

CHAPTER 12

A Complete Accounting System for a Merchandising Company

CHAPTER OBJECTIVES

This chapter will describe how to generate financial statements in a relational-database accounting information system (AIS) such as Microsoft Access. Chapters 8 through 11 presented queries for financial-statement line items that used data from individual business processes. These line items include sales, accounts receivable, accounts payable, wages expense, and loans payable. In this chapter, you will learn how to create queries for financial-statement line items that span multiple business processes such as cash, cost of goods sold, and retained earnings. We will also discuss how to capture data to enable the system to report items such as fixed assets, depreciation expense, and support activities such as rent, advertising, and insurance. Specifically, in this chapter, you will learn how to use Microsoft Access to design tables, queries, macros, and reports that can help you:

- Create tables to store data from non-inventory purchases and queries to compute financial statement balances related to this data.
- Create queries to compute income statement and balance sheet items related to end-of-period adjustments.
- Create queries for financial statement items that span multiple business processes.
- Create income statement and balance sheet reports.
- Create macros to automate the financial statement creation process.

Introduction

One of the main functions of an accounting system, whether manual or computerized, flat-file or relational, is to produce general-purpose financial statements for internal and external users. As you learned in Chapters 8 through 11, account balances are not stored in a relational database—rather, they are derived from the raw data stored in the system. Although many account balances on Pipefitters Supply Company’s income and statement balance sheet can be computed from data within a single business process, other items require data from multiple business processes. Figures 12.1 and 12.2 show an income

Fig. 12.1 Business processes used to derive Pipefitters’ current income statement items.

Pipefitters Supply Company	
Income Statement	
For the Period between 1/1/2010 and 3/31/2010	
Sales	Sales/collection process
Cost of Goods Sold	Sales/collection process & Acquisition/payment process
Gross Profit	Sales – Cost of Goods Sold
<u>Operating Expenses</u>	
Wages	Human resources process
Payroll Taxes	Human resources process
Total Operating Expenses	Sum of Operating Expenses
Operating Income:	Gross Profit – Total Operating Expenses
Interest Expense	Financing
Net Income	Operating Income – Interest Expense

Fig. 12.2 Business processes used to derive Pipefitters’ current balance sheet items.

Pipefitters Supply Company	
Balance Sheet	
3/31/2010	
<u>Assets</u>	
Cash	All business processes
Inventory	Sales/collection process & Acquisition/payment process
Accounts Receivable	Sales/collection process
Total Assets	Sum of all Asset balances
<u>Liabilities and Equity</u>	
<u>Liabilities</u>	
Accounts Payable	Human resources process
Wages Payable	Human resources process
Payroll Taxes Payable	Human resources process
Dividends Payable	Financing process
Interest Payable	Financing process
Loans Payable	Financing process
Total Liabilities	Sum of all Liability balances
<u>Equity</u>	
Common Stock	Financing process
Retained Earnings	Financing process and Net Income
Total Equity	Common Stock + Retained Earnings
Total Liabilities and Equity	Total Liabilities + Total Equity

statement and a balance sheet for Pipefitters Supply Company and the business process or processes used to derive each item. The income statement and balance sheet items in the figures are based on the system you designed upon completion of Chapter 11.

Download to your computer or other storage device the Access database from the book's companion Web site. The unzipped database is named *Ch12.accdb*. It contains all of the objects you will need to follow the illustrations in this chapter as well as to complete the end-of-chapter exercises and problems.

Financial Statement Items Connected with Support Activities

Pipefitters Supply Company's database accounting system currently captures all data necessary to produce financial statement balances related to the events in the four main business processes of a merchandising company: financing, acquisition/payment, human resources (HR), and sales/collection. However, enterprises purchase fixed assets and other non-inventory items to support their main activities. Pipefitters uses some of these non-inventory acquisitions at the time of purchase, such as rent and utilities, while other items have useful lives that last from a few months to many years, such as equipment and insurance policies.

Adding Tables and Relationships to the Database for Support Activities

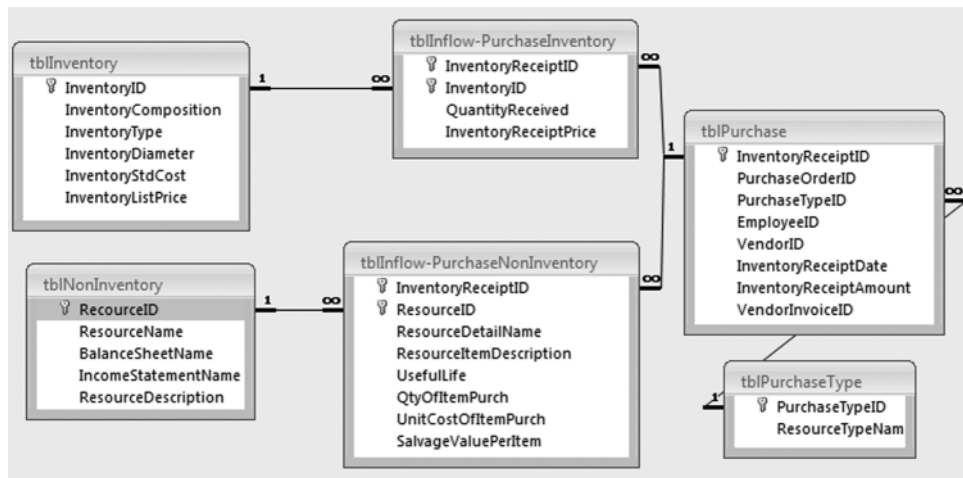
We need to add three additional tables to capture Pipefitters' data about non-inventory items. First, Pipefitters' database needs a non-inventory table to record the types of items that it acquires. This list of items will look like a partial chart of accounts (see Figure 12.3). We included the account names for the balance sheet and income statement because the resource names are often not exactly what appear on financial statements. Like the purchase–inventory relationship, a non-inventory purchase can be for many different items (e.g., a purchase from Home Depot could include items used for repairs and maintenance, and other items classified as supplies) and a non-inventory item, such as utilities, can participate in many purchases. Therefore, we need a relationship table to store details about the individual items on each purchase. Since this relationship table will contain items that are expensed immediately (e.g., utilities), items that are capitalized and amortized or depreciated (e.g., prepaid insurance and equipment), as well as items that are capitalized and not amortized or depreciated (e.g., land), we will add the attributes shown in Figure 12.4 to *tblInflow-PurchaseNonInventory*. In addition to creating *tblNonInventory* and *tblInflow-PurchaseNonInventory*, we need to create a purchase-type table to make the queries and forms that need to distinguish inventory purchases from non-inventory purchases easier to create.

Fig. 12.3
tblNonInventory
in Datasheet view.

Microsoft product screen shot
reprinted with permission from
Microsoft Corporation.

Item #	Resource Name	BalanceSheetName	IncomeStatementName	Description
101	Advertising	Prepaid Advertising	Advertising Expense	Advertising costs
102	Office Supplies	Office Supplies	Supplies Expense - Office	Office supplies
103	Product Supplies	Product Supplies	Supplies Expense - Other	Supplies related to acquiring, storing, and delivering merchandise
104	Insurance	Prepaid Insurance	Insurance Expense	All insurance costs
105	Equipment	Equipment	Depreciation Expense	Equipment, furniture, and fixtures
106	Land	Land		Land
107	Rent	Prepaid Rent	Rent Expense	Building and equipment rental costs
108	Repairs and Maintenance		Repairs and Maintenance Expense	Repairs and maintenance costs not included in other categories
109	Utilities	Prepaid Utilities	Utilities Expense	Electric, gas, water, phone, internet, etc.
110	Bank Charges		Bank Charges	Bank fees related to account maintenance and credit cards

Fig. 12.4 The relationship window showing resource, event, and type tables for purchases.



EXERCISE 12.1: CREATING TABLES AND RELATIONSHIPS TO STORE NON-INVENTORY DATA

1. Create *tblNonInventory*. Click the **Create** tab; click **Table Design** in the Tables group and then click the **Design View** command that appears in the Datasheet context tab. Name the table *tblNonInventory*. Add the primary key and non-key attributes (fields) shown in Figure 12.5. Switch to Datasheet view and add in the records shown in Figure 12.3. Save and close the table.
2. Create *tblInflow-PurchaseNonInventory* using the same procedure as you did for *tblNonInventory*. Add the concatenated primary key (InventoryReceiptID and ResourceID) and non-key attributes (fields) shown in Figure 12.6. Save and close the table.
3. Create *tblPurchaseType* using the same procedure as you did for *tblNonInventory*. Add the primary key and non-key attributes (fields) shown in Figure 12.7. Switch to Datasheet view and add in the two records shown in Figure 12.7. Save and close the table.
4. Add PurchaseTypeID to *tblPurchase* as a foreign key. Open *tblPurchase* in Design view. Insert a blank row after PurchaseOrderID. Type **PurchaseTypeID** in the Field Name column. Use the Field Properties for PurchaseTypeID in Figure 12.7, except enter **No** for the Required property and **Yes (Duplicates OK)** for the Indexed property. Save your changes and switch to Datasheet view. Enter **01** in the PurchaseTypeID column for all records in the table. When you are finished, switch back to Design view and change the Required property to **Yes**.
5. Change the Required property for PurchaseOrderID to **No**. This is necessary because purchases of non-inventory items will not always have a purchase order. Save and close *tblPurchase*.
6. Create the relationships between *tblPurchase*, *tblPurchaseType*, *tblInflow-PurchaseNonInventory*, and *tblNonInventory*. In the **Database Tools** tab, click **Relationships** in the Show/Hide group to reveal a blank Relationships window. Add *tblPurchase*, *tblPurchaseType*, *tblInflow-PurchaseNonInventory*, and *tblNonInventory* to the Relationships window. Use Figure 12.4 as a guide to create the relationships. Notice that Figure 12.4 includes *tblInventory* and *tblInflow-PurchaseInventory* to illustrate that recording acquisitions of inventory items is the same as recording non-inventory acquisitions. Also, Figure 12.4 shows that referential integrity can be enforced between the purchase table and

both resource tables, inventory and non-inventory, by using relationship tables. Save your changes to the Relationships window and close it.

7. Add records for non-inventory purchases to *tblPurchase*. Open *tblPurchase* in Datasheet view. Open *Ch12.xlsx*. **Copy and Paste** the records from the *tblPurchase* worksheet into *tblPurchase* in Access. Save and close *tblPurchase*.
8. Add the purchase line item detail for the non-inventory purchases to *tblInflow-PurchaseNonInventory*. **Copy and Paste** the records from the *tblInflow-PurchaseNonInventory* worksheet into *tblInflow-PurchaseNonInventory* in Access. Save and close *tblInflow-PurchaseNonInventory*.
9. Add cash disbursements related to non-inventory purchases to *tblCashDisbursement*. **Copy and Paste** the records from the *tblCashDisbursement* worksheet into *tblCashDisbursement* in Access. Save and close *tblCashDisbursement*.

Fig. 12.5 Fields for *tblNonInventory*.

Field Name	ResourceID	ResourceName	BalanceSheetName
Key or Attribute	Primary Key	Non-key Attribute	Non-key Attribute
Data Type	Text	Text	Text
Field Size	3	25	50
Format			
Input Mask	000;_		
Caption	Item #	Resource Name	
Required	Yes	Yes	No
Allow Zero Length	No	No	Yes
Indexed	Yes (No Duplicates)	No	No

Field Name	IncomeStatementName	ResourceDescription
Key or Attribute	Non-key Attribute	Non-key Attribute
Data Type	Text	Text
Field Size	50	100
Format		
Input Mask		
Caption		Description
Required	No	Yes
Allow Zero Length	Yes	No
Indexed	No	No

Support Items to be Expensed as Incurred

Pipefitters Supply Company expenses many support items that are expensed in the period in which they are purchased. Pipefitters management decided that any item acquired with a useful life of one month or less will be immediately expensed. Pipefitters also expenses supplies when they are purchased. To capture this information, we need to create a set of queries that will sum the non-inventory items purchased during the period, which means that the date constraints must have both a beginning date and an ending date.

Fig. 12.6 Fields for *tblInflow-PurchaseNonInventory*.

Field Name	InventoryReceiptID	ResourceID	ResourceDetailName	ResourceItemDescription
Key or Attribute	Primary Key	Primary Key	Non-key Attribute	Non-key Attribute
Data Type	Text	Text	Text	Text
Field Size	6	3	50	255
Format				
Input Mask	000000;:_	000;:_		
Caption	Inventory Receipt #	Item #	Name	Description
Required	Yes	Yes	Yes	Yes
Allow Zero Length	No	No	No	No
Indexed	Yes (Duplicates OK)	Yes (Duplicates OK)	No	No

Field Name	UsefulLife	QtyOfItemPurch	UnitCostOfItemPurch	SalvageValuePerItem
Key or Attribute	Non-key Attribute	Non-key Attribute	Non-key Attribute	Non-key Attribute
Data Type	Number	Number	Currency	Currency
Field Size	Long Integer	Integer		
Format		Fixed	Currency	Currency
Decimal Places	Auto	0	2	2
Caption	Useful Life (months)	Quantity	Unit Cost	Salvage Value
Default Value	0	0	0	0
Validation Rule	>=0	>=0	>=0	>=0
Validation Text	Useful life cannot be negative	Quantity of item purchased must be non-negative	Unit cost of item purchased must be non-negative	Salvage value cannot be negative
Required	Yes	Yes	Yes	Yes
Indexed	No	No	No	No

Fig. 12.7 Fields for *tblPurchaseType* and table in Datasheet view.

Field Name	PurchaseTypeID	ResourceTypeName
Key or Attribute	Primary Key	Non-key Attribute
Data Type	Text	Text
Field Size	2	25
Format		
Input Mask	00;:_	
Caption	Purchase Type #	Resource Type
Required	Yes	Yes
Allow Zero Length	No	No
Indexed	Yes (No Duplicates)	No

tblPurchaseType	
Purchase Type #	Resource Type
01	Inventory
02	Non-inventory

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

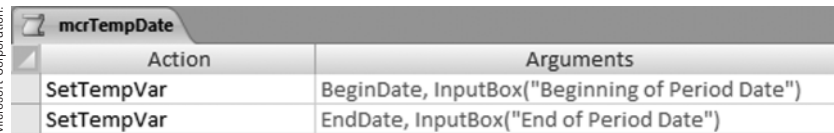
Since the beginning dates and ending dates for all financial statements are the same, you will create a macro to store a user-specified beginning date and ending date in a temporary table. You will use these *temp variables* as date criteria rather than using parameters so that users will only need to enter the beginning and ending dates one time.

EXERCISE 12.2: CREATING A MACRO TO STORE USER-SPECIFIED DATES

1. Click the **Create** tab and click **Macro** in the Other group. Type **SetTempVar** into the Action column of the first row.
2. Click on the **Name** box in the Action Arguments Pane. Type **BeginDate**. In the Expression box, type **InputBox("Beginning of Period Date")**. When the user runs the macro, he or she will see an input box with the text, "Beginning of Period Date," above the input area. Once the user enters the date and clicks OK, the value entered will be stored in **[TempVars]![BeginDate]**.
3. Follow Step 2 again to add a second **SetTempVar** Action with a name of **EndDate**. In the Expression box, type **InputBox("End of Period Date")**.
4. Close the macro. In the Save As dialog box, name your macro **mcrTempDate**. Figure 12.8 shows the macro in Design view.

Fig. 12.8
mcrTempDate in
Design view.

Microsoft product
screen shot reprinted
with permission from
Microsoft Corporation.



Action	Arguments
SetTempVar	BeginDate, InputBox("Beginning of Period Date")
SetTempVar	EndDate, InputBox("End of Period Date")

EXERCISE 12.3: CREATING QUERIES TO COMPUTE ITEMS IMMEDIATELY EXPENSED

The first query computes the purchase line extensions and the extended salvage value. We added other items so that you can use the same query to create a non-inventory purchase form, which is one of the end-of-chapter exercises.

1. Click the **Create** tab and click **Query Design** in the Other group. Add **tblInflowPurchaseNonInventory** to the Table Pane. Add all fields from the Field Roster to the Criteria Pane. Save the query as **qryNonInvExp01-FsubPurchNonInventory**.
2. Click the first available column. Use Expression Builder to create the following expression to compute the extended salvage value: **ExtendedSalvageValue: [SalvageValuePerItem]*[QtyOfItemPurch]**. Open the Property Sheet and select **Currency** Format. Type **Total Salvage Value** for the Caption. Move this column to the immediate right of **SalvageValuePerItem**.
3. Click the next available column. Use Expression Builder to create the following expression to compute the purchase line extension: **PurchaseLineExtension: [UnitCostOfItemPurch]*[QtyOfItemPurch]**. In the Property Sheet, select **Currency** Format. Type **Item Extension** for the Caption.
4. Save your changes; run the query. Your query result should look like the dynaset in Figure 12.9. Close the query.

Create a query to sum non-inventory purchases during the period by resource type that will be immediately expensed.

5. Click the **Create** tab and click **Query Design** in the Other group. Add **tblPurchase**, **tblNonInventory**, and **qryNonInvExp01-FsubPurchNonInventory** to the Table Pane. Link **tblNonInventory** and **qryNonInvExp01-FsubPurchNonInventory** on **ResourceID**. Add the following fields to the Criteria Pane: **ResourceID** and **IncomeStatementName** from **tblNonInventory**, **PurchaseLineExtension** and **UsefulLife** from **qryNonInvExp01-FsubPurchNonInventory**, and **InventoryReceiptDate** from **tblNonInventory**. Move **InventoryReceiptDate** to the left of **UsefulLife**. Save the query as **qryNonInvExp02-ImmediateExpenses**.

6. Click **Totals** in the Show/Hide group. Leave the Total field set to **Group By** for **ResourceID** and **IncomeStatementName**, and set the Total field for **PurchaseLineExtension** to **Sum** to sum expenses by item.
7. Use **Where** in the Total field for **InventoryReceiptDate** and **UsefulLife**—the two fields that are used to limit which records are summed. Enter **Between [tempVars]![BeginDate] And [TempVars]![EndDate]** for the date criteria to select items purchased during the current period. Setting the Criteria for **UsefulLife** to **<=1** enforces Pipefitters' rule that only items with a useful life of one month or less are to be expensed in full in the period acquired (see Figure 12.10). Save your query.
8. Test the query. Double-click **mcrTempDate** to set the beginning of period date to **1/1/2010** and the end of period date to **1/31/2010**. Double-click **qryNonInvExp02-ImmediateExpenses** to run it. Your result should look like the dynaset in Figure 12.11. Why are some of the non-inventory types missing? Close the query.

Fig. 12.9 Dynaset for *qryNonInvExp01-FsubPurchNon-Inventory*.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Inventory	Resource #	Name	Description	Useful Life	Salvage Value	Total Salv	Quantity	Unit Cost	Item Exten
100028	102	Paper	Copy Paper - 10 Ream	0	\$0.00	\$0.00	5	\$35.99	\$179.95
100028	105	Copier/Printer	HP LaserJet M4345	48	\$200.00	\$200.00	1	\$2,600.00	\$2,600.00
100029	110	Bank Fees	Monthly bank charges	0	\$0.00	\$0.00	1	\$37.50	\$37.50
100030	101	Print Advertisi	Annual Yellow Book A	12	\$0.00	\$0.00	1	\$1,749.00	\$1,749.00
100031	107	Equipment Ren	Fork Lift Monthly Ren	1	\$0.00	\$0.00	1	\$450.00	\$450.00
100032	103	Warehouse sup	Various supplies	0	\$0.00	\$0.00	1	\$324.51	\$324.51
100033	109	Water	Monthly water and se	0	\$0.00	\$0.00	1	\$214.85	\$214.85
100034	109	Internet	Monthly internet serv	0	\$0.00	\$0.00	1	\$267.33	\$267.33
100035	109	Gas and Electric	Monthly gas and elect	0	\$0.00	\$0.00	1	\$405.73	\$405.73
100036	104	Property and Li	Annual property and l	12	\$0.00	\$0.00	1	\$14,923.00	\$14,923.00
100037	105	Furniture	Chairs	60	\$25.00	\$300.00	12	\$412.59	\$4,951.08
100038	110	Credit Card Fee	Fees for credit card se	0	\$0.00	\$0.00	1	\$231.45	\$231.45
100039	107	Equipment Ren	Fork Lift Monthly Ren	1	\$0.00	\$0.00	1	\$450.00	\$450.00
100040	103	Warehouse sup	Various supplies	0	\$0.00	\$0.00	1	\$424.00	\$424.00
100041	109	Water	Monthly water and se	0	\$0.00	\$0.00	1	\$219.44	\$219.44
100042	109	Internet	Monthly internet serv	0	\$0.00	\$0.00	1	\$267.33	\$267.33
100043	109	Gas and Electric	Monthly gas and elect	0	\$0.00	\$0.00	1	\$405.73	\$405.73
100044	104	Auto	Delivery truck insurar	6	\$0.00	\$0.00	1	\$3,680.00	\$3,680.00
100045	110	Bank Fees	Monthly bank charges	0	\$0.00	\$0.00	1	\$37.50	\$37.50
100046	110	Bank Fees	Monthly bank charges	0	\$0.00	\$0.00	1	\$37.50	\$37.50
100047	107	Equipment Ren	Fork lift monthly rent	1	\$0.00	\$0.00	1	\$450.00	\$450.00
100048	108	Repair materia	Materials to fix hole i	1	\$0.00	\$0.00	1	\$721.88	\$721.88
100049	109	Water	Monthly water and se	0	\$0.00	\$0.00	1	\$175.90	\$175.90
100050	109	Internet	Monthly internet serv	0	\$0.00	\$0.00	1	\$267.33	\$267.33
100051	109	Gas and Electric	Monthly gas and elect	0	\$0.00	\$0.00	1	\$598.44	\$598.44

Fig. 12.10 *qryNonInvExp02-ImmediateExpenses* in Design view.

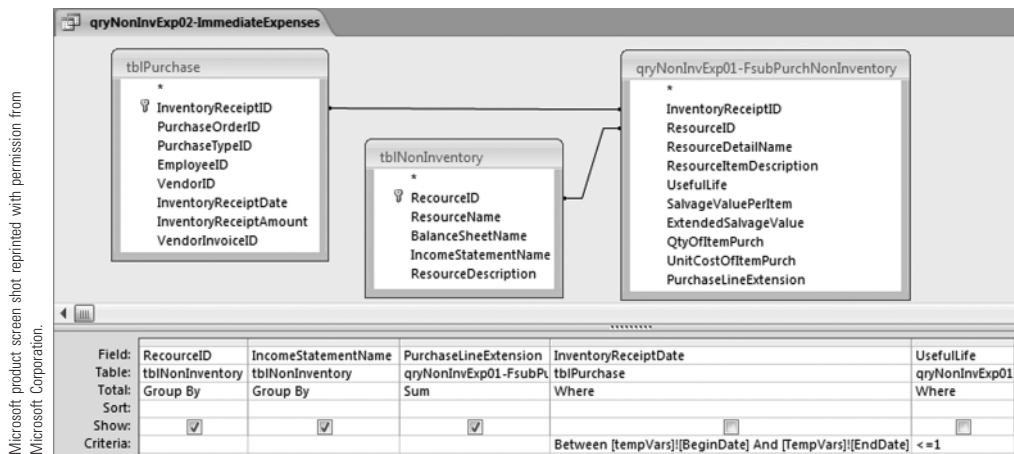


Fig. 12.11 Dynaset for *qryNonInvExp02-ImmediateExpenses* between 1/1/2010 and 1/31/2010.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Item #	IncomeStatement	SumOfPurchaseLineExtension
102	Supplies Expense -	\$179.95
103	Supplies Expense -	\$324.51
107	Rent Expense	\$450.00
109	Utilities Expense	\$482.18
110	Bank Charges	\$37.50

Support Items to Be Capitalized and Amortized or Depreciated

In the previous section, you used the useful life to identify which items are expensed in the period acquired. You also used a date constraint to determine which of these items are expensed in the current period. In this section, the useful life will be used to identify items to be capitalized and amortized. You will then use date functions and date constraints to compute the current-period expense as well as prepaid expense balances and accumulated depreciation.

EXERCISE 12.4: QUERY TO COMPUTE CURRENT PERIOD AMORTIZATION/DEPRECIATION BY ITEM

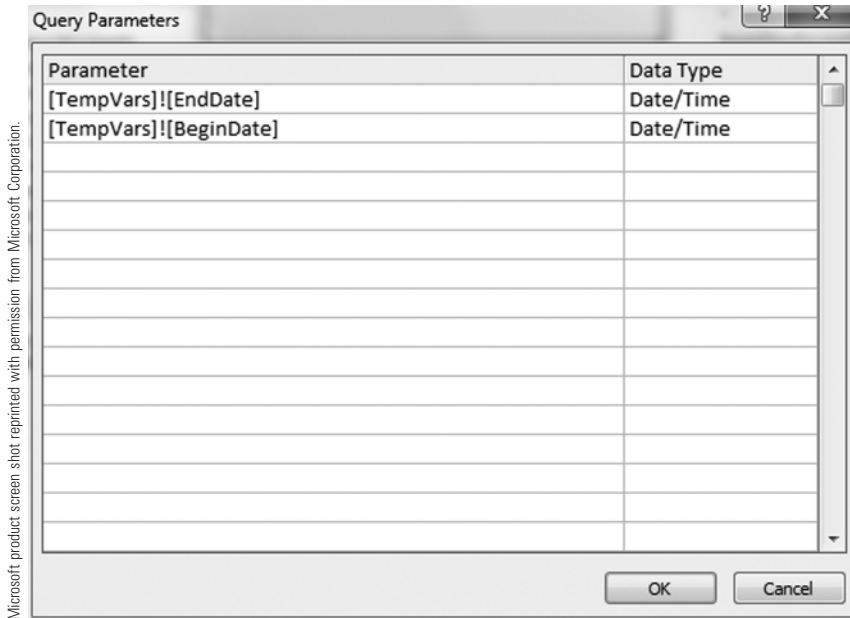
1. Make a copy of *qryNonInvExp02-ImmediateExpenses* and name it *qryNonInvExp03-PrepaidItems*. Open *qryNonInvExp03-PrepaidItems* in Design view. Make sure that the Totals row is visible in the Criteria Pane. Move *InventoryReceiptDate* to the right of *UsefulLife* in the Criteria Pane. Change the Criteria of *UsefulLife* to >1 to limit purchases to items that are to be capitalized and amortized/depreciated. Change the criteria for *InventoryReceiptDate* to $<=[TempVars]![EndDate]$, because any amortizable/depreciable asset acquired prior to the end of the period (not limited to items acquired *during* the current period) may have a useful life that extends into (or beyond) the current period. Drag *ExtendedSalvageValue* from the *qryNonInvExp01-FsubPurchNonInventory* Field Roster to the Criteria Pane to the left of *UsefulLife*. Save your changes.
2. Add the temporary variable parameters—the user-specified beginning and ending reporting dates—to the Parameters list. You will use the temporary variables in several expressions in this query. Because Access does not automatically assign the *Date* Data Type to temporary variables when they are used in a field expression, you will specify in the Parameter List that the parameters are of Data Type *Date*. Click *Parameters* in the Show/Hide group to open the Parameter list. Enter the Parameter names as indicated in Figure 12.12. Select *Date/Time* as the Data Type for each parameter. Click *OK* to close the Parameter List.
3. Compute the daily expense. In the first available column, use Expression Builder to create the following formula: *DailyExpense: ([PurchaseLineExtension]-[ExtendedSalvageValue])/([UsefulLife] * (12/365))*. Why is it necessary to multiply useful life by 12/365? Open the Property Sheet and set the Format to *Currency* and Decimal Places to 2.
4. Compute the end of the asset's useful life. In the next available column, use Expression Builder to create the following formula: *EndOfLifeDate: DateAdd("m", [UsefulLife],[InventoryReceiptDate])*. The *DateAdd* function adds the amount

specified in the second argument ([UsefulLife]) in the unit of measurement specified in the first argument ("m" for months) to the date specified in the third argument ([InventoryReceiptDate]). Since we are only interested in assets whose useful lives extend beyond the beginning of the current period, add `>=[TempVars]![BeginDate]` as the Criteria for `EndOfLifeDate`. In the Property Sheet, set the Format to `Short Date`.

5. Amortization/depreciation expense for the current period begins on the later of the beginning of the period date or the acquisition date. Use an **IIF** statement to make this comparison. By entering the expression below in the next available column in the Field Roster, you will assign the appropriate date to the attribute, `BeginningOfComputationDate`, which will be used as the starting date to compute the current period amortization/depreciation for each asset: `BeginningComputationDate: IIf([InventoryReceiptDate]<[TempVars]![BeginDate],[TempVars]![BeginDate],[InventoryReceiptDate])`. The expression to be evaluated, "Is the purchase date earlier than the beginning date for the reporting period?" is the first argument of the function. If this is true, then assign the value of the second argument, the beginning of period date, to `BeginningComputationDate`. If the expression is false, then the third argument, purchase date, will be assigned to `BeginningComputationDate`. In the Property Sheet, set the Format to `Short Date`.
6. Now that you have established the `BeginningComputationDate`, you will use another IIF statement to compute the `EndingComputationDate`, which is the earlier of the end of the reporting period or the end date of the asset's useful life. Enter the following expression in the next available column in the Field Roster: `EndingComputationDate: IIf([EndOfLifeDate]>[tempvars]![EndDate],[tempvars]![EndDate],[EndOfLifeDate])`. If the end of the asset's useful life is later than the end of the reporting period date (the first argument), then assign the end of period date to `EndingComputationDate` (the second argument). If the expression is false, meaning that the end of the asset's useful life falls within the reporting period, then assign the end of useful life date (the third argument) to `EndingComputationDate`. In the Property Sheet, set the Format to `Short Date`.
7. Save the query so that the new fields you have created will show up in Expression Builder. Use the **DateDiff** function to compute the number of days—denoted by "d" in the function's first argument—between `BeginningComputationDate` (the function's second argument) and `EndingComputationDate` (the function's third argument). You will then multiply the result by the expense per day. Create the expression to compute the amortization/depreciation expense of each asset for the reporting period in Expression Builder: `PeriodExpense: DateDiff("d",[BeginningComputationDate],[EndingComputationDate])*[DailyExpense]`. Open the Property Sheet and select `Currency` Format.
8. Test the `BeginningComputationDate` and `EndingComputationDate` expressions. Double-click `mcrTempDate` and set the beginning and ending dates to `2/1/2010` and `10/31/2010`, respectively. Click `Run` in the Results group to run `qryNonInvExp03-PrepaidItems`. Notice in the top panel of Figure 12.13 that the `BeginningComputationDate` for the last three assets is later than `2/1/2010`. Why? Why is the `EndingComputationDate` for the fifth asset (Insurance purchased for \$3,680.00) earlier than `10/31/2010`?
9. Test the date constraints for `InventoryReceiptDate` and `EndOfLifeDate`. Run `mcrTempDate` again and set the beginning and ending dates to `1/1/2010` and `1/31/2010`, respectively, and run `qryNonInvExp03-PrepaidItems` once more.

Compare the first and second dynasets in Figure 12.13. Notice in the second panel how the InventoryReceiptDate constraint only selects assets acquired before the end of the reporting period. Now run `mcrTempDate` a third time. Set the beginning and ending dates to `4/1/2011` and `10/31/2011`, respectively. Rerun `qryNonInvExp03-PrepaidItems`. Compare the first and third dynasets. How does the date constraint for `EndOfLifeDate` correctly prevent the query from selecting the advertising item and two insurance items?

Fig. 12.12 Query Parameters window for `qryNonInvExp03-PrepaidItems`.



Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.13 Dynasets for `qryNonInvExp03-PrepaidItems`.

Item	IncomeStatementN	Item Ext	Total Sal	Useful Life	Date Rec	DailyExp	EndOfLife	Beginning	EndingCo	PeriodExp
105	Depreciation Expense	\$2,600.00	\$200.00	48	1/10/2010	\$1.64	1/10/2014	2/1/2010	10/31/2010	\$447.12
101	Advertising Expense	\$1,749.00	\$0.00	12	1/28/2010	\$4.79	1/28/2011	2/1/2010	10/31/2010	\$1,303.36
104	Insurance Expense	\$14,923.00	\$0.00	12	2/14/2010	\$40.88	2/14/2011	2/14/2010	10/31/2010	\$10,589.20
105	Depreciation Expense	\$4,951.08	\$300.00	60	2/15/2010	\$2.55	2/15/2015	2/15/2010	10/31/2010	\$657.52
104	Insurance Expense	\$3,680.00	\$0.00	6	2/28/2010	\$20.16	8/28/2010	2/28/2010	8/28/2010	\$3,649.75

Between 2/1/2010 and 10/31/2010.

Item	IncomeStatementN	Item Ext	Total Sal	Useful Life	Date Rec	DailyExp	EndOfLife	Beginning	EndingCo	PeriodExp
105	Depreciation Expense	\$2,600.00	\$200.00	48	1/10/2010	\$1.64	1/10/2014	1/10/2010	1/31/2010	\$34.52
101	Advertising Expense	\$1,749.00	\$0.00	12	1/28/2010	\$4.79	1/28/2011	1/28/2010	1/31/2010	\$14.38

Between 1/1/2010 and 1/31/2010.

Item	IncomeStatementN	Item Ext	Total Sal	Useful Life	Date Rec	DailyExp	EndOfLife	Beginning	EndingCo	PeriodExp
105	Depreciation Expense	\$2,600.00	\$200.00	48	1/10/2010	\$1.64	1/10/2014	4/1/2011	10/31/2011	\$350.14
105	Depreciation Expense	\$4,951.08	\$300.00	60	2/15/2010	\$2.55	2/15/2015	4/1/2011	10/31/2011	\$542.84

Between 4/1/2011 and 10/31/2011.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

In Exercise 12.4, we showed you how to compute amortization/depreciation expense for all non-inventory assets with a useful life greater than one month. We will finish this process in Exercise 12.5 by summing these expenses by non-inventory type.

EXERCISE 12.5: CREATING A QUERY TO SUM AMORTIZATION/DEPRECIATION EXPENSE BY NON-INVENTORY TYPE

1. Click the **Create** tab and click **Query Design** in the Other group. Add `qryNonInvExp03-PrepaidItems` to the Table Pane. Add `ResourceID`, `IncomeStatementName`, and `PeriodExpense` to the Criteria Pane. Save the query as `qryNonInvExp04-SumPrepaysByItem`.
2. Click **Totals** in the Show/Hide group and change the Total for `PeriodExpense` to **Sum**. Save your query. The dynaset for the period between 1/1/2010 and 3/31/2010 is shown in Figure 12.14.

Fig. 12.14 Dynaset for `qryNonInvExp04-SumPrepaysByItem` for the period between 1/1/2010 and 3/31/2010.

Resource #	IncomeStatementName	SumOfPeriodExpense
101	Advertising Expense	\$297.09
104	Insurance Expense	\$2,464.92
105	Depreciation Expense	\$243.64

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Producing Income Statement and Balance Sheet Line-Items for Support Activities

Pipefitters Supply Company will have several items on its income statement and balance sheet related to its support activities. In the previous section, you computed amounts immediately expensed by non-inventory type as well as the amortization/depreciation expense by non-inventory type. In this section, we will show you how to combine these items for the income statement. We will also show you how to compute the prepaid-expense balance and balances related to depreciable assets—equipment and accumulated depreciation.

EXERCISE 12.6: COMPUTING EXPENSES FOR AN INCOME STATEMENT RELATED TO SUPPORT ACTIVITIES

Combine, by non-inventory type, immediately expensed items with amortization/depreciation expenses.

1. Click the **Create** tab and click **Query Design** in the Other group. Add `tblNonInventory`, `qryNonInvExp02-ImmediateExpenses`, and `qryNonInvExp04-SumPrepaysByItem` to the Table Pane. Add the following fields to the Criteria

Pane: ResourceID and IncomeStatementName from tblNonInventory, SumOfPurchaseLineExtension from qryNonInvExp02-ImmediateExpenses, and SumOfPeriodExpense from qryNonInvExp04-SumPrepaysByItem.

- Use outer joins to show both immediately expensed items and amortization/depreciation for all non-inventory types other than land. All resource types will appear in the dynaset, even if there are no corresponding immediate expenses or amortization/depreciation expenses. Drag ResourceID from tblNonInventory to ResourceID in the qryNonInvExp02-ImmediateExpenses Field Roster. Double-click the relationship between the two Field Rosters. Select Option 2 to Include ALL records from tblNonInventory and only those records from qryNonInventoryExpenses-PeriodExpenses where the joined fields are equal. Create a similar outer join between tblNonInventory and qryNonInvExp04-SumPrepaysByItem (see Figure 12.15). Prevent land from being selected from tblNonInventory by adding <>"106" (the ResourceID for land) to the Criteria for ResourceID. Save the query as qryNonInvExp05-SumImmediatesAndPrepays.
- Combine the immediate-expensed items with amortization/depreciation. Click the next available column in the Criteria Pane and enter the following expression: IncomeStatementExpense: Nz([SumOfPurchaseLineExtension]) + Nz([SumOfPeriodExpense]). Open the Property Sheet and set the Format to Currency and Decimal Places to 2. Save the query. Figure 12.16 shows the dynaset when the query is run for the period between 1/1/2010 and 1/31/2010. Notice that the Nz function allows totals to be computed when one or both expenses are null.

Fig. 12.15 qryNonInvExp05-SumImmediates-AndPrepays in Design view.

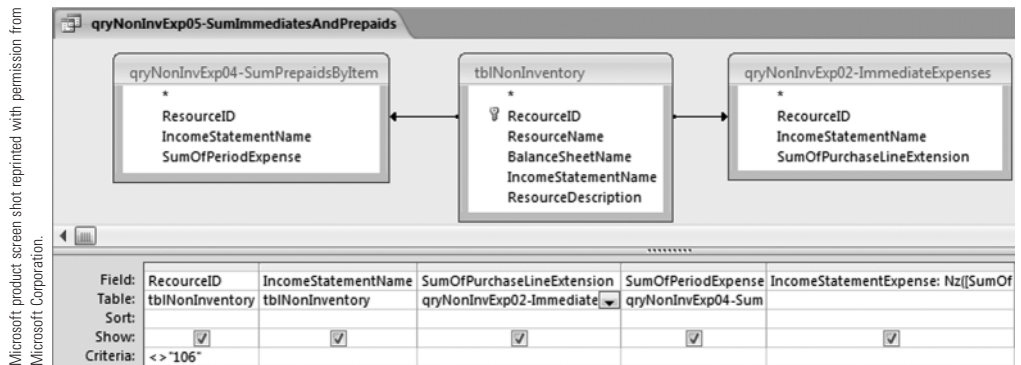


Fig. 12.16 Dynaset for qryNonInvExp05-SumImmediatesAnd-Prepays between 1/1/2010 and 1/31/2010.

Item #	IncomeStatementName	SumOfPurchaseLineExtension	SumOfPeriodExpense	IncomeStatementExpense
101	Advertising Expense		\$14.38	\$14.38
102	Supplies Expense - Office	\$179.95		\$179.95
103	Supplies Expense - Other	\$324.51		\$324.51
104	Insurance Expense		\$0.00	\$0.00
105	Depreciation Expense		\$34.52	\$34.52
107	Rent Expense	\$450.00		\$450.00
108	Repairs and Maintenance Expense			\$0.00
109	Utilities Expense	\$482.18		\$482.18
110	Bank Charges	\$37.50		\$37.50

The next set of exercises illustrates how to compute the prepaid expense and fixed asset balances for the balance sheet.

EXERCISE 12.7: COMPUTING PREPAID EXPENSE BALANCE FOR EACH CAPITALIZED SUPPORT ITEM

1. Make a copy of `qryNonInvExp03-PrepaidItems` and save the copy as `qryPrepaidExpenses01-PrepaysAndFixedAssets`. Open `qryPrepaidExpenses01-PrepaysAndFixedAssets` in Design view. Change the second field from `IncomeStatementName` to `BalanceSheetName`. Delete `BeginningComputationDate` and `PeriodExpense` from the Criteria Pane. Save your changes to the query.
2. Compute the accumulated amortization/depreciation through the `EndingComputationDate`, which is the earlier of the end of the reporting period or the end of the asset's useful life. Use Expression Builder to add the following field to the Criteria Pane: `AccumAmortAndDepr: DateDiff("d",[InventoryReceiptDate],[EndingComputationDate])*[DailyExpense]`. Open the Property Sheet and set the Format to `Currency` and Decimal Places to `2`. Save the query.
3. Compute the prepaid expense/fixed asset balance by subtracting the accumulated amortization/depreciation from the purchase price. Use Expression Builder to add the following field to the Criteria Pane: `PrepaidExpense: [PurchaseLineExtension]-[AccumAmortAndDepr]`. In the Property Sheet, set the Format to `Currency` and Decimal Places to `2`. Save the query. See the Dynaset for the period between 3/1/2010 and 3/31/2010 in Figure 12.17.

Fig. 12.17 Dynaset for `qryPrepaidExpenses01-PrepaysAndFixedAssets` for the period between 3/1/2010 and 3/31/2010.

Rt	BalanceSheetName	Item Extel	Total Sal	Useful Life	Date Rec	DailyExp	EndOfLife	EndingCo	AccumAmor	PrepaidExpense
105	Equipment	\$2,600.00	\$200.00	48	1/10/2010	\$1.64	1/10/2014	3/31/2010	\$131.51	\$2,468.49
101	Prepaid Advertising	\$1,749.00	\$0.00	12	1/28/2010	\$4.79	1/28/2011	3/31/2010	\$297.09	\$1,451.91
104	Prepaid Insurance	\$14,923.00	\$0.00	12	2/14/2010	\$40.88	2/14/2011	3/31/2010	\$1,839.82	\$13,083.18
105	Equipment	\$4,951.08	\$300.00	60	2/15/2010	\$2.55	2/15/2015	3/31/2010	\$112.14	\$4,838.94
104	Prepaid Insurance	\$3,680.00	\$0.00	6	2/28/2010	\$20.16	8/28/2010	3/31/2010	\$625.10	\$3,054.90

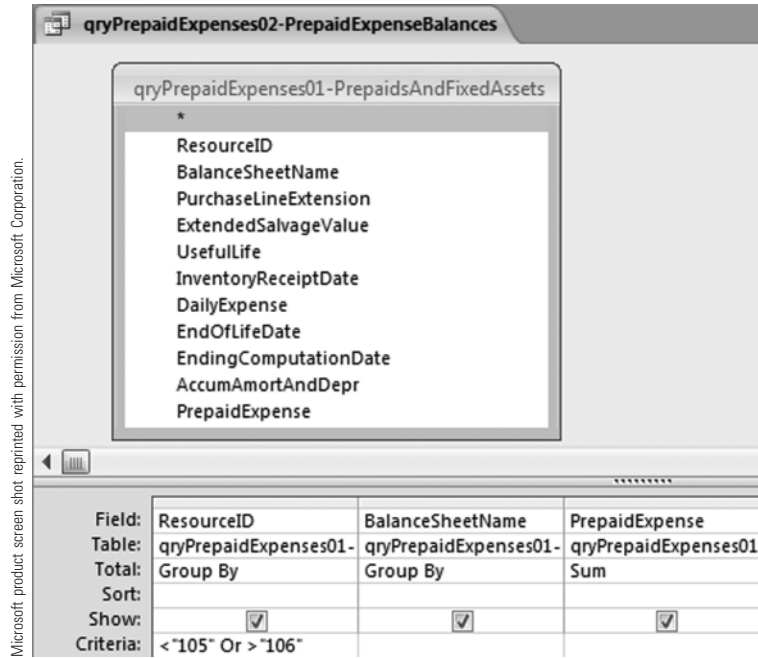
Microsoft product screen shot reprinted with permission from Microsoft Corporation.

EXERCISE 12.8: COMPUTING THE BALANCE SHEET AMOUNTS FOR PREPAID EXPENSES

1. Click the `Create` tab and click `Query Design` in the Other group. Add `qryPrepaidExpenses01-PrepaysAndFixedAssets` to the Table Pane. Add the following fields to the Criteria Pane: `ResourceID`, `BalanceSheetName`, and `PrepaidExpense`. Type `<"105" Or >"106"` in the Criteria property for `ResourceID` to prevent fixed assets from being selected. Save your query as `qryPrepaidExpenses02-PrepaidExpenseBalances`.
2. Click `Totals` in the Show/Hide group. Change the Total cell for `PrepaidExpense` to `Sum`. Save your changes (see Figure 12.18).

- Double-click `mcrTempDate`. Set the beginning and ending dates to `3/1/2010` and `3/31/2010`, respectively. Run `qryPrepaidExpenses02-PrepaidExpenseBalances`. Compare your dynaset to the dynaset in Figure 12.19. Notice how the two prepaid insurance balances were combined. Why was the equipment balance not computed? Close the query.

Fig. 12.18
qryPrepaidExpenses02-PrepaidExpenseBalances in Design view.



Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.19 Dynaset for *qryPrepaidExpenses02-PrepaidExpenseBalances* for the period between 3/1/2010 and 3/31/2010.

Resource #	BalanceSheetName	SumOfPrepaidExpense
101	Prepaid Advertising	\$1,451.91
104	Prepaid Insurance	\$16,138.08

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

EXERCISE 12.9: COMPUTING THE BALANCE SHEET AMOUNTS FOR FIXED ASSETS

- Click the `Create` tab and click `Query Design` in the `Other` group. Add `qryPrepaidExpenses01-PrepaysAndFixedAssets` to the `Table Pane`. Add the following fields to the `Criteria Pane`: `PurchaseLineExtension`, `AccumAmortAndDepr`, `PrepaidExpense`, and `ResourceID`. Type `Between "105" and "106"` in the `Criteria` property for `ResourceID` to select only fixed assets. Save your query as `qryPrepaidExpenses03-FixedAssets`.

2. Click **Totals** in the Show/Hide group. Change the Total cell for ResourceID to **Where**. Change the Totals for the other three fields to **Sum**.
3. Open the Property Sheet. For each of the summed fields, set Format to **Currency** and Decimal Places to **2**. Save the query. Add the following captions: **Property, Plant, and Equipment** for PurchaseLineExtension, **Accumulated Depreciation** for AccumDeprAndAmort, and **Property, Plant, and Equipment, Net** for Prepaid Expense. Save your changes.
4. Double-click **mcrTempDate**. Set the beginning and ending dates to **3/1/2010** and **3/31/2010**, respectively. Run **qryPrepaidExpenses03-FixedAssets** and compare your dynaset to the dynaset in Figure 12.20. Notice how the balances for the two equipment purchases were combined. Close the query.

Fig. 12.20
Dynaset for
*qryPrepaidExpenses03-
FixedAssets* for the
period between
3/1/2010 and 3/31/2010.

Property, Plant, & Equipment	Accumulated Depreciation	Property, Plant, & Equipment, Net
\$7,551.08	\$243.64	\$7,307.44

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Now that the database has been modified to accommodate acquisitions for support activities and compute the related financial statement balances, you are ready to tackle financial statement balances that span multiple business processes.

Financial Statement Balances That Span Multiple Business Processes

Pipefitters Supply Company's income statement and balance sheet contain three balances that span multiple business processes: cash, merchandise inventory, and "cost of goods sold." The merchandise inventory balance and cost of goods sold are based on a common set of queries because cost of goods sold is the cost of inventory purchased that is sold, and the ending inventory balance is the cost of inventory purchased that is not sold. In this section, you will compute these three balances for reporting on financial statements.

Computing the Cash Balance

The cash balance is the first item that appears on most companies' balance sheets. Although the cash balance is affected by all exchanges (other than barter transactions) between an enterprise and its external partners, there are only two events that affect the cash balance—cash receipts and cash disbursements. Pipefitters Supply Company receives cash from financing activities as loan proceeds from creditors and as contributed capital from stockholders. Pipefitters also receives cash from customers in the sales/collection process. Cash disbursements occur in Pipefitters' financing process to repay loans to creditors and to pay dividends to stockholders. The other cash disbursements occur in the acquisition/payment process, to pay vendors for goods and services purchased, and in the human resources (HR) process to pay its employees for their labor.

The balance in the cash account is computed in three queries: summing all cash receipts, summing all cash disbursements, and then subtracting the sum of the cash disbursements from the sum of the cash receipts. Since cash is a balance-sheet item, the cash receipts and cash disbursements are summed from the company's inception through the balance sheet date. Therefore, the summation queries will only have an ending date constraint.

EXERCISE 12.10: USING QUERIES TO COMPUTE THE CASH BALANCE

The first query sums cash receipts through the balance sheet date.

1. Click the **Create** tab and click **Query Design** in the Other group. Double-click **tblCashReceipt** in the Show Table dialog box. Close the dialog box.
2. Add **CashReceiptAmount** and **CashReceiptDate**, respectively, to the Criteria Pane. Click **Totals** in the Show/Hide group. Change the Total cell for **InventoryReceiptAmount** to **Sum**. Open the **Property Sheet** and type **Cash Receipts** in the Caption property.
3. Change the **InventoryReceiptDate** Total to **Where**. In the Criteria cell of **InventoryReceiptDate**, type **<=[TempVars]![EndDate]** to sum all purchases up through the balance sheet date. Save your query as **qryCash1-SumOfCR**.
4. Check your query; run **mcrTempDate**. Set the beginning and ending dates to **1/1/2010** and **1/31/2010**, respectively, and run **qryCash1-SumOfCR**. Total cash receipts on the balance sheet date of 1/31/2010 should be **\$146,021.77**. Close the query.

Create a query to compute the sum of the cash disbursements. It is similar to the query used to compute the sum of the cash receipts.

5. Follow steps 1 through 4 above again and make the following changes (see Figure 12.21).
 - In Step 1, select **tblCashDisbursement** instead of **tblCashReceipt**.
 - In Step 2, add **CashDisbursementAmount** and **CashDisbursementDate**, respectively.
 - In Step 3, save your query as **qryCash2-SumOfCD**.
 - In Step 4, the total cash disbursements for a 1/31/2010 balance sheet date is **\$32,022.50**.

The third and final query uses the results of the two queries to compute the ending cash balance.

6. Click the **Create** tab and click **Query Design** in the Other group. Add **qryCash1-SumOfCR** and **qryCash2-SumOfCD** to the Table Pane. The fields in these two queries are independent of one another and do not need to be joined.
7. Add to the Criteria Pane **SumOfCashReceiptAmount** from the **qryCash1-SumOfCR** field roster and **SumOfCashDisbursementAmount** from the **qryCash2-SumOfCD** field roster. Save your query as **qryCash3-CashBalance**.
8. Click the first available Field on the Criteria Pane. Click **Builder** in the Query Setup group. Enter **CashBalance: Nz([SumOfCashReceiptAmount],0) - Nz([SumOfCashDisbursementAmount],0)**. Although Pipefitters would likely have both cash receipts and cash disbursements within the first few days or weeks of whenever it began business, we chose to use the Nz function so that the cash balance could be calculated for *any* date (see the top panel of Figure 12.22).
9. Open the **Property Sheet**. Set the Format to **Currency** and Decimal Places to **2**. Type **Cash** as the caption for your newly created field, **CashBalance**. Save your query.
10. Run **mcrTempDate**. Set the beginning and ending dates to **1/1/2010** and **1/31/2010**, respectively, and run **qryCash3-CashBalance**. Your result should be **\$113,999.27** (see the second panel of Figure 12.22). Close the query.

Computing the Inventory and Cost of Goods Sold Balances

Enterprises have two major decisions when it comes to computing inventory and cost of goods sold (represented as “CoGS”). First, management must decide whether to use the periodic method or the perpetual inventory method. Second, management must decide which inventory cost flow assumption it will use: first-in-first-out (FIFO), last-in-first-out (LIFO), weighted average, or specific identification.

Fig. 12.21
qryCash2-SumOfCD
 in Design view.

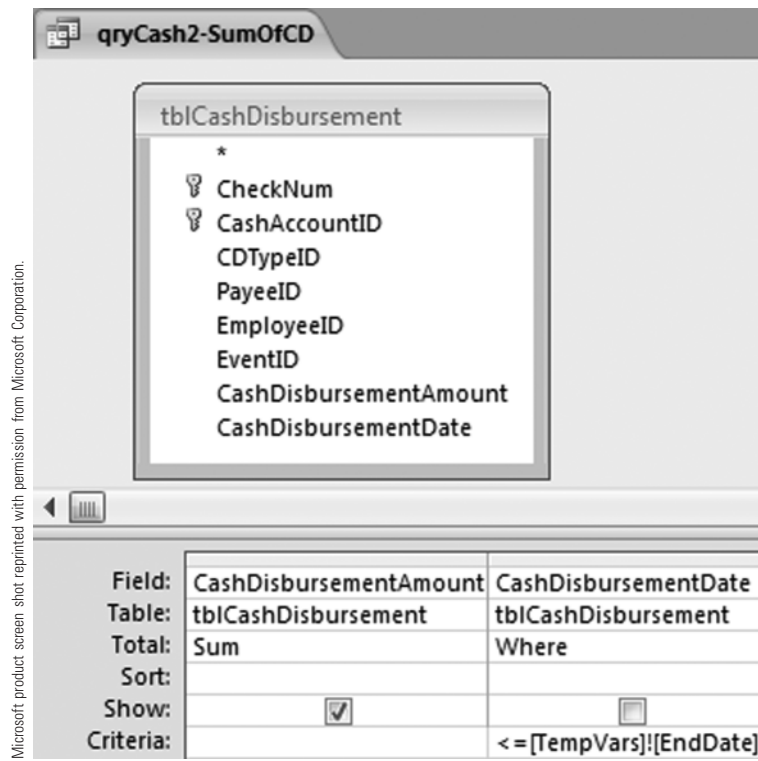
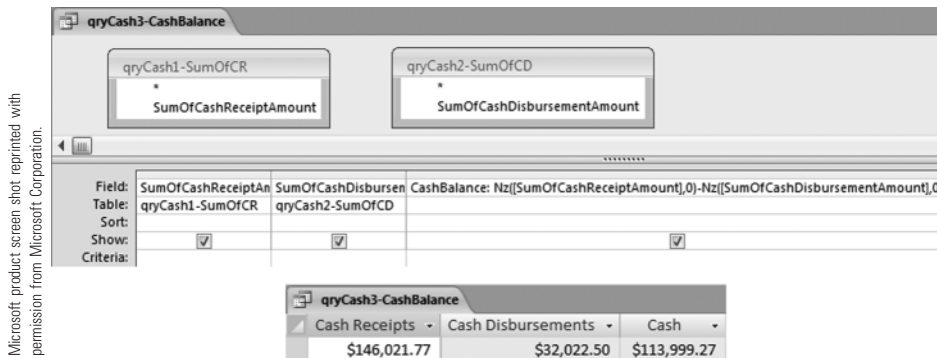


Fig. 12.22
qryCash3-CashBalance
 in Design view and
 dynaset for period
 ended 1/31/2010.



Inventory Methods and Cost-Flow Assumptions

Under the *periodic* inventory method, ending inventory and CoGS are computed only at the end of the accounting period, usually after a company takes a physical count of the inventory on hand. Prior to the end of the period, companies use estimating techniques to compute the inventory balance and/or CoGS. This was a popular technique prior to computerized accounting systems. Most companies currently using computerized accounting systems use the *perpetual* inventory method. Perpetual inventory methods update the cost of goods sold and inventory balance after every sale. Clearly, this method is far more computationally intensive than using the periodic method.

The second decision is to choose the inventory cost flow assumption. Under the *specific identification* method, each individual item in the inventory has a unique identifier so that each item sold already has a cost (e.g., purchase price) associated with it. Pipefitters cannot use this method because it only tracks inventory by type (e.g., 1-inch brass elbow) rather than by assigning a unique number to each 1-inch brass elbow. Therefore, Pipefitters, and most merchandisers, use either weighted average, LIFO, or FIFO.

Weighted average CoGS and ending inventory computations are relatively easy, because all you have to do is sum the quantity available for sale for each item; sum total purchase prices for each item; and then divide the cost by the quantity of each item to get the weighted-average unit cost for each item of inventory. Then, multiply the quantity sold for each item by its weighted-average unit cost to get the cost of goods sold. The inventory balance is simply the quantity on hand of each item multiplied by its weighted-average unit cost. Weighted average presents two problems. First, the cost assigned to the unsold items will change every time the weighted-average computations are performed. Second, the balances in CoGS and in ending inventory will change depending on the frequency of the computations. Therefore, to have comparative numbers from one period to the next, the computations must be performed every time an item is sold. We will not demonstrate this method because computational challenges of updating the inventory costs after each sale are beyond the scope of this book. However, you will have a chance to build a set of queries to compute weighted-average CoGS and ending inventory balances, without updating the cost assigned to ending inventory, as an end-of-chapter exercise.

The *first-in-first-out (FIFO)* method assumes that the oldest goods, *first* items purchased, are the *first* items sold. Thus, the ending inventory consists of the most recent purchases. This is the most common inventory method used by businesses. One benefit of this method is that the cost of goods sold and the ending inventory balances will be the same regardless of how frequently these balances are computed. An advantage of FIFO over weighted average is that costs assigned to ending inventory items do not need to be updated. Although this method is challenging to implement in a relational database without writing code, the process is straightforward. Therefore, we will implement a FIFO inventory system for Pipefitters Supply Company.

The remaining inventory costing method is *last-in-first-out (LIFO)*. LIFO assumes that the newest goods, *last* items purchased, are the *first* items sold. Therefore, CoGS consists of the most recent purchases. Like FIFO, the cost assigned to ending inventory items does not change. However, the cost of goods sold and the ending inventory balances will change depending on the frequency of the computations. Therefore, to have comparative numbers from one period to the next, the computations must be performed every time an item is sold. Although less challenging to implement than weighted average (no need to change the cost assigned to ending inventory items), the additional queries and macros needed to update the ending inventory and CoGS after every sale makes it more difficult to implement than FIFO. However, since the steps to implement LIFO are similar to FIFO, you will have the opportunity to add LIFO inventory to Pipefitters Supply Company's database in the end-of-chapter problems.

Computing Ending Inventory

The first step in computing the ending inventory balance is to determine the quantity of each item on hand at the end date, which can be the end of the accounting period or any other date on which Pipefitters wishes to know its ending inventory balance. This step requires three queries: (1) sum the quantity purchased by item through the end date, (2) sum the quantity sold by item through the end date, and (3) subtract quantity sold

by item from quantity purchased by item to obtain the quantity on hand of each item. This set of queries is similar to the queries in Exercise 12.1 used to compute the cash balance. The main difference is that you are computing a balance for each inventory item.

EXERCISE 12.11: COMPUTING QUANTITY OF INVENTORY ITEMS ON HAND

The first query sums quantity purchased by item through the balance sheet date.

1. Click the **Create** tab and then click **Query Design** in the Other group. Add **tblInflow-PurchaseInventory** and **tblPurchase** to the Table Pane. Add **InventoryID**, and **QuantityReceived** from **tblInflow-PurchaseInventory** to the Criteria Pane. From **tblPurchase**, add **PurchaseTypeID** to the right of **InventoryID** and add **InventoryReceiptDate** to the right of **QuantityReceived**.
2. Click **Totals** in the Show/Hide group. Change the Total cell for **QuantityReceived** to **Sum** and the Total for **InventoryReceiptDate** to **Where**. Add "01" to the Criteria cell of **PurchaseTypeID** to limit purchases to inventory purchases. Although the link between **tblPurchase** and **tblInflow-PurchaseInventory** makes it unnecessary to have **PurchaseTypeID** in the query, the **PurchaseTypeID** field is needed for queries later in the computation process.
3. Add **<=[TempVars]![EndDate]** to the Criteria cell of **InventoryReceiptDate** to sum all purchases through the balance sheet date. Save your query as **qryInventory01-SumQtyPurchByItem**.
4. Run **mcrTempDate**. Set the beginning and ending dates to **1/1/2010** and **1/31/2010**, respectively, and run **qryInventory01-SumQtyPurchByItem**. You should have 50 records in your dynaset. Close the query.

Create a query to compute quantity sold by item through the balance sheet date. It is very similar to the query to compute the sum of the cash receipts.

5. Follow steps 1 through 4, above, making the following changes.
 - In Step 1, select **tblOutflow-SaleInventory** and **tblSale**. Add **InventoryID**, **QuantitySold**, and **ShippingDate**, respectively.
 - In Step 2, add **Sum** to **QuantitySold** and **Where** to **ShippingDate**.
 - In Step 3, Add **<=[TempVars]![EndDate]** to the Criteria cell of **ShippingDate**. Save your query as **qryInventory02-SumQtySoldByItem** (see Figure 12.23).
 - In Step 4, you should have 39 records for a 1/31/2010 balance sheet date.

The third and final query uses the results of the two queries to compute the quantity on hand.

6. Click the **Create** tab and click **Query Design** in the Other group. Add **qryInventory01-SumQtyPurchByItem** and **qryInventory02-SumQtySoldByItem** to the Table Pane. Create an outer join between the two queries using **InventoryID** to select all items from **qryInventory01-SumQtyPurchByItem** and those items that match **qryInventory02-SumQtySoldByItem** (see Figure 12.24). Add to the Criteria Pane **InventoryID** from the **qryInventory01-SumQtyPurchByItem** field roster.
7. Use the **Nz** function for quantity purchased and quantity sold to allow computations if one of the values is null. Click the first available Field on the Criteria Pane. Enter **QtyPurchased: Nz([SumOfQuantityReceived],0)**. In the next available column, enter **QtySold: Nz([SumOfQuantitySold],0)**. Save the query as **qryInventory03-QtyOnHandByItem**.
8. Use Expression Builder to enter the last expression: **QtyOnHand: [QtyPurchased] - [QtySold]**. Save your query (see Figure 12.24).
9. Run **mcrTempDate** using **1/1/2010** and **1/31/2010** as the beginning and ending dates, respectively, and run **qryInventory03-QtyOnHandByItem**. You should have 50 records in your dynaset. Close the query.

Fig. 12.23
qryInventory02-SumQtySoldByItem
 in Design view.

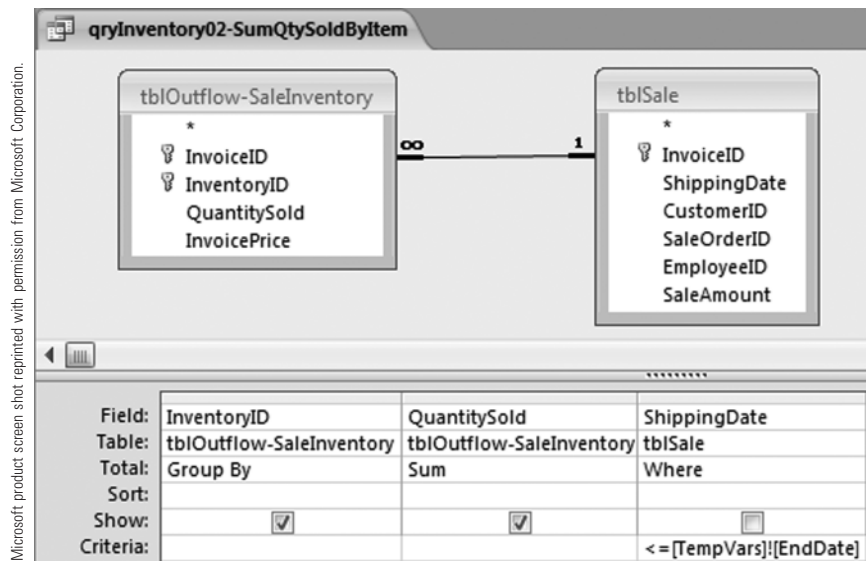
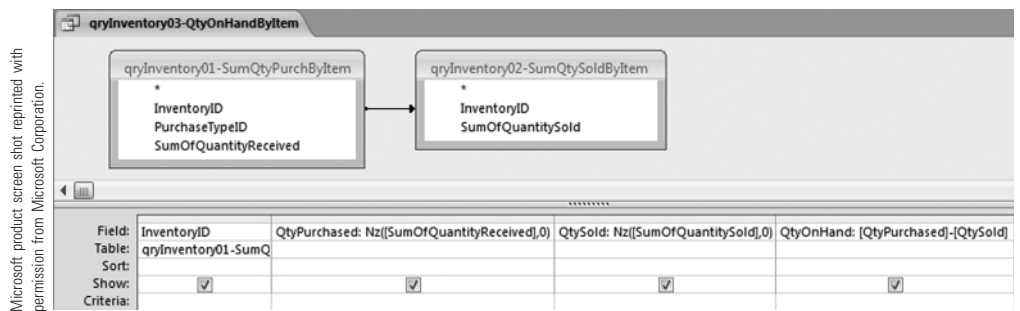


Fig. 12.24
qryInventory03-QtyOnHandByItem
 in Design view.



The FIFO inventory cost flow method assumes that the inventory on hand, the unsold inventory, are the most recent items purchased. Therefore, to compute the cost of ending inventory, we need to identify for each inventory item the quantity, cost per item, and date purchased in descending order by date. Then, we can assign the quantity on hand to the most recent purchases until all units on hand have been assigned.

Look at purchases related to Item 1002 in Figure 12.25. First, we used a make-table query to create the listing of items purchased, grouped by item, and sorted in descending order by date. We added a column for quantity unassigned and gave the column an initial value of -1 to indicate that no activity had taken place in that cell. We also created a quantity assigned column and assigned an initial value of 0 , indicating that none of the units on hand had been assigned to that particular purchase. Next, we placed the quantity on hand, 85 units, into the QtyUnassigned cell for the most recent purchase. We then assigned the lesser of the quantity purchased or the unassigned items. Since there were 85 items on hand and the most recent purchase was for only 30 items, we assigned 30 items to the 3/8/2010 purchase. The 55 remaining unassigned items ($85 - 30 = 55$)

Fig. 12.25
tblInventory-Compute-
CostOfItemsOnHand
in Datasheet view for
the period ended
3/31/2010.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

tblInventory-ComputeCostOfItemsOnHand	InventoryID	QuantityReceived	InventoryReceiptPrice	InventoryReceiptDate	QtyUnassigned	QtyAssigned
1001	60	\$13.20	2/19/2010	33	33	
1001	20	\$13.00	2/9/2010	-1	0	
1001	75	\$13.00	1/25/2010	-1	0	
1001	75	\$13.20	1/7/2010	-1	0	
1002	30	\$16.00	3/8/2010	85	30	
1002	40	\$15.95	2/19/2010	55	40	
1002	65	\$16.00	2/9/2010	15	15	
1002	60	\$16.00	1/25/2010	-1	0	
1002	25	\$15.95	1/7/2010	-1	0	
1003	35	\$20.89	2/19/2010	25	25	
1003	15	\$21.00	2/9/2010	-1	0	
1003	50	\$21.00	1/25/2010	-1	0	
1003	15	\$20.89	1/7/2010	-1	0	
1004	30	\$26.30	2/22/2010	18	18	
1004	15	\$26.30	2/6/2010	-1	0	
1004	25	\$26.30	1/29/2010	-1	0	
1004	15	\$26.30	1/17/2010	-1	0	
1005	40	\$33.20	2/22/2010	15	15	
1005	30	\$33.20	2/6/2010	-1	0	
1005	15	\$33.25	1/29/2010	-1	0	
1006	10	\$39.20	2/6/2010	25	10	
1006	10	\$39.25	1/29/2010	15	10	
1006	35	\$39.25	1/17/2010	5	5	
1007	30	\$24.50	3/8/2010	53	30	
1007	20	\$24.75	2/19/2010	23	20	
1007	50	\$24.50	2/9/2010	3	3	

Record: 1 of 172 | No Filter | Search

are then entered for the 2/19/2010 purchase. We assigned 40 units to this purchase because the 40 items purchased was less than the 55 items unassigned. The remaining 15 unassigned items were entered in the QtyUnassigned column of the 2/9/2010 purchase. All 15 remaining items were assigned to the 2/9/2010 purchase because more than 15 items were purchased on 2/9/2010.

After the inventory on hand is assigned to the most recent purchases, we will create a query to compute the line extension, QtyAssigned × InventoryReceiptPrice, for all inventory on hand (QtyAssigned > 0). Finally, we will sum the line extensions to get the ending inventory balance for the balance sheet. We can also compute the ending inventory by item for internal purposes, though we will not do that here.

EXERCISE 12.12: ASSIGNING QUANTITIES ON HAND TO THE MOST RECENT PURCHASES

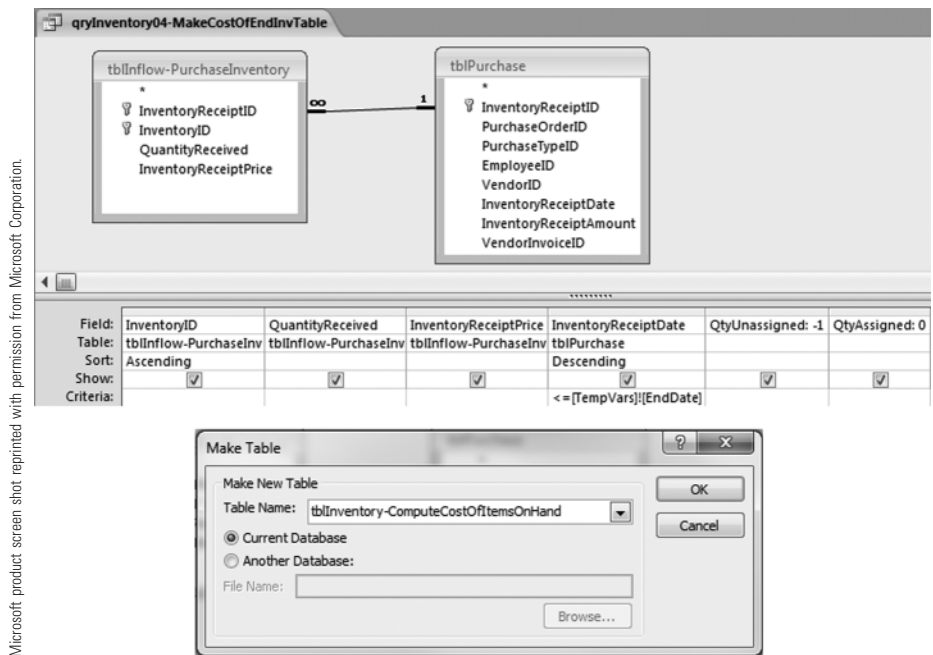
1. Create a Make Table query similar to *qrySumOfPOLineExtensionsByIR* in Exercise 9.35, in order to create a table like what appears in Figure 12.25. This table will be used to assign items on hand to the most recent purchases. Click the **Create** tab and click **Query Design** in the Other group. Add **tblInflow-PurchaseInventory** and **tblPurchase** to the Table Pane. Add to the Criteria Pane **InventoryID** and **QuantityReceived**, and **InventoryReceiptPrice** from **tblInflow-PurchaseInventory**, and **InventoryReceiptDate** from **tblPurchase**. Enter **<=[TempVars]![EndDate]** as the Criteria for **InventoryReceiptDate**. Save the query as **qryInventory04-MakeCostOfEndInvTable**.

2. Add fields for quantity unassigned and for quantity assigned and populate them with an initial value. In the first two available columns in the Criteria Pane, enter [QtyUnassigned: -1](#) and [QtyAssigned: 0](#), respectively (see Figure 12.26). Save your changes.
3. Click [Make Table](#) in the Query Type group and enter [tblInventory-ComputeCostOfItemsOnHand](#) as shown in the Make Table dialog box in Figure 12.26. Save the query. Run [mcrTempDate](#) using [1/1/2010](#) and [3/31/2010](#) as the beginning and ending dates, respectively, and double-click [qryInventory04-MakeCostOfEndInvTable](#) to run it. Click [Yes](#) in the warning dialog boxes. Open [tblInventory-ComputeCostOfItemsOnHand](#) to verify that it has 172 records. Notice that QtyUnassigned is populated with -1s and QtyAssigned is populated with 0s. Close the table.

The next few queries populate the QtyUnassigned for the most recent purchase of each item in [tblInventory-ComputeCostOfItemsOnHand](#) with quantity on hand.

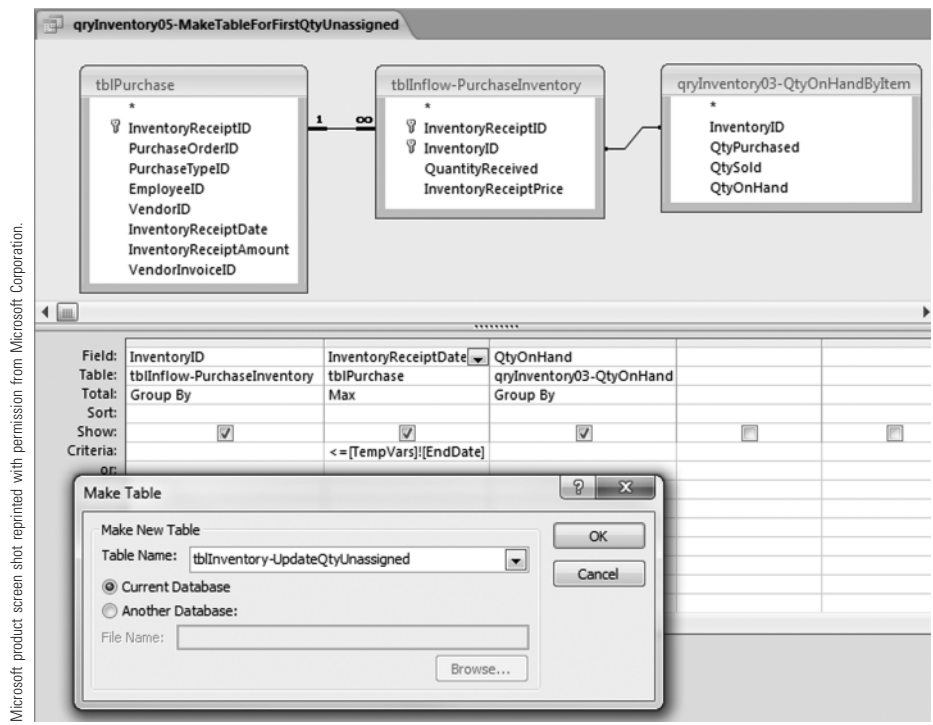
4. Click the [Create](#) tab and click [Query Design](#) in the Other group. Add [tblPurchase](#), [tblInflow-PurchaseInventory](#) and [qryInventory03-QtyOnHandByItem](#) to the Table Pane. Link [tblInflow-PurchaseInventory](#) and [qryInventory03-QtyOnHandByItem](#) on [InventoryID](#). Save your query as [qryInventory05-MakeTableForFirstQtyUnassigned](#).
5. Add the fields in Figure 12.27 to the Criteria Pane. Enter [<=\[TempVars\]!\[EndDate\]](#) as the Criteria for [InventoryReceiptDate](#). Click [Totals](#) in the Show/Hide group. Change the Total of [InventoryReceiptDate](#) to [Max](#) to select the most recent purchase on or before the balance sheet date. Click [Make Table](#) in the Query Type group and enter [tblInventory-UpdateQtyUnassigned](#) as shown in the Make Table dialog box in Figure 12.27. Save the query.
6. Run [mcrTempDate](#) using [1/1/2010](#) and [3/31/2010](#) as the beginning and ending dates, respectively, and double-click [qryInventory05-MakeTableForFirstQtyUnassigned](#) to run it. Click [Yes](#) in the warning dialog boxes. Close the query. Open [tblInventory-UpdateQtyUnassigned](#) to verify that it has 58 records. Close the table.
7. Use the values in [tblInventory-UpdatedQtyUnassigned](#) to update [tblInventory-ComputeCostOfItemsOnHand](#). Create a new query. Add [tblInventory-ComputeCostOfItemsOnHand](#) and [tblInventory-UpdateQtyUnassigned](#) to the Table Pane. Connect the two tables on [InventoryID](#) and by their date fields—[InventoryReceiptDate](#) and [MaxOfInventoryReceiptDate](#)—to update the most recent purchase of each item in [tblInventoryComputeCostOfItemsOnHand](#). Click [Update](#) in the QueryType group. Add [QtyUnassigned](#) from [tblInventory-ComputeCostOfItemsOnHand](#) to the Criteria Pane. Enter [\[tblInventory-UpdateQtyUnassigned\].\[QtyOnHand\]](#) in the Update To cell. Save the query as [qryInventory06-UpdateFirstQtyUnassigned](#).
8. Copy [qryInventory06-UpdateFirstQtyUnassigned](#) and save it as [qryInventory07-UpdateFirstQtyAssigned](#). Change the Field name in the Criteria Pane to [QtyAssigned](#). Replace the Update To cell with the following expression that assigns the lesser of the quantity purchased or the quantity on hand to QtyAssigned: [IIf\(\[QuantityReceived\] < \[QtyUnassigned\],\[QuantityReceived\],\[QtyUnassigned\]\)](#) (see Figure 12.28). Save and close the query.
9. Run [mcrTempDate](#) using [1/1/2010](#) and [3/31/2010](#) as the beginning and ending dates. Run [qryInventory04-](#), [qryInventory05-](#), [qryInventory06-](#), and [qryInventory07-](#), respectively. (We will use this shortcut notation to refer to consecutively numbered queries and tables with long names to make them easier to read.) Click [Yes](#) in the warning dialog boxes. Open [tblInventory-UpdateQtyUnassigned](#) to verify that the QtyUnassigned and QtyAssigned are updated for the most recent purchase (see Figure 12.29). Close the table.

Fig. 12.26
*qryInventory04-
 MakeCostOf-
 EndInvTable* in
 Design view.



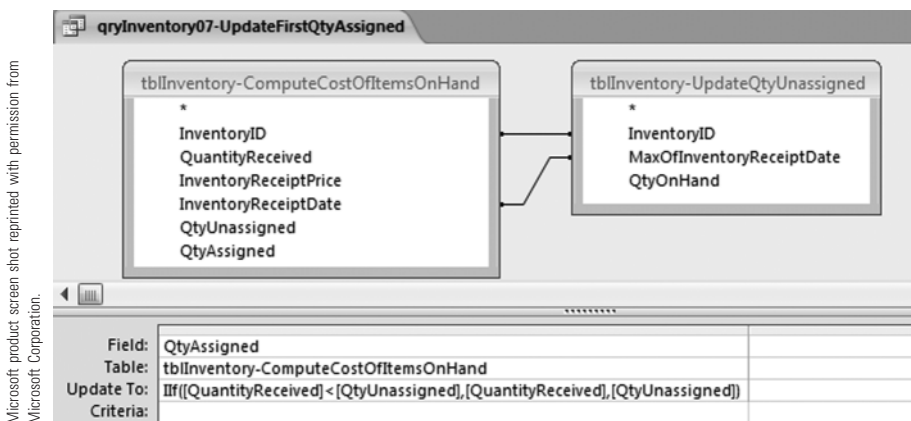
Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.27
*qryInventory05-
 MakeTableForFirstQty-
 Unassigned* in
 Design view.



Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.28
qryInventory07-UpdateFirstQtyAssigned
 Assigned in Design view.



Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.29 First 27 records of *tblInventory-ComputeCostOfItemsOnHand* after initial update.

InventoryID	QuantityRec	InventoryRe	InventoryRe	QtyUnassign	QtyAssignec
1001	60	\$13.20	2/19/2010	33	33
1001	20	\$13.00	2/9/2010	-1	0
1001	75	\$13.00	1/25/2010	-1	0
1001	75	\$13.20	1/7/2010	-1	0
1002	30	\$16.00	3/8/2010	85	30
1002	40	\$15.95	2/19/2010	-1	0
1002	65	\$16.00	2/9/2010	-1	0
1002	60	\$16.00	1/25/2010	-1	0
1002	25	\$15.95	1/7/2010	-1	0
1003	35	\$20.89	2/19/2010	25	25
1003	15	\$21.00	2/9/2010	-1	0
1003	50	\$21.00	1/25/2010	-1	0
1003	15	\$20.89	1/7/2010	-1	0
1004	30	\$26.30	2/22/2010	18	18
1004	15	\$26.30	2/6/2010	-1	0
1004	25	\$26.30	1/29/2010	-1	0
1004	15	\$26.30	1/17/2010	-1	0
1005	40	\$33.20	2/22/2010	15	15
1005	30	\$33.20	2/6/2010	-1	0
1005	15	\$33.25	1/29/2010	-1	0
1006	10	\$39.20	2/6/2010	25	10
1006	10	\$39.25	1/29/2010	-1	0
1006	35	\$39.25	1/17/2010	-1	0
1007	30	\$24.50	3/8/2010	53	30
1007	20	\$24.75	2/19/2010	-1	0
1007	50	\$24.50	2/9/2010	-1	0
1007	60	\$24.50	1/25/2010	-1	0

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

The ending-inventory-computation table is now set up and updated for the initial quantities unassigned and assigned. In Exercise 12.13, we will create queries to update the quantity unassigned by item—quantity on hand less the sum of all items assigned. Then, Exercise 12.14 will illustrate how to use macros to repeat this process until all items on hand are assigned to the appropriate purchases.

EXERCISE 12.13: UPDATING QTYASSIGNED AND QTYUNASSIGNED IN *TBLINVENTORY-COMPUTECOSTOFITEMSONHAND*

1. Use Figure 12.30 to create a query to sum quantity assigned by item. Remember to click **Totals** in the Show/Hide group to add the Total row to the Criteria Pane. Save your query as *qryInventory08-SumOfQuantityAssigned*.
2. Update the quantity unassigned by subtracting total quantity assigned from quantity on hand. Create a new query in Design view. Add *qryInventory03-QtyOnHandByItem* and *qryInventory08-SumOfQuantityAssigned* to the Criteria Pane. Join the two queries on *InventoryID*. Add *InventoryID* from *qryInventory08-QtyOnHand* from *qryInventory03-*, and *SumOfQtyAssigned* from *qryInventory08-*, respectively. Save the query as *qryInventory09-UpdatedQtyUnassigned*. Use Expression Builder to add *NewQtyUnassigned: [QtyOnHand]-[SumOfQtyAssigned]* to the Criteria Pane. Save and close the query.
3. Create a Make Table query similar to *qryInventory05-MakeTableForFirstQtyUnassigned* that will replace the old *tblInventory-UpdateQtyUnassigned* with a new version for each update iteration. Use Figure 12.31 as a guide. The Criteria of *>0* for *NewQtyUnassigned* in *qryInventory09-* only allows inventory items that have quantities unassigned to be selected. Additionally, the Criteria of *-1* for *QtyUnassigned* in *tblInventory-* requires that the most recent purchase date for each item to be selected from purchases that have not yet had items assigned to it. Save your query as *qryInventory10-MakeTableForUpdatedQtyUnassigned*. Close the query.
4. Run *mcrTempDate* using *1/1/2010* and *3/31/2010* as the beginning and ending dates and run *qryInventory10-*. Open *tblInventory-UpdateQtyUnassigned*. Notice that it now has only 26 records. Why?
5. Create a query similar to *qryInventory06-UpdateFirstQtyUnassigned* that will use the values in *tblInventory-UpdatedQtyUnassigned* to update *tblInventory-ComputeCostOfItemsOnHand*. Use Step 7 in Exercise 12.13 and Figure 12.32 to guide you. Name your query *qryInventory11-UpdateNextOldestQtyUnassigned*.
6. Create a query similar to *qryInventory07-UpdateFirstQtyAssigned* that will use the values in *tblInventory-UpdatedQtyUnassigned* to update *tblInventory-ComputeCostOfItemsOnHand*. Use steps 7 and 8 in Exercise 12.13 and Figure 12.33 to guide you. Name your query *qryInventory12-UpdateNextOldestQtyAssigned*.
7. Run *mcrTempDate* using *1/1/2010* and *3/31/2010* as the beginning and ending dates and run *qryInventory11-* and *qryInventory12-*. Open *tblInventory-ComputeCostOfItemsOnHand*. Notice that no additional records for Item 1001 were updated, while a second record for Item 1002 was updated (see Figure 12.34). Why?

Fig. 12.30
qryInventory08-SumOfQuantityAssigned
 SumOfQuantityAssigned in Design view.

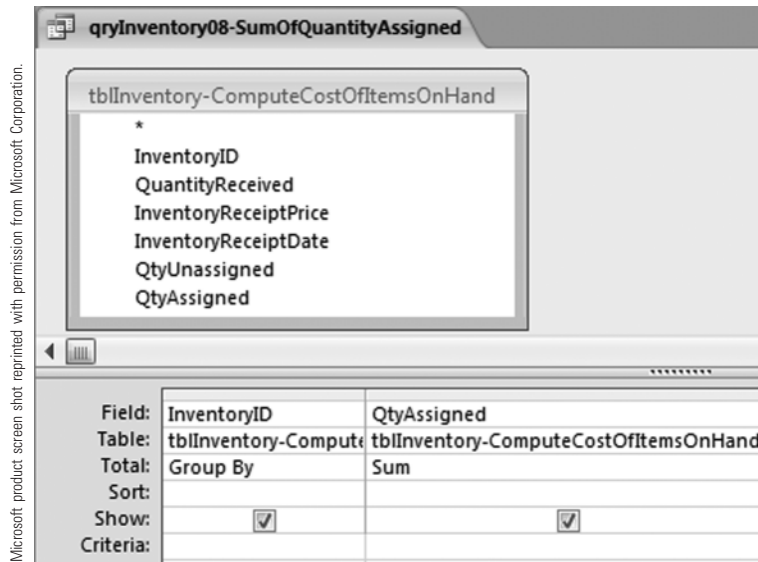


Fig. 12.31 *qryInventory10-MakeTableForUpdatedQtyUnassigned*
 MakeTableForUpdatedQtyUnassigned in Design view.

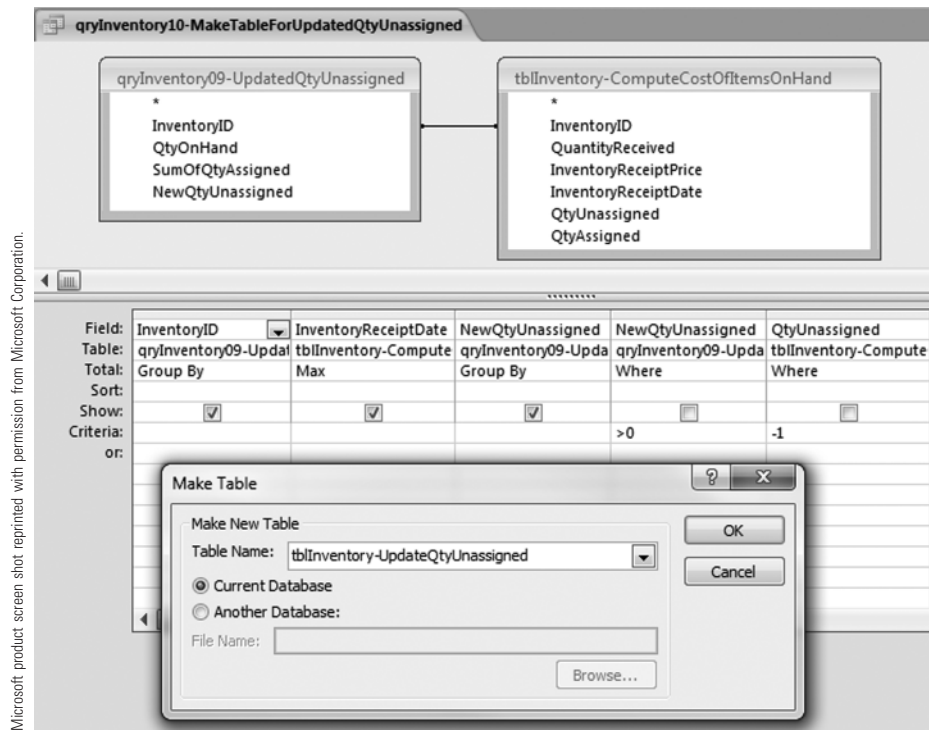


Fig. 12.32
*qryInventory11-UpdateNext-
 OldestQtyUnassigned*
 in Design view.

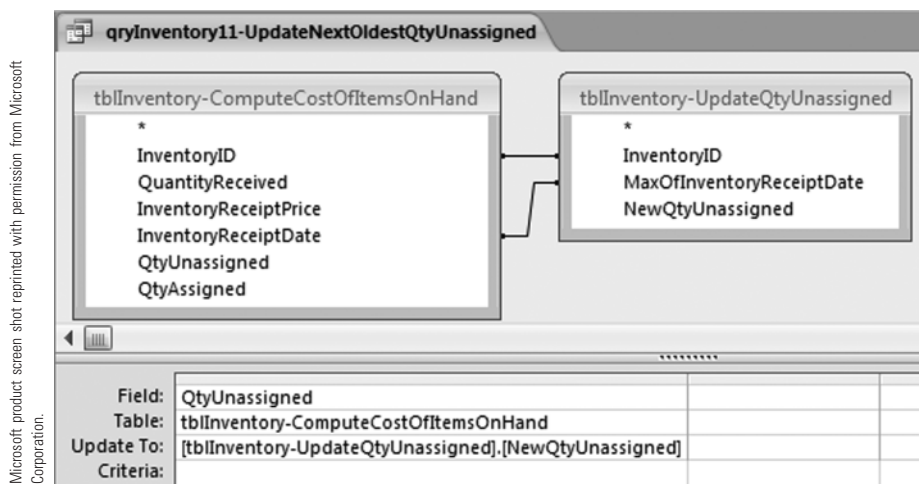


Fig. 12.33
*qryInventory12-UpdateNext-
 OldestQtyAssigned*
 in Design view.

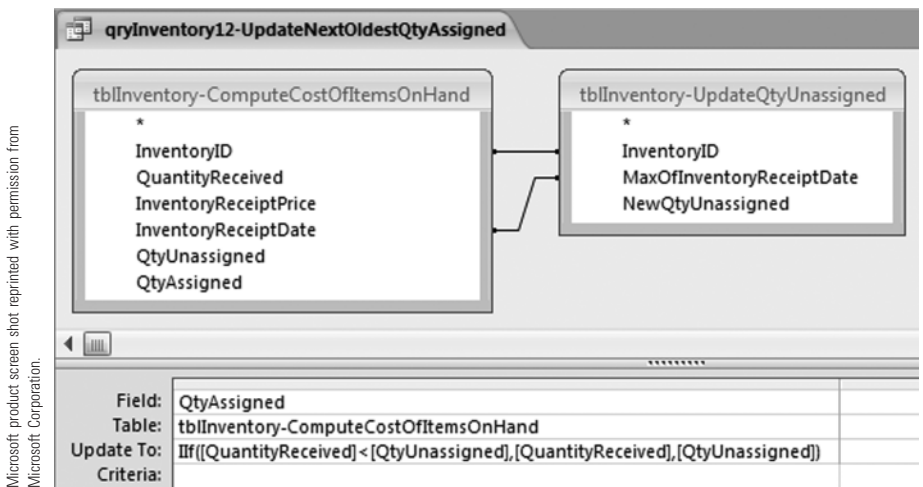


Fig. 12.34 Records
 for Items 1001 and
 1002 in *tblInventory-
 ComputeCostOfItems-
 OnHand* after one
 subsequent update.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

InventoryID	QuantityRec	InventoryRe	InventoryRe	QtyUnassign	QtyAssignec
1001	60	\$13.20	2/19/2010	33	33
1001	20	\$13.00	2/9/2010	-1	0
1001	75	\$13.00	1/25/2010	-1	0
1001	75	\$13.20	1/7/2010	-1	0
1002	30	\$16.00	3/8/2010	85	30
1002	40	\$15.95	2/19/2010	55	40
1002	65	\$16.00	2/9/2010	-1	0
1002	60	\$16.00	1/25/2010	-1	0
1002	25	\$15.95	1/7/2010	-1	0

EXERCISE 12.14: COMPLETING THE COMPUTATION FOR ENDING INVENTORY BALANCE

Queries *qryInventory08-*, *qryInventory09-*, *qryInventory10-*, *qryInventory11-*, and *qryInventory12-* need to be run together until all inventory on hand are assigned. Once the assignment process is complete, line extensions are computed for inventory on hand. Finally, the line extensions are summed to produce the ending inventory balance.

1. Create a macro to run *qryInventory08-*, *qryInventory09-*, *qryInventory10-*, *qryInventory11-*, and *qryInventory12-*. Click the **Create** tab and click **Macro** in the Other group. Click **Show All Actions** in the Show/Hide group, because the Set Warnings action is not available in the default set of actions. In the Action column of the first row, type **SetWarnings**. Leave the default setting, No, to suppress warning dialog boxes. Add three **Open Query** Actions, as indicated in Figure 12.35, to run *qryInventory10-*, *qryInventory11-*, and *qryInventory12-*. We do not need to add *qryInventory08-* and *qryInventory09-* because they will be run from within *qryInventory10-*. Since this macro will be run from within the main inventory macro, save it as **mcrInventory02-UpdateComputeEndInvTable**. Close the macro.
2. Create the main inventory macro. Click the **Create** tab and click **Macro** in the Other group. Click **Show All Actions** in the Show/Hide group in the Action column of the first row, type **SetWarnings**. Leave the default setting to No. Add four **Open Query** Actions as indicated in Figure 12.36 to run *qryInventory01-* through *qryInventory07-*. We do not need to add *qryInventory01-*, *qryInventory02-*, and *qryInventory03-* because they will be run from within *qryInventory04-*. Save the macro as **mcrInventory01-EndingInventory**.
3. Add a Macro Action to keep running *mcrInventory02-* until *tblInventory-ComputeCostOfItemsOnHand* is completely updated. Type **RunMacro** in the next available Action cell and select **mcrInventory02-UpdateComputeEndInvTable** in the Macro Name cell in the Action Arguments pane. We will use the **DCount** function in the Repeat Expression argument to run *mcrInventory02-* in succession until *tblInventory-UpdateQtyUnassigned* does not have any records in it. When this occurs, the DCount function will be false and *mcrInventory02-* will stop running. Enter **DCount("[InventoryID]", "[tblInventory-UpdateQtyUnassigned]")** in Repeat Expression. Save and close the macro.
4. Run **mcrTempDate** using 1/1/2010 and 3/31/2010 as the beginning and ending dates and run *mcrInventory01-*. Open *tblInventory-UpdateQtyUnassigned*. Notice that there are no records in this table. Why? Close the table. Now, open *tblInventory-ComputeCostOfItemsOnHand*. It should look like the datasheet in Figure 12.25. Scroll through to see that all inventory on hand was properly assigned to the most recent purchases for each item.
5. Create a new query in Design view to compute the inventory-on-hand line extensions. Add *tblInventory-ComputeCostOfItemsOnHand* to the Table Pane. Add **InventoryID**, **InventoryReceiptPrice**, and **QtyAssigned** to the Criteria Pane. Save your query as **qryInventory13-InvOnHandLineExtension**. Enter the expression to compute the line extension in the next column of the Criteria Pane: **InvOnHandLineExtension: [InventoryReceiptPrice]*[QtyAssigned]**. Save the query. When you run the query, why are many of the line extensions \$0.00?
6. Sum the line extensions to arrive at the ending inventory balance. Create a new query in Design view. Add *qryInventory13-InvOnHandLineExtension* to the Table Pane. Add **InvOnHandLineExtension** to the Criteria Pane. Click **Totals** in the Show/Hide group. Change the Total cell to **Sum**. Open the Property Sheet and enter **Inventory** as the Caption property. Save your query as **qryInventory14-EndingInventory**. Close the query.
7. Open **mcrInventory01-EndingInventory** in Design view to add the last step to the macro. Add another **OpenQuery** Action and select **qryInventory14-EndingInventory**

in the Query Name control. Since this macro will be run as part of a macro to generate financial statements, add a **Close** Action to close *qryInventory14-* after the macro runs (see Figure 12.36).

8. Test your ending inventory queries and macros. Run *mcrTempDate* using *2/1/2010* and *2/28/2010* as the beginning and ending dates. Double-click *mcrInventory01-EndingInventory* to run it. Then double-click *qryInventory14-EndingInventory* to verify the ending inventory balance of **\$30,499.52**.

Fig. 12.35
mcrInventory02-UpdateCompute-EndInvTable in Design view.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

mcrInventory02-UpdateComputeEndInvTable	
Action	Arguments
SetWarnings	No
OpenQuery	qryInventory10-MakeTableForUpdatedQtyUnassigned, Datasheet, Edit
OpenQuery	qryInventory11-UpdateNextOldestQtyUnassigned, Datasheet, Edit
OpenQuery	qryInventory12-UpdateNextOldestQtyAssigned, Datasheet, Edit

Fig. 12.36
mcrInventory01-EndingInventory in Design view.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

mcrInventory01-EndingInventory	
Action	Arguments
SetWarnings	No
OpenQuery	qryInventory04-MakeCostOfEndInvTable, Datasheet, Edit
OpenQuery	qryInventory05-MakeTableForFirstQtyUnassigned, Datasheet, Edit
OpenQuery	qryInventory06-UpdateFirstQtyUnassigned, Datasheet, Edit
OpenQuery	qryInventory07-UpdateFirstQtyAssigned, Datasheet, Edit
RunMacro	mcrInventory02-UpdateComputeEndInvTable, DCount("[InventoryID]","[tblInventory-UpdateQtyUnassigned]")
OpenQuery	qryInventory14-EndingInventory, Datasheet, Edit
Close	Query, qryInventory14-EndingInventory, Yes

Computing Cost of Goods Sold

When using the FIFO inventory method, the inventory items that make up CoGS are the oldest of the goods available for sale (GAS). This is difficult to compute directly because we need to know which items are on hand at the beginning of the period. However, because we know the quantity sold and the quantity on hand at the end of the period, we can compute the quantity available for sale by adding quantity sold to the ending quantity on hand. Since the quantity available for sale are the most recent items purchased, we can compute the cost of GAS using the same method we used to compute ending inventory. Once we have a value for GAS, we can compute CoGS by subtracting ending inventory from GAS.

EXERCISE 12.15: COMPUTING COST OF GOODS AVAILABLE FOR SALE

Compute the quantity of goods available for sale by item and assign these quantities to the most recent purchase line items. These queries are very similar to the queries you created for ending inventory.

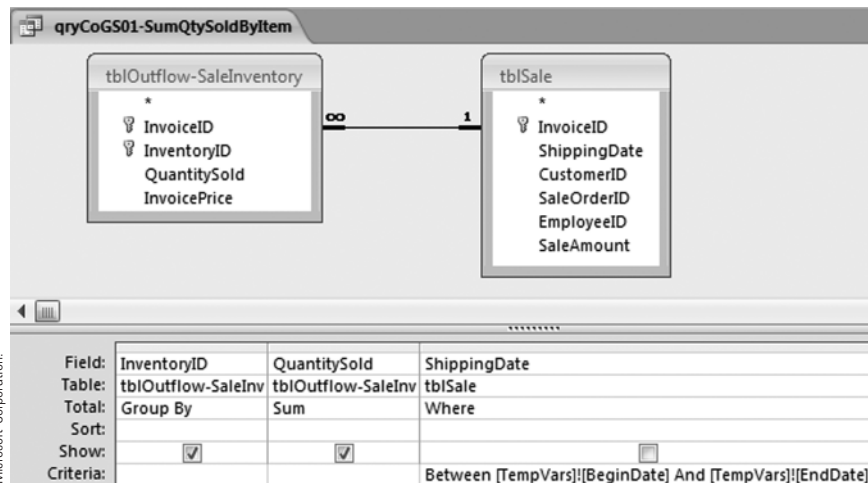
1. Use Exercise 12.11, steps 1 and 2 (for *qryInventory01-SumQtyPurchByItem*), and Figure 12.37 for guidance to sum quantity sold by item for the reporting period. Name the query *qryCoGS01-SumQtySoldByItem*. Notice that the ShippingDate Criteria specifies the period *between* the beginning and end of the period instead of *on or before* the end date. Why?

2. Create a query to add quantity on hand to quantity sold to arrive at quantity available for sale. Use Exercise 12.11, steps 6 through 8 (for *qryInventory03-QtyOnHandByItem*), and Figure 12.38 for guidance. Create the outer join by dragging *InventoryID* from *qryInventory03-QtyOnHandByItem* to *qryCoGS01-SumQtySoldByItem* and selecting Option 2 in the Join Properties dialog box. Save the query as *qryCoGS02-QtyAvailableForSale*.
3. Run *mcrTempDate* using 1/1/2010 and 1/31/2010 as the beginning and ending dates, respectively, and run *qryCoGS02-QtyAvailableForSale*. You should have 50 records in your dynaset. Close the query.

Assign quantities on hand to the most recent purchases.

4. Make a copy of query *qryInventory04-MakeCostOfEndInvTable* and name it *qryCoGS03-MakeCostOfGASTable*. Click *Make Table* in the Query Type group and change the table name to *tblCoGS03-ComputeCostOfGAS*. Save and close the query.
5. Make a copy of *qryInventory05-MakeTableForFirstQtyUnassigned* and name it *qryCoGS04-MakeTableForFirstQtyUnassigned*. Open *qryCoGS04-* in Design view. Add *qryCoGS02-QtyAvailableForSale* to the Table Pane. Change the Table for *QuantityOnHand* to *qryCoGS02-* and change the Field to *QtyAvailableForSale*. Delete *qryInventory03-QtyOnHandByItem* from the Table Pane. Link *tblInflow-PurchaseInventory* to *qryCoGS02-* by *InventoryID*. Click *Make Table* in the Query Type group and change the table name to *tblCoGS03-ComputeCostOfGAS* (see Figure 12.39). Save and close the query.
6. Create two queries to update the quantities unassigned and assigned for the most recent purchase using quantity of GAS. Use Figure 12.40 and Exercise 12.12, steps 7 and 8, to create *qryCoGS05-ComputeFirstQtyUnassigned* and *qryCoGS06-UpdateFirstQtyAssigned*.
7. Use Figure 12.41 and Exercise 12.13, Step 1, to create a query to sum quantities assigned by item. Save the query as *qryCoGS07-UpdateFirstQtyAssigned*.
8. Use Figure 12.42 and Exercise 12.13, Step 2, to create *qryCoGS08-UpdatedQtyUnassigned*.
9. Use Figure 12.43 and Exercise 12.13, Step 3, to create the Make Table query *qryCoGS09-MakeTableForUpdatedQtyUnassigned*.
10. Create two Update queries to update *QtyUnassigned* and *QtyAssigned* in *tblCoGS-ComputeCostOfGAS*. The queries are similar to the ones you created in Step 7. Use Figure 12.44 as a guide. Name the queries *qryCoGS10-UpdateNextOldestQtyUnassigned* and *qryCoGS11-UpdateNextOldestQtyAssigned*.

Fig. 12.37
qryCoGS01-SumQtySoldByItem
in Design view.



Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.38
qryCoGS02-QtyAvailableForSale
 in Design view.

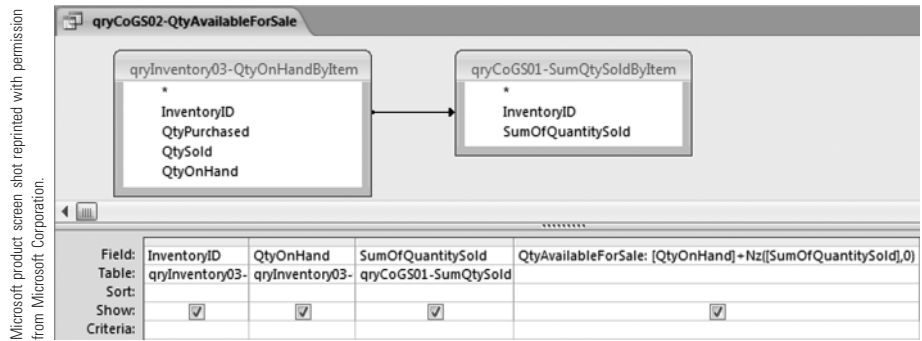


Fig. 12.39
qryCoGS04-MakeTable-ForFirstQtyUnassigned
 in Design view.

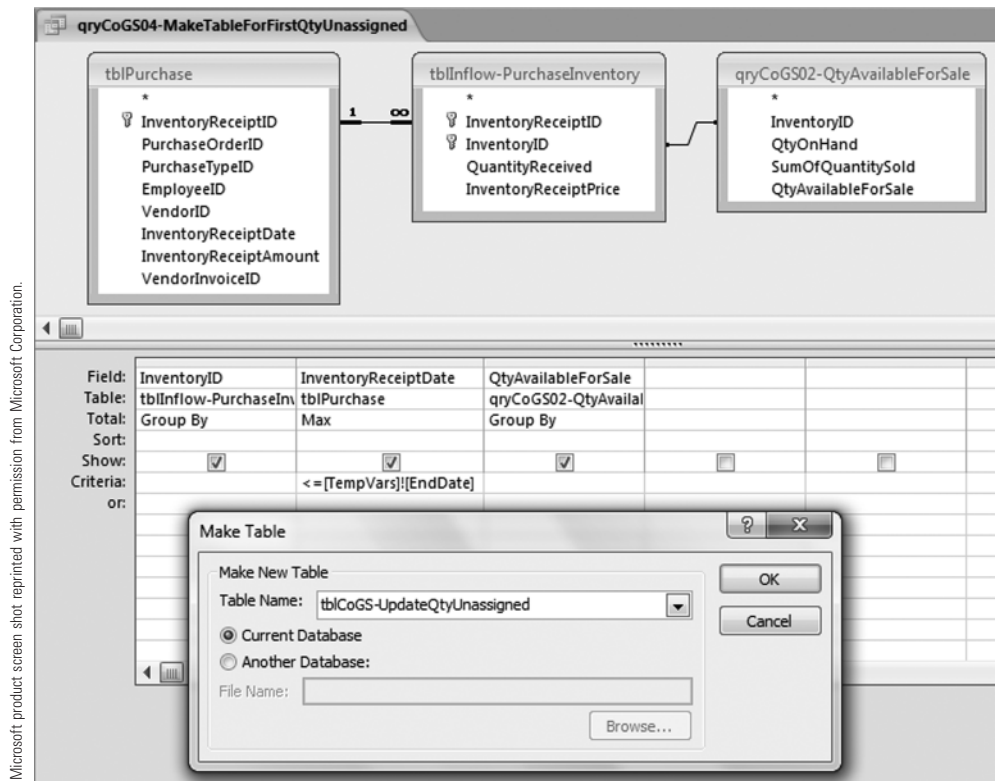


Fig. 12.40
qryCoGS05-Update-FirstQtyUnassigned
 and *qryCoGS06-UpdateFirstQty-Assigned* in Design view.

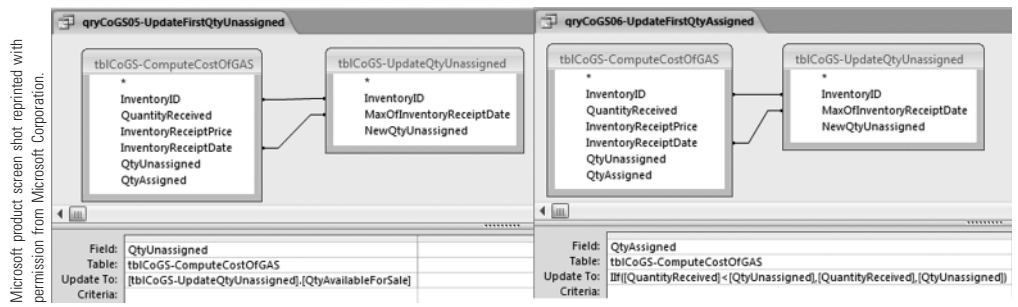


Fig. 12.41
qryCoGS07-SumOfQuantityAssigned
 in Design view.

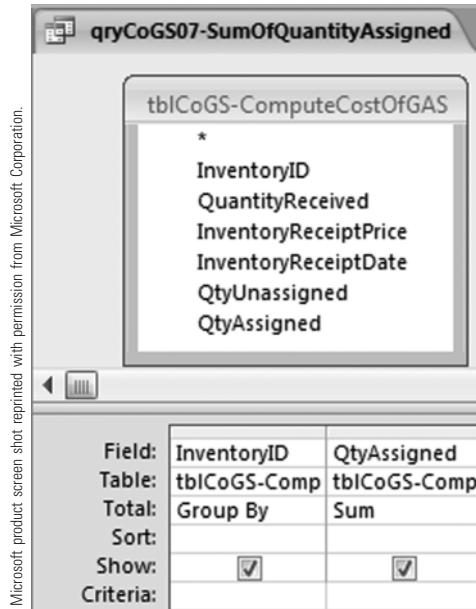


Fig. 12.42
qryCoGS08-UpdateQty-Unassigned in Design view.

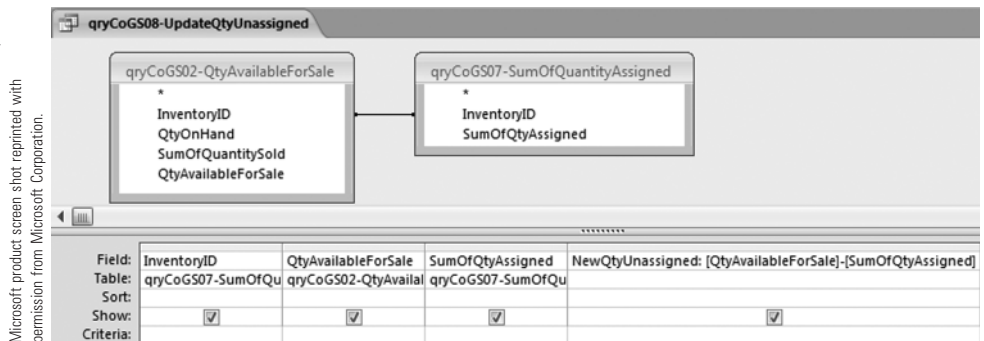


Fig. 12.43
qryCoGS09-MakeTableForUpdatedQtyUnassigned in Design view and Form view.

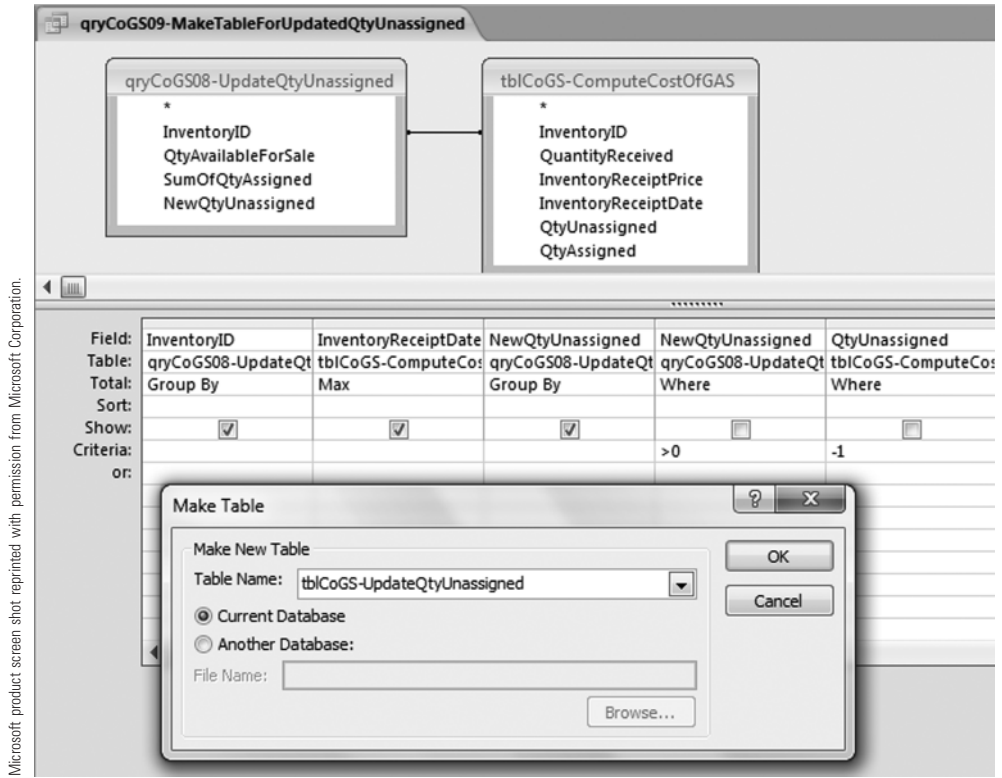
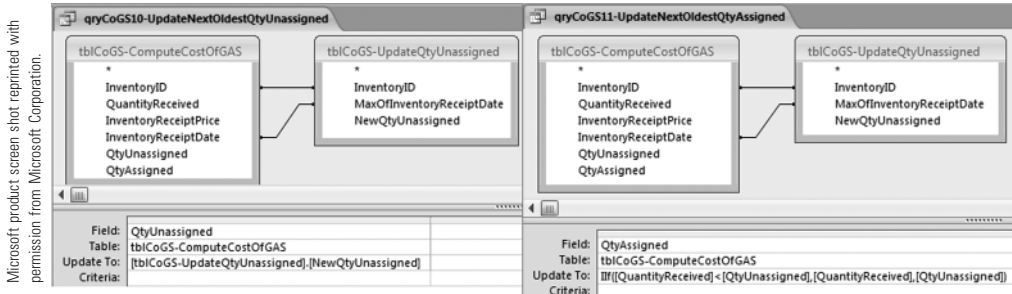


Fig. 12.44
qryCoGS10-UpdateNextOldestQtyUnassigned and *qryCoGS11-UpdateNextOldestQtyAssigned* in Design view.



EXERCISE 12.16: COMPLETING THE COMPUTATION FOR COST OF GOODS SOLD

1. Create a macro to run *qryCoGS07*-, *qryCoGS08*-, *qryCoGS09*-, *qryCoGS10*-, and *qryCoGS11*-. Click the **Create** tab and click **Macro** in the Other group. Click **Show All Actions** in the Show/Hide group. Add a **SetWarnings** Action. Add three **Open Query** Actions as indicated in Figure 12.45 to run *qryCoGS09*-, *qryCoGS10*-, and *qryCoGS11*-. Save the macro as *mcrCoGS02-UpdateComputeGAS*Table. Close the macro.
2. Create the main CoGS macro. Click the **Create** tab and click **Macro** in the Other group. Use Figure 12.46 as a guide. Add four **OpenQuery** Actions for *qryCoGS03*- through *qryCoGS06*-. Save the macro as *mcrCoGS01-CoGS*.
3. Add a Macro Action to keep running *mcrCoGS02*- until *tblCoGS-ComputeCostOfGAS* is completely updated. Type **RunMacro** in the next available

- Action cell and select `mcrCoGS02-UpdateComputeGASTable` in the Macro Name cell in the Action Arguments pane. Enter `DCount("[InventoryID"],"[tblCoGS-UpdateQtyUnassigned]")` in Repeat Expression. Save and close the macro.
- Run `mcrTempDate` using `3/1/2010` and `3/31/2010` as the beginning and ending dates and Run `mcrCoGS01-`. Open `tblCoGS-UpdateQtyUnassigned`. There should not be any records in this table. Close the table. Now open `tblCoGS-ComputeCostOfGAS`. It should look like the datasheet in Figure 12.47. Scroll through it to see that all inventory available for sale have been properly assigned to the most recent purchases for each item.
 - Create a new query in Design view to compute the GAS line extensions. Add `tblCoGS-ComputeCostOfGAS` to the Table Pane. Add `InventoryID`, `InventoryReceiptDate`, and `QtyAssigned` to the Criteria Pane. Save your query as `qryCoGS12-GASLineExtension`. Enter the expression to compute the line extension in the next column of the Criteria Pane: `GASLineExtension: [InventoryReceiptPrice]*[QtyAssigned]`. Save and close the query.
 - Sum the line extensions to arrive at the GAS balance. Create a new query in Design view. Add `qryCoGS12-GASLineExtension` to the Table Pane. Add `GASLineExtension` to the Criteria Pane. Click `Totals` in the Show/Hide group. Change the Total cell to `Sum`. Open the Property Sheet and enter `Goods Available for Sale` as the Caption property. Save your query as `qryCoGS13-GoodsAvailableForSale`. Close the query.
 - Create a new query in Design view to compute the cost of goods sold balance. Use Figure 12.48 as a guide. Enter the following expression for cost of goods sold: `CostOfGoodsSold: [SumOfGASLineExtension]-Nz([SumOfInvOnHandLineExtension],0)`. Open the Property Sheet and enter `Cost of Goods Sold` as its Caption property. Save your query as `qryCoGS14-CostOfGoodsSold` and close the query.
 - Open `mcrCoGS01-CoGS` in Design view to add the last steps to the macro as shown in Figure 12.46. Save the macro.
 - Test your CoGS queries and macros. Run `mcrTempDate` using `2/1/2010` and `2/28/2010` as the beginning and ending dates. Then run `mcrInventory01-EndingInventory` and `mcrCoGS01-CoGS`, respectively. Why is it necessary to run `mcrInventory01-` to compute CoGS? Then double-click `qryCoGS14-CostOfGoodsSold` to verify a cost of goods sold of `$27,144.68`.

Fig. 12.45
mcrCoGS02-Update-ComputeGASTable in Design view.

mcrCoGS02-UpdateComputeGASTable	
Action	Arguments
SetWarnings	No
OpenQuery	qryCoGS09-MakeTableForUpdatedQtyUnassigned, Datasheet, Edit
OpenQuery	qryCoGS10-UpdateNextOldestQtyUnassigned, Datasheet, Edit
OpenQuery	qryCoGS11-UpdateNextOldestQtyAssigned, Datasheet, Edit

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.46
mcrCoGS01-CoGS in Design view.

mcrCoGS01-CoGS	
Action	Arguments
SetWarnings	No
OpenQuery	qryCoGS03-MakeCostOfGASTable, Datasheet, Edit
OpenQuery	qryCoGS04-MakeTableForFirstQtyUnassigned, Datasheet, Edit
OpenQuery	qryCoGS05-UpdateFirstQtyUnassigned, Datasheet, Edit
OpenQuery	qryCoGS06-UpdateFirstQtyAssigned, Datasheet, Edit
RunMacro	mcrCoGS02-UpdateComputeGASTable, , DCount("[InventoryID"],"[tblCoGS-UpdateQtyUnassigned])
OpenQuery	qryInventory14-EndingInventory, Datasheet, Edit
OpenQuery	qryCoGS14-CostOfGoodsSold, Datasheet, Edit
Close	Query, qryInventory14-EndingInventory, Yes
Close	Query, qryCoGS14-CostOfGoodsSold, Yes

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.47 *tblCoGS-ComputeCostOfGAS* in Datasheet view for the period between 3/1/2010 and 3/31/2010.

InventoryID	QuantityRec	InventoryRe	InventoryRe	QtyUnassigned	QtyAssigned
1001	60	\$13.20	2/19/2010	88	60
1001	20	\$13.00	2/9/2010	28	20
1001	75	\$13.00	1/25/2010	8	8
1001	75	\$13.20	1/7/2010	-1	0
1002	30	\$16.00	3/8/2010	115	30
1002	40	\$15.95	2/19/2010	85	40
1002	65	\$16.00	2/9/2010	45	45
1002	60	\$16.00	1/25/2010	-1	0
1002	25	\$15.95	1/7/2010	-1	0
1003	35	\$20.89	2/19/2010	55	35
1003	15	\$21.00	2/9/2010	20	15
1003	50	\$21.00	1/25/2010	5	5
1003	15	\$20.89	1/7/2010	-1	0
1004	30	\$26.30	2/22/2010	43	30
1004	15	\$26.30	2/6/2010	13	13
1004	25	\$26.30	1/29/2010	-1	0
1004	15	\$26.30	1/17/2010	-1	0
1005	40	\$33.20	2/22/2010	75	40
1005	30	\$33.20	2/6/2010	35	30
1005	15	\$33.25	1/29/2010	5	5
1006	10	\$39.20	2/6/2010	25	10
1006	10	\$39.25	1/29/2010	15	10
1006	35	\$39.25	1/17/2010	5	5
1007	30	\$24.50	3/8/2010	108	30
1007	20	\$24.75	2/19/2010	78	20

Record: 1 of 172 No Filter Search

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.48 *qryCoGS14-CostOfGoodsSold* in Design view.

Field:	SumOfGASLineExtension	SumOfInvOnHandLineExtension	CostOfGoodsSold: [SumOfGASLineExtension]-Nz([SumOfInvOnHandLineExtension],0)
Table:	qryCoGS13-GoodsAvailable	qryInventory14-EndingInventory	
Sort:			
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:			

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

The ending inventory and cost of goods sold computations are now complete.

Creating Income Statement and Balance Sheet Reports

In Chapters 8 through 11, you derived financial statement balances related to the sales/collection process, acquisition/payment process, HR process, and the financing process. Additionally, in Chapter 12, you derived financial statement balances related to support activities as well as balances that span multiple business processes. Now, you are finally ready to put them all together in an income statement report and a balance sheet report. One of the advantages of using a database for an accounting information system is that these reports can be run for any period of time.

Creating an Income Statement Report

We can use Report Wizard to help us create an income statement report. To use the wizard effectively, we will create an income-statement query. This query contains all the information we need other than the support expenses. The support expenses are in a form that lends itself to using a subreport, which is like a subform. Therefore, we will add a subreport for the support expenses after creating the main income-statement report. Additionally, we need a query to compute the sum of the support expenses so that we can use it in the income statement query to compute net income. Queries from prior end-of-chapter problems that are required for the income statement are included in your Chapter 12 database.

EXERCISE 12.17: CREATING QUERIES FOR THE INCOME STATEMENT REPORT

1. Create a new query in Design view to sum the support expenses. Add `qryNonInvExp05-SumImmediatesAndPrepays` to the Table Pane. Add `IncomeStatementExpense` to the Criteria Pane. Click **Totals** in the Show/Hide group. Change the Total cell to **Sum**. Save your query as `qryNonInvExp06-TotalNonInventoryExpenses`.
2. Replace the date criteria in the following queries from Chapters 8 through 11 that will be used in the income statement query with the temporary variables created in `mcrTempDate` to `Between [TempVars]![BeginDate] And [TempVars]![EndDate]`:
 - `qrySales1-Sales` (Chapter 8): Change the Criteria for `ShippingDate`.
 - `qryGrossPay` (Chapter 10), which is used by `qryPayrollExpenses`: Change the Criteria for `LAPayPeriodEnded`.
 - `qryInterestExpense` (Chapter 11 Practice Exercise 2; see Figure 12.49): Change the Criteria for `CashDisbursementDate`.
3. Create a new query in Design view for the income statement report. Add the following queries to the Table Pane: `qrySales1-Sales`, `qryCoGS14-CostOfGoodsSold`, `qryPayrollExpenses`, `qryNonInvExp06-TotalNonInventoryExpenses`, and `qryInterestExpense`. Add `SumOfSaleAmount` from `qrySales1-`, and `CostOfGoodsSold` from `qryCoGS14-` to the Criteria Pane. Save the query as `qryIncomeStatement`.
4. Compute gross profit in the next column within the Criteria Pane: `GrossProfit: [SumOfSaleAmount]-[CostOfGoodsSold]`. Open the Property Sheet and set the Caption property to `Gross Profit`.
5. Add all of the operating expenses to the Criteria Pane: `SumOfGrossPay`, `SumOfFICA`, and `SumOfMedicare` from `qryPayrollExpenses`; and `SumOfIncomeStatementExpense` from `qryNonInvExp06-`. Save your changes. Use Expression Builder to compute total operating expenses: `TotalOperatingExpenses: [SumOfFICA]+[SumOfGrossPay]+[SumOfMedicare]+[SumOfIncomeStatementExpense]`. Set the Caption property to `Total Operating Expenses`. Save your changes.
6. Use Expression Builder in the next column within the QBE grid (another name for Criteria Pane) to compute operating income: `OperatingIncome: [GrossProfit]-[TotalOperatingExpenses]`. In the Property Sheet, set the Caption property to `Operating Income`.
7. Add `SumOfInterestAmount` from `qryInterestExpense` to the QBE grid. Save the query. Use Expression Builder in the next column within the QBE grid to compute net income: `NetIncome: [OperatingIncome]-Nz([SumOfInterestAmount])`. In the Property Sheet, set the Caption property to `Net Income`. Save your changes.
8. Test your query. Run `mcrTempDate` using `3/1/2010` and `3/31/2010` as the beginning and ending dates. Then double-click `qryIncomeStatement`. You should have the same dynaset, as shown in Figure 12.50.

Fig. 12.49
qryInterestExpense in Design view.

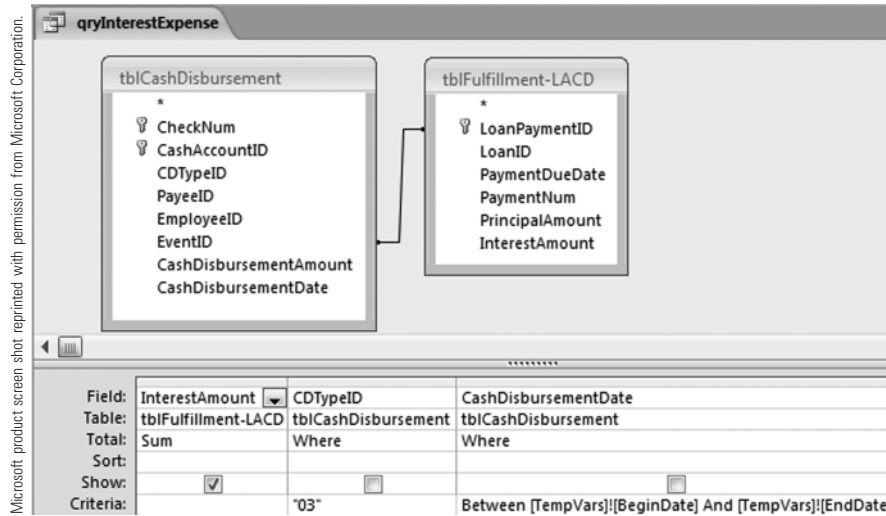


Fig. 12.50
qryIncomeStatement in Design view.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Sales	Cost of Goods	Gross Profit	Wages Exp	FICA Tax	Medicare	Other Op	Total Operatin	Operating Inc	Interest	Net Income
\$26,208.55	\$15,828.01	\$10,380.54	\$15,531.10	\$962.90	\$225.18	\$4,352.05	\$21,071.23	(\$10,690.69)	\$963.54	(\$11,654.23)

EXERCISE 12.18: CREATING THE INCOME STATEMENT REPORT AND SUBREPORT USING REPORT WIZARD

Create the *rptIncomeStatement* using Report Wizard.

1. Click *qryIncomeStatement* in the Unassigned Objects section of the Navigation Pane. Click the **Create** tab and click **Report Wizard** in the Reports group. Click **>>** to select all available fields.
2. Click **Next** three times, which will take you to the layout dialog box. Click **Columnar** from the Layout options. Leave the Orientation set to **Portrait**, and leave the box checked to adjust the field width to fit all fields on one page.
3. Click **Next** twice to take you to the last dialog box. Name your report *rptIncomeStatement*. Click **Finish**. Click **Close Print Preview** to return to Design view. Your report should look like the one in Figure 12.51. Close the report.

Create a subreport for the support expenses using Report Wizard.

4. Click *qryNonInvExp05-ImmediatesAndPrepays* in the Unassigned Objects section of the Navigation Pane. Click the **Create** tab and click **Report Wizard** in the Reports group. Double-click *IncomeStatementName* and *IncomeStatementExpense* to select them.
5. Click **Next** twice, which will take you to the sorting dialog box. Sort by *IncomeStatementName* in **Ascending** order.
6. Click **Next** three times to take you to the last dialog box. Name your report *rsubIncomeStatement*. Click **Finish**. Notice that all of the support expenses are displayed. Click **Close Print Preview** to return to Design view. Your report should look like the one in Figure 12.52. Close the report.

Fig. 12.51
rptIncomeStatement in Design view after running Report Wizard.

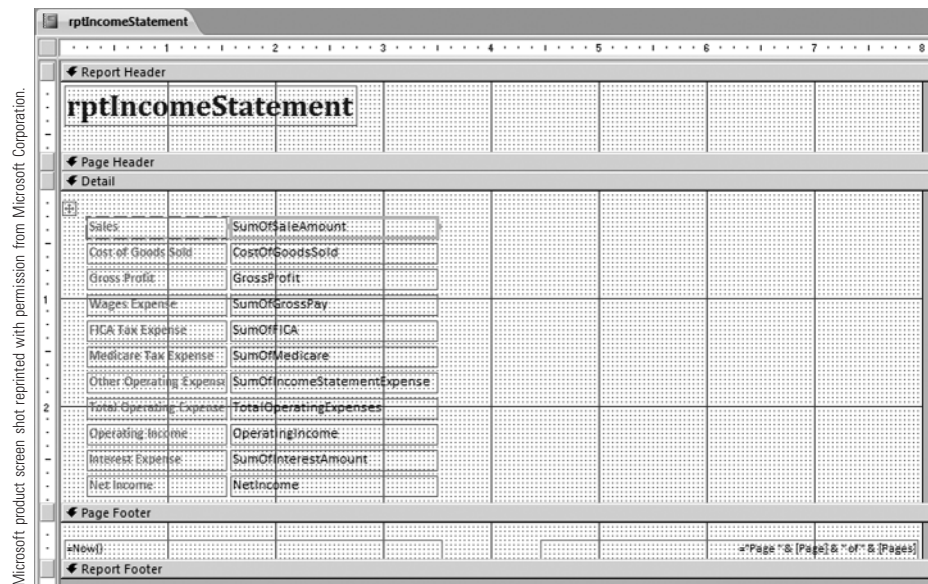
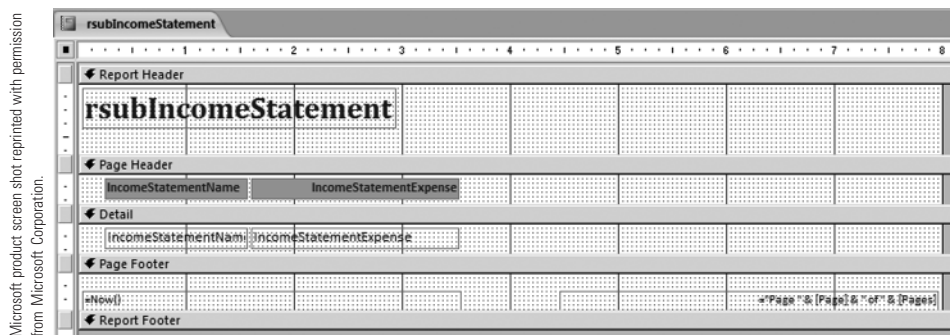


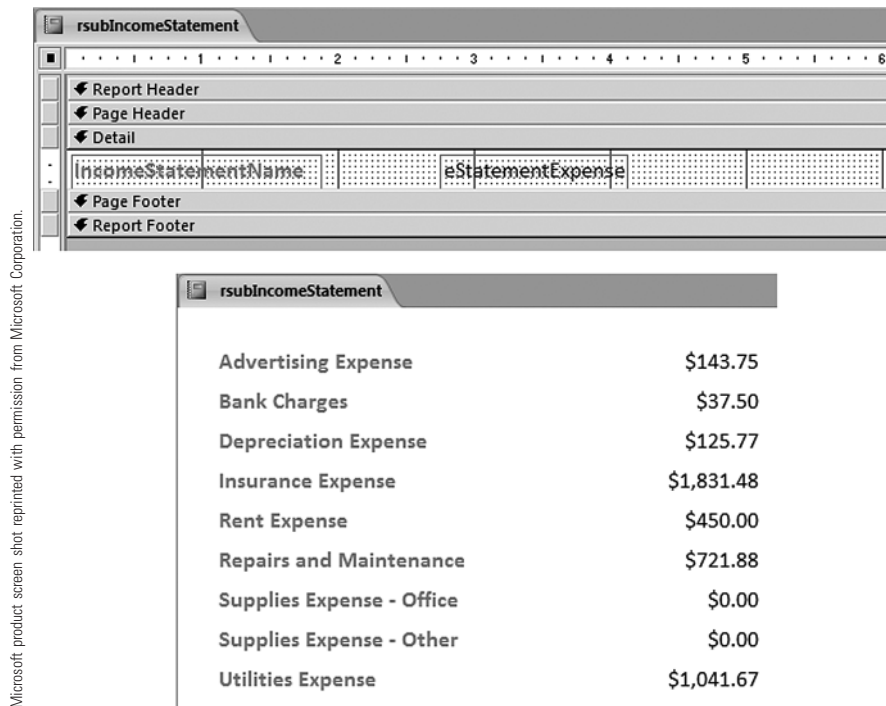
Fig. 12.52
rsubIncomeStatement in Design view after running Report Wizard.



EXERCISE 12.19: COMPLETING THE INCOME STATEMENT SUBREPORT

1. Open *rsubIncomeStatement* in Design view. Use Figure 12.53 as a guide. Press **Ctrl+A** to select all labels and controls in the report. Click the **Arrange** tab and click **Remove** in the Control Layout group.
2. Open the Property Sheet. **Report** should be displayed in the Property Sheet combo box. Click the **Format** tab and change the value of Scroll Bars to **None** to prevent them from being displayed in the main report.
3. Delete the title in the Report Header. Delete the Report Header area by clicking and dragging the top of the **Page Header** bar up until it meets the **Report Header** bar. Delete the two labels in the Page Header. Remove the Page Header by clicking and dragging the top of the **Detail** bar up until it meets the **Page Header** bar. Delete the two controls in the Page Footer and remove the Page Footer.
4. Change the font on both controls to **Calibri** and the font size to **12**. **Left Justify** (Text Align on the Format tab of the Property Sheet) **IncomeStatementName** and **Right Justify** **IncomeStatementExpense**. You can either use the **Font** section on the **Home** ribbon or use the **Format** tab on the Property Sheet. Change the font color for **IncomeStatementName** to **#5C83B4**, the same color as the labels in *rptIncomeStatement*. Save and close the report.

Fig. 12.53
Completed *rsubIncomeStatement* in Design view and Report view for the period between 3/1/2010 and 3/31/2010.



EXERCISE 12.20: COMPLETING THE INCOME STATEMENT REPORT

1. Open *rptIncomeStatement* in Design view. Use Figures 12.54 and 12.55 to guide you. Click and drag the right edge of the Report Detail to make it 7-1/2 inches wide. Press **Ctrl+A** to select all labels and controls in the report. Click the **Arrange** tab and click **Remove** in the Control Layout group. Delete the two controls in the Page Footer and remove the Page Footer.
2. Change the font size of the Report Header title to **16**. Change the title to *Pipefitters Supply Company* and Center Justify it. Also, center the title in the Report Header. Drag the top of the **Page Header bar** down until the Report Header area is 1-1/8 inches tall (one line past the 1-inch mark). Make a copy of the Pipefitters Supply Company title and place it directly below the original. Change the name to *Income Statement*. Make a second copy of the title and change the text to *For the Period between*. Make one last copy of the title. Change the name to *and* and resize the control box to just fit the word inside it.
3. Add the temporary variables for the dates to the Report Header. Click **Text Box** in the Controls group. Click inside the Report Header to create a text box control. Delete the label. Open the Property Sheet and type `= [TempVars]![BeginDate]` in the Record Source box. Make a copy of this control and change its Record Source to `= [TempVars]![EndDate]`. Arrange the titles and controls as they appear in Figure 12.54 and 12.55. Make sure all of the text is Center Justified. Save your changes to the report.
4. Select all of the labels and controls in the Detail section. Change the font size to **12**. Left Justify the labels and Right Justify the controls. Arrange the labels and controls

as they appear in Figures 12.54 and 12.55. Delete the **OtherOperatingExpenses** label and control. Make the gap between Medicare Tax Expense and TotalOperatingExpenses about 1-1/8 inches by moving the last three items down. This is where you will insert the subreport. Use the Line tool in the Controls group to create the horizontal lines. If you cannot see the line in Report view, change the Border Style on the Format tab of the Property Sheet to **Solid** and the Border Color to **#000000** (black). You only need to create one line and then copy it to create the other lines.

5. Insert the subreport. Click **Subform/Subreport** in the Controls group. Click and drag from just below the bottom left corner of the **Medicare Tax Expense** label to just above the top right corner of the **OperatingExpense** control. In the first dialog box, select **Use an existing report or form**. Click **rsubIncomeStatement** and click **Next**. Select **Define my own** in the linking dialog box and click **Next**. Then click **Finish**. Delete the subform label by clicking it and pressing the **Delete** key. On the Format tab of the Property Sheet, change Border Style to **Transparent**.
6. To test the report, run **mcrTempDate** for the period between **3/1/2010** and **3/31/2010**. Then run **mcrInventory01-** and **mcrCoGS01-** before opening the report in Report view. Close the report.

Fig. 12.54
Completed
rptIncomeStatement in
Design view.

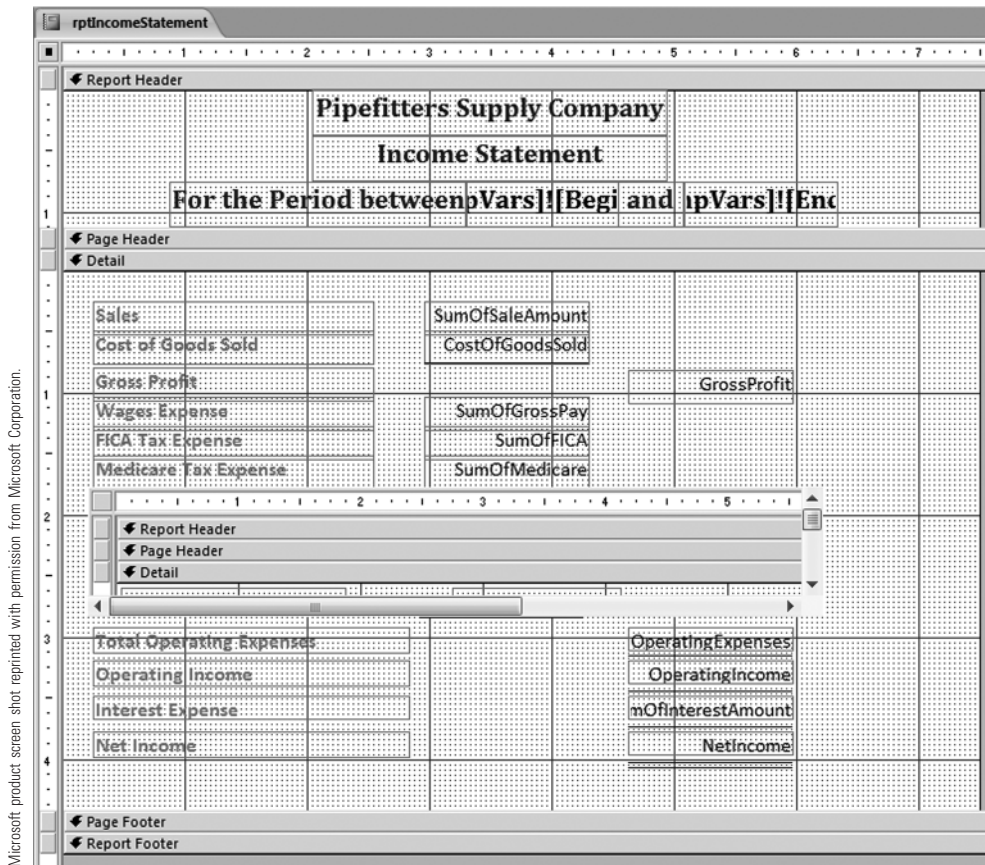


Fig. 12.55
Completed
rptIncomeStatement in
Report view for the
period between
3/1/2010 and
3/31/2010.

Pipefitters Supply Company		
Income Statement		
For the Period between 3/1/2010 and 3/31/2010		
Sales	\$26,208.55	
Cost of Goods Sold	\$15,828.01	
Gross Profit		\$10,380.54
Wages Expense	\$15,531.10	
FICA Tax Expense	\$962.90	
Medicare Tax Expense	\$225.18	
Advertising Expense	\$143.75	
Bank Charges	\$37.50	
Depreciation Expense	\$125.77	
Insurance Expense	\$1,831.48	
Rent Expense	\$450.00	
Repairs and Maintenance	\$721.88	
Supplies Expense - Office	\$0.00	
Supplies Expense - Other	\$0.00	
Utilities Expense	\$1,041.67	
Total Operating Expenses		\$21,071.23
Operating Income		(\$10,690.69)
Interest Expense		\$963.54
Net Income		(\$11,654.23)

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Creating a Balance Sheet Report

The balance sheet report that we will create for Pipefitters Supply Company will have three subreports: assets, liabilities, and equity. We will use the financial statement queries from Chapters 8 through 11. We will also use queries from this chapter for cash, ending inventory, prepaid expenses, and fixed assets. Before we build the subreports, we will start by creating separate queries for assets, liabilities, and equity. We included queries from prior end-of-chapter problems that are required for the balance sheet in the Chapter 12 database.

EXERCISE 12.21: CREATING QUERIES FOR THE ASSETS BALANCE SHEET SUBREPORT

1. To make the asset subreport easier to build, we will create a new query to sum all of the prepaid expenses. Create a new query in Design view. Add `qryPrepaidExpenses02-PrepaidExpenseBalances` to the Table Pane. Add `SumOfPrepaidExpense` to the QBE grid. Click **Totals** in the Show/Hide group. Change the Total cell to `Sum`. Open the Property Sheet and type `Prepaid Expenses` as the Caption. Save your query as `qryPrepaidExpenses04-PrepaidExpenseBalance`. We use *Balance* instead of *Balances* to indicate that this query produces the balance sheet line item Prepaid Expenses.

- Change the date criteria in the following queries used to compute accounts receivable, in Chapter 8, to $\leq [\text{TempVars}]![\text{EndDate}]$.
 - `qryAccRec1-SumOfSales` (Chapter 8): Change the Criteria for `ShippingDate`.
 - `qryAccRec2-SumOfCR` (Chapter 8): Change the Criteria for `CashReceiptDate`.
- Create a new query in Design view for the asset subreport. Add the following queries to the Table Pane: `qryCash3-CashBalance`, `qryAccRec3-AccountsReceivable`, `qryInventory14-EndingInventory`, `qryPrepaidExpenses04-PrepaidExpenseBalance`, and `qryPrepaidExpenses03-FixedAssets`. Add the following fields to the QBE grid:
 - `CashBalance` from `qryCash3-`.
 - `AccountsReceivable` from `qryAccRec3-`.
 - `SumOfInvOnHandLineExtension` from `qryInventory14-`.
 - `SumOfSumOfPrepaidExpense` from `qryPrepaidExpenses04-`.
 - `SumOfPurchaseLineExtension`, `SumOfAccumAmortAndDepr`, and `SumOfPrepaidExpense` from `qryPrepaidExpenses03-`.
 Save the query as `qryBalanceSheet01-Assets`.
- Use Expression Builder to compute total assets in the next column of the QBE grid: `TotalAssets: [CashBalance]+[AccountsReceivable]+[SumOfInvOnHandLineExtension]+[SumOfSumOfPrepaidExpense]+[SumOfPrepaidExpense]`. Open the Property Sheet and set the Caption property to `Total Assets`. Figure 12.56 shows the dynaset as of 3/31/2010. Notice that balance sheet items are for a specific point in time rather than for a period of time.

Fig. 12.56 Dynaset for `qryBalanceSheet01-Assets` as of 3/31/2010.

qryBalanceSheet01-Assets							
Cash	Accounts Rec	Inventory	Prepaid Exp	Property, Plant	Accumulated Depr	Property, Plant	Total Assets
\$311,830.70	\$11,874.05	\$19,439.51	\$17,589.99	\$7,551.08	\$243.64	\$7,307.44	\$368,041.69

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

EXERCISE 12.22: CREATING QUERIES FOR THE LIABILITIES BALANCE SHEET SUBREPORT

- Change the date criteria in the following queries, used in Chapters 9 through 11, to $\leq [\text{TempVars}]![\text{EndDate}]$:
 - `qryAccPay1-SumOfPurchases` (Chapter 9): Change the Criteria for `InventoryReceiptDate`.
 - `qryAccPay2-SumOfCD` (Chapter 9): Change the Criteria for `CashDisbursementDate`.
 - `qryWagesPayable1-GrossPay` (Chapter 10): Change the Criteria for `LAPayPeriodEnded`.
 - `qryWagesPayable5-SumOfCDPayroll` (Chapter 10): Change the Criteria for `CashDisbursementDate`.
 - `qryDividendsPayable1-DividendsDeclared` (Chapter 11 Practice Exercise 3): Change the Criteria for `DividendDeclarationDate`.
 - `qryDividendsPayable3-SumOfDividendsPaid` (Chapter 11 Practice Exercise 3): Change the Criteria for `CashDisbursementDate`.
 - `qryLoansPayable1-SumOfCDPrincipal` (Chapter 11 Practice Exercise 1): Change the Criteria for `CashDisbursementDate`.
 - `qryLoansPayable2-SumOfCR` (Chapter 11 Practice Exercise 1): Change the Criteria for `CashReceiptDate`.
- Create a new query in Design view for the liabilities subreport. Add the following queries to the Table Pane: `qryAccPay3-AccountsPayable`, `qryWagesPayable6-WagesPayable`, `qryPayrollTaxesPayable`, `qryDividendsPayable4-DividendsPayable`, and `qryLoansPayable3-LoansPayable`. Add the following fields to the QBE grid:
 - `AccountsPayable` from `qryAccPay3-`.
 - `WagesPayable` from `qryWagesPayable6-`.

- `PayrollTaxesPayable` from `qryPayrollTaxesPayable`.
- `DividendsPayable` from `qryDividendsPayable4-`.
- `LoansPayable` from `qryLoansPayable3-`.

Save the query as `qryBalanceSheet02-Liabilities`.

- Use Expression Builder to compute total assets in the next column of the QBE grid: `TotalLiabilities: [AccountsPayable]+[WagesPayable]+[PayrollTaxesPayable]+[DividendsPayable]+[LoansPayable]`. Open the Property Sheet and set the Caption property to `Total Liabilities`. Save your changes. Figure 12.57 shows the dynaset as of 3/31/2010.

Fig. 12.57 Dynaset for `qryBalanceSheet02-Liabilities` as of 3/31/2010.

qryBalanceSheet02-Liabilities					
Accounts Payable -	Wages Payable -	Payroll Taxes Payable -	Dividends Payable -	Loans Payable -	Total Liabilities -
\$2,458.79	\$13,141.57	\$9,866.33	\$0.00	\$144,607.72	\$170,074.41

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

EXERCISE 12.23: CREATING QUERIES FOR THE EQUITY BALANCE SHEET SUBREPORT

- Create a new query to compute the common stock issued by summing cash receipts related to stock issues. Create a new query in Design view. Use Figure 12.58 as a guide. Add `tblCashReceipt` to the Table Pane. Add `CashReceiptAmount`, `CRTTypeID`, and `CashReceiptDate` to the QBE grid.
- Click `Totals` in the Show/Hide group. Change the Total cell for `CashReceiptAmount` to `Sum`, and change the Total cell for the other two fields to `Where`. Open the Property Sheet and type `Dividends Payable` as the Caption for `CashReceiptAmount`. Set the Criteria for `CRTTypeID` to "03" in order to select cash receipts from stock issues. Set the Criteria for `CashReceiptDate` to `<=[TempVars]![EndDate]`. Save your query as `qryCommonStock`. Your total for common stock on 3/31/2010 should be `$240,000.00`.
- Create a new query in Design view for the equity subreport. Add the following queries to the Table Pane: `qryBalanceSheet01-Assets`, `qryBalanceSheet02-Liabilities`, and `qryCommonStock`. Add the following fields to the QBE grid:
 - `TotalAssets` from `qryBalanceSheet01-`.
 - `TotalLiabilities` from `qryBalanceSheet02-`.
 - `SumOfCashReceiptAmount` from `qryCommonStock`.
 Save the query as `qryBalanceSheet03-Equity`.
- Use Expression Builder to compute retained earnings in the next column of the QBE grid: `RetainedEarnings: [TotalAssets]-[TotalLiabilities]-[SumOfCashReceiptAmount]`. Open the Property Sheet and set the Caption property to `Retained Earnings`. Save your changes. You can verify the retained earnings balance of `($42,032.72)` in Figure 12.59, as of 3/31/2010, by subtracting total dividends paid through 3/31/2010 from net income for the period between 1/1/2010 and 3/31/2010. Before running `rptIncomeStatement`, you first need to run `mcrTempDate` (1/1/2010 and 3/31/2010 as begin date and end date respectively), `mcrInventory01-` and `mcrCoGS01-`, respectively. Run `rptIncomeStatement` to obtain the net loss of `($20,032.72)`, and run `PE5-3qryDividends-DividendsPaid` to obtain dividends paid of `$22,000.00`.
- Use Expression Builder to compute total equity in the next column of the QBE grid: `TotalEquity: [SumOfCashReceiptAmount]+[RetainedEarnings]`. Open the Property Sheet and set the Caption property to `Total Equity`. Save your changes.
- Use Expression Builder again to compute total liabilities and equity in the next column of the QBE grid: `TotalLiabilitiesAndEquity: [TotalLiabilities]+[TotalEquity]`. Open the Property Sheet and set the Caption property to `Total Liabilities and Equity`. Save your changes. Figure 12.59 shows the dynaset as of 3/31/2010.

Fig. 12.58
qryCommonStock in
Design view.

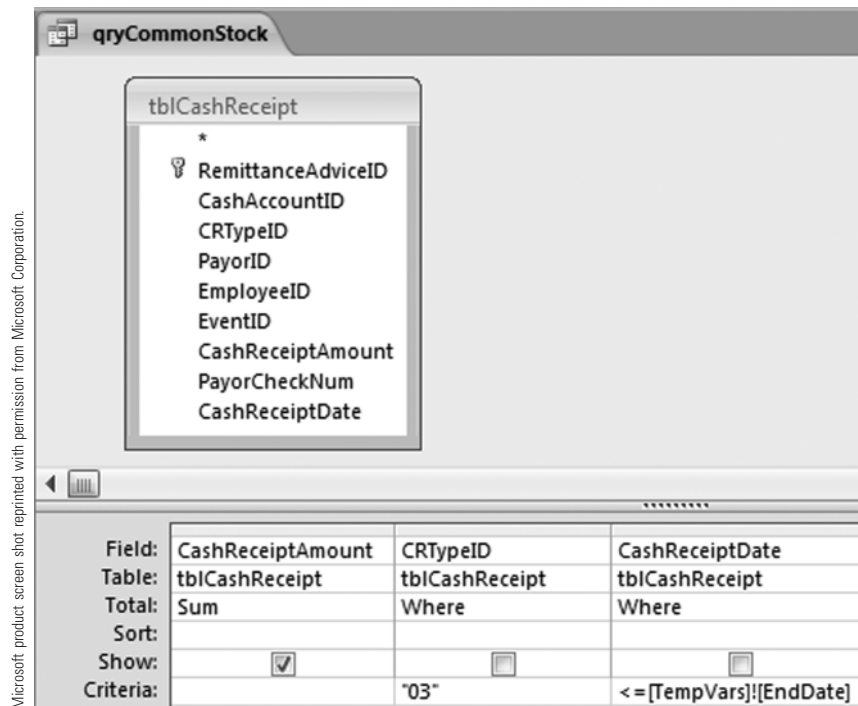


Fig. 12.59 Dynaset
for *qryBalanceSheet03-Equity*
as of 3/31/2010.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Total Assets	Total Liabilities	Common Stock	Retained Earnings	Total Equity	Total Liabilities and Equity
\$368,041.69	\$170,074.41	\$240,000.00	(\$42,032.72)	\$197,967.28	\$368,041.69

EXERCISE 12.24: CREATING THE BALANCE SHEET SUBREPORTS

Create a subreport for the asset section of the balance sheet.

1. Click [qryBalanceSheet01-Assets](#) and start **Report Wizard**. Click **>>** to select all available fields.
2. Click **Next** three times and click **Columnar** for the Layout option. Click **Next** twice to take you to the last dialog box. Name your report **rsubBalanceSheet-Assets**. Click **Finish**.
3. Switch to Design view. Use Figure 12.60 and Exercise 12.19 to help you transform the report into the finished product.
 - The Font for **Assets** is **14** and **Underlined**.
 - Set the Format of **Accumulated Depreciation** to **Currency**.
 - Set the Scroll Bars property for **Report** to **Neither**.

Repeat steps 1 through 3 to create a subreport for the liabilities section of the balance sheet.

4. Use [qryBalanceSheet02-Liabilities](#) as the query for the subreport. Name the subreport **rsubBalanceSheet-Liabilities**. Figure 12.61 shows the completed subreport in Design view and Report view.

Repeat steps 1 through 3 to create a subreport for the equity section of the balance sheet.

5. Use [qryBalanceSheet03-Equity](#) as the query for the subreport. Name the subreport **rsubBalanceSheet-Equity**. Figure 12.62 shows the completed subreport in Design view and Report view.

Fig. 12.60
Completed
rsubBalanceSheet-Assets
in Design view
and Report view on
3/31/2010.

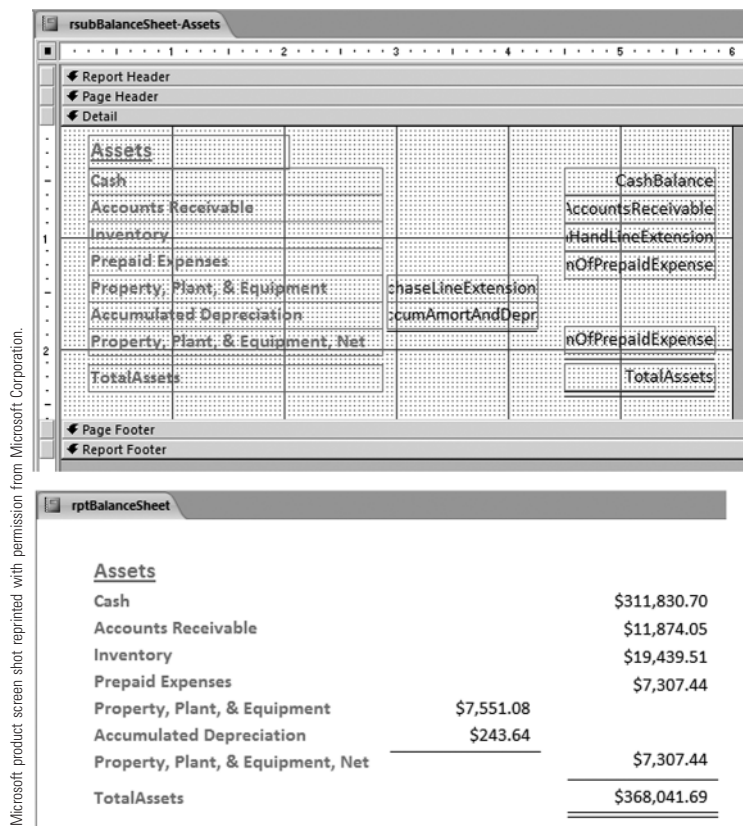


Fig. 12.61
Completed
rsubBalanceSheet-Liabilities
in Design view
and Report view on
3/31/2010.

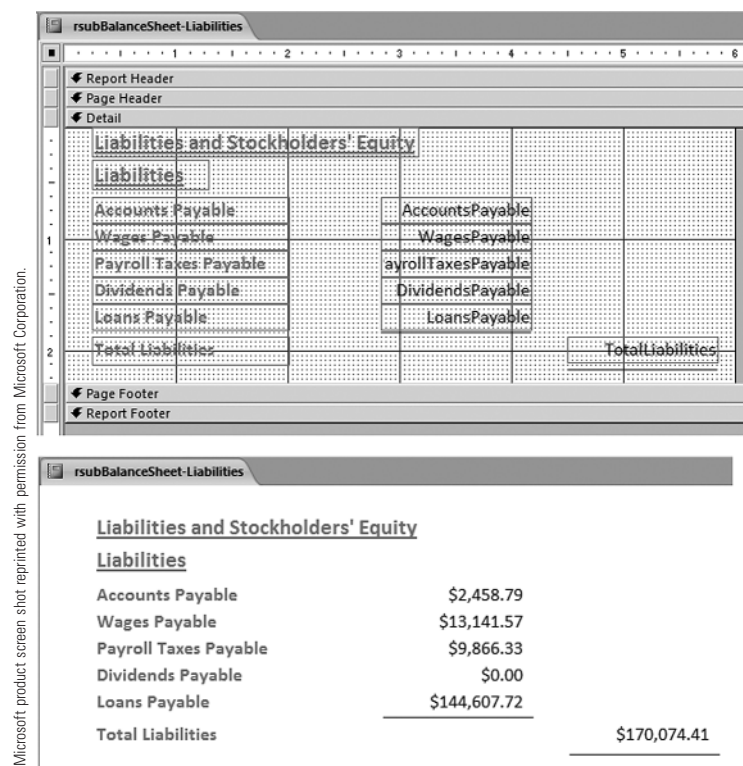


Fig. 12.62
Completed
rsubBalanceSheet-Equity
in Design view
and Report view on
3/31/2010.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Stockholders' Equity			
Common Stock		CashReceiptAmount	
Retained Earnings		RetainedEarnings	
Total Equity			TotalEquity
Total Liabilities and Equity			LiabilitiesAndEquity

Stockholders' Equity			
Common Stock		\$240,000.00	
Retained Earnings		(\$42,032.72)	
Total Equity			\$197,967.28
Total Liabilities and Equity			\$368,041.69

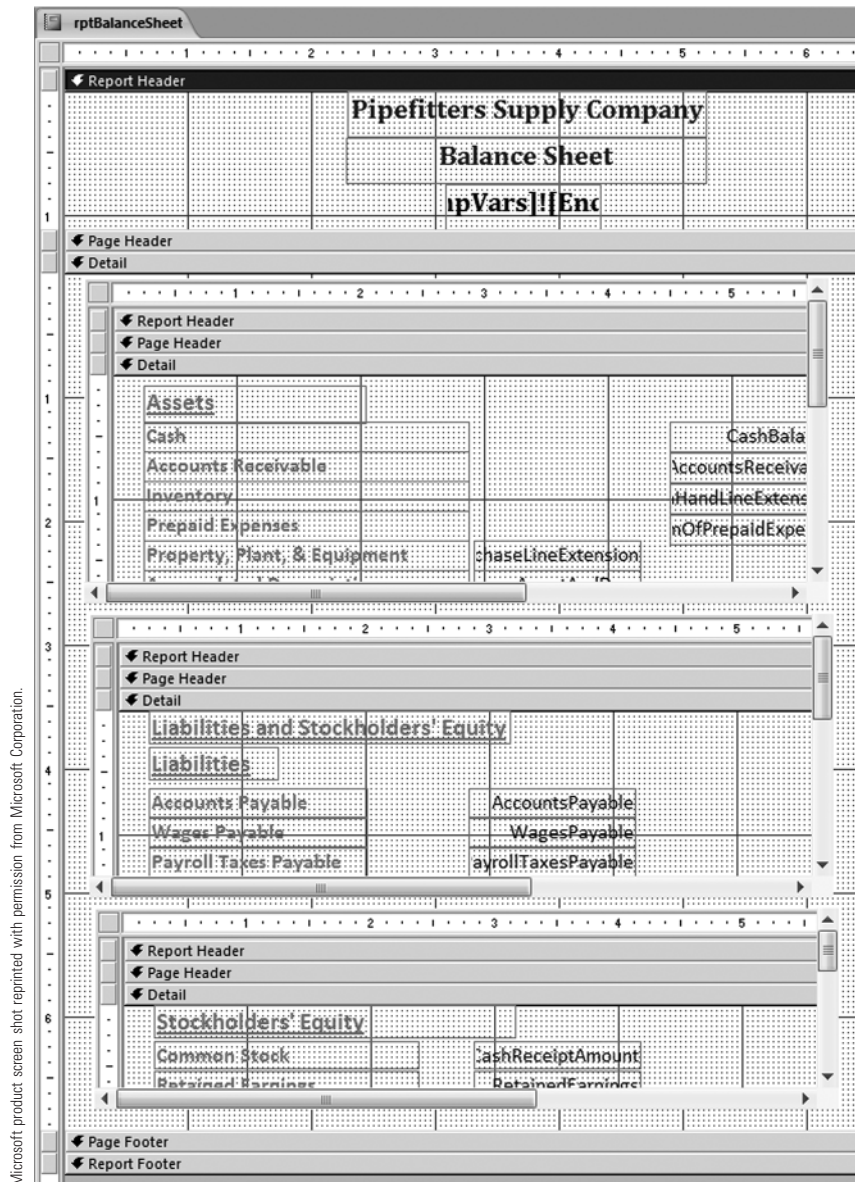
EXERCISE 12.25: CREATING THE BALANCE SHEET REPORT

1. Click **Blank Report** in the Reports group on the **Create** tab. Switch to Design view. Click **Title** in the Controls group to cause the Report Header to appear. Click the title and press the **Delete** key to delete it. Use the vertical scroll bar to move to the bottom of the report. Click and drag the bottom of the **Report Footer** area up to eliminate the Report Footer. Click and drag the top of the **Report Footer** bar up to eliminate the Page Footer. Save the report as **rptBalanceSheet**. Use Figures 12.63 and 12.64 as guides for the rest of this exercise.
2. Open **rptIncomeStatement** in Design view. Select all of the objects in the Report Header. Press **Ctrl+C** to copy them to the clipboard. Close **rptIncomeStatement**. Click anywhere in the Report Header of the blank **rptBalanceSheet** and press **Ctrl+V** to paste the income statement header. Change the second line of the header to **Balance Sheet**. On the third line, delete all objects except the last one on the right, **=[TempVars]![EndDate]**, and center it.
3. Click and drag **rptBalanceSheet-Assets** from the Navigation Pane into the Detail section of the report. Delete the subreport label. Open the Property Sheet and change the Border Style to **transparent**.
4. Repeat Step 4 with **rptBalanceSheet-Liabilities** and again with **rptBalanceSheet-Equity**.
5. Save your changes to **rptBalanceSheet**.
6. Create a macro to automate the steps to generate an income statement and balance sheet. Use Figure 12.65 as a guide. Open a new macro in Design view. Add three

RunMacro Actions for `mcrTempDate`, `mcrInventory01-EndingInventory`, and `mcrCoGS01-CoGS`. Add `OpenReport` Actions for `rptBalanceSheet` and `rptIncomeStatement`. Save your new macro as `mcrFinancialStatements`. Close the macro.

7. Run the macro by double-clicking it. Enter `OpenReport` for the beginning date and `OpenReport` for the ending date. We purposely left the income statement and balance sheet reports open so that the user can examine them right after the macro executes all of its actions.

Fig. 12.63
Completed
`rptBalanceSheet` in
Design view.



Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Fig. 12.64
Completed *rptBalanceSheet* in Report view on 3/31/2010.

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Pipefitters Supply Company		
Balance Sheet		
3/31/2010		
<u>Assets</u>		
Cash		\$311,830.70
Accounts Receivable		\$11,874.05
Inventory		\$19,439.51
Prepaid Expenses		\$7,307.44
Property, Plant, & Equipment	\$7,551.08	
Accumulated Depreciation	\$243.64	
Property, Plant, & Equipment, Net		\$7,307.44
TotalAssets		\$368,041.69
<u>Liabilities and Stockholders' Equity</u>		
<u>Liabilities</u>		
Accounts Payable	\$2,458.79	
Wages Payable	\$13,141.57	
Payroll Taxes Payable	\$9,866.33	
Dividends Payable	\$0.00	
Loans Payable	\$144,607.72	
Total Liabilities		\$170,074.41
<u>Stockholders' Equity</u>		
Common Stock	\$240,000.00	
Retained Earnings	(\$42,032.72)	
Total Equity		\$197,967.28
Total Liabilities and Equity		\$368,041.69

Fig. 12.65
mcrFinancialStatements in Design view.

mcrFinancialStatements	
Action	Arguments
RunMacro	mcrTempDate, ,
RunMacro	mcrInventory01-EndingInventory, ,
RunMacro	mcrCoGS01-CoGS, ,
OpenReport	rptBalanceSheet, Report, , , Normal
OpenReport	rptIncomeStatement, Report, , , Normal

Microsoft product screen shot reprinted with permission from Microsoft Corporation.

You have just completed the last and, perhaps, the most important step in building an accounting information system—getting useful information out of the system. Congratulations!

Summary

This chapter described how to complete Pipefitters Supply Company's accounting information system. The only change in the Pipefitters' database design is the addition of tables to allow Pipefitters to store data related to support activities.

However, the main focus of this chapter was on getting financial accounting information out of the system. After all, if an accounting information system does a poor job of providing useful information, it does not matter how effective it is at storing the data. Gathering data from multiple business processes is no different than aggregating data from within a single business process as long as all of the business processes are fully integrated. This made deriving the cash balance easy because all cash receipts are stored in one table, as are all cash disbursements. Additionally, we used type tables (*tblCRTYPE* and *tblCDTYPE*) to easily identify cash receipts and disbursements by the type of event to which they are related.

On the other hand, computing the ending inventory balance and cost of goods sold proved to be quite a challenge. Though, with the help of action queries (make-table and update queries) and programming with macros, we showed you how a database can accomplish complex tasks. One other challenge was to amortize/depreciate prepaid expenses and fixed assets. We showed you how to use date functions to solve these problems.

Hopefully, you saw the payoff for all of your hard work when you ran *mcrFinancialStatements* for the first time and saw a balance sheet and income statement simply appear on your screen. Incorporating subreports into these financial statements allowed us to create reports with information that is not linked. Additionally, using temporary variables with input boxes in *mcrTempDate* allows users to enter reporting dates just one time, rather than forcing users to enter the dates for each set of financial-statement queries.

Questions and Problems for Review

Multiple-Choice Questions

1. Support activities include all of the following except
 - a. purchasing office supplies.
 - b. depreciating equipment.
 - c. paying employees.
 - d. computing the prepaid expense balances.
2. Which inventory cost flow assumption provides the same results for ending inventory and cost of goods sold (CoGS) for both perpetual and periodic inventory methods?
 - a. First-in-first-out (FIFO)
 - b. Last-in-first-out (LIFO)
 - c. Weighted average
 - d. All methods produce the same results for both perpetual and periodic inventory methods.
3. Which inventory cost flow assumption requires the cost of items on hand to change?
 - a. FIFO
 - b. LIFO
 - c. Weighted average
 - d. Specific identification
4. The FIFO inventory method assigns quantity on hand to the
 - a. items purchased most recently.
 - b. oldest items purchased.
 - c. specific inventory items identified in *tblOutflow-SaleInventory*.
 - d. None of the above are true.
5. What tables are used to identify inventory items purchased by date?
 - a. *tblPurchase* and *tblSale*
 - b. *tblInflow-PurchaseInventory* and *tblOutflow-SaleInventory*
 - c. *tblPurchase* and *tblInflow-PurchaseInventory*
 - d. *tblPurchase* and *tblOutflow-SaleInventory*

6. Which of the following is NOT true about the temporary variables created by *mcrTempDate*?
 - a. They can be used by any table, query, form, or report in the database.
 - b. They have their values set by the user.
 - c. They can be used to replace parameter queries.
 - d. All of the above are true.
7. Why does the query to add the sum of items immediately expensed by non-inventory type to the sum of amortization/depreciation expense by non-inventory type use outer joins?
 - a. All items immediately expensed should be reported even if there is not a related amortization/depreciation expense.
 - b. All amortization/depreciation expenses should be reported even if there is not a related item immediately expensed.
 - c. Both a and b are true.
 - d. Neither a nor b is true.
8. Which of the following is true about subreports?
 - a. Subreports do not need to have any fields linked to another subreport or to the main report.
 - b. At least one subreport must have fields linked to one or more fields in the main report.
 - c. Subreports must be linked to one or more fields either in the main report or in another subreport.
 - d. Subreports must be linked to one or more fields in the main report.
9. How can the retained earnings balance be verified for accuracy?
 - a. Compare it to net income generated by *qryIncomeStatement* when run from the beginning of the company's existence through the balance sheet date.
 - b. Compare it to net income generated by *qryIncomeStatement* when run for the current period, and subtracting dividends during the current period.
 - c. Compare it to net income generated by *qryIncomeStatement* when run from the beginning of the company's existence through the balance sheet date, and subtracting all dividends paid through the balance sheet date.
 - d. There is no way to verify the accuracy of retained earnings without using debits and credits.
10. How would you represent unearned revenue in a database AIS?
 - a. A record in *tblSale* without a related record in *tblCashReceipt*
 - b. A record in *tblSaleOrder* without a related record in *tblSale*
 - c. A record in *tblSale* with a related record in *tblCashReceipt*
 - d. A record in *tblSaleOrder* with a related record in *tblCashReceipt*

Discussion Questions

1. The macros to compute ending inventory and CoGS have OpenQuery Actions for only some of the queries needed to compute ending inventory and CoGS. Why?
2. Describe how you would change the ending inventory and CoGS queries to compute these balances based on the LIFO inventory cost flow assumption.
3. In this chapter, we showed you how to create a balance sheet and an income statement. What changes to the database would you make to produce a statement of cash flows?
4. How would you change the database and balance sheet queries to classify assets and liabilities into current and long-term?
5. What information do you feel is missing for non-inventory purchases? Why is it important?

Practice Exercises

Note: Before attempting any of the following Practice Exercises, save a new copy of *Ch12.accdb* from the Companion Web site to use for the Practice Exercises. Then, clear the copied database's Read-only file attribute. Begin

the names of all Access objects you create in the Practice Exercises with PE#-. Replace # with the Practice Exercise number.

1. Create a form to record non-inventory purchases. It will be similar to the inventory purchase form. You need to add PurchaseTypeID to your query and add it to the form. Since all values for PurchaseTypeID will be "02," set the default value to "02" and set the Visible property in the Format tab of the Property Sheet to No. Use the prefix PE1- for all Access objects you create for this form.
2. Computing weighted average ending inventory and cost of goods sold (CoGS) requires that you divide the total quantity available for sale by item by the sum of the line extensions by item. Create a set of queries to compute ending inventory and CoGS using weighted average. Assume that goods available for sale are all items purchased through the reporting date. Use the prefix PE2- for all Access objects you create for these computations.
3. Change the ending inventory and CoGS Access objects in this chapter to compute ending inventory and CoGS under the LIFO cost flow assumption. Use the prefix PE3- for all Access objects you create for these computations.
4. A company's operating cycle is the time it takes a business to convert inventory into accounts receivable, plus the average time it takes to convert accounts receivable into cash. The first component of this formula is to compute the average number of days to sell inventory. Since we do not identify which specific items are sold (we do not track inventory by serial number), we can approximate the average time to sell inventory: $365 \div \text{inventory turnover}$. $\text{Inventory turnover} = \text{CoGS} \div \text{average inventory}$. Use the ending inventory instead of average inventory ($[\text{beginning inventory} + \text{ending inventory}] / 2$). Use the prefix PE4- for all Access objects you create for these computations.
5. The second component that the operating cycle formula is to compute is the average number of days to collect accounts receivable. One way to do this is to compute the average number of days to collect accounts receivable: $365 \div \text{accounts receivable turnover}$, where $\text{accounts receivable turnover} = \text{sales} \div \text{average accounts receivable}$. Create queries to compute this for any period. Use ending accounts receivable instead of average accounts receivable. Use the prefix PE5- for all Access objects you create for these computations.
6. Based on your solutions to Practice Exercises 4 and 5, compute the Operating Cycle. Use the prefix PE6- for all Access objects that you create for these computations.