity instance (or row in the parent table) can be associated with only one child entity instance (or row in the child table). At the same time, one child entity instance (or row in the child table) can be associated with only one parent entity instance (or row in the parent table).

Amazon Ex. 1008.0604

Open Database Connectivity standard (ODBC). A standard interface by which application programs can access and process relational databases, spreadsheets, text files, and other table-like structures in a DBMS or in a program-independent manner. The driver manager portion of ODBC is incorporated into Windows. ODBC drivers are supplied by DBMS vendors, Microsoft, and by third-party software developers.

Operational system. A database system in use for the operations of the enterprise, typically an OLTP system. *See* Online transaction processing (OLTP) system.

Optimistic locking. A locking strategy that assumes no conflict will occur, processes a transaction, and then checks to determine whether conflict did occur. If conflict did occur, no changes are made to the database and the transaction is repeated. *See also* pessimistic locking.

Oracle SQL Developer. The GUI utility for Oracle Database 11g.

Outer join. A join in which all of the rows of a table appear in the join result, regardless of whether they have a match in the join condition. In a left outer join, all of the rows in the left-hand relation appear; in a right outer join, all of the rows in the right-hand relation appear.

Overlapping candidate keys. Two candidate keys are said to be overlapping candidate keys if they have one or more attributes in common.

Parameter. A data value that is passed as input to a stored procedure or other application.

Parent. An entity or row on the one side of a one-to-many relationship.

Parent mandatory and child mandatory (M-M). A relationship where the minimum cardinality of the parent is 1 and the minimum cardinality of the child is 1.

Parent mandatory and child optional (M-O). A relationship where the minimum cardinality of the parent is 1 and the minimum cardinality of the child is 0.

Parent optional and child mandatory (O-M). A relationship where the minimum cardinality of the parent is 0 and the minimum cardinality of the child is 1.

Parent optional and child optional (O-O). A relationship where the minimum cardinality of the parent is 0 and the minimum cardinality of the child is 0.

Partially dependent. In normalization, a condition where an attribute is dependent on only part of a composite primary key instead of on the whole key.

Passive repository. Repositories that are filled only when someone takes the time to generate the needed metadata and place it in the repository. *See* data repository.

Persistent object. In object-oriented programming, an object that has been written to persistent storage.

Personal database system. A DBMS product intended for use by an individual or small workgroup. Such products typically include application development tools such as form and report generators in addition to the DBMS. For example, Microsoft Access 2007.

Pessimistic locking. A locking strategy that prevents conflict by locking data resources, processing the transaction, and then unlocking the data resources. *See also* optimistic locking and deadlock.

Phantom reads. The situation that occurs when a transaction reads data it has previously read and finds new rows that were inserted by a committed transaction.

PHP. *See* PHP: Hypertext Processor.

PHP Data Objects (PDO). A consistent data-access specification for PHP that allows a programmer to use the same functions independent of which DBMS is being used.

PHP: Hypertext Processor (PHP). A Web page scripting language used to create dynamic Web pages. It now includes an object-oriented programming component and PHP Data Objects (PDO). *See* PHP Data Objects (PDO).

PL/SQL. *See* Procedural Language/SQL.

POST method. In PHP, a method of passing data values from one Web page to another for processing.

PowerShell sqlps utility [MSSQL]. In SQL Server 2008, an add-in to the Microsoft PowerShell command line utility that allows it to work with SQL Server.

Primary key. A candidate key selected to be the key of a relation; the primary key is used as a foreign key for representing relationships.

PRIMARY KEY constraint. In SQL, a constraint statement used to create a primary key for a table.

Prepared Statement objects.

Amazon Ex. 1008.0605

Procedural language. A programming language where each step necessary to obtain a result must to specified. The language may have the ability to contain sets of steps in structures called procedures or subprocedures.

Procedural Language/SQL (PL/SQL). An Oracle-supplied language that augments SQL with programming language structures such as while loops, if-then-else blocks, and other such constructs. PL/SQL is used to create stored procedures and triggers.

Processing rights and responsibilities. Organizational policies regarding which groups can take which actions on specified data items or other collections of data.

Program/data independence. The condition existing when the structure of the data is not defined in application programs. Rather, it is defined in the database and then the application programs obtain it from the DBMS. In this way, changes can be made in the data structures that may not necessarily be made in the application programs.

Programmer. A person who creates application programs in a programming language.

Project-Join normal form (PJ/NF). Another name for 5NF. *See* Fifth normal form (5NF).

Property. Same as attribute.

Prototype. A quickly developed demonstration of an application or portion of an application.

Pull report. In reporting systems, a report that must be requested by users.

Push report. In reporting systems, a report that is sent to users according to a schedule.

QBE. *See* query by example.

Query. A request for database data that meets specific criteria. This can be thought of as asking the database a question and getting an answer in the form of the data returned.

Query by example (QBE). A style of query interface, first developed by IBM but now used by Microsoft Access 2007 and other DBMS products, that enables users to express queries by providing examples of the results they seek.

Question mark (?) wildcard character. A character used in Access 2007 queries to represent a single unspecified characters. *See* SQL underscore (_) wildcard character.

R score. In RFM analysis, the "how recently" score, which reflects how recently a customer made a purchase. *See* RFM Analysis.

Range constraint. In SQL, a constraint that specifies that data values must be within a specific range of values.

Read committed. A level of transaction isolation that prohibits dirty reads but allows nonrepeatable reads and phantom reads.

Read uncommitted. A level of transaction isolation that allows dirty reads, nonrepeatable reads, and phantom reads.

Record. (1) In a relational model, a synonym for row and tuple. (2) A group of fields pertaining to the same entity; used in file-processing systems.

Recordset. An ADO.NET object that encapsulates a relation; created as the result of the execution of an SQL statement or a stored procedure.

Recovery via reprocessing. Recovering a database by restoring the last full backup, and then recreating each transaction since the backup.

Recovery via rollback/rollforward. Recovering a database by restoring the last full backup, and then using data stored in a transaction log to modify the database as needed by either adding transactions (roll forward) or removing erroneous transactions (rollback).

Recursive relationship. A relationship among entities or rows of the same type. For example, if CUSTOMERs refer to other CUSTOMERs, the relationship is recursive.

ReDo files. In Oracle, backups of rollback segments used for backup and recovery. ReDo files may be online or off-line.

Referential integrity (RI) actions. In general, rules that specify the activities that must take place when insert, update, or delete actions occur on either the parent or child entities in a relationship. In this text, we use referential integrity actions only to document activities needed to preserve required parents. Other actions can be defined as part of the database design. See minimum cardinality enforcement actions and Figure 6-28.

Referential integrity constraint. A relationship constraint on foreign key values. A referential integrity constraint specifies that the values of a foreign key must be a subset of the values of the primary key to which it refers.

Amazon Ex. 1008.0606

Regression analysis. A form of supervised data mining in which the parameters of equations are estimated by data analysis.

Relation. A two-dimensional array containing single-value entries and no duplicate rows. Values for a given entity are shown in rows; values of attributes of that entity are shown in columns. The meaning of the columns is the same in every row. The order of the rows and columns is immaterial.

Relational data model. A data model in which data are stored in relations, and relationships between rows are represented by data values.

Relational database. A database consisting of relations. In practice, relational databases contain relations with duplicate rows. Most DBMS products include a feature that removes duplicate rows when necessary and appropriate. Such a removal is not done as a matter of course because it can be time-consuming to enforce.

Relational schema. A set of relations with interrelation constraints.

Relationship. An association between two entities or rows.

Relationship cardinality constraint. A constraint on the number of rows that can participate in a relationship. Minimum cardinality constraints determine the number of rows that must participate; maximum cardinality constraints specify the largest number of rows that can participate.

Relationship class. An association between entity classes.

Relationship instance. (1) An association between entity instances, (2) a specific relationship between two tables in a database.

Repeatable read. A level of transaction isolation that disallows both dirty reads and nonrepeatable reads. Phantom reads can occur.

Replication. For both Oracle, SQL Server, and MySQL, a term that refers to databases that are distributed on more than one computer.

Report. A formatted set of information created to meet a user's need.

Report authoring. In a reporting system, connecting to the data source, creating the report structure, and formatting the report.

Report delivery. In a reporting system, pushing the reports to users, or allowing them to pull the reports as needed.

Report management. In a reporting system, defining who receives which reports, when, and by what means.

Reporting system. A business intelligence system that processes data by filtering, sorting, and making simple calculations. OLAP is a type of reporting system. Contrast with data mining systems.

Repository. A collection of metadata about database structure, applications, Web pages, users, and other application components. Active repositories are maintained automatically by tools in the application-development environment. Passive repositories must be maintained manually.

Reserved word. A word that has a special meaning in the DBMS or ODBC, and should not be used as a table, column, or other name in a database. See DBMS reserved word.

Resource locking. *See* lock.

Reverse-engineered (RE) data model. The structure that results from reverse engineering. It is not really a data model, because it includes physical structures such as intersection tables. It is, instead, a thing unto itself; midway between a data model and a relational database design.

Reverse engineering. The process of reading the structure of an existing database and creating a reverse-engineered data model from that schema.

RFM analysis. A type of reporting system in which customers are classified according to how recently (R), how frequently (F), and how much money (M) they spend on their orders.

RIGHT OUTER constraint. A join that include all the rows of second table listed in the SQL statement (the "right" table) regardless of whether or not they have a matching row in the other table.

Role. In database administration, a defined set of permissions that can be assigned to users or groups.

Amazon Ex. 1008.0607

Rollback. The process of recovering a database in which before-images are applied to the database to return to an earlier checkpoint or other point at which the database is logically consistent.

Rollforward. The process of recovering a database by applying after-images to a saved copy of the database to bring it to a checkpoint or other point at which the database is logically consistent.

Root. (1) In MySQL, the name of the DBMS administrator account. (2) The top record, row, or node in a tree. A root does not have a parent.

Row. A group of columns in a table. All the columns in a row pertain to the same entity. A row is the same as a tuple and a record.

Rowset. In OLE DB, an abstraction of data collections such as recordsets, e-mail addresses, and nonrelational and other data.

SAX. Simple API (Application Program Interface) for XML. An event-based parser that notifies a program when the elements of an XML document have been encountered during document parsing.

Schema. (1) In MySQL, a synonym for *database*. (2) A complete logical view of the database.

Schema-valid document. An XML document that conforms to its XML Schema definition.

SCN. *See* system change number.

Scrollable cursor. A cursor type that enables forward and backward movement through a recordset. Three scrollable cursor types discussed in this text are snapshot, keyset, and dynamic.

Second normal form (2NF). A relation in first normal form in which all nonkey attributes are dependent on all of the key attributes.

Self-describing. In a database, the characteristic of including data about the database in the database itself. Thus, the data that defines a table is included in a database along with the data that is contained in that table. This descriptive data is called *metadata*. *See* table, relation, and metadata.

Semantic object model. The constructs and conventions used to create a model of the users' data. The things in the users' world are represented by semantic objects (sometimes called objects). Relationships are modeled in the objects, and the results are usually documented in object diagrams.

Sequence. The Oracle Database 11g SQL statement used to create surrogate key values.

Serializable. A level of transaction isolation that disallows dirty reads, nonrepeatable reads, and phantom reads.

Service provider. An OLE DB data provider that transforms data. A service provider is both a data consumer and a data provider.

Servlet. A compiled, machine-independent Java bytecode program that is run by a Java virtual machine located on a Web server.

SGML. *See* Standard Generalized Markup Language.

Shared lock. A lock against a data resource in which only one transaction may update the data, but many transactions can concurrently read that data.

Shrinking phase. In two-phase locking, the stage at which locks are released but no lock is acquired.

Sibling. A record or node that has the same parent as another record or node.

Simple Object Access Protocol. A standard used for remote procedure calls. It uses XML for definition of the data and HTTP for transport. Contrast with SOAP.

Slowly changing dimension. In a dimensional database, a data column with values that change occasionally but irregularly over time, For example, a customer's address or phone number.

Snowflake schema. In a dimensional database or an OLAP database, the structure of tables such that dimension tables may be several levels away from the table storing the measure values. Such dimension tables are usually normalized. Contrast with star schema.

SOAP. Originally, Simple Object Access Protocol. Today, it is a protocol for remote procedure calls that differs from the Simple Object Access Protocol because it involves transport protocols in addition to HTTP. It is no longer an acronym.

Software development kit (SDK). A group of development tools provided to programmers to help them create applications.

Amazon Ex. 1008.0608

SQL. *See* Structured Query Language.
SQL AND operator. The SQL operator used to combine conditions in an SQL WHERE clause.
SQL built-in functions. In SQL, the functions COUNT, SUM, AVG, MAX, or MIN.
SQL CMD utility. A command line utility used with SQL Server 2008.
SQL CREATE TABLE statement. The SQL command used to create a database table.
SQL CREATE VIEW statement. The SQL command used to create a database view.
SQL FROM clause. The part of an SQL SELECT statement that specifies conditions used to determine which tables are used in a query.
SQL GROUP BY clause. The part of an SQL SELECT statement that specifies conditions for grouping rows when determining the query results.
SQL HAVING clause. The part of an SQL SELECT statement that specifies conditions used to determine which rows are in the qroupings in a GROUP BY clause.
SQL OR operator. The SQL operator used to specify alternate conditions in an SQL WHERE clause.
SQL ORDER BY clause. The part of an SQL SELECT statement that specifies how the query results should be sorted when they are displayed.
SQL percent sign (%) wildcard character. The standard SQL wildcard character used to specify multiple characters. Microsoft Access 2007 uses an asterisk (*) character instead of the underscore character.
SQL script. A set of SQL statements that are intended to be executed as a group.
SQL script comment. A comment in an SQL script. *See* SQL script.
SQL script file. A file that holds an SQL script for repeated use. *See* SQL script.
SQL SELECT clause. The part of an SQL SELECT statement that specifies which columns are in the query results.
SQL SELECT * statement. A variant of an SQL SELECT query that returns all columns for all tables in the query.
SQL SELECT . . . for XML statement. A variant of an SQL SELECT query that returns the query results in XML format.
SQL SELECT/FROM/WHERE framework. The basic structure of an SQL query. *See* SQL SELECT clause, SQL FROM clause, SQL WHERE clause, SQL ORDER BY clause, SQL GROUP BY clause, SQL HAVING clause, SQL AND operator, and SQL OR operator.
SQL underscore (_) wildcard character. The standard SQL wildcard character used to specify a single character. Microsoft Access 2007 uses a question mark (?) character instead of the underscore character.
SQL WHERE clause. The part of an SQL SELECT statement that specifies conditions used to determine which rows are in the query results.
SQL view. A relation that is constructed from a single SQL SELECT statement. SQL views have at most one multivalued path. The term *view* in most DBMS products, including Access, Oracle, and SQL Server, means SQL view.
SQL*Plus. A command line utility in Oracle Database 11g.
Standard Generalized Markup Language (SGML). A standard means for tagging and marking the format, structure, and content of documents. HTML is an application of SGML. XML is a subset of SGML.
Star schema. In a dimensional database or an OLAP database, the structure of tables such that every dimension table is adjacent to the table storing the measure values. In the star schema, the dimension tables are often not normalized. Contrast with snowflake schema.
Statement-level consistency. All rows impacted by a single SQL statement are protected from changes made by other users during the execution of the statement. Contrast with transaction-level consistency.
Static cursor. A cursor that takes a snapshot of a relation and processes that snapshot.
Static report. In reporting systems, a report that is prepared once from underlying data and does not change when the underlying data change. Contrast with dynamic report.
Stock-keeping unit (SKU). A unique identifier for each product available from a vendor.
Stored procedure. A collection of SQL statements stored as a file that can be invoked by a single command. Usually, DBMS products provide a language for creating stored procedures that augments SQL with programming language constructs. Oracle provides PL/SQL

Amazon Ex. 1008.0609

for this purpose; SQL Server provides T-SQL; MySQL also adds procedural capabilities, but does not use a separate name for these additions. With some products, stored procedures can be written in a standard language such as Java. Usually, stored procedures are stored within the database itself.

Strong entity. In an entity-relationship model, any entity whose existence in the database does not depend on the existence of any other entity. *See also* ID-dependent entity and weak entity.

Strong password. A password that meets requirements intended to make it difficult to unencrypt.

Structured Query Language (SQL). A language for defining the structure and processing of a relational database. It can be used as a stand-alone language or it may be embedded in application programs. SQL has been adopted as a national standard by the American National Standards Institute (ANSI). The most common version used today is SQL-92, the version adopted by ANSI in 1992. SQL was originally developed by IBM.

Structured schema. An XML schema that is not flat.

Stylesheet. A document used by XSLT to indicate how to transform the elements of an XML document into another format.

Subquery. In SQL, a SELECT statement within another SELECT statement.

Subtype. In generalization hierarchies, an entity or object that is a subspecies or subcategory of a higher-level type, called a supertype. For example, ENGINEER is a subtype of EMPLOYEE.

SUM. In SQL, a function that adds up a set of numbers. *See* SQL built-in functions.

Supertype. In generalization hierarchies, an entity or object that logically contains subtypes. For example, EMPLOYEE is a supertype of ENGINEER, ACCOUNTANT, and MANAGER.

Supervised data mining. A form of data mining in which an analyst creates a prior model or hypothesis and then uses the data to test that model or hypothesis.

Support. In market basket analysis, the probability that two items will be purchased together.

Surrogate key. A unique, system-supplied identifier used as the primary key of a relation. It is created when a row is created, it never changes, and it is destroyed when the row is deleted. The values of a surrogate key have no meaning to the users and are usually hidden within forms and reports.

System change number (SCN). In Oracle, a database-wide value that is used to order changes made to database data. The SCN is incremented whenever database changes are committed.

System data source. An ODBC data source that is local to a single computer and can be accessed by that computer's operating system and select users of that operating system.

Table. A database structure of rows and columns to create cells that hold data values. Also known as a *relation* in a relational database, although strictly only tables that meet specific conditions can be called relations. *See* relation.

TableName.ColumnName syntax. A syntax used to indicate which table a column is associated with. For example, CUSTOMER.LastName indicates the LastName column in the CUSTOMER table.

Tabular data provider. An OLE DB data provider that presents data in the form of rowsets.

Ternary relationship. A relationship between three entities.

Third normal form (3NF). A relation in second normal form that has no transitive dependencies.

Three-tier architecture. A system of computers having a database server, a Web server, and one or more client computers. The database server hosts a DBMS, the Web server hosts an HTTP server, and the client computer hosts a browser. Each tier can run a different operating system.

Time dimension. A required dimension table in a dimensional database. The time dimension allows the data to be analyzed over time.

Transaction. (1) A group of actions that is performed on the database automatically; either all actions are committed to the database, or none of them are. (2) In general, the record of an event in the business world.

Transaction isolation level. The degree to which a database transaction is protected from actions by other transactions. The 1992 SQL standard specified four isolation levels: Read Uncommitted, Read Committed, Repeatable Reads, and Serializable.

Amazon Ex. 1008.0610

Transaction-level consistency. All rows impacted by any of the SQL statements in a transaction are protected from changes during the entire transaction. This level of consistency is expensive to enforce and reduces throughput. It may also mean that a transaction cannot see its own changes. Contrast with statement-level consistency.

Transact-SQL (T-SQL). A Microsoft-supplied language that is part of SQL Server. It augments SQL with programming language structures such as while loops, if-then-else blocks, and other such constructs. Transact-SQL is used to create stored procedures and triggers.

Transitive dependency. In a relation having at least three attributes, for example, R (A, B, C), the situation in which A determines B, B determines C, but B does not determine A.

Transactional system. A database dedicated to processing transactions such as product sales and orders. It is designed to make sure that only complete transactions are recorded in the database.

Tree. A collection of records, entities, or other data structures in which each element has at most one parent, except for the top element, which has no parent.

Trigger. A special type of stored procedure that is invoked by the DBMS when a specified condition occurs. BEFORE triggers are executed before a specified database action, AFTER triggers are executed after a specified database action, and INSTEAD OF triggers are executed in place of a specified database action. INSTEAD OF triggers are normally used to update data in SQL views.

T-SQL. *See* Transact-SQL.

Tuple. Same as row.

Two dashes (- -). Symbology to indicate a single line comment in a stored procedure or a trigger in SQL Server 2008, Oracle Database 11g, and MySQL 5.1.

Two-phase locking. The procedure by which locks are obtained and released in two phases. During the growing phase, the locks are obtained; during the shrinking phase, the locks are released. After a lock is released, no other lock will be granted that transaction. Such a procedure ensures consistency in database updates in a concurrent-processing environment.

Two-tier architecture. In a Web-based database processing environment, the Web server and the DBMS are running on the same computer. One tier is for the Web browsers, and one is for the Web server/DBMS computer.

Type domain. In IDEF1X, a domain that is defined as a subset of a base domain or another type of domain.

Type-valid document. An XML document that conforms to its Document Type Declaration (DTD). Contrast with not-type-valid document.

UML. *See* Unified Modeling Language.

Unified Modeling Language (UML). A set of structures and techniques for modeling and designing object-oriented programs and applications. It is a set of tools for object-oriented development that has led to a development methodology. UML incorporates the entity-relationship model for data modeling.

UNIQUE constraint. In SQL, a constraint that specifies that the values in a column must be unique.

Unsupervised data mining. A form of data mining in which analysts do not create a prior model or hypothesis, but rather let the data analysis reveal a model.

Updatable view. An SQL view that can be updated. Such views are usually very simple, and the rules that allow updating are normally quite restrictive. Nonupdatable views can be made updatable by writing application-specific INSTEAD OF triggers.

User. A person using an application.

User data source. An ODBC data source that is available only to the user who created it.

User group. A group of users. *See* user.

Variable. A value that may be assigned or calculated by a stored procedure or a trigger in SQL Server 2008, Oracle Database 11g, and MySQL 5.1.

VBScript. An easily learned, interpreted language used for both Web server and Web client applications processing.

WAMP. AMP running on a Windows operating system. *See* AMP.

Amazon Ex. 1008.0611

Weak entity. In an entity-relationship model, an entity whose logical existence in the database depends on the existence of another entity. All ID-dependent entities are weak, but not all weak entities are ID-dependent.

Web portal. A Web page designed to be an entrance point for a Web site. It may display information from several sources, and may require authentication to access.

World Wide Web Consortium (W3C). The group that creates, maintains, revises, and publishes standards for the World Wide Web including HTML, XML, and XHTML.

Wwwroot folder. The root folder or directory of a Web site on a Microsoft IIS Web server.

x..y cardinality format [UML]. The symbology format used in UML E-R diagrams to document minimum and maximum cardinalities. X records the minimum cardinality, and y records the maximum cardinality.

XHTML. The Extensible Hypertext Markup Language. A reformulation of HTML to XML standards of well-formed documents.

XML (Extensible Markup Language). A standard markup language that provides a clear separation between structure, content, and materialization. It can represent arbitrary hierarchies and hence can be used to transmit any database view.

XML Namespaces. A standard for assigning names to defined collections. X:Name is interpreted as the element Name as defined in namespace X. Y:Name is interpreted as the element Name as defined in namespace Y. Useful for disambiguating terms.

XML Schema. An XML document that defines the structure of other XML documents. Extends and replaces Document Type Declarations (DTDs).

XPath. A sublanguage within XSLT that is used to identify parts of an XML document to be transformed. Can also be used for calculations and string manipulation. Commingled with XSLT.

XPointer. A standard for linking one document to another. XPath has many elements from XPointer.

XQuery. A standard for expressing database queries as XML documents. The structure of the query uses XPath facilities, and the result of the query is represented in an XML format. Currently under development and likely to be important in the future.

XSL (XSLT Stylesheet). The document that provides the {match, action} pairs and other data for XSLT to use when transforming an XML document.

XSLT (Extensible Style Language: Transformations). A program (or process) that applies XSLT Stylesheets to an XML document to produce a transformed XML document.

Amazon Ex. 1008.0612

Index

Symbols & Numbers
* (asterisk)
  for COUNT function, 61
  as shortcut for all columns, 38
; (semicolon)
  and CREATE VIEW statement, 266
  for SQL statement termination, 37
1:1 relationship. *See* one-to-one (1:1) relationship
1NF (first normal form), 107, 108
1:N relationship. *See* one-to-many (1:N) relationship
2NF (second normal form), 107, 108–109
3NF (third normal form), 107, 109
4NF (fourth normal form), 108, 120
5NF (fifth normal form), 108, 120
/* and */, for SQL*Plus comments, 383
_ (underscore character), as wildcard, 58–59
' (single quotation marks), for comparison text in WHERE clause, 39
//, for PHP code segments, 461
/ (slash) for tags, 453
@@Identity function (SQL) Server, 387, 388
@@rowCount function, 394, 396
@ for T-SQL parameters, 383
[,] (square brackets), to use reserved words, 248
% (percent) wildcard character, 57

A
abstract data types, 439
abstraction in OLE DB, 439
Access. *See* Microsoft Access
ACID transactions, 332
ACT!, 6
Active Data Objects (ADO), 428, 430, 442–446
  purpose, 442
active repositories, 345
Active Server Pages, invoking ADO from, 430
ad-hoc queries, 31–32
ADO. *See* Active Data Objects (ADO)
ADO.NET
  datasets, 443
  object model, 442–446
    Command object, 444
    Connection object, 443, 444
    Errors collection, 440
    Fields collection, 440
    RecordSet objects, 440
after images, 341–343
AFTER triggers, 275, 276
  in SQL Server, 393, 394, 396
alerts, 545
aliases
  for columns, 270
  for table names, views and, 267–268
ALTER PROCEDURE statement, 385
ALTER statement (SQL), 251–252
ALTER TABLE statement (SQL)
  ADD clause, 306
  ALTER COLUMN, 307–308
  DROP COLUMN, 307
ambiguity, domains to reduce, B-6–7
American National Standards Institute (ANSI), 35
AmeriLink, 530
AMP (Apache-MySQL-PHP), 430
AND operator, 55
anomalies
  from data modification, 106–107
  eliminating
    from functional dependencies, 110–117
    from multivalued dependencies, 117–120
  from multivalued dependency, 108
Apache Tomcat, 449
Apache Web server, 352, 430, 449
applets, 447
application logic for Oracle, 10A-39–61
  PL/SQL language components, 10A-40–42
    block structure, 10A-40
    Control-of-Flow statements, 10A-40–41
    cursor statements, 10A-41–42
    output statements, 10A-42
  stored procedures, 10A-42–49
application logic for SQL Server, 381–407
  stored procedures, 282
  triggers, 393–407
application program interface for ODBC driver, 430, 432–433
applications, 339–340
  security, 335–340
archetype/instance pattern, 166–168
archetype/version objects, in semantic object model, E-26–29
ascending sort order, 54
AS keyword, 60
  to create table aliases, 261
association objects, in semantic object model, E-22–24
association relationships, 201–203
association tables, *vs.* intersection tables, 201–203
asterisk (*)
  for COUNT function, 61
  as shortcut for all columns in table, 38
atomic transactions, need for, 324–328
at sign (@) for T-SQL parameters, 383

Amazon Ex. 1008.0613

attributes, 98
of entities, 146
multivalued
in data model, 164–166
in ID-dependent entities, 204
in semantic object model, E-7
of semantic objects
cardinality, E-4
object instances, E-4–5
paired attributes, E-5
in XML Schema, 495–496
auditing user accounts, 338
authentication in SQL Server, 410, 412
authoring reports, 545–546
auto-increment column, 478
automation, and database redesign, 312–313
AVG function, 59, 61

B
backup data, 13
backups
consistent/inconsistent, 10A-65
of database during redesign, 304
MySQL, 10B-58–59
in Oracle, 10A-64–65
in SQL Server, 414–416, 414–417
types, 414–415
base domains, B-7–8
BCNF. *See* Boyce-Codd Normal Form (BCNF)
before images, 341
BEFORE triggers, 275
BEGIN TRANSACTION statement, 331–332, 389
WITH MARK, 414
behaviors of OLE DB rowsets, 439
BETWEEN keyword in WHERE clause, 56
Binary data type, 240–241
binary relationships, 148, D-8–14
representing complex networks, D-13–14
representing simple networks, D-12–13
representing trees, D-10
indexes, D-12
linked lists, D-10–11
blank spaces, stripping from strings, 63
blank value, 137
BLOB data type, 240, 241
Boyce-Codd Normal Form (BCNF), 107, 109–110
choosing not to use, 131–132
process for putting relation into, 109–110
bridge drivers, 446
B-trees, D-5–7
built-in functions, in HAVING clause, 66–67
bulk INSERT statement, 253
bulk-logged database recovery, 416
bulk updates, 259
business intelligence (BI) systems, 31, 526, 527, 528, 539–552. *See also* reporting systems
components, 544–545
data mining applications, 527, 528, 552–557
market basket analysis, 555–557
supervised, 555
techniques, 554
unsupervised, 554
data warehouses and data marts, 528, 531–532
functions, 546–549
OLAP (OnLine Analytical Processing), 549–552
vs. operational systems, 526–527
producing RFM report, 541–544
report media, 545–546
report mode, 546
report types, 545
RFM analysis, 540–541
bytecode interpreters, 447

C
calculations in SQL queries, 59–63
arithmetic, 62–63
built-in functions, 59–62
string manipulation, 63
Callable Statement objects (JDBC), 448
candidate (alternate) keys, 104–105
in data design, 104–105
determinant as, 104
Cape Codd Outdoor Sports (fictitious company), 32–35
ORDER_ITEM data table, 34–35
functional dependency in, 101–103
RETAIL_ORDER data table, 33–34
SKU_Data table, 35
functional dependency in, 103–104
cardinality of relationships, 149
in IDEF1X, 598–599
reducing, and data loss, 311–312
cascading deletions, 216
cascading updates, 216
case sensitivity, 243
in MySQL, 243
in Unix, 448
CASE tools, 323
casual relationships, 245–246
categorization clusters, B-4
categorization relationships, B-4–6
category entity, B-1, B-4
C#/C++ programming languages for triggers, 281
cell, A-17
Char data type, 240
CHAR(n) data type, 240
CHECK constraint, 243, 247–248
table of values as, 397
checkpoint, 343
Chen, Peter, 145
child constraint, trigger to enforce required, 309, 401–407
in MySQL, 10B-52–53
in Oracle, 10A-56–61
child in entity relationship, 135, 199
data design for required, 217
child tables
changing minimum cardinalities, 309
order of defining, 245
class attributes in UML, C-3–4
click-stream data, 530
client/server applications, 15
cluster analysis, 554
clustered indexes in SQL Server, 368–369
cmdlets, 356
CODASYL Committee, 22

Amazon Ex. 1008.0614

Codd, E. F., 22
Communications of the ACM, 22
normal forms, 107
on relationship characteristics, 98
collection of attributes, E-2
collections, 440
column constraints
adding and dropping, 308
adding in database redesign, 306–307
changing, 307
columns in database table, 3
adding and dropping, 251
in database redesign, 306–307
adding NOT NULL, 307
* (asterisk) as shortcut for all, 38
data constraints, 197
data type, 196
default value, 197
dropping, 307
for entity attributes, 204
INSERT command (SQL) with names of, 252–253
naming, 3
for SQL function results, 39
null status, 195
order in SQL results table, 53
properties, 195–197
selecting, with specific rows, 38–40
variable length, 99
views to display computed results, 269–270
views to hide, 270
command-line utility
in MySQL, 10B-8
in Oracle, 10A-15–16, 356
Command objects (ADO), 444, 445, 448
comments
in Microsoft Access remarks, A-11
in PHP (//), 461
in SQL*Plus (/* and */), 383
COMMIT command, and lock release, 329
Commit Transaction command, 324, 331
Communications of the ACM (Codd), 22
complete backup, 304, 415
complete category cluster, B-5
complex elements in XLM, 496
complex network, D-9–10
binary relationships representing, D-13–14
Component Object Model (COM), 439
composite determinant, 101
composite functional dependency, 101
composite identifiers, 147
composite keys, 104
composite objects in semantic object model, E-8–11
compound objects in semantic object model, E-11–12
computed columns, SQL views to display, 269
computer memory, 443
computers
backup and restore to move database between, 416
physical security, 338
concatenating tables with joins, 73, 75
concurrency control, 11, 323–328
atomic transactions needed, 324–328
in MySQL, 10B-54–55
in Oracle, 10A-61–63
in SQL Server, 407–409
transaction isolation levels, 408
concurrent transaction processing, 324
concurrent update problem, 327
confidence in marketing, 556
configuration control, 322
conformed dimension, 538
connecting to database, with JDBC, 430
Connection object (ADO), 443, 444
consistent backup, 10A-65
consistent transactions, 332–333
CONSTRAINT keyword, 247
constraints
adding and dropping, 251
names for, 244
contact manager, 6, 7
control-of-flow statements (SQL), 10B-34–35
conventions for database names, 3
correlated subqueries, 77, 297–301
to check functional dependencies, 299–300
vs. regular, 299
COUNT function, 59–61
for table assessment, 128
CREATE DATABASE command (SQL), 10A-6
Create Form using Wizard (Access), A-26–27
CREATE INDEX statement, 416
CREATE PROCEDURE statement (SQL), 385
Create Report using Wizard (Access), A-27
CREATE TABLE statement (SQL), 238–239
Query Analyzer to process, 418
summary of relationship definitions using, 246
CREATE VIEW statement (SQL), 266
in SQL Server, 379–381
crow's foot model, 151
cube (OLAP), 549
Cullinane Corporation, 33
curse of dimensionality, 530
cursor, 275, 334–335
concurrency, 408–409
vs. rowset, 441
cursor variable, 386–387
customer relationship management (CRM), 6
customers, RFM analysis of, 540–544

D

Data Adaptor, 443
data administration, 321
database administration, 19, 321–323
structure administration, 322–323
tasks summary, 322
database administrator (DBA), 322
skill requirements, 19–20
database application, 8
Database Configuration Assistant (DBCA), 6, 10A-5
database design, 16–19
in Access, A-1–4
data types for columns, A-3–4
relationships, A-2
table keys, A-2

Amazon Ex. 1008.0615

database design (*continued*)
common problems, 135–139
general remarks columns, 138–139
inconsistent values, 136–137
missing values, 137–138
multivalue, multicolumn, 135–137
data models, 18
from existing data, 16–17
from new information systems development, 17–18
for read-only databases, 132–134
customized duplicated tables, 134
denormalization, 132–134
for updatable databases, 129–132
database instance, 10A-6
database management system (DBMS). *See* DBMS (database management system)
database processing, 487
data structures for, D-1–18
binary relationships, D-8–14
flat files, D-1–8
secondary-key representations, D-16–18
and document processing, 487
history, 20–24
XML integration with, 19
database redesign, 18–19
adding and dropping columns, 306–307
additional SQL statements, 297–301
correlated subquery, 297–301
EXISTS and NOT EXISTS, 300–302
analysis of existing, 302–305
database backup and test databases, 304–305
dependency graphs, 303–304
reverse engineering, 302–303
changing data type or column constraints, 307–308
changing table names, 305–306
forward engineering, 312–313
need for, 296
relationship cardinalities, 308–311
maximum cardinality, 309–311
minimum cardinality, 308–309
relationships, adding and dropping, 312
databases, 3–6, 8, 12
backup and restore to move between computers, 416
characteristics of, 3
creating in Access, A-1–4
creating in SQL Server, 354–360
examples, 6–7
data mining applications, 8–9
digital dashboards, 7–8
e-commerce applications, 7
multiuser applications, 6–7
single-user applications, 6
size of, 7, 8
system components, 8–13
applications, 8
DBMS, 8, 11–12 [*See also* DBMS (database management system)]
SQL (Structured Query Language), 9–11
users, 8
database save, 341
database storage engines, 10B-3–4
Database Task Group (DBTG), 22
data constraints
as column property, 197
creating with SQL, 246–247
implementing, 247–248
triggers to enforce, 278–279
in MySQL, 10B-46–49
in Oracle, 10A-52–54
in SQL Server, 309
data consumers, 440
data, defined, 3
data definition language statements in SQL, 36
data design from data models
creating relationships, 198–214
among strong entities, 198–201
in Highline University data model, 213–214
recursive relationships, 208–210
subtype relationships, 208
ternary and higher-order relationships, 210–212
using ID-dependent entities, 201–208
creating tables for entities, 193–198
candidate (alternate) keys, 195
column properties, 195–197
primary key selection, 194
verifying normalization, 197–198
design for minimal cardinality, 214–222
documenting, 220–222
for mandatory-to-mandatory relationships, 219–220
for mandatory-to-optional relationships, 218
for optional-to-mandatory relationships, 219
summary, 222
when child is required, 217
when parent is required, 216–217
data dictionaries, 323
data entry
in Access, with Datasheet view, A-17–19
with ADO, 464
data extraction process, 35
data files, 136, 355
data grids, 168
data integrity problems from duplicated data, 107
Data Language/I (DL/I), 21–22
data loss, when reducing cardinalities, 311–312
data manipulation statements in SQL, 36
data marts, 16
vs. data warehouses, 531–532
data mining applications, 7–8, 527, 528, 552–557
market basket analysis, 555–557
supervised, 555
techniques, 555
unsupervised, 554
data model, 18. *See also* data design from data models; relational model
CODASYL DBTG model, 21, 22
complexity, 132, 133, 147, 174
entity-relationship data modeling, 145
variations in products, 151–157
inferring from user forms and reports, 157–174
for-use-by pattern, 171
ID-dependent relationships, 162–168

Amazon Ex. 1008.0616

mixing identifying and nonidentifying patterns, 168–171
recursive patterns, 172–174
strong entity patterns, 158–161
purpose, 145
for View Ridge Gallery database, 223–224
data modeling process, 174–180
letters (correspondence) in, 178–180
report analysis, 175–178
data providers, 440
in ADO.NET, 442–443
in OLE DB, 440–441, 443
data repository, maintaining, 344–345
datasets in ADO.NET, 442, 443
data source for ODBC, 431
name, 434–437
data sublanguage, 36
data types, 196
adding in database redesign, 306–307
for columns
in Access, A-12
changing, 307–308
DBMS support for, 239
in Oracle, 240
varieties in Web environment, 429
data, *vs.* information, 5–6
data warehouses, 16, 528–532
components, 528–531
vs. data marts, 531–532
DATE data type, 240, 241
Datetime data type, 240, 241
DBA (database administrator), 322
skill requirements, 19–20
DBMS (database management system), 8, 11–12
functions for formatting query results, 68
history, 23–24
managing, 343–345
security, 337
guidelines, 337–339
SQL statement submission to, 40–52
dbo schema, 411
deadlock, 329
deadly embrace, 329
decision support systems (DSSs). *See* business intelligence (BI) systems
decision tree analysis, 555
DECLARE CURSOR statement (SQL), 384, 408–409
DEFAULT constraint, 242, 306
default logins, 338
default values, 197
as column property, 197
creating with SQL, 246–248
implementing, 247
triggers to set, 276–277
in SQL Server, 394–397
degree of relationships, 148
DELETE CASCADE, 245, 252
DeleteCommand property of data adapter, 444
DELETE NO ACTION, 245
DELETE statement (SQL), 260
deletion anomaly, 106, 110, 118
delimited identifier, 10A-20
delimiter statements (SQL), 10B-34
delivery of reports, 549
denormalization, 132–133
dependency. *See* functional dependencies
dependency graphs, 303–304
descending sort order, 54
DESC keyword (SQL), 54
determinant, 100
composite, 101
unique values for, 104
Developer GUI tool
in MySQL, 10B-4–6
in Oracle, 10A-5, 10A-23–26
diamond, in IDEF1X, 598
differential backup, 414
digital dashboards, 7–8, 545
dimensional database, 532–539
dimensional model, 537
multiple fact tables and conformed dimensions, 537–539
star schema, 533–537
dimension in OLAP, 549
dimension tables, 533
dirty data, 530
dirty reads, 10A-62, 10A-63, 333
discriminator, 157
in category clusters, B-5
displays
in MySQL, 10B-8–10, 10B-16–17
in Oracle, 10A-5, 10A-23–26
distinct identities, E-3
DISTINCT keyword (SQL), 37–38
with COUNT function, 61
in subqueries, 72
distributed database processing, 430
DK/NF (domain key normal form), 108
DOCTYPE, 488, 489
documentation, 323
for default values, 197
minimum cardinality design, 220
of required children, 221
document processing, 487
and database processing, 487
document type declarations (XML), 488–489
external and internal, 489
DOM (Document Object Model), 520, 521
domain constraints, 197
domain key normal form (DK/NF), 120–121
domains
in IDEF1X, B-7–8
of semantic object attributes, E-6
double negative, NOT EXISTS form of subquery in, 300–301
drill down into data, 551
driver for ODBC, 431–432
Driver Manager (JDBC), 448
driver manager for ODBC, 432
driver types for JDBC, 446–447
DROP statement (SQL), 131
DROP TABLE statement (SQL), 252
DROP TRIGGER command (SQL), 10B-53
Drugstore.com, 7

Amazon Ex. 1008.0617

duplicated data, 107
  data integrity problems from, 119
  inconsistency from, 134
  reducing *vs.* eliminating, 129
duplicate rows, 98
durable transaction, 332
dynamic cursor, 334
dynamic reports, 455
dynamic Web pages. *See* Java Server Pages (JSP)

E

Eclipse IDE, 456–457
Eclipse PDT, 456–457
e-commerce applications, 7
enterprise-class database system components, 13–16
  database applications, 15
  DBMS (database management system), 16
enterprise data warehouse (EDW) architecture, 531–532
Enterprise Manager (SQL Server), 385
Enterprise Manager 11g Database Control, 10A-5
Enterprise Miner, 16, 552, 555
Enterprise Resource Planning (ERP), 6–7, 16, 17
entities, 145–146, E-2. *See also* semantic object model
  adjustments from report analysis, 176
  attributes, 146
  database table creation
    candidate (alternate) keys, 105
    column properties, 195–197
    primary key selection, 194
    verifying normalization, 107–198
  ID-dependent, relationships, using, 201–205
  in IDEF1X, B-1–2
  importance of weak, non-ID-dependent, 154–156
  relationships of, 147–149
  subtype, 156–157
  vs. table, 148
  in UML, C-1–2
  weak or strong, 153
entity class, 145–146, C-1
entity instance, 145–146
  identifiers, 146–147
entity-relationship data modeling, 18. *See also* IDEF1X (Integrated Definition 1, Extended)
  diagrams, 151
  extended, 145
  vs. IDEF1X standard, B-2
  vs. semantic object model, E-29–31
  variations, 151
entity-relationship (E-R) diagrams, 151
  using crow's foot model, 152–153
entity-relationship model, 145–157
  attributes, 146
  entities, 145–146
    ID-dependent, 153–154
    non-ID-dependent weak entities, 154–156
    strong and weak, 153
    subtype, 156–157
  E-R diagrams and their versions, 151
  identifiers, 146–147
  maximum cardinality, 149–150
  minimum cardinality, 150–151
  relationships, 147–149
  variations of, 151
enumerated list, for attribute domain, E-6
equations, and functional dependency, 100–101
equijoin, 75
E-R model. *See* entity-relationship data modeling
ERP (Enterprise Resource Planning), 6–7, 16, 17
error reporting and tracking system, 137
Errors collection (ADO), 440
ERwin
  alternate key (AK), 195
  free trial version, 153
Excel pivot table, 551
exclusive lock, 328
exclusive subtype entities, E-26
Exec command, 461
EXISTS form of subquery, 300
explicit locks, 328
extended E-R model, 145
Extensible Markup Language (XML). *See* XML (Extensible Markup Language)
Extensible Style Language: Transformations (XSLT). *See* XSLT (Extensible Style Language: Transformations)
external DTD, 489

F

fact table, 533
Fagin, R., 108
FETCH NEXT statement, 385, 388
field list, A-24
fields, 99, 439, 440, A-24. *See also* columns in database table
Fields collection (ADO), 440
fifth normal form (5NF), 108, 120
file data source for ODBC, 435
files, 99
firewall, 337–338
first normal form (1NF), 107, 108
flat files, D-1–8
  B-trees, D-5–7
  indexes for order, D-5
  linked lists for order, D-3–5
  processing in multiple orders, D-2
  record addressing, D-2–3
flat XML Schema, 496–497
  vs. structured, 496–498
for-each loop in XSLT, 490
foreign key, 5, 105–106, A-2. *See also* relationships
  checking for null values in, 137–138
  dropping column for, 307
  in intersection table, 200, 201
  testing referential integrity constraints, 128
FOREIGN KEY constraint, 243
  and dropping tables, 312
  in SQL CREATE TABLE, 239
ForeignKeyConstraint objects, 443
formatting, 68
forms
  grayed out sections, and subtype, 171
  inferring data model from, 157–174
  multitable, in Access, A-26–27
for-use-by pattern, 171
forward engineering, 312–313

Amazon Ex. 1008.0618

forward only cursor, 334, 335, 441
fourth normal form (4NF), 108, 120
FoxPro, 23
FROM keyword in SELECT statement (SQL), 36
  using view in, 266–274
full recovery, 416
functional dependencies, 100–104, 129
  anomalies, eliminating from, 110–117
    multivalued dependencies, 117–120
  boundaries of form and, 197
  composite, 101
  correlated subqueries to check, 77, 297–301
  and equations, 100–101
  finding
    in ORDER_ITEM table, 103–104
    in SKU_DATA table, 101–103
  importance of identifying, 104
  and normal forms 2NF through BCNF, 108–120
functions in SQL
  layering built-in functions, 270–272
  limitations, 61

G
gap locks, 10B-54
Gates, Bill, on XML, 487
generic data types, 196
generic entity, B-4
GETDATE() function, 283, 389
global elements, in XML Schema, 500–501
GoldMine, 6, 7
GRANT statement (SQL), 10B-56
graphical tools, *vs.* SQL code, for table creation, 238
graphical user interface (GUI), 32
  displays, 357, 364–368
    MySQL, 10B-8–10, 10B-16–17
    Oracle, 10A-23–26, 48
group attributes of semantic objects, E-3
GROUP BY clause (SQL), 64
  as check on inconsistent values, 137
  in joins, 75
  in subqueries, 72
  and WHERE clause placement, 65
group identifier, of semantic objects, E-5
group permissions, 337, 338
growing phase for transactions, 329
guest user accounts, 338

H
HAS-A relationships, 150
  design transformation process for, 207
  UML representation, C-2
HAVING clause (SQL), 66
heads, D-16
Highline University data model, 213–214
history of database processing, 20–24
HTML Web pages, 453–456
HTTP, 463
HttpServlet class, 449
hybrid objects, in semantic object model, E-17–20
hybrid relationships, in semantic object model, E-20–21

I
IAccessor interface, 441
IBM Corporation, 11, 16, 22, 35
IBM DB2, 22
IColumnsInfo interface, 441
ID-dependent entities, 153
ID-dependent relationships, 162–168
  archetype/instance pattern, 166–168
  association pattern, 162–163
  multivalued attribute pattern, 164–166
IDEF1X (Integrated Definition 1, Extended), 151, B-1–8
  categorization relationships, B-4–6
  domains, B-6–8
    base and typed, B-7–8
    usefulness, B-7
  entities, B-1–2
  relationships, B-2–4
    identifying connection, B-3–4
    nonidentifying connection, B-2–3
    nonspecific, B-4
identifiers, 146–147
identifying connection relationships in IDEF1X, B-3–4
identifying patterns, mixed and nonidentifying, 168–170
  line-item pattern, 168–170
  other mixed patterns, 170
identifying relationship, 153–154
IDENTITY data type, in SQL Server, 240
@@Identity function (SQL Server), 387, 388
IDMS, 21–22
Image data type, 341–343
implementation in OLE DB, 439
implicit locks, 328
IMS, 21, 22
inclusive subtype entities, 157
inconsistent read problem, 327
indexes
  for flat file order, D-5
  in MySQL, 10B-17–22
  in Oracle, 10A-26–28
  representing trees, D-12
  in SQL Server, 368–369
index.html, 453–456
inetpub folder, 451
Information Engineering (IE) model, 151
information, *vs.* data, 5, 6
inheritance, for parent/subtype objects, E-25
in-memory database, importance, 444
inner joins, 264
IN operator (SQL), 55
INSERT command (SQL), 130
  with column names, 252–253
  for normalization, 130–131
InsertCommand object, 444
insertion anomaly, 107
Insightful Corporation, I-Miner, 16, 552, 555
installing SQL Server, 352–353
instances, 3
  database table rows for, 3
  of object class, 443
  of semantic objects, E-4

Amazon Ex. 1008.0619

INSTEAD OF trigger, 274, 275–276, 279–280
in SQL Server, 393
for updating view, 401, 404
integer data type, 240, 241
Integrated Definition 1, Extended (IDEF1X model). *See* IDEF1X (Integrated Definition 1, Extended)
Integrated Development Environment (IDE), 10A-5, 356, 456–457
integrated tables, 12
integration of databases, 19
Intel, 22
interface. *See also* graphical user interface (GUI)
IAccessor interface, 441
IColumnsInfo interface, 441
IRowSet, 441
Microsoft Office Fluent user interface, A-5–7
object, 440
object-oriented interface, OLE DB as, 430
of rowset in OLE DB, 441
internal DTD, 487
International Organization for Standardization (ISO), 35
Internet, development, 23
Internet Information Server (IIS), 15
ASP pages, 451–453
Internet Information Services Manager, 451
interrelation constraint, 197
intersection record, D-13
intersection table, 200, 230
vs. association tables, 201–203
intersection tables, *vs.* association tables, 201–203
intrarelation constraint, 197
inverted lists, D-5
IRowSet interface, 441
IS-A attribute, E-26
IS-A relationships, 157, 208, B-4
IS NULL (SQL), 138
isolation level of transactions, 333
isolation of data tables, views for, 272

J

Jakarta project, 449
Java, 15, 430. *See also* JDBC
and Oracle, 10A-5
stored procedures written in, 281
for triggers, 275
for XML in Oracle, 446, 502
Java beans, 478
Java Development Kit (JDK), 10A-5
Java Enterprise Edition (Java EE), 456
Java Runtime Environment (JRE), 10A-5
JavaScript, 456
Java Server Pages (JSP), 430
Apache Tomcat, 449
setup for JSP processing, 449–450
compilation process, 450
Java code, 447
and servlets, 448–449
Java virtual machines, 447
JDBC
driver types, 446–447
establishing connection to database, 447
loading driver, 447–448
statements
creating, 502
prepared and callable, 448
using, 447–448
Jet DBMS engine, 14
joining the two tables, 75
join operation, 75
joins, 73–77, 260–269
JOIN ON, 260–261
outer joins, 262–266
vs. subqueries, 76–77
views to hide, 268–269
XML SELECT statement for, 506
JSP. *See* Java Server Pages (JSP)

K

keys
for access tables, A-2
defining, A-10
and identifiers, 147
in relational model, 104–106
keyset cursors, 334
keywords in SQL Server, 383

L

LAMP, 430
leading edge jobs in technology, 19
left outer joins, 263, 264
legacy systems, adapting, 19
letters (correspondence), in data modeling process, 178–180
licensed software, service packs, 337
lift in marketing, 556
LIKE keyword, 57
line-item pattern, 168–170
linked lists, D-3
for flat file order, D-3–5
order maintained with, B-3–5
representation of secondary keys, D-16–17
representing simple networks, D-8–9
representing tress, D-10–11
two-way linked list, B-4
Linux, 24, 430
lock command, 328
lock granularity, 328
locking hints, 409
locking resources, 328
declaring characteristics, 331
optimistic *vs.* pessimistic, 330–331
locks in SQL Server, 409
locks
in MySQL, 10B-54
in Oracle, 10A-63
in SQL Server, 409
logical units of work (LUWs), 324
login, with SQL*Plus, 411–412
logistic regression, 555
logs
of before images and after images, 341
in SQL Server, placing labeled mark in, 414
of transaction results, 341
lost update problem, 324–328

Amazon Ex. 1008.0620

M
machine-independent bytecode, 447
magnetic tape, 20–21
management of reports, 547–549
mandatory relationship, 150
mandatory-to-mandatory (M-M) relationship, 150, 214, 215
  enforcing, 220
  implementing actions for, 219
    summary, 222
  special case M-M relationships, designing, 219–220
mandatory-to-optional (M-O) relationship, 214, 215
  implementing actions for, 218
    summary, 222
many-to-many (N:M) relationship, 160–162
  changing 1:N to, 310–311
  in IDEF1X, B-2, B-4
  recursive, 210
  in semantic object model, E-14–16
many-to-one relationships, in semantic object model, E-14
market basket analysis, 530, 555–557
  SQL used for, 556–557
Martin, James, 151
MAX function, 59, 61
maximum cardinality, 149–150
  in database redesign, 309–311
  of semantic object attributes, E-4
measure in OLAP, 549
memory in early computers, 20
metadata, 12
  about database, 344–345, 494
  data warehouse database of, 529
  example tables, 12
  on reporting system, 545, 548
meta-metadata, 494
methods of objects, 339
Microsoft, 428–429
Microsoft Access, 11
  closing and exiting, A-7–8, A-27
  components of, 14
  connecting to SQL Server, 417
  under the cover, 14, 15
  database creation, A-3–5
    tables, A-9–17
  database design, A-1–4
    data types for columns, A-3–4
    relationships, A-2
    table keys, A-2
  data entry in tables
    with Datasheet View, A-17–19
    with your own form, A-20–22
  data type for currency, 196
  first release, 23
  limitations, 40–41
  Microsoft Office Fluent user interface, A-5–7
  multiple forms and reports, A-26–27
    Create Form using Wizard, A-26–27
    Create Report using Wizard, A-27
  opening existing Access database, A-8–9
  other features, A-26–27
  queries, 40, 41, A-23–26
  relationships, A-2
  saved queries, 266
  saving SQL statement, 45
  SQL hidden by, 36
  SQL statement submission to, 40–45
  surrogate key support, 194
  viewing SQL statement in, A-17–19
  what it is, 13–14
  wildcards, 57
Microsoft .NET Framework, 437–446
  ADO, 442
  ADO.NET, 442–446
  OLE DB, 438–441
Microsoft .NET languages, 15, 446, 449
Microsoft Office Fluent user interface, A-5–7
  Access Navigation Pane, A-6–7
  command tabs, A-5
  contextual command tabs, A-5–6
  Ribbon, A-5
Microsoft Transaction Manager (MTS), 441
Microsoft Windows PowerShell, 356
migration of database, 18. *See also* database redesign
MIN function, 59, 61
minimum cardinality, 149, 150–152
  in database redesign, 308–309
  data design for, 214–222
    documenting, 220–222
    for mandatory-to-mandatory relationships, 219–220
    for mandatory-to-optional relationships, 218
    for optional-to-mandatory relationships, 219
    summary, 222
    when child is required, 217
    when parent is required, 216–217
  enforcement action, 220
  of relationships, 308–309
  of semantic object attributes, E-4
misspellings, problems from, 137
mixed entity relationships, 206–207
mixed patterns in data model, 168–171
M-M relationship. *See* mandatory-to-mandatory (M-M) relationship
modification anomalies, 106–107
Money data type, 196
Moore, Gordon, 22
M-O relationship. *See* mandatory-to-optional (M-O) relationship
moving databases between computers with backup and restore, 416
multiple conditions in WHERE clause, 55–56
multiple-tier driver for ODBC, 432
multitable forms and reports in Access, A-26–27
multiuser database applications, 6–7
  challenges, 304
multivalued attributes, 204
  in data model, 164–166
  in ID-dependent entities, 204
  in semantic object model, E-7
multivalued dependency, 96, 108
  anomalies from, 117–120
  eliminating anomalies, 117–120
multivalue paths, schema with two, 444, 514
MUST constraint, 211
MUST COVER constraint, 211
MUST NOT constraint, 211

Amazon Ex. 1008.0621

MySQL, 16, 50–52
Apache Web server, 430
case sensitivity in, 243
data types, 239, 249
history of, 23–24
JDBC driver for, 446, 447
as open-source database, 352
passwords, 461
PHP support, 450, 463
security, 337, 339
surrogate key support, 239
triggers, 275, 276
view support, 266
Workbench, 153, 196, 201
MySQL, 10B-1–61, 16, 50–52
Apache Web server, 430
application logic, 10B-33–55
procedural statements, 10B-34–36
stored procedures, 10B-36–43, 10B-53
triggers, 10B-43–54
backup, 10B-58–59
case sensitivity in, 243
concurrency controls, 10B-54–55
configuring, 10B-2–3
database creation, 10B-6–7
database storage engines, 10B-3–4
data types, 239, 249
GUI Tools, 10B-4–6
history of, 23–24
installing, 10B-2
JDBC driver for, 446, 447
as open-source database, 352
passwords, 461
PHP support, 450, 463
recovery, 10B-59–61
security, 10B-55–57, 337, 339
surrogate key support, 10B-13, 239
triggers, 275, 276
utilities, 10B-7–13
command-line utility, 10B-8
GUI displays, 10B-8–10, 10B-16–17
statements and scripts, 10B-10–13
View Ridge database tables, 10B-13–33
view support, 266
Workbench, 153, 196, 201
MySQL BOOLEAN data type, 10B-47
MySQL Connector/J, 446, 447
MySQL, 5.1
managing databases with, 425
using SQL in, 50–52
MySQL Workbench, 10B-4

N

names
for constraints, 244
for tables, changing, 305–306
namespaces for XML, 494–495
namespaces
for Oracle, 10A-6
for XML, 494–495
National Institute of Standards and Technology, 151
nested subqueries, 299
NetBeans, 10A-5, 478
networks (data structures), 22
neural networks, 555
New Database Diagram (SQL), 366
next-key locks, 10B-54
N:M relationship. *See* many-to-many (N:M) relationship
nonclustered indexes in SQL Server, 368–369
nonidentifying patterns, 154
mixing identifying and, 168–171
nonidentifying relationships, 154
in IDEF1X, B-2–3
nonintegrated data, 530
nonmatching rows, inner joins and, 265
nonobject attribute, E-7
nonrepeatable reads, 333
nonspecific relationships in IDEF1X, B-4
nonunique secondary keys, D-16, D-17–18
normal forms, 17, 106–121
eliminating anomalies
from functional dependencies, 110–117
from multivalued dependencies, 117–120
history, 107–108
modification anomalies, 106–107
normalization categories, 108
normalization, 17
advantages and disadvantages, 129–130
with SQL (Structured Query Language), 130–131
verifying, 197–198
normalized table, 115
Notepad, 453, 455, 456
NOT EXISTS form of subquery, 300–301
in double negative, 300–301
NOT IN operator (SQL), 55–56
NOT NULL columns, adding, 307, 453
not-type-valid XML document, 489
NULL/NOT NULL constraint in SQL CREATE TABLE, 247
null status, as column property, 195
null value, 137–138
number data in Access, 249
NUMBER (n, d) data type, 240
numeric columns, comparison value for, 39
Numeric data type, 196

O

object, 439
object class, 439
object class name, 439, E-2
object identifier, E-5
object interface, 440
Object Linking and Embedding (OLE), 439
object links, E-3
object-oriented data processing, E-1
object-oriented DBMS (OODBMS or ODBMS), 23
object-oriented interface, OLE DB as, 430
object-oriented programming (OOP), 23
objects, 439
object-oriented structures, UML representations, C-3–4
object-relational databases, 23
object-relational DBMS, 23
object types in semantic object model
archetype/version objects, E-26–29
association objects, E-22–24

Amazon Ex. 1008.0622

composite objects, E-8–11
compound objects, E-11–12
hybrid objects, E-17–20
parent/subtype objects, E-25–26
representing hybrid relationships, E-20–21
representing many-to-many relationships, E-14–16
representing one-to-many and many-to-one relationships, E-14
representing one-to-one compound objects, E-12–14
simple objects, E-7–8
ODBC (Open Database Connectivity), 431–437
architecture, 431–432
conformance levels, 432–434
data source name, 434–437
drivers, 431–432
purpose, 431
ODBC Data Source Administrator Service, 435
OLAP (OnLine Analytical Processing), 528, 545, 549–552
OLAP cube, 549
OLAP reports, 549
OLAP servers, 551
OLE DB, 438–441
goals, 440–441
implementation in, 439
terminology, 441
OLE DB data provider, 440–441, 443
O-M (optional-to-mandatory) relationship, 214
data design for, 219
summary, 222
one-to-many (1:N) relationship, 149
changing 1:1 to, 309–310
changing to many-to-many, 310–311
in IDEF1X, B-2
recursive, 209
in semantic object model, E-21
one-to-one (1:1) relationship, 149
changing to one-to-many, 309–310
for compound objects, E-12–14
in IDEF1X, B-2
implementing, 245
recursive, 208–209
OnLine Analytical Processing (OLAP), 528, 545, 549
online transaction processing (OLTP) systems, 526–527
O-O (optional-to-optional) relationship, 150, 214
Open Database Connectivity. *See* ODBC (Open Database Connectivity)
Open method of RecordSet object, 439, 440
open source movement, 552
operational systems
vs. business intelligence systems, 526–527
problems with, 528
optimistic locking
for datasets, 445
vs. pessimistic, 330–331
optimizing DBMS, 344
optional relationship, 150
optional-to-mandatory (O-M) relationships, 214
data design for, 219
summary, 222
optional-to-optional (O-O) relationship, 150
Oracle
data types, 239, 240, 249
embedding SQL statements, 274
PHP support, 450
PL/SQL, 239
security, 337, 339
SQL Developer GUI tool, 48
SQL*Plus command line tool, 48
stored procedures written in, 281
for stored procedures, 281
surrogate key support, 239
triggers, 275, 276
user names for, 336
view support, 266
XML in, Java for, 446, 502
Oracle
administration and development tools, 10A-5–7
application development tools, 10A-15–18
application logic, 10A-39–61
backup and recovery, 10A-64–65
command-line tool, 10A-15–16, 48
concurrency control, 10A-61–63
data types, 239, 240, 249
Developer GUI tool (SQL), 10A-23–26, 48
documentation, 10A-4
embedding SQL statements, 274
installing, 5, 10A-2–4
and Java, 10A-5
log in/out, 10A-7
other features, 10A-65
PHP support, 450
PL/SQL, 239
rollback segment, 10A-61
security, 10A-11–15, 337, 339
Roles creation, 10A-13–15
user accounts, 10A-11–13
user privileges, 10A-11
SQL Developer GUI tool, 10A-23–26, 48
statements and scripts (SQL), 10A-18–19
for stored procedures, 10A-42–49, 281
surrogate key support, 239
System Change Number (SCN), 10A-61–62
tablespaces, 10A-8–10
triggers, 10A-50–61, 275, 276
for enforcing child constraint, 10A-56–61
for enforcing data constraint, 10A-52–54
exception handling, 10A-61
for setting default values, 10A-50–52
tablespaces, 10A-10
for updating views, 10A-54–56
user names for, 336
using SQL in, 48–50
View Ridge database table creation, 10A-6, 10A-20–39
view support, 266
XML in, Java for, 446, 502
OracleClient data provider, 443
Oracle Database 2 Day + PHP Developer's Guide, 450
Oracle Enterprise Manager 11g Database Control (Database Control), 10A-6–7
Oracle Database 11g, using SQL in, 48–50
Oracle Recovery Manager (RMAN), 10A-65
Oracle SQL Developer, 10A-16–17
Oracle SYS system account, 10A-7
Oracle Universal Installer (OUI), 5, 10A-3–4

Amazon Ex. 1008.0623

ORDER BY clause (SQL), 53
in joins, 74–76
in SELECT statement (SQL), 53, 65
in subqueries, 72–73
with WHERE clause, 59
OR operator, 55
orphans, preventing, 216
overhead for cursor, 334

P
paired attributes of semantic objects, E-5
Paradox, 23
Parameters collection (ADO), 440
parameters in Transact-SQL, 383
parameters
in MySQL, 10B-34
in Transact-SQL, 383
parent in entity relationship, 149, 199
in association pattern, 163
data design for required, 216–217
documenting required, 220
identifier replacement with surrogate key, 215
parent rows, implementing required, 216
parent/subtype objects, in semantic object model, E-25–26
parent tables
changing minimum cardinalities, 308–309
dropping, 252
order of defining, 245
parsing XML documents, 520, 521
passive repositories, 345
passwords
for MySQL, 461
for users, 338
patches, to prevent worm attack, 339, 353
percent (%) wildcard character, 57
performance, run-time statistics to evaluate, 344
permissions, views for multiple, 272
personal database system, 13–14
pessimistic locking, *vs.* optimistic, 330–331
phantom reads, 333
phone numbers, 98, 136, 164–165
database design for, 136
PHP
code segments (//), 461
Web database processing with, 449–451, 456–463
creating a RecordSet, 461–462
creating database connection, 461
disconnecting from database, 462–463
displaying results, 462
Eclipse IDE, 456–457
PHP scripting language, 456
ReadArtist.php file, 457–460
Web page examples with, 463–477
invoking a stored procedure, 470–473
updating a table, 464–469
using PHP Data Objects (PDO), 470
PHP Development Tools Project, 456–457
physical existence, E-3
physical security for computers, 338
PivotTable in Microsoft Excel, 549
PL/SQL (Programming Language for SQL), 239
for stored procedures, 281
for triggers, 275
vs. T-SQL, 239, 283
pointers, D-16
post method in HTML, 464–465
PowerShell sqlps utility, 356
precompiled objects on JSP pages, 449
Prepared Statement objects (JDBC), 448
primary key, 5, 105
as composite, 104
in data design, 194
dropping column for, 307
and strong entity relationships, 198
switching, 302
PRIMARY KEY constraint in SQL CREATE TABLE, 239
private attributes in UML, C-4
procedural language, 360
Procedural Language/SQL (PL/SQL), 239
processing rights and responsibilities, 335–336
processing speed, table size and, 132
program code in SQL, 274–275
properties, 439
of objects, 430, 439–440
of OLE DB rowsets, 439
of relationships in Access, A-2
protected attributes in UML, C-4
pseudofiles, 275
pseudotables in SQL Server, 394
public attributes in UML, C-4
pull report, 546
punched cards, 243
punctuation for SQL statement. *See* symbols and numbers
push report, 546

Q
queries. *See also* SELECT statement (SQL)
ad-hoc, 31–32
calculations in (SQL), 59–63
in Microsoft Access, 40, 41, A-23–26
saved, 266
string manipulation in (SQL), 63
Query Analyzer, 418
query by example (QBE), 32
query reports, 545, 549
quotation marks, single ('), for WHERE comparison text, 39

R
Rand Corporation, 19
range constraints, 197
ranges in WHERE clause, 56
Ratliff, Wayne, 22
R:base, 21, 23
ReadArtist.php file, 457–460
read-committed isolation level, 333
read-only database design, 132–134
customized duplicated tables, 134
denormalization, 132–133
read-only isolation level, 408
read-only lock, 408, 409
record locks, 10B-54
read-uncommitted isolation level, 408

Amazon Ex. 1008.0624

records, 3. *See also* rows in database table
recordset in OLE DB, 439
RecordSet objects (ADO), 439
recovery data, 13
recovery of database, 340–343
  in MySQL, in 10B-59–61
  in Oracle, 10A-64–65
  in SQL Server, 414–416
  via reprocessing, 341
  via rollback/rollforward, 341–343
recursive patterns, 172–174
recursive relationships, 208–210
  many-to-many (N:M), 173–174, 210
  one-to-many (1:N), 172–173, 209
  one-to-one (1:1), 172, 208–209
ReDo files, 341, 342
referential integrity actions, 11
  defining, 218
  implementing with triggers, 279–280
referential integrity constraints, 11
  in Access, A-21–22
  creating in ADO.NET, 443
  defining, 243
  for foreign key value, 213
  as misspelling check, 137
  testing enforcement for foreign key, 128
regression analysis, 555
relational algebra, 22
relational database, 3
relational model, 22
  emergency and dominance, 22–23
  keys, 104–106
  terminology, 96–106
relations. *See also* tables in database
  characteristics, 97–98
  process for putting into BCNF, 109–110
  vs. tables, 98
  with variable-length column, 99
relationship cardinalities
  in database redesign
    maximum cardinality, 309–311
    minimum cardinality, 308–309
relationship classes, 147
relationship instances, 147
relationships, 5
  in Access, A-2
  adjustments from report analysis, 196
  casual, 245–246
  in database redesign, adding and deleting, 312
  in data design
    among strong entities, 198–201
    in Highline University data model, 213–214
    recursive relationships, 208–210
    subtype relationships, 208
    ternary and higher-order relationships, 210–212
    using ID-dependent entities, 162–168
  of entities, 147–149
  in IDEF1X, B-2–4
  IS-A, 157, 208, B-4
  mixed entity, 206–207
  multiple between tables, 222
  names, 147
  recursive, 208–210
    many-to-many (N:M), 173–174, 210
    one-to-many (1:N), 172–173, 209
    one-to-one (1:1), 172, 208–209
  subtype, 208
  summary of definitions using CREATE TABLE, 246
  summary of representations, D-14
  in UML, C-1–3
  viewing properties in SQL Server, 411
relative record number (RRN), D-2
remarks column in existing data, 138–139
repeatable-read isolation level, 333
replication, 417
report authoring, 546–547
report delivery, 549
reporting applications, 15
reporting systems, 7–8, 527, 528, 539–552
  components, 544–545
  functions, 546–549
  OLAP (OnLine Analytical Processing), 549–552
  producing RFM report, 541–544
  report media, 545–546
  report mode, 546
  report types, 545
  RFM analysis, 540–541
reports
  analysis in data modeling process, 175–178
  for DBMS management, 344
  inferring data model from, 157–174
  multitable, in Access, A-23
reserved words, 10A-20, 223
resource locking, 328–332
  declaring characteristics, 331–332
  optimistic *vs.* pessimistic, 330–331
responsibilities, 335–336
results table for SQL statement
  column order, 53
  DBMS functions for formatting, 68
  removing duplicate rows, 37–38
  sorting, 53–54
reverse engineering, 313–314
RFM analysis, 540–544
right outer joins, 263–264
rights and responsibilities, processing, 335–336
ROLE component, 337, 413
roles, 338
  security on, 412
Roles creation (Oracle), 10A-13–15
ROLLBACK command, and lock release, 329
rollback/rollforward, for recovery of database, 341–343
rollback segment, 340
Rollback Transaction command, 324, 331
rowCount function (@@), 394, 396
rowset in OLE DB, 439
  interfaces, 441
row share lock, 409
rows in database table, 3
  duplicate, 98
  nonmatching, inner joins and, 265
  removing duplicate from SQL results, 37–38
  searching for multiple with given entry, 297–298
  selecting, with specific columns, 40

Amazon Ex. 1008.0625

rows in database table (*continued*)
SQL for reading specific, 38–40
UPDATE to change values of existing, 259–260
views to hide, 268–269
run-time statistics, 344

S
SAS, Enterprise Miner, 16, 529, 552, 555
saving SQL statement in Access, 45
SAX (Simple API for XML), 520, 521
schemas. *See also* XML Schema
MySQL, 10B-5, 10B-7
Oracle, 10A-17–18
script comments (SQL), 10A-18
scrollable cursors, 334
SCROLL_LOCK, 408
secondary-key representations, D-16–18
index, D-17–18
linked lists, D-16–17
second normal form (2NF), 107, 108–109
security, 335–340
application security, 339–340
DBMS, 337–339
processing rights and responsibilities, 335–336
SQL injection attack, 340
in SQL Server, 410–412
on database accounts, 412–414
security data, 13, 339–340
SELECT clause, for explicit locks, 407–408
SelectCommand object, 444
SELECT statement (SQL)
calculations, 59–64
arithmetic, 62–63
built-in functions, 59–62
string manipulation, 63–64
to define cursor, 334
DISTINCT keyword, 37–38
formatting, 68
GROUP BY clause, 64–67
HAVING clause, 66
FOR XML, 494–502 (*See also* XML Schema)
EXPLICIT keyword, 504, 507
multitable, 506–510
RAW keyword, 502–503
FROM keyword, 36
WITH parameter for locking, 409
using view in, 266–274 (*See also* SQL views)
multiple tables, 70–76
joins, 73–76
with subqueries, 70–73
ORDER BY clause, 53–54
with WHERE clause, 59
to read columns from single table, 36–38
to read specific rows and columns from single table, 38–40
sorting results, 53–54
TOP keyword, 128
WHERE clause, 55–59
with built-in functions, 59–60
compound clauses, 55–56
and GROUP BY placement, 64
ranges, 56–57
in SQL joins, 73–76
in view definition, 268
wildcards, 57–58
semantic object attributes, E-3
semantic object diagram, E-3
semantic object model, E-1–31
vs. E-R model, E-29–31
object types, E-7–29
archetype/version objects, E-26–29
association objects, E-22–24
composite objects, E-8–11
compound objects, E-11–14
hybrid objects, E-17–20
parent/subtype objects, E-24–26
representing hybrid relationships, E-20–21
representing many-to-many relationships, E-14–16
representing one-to-many and many-to-one relationships, E-14
simple objects, E-7–8
semantic objects, E-2–7
attribute domains, E-6
attributes, E-3–5
defining, E-2–3
object identifiers, E-5–6
views, E-6–7
semicolon (;)
and CREATE VIEW statement, 266
for SQL statement termination, 37
sequence (Oracle), 10A-20
sequence set in B-Tree, D-5
sequential list, D-2
SERIALIZABLE isolation level, 409
serializable isolation level, 10A-63, 333
serializable transactions, 329
server principles, 411
service provider, 441
servlets, 447
and Java Server Pages (JSP), 448–449
SET TRANSACTION ISOLATION LEVEL statement, 407–408
SGML (Standard Generalized Markup Language), 488
shared lock, 328
SHOW TRIGGERS command (SQL), 10B-53
shrinking phase, 329
shrinking phase for transactions, 329
simple attributes of semantic objects, E-3
simple network, D-8–9
binary relationships representing, D-12–13
Simple Object Access Protocol (SOAP), 487
simple objects in semantic object model, E-7–8
simple recovery model, 415
single-tier driver for ODBC, 432–433
single-user database applications, 6
single-value attribute, E-7
SKU (stock keeping unit), 33
Slammer worm, 339
slash (/) for tags, 453
Smalldatetime data type, 240
Smallint data type, 240, 241
snowflake schema, 533
SOAP (Simple Object Access Protocol), 487
software development kit (SDK), 10A-5

Amazon Ex. 1008.0626

SPSS, Clementine, 16, 552, 555
SQL (Structured Query Language), 9–11
background, 35–36
case in, 243
coding *vs.* graphical tools, for table creation, 238
as data sublanguage, 36
for market basket analysis, 556–557
normalization with, 130–131
in program code, 274–275
to query metadata, 13
statement submission to DBMS, 40–52
statement termination with semicolon, 37
using in MySQL 5.1, 50–52
using in Oracle Database 11g, 48–50
using in SQL Server 2008, 46–48
viewing statement in Access, 40–45
SQL Access Group committee, 431
SQL Client data provider, 443
SQL CMD utility, 356
SQL Common Language Runtime (CLR), 356–357
SQL conformance level, 434
.sql files, 49
SQL injection attack, 340
SQL-92 standard, 37
statement for database creation, 238
SQL outer join, 262
SQL*Plus, 48, 281
command-line tool, 48
running stored procedures on, 281
SQL*Plus in Oracle, 48–49, 281
SQL script comments, 359
SQL SELECT . . . FOR XML statement, 502–510
SQL Server. *See also* SQL Server 2008
Analysis Services, 353, 550, 553
CREATE VIEW statement, 266
graphical displays, 303
IDENTITY data type in, 240
indexes, 368–369
replacing Jet with, 14
Slammer worm and, 339
SQL statement submission to, 46–48
surrogate key support, 194
viewing relationship properties, 366
XML for, 368
SQL Server 2005, 410
SQL Server 2008
application logic, 381–407
stored procedures, 382
Transact-SQL, 382–385
triggers, 393–407
backup and recovery, 414–416
concurrency control, 407–409
cursor concurrency, 408–409
transaction isolation levels, 333, 408
database creation, 354–356
installing, 352–353
Management Studio, 253–254
other important topics, 417
security, 410–414
utilities
GUI displays, 357, 364–368
Microsoft SQL CLR, 356–357
SQL CMD and Microsoft PowerShell, 356
SQL statements and scripts, 357–360
View Ridge database tables, creating and populating, 360–381
SQL Server 2008 Express, 352
SQL Server 2008 Management Studio, 353–354
SQL Server ODBC driver, table name for, 434–435
SQL Server schema, 411
SQL Server securable objects, 411
SQL views, 266–276
to display computed column results, 269–270
to hide columns and rows, 268–269
to hide SQL syntax, 270
layering built-in functions, 270–272
updating, 273–274
with triggers, 275–276
uses for, 367
SQLXML (class library), 502
square brackets ([,]) to use reserved words, 248
Standard Generalized Markup Language (SGML), 487
Star schema, 533–537
Start Transaction command, 324
stateless protocol, 23
statement-level consistency, 332–333
Statement object (JDBC), 448
statements (SQL)
MySQL, 10B-10–13
procedural, 10B-34–36
Oracle, 10A-18–19
static cursor, 334
static reports, 545
statistics, collection and reporting, 343–344
stored procedures, 13, 281–283
ADD_WORK, 283
ADO to invoke, 448
advantages, 281, 283
displaying output from, 385
in MySQL
InsertCustomerAndInterests, 10B-36–38
InsertCustomerWithTransaction, 10B-38–43
in Oracle, 10A-42–49
for enforcing child constraint, 10A-56–61
for enforcing data constraint, 10A-52–54
and security, 337, 338
in SQL Server, 384–393
InsertCustomerAndInterests, 385–388
InsertCustomerWithTransaction, 389–393
vs. triggers, 282
string functions, stripping trailing spaces, 63
string manipulation in SQL queries, 63
strong entities, 153
cascading deletions and, 216
inferring from forms and reports, 156
many-to-many (N:M) relationship, 160–162
one-to-many (1:N) relationship, 160
one-to-one (1:1) relationship, 158–159
recognizing, 160
relationships among
many-to-many (N:M), 199–201
one-to-many (1:N), 199
one-to-one (1:1), 198–199
strong passwords, 338

Amazon Ex. 1008.0627

structured XML Schema, 498
vs. flat, 496–498
stylesheet for XML documents, 489–490
subqueries, 71
correlated, 298
vs. regular, 299
vs. joins, 76–77
limitations, 73
subtype entities, 156–157
grayed out sections of form and, 171
subtype relationships, 208
UML representation, C-2–3
sufficient description, E-2
SUM function, 59, 61
supertype entities, 156
supervised data mining applications, 555
surrogate key, 105
and database redesign, 225
enforcement actions for update, 217
in MySQL, 10B-13
parent identifier replacement with, 206
policy on, 194
in View Ridge Gallery database, 225
synchronization of logs and database, 343
syntax-error messages, reserved words and, 248
syntax, specifying column in table, 74
System Change Number (SCN), 10A-61–62
system data source for ODBC, 435

T

tables in database
ADO to read, 430, 437–446
in generalized way, 445
table selection by user for, 443
ADO to update, 444
CREATE TABLE statement (SQL), 238, 239
creating for entities
candidate (alternate) keys, 195
column properties, 195
primary key selection, 194
verifying normalization, 197–198
customized duplicated, 134
vs. entities, 148
multiple relationships, 222
name change in database redesign, 305–306
naming, 3
removing, 252
structure assessment, 128
tablespaces (Oracle), 10A-8–10
tabular data providers in OLE DB, 441
technology, leading edge jobs, 19
ternary relationships, 148, 210–212
test databases, 304–305
Text data type, 240
text editor for SQL*Plus, 39
third normal form (3NF), 107, 109
three-tier architecture, 430
Tinyint data type, 240, 241
TOP keyword (SQL), 128
transactional systems, 526–527
transaction-level consistency, 333
transaction log, backups, 414
transactions
concurrent processing, 324, 327, 328
consistent, 332–333
cursor type, 334–335
isolation levels, 333
in MySQL, 10B-54–55
in Oracle, 10A-63
in SQL Server, 408
serializable, 329
Transact-SQL (T-SQL), 239, 360
cursor statements, 10B-35–36, 384–385
output statements, 10B-36, 385
parameters and variables in, 383
vs. PL/SQL, 239, 283
for triggers, 283
tree relationship, D-8
triggers, 13, 275–281
in database redesign, 275–281
on DeleteEmployee, 279–280
to enforce minimum cardinality, 193, 194
to enforce required children, 219, 220
implementing referential integrity actions, 279–280
in MySQL, 10B-43–54
for enforcing child constraint, 10B-52–53
enforcing data constraint, 10B-46–49
for setting default values, 10B-44–46
SHOW TRIGGERS command, 10B-53–54
for updating a view, 10B-49–51
in Oracle, 10A-50–61
for enforcing child constraint, 10A-56–61
for enforcing data constraint, 10A-52–54
exception handling, 10A-61
for setting default values, 10A-50–52
tablespaces, 10A-10
for updating views, 10A-54–56
for SQL Server, 393–407
to enforce data constraints, 278–279, 309
to set default values, 276–277
to update views, 275–276, 279
vs. stored procedures, 282
views for multiple, 272
tuning DBMS system, 344
tuples, 98
two-phased locking, 329
two-tier architecture, 430
two-way linked list, D-4
typed domains, B-7
type-valid XML document, 489

U

UDDI (Universal Description, Discovery and Integration), 24
uncommitted changes in datafiles, 408
underscore character (_), as wildcard, 58–59
Unified Modeling Language (UML), 151, C-1–4
current role in database processing, C-4
entities and relationships, C-1–2
OOP constructs, C-3–4
representation of subtypes, C-2–3
representation of weak entities, C-2
UNIQUE constraint, 225, 239, 243, 246
unique secondary key, D-16
unique values, 102, 104, 105

Amazon Ex. 1008.0628

Universal Description, Discovery and Integration (UDDI), 24
unsupervised data mining applications, 555
updatable databases, design, 129–132
update anomaly, 107
UPDATE CASCADE, 245
UPDATE command (SQL), 259
  using values from other tables, 259–260
UpdateCommand property, of data adapter, 444
UPDATE NO ACTION, 245
user data source for ODBC, 435
user forms and reports, inferring data model from, 157–174
user groups, 336
user names for Oracle, 336
users, 8
  auditing accounts, 338
  passwords, 338
  skill requirements, 322
USER_ tables, 12

V
validation
  of data model, forms and reports for, 157
  of XML Schema, 494–495
VarChar data type, 240
VARCHAR(n) data type, 240
variables
  assigning SQL statement results to, 274–275
  in MySQL statements, 10B-34
  in Transact-SQL, 383
VBScript, 430, 442
version-control software, 323
View Ridge Gallery database, 222
  creating, 248–251
    in SQL Server, 248–251, 360–364
  database design, 224–225
  data model, 223–224
  minimum cardinality enforcement, 220
    for required child, 227–228
    for required parents, 225–227
  requirements summary, 222–223
  table creation
    ARTIST table, 240–244
    WORK table, 24–245
views. *See also* SQL views
  in MySQL, 10B-23–33
  in Oracle, 10A-36–39
  for semantic objects, E-6–7
  in SQL Server, 401
virtual directory for ASP pages, 446
Visio
  data modeling template, 151
  and many-to-many relationships, 201
Visual Basic .NET, 356–357, 442
Visual FoxPro, 23
Visual Studio .NET, 546

W
WAMP, 430
weak child entities, cascading deletions and, 216
weak entities, 154–156
  UML representation, C-2
Web activity databases, 7
Web browsers, support for XSLT, 489–493
Web developers, and database practitioners, 486
Web resources
  for Apache Tomcat, 449
  for JDBC, 447–448
  on standards updates, 487
  for textbook, 40
    EQUIPMENT_REPAIR table, 106–107
    NDX table, 67–70
  on XMLSpy, 496–497, 502
  on XSLT, 489–494
Web server, data environment, 429–430
Web Services Description Language (WSDL), 24
WHERE clause (SQL), 36, 38–40
  with built-in functions, 59
    limitations, 61
  compound clauses, 55–56
  and GROUP BY placement, 65
  ranges, 56–57
  in SQL joins, 73
  in view definition, 268–269
  wildcards, 57–59
World Wide Web Consortium (W3C), 453
WSDL (Web Services Description Language), 21, 24

X
Xbase file managers, 432
XHTML, 453
XML (Extensible Markup Language), 36
  additional standards, 520–522
  document type declarations, 488–489
  importance, 487, 514–520
  industry standards, 519
  integration with database processing, 19
  as markup language, 488
  XSLT for materializing documents, 489–494
XML documents, creating from database data, 501–513
  using multiple SELECT with FOR XML, 506–509
  using schema for ALL CUSTOMER purchases, 509–513
  using schema with two multivalue paths, 513
  using SQL SELECT . . . FOR XML statement, 501–506
XML namespaces, 493–494
XML Schema, 494–501
  for all customer purchases, 510–514
  elements and attributes, 495–496
  flat *vs.* structured schemas, 496–499
  global elements, 500–501
  for multivalue paths, 514
  validation, 494–495
XMLSpy, 496–497, 502
XML Web services, 15, 546
X/Open committee, 431
XPath, 520, 521
XPointer standard, 520, 521
XQuery, 521
XSL, 493, 514, 515, 519, 520
XSLT (Extensible Style Language: Transformations)
  materializing XML documents with, 489–494
  processors in Web browsers, 494

Amazon Ex. 1008.0629



ISBN-13: 978-0-13-230267-8
ISBN-10: 0-13-230267-5



Prentice Hall
is an imprint of

PEARSON

www.pearsonhighered.com

Amazon Ex. 1008.0630