Lab 8.2: Building a Database Editor with DAO

Objectives

After completing this lab, you will be able to:

- Create an AppWizard application using DAO.
- Connect controls in a CDaoRecordView to their member variables.
- Add filters for data retrieval.
- Process CDaoRecordView::OnMove for multitable record coordination, update, deletion, and addition.
- Use transactions in a program.

Prerequisites

You should have completed Chapter 8 and mastered the use of the dialog editor before attempting this lab. You also should be familiar with the application created in Lab 8.1.

Lab Setup

This demonstration shows what you will accomplish during the lab.

Estimated time to complete this lab: **60 minutes**.

Exercises

The following exercises provide practice working with the concepts and techniques covered in this chapter.

Exercise 1: Building a DAO Database Application

In this exercise, you will implement a complete two-table database application using DAO.

Exercise 2: Editing the Underlying Database

In this exercise, you will edit Employee in the following ways:

- Add a new record.
- Modify a record.
- Delete a record.

There is no setup for this lab. The completed code for these exercises is in \Labs\C08\Lab02\Xxx, where Xxx is the exercise number.

Exercise 1: Building a DAO Database Application

In this exercise, you will implement a complete two-table database application using DAO. You will implement a more complete version of the simple record-oriented database viewer that was developed in Lab 8.1.

The database for this lab, Personnel.Mdb contains two tables: the Employee Pay Table and Employee Personal Info Table. Their schemas are:

Table: Employee Pay Table

Field Name	Data Type	Description
Employee Number	Number	Primary Index - Employee Number
Last Name	Text	Employee's Last Name (28 char max)
First Name	Text	Employee's First Name (28 char max)
Department #	Number	0-unassigned, 1-manufacturing, 2-administration, 3-information
Employee Pay Type	Number	0-Unknown, 1-Hourly Employee, 2-Salaried, 3-Salesperson
Hours	Number	Hourly employee's weekly hours
Hourly Rate	Number	Hourly employee's hourly wage
Weekly Salary	Number	Salaried employee's weekly salary
Sales Bonus Rate	Number	Salaesperson's bonus rate
Weekly Sales	Number	Salaesperson's weekly total sales

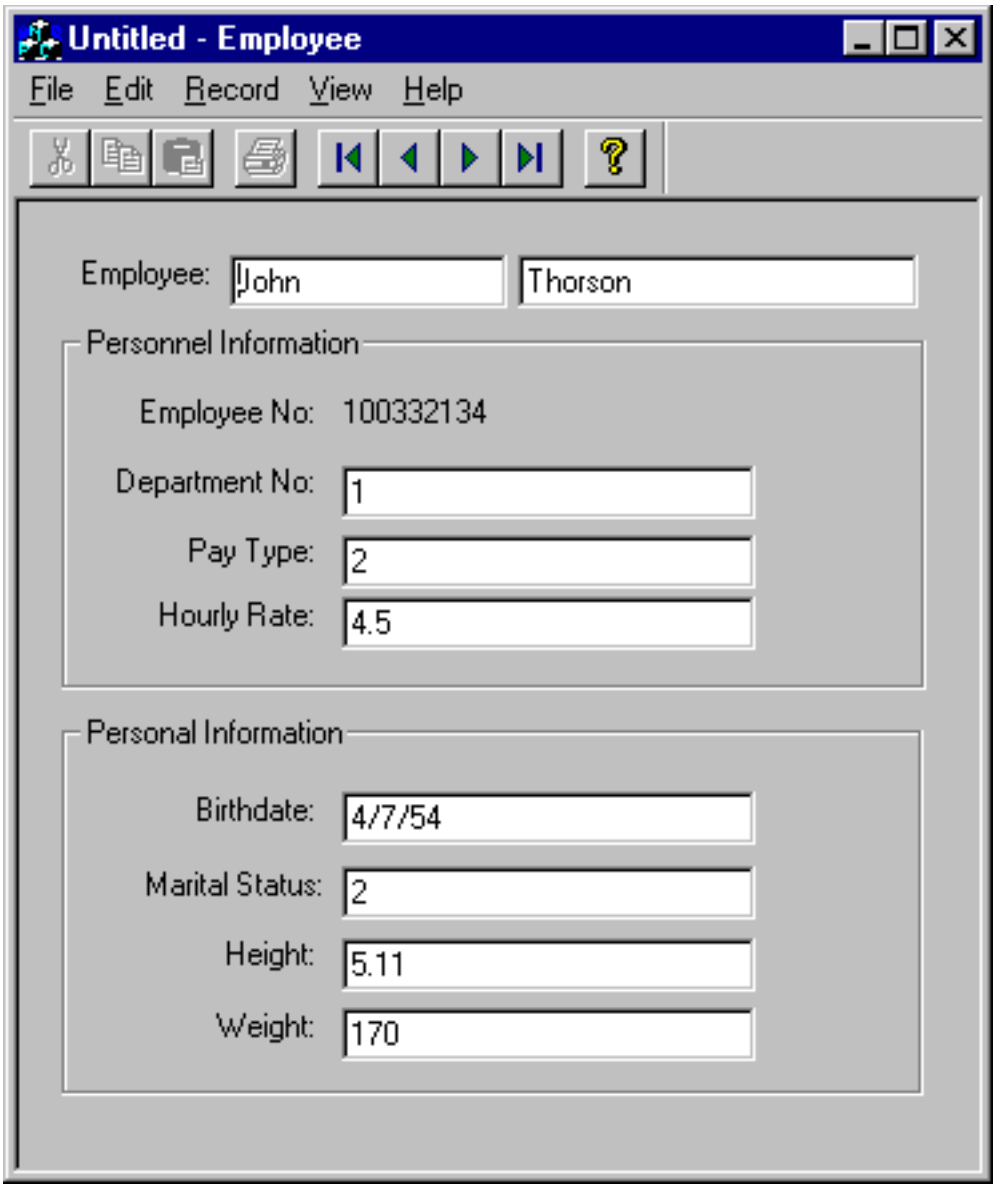
Table: Employee Personal Info Table

Field Name	Data Type	Description
Employee Number	Number	Primary Index - Employee Number
Birthdate	Date/Time	Employee's date of birth
Sex & Marital Status	Number	1-single male, 2-married male, 3-single female, 4-married female
Height	Number	Employee's height
Weight	Number	Employee's weight

Employee Number is a common key between the two tables. The SQL query behind this viewer is:

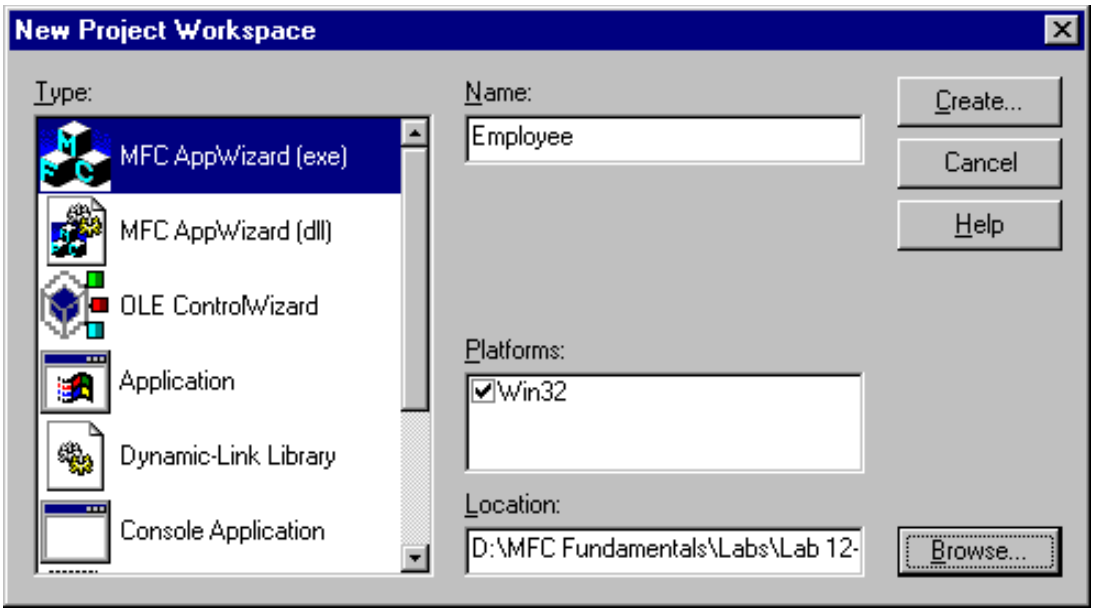
```
Select *
    from [Employee Personal Info Table], [Employee Pay Table]
    where [Employee Personal Info Table].[Employee Number] =
          [Employee Pay Table].[Employee Number]
```

Your complete application will look like the following graphic.

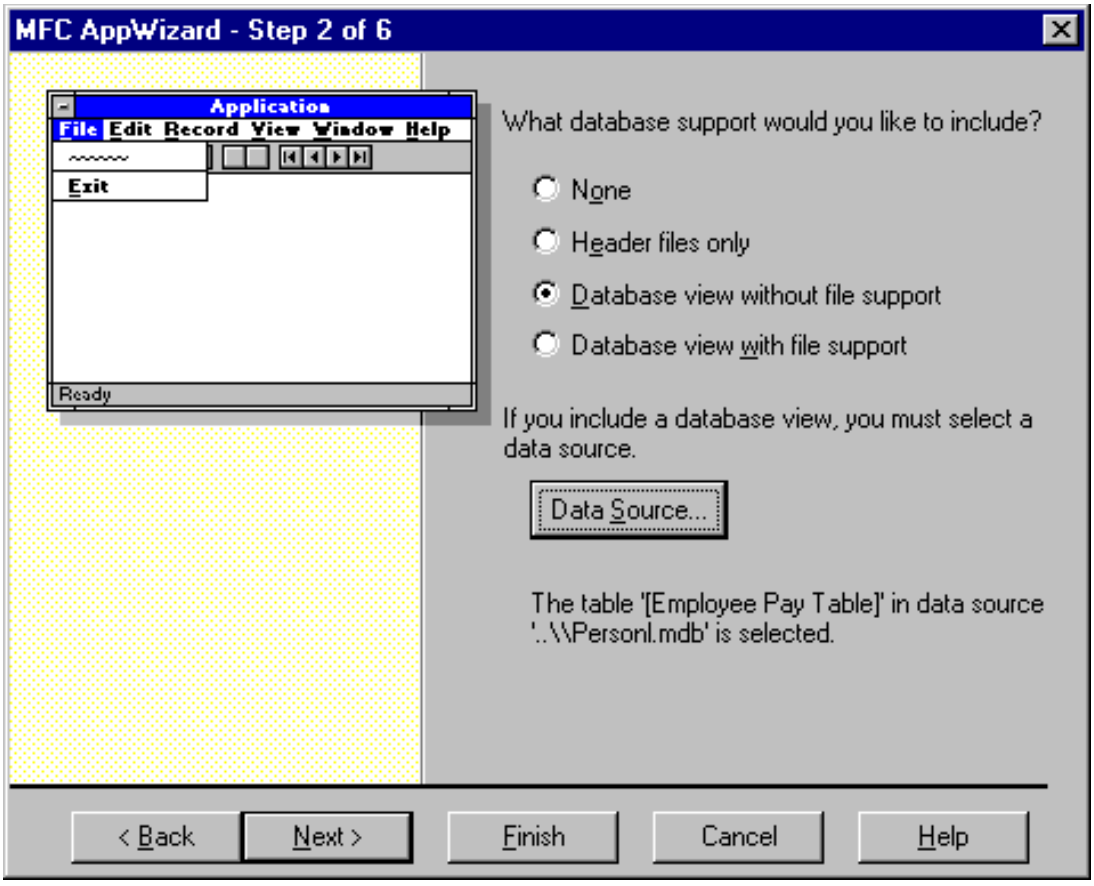


➢ **Create a new DAO database application**

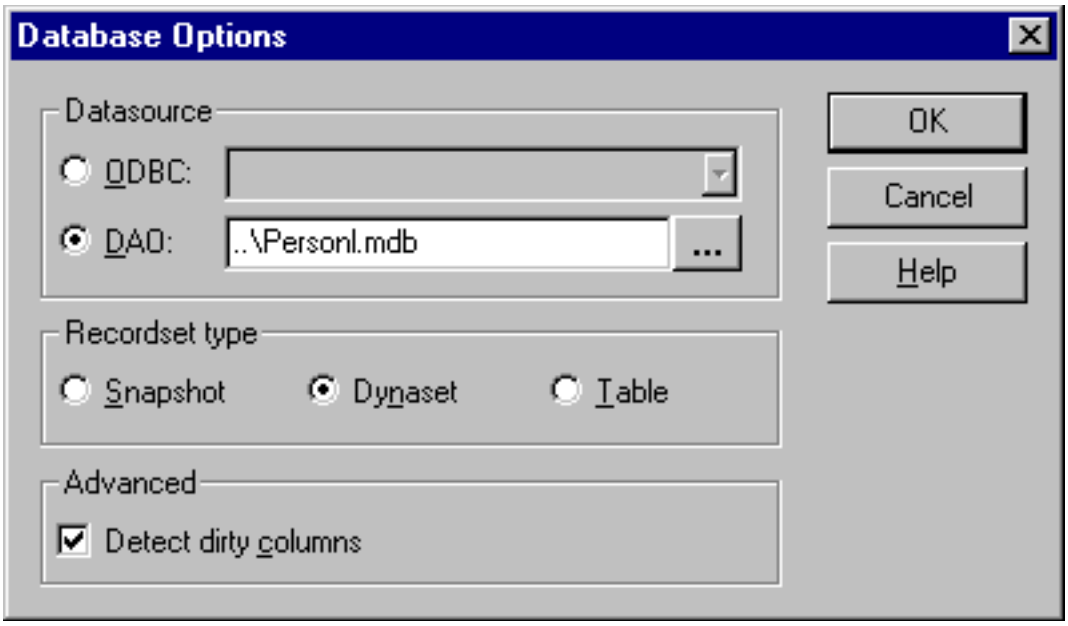
1. From the File menu, choose New.
2. Select the Projects tab; choose MFC AppWizard (exe), and type **Employee** in the Project Name box. Click OK.



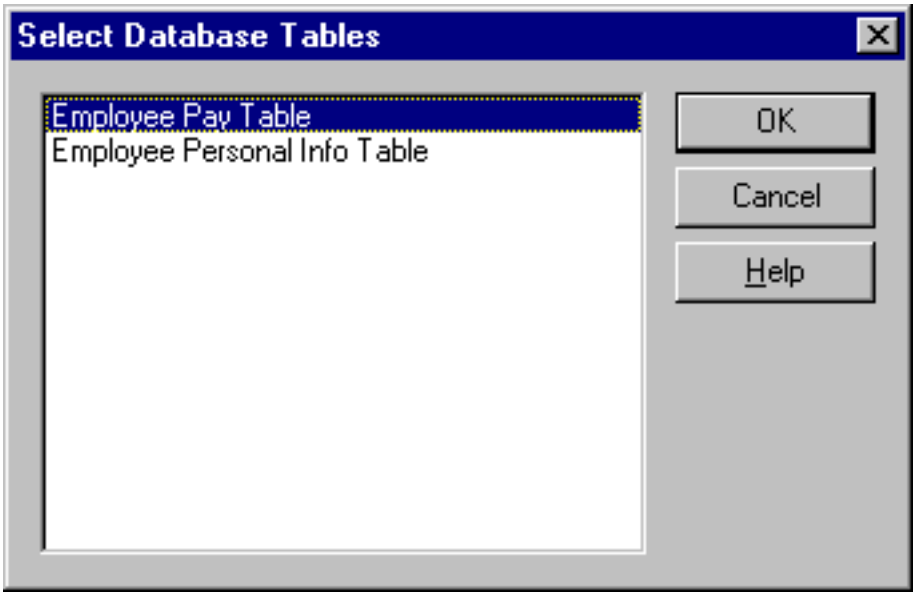
3. In Step 1, choose Single Document Interface.
4. In Step 2 (the Database Options Page):
5. Select the “Database view without file support” option.



6. Click Data Source. The Database options page will be displayed. Choose the DAO option. Click Browse to find the Personnel.Mdb database. Accept Dynaset as the Resource type. Leave the Detect dirty columns option selected.



When you click OK, the Select Database Tables dialog will be displayed. Select the Employee Pay Table and click OK. You will be returned to the Database Options page.



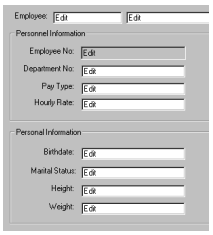
7. Accept the defaults in Steps 3, 4, and 5.
8. In Step 6, notice the base classes for the various classes in the application:

Class	Base Class
CEmployeeApp	**CWinApp**
CEmployeeDoc	**CDocument**
CMainFrame	**CFrameWnd**
CEmployeeView	**CDaoRecordView**
CEmployeeSet	**CDaoRecordSet**

9. Change the name of **CEmployeeSet** to **CEmployeePaySet**. You will be creating another **CDaoRecordSet**-derived class for the other table in the database soon, and you will need to be able to tell them apart. Change the files for **CEmployeePaySet** to EmployeePaySet.H and EmployeePaySet.Cpp.
10. Click OK in the Project Information dialog box to create the project.

➢ **Customize the Dialog Template for the Employee form**

1. Because **CEmployeeView** is derived from **CDaoRecordView**, and **CDaoRecordView** is derived from **CFormView**, you will use a dialog template to define the client area. Along with the classes it created, AppWizard created a dialog resource, IDD_EMPLOYEE_FORM, for you to lay out. AppWizard places one static text control in the resource, labeled “TO DO: Place form controls on this dialog.” Open IDD_EMPLOYEE _FORM by clicking this dialog’s icon in the Resource View.
2. Delete the one static control. You will develop a dialog box similar to the one below. If you have completed the previous lab, you can copy the dialog from that project and paste it into this project to speed up the dialog creation. If you do this, remember to turn off the Read-Only properties of the Edit controls.



Type	ID	Caption
Dialog	IDD_EMPLOYEE_FORM	
Right aligned text	IDC_STATIC	Employee:
Right aligned text	IDC_STATIC	Employee No:
Right aligned text	IDC_STATIC	Department No:
Right aligned text	IDC_STATIC	Pay Type:
Right aligned text	IDC_STATIC	Hourly Rate:
Right aligned text	IDC_STATIC	Birthdate:
Right aligned text	IDC_STATIC	Marital Status:
Right aligned text	IDC_STATIC	Height:
Right aligned text	IDC_STATIC	Weight:
Group box	IDC_STATIC	Personnel Information
Group box	IDC_STATIC	Personal Information
Edit control	IDC_FNAME	First Name
Edit control	IDC_LNAME	Last Name
Edit control	IDC_EMP_NO	**Note:** This control is read-only
Edit control	IDC_DEPT	
Edit control	IDC_PAY_TYPE	
Edit control	IDC_HOUR_RATE	
Edit control	IDC_BIRTH	
Edit control	IDC_MARITAL_STATUS	

Edit control	IDC_HEIGHT
Edit control	IDC_WEIGHT

3. Save Employee.Rc.

➢ **Bind controls to member variables**

You can use an extension to the dialog editor, or the ClassWizard, to bind controls to the appropriate member variables of **CEmployeePaySet**, according to the following table. You will bind the last four controls after you have created a **CDaoRecordSet** for their table.

Control	Data Type	Variable
IDC_FNAME	CString	m_pSet->m_First_Name
IDC_LNAME	CString	m_pSet->m_Last_Name
IDC_EMP_NO	long	m_pSet->m_Employee_Number
IDC_DEPT	short	m_pSet->m_Department__
IDC_PAY_TYPE	short	m_pSet->m_Employee_Pay_Type
IDC_HOUR_RATE	double	m_pSet->m_Hourly_Rate

1. To bind using the Dialog editor, CTRL+double-click the control to display the Add Member Variable dialog.

   To use the ClassWizard, press CTRL+W, then click the Member Variables tab and choose **CEmployeeView** from the Class name list box. You can browse the Control IDs. Click Add Variable to display the Add Member Variable dialog box.
2. Use the Member variable name combo box to select the variable name. Leave the category as Value, and the Variable type should default to the data types listed above.
3. Close the ClassWizard or the Dialog editor and save all files.

➢ **Order the display of the records**

In Lab 8.1, you created a single query for the join in which you included an ORDER BY clause. To produce the same effect in this two-query application, you need to make a query for **CEmployeePaySet** that is equivalent to the following:

```
SELECT *
FROM   [Employee Pay Table]
ORDER BY [Last Name],[First Name]
```

1. In DAO, you need specify only the ORDER BY in m_strSort. Add this to the end of the **CEmployeePaySet** constructor.

   ```
   m_strSort = "[Last Name],[First Name]";
   ```

2. Save EmployeePaySet.Cpp.

➢ **Create a class for a second table and bind its variables**

1. Start the ClassWizard from the View menu or press CTRL+W. Click Add Class, then click New.
2. Set the name of this new class to **CEmployeePersonalSet**. Set its base class to **CDaoRecordSet**, and click OK.
3. The Database Options dialog that you used to create **CEmployeePaySet** will be displayed, with one additional option: Bind all columns. Make sure that this option is checked. Leave the DAO option selected and choose the Personnel.Mdb database. Click OK.
4. In the Select Database Tables dialog box, choose Employee Personal Info Table and click OK. **CEmployeePersonalSet** will be created and added to your project.
5. Add a **CEmployeePersonalSet** instance variable to the CEmployeeView class within the AFX_DATA section of EmployeeView.H.

```
public:
    //{{AFX_DATA(CEmployeeView)
    enum { IDD = IDD_EMPLOYEE_FORM };
    CEmployeePaySet*    m_pSet;
    CEmployeePersonalSet*   m_pEmplInfoSet;
    //}}AFX_DATA
```

6. Save EmployeeView.H.
7. You need to edit the DDX links manually. Following the AFX_DATA_MAP section of **CEmployeeView::DoDataExchange**, validate m_pEmplInfoSet and then set the following DDX links.

Control	Datatype	Variable
IDC_BIRTH	COleDateTime	m_pSet->m_Birthdate
IDC_MARITAL_STATUS	BYTE	m_pSet->m_Sex___Marital_Status
IDC_HEIGHT	double	m_pSet->m_Height
IDC_WEIGHT	double	m_pSet->m_Weight

8. Save EmployeeView.Cpp. The complete function follows.

```
void CEmployeeView::DoDataExchange(CDataExchange* pDX)
{
    CDaoRecordView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CEmployeeView)
    DDX_FieldText(pDX, IDC_FNAME, m_pSet->m_First_Name, m_pSet);
    DDX_FieldText(pDX, IDC_LNAME, m_pSet->m_Last_Name, m_pSet);
    DDX_FieldText(pDX, IDC_EMP_NO, m_pSet->m_Employee_Number, m_pSet);
    DDX_FieldText(pDX, IDC_DEPT, m_pSet->m_Department__, m_pSet);
    DDX_FieldText(pDX, IDC_PAY_TYPE, m_pSet->m_Employee_Pay_Type, m_pSet);
    DDX_FieldText(pDX, IDC_HOUR_RATE, m_pSet->m_Hourly_Rate, m_pSet);
    //}}AFX_DATA_MAP

    //  Make sure this dataset pointer has been setup before trying
    //  to access it.
    if(m_pEmplInfoSet != NULL)
    {
        DDX_FieldText(pDX, IDC_BIRTH, m_pEmplInfoSet->m_Birthdate,
m_pEmplInfoSet);
        DDX_FieldText(pDX, IDC_MARITAL_STATUS, m_pEmplInfoSet-
>m_Sex___Marital_Status, m_pEmplInfoSet);
        DDX_FieldText(pDX, IDC_HEIGHT, m_pEmplInfoSet->m_Height,
m_pEmplInfoSet);
        DDX_FieldText(pDX, IDC_WEIGHT, m_pEmplInfoSet->m_Weight,
m_pEmplInfoSet);
    }

}
```

9. Embed **CEmployeePersonalSet** into **CEmployeeDoc** by adding an instance of **CEmployeePersonalSet** to EmployeeDoc.H.

```
    CEmployeePersonalSetm_employeePersonalInfoSet;
```

10. Add accessor functions for both the **CDaoRecordSet** objects.

```
// Operations
public:
    CEmployeePaySet*    GetEmployeePaySet()
                            {return &m_employeePaySet;}
```

```
CEmployeePersonalSet*   GetEmployeeInfoSet()
                            {return &m_employeePersonalInfoSet;}
```

11. Include EmployeePersonalSet.H in EmployeeDoc.Cpp, EmployeeView.Cpp, and Employee.Cpp. Since the document object embeds a **CEmployeePersonalSet** object within it, place the include before the include of the EmployeeDoc.H file.
12. Save all files.

➢ **Set up the table join**

Because you cannot assume that a join is updatable, you have to maintain the relationship between the two tables. The simplest way is to create a parameterized query that finds the record in Employee Personal Info Table that corresponds to the record found in the Employee Pay Table.

There are two methods for creating a parameterized query. You can use the **PARAMETERS** statement in the SQL query (as outlined in Chapter 10), or you can update the recordset class as shown in the following method.

1. Declare a **long** for the parameter in **CEmployeePersonalSet** after the AFX_FIELD block.

```
long m_EmployeeNumberParam;
```

2. Save EmployeePersonalSet.H. In the constructor for **CEmployeePersonalSet**, define the query.

```
m_strFilter = "[Employee Number] = EmployeeNumberParam";
```

3. Identify that there is one parameter, and initialize that parameter to 0.

```
m_nParams   = 1;
m_EmployeeNumberParam = 0;
```

4. In **CEmployeePersonalSet::DoFieldExchange**, you need to identify the query as parameterized. After the AFX_FIELD_MAP, use SetFieldType to parameterize the recordset.

```
pFX->SetFieldType(CDaoFieldExchange::param);
```

5. Set the value of the parameterized field.

```
DFX_Long(pFX, _T("EmployeeNumberParam"), m_EmployeeNumberParam);
```

6. Save EmployeePersonalSet.Cpp.

➢ **Implement CEmployeeView::OnInitialUpdate**

1. Before the **CDaoRecordView::OnInitialUpdate** statement, set the instance variable for **CEmployeePersonalSet**.

```
m_pEmplInfoSet  = GetDocument()->GetEmployeeInfoSet();
```

2. Resize the main window so that it matches the dialog template.

```
GetParentFrame()->RecalcLayout();
ResizeParentToFit(FALSE);
```

3. Set the employee number parameter for the **CEmployeePersonalSet** filter from the employee number field of **CEmployeePaySet**.

```
m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
```

4. Open **CEmployeePersonalSet** and position the set to the same employee number. Because **CDaoRecordSet::Open** is a likely place for exceptions to be thrown, you should place this call in a **try** block.

```
try
{
    m_pEmplInfoSet->Open();
```

```
    }
```

5. If there is an error in the execution of the function **Open**, use **CDaoException** to recognize the exception and display it to the user.

```
    catch (CDaoException* e)
    {
        AfxMessageBox(e->m_pErrorInfo->m_strDescription);
```

6. Delete the exception and return from the function without updating the display.

```
        e->Delete();
        return;
    }
```

7. Using DDX, update the screen display from the member variables.

```
    UpdateData(FALSE);
```

8. Save EmployeeView.Cpp. The complete function follows.

```
void CEmployeeView::OnInitialUpdate()
{
    m_pSet          = GetDocument()->GetEmployeePaySet();
    m_pEmplInfoSet  = GetDocument()->GetEmployeeInfoSet();

    CDaoRecordView::OnInitialUpdate();

    //resize the window
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit();

    //set the parameter for the info query
    m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;

    //open the Info recordset
    try
    {
        m_pEmplInfoSet->Open();
    }
    catch (CDaoException* e)
    {
        AfxMessageBox(e->m_pErrorInfo->m_strDescription);
        e->Delete();
        return;
    }

    UpdateData(FALSE);
}
```

➢ **Implement CEmployeeView::OnMove**

The **OnMove** message is sent to the view from the default handlers of the **OnRecordFirst**, **OnRecordPrevious, OnRecordNext,** and **OnRecordLast** messages in **CDaoRecordView**. In the simplest case, such as in Lab 8.1, **OnMove** is passed back up the hierarchy, unhandled. In this lab, you will force a requery of **CEmployeePersonalSet** after the default set for this view, **CEmployeePaySet**, has been updated.

1. Use the ClassWizard or the WizardBar to create a handler for **CEmployeeView::OnMove**. The ClassWizard will include a call of the default handler. Remove the **return**.

```
BOOL CEmployeeView::OnMove(UINT nIDMoveCommand)
{
```

```
    CDaoRecordView::OnMove(nIDMoveCommand);
```

2. Set the parameter for **CEmployeePersonalSet** from the employee number.

```
    m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
```

3. Execute the query on **CEmployeePersonalSet**.

```
    m_pEmplInfoSet->Requery();
```

4. Initiate a data transfer into the dialog and set the return status.

```
    UpdateData(FALSE);
    return TRUE;
```

5. Save EmployeeView.Cpp. The complete function follows.

```
BOOL CEmployeeView::OnMove(UINT nIDMoveCommand)
{
    CDaoRecordView::OnMove(nIDMoveCommand);

    //set the parameter for the info query
    m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;

    //do the secondary query
    m_pEmplInfoSet->Requery();

    UpdateData(FALSE);

    return TRUE;

}
```

➢ **Build and run Employee.Exe**

It has the same functionality as Join.Exe from Lab 8.1, except that you can enter data into the edit controls.

The completed code for this exercise is in \Labs\C08\Lab02\Ex01.

Exercise 2: Editing the Underlying Database

Continue with the files you created in Exercise 1 or, if you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C08\Lab02\Ex01.

In this exercise, you will edit Employee in the following ways:

- Add a new record.
- Modify a record.
- Delete a record.

There are five parts to this exercise:

1. Adding items to the menu.
2. Preparing to add a record.
3. Adding a record.
4. Updating a record.
5. Deleting a record.

Adding Items to the Menu

1. Open the IDR_MAINFRAME menu.

2. Add a separator to the end of the Record menu.
3. Add the following items to the Record menu.

ID	Caption	Prompt
ID_RECORD_CLEAR	&Clear Record	Clear the fields in the form
ID_RECORD_ADD	&Add Record	Add this record to the database
ID_RECORD_DELETE	&Delete Record	Delete this record from the database

4. Save Employee.Rc.
5. Use the ClassWizard to create command handlers for each of these menu items in **CEmployeeView**.
6. Create an Update Command UI handler for ID_RECORD_DELETE.
7. Create an Update Command UI handler for ID_RECORD_ADD.

Preparing to Add Records

The first step in adding a record to a database is to get an empty recordset, or in the case of Employee.Exe, two empty recordsets. In this exercise, you will explicitly clear and add records. When the user moves to another record, you will end the editing session and return to normal viewing. You also will provide a function for canceling the add process.

➢ Implement OnRecordClear

1. Because **OnRecordClear** is the first step in adding records, start a transaction as follows:

```
m_pSet->m_pDatabase->m_pWorkspace->BeginTrans();
```

2. Call **CDaoRecordSet::AddNew** for both recordsets.

```
m_pSet->AddNew();
m_pEmplInfoSet->AddNew();
```

3. Display the blank fields in the form.

```
UpdateData(FALSE);
```

4. Set the member variable for the Add Mode, and set the state of the Employee Number edit control.

```
SetAddMode(TRUE);
```

5. Save EmployeeView.Cpp. The complete function follows.

```
void CEmployeeView::OnRecordClear()
{
    m_pSet->m_pDatabase->m_pWorkspace->BeginTrans();
    m_pSet->AddNew();
    m_pEmplInfoSet->AddNew();
    UpdateData(FALSE);
    SetAddMode(TRUE);
}
```

➢ Implement SetAddMode

1. In the ClassView pane, right-click **CEmployeeView**, choose Add Member Function. Type **void** in the Return type box and type **SetAddMode(BOOL bAddMode = TRUE)** in the Declaration box.
2. In the ClassView pane, right-click **CEmployeeView**, choose Add Member Variable. Type **m_bAddMode** in the Name box and choose Protected for the access mode.
3. Open EmployeeView.Cpp. In the body of the **SetAddMode** function, set the add mode state to the passed value.

```
m_bAddMode = bAddMode;
```

4. Get a pointer to the employee number edit control.

```
CEdit* pField = (CEdit* )GetDlgItem(IDC_EMP_NO);
```

5. Set the state of that edit control to readable when you are in the add mode, or read-only when you are not in add mode.

```
pField->SetReadOnly(!m_bAddMode);
```

6. If you are in the add mode, set the focus to the first-name edit control.

```
if(m_bAddMode)
{
    pField = (CEdit* )GetDlgItem(IDC_FNAME);
    pField->SetFocus();
}
```

7. Save EmployeeView.Cpp The complete function follows.

```
void CEmployeeView::SetAddMode(BOOL bAddMode /* = TRUE */)
{
    m_bAddMode = bAddMode;

    CEdit* pField = (CEdit* )GetDlgItem(IDC_EMP_NO);

    pField->SetReadOnly(!m_bAddMode);

    if(m_bAddMode)
    {
        pField = (CEdit* )GetDlgItem(IDC_FNAME);
        pField->SetFocus();
    }
}
```

➢ **Cancel the Add Mode**

1. In the ClassView pane, right-click **CEmployeeView**, choose Add Member Function. Type **void** in the Return type box and type **AddRecordCancel()** in the Declaration box.
2. In the ClassView pane, right-click **CEmployeeView**, choose Add Member Variable. Type **m_bAddMode** in the Name box and choose Protected for the access mode.
3. Open EmployeeView.Cpp. In the body of **AddRecordCancel** function, add code to implement if not in add mode, **return**.

```
if (!m_bAddMode)
    return;
```

4. Turn off the add mode.

```
SetAddMode(FALSE);
```

5. Cancel the update on both tables.

```
m_pSet->CancelUpdate();
m_pEmplInfoSet->CancelUpdate();
```

6. AddRecordCancel aborts the transaction started in OnRecordClear. Roll back the system to its prior state.

```
    m_pSet->m_pDatabase->m_pWorkspace->Rollback();
```

7. Requery both the tables.

```
m_pSet->Requery();
```

```
    m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
    m_pEmplInfoSet->Requery();
```

8. Refresh the form.

```
    UpdateData(FALSE);
```

9. Save EmployeeView.Cpp. The complete function follows.

```
void CEmployeeView::AddRecordCancel()
{
    SetAddMode(FALSE);

    m_pSet->CancelUpdate();
    m_pEmplInfoSet->CancelUpdate();
    m_pSet->m_pDatabase->m_pWorkspace->Rollback();

    m_pSet->Requery();
    m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
    m_pEmplInfoSet->Requery();

    UpdateData(FALSE);
}
```

➢ **Implement the Command UI handlers**

1. Enable the Add menu item only when you are in add mode.

```
void CEmployeeView::OnUpdateRecordAdd(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bAddMode);

}
```

2. Enable the Delete menu item only when you are not in add mode.

```
void CEmployeeView::OnUpdateRecordDelete(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(!m_bAddMode);

}
```

3. Set add mode to **FALSE** in **CEmployeeView::CEmployeeView**.

```
CEmployeeView::CEmployeeView()
    : CDaoRecordView(CEmployeeView::IDD)
{
    //{{AFX_DATA_INIT(CEmployeeView)
    m_pSet = NULL;
    //}}AFX_DATA_INIT
    // TODO: add construction code here
    m_bAddMode = FALSE;
}
```

4. Save EmployeeView.Cpp.

➢ **Reset add mode on navigation**

1. The default handler for **CDaoRecordView::OnMove** updates the record sets before navigating off them. Because you will be aborting new record creation on navigation, check whether you are in add mode. If so, cancel it before calling the default handler in **CEmployeeView::OnMove**.

```
if(m_bAddMode)
{
    AddRecordCancel();
}

CDaoRecordView::OnMove(nIDMoveCommand);  //existing line
```

2. Save EmployeeView.Cpp.

Adding or Updating a Record

In adding and updating records, you update two tables simultaneously. These two functions use transactions to encapsulate database modifications so that synchronization of adds are assured.

➢ **Add a record**

1. In **OnRecordAdd**, refresh the recordsets from the form.

```
UpdateData();
```

2. Because you are in a transaction, you will perform all your updates inside a **try** block so that you can roll back the entire failed transaction, leaving both tables as you found them. DDX does not update m_pEmplInfoSet->m_Employee_Number; begin by setting it from m_pSet.

```
try
{
    m_pEmplInfoSet->m_Employee_Number = m_pSet->m_Employee_Number;
```

3. Update both recordsets to their corresponding tables.

```
    m_pSet->Update();
    m_pEmplInfoSet->Update();
```

4. If there are no exceptions in either of the calls to the function **Update**, commit the transaction in the database.

```
    m_pSet->m_pDatabase->m_pWorkspace->CommitTrans();
```

5. If an exception occurs, use the **Rollback** function to restore the database to its state at the time the function **BeginTrans** was called. There are many reasons for DAO to get an exception. The most common reasons in Employee.Exe are having NULL data for database fields that require data, or the database having specific acceptable data needs that are not implemented by DDV.

```
catch (CDaoException* e)
{
    m_pSet->m_pDatabase->m_pWorkspace->Rollback();
```

6. Tell the user why the transaction failed, and delete the exception.

```
    AfxMessageBox(e->m_pErrorInfo->m_strDescription);
    e->Delete();
```

7. Because most exceptions are caused by trivial user errors, restart the transaction and **return**.

```
    m_pSet->m_pDatabase->m_pWorkspace->BeginTrans();
    m_pSet->AddNew();
    m_pEmplInfoSet->AddNew();
    return;
}
```

8. The record you just added is not part of the current recordsets. Requery the database to make the recordsets current. This will place you on the first record of the new sets.

```
m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
```

```
    m_pSet->Requery();
    m_pEmplInfoSet->Requery();
```

9. Update the form and turn off the add mode.

```
    UpdateData(FALSE);
    SetAddMode(FALSE);
```

10. Save EmployeeView.Cpp. The complete function follows.

```
void CEmployeeView::OnRecordAdd()
{
    UpdateData();

    try
    {

        m_pEmplInfoSet->m_Employee_Number = m_pSet->m_Employee_Number;
        m_pSet->Update();
        m_pEmplInfoSet->Update();

        //if we didn't except out, commit it to the database
        m_pSet->m_pDatabase->m_pWorkspace->CommitTrans();
    }
    catch (CDaoException* e)
    {
        //something happened, so restore the db
        m_pSet->m_pDatabase->m_pWorkspace->Rollback();

        //tell why
        AfxMessageBox(e->m_pErrorInfo->m_strDescription);
        e->Delete();

        //restore the add mode
        m_pSet->m_pDatabase->m_pWorkspace->BeginTrans();
        m_pSet->AddNew();
        m_pEmplInfoSet->AddNew();
        return;
    }

    //load it all back in, reset current record
    m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
    m_pSet->Requery();
    m_pEmplInfoSet->Requery();

    //copy back to the form
    UpdateData(FALSE);

    SetAddMode(FALSE);
}
```

➢ **Update a record**

Updating single-table recordsets is managed as a part of **CDaoRecordView::OnMove**. Wrap that message with additional functionality to update the second recordset.

1. When you are navigating and a transaction fails, the most logical place to return to is the current record. Set a bookmark to the current record in **CEmployeePaySet**.

```
    COleVariant varRecordToReturnTo;
```

```
varRecordToReturnTo = m_pSet->GetBookmark();
```

2. Wrap your database calls in a **try** block and start the transaction.

```
try
{
    m_pSet->m_pDatabase->m_pWorkspace->BeginTrans();
```

3. Put the employee info set into edit mode.

```
    m_pEmplInfoSet->Edit();
```

4. Call **CDaoRecordView::OnMove**. **OnMove** will update the employee pay set and move to the record specified by nIDMoveCommand.

```
    CDaoRecordView::OnMove(nIDMoveCommand);
```

5. Update the employee info set.

```
    m_pEmplInfoSet->Update();
```

6. Commit the transaction. If something has failed, it will be addressed in the exception block before this line:

```
    m_pSet->m_pDatabase->m_pWorkspace->CommitTrans();
```

7. Catch the DAO exception, if necessary, and roll back the transaction.

```
catch (CDaoException* e)
{
    m_pSet->m_pDatabase->m_pWorkspace->Rollback();
```

8. Set the employee pay back to the bookmark.

```
    m_pSet->SetBookmark( varRecordToReturnTo );
```

9. Inform the user what happened.

```
    AfxMessageBox(e->m_pErrorInfo->m_strDescription);
    e->Delete();
```

10. Find the employee information record, whether the transaction succeeded or not.

```
m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
m_pEmplInfoSet->Requery();
```

11. Show the data in the form.

```
UpdateData(FALSE);
```

12. Save EmployeeView.Cpp. The complete function follows.

```
BOOL CEmployeeView::OnMove(UINT nIDMoveCommand)
{
    if(m_bAddMode)
    {
        AddRecordCancel();
    }

    COleVariant varRecordToReturnTo;
    varRecordToReturnTo = m_pSet->GetBookmark();
    try
    {
        //  Begin update transaction
        m_pSet->m_pDatabase->m_pWorkspace->BeginTrans();
```

```
        //  Put the EmployeeInfoSet into Edit mode before
        //  calling CDaoRecordSet::OnMove()
        m_pEmplInfoSet->Edit();

        //  CDaoRecordView::OnMove() will call UpdateData
        CDaoRecordView::OnMove(nIDMoveCommand);

        //  Finish the update of the EmployeeInfoSet
        m_pEmplInfoSet->Update();
        //  Commit update transaction
        m_pSet->m_pDatabase->m_pWorkspace->CommitTrans();
    }
    catch (CDaoException* e)
    {
        //  Rollback changes
        m_pSet->m_pDatabase->m_pWorkspace->Rollback();

        //  Make sure PayInfo record set is returned
        //  to the proper position
        m_pSet->SetBookmark( varRecordToReturnTo );

        AfxMessageBox(e->m_pErrorInfo->m_strDescription);
        e->Delete();
    }

    //  Update EmployeeNumber parameter and requery
    m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
    m_pEmplInfoSet->Requery();

    // Show results of Employee Info requery
    UpdateData(FALSE);

    return TRUE;
}
```

Deleting a Record

You delete the two records in much the same way that you add and update them. You will start a transaction, delete the records and, if the transaction fails, recover.

➢ **Delete a record**

1. Start a try block in **CEmployeeView::OnRecordDelete** with **BeginTrans**.

```
    try
    {
        m_pSet->m_pDatabase->m_pWorkspace->BeginTrans();
```

2. Delete the records and end the transaction.

```
        m_pSet->Delete();
        m_pEmplInfoSet->Delete();
        m_pSet->m_pDatabase->m_pWorkspace->CommitTrans();
```

3. If the block excepted out, roll back the transaction, and tell the user what happened.

```
    catch (CDaoException* e)
    {
        m_pSet->m_pDatabase->m_pWorkspace->Rollback();
```

```
        AfxMessageBox(e->m_pErrorInfo->m_strDescription);
        e->Delete();
        return;
    }
```

4. Requery the recordset.

```
    m_pSet->MoveNext();
    m_pSet->Requery();
```

5. If you are past the end of the set, move to the last record.

```
    if(m_pSet->IsEOF() && !m_pSet->IsBOF())
        m_pSet->MoveLast();
```

6. If you just deleted the last record in the set, clear it out.

```
    if(m_pSet->IsBOF())
        m_pSet->SetFieldNull(NULL);
```

7. Find the corresponding record in the Employee Info set.

```
    m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
    m_pEmplInfoSet->Requery();

    if(m_pEmplInfoSet->IsEOF() && m_pEmplInfoSet->IsBOF())
            m_pEmplInfoSet->SetFieldNull(NULL);
```

8. Display the record in the form.

```
    UpdateData(FALSE);
```

9. Save EmployeeView.Cpp. The complete function follows.

```
void CEmployeeView::OnRecordDelete()
{
    try
    {
        //begin the transaction
        m_pSet->m_pDatabase->m_pWorkspace->BeginTrans();

        //delete both records
        m_pSet->Delete();
        m_pEmplInfoSet->Delete();

        //end the transaction
        m_pSet->m_pDatabase->m_pWorkspace->CommitTrans();
    }
    catch (CDaoException* e)
    {
        //something happened, so restore the db
        m_pSet->m_pDatabase->m_pWorkspace->Rollback();

        //you could clear stuff out here...

        //tell why
        AfxMessageBox(e->m_pErrorInfo->m_strDescription);
        e->Delete();

        //and bail out
        return;
```

```
    }

    m_pSet->MoveNext();
    m_pSet->Requery();

    if(m_pSet->IsEOF() && !m_pSet->IsBOF())
        m_pSet->MoveLast();

    if(m_pSet->IsBOF())
        m_pSet->SetFieldNull(NULL);

    m_pEmplInfoSet->m_EmployeeNumberParam = m_pSet->m_Employee_Number;
    m_pEmplInfoSet->Requery();

    if(m_pEmplInfoSet->IsEOF() && m_pEmplInfoSet->IsBOF())
            m_pEmplInfoSet->SetFieldNull(NULL);

    UpdateData(FALSE);
}
```

10. Build and run Employee.Exe.

The completed code for this exercise is in \Labs\C08\Lab02\Ex02.