```
    m_rsDataSet.Requery();
    DisplayRecord();
    SetButtonState();

}
```

第十三步，在“删除”按钮中加入如下的事件脚本（程序 6-6-13）：

程序 6-6-13：

```
void CBookMDlg::OnDelete()
{
    // TODO: Add your control notification handler code here
    m_rsDataSet.Delete();   //删除记录
    DisplayRecord();        //显示记录
    SetButtonState();       //设置按钮状态
}
```

第十四步，在“退出”按钮中加入如下的事件脚本（程序 6-6-14）：

程序 6-6-14：

```
void CReaderMDlg::OnExit()
{
    // TODO: Add your control notification handler code here
    CDialog::OnOK();
}
```

6.7　还书模块

最后，介绍“还书模块”的实现。介绍这个模块的主要目的是介绍在记录集中引入多表连接的技术。因而，在这一节中，只简单罗列操作步骤，着重介绍记录集 CBorrowSet 的生成。

第一步，生成如图 6-15 所示的对话框。

图 6-15 第一步 操作对话框

第二步，设相应的编辑框的成员变量为如下所得变量名称（程序 6-7-1）。

程序 6-7-1：

```
    // Dialog Data
//{{AFX_DATA(CReturnBookDlg)
enum { IDD = IDD_ReturnBook };
CString m_Book_ID;          //书 ID
CString m_BorrowDate;       //借阅日期
int     m_Days;             //借阅天数
CString m_ReaderName;       //读者姓名
CString m_ReturnDate;       //还书日期
//}}AFX_DATA
```

第三步，生成记录集 CBorrowSet。CborrowSet 的记录集的 SQL 语句为：

```
SELECT borrow.* , book.* from borrow , reader where borrow.reader_id
=book.reader_id ;
```

（1）首先使用 ClassWizard 生成第一个表 Borrow 的记录集。绑定 Borrow 的字段。

（2）选择 ClassWizard 的 update columns 的按钮，第二次打开数据源对话框，选择第二个表 Reader，绑定第二个表 Reader 的字段。注意如果第二个表的字段名与第一个表的字段名相同，在绑定时应保证成员变量名不同。最后得到的成员变量名如下所示（程序 6-7-2）。

程序 6-7-2：

```
void CBorrowSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(CBorrowSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[BORROW].[READER_ID]"), m_READER_ID);
    RFX_Text(pFX, _T("[BOOK_ID]"), m_BOOK_ID);
    RFX_Date(pFX, _T("[BORROW_DATE]"), m_BORROW_DATE);
    RFX_Text(pFX, _T("[NAME]"), m_READERNAME);
    //}}AFX_FIELD_MAP
}
```

（3）关闭 Classwizard 对话框。

（4）修改生成的源代码，修改 CBorrowSet::GetDefaultSQL()的源代码为如下形式（程序 6-7-3）。

程序 6-7-3：

```
CString CBorrowSet::GetDefaultSQL()
{
    return _T("[BORROW], [READER]");
}
```

第四步，在编辑框 IDC_EDITBOOKID 的 KILLFOCUS 事件中插入如下的事件脚本（程序 6-7-4）。

程序 6-7-4：

```
void CReturnBookDlg::OnKillfocusEDITBookID()
```

```
{
    int rs;
    CWnd *pWnd;
    CString m_strtmp;
    rs=QryBorrow();/*查询借阅记录，并看其是否过期还书.....代码1*/
    if(rs==1)   /*图书超期*/
    {
        m_strtmp.Format("超期%d天，还书处理正常完成。是否继续处理还书流程？",
        m_Days);
        if(::MessageBox(this->m_hWnd, m_strtmp, "提示",
          MB_YESNO)==IDYES)
            SetTxtNull();
        else
            CDialog::OnClose();
    }
    else if(rs==2) /*还书正常处理*/
    {
        if(::MessageBox(this->m_hWnd, "还书处理正常完成", "提示",
MB_YESNO)==IDYES)
            SetTxtNull();
        else
            CDialog::OnClose();
    }
    else if(rs==0) /*输入错误的书号*/
    {
        ::MessageBox(this->m_hWnd, "该书号信息不存在！", "警告", MB_OK);
    }
    // TODO: Add your control notification handler code here
```

```
    pWnd=GetDlgItem(IDC_EDITBook_ID);
    pWnd->SetFocus();
}
```

代码 1 中调用 QryBorrow 函数，该函数是用户自己定义的函数，它的具体实现可在下面看到。

第五步，在 CReturnDialogBook 中加入成员函数 QryBorrow()（程序 6-7-5）。

程序 6-7-5：

```
int CReturnBookDlg::QryBorrow()
{
    int  result;
    CTime  m_ReturnDate_tmp; /*存放还书日期临时变量.....代码 1*/
    Cstring  m_strtmp;
    CTime  m_Current_Time=CTime::GetCurrentTime();//获得当前时间
    CTimeSpan m_TimeSpan_tmp(30, 23, 59, 59);/*.....代码 2*/
    UpdateData(true);                    //将屏幕的数据保存到成员变量中
    if (strcmp(m_Book_ID, "")==0)    //book_ID 编辑框必须有值
        return 0;
    //设置过滤条件
m_rsDataSet.m_strFilter="reader.reader_ID=borrow.reader_ID and
BOOK_ID='"+m_Book_ID+"'";
    if (!m_rsDataSet.Open())              //打开记录集
    {
        AfxMessageBox("数据打开失败！");
        return -1;
    }
    m_rsDataSet.Requery();
    if (m_rsDataSet.GetRecordCount()!=0)
    {
```

```
        m_ReaderName=m_rsDataSet.m_ReaderName ;
        m_BorrowDate=m_rsDataSet.m_BORROW_DATE.Format("%Y, %B%d");
        //...代码 3

m_ReturnDate_tmp=m_rsDataSet.m_BORROW_DATE+m_TimeSpan_tmp;//代码 4
m_ReturnDate=m_ReturnDate_tmp.Format("%Y, %B %d"); //.....代码 5
  if (m_ReturnDate_tmp<m_Current_Time)  /*超期还书 .....代码 6*/
        {
            m_TimeSpan_tmp=CTime::GetCurrentTime()-m_ReturnDate_tmp;
            //.....代码 7
            m_Days=m_TimeSpan_tmp.GetDays(); /*将时间间隔所包含的值转换
                                                为天数*/
            UpdateData(false);    /*将从数据库中取得的值显示在屏幕上*/
            Insert_Fine();  //在罚款数据库表中插入一条记录
            result=1;
        }
        else     //正常还书
        {
            UpdateData(false);
            Insert_History();//正常还书
            result=2;
        }
        m_rsDataSet.Close();
        return result;
  }
  else
  {
```

```
        return 0;
    }
}
```

在 Visual C++中处理时间运算是比较麻烦的。在 MFC 类库中提供了 Ctime 类和 CtimeSpan 类。Ctime 对象表示了绝对的时间和日期，而 CTimeSpan 表示时间间隔，即两个时间点的差值。CtimeSpan 是用秒来表示时间间隔的。在代码 1 中已声明了 Ctime 变量。在代码 2 中声明了一个 CtimeSpan 变量，m_Timespan_tmp。初始化 m_Timespan_time 为 30 天，23 小时，59 秒。这个时间间隔正好是一个月，它表示允许读者借书的最大期限为一个月。

在代码 3 中我们调用了 Ctime 的方法 Format。Format 方法的作用返回一个格式化时间。它的函数原型是：CString Format(LPCTSTR *pFormat*) const; 参数 pFormat 可用的格式如下几种：

- %D 当前时间的天数
- %H 当天的小时数
- %M 当前小时的分钟数
- %S 当前分钟的秒数
- %% 百分号

代码 4 将借书日期加上最大允许借书的时间范围得到应该还书的日期。这里使用了 Ctime 的“+”号运载符。它的原型说明是：CTime operator +(CTimeSpan *timeSpan*) const；注意 Ctime+CtimeSpan 而不能 Ctime+Ctime。

代码 5 格式化还书日期。代码 6 判断还书日期是否超过。注意 Ctime 可以和 Ctime 进行比较运算。代码 7 是计算超期几天，使用“-”操作符。下面给出 Ctime 几个操作符和 CtimeSpan 的几个操作符。

```
CTime operator +( CTimeSpan timeSpan ) const;
CTime operator -( CTimeSpan timeSpan ) const;
CTimeSpan operator -( CTime time ) corst;
BOOL operator ==( CTime time ) const;
BOOL operator !=( CTime time ) const;
BOCL operator <( CTime time ) const;
BOCL operator >( CTime time ) const;
```

```
BOOL operator <=( CTime time ) const;
BOOL operator >=( CTime time ) const;
CTimeSpan operator +( CTimeSpan timeSpan ) const;
CTimeSpan operator -( CTimeSpan timeSpan ) const;
```

至此，已经介绍了登录模块、读者资料维护模块、还书处理模块。这几个模块涉及了通过 MFC ODBC 访问数据库的主要技术细节。其他模块的实现方式与这几个模块类似，在此不再赘述。读者可参考光盘的源代码。

小　结

在这一部分，通过介绍一个简单的图书管理系统来说明了传统的结构化系统的分析和设计方法。传统的结构化系统分析和设计主要是通过对整个系统的数据流和要完成的功能的角度出发来理解和设计整个信息系统。

此外，还介绍了如何通过 Visual C++访问数据库。在这里只介绍了通过 MFC ODBC 访问数据库。这种方法比较简单，适合于在这一部分介绍。

第三部分　分布式应用系统

—— 考试系统

随着计算机计算能力的提高、网络技术的发展以及企业对降低运算成本的迫切要求，企业信息系统逐步由集中式计算向分布式计算演进。90 年代 C/S 技术风靡全球。在 C/S 环境中，大多数应用逻辑位于客户机并使用集中或分散的数据库服务器。较之主机环境而言，C/S 环境使企业计算大大地迈进了一步。但在实践中，人们发现这种方法可能导致过高的网络流量并使维护变得困难。作为这类问题的解决方法之一，人们提出了多层应用体系结构，即将某些应用逻辑独立于客户机及数据库服务器，发布在一个或多个物理服务器上（称为应用逻辑服务器），由应用逻辑服务器负责访问数据，而大部分的更改也只需在服务器端进行。

多层应用体系结构需要中间件的支持，目前分布式对象中间件成为技术与市场的一个主要方向。分布式对象技术允许开发者将运行于分布在网络中的不同平台上的对象集成为应用。对象的封装和复用技术可以有助于控制系统的改动。它们可使服务器编程者隔离改动并将维护的费用最小化。问题域和解题域的紧密映射允许对企业建模并构造能够保证快速而准确地进行改动的实现。

为了方便与简化多层应用的构造，人们提出了“应用服务器”的概念。应用服务器在分布式对象中间件的基础上实现了一些多层应用所必需的服务（如安全、事务管理、目录管理等等），使得应用逻辑服务器的开发者无需关注这些基本服务的具体实现，而可以专注于业务逻辑组件的开发。这极大地方便了多层应用的构造、分发与管理。可以预见，基于分布式对象的多层应用体系结构将成为企业计算的新的技术趋势。

本章将从软件体系结构及设计方法学的角度出发，以考试系统的原型为主要手段，介绍了基于分布式对象的多层应用系统的开发。