第6章　先进的窗口和菜单

一旦了解了Oracle Developer的Form Builder组件的处理过程，就得到了一种结构，可以在这种结构中扩展表单应用程序的范围。另外有两种要用 Oracle Developer研究的问题，可以使你设计大规模完整的应用程序的能力更加圆满。这就是窗口和菜单。用 Form Builder的这些特性来设计，只须很少或完全不用编程。

在第4章中，实验模型的表单只在单个窗口上呈现单个画布。本章讲述如何利用多个画布把应用程序扩展到多个窗口，这些特性为提供建立对话框、工具条和警报的能力。

在第4章中，应用程序的缺省菜单不包括在用户可以利用的范围中。用户可以建立自己定制的菜单，把表单的报表、图形或想从表单中调用的无论什么其他的东西结合起来。

6.1　窗口

在Form Builder中制作窗口意味着要作两个事情之一：创建窗口的内容或创建和管理窗口本身。下面的讨论扩展第 4章的内容，讲述如何充分利用 Oracle Developer画布和窗口在应用程序的设计中起重要作用。

6.1.1　描绘画布

回想在第2章和第4章中用画布来显示块和项，在Layout Editor中编辑画布，画布成了建立表单中所做的许多工具的基础。那么确切地说，画布是什么？

画布是摆放样本文本和项目的背景。每个项目在它们的 Property Palette上都准确地涉及一个画布。可以在不同的画布之间分配块的各个项目。

画布并不是孤立地作为一个界面对象。要观看画布和它的项，必须在窗口上显示画布。Oracle Developer使这些独立的对象为你建立窗口，从而为画布提供一个视图：包括部分画布

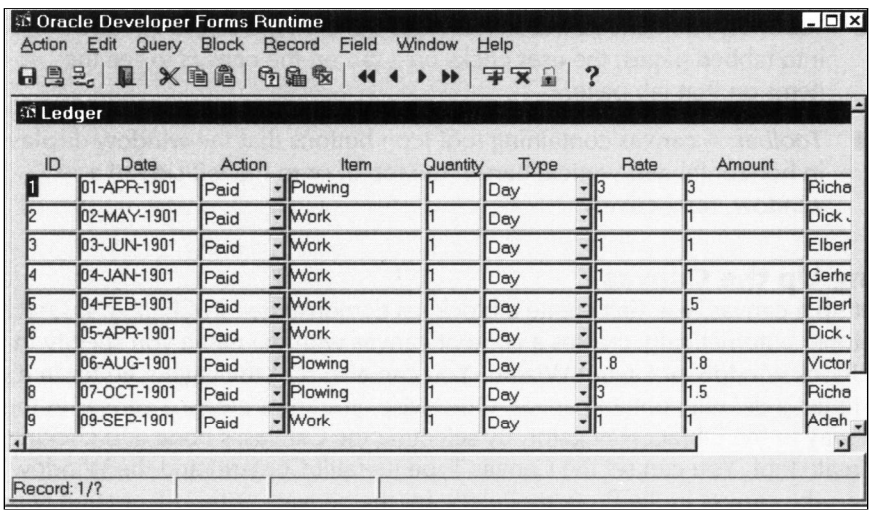

图6-1　Ledger应用程序的一个窗口

或全部画布的长方形。例如图 6-1显示Ledger应用程序的一个窗口，这个窗口显示了基础画布的一大半。可以通过窗口看到的这部分画布叫视图。窗口有水平滚动条和垂直滚动条，可以通过滚动画布来观看视图。

这样把显示方式和显示内容之间的工作划分成独立但是相当的两部分工作，使建立表单与显示它们的窗口的大小无关，这样作加大了视觉范围但却缩小了界面。你会发现用户对于需要不断滚动通过画布的界面不满意。好的界面应在所有的时间显示画布上的所有域和项。但是，就像生活中的大多数状况一样，与能够同时显示项比较，有更多的需要。窗口和画布提供了多方面的适应性。

1. 选择适当的背景

为了得到多方面的适应性，首先要考虑画布所使用的类型。某些画布不只是可以在窗口内滚动，还可以做更多的事情。有四种类型的画布：

- 内容画布(Content)：包含窗口“内容”的画布，每个窗口至少有一个内容画布，而且通常只有一个。
- 层叠画布(Stacked)：在另一个画布上面显示的画布，通常包括和基础的内容画布上的项分开的组内的一些项。
- 标签画布(Tabbed)：一种内容画布，它把窗口的内容分成指定的页，用户单击画布的标签页观看在那个标签页上的项目内容。
- 工具条(Toolbar)：包含工具图标的画布。分别在窗口的顶部或左边上的水平和垂直条上显示这些图标按钮。

2. 填充画布

为了建立一个画布，首先要创建一个块，就像在第4章中说明的那样。Oracle Developer用在New Block窗口或Layout Wizard中指定的名字自动创建内容画布。也可以在新表单中启动Layout Editor得到缺省的内容画布。否则，必须在Object Navigator中用通常的方法，通过选择Canvases结点并单击Creat工具来创建画布。可以在画布的属性设置板上设置 Canvas Type属性(缺省设置是Content)，显示画布的窗口。像图6-2显示的那样。

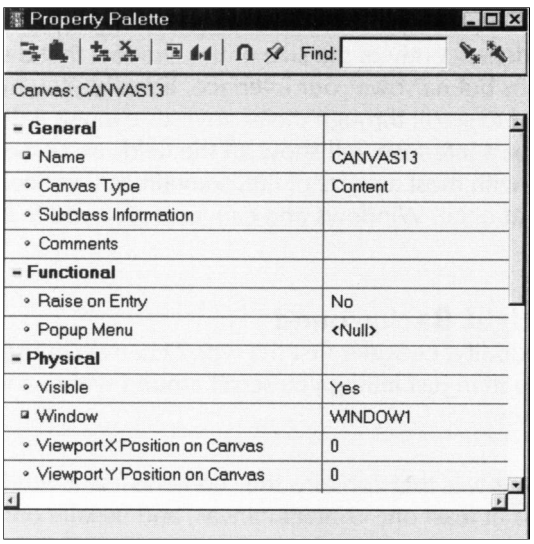

图6-2 画布窗口

要查看Layout Editor中的画布，双击这个对象的Object Navigator图标。第4章讲述过如何利用Layout Editor把项目和样本文本放到画布上，这里唯一的区别是由 scratch(擦除)建立画布，而不是用创建块时 Oracle Developer产生的一组项目的画布去工作。可以在 Object Navigator中创建一个项，然后把这个项的 Canvas属性设置为新画布。也可以在 Layout Editor中创建项目，并通过它的Property Palette改变这个项目的属性。可以创建第二个画布，并通过改变已经存在的项目的 Canvas属性把项目从基础画布移动到它的上面。如果你把这个属性设置为NULL，那么这个项将不出现在画布上，这可以定义一个项目不显示它的值。

可以通过View | Show Canvas菜单控制画布背景的显示。当Layout Editor有效时，可以得到这个菜单项。如果将这个菜单项设置为 off，那么将直接在项目下的背景上看到网格。如果把它设置为on，那么将看到没有网格的画布。其次，如果想重新决定画布的大小，可以通过拉动画布右下角的选择手柄来做。要了解这个过程，可接通 Viaw |Show Canvas菜单项，然后在画布的右面或底部边缘上单击。也可以改变画布属性设置板 Width属性和Height属性。

使用View | Show View菜单可以观看视图。如果在画布的窗口上显示画布，这个视图就是所看到的画布的区域。 Layout Editor把视图显示为带有手柄的一个黑色的长方形，可以利用这些手柄来重定缺省窗口的大小，也可以通过改变窗口 Property Palette上的窗口的Width和Height来改变这个尺寸。通常，这项工作是把窗口的宽度和高度设置得和内容画布的 Width和Height相同，这样可以使窗口正好包围画布。也可以通过拉动这个长方形，或通过改变画布的X | Y Position属性来移动视图，这样，可以把画布定位在窗口内的某个位置上而不是在它的左上角。

注意 如果是层叠画布，则在画布的边界有另外的视图。可以在层叠画布上放一个滚动条并在窗口内滚动它。“ 层叠画布 ”小节会给出更多的细节。

可以利用所有这些特性来放置内部项和它们的画布，在显示它们的窗口中取得最好的效果。例如，通常希望窗口紧紧地包围着画布，而且希望给出的画布的项目布局尽可能小。如果可能，希望用户不用滚动就可以看到窗口内的所有的项，使表单使用容易些。为此，重新把画布的尺寸定为正好包围项的布局，然后把画布的 Width属性和Height属性拷贝为对应的窗口的Width和Height属性。这样，就把窗口的尺寸精确地定为画布的尺寸了。

用户可以通过选择一个边缘并拉动它来重定画布的尺寸，这就是说，可以为了屏幕显示而调整窗口和画布的大小。可以确定一个初始窗口尺寸，从画布的一边滚动到另一边，以适合较小的视频显示。而用户可以用高分辨显示重定画布的尺寸，使得可以同时看见每一样东西。

通过选择画布并单击Delete 工具，可以删除Object Navigator上的画布。通过把它们的画布属性设置为NULL，Oracle Developer可以清除任何与被删除画布有关的项。

在显示窗口时，自动显示窗口的基础画布。可以通过触发器中的 Show_View和Hide_View内部子程序控制其他画布的显示。

3. 层叠画布

层叠画布允许产生几个特殊的效果：

- 可以在背景的独立簇中创建一组按钮或其他项，用图形把簇和基础内容画布区分开。
- 可以创建可分离和可重用的项目组，比如按钮组。通过复制，可以在不同的画布上再使用(第10章讲述细节)。
- 可以有计划地隐藏或显示层叠画布，创建一个当用户执行某个动作时自动变化的视图。
- 可以在多个动态变入变出的内容画布的上面显示不变化的文本和域。有一种方法，不用创建很多个项就可以有重复的元素。
- 在滚动基础的内容画布查看其他项目时，可以让在多记录显示中的一组项目留在屏幕上。例如，当通过滚动检查其余列时，可以在改变的列中显示主关键字，使得不管在画布的什么地方总可以看到主关键字，这个特殊效果就像电子表格中的多显示视图一样。

在主-从表单中，当在合适的窗口上同时要看的项太多时，可以使用一个技巧，但是可能帮助不会太大。例如，如果目标显示平台是一个 640×480 VGA的Windows显示器，而表单和Ledger表单类似，那么，将没有办法让用户可以同时看见具有全部的域的记录。 Windows采用滚动可以解决这个问题。把应用程序改为显示面向人员的 Ledger内容，显示给定的人员的所有Ledger项，个人的信息在应用程序的顶部，而在它下面显示 Ledger信息多个项的多个记录。如果有滚动，则当滚动屏幕时，就会丢失个人的信息。而滚动层叠画布可以帮助应付这种情况。

1) 单击Create 工具，创建Person数据块，利用Data Block Wizard并且只显示Name列。在Layout Wizard中创建一个新的Content画布，并使数据块的类型为Form。

2) 在Object Navigator中选择Person数据块后，在Create工具上单击，创建Ledger数据块。利用Data Block Wizard添加所有的Ledger列和Person.Name查找列。

3)在Ledger数据块的Layout Wizard中创建一个画布作为新 Canvas，并把Type 设置为Stacked (层叠的)，把数据块类型设置为Tabular ，并用滚动条使它包含5行数据。

4) 把新表单作为Person_Ledger保存。

5) 导航到Object Navigator中的Person块Relation结点上和(Person_Ledger | Data Blocks | Person | Relations)并单击Create工具创建主-从关系。像在第4章那样根据Ledger和Person之间的对象关系创建关系。

6) 在Object Navigator结点上双击，打开Layout Editor中的Person画布视图。

7) 为了帮助看清画布之间的关系，选择View | Stacked Views菜单项。显示Stacked/Tab Canvases对话框(见图6-3)。提示选择要显示的层叠画布。这个对话框列出所有的层叠画布并让你用一种不平常的方法接通和判断它们。代替用顺序的单击或双击操作不断地点击画布，用单击画布来进行选择画布，用CTRL-单击操作来淘汰画布。就是说，在层叠画布名字上单击来显示它，并在它上面CTRL-单击操作从显示中除去它。

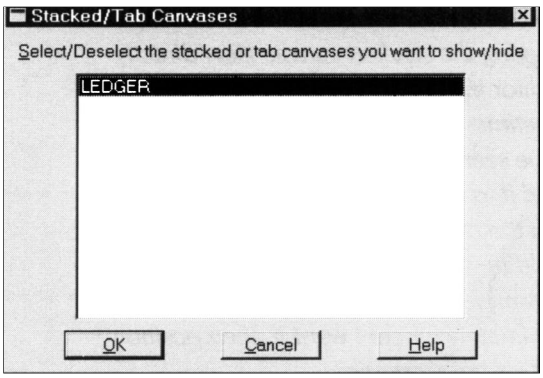

图6-3 Stacked/Tab Canvases对话框

8) 单击OK，在Layout Editor中显示或删除层叠画布。对于Person-Ledger表单，所得到的Layout Editor显示出的Ledger块完全遮住了Person块。

层叠画布视图是画布的可见部分，虽然内容画布的视图与窗口的边界有关，但层叠画布的视图与窗口没有关系。可以通过在 Object Navigator中选择画布，查看层叠画布的边界，然后显示Layout Editor。在层叠画布的边界上可以看到控制尺寸的手柄，而且在内容画布的标尺变成灰色时，画布的标尺是活动的。拖动手柄改变视图的大小以显示更多或更少的层叠画布。

定位层叠画布最容易的方法是在画布的 Property Palette中设置视孔坐标。

9) 在Object Navigator中选择Ledger画布。

10) 通过选择Tools | Property Palette菜单项显示弹出菜单上的Property Palette，或通过右

击画布对象，显示属性调色板。

11) 找出Viewport header和Viewport Position两个属性。把Y位置设置为0.8英寸，把层叠视图放在Name 项的下面。图6-4显示了在Layout Editor中的结果，可以看到利用明暗区别的层叠画布标尺。

注意 不要混淆出现在Physical属性组上Viewport Y Position和Canvas属性上的Viewport Y Position。后者定位观测孔中的层叠画布，不是观测孔本身。这是让你第一次显示层叠画布时，把滚动画布放置在不是最左边或顶部的其他地方。也可以在Layout Editor中直接拉动层叠画布视图，而不通过它的属性来设置它。有时候层叠画布的边界看不见，而且要找到它来拖动可能有点困难，可以在画布背景单击，然后寻找重定尺寸手柄，再单击定尺寸手柄并保持拉到某处。我通常喜欢拖动画布，直到它接近我希望的地方为止，然后通过设置属性再准确地定位它。

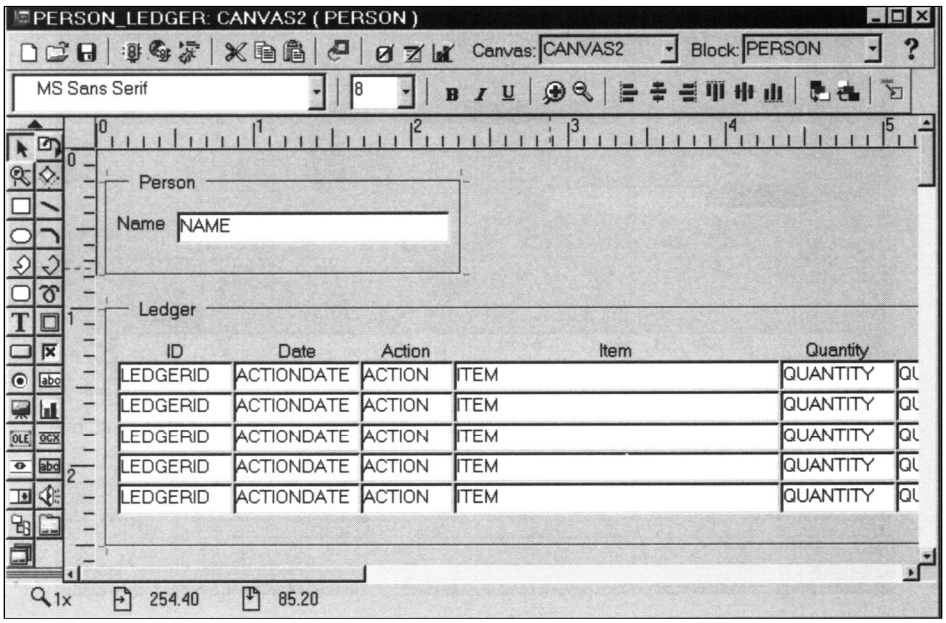

图6-4 在内容画布内显示的层叠画布的Layout Editor

12) 把Show Horizontal Scroll Bar(显示水平滚动条)属性设置为YES，使水平滚动条有效。

这个滚动条出现在视图底部边缘的下面，所以必须适当地确定视图的大小，使滚动条位于所希望的地方。在我们的例子中，这个视图将扩大到刚好在第五个 Ledger记录的最后一行的底部，滚动条正好出现在记录块的下面。在 Layout Editor中看到的滚动条是内容画布的而不是层叠画布的。

13) 最后，必须通过Layout Editor边界或通过画布和窗口的Property Palettes，确定内容画布的尺寸，使它包含它自己的项和层叠画布，必须确定窗口的尺寸使它包含内容画布。

运行这个例子所得的显示展示在图6-5中。

注意，在图6-5中有两个水平滚动条，下面一个是和窗口有关的滚动条，只有画布扩展超出了窗口时，这个滚动条才出现。在这里是刚超出一点点。上面的滚动条和层叠 Ledger画布有关，可以利用它在Amount域记录滚动条上滚动。

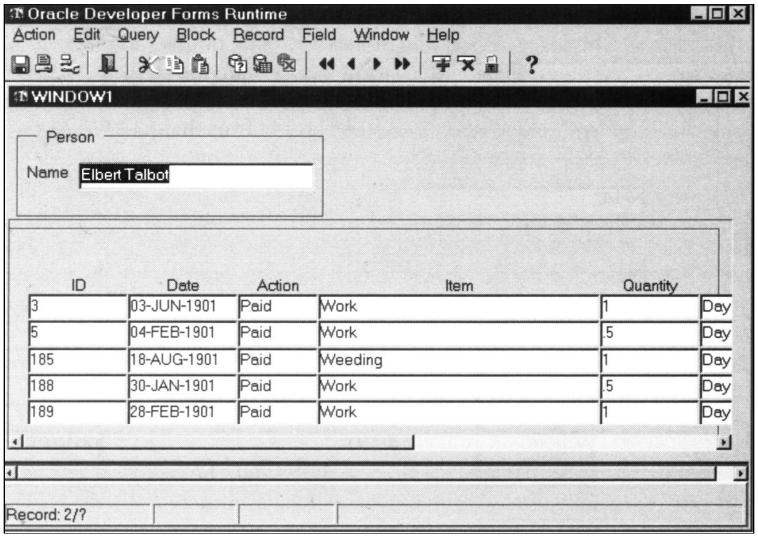

图6-5 Person Ledger表单和它的滚动视图

注意 如果你看不到层叠 Ledger画布，应检查两样东西。首先，确保 Person画布在 Navigator Lagout组中的次序是第一，第一个先显示，第二个画布在它的顶部显示。如果 Person画布排在第二，那么就遮住了层叠画布Ledger。其次，确保层叠画布观测孔把层叠画布框架放置在人员块框架的下面(查看兰色阴影线和框架手柄)。试着把观测孔往下移动一点点看会发生什么。如果层叠画布的项目遮住了内容画布的项目，即使是遮住一小点，那么在导航它时，内容画布自己升高，显示全体被遮住的项目。这将是显示画布之后表单做的第一件事。你会看见层叠画布出现，然后消失，就像是变戏法一样。

4. 标签画布

标签画布在一个或多个标签页中显示它的项目，标签页是一种微型画布，与标签文件夹类似。在标签的标记上单击， Oracle Developer就显示单击的那一页。利用标签画布可以按组为单位来排列项。这就允许显示某一组而隐藏另外一组，为用户简化了显示。可以利用这个特性建立有许多项目的表单，但可以用这样一种高级结构和控制的方法把它们提供给用户，使表单非常容易使用。不仅如此，而且标签画布对 Oracle Developer支持的所有图形接口都是完全可移植的。

第4章说明了创建标签画布的基本过程，这个部分讲述如何把标签画布当作层叠画布来处理。这样可以把一个主要内容的画布和一系列从属内容的画布结合起来。例如，第 4章中的 Worker Skill表单用在一个标签页上的人员和在另一个标签页上的技能，把一个人和一组技能连接在一起。这是一个缺点，在寻找技能时，要把人的内容从显示中除去。反之亦然。理想的情况是，用户同时可以看到人员的信息和技能的信息。

WorkerSkill表单的改型是更通用的HumanResources表单，它用主-从关系把住所与技能和人员连接起来。这个表单有三个数据块，对应于 Person、Lodging和WorkerHasSkills三个表，Person画布在内容画布中有Person的名字。在布置另外两个数据块时，不选择内容画布或层叠画布，而是选择带有一个Lodging页面和另一个WorkerHasSkills页面的独立的标签画布。第一次通过Data Block Wizard时，选择一个新的标签画布。第二次通过时，选择第一次通过时创建的画布，然后选择(New Tab Page)。Lodging画布有表单布局，而Skills页有表格布局，列出

不同的技能。

一旦完成了画布的布置，给定了尺寸和布置好它们的项目，就可以利用层叠画布组合Person画布和Ledger画布的方法，组合这两个画布。

1) 显示Person画布的Layout Editor并选择View |Stacked |Views菜单项。

2) 从Stacked/Tab Canvases 对话框的列表中选择标签画布。像前面一样，选择调整在内容画布项顶部下方的画布。

3) 选择标签画布并拉它向下，或改变它的 Viewport Y Position 为大约0.25英寸的值或稍大。

这就产生了显示在图6-6的布局和显示在图6-7的运行的表单。

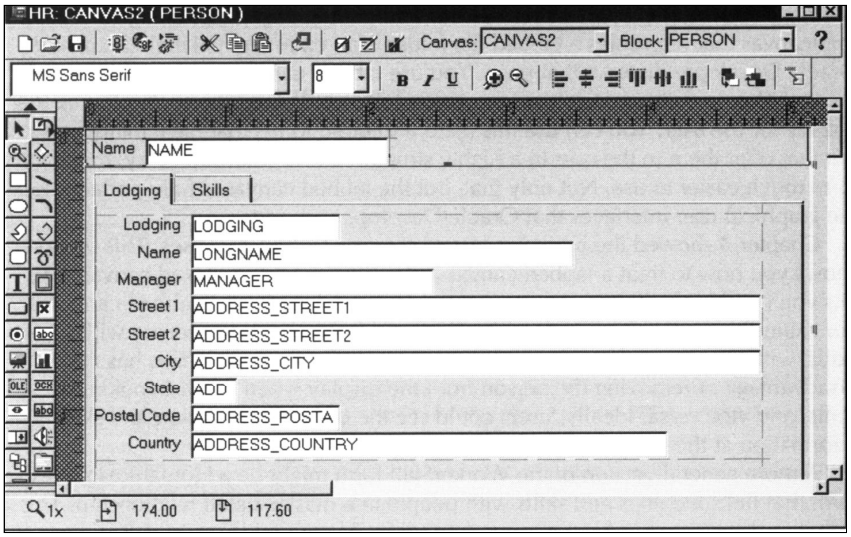

图6-6 带有层叠标签画布的Layout Editor

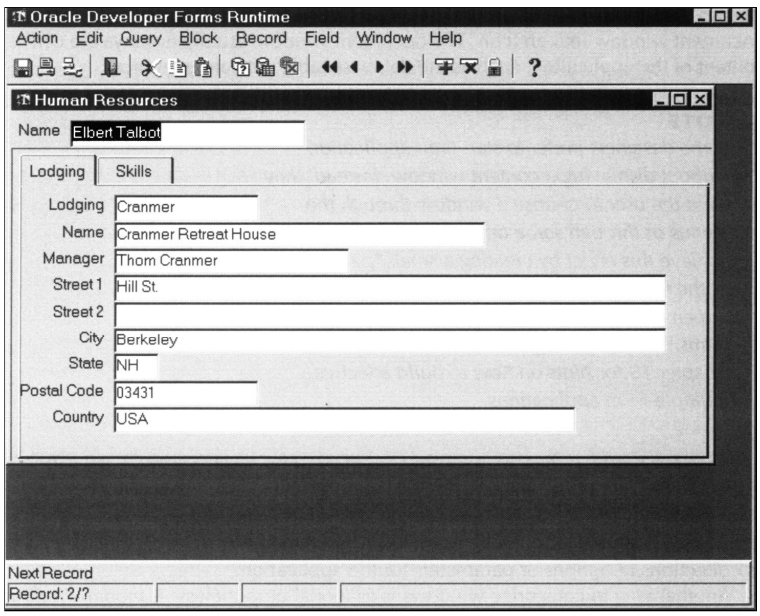

图6-7 运行时的标签画布

布置完画布以后，下一个步骤是通过窗口显示它。以下部分讲述创造性地利用画布的不同种类的视图，这些视图是可以创建的。

6.1.2 通过窗口观看

用画布组织表单的内容，通过窗口让用户看到它。通过在表单中添加画布和窗口，可以创建易于理解和使用的表单。每个窗口至少有一个定义窗口内容的内容画布。

1. 利用多种文档

微软Windows平台引入多种文档接口(MDI)，在应用程序中组织多个窗口的使用。 Oracle Developer通过采用特定的假设和可利用的条件采用 MDI体系结构并把它扩展到其他平台。

MDI应用程序有一个“ 应用程序窗口 ” (application window)，它拥有应用程序中所有其他的窗口。“ 应用程序窗口 ”没有画布，而是显示其他的窗口。主菜单属于“ 应用程序窗口 ”，而这个窗口总是打开的。这个窗口的子女有表单窗口，而且还可以是文件或对话窗口。

注意 除了Windows和NT以外，其他平台不使用MDI体系结构。Motif和Macintosh不采用它。

文档窗口(document window)是这样一种窗口，它完全被包含在应用程序窗口内。如果移动文档窗口超出了应用程序窗口，那么移动超出这个窗口的那部分文件将消失。通常，在应用程序窗口显示区域的一个重叠组或堆砌组上显示文档窗口。也可以把文档窗口增加到最大值，占去应用程序窗口的整个显示区域，或者把文档窗口减小成图标。文档窗口通常显示应用程序的主要内容，比如数据库表数据和图形。

注意 一些设计者喜欢启动他们的应用程序时不显示内容窗口。他们希望用户通过菜单或通过某些其他的界面来选择窗口。可以通过创建一个小的“ 父 ”表单作为运行表单来取得这种效果。这个表单可以有一个溅开的(Splash)屏幕图形或一组打开其他表单的按钮。有关如何建立有效的多表单程序请参看第10章“ 使用多重表单 ”的内容。

对话框是一个窗口，它与应用程序窗口无关。应用程序窗口不限制对话窗口，也可以把对话窗口移到应用程序窗口外面(如果它的属性允许这样的话)。对话框通常包含有让应用程序和用户交互的域，比如应用程序选项或参数的集合等。

窗口分类的另一种方法是分为模式的或无模式的。模式窗口 (modal window)要求用户在应用程序的任何其他窗口做任何事情之前响应并消除这个窗口。因为对话框的主要功能是要得到往前进行操作所需要的信息，所以对话框通常是模式的。也可以有无模式对话框。如果这个对话框的信息不是以后处理的部分信息，那么在应用程序中做其他事情时，这个对话框可以继续存在。例如搜索字符串的 Find对话框，就可以是无模式的。无模式窗口 (Modeless window)是在移动到另外一个窗口时并不消除第一个窗口。通常模式窗口有显著的特性，比如没有滚动条、具有固定的尺寸、不能缩小为图标等。在某些平台上，模式还扩展到其他的应用程序。就是说，在系统中做任何其他事情之前，必须响应这个窗口并消除它，而不只是应用程序(系统模式与应用程序模式相反)。

在Oracle Developer应用程序中，应用程序窗口和当前执行的表单相对应。应用程序的主菜单是表单的菜单。通过指定它们的画布为表单的 Horizontal MDI Toolbar属性和Vertical MDI Toolbar属性，也可以具有水平的和垂直的工具条。

通过使用内部过程Set_Window_Property中的名字FORMS-MDI-WINDOW，可以在触发器的编码中用程序设置应用程序窗口的属性。

2. 创建窗口

在创建新表单时，就得到一个新窗口。第一个画布将自动地使用这个窗口。如果想创建更多的窗口，通常的方法是使用Object Navigator选择Windows标题并在Create工具上单击。

利用在Window Property Palette上的属性，几乎可以确定用窗口做的每一件事。“功能”部分可以建立窗口的基本交互能力。展示在表 6-1的属性叫做“提示”，因为它们不是所有的都适应于所有的平台。

要建立窗口的基本结构，可以利用表6-2中的属性。

表6-1 功能Forms窗口属性

属 性	说 明
Close Allowed	是否可以用平台特有的关闭命令关闭窗口
Resize Allowed	窗口的高度和宽度是否固定，或用户是否可以重定窗口的尺寸，Resize Allowed设置为NO只在Maximize Allowed设置为False时有效
Minimize Allowed	用户是否可以把窗口缩小成图标
Inherit Menu	窗口是否显示表单菜单，在微软Windows中无效
Move Allowed	用户是否可以移动窗口
Maximize Allowed	用户是否可以用平台特有的操作最大化或放大窗口，只在 Resize Allowed为False有效

表6-2 结构Forms窗口属性

属 性	说 明
Primary Canvas	在作为主内容视图的窗口中显示的画布的名字。主内容视图是一个基础视图，当你打开窗口时，窗口总要显示它，这是可任意选择的，但是如果你使用Show-Window来显示窗口或者如果使用层叠画布并直接导航到它们那里，就可能是必须的了
Vertical Toolbar Canvas, Horizontal Toolbar Canvas	在垂直或水平工作条显示的画布，请参看“创建工具条”的内容
Window Style	窗口、文件或对话的种类
Modal	窗口是否是模式的
Hide on Exit	当导航离开一个无模式窗口进入下一个窗口时，是否关闭和使这个窗口不活动，只在Modal是False时有效
Direction	文本流的方向，Default (缺省)设置通常是所希望的方向，因为它从Oracle的NLS建立中继承正确的方向
Icon Filename、Minimized Title	图标文件的名字和当用户最小化窗口时显示的标题，只有当Iconifiable为True时有效。这个图标文件的路径和特性取决于平台，而且使用图标文件要求作额外的工作把应用程序停泊到其他平台
Show Horizontal Scroll Bar、Show Vertical Scroll Bar	是否要显示水平和垂直滚动条，如果画布比窗口大，可以在画布上滚动视图，只对无模式窗口有效

注意 当你导航到在当前窗口外面的画布上的项时，Oracle Developer自动地滚动窗口的画布，也可以使用Set_Canvas_Property内部子程序通过触发器，或者通过画布的Canvas属性上设置 Set_View_Property 的X或Y的位置来控制滚动。应当使用Set_Canvas_Property，而Scroll_View只是提供来用于Oracle Developer前面的版本向后的兼容性的。

Oracle Developer可以很容易地显示和关闭窗口，为通过导航它自动控制这个过程。当导航到没激活的画布上的项目时，Oracle Developer在它有关的窗口显示这个画布。当导航离开这个画布进入不同窗口的另一个画布时，如果把 Remove on Exit属性设置为True时，Oracle Developer自动关闭第一个画布，否则，这个窗口只是放到后面一层，但仍然显示。

注意 有关模态窗口的属性请参看以下部分的内容；Remove on Exit(在退出时除去)属性不适用于模态对话。

也可以通过用内部的导航子程序或用内部的 Show_Window和Hide_Window程序导航，在触发器中打开和关闭窗口。如果在当前活动窗口后面有一个静止的显示的窗口要这样做，那么Oracle Developer把这个静止的窗口提高到窗口堆堆栈顶部并激活它。可以用导航或用Show_View和Hide_View，采用同样的方法显示或隐藏特殊的画布。

3. 创建和关闭模式对话框

创建一个模式的对话框的步骤：

1) 把Window Style 设置为Dialog (对话) 并把Modal 设置为True。

2) 与它们的块一起创建内容画布和组成对话框的项，如果需要，把画布和对话框联系起来。

3) 固定尺寸对大多数对话框有利，所以把 Fixed Size属性或Resize Allowed属性(允许重定尺寸)设置为True，并把窗口的Width 和Height属性设置为画布的尺寸，确保窗口显示整个画布。

4) 使对话框比这个窗口或比它将要在其上出现的窗口小，这可以使用户知道应用程序的模式状态。

应该把模式对话框的显示建立在适当的事件上。做这项工作有两种简单的方法：第一，可以按照用户在对话的画布导航上的项的次序排列项目导航的次序。也可以不考虑导航的次序，用按钮触发器或菜单命令明确地导航。在任何一种情况下，当用户导航到对话的项目时，Oracle Developer自动显示对话框。

第二，可以从按钮触发器或菜单命令中调用 Show_Window内部子程序，这个程序显示窗口并导航到对话框画布的第一项。

Oracle Developer中的Modal属性防止用户用鼠标离开对话框导航。它不允许，但暂停(Suspend)按键允许。所以用户可以利用导航键导航出去。当用户这样做时，Oracle Developer自动地关闭这个对话框。然而这种特性不是用户在对话框所期待的。Modal对话框作为活动的窗口应该保持在屏幕上，直到用户用确定的动作消除它们为止，通常是通过在 OK或Cancel按钮上单击。

有几种方法关闭(断开)按钮导航键：

- 用对话框内的项把这些块的Previous Navigation Block属性和Next Navigation Block属性设置为这个块本身，使导航循环。如果这样做，应该有单独的块和所有出现在对话框画布上的项。
- 在拥有正在显示的项的块上创建Key-Others触发器，包含一个null语句：

```
NULL;
```

除了已定义了键触发器的键以外，断开了其他所有的键。需要最小限度地定义 Key-Next-Item和Key-Previous-Item以便允许在对话项之间跳格。

然后需要在OK和Cancel按钮上建立触发器，或用任何一种方法指示 Oracle Developer关闭对话框。例如，When-Button-Pressed触发器导航离开窗口时，Oracle Developer自动关闭这个对话框。必须要用在对话画布外面的块导航到无模式窗口的项。

可以从一个模式对话框中打开另一个模式对话框。例如，可以有一个 Options按钮，显示选项所扩展的第二个对话框。还可以达到某些奇特的效果，比如用一个 Extended按钮，产生第二个更大的对话框代替当前的对话框。自己试验作一下。

4. 创建工具条

工具条是沿着窗口顶部或左边的一条图标。要创建它，必须首先创建包含按钮项和样本图形的工具条画布 (画布类型是Horizontal Toolbar 或Vertical Toolbar)。可参看前面部分有关创建画布的内容。

1) 为画布指定将要显示工具条的窗口的 Window (窗口)属性。

2) 给工具条设置宽度和高度，设置值为所设计的应用程序最优的尺寸。例如，希望水平工具条的工具条画布的宽度和窗口的宽度相同，而高度正好包围图标。

3) 下一步是为窗口对象的Horizontal Toolbar 或Vertical Toolbar 属性设置为适当的画布的名字，告诉窗口在适当的位置显示这些工具条。

注意 在微软的Windows中，也可以通过表单对象的Horizontal MDI Toolbar或Vertical MDI Toolbar属性显示MDI应用程序窗口工具条。Oracle Developer对这些工具条忽略工具条画布的Window(窗口)属性。

单个窗口可以有多个工具条画布。 Oracle Developer使用标准的规则来显示它们。要显示它，只要导航到工具条上的第一项上，而不是原来的工具条上。通常，动态变化的工具条并不好，因为这种变化会使用户感到混乱。可以用不同的图标代替禁止工具 (例如，暗淡、灰色、十字画叉，等等)。可以利用这个特性禁止和允许工具条上的工具有效。

因为工具条上的项就是块上的项，必须进行一些专门的测量，确保这些项所起的作用和表单的其他项都不同，特别是有关导航的作用。通常，把工具条的按钮项作为单独的项比较好。把前一块的Next Navigation Data Block属性设置为是对应块而不是工具条块，把这个块从块导航中取出，可以用 Property Palette去做这件事。

4) 对于每个项，把Property Palette中的Keyboard Navigable 属性和Mouse Navigate属性设置为NO。把这些属性断开是防止在工具条上的工具上单击引起游标导航进入工具块。当在工具上单击时，用户能够保证停留在他们所在的块内。

注意 如果不用用户自己的菜单代替标准的Oracle Developer菜单来运行MDI应用程序，那么将会看见在MDI窗口顶部的主菜单下面出现工具条。这是一个特殊的菜单工具条。缺省的菜单工具条包含有和普通菜单项对应的图标，比如Enter Query、Next Record等。不提供自己的完整的主菜单模块，便不能改变这个工具条。也不能移动它。有关增加定制菜单的细节，请参看本章后面的“菜单”部分的内容。

5. 创建警报

警报是一个模式窗口。它显示信息或询问简单的问题，要用户给出 Yes / No类型的响应信息。Oracle Developer有一个专门用于报警的对象，可以简化对那些非常普通的窗口的编程。

注意 当在PL/SQL代码中使用Message函数或当存在一个内部消息而且有多个信息时，

Oracle Developer自动显示警报。通常，消息出现在状态条中。如果有多条信息，则用户只能看见最后一条信息，并丢失临时的信息。而Oracle Developer把临时信息作为专门的警报显示。通过使用内部的Synchronize子程序，在调用内部的Message子程序之后，立即在PL/SOL代码中显示这些信息。有时候可以避免这些情况，比较好的方法是，对错误使用明确的报警对象去完全代替信息系统。例如，可以为标准的错误建立一组可再用的报警对象并把它装入标准对象库中。

要创建警报，在对象导航器上选择 Alerts标题，并用通常的方法在Create 工具上单击。有三种警报。通过警报属性设置板的 Alert Style属性规定使用哪种类型：

■ Stop (停止)：显示一个停止信息图标和信息 (见图6-8)。

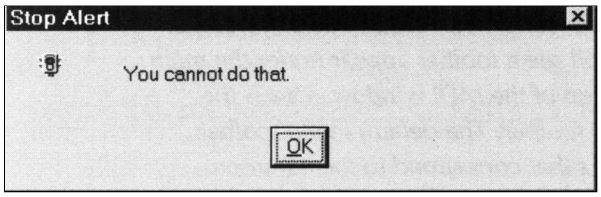

图6-8 Stop Alert对话框

■ Caution (警告)：显示一个感叹号图标和信息 (见图6-9)。

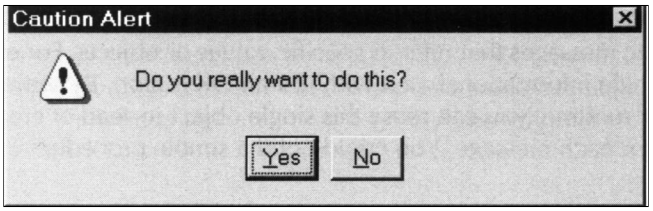

图6-9 Caution Alert对话框

■ Note (注意)：显示一个信息图标和信息 (见图6-10)。

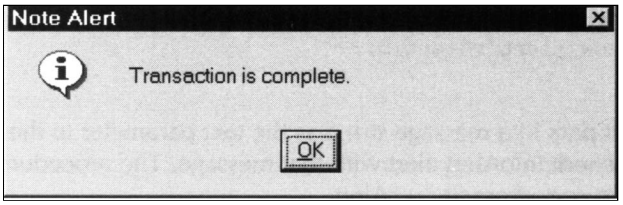

图6-10 Note Alert对话框

在Message属性中输入要在警报中显示的信息。可以输入多达 200个字符(虽然不是所有的平台都可以显示全部数目的字符)。

用Property Palette中的Button1、Button2和Button3 标明显示警报的按钮。如果不输入标记，那么警报不显示按钮。例如，如果使用缺省的 OK和Cancel 设置值，那么就得到带有“ OK ”和“ Cancel ”两个标记按钮的警报。如果删除 Button2的 Cancel设置，那么就得到只有“ OK ”标记的单个按钮。至少必须标明一个按钮。还应该指定一个按钮作为缺省按钮。当用户按ENTER键时，Oracle Developer选择缺省按钮。

利用Show_Alert(显示报警)内部子程序由触发器代码显示警报。这个子程序根据用户选择

了的按钮返回编号ALERT_BUTTON1，ALERT_BUTTON2或ALERET_BUTTON3，然后可以测试返回的编号，并在触发器代码中采取适当的动作。例如，下面的代码用 Caution警报提示用户决定是用触发器继续，还是不继续。按钮 1是Yes，而按钮2是No。

```
DECLARE
  vAlertButton NUMBER;
BEGIN  -- Initial processing
  vAlertButton := Show_Alert('Caution');
  IF vAlertButton = ALERT_BUTTON1 THEN
    NULL; -- Continue processing
  END IF;  -- do nothing for button 2
END;
```

用少量代码，也可以创建可重用的警报对象和警报，来显示引用专门的值或对象。例如，只用一个OK按钮创建单个指示性的警报。通过在运行时设置信息文本，可以再利用这个对象，而不必为每个信息创建一个单独的警报。可以用以下代码在库中建立一个简单的程序。

```
PROCEDURE Show_Info_Alert(pText IN VARCHAR2) IS
  vAlertID ALERT := Find_Alert('GenericInfoAlert');
  vDummy NUMBER;
BEGIN  Set_Alert_Property(vAlertID, ALERT_MESSAGE_TEXT, pText);
  vDummy := Show_Alert(vAlertID);
END;
```

然后以信息行的方式传递文本参数到这个程序调用中，用传递的信息显示 GenericInfoAlert警报。这个程序有意忽略从Show_Alert返回的代码。

还可以建立更复杂的可重用的函数，使信息接受参数，通过并置参数创立信息文本，显示动态信息，并返回适当的按钮编号。为了在其他的模块中再使用，应该把这个函数放入库内。

```
FUNCTION Show_Info_Alert(pText1 IN VARCHAR2, pText2 IN VARCHAR2)
      RETURN NUMBER IS
  vAlertID ALERT := Find_Alert('GenericInfoAlert');
  vMessage VARCHAR2(200);
BEGIN
  vMessage := 'The '||pText1||' is '||pText2||'.';
  Set_Alert_Property(vAlertID, ALERT_MESSAGE_TEXT, vMessage);
  RETURN Show_Alert(vAlertID);
END;
```

6.2 数据块

Oracle Developer提供了几个与块和块的项有关的特殊特性。用这些特点可以显著地改善表单的使用性能。如果使用列表，那么对不同的用途，可以用几种不同的方法构造它们。利用单选按钮可以容易地创建互不相容的选择项，利用计算域使总和数的自动显示和表单中的其他计算工作变得非常容易。使用正确的日期格式掩码，可以大大减少数据库中与日期和时间有关的错误。最后，可以利用程序的封装与 Data Block Wizard配合，把数据块建立在存贮的程序的基础上而不是表的基础上，从而增加一个面向对象和对系统隐藏数据的方法。

6.2.1 列表

列表是一个表单的项目。它在独立的位置显示文本项的列表。理想的情况是通过查看选择列表并选择一项让用户在几个互不相容的选项中选择。已经见过两种列表 —— 在第4章中

WorkHasSkills表单的下拉列表，和Designer 中见过的用来显示选择的值的列表 (LOV)。

有四种类型的列表，其中三个是列表项：

- Poplist ：文本项下拉列表。通过靠近列表框的小箭头激活它，使列表在项目附近的窗口上弹出。
- Tlist：在滚动列表上显示所有文本项的列表框，像一个单独的画布，但在表单的单个域内。
- Combo box （组合框)：下拉列表和文本项的组合。可以拉出文本项的列表，但也可以输入新值。
- List of values(LOV)(值的列表)：一种对话框。它显示正文项并提供用于列表中搜索的模型匹配能力。

这个对话框返回所选择的单个列表项，并把它拷贝到另外的块项。在 Form Builder中，这是一个完全独立的对象，不是列表项，而且是用每个项的专门的一组属性把它连接到项上。LOV提供在一个值的长列表中寻找一个项时使用模型匹配的能力。有关利用 LOV向导为项创建LOV的辅导，请参看第4章的内容。

所有这些列表都以某种方法显示文本项的列表，但这些字串不需要项目的值。列表把你在屏幕上看到的标签之类的东西转换为项上的一个实际值，即 Forms存贮在数据库内的值。由于标签和值之间的间接关系，可以用文本显示那些在域中使用其他数据类型的内容。也就是说，列表给了这样的一种方法：在表单中显示格式精美的有意义的正文，但在数据库中仍然存贮没有意义的、难以使用的代码。

例如，可以利用一个比较旧的、以数字形式存储不同选择代码的数据库。数字 1代表红的，2代表绿的， 3代表紫的，等等。列表项给用户显示的是文本字串“ 绿的 ”，而把数字2放入数据库中，在从数据库查询记录时，用“ 绿的 ”去置换 2 ，而不显示“ 绿的 ”这个字串。

1. 有效的列表

列表是用有效的方法显示信息的有力工具，有几种构造列表的方法使它们更实用。

我建议把所显示的字符当作数据库中的值使用。代码一般在非常需要磁盘空间并且程序员只需要读取数字时使用。现在，大多数的程序开发者学会了读正文。使用代码的理由是存在的，但是对大部分情况来说，是因为必须要适应“ 以前拥有 ”的数据库，那些数据库的设计没有使用现代的设计方法，没有对有数十亿实例记录值的非常大的数据库的空间进行优化。如果可以选择的话，保证列表的标签和数据库的值相同，可以使它变简单。

提示 在查询一组记录时，由于某种原因，数据库服务器返回的记录违犯触发器或表单中某些其他条件的要求，那么Oracle Developer会丢掉这个记录，但不告诉你。为什么我在这里提出这个问题呢？这些要求的条件之一是列表项的值要在代码列表中。如果从数据库中查询一个记录，有一个不在列表中，那么表单就不显示这个记录，而且不声不响地把它丢掉。表单假设正在使用的数据库的值和装入应用程序的确认条件是一致的。如果这个假设不成立，可能会引起混乱。应该确保数据库和在应用程序中建立的业务规则的设置的一致性。

列表项中Other Values属性存在的主要原因是因为有一种可能性，即：数据在数据库中有值，但在应用程序列表中却没有值。这种可能性不包括 LOV对话框，因为它不是列表项而完全是单独的一种对象。列表项的这个属性缺省设置是 NULL。如果不把它设置为某个值，那么

Forms Runtime会丢掉那些在列表上没有值的记录。如果把它设置为某个值，那么 Forms Runtime将显示这个记录，但是，显示的是 Other Values属性的值，而不是数据库的值。列表在Forms Runtime不显示对应于这个值的任何字符串。这个值可以是类似于“ Other ”那样的内容。必须把这个值加到列表中，给它一个适当的内容。

注意 不要求你提供Other Values的设置。例如，在检查与项对应的数据库列的约束的情况下没有“ other ”值从数据库中返回，可以设置Other Values为NULL，不会引起任何问题，但是如果不能保证返回的值在列表中，那么最好定义这个属性，不然表单的性能会相当混乱。

组合框为确认问题创建另一个机会。组合框总的思想是允许用户在域中自由输入文本，并且下拉一个提供选择值的列表。但结果是： Developer并不确认对列表所输入的正文。如果这是所希望的特性，很好。如果不是，那么将需要增加确认的编码。或解释违犯约束的数据库错误，或解释用户用某种方式输入的正文。也可以利用某些内部的管理列表项的子程序，把用户输入的项加到列表中去。这要求相当高级的编码。所以，最好是利用组合框作为建议性的列表，而不用去承担编制确认程序的工作负担。

2. 硬代码列表

硬代码列表(Hard-Coded List)是在Form Builder中完全规定的列表。用户用字串和相应的值填满一个对话框，这些值便成为用户的应用程序的一部分。这种方法的优点是简单和快捷，可以非常迅速地创建这些列表，而且在打开应用程序时这个列表是在存贮器中，所以运行得非常好。缺点是如果不重新编制应用程序就不能改变这些列表。对那些简单的不改变的列表可以使用硬代码列表：

创建硬代码列表的步骤：

1) 在块中创建列表项。

2) 显示这个项的Property Palette。

3) 设置项的Type Property为List。

4) 把Width属性设置为想要输入的最长的字串标签的长度。

5) 进入Property Palette的Function部分，把List Style属性设置为列表的三种类型之一：Poplist、TList或Combo Box。

6) 双击列表属性中的Elements。

将看到List Elements 对话框如图6-11所示。

7) 输入标签和相应的值。

8) 如果想要使值为 NULL，则让 Value的单元为空白。

9) 利用DOWN-ARROW(下箭头)键或鼠标移动到列表的新项处，利用 CTRL-K键查看可以在这个对话框中使用的功能键列表。利用 CTRL-<键来删除当前所选择的列表项 (不要忘记按SHIFT键以得到“ < ”而不是“ , ”)。

10) 完成后，单击对话框上的OK。

11) 如果希望能够处理那些不在列表中的值，

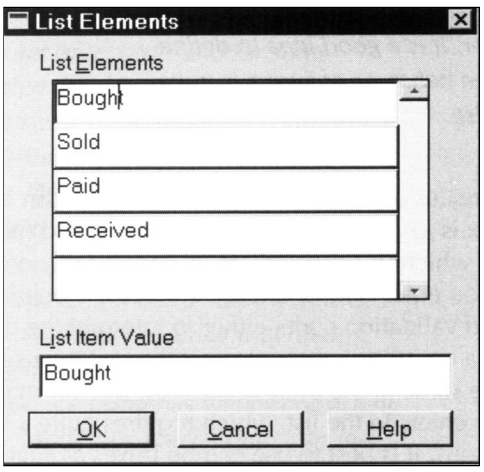

图6-11 List Elements对话框

那么在Other Values属性中输入一个值。

如果运行表单，当显示列表的内容时，将在列表中看到在对话框中输入的标签字串，可以选择列表中的任何项。

注意 有时，在列表中可能得到多余的空白行，这是由于删除了标签和值，又没有用CTRL-<删除列表元素而引起的。可以试试看。

3. 动态列表

动态列表是在运行时填入的列表。有两种方法创建动态列表。最普通的方法是根据数据库的表建立列表。列表由用户创建，是用SELECT语言填写的记录组对象得到列表的标签和值，然后根据记录组填充列表。另一种方法需要利用内部函数 Add_List_Element添加值。后面将较详细讲述第二种方法。首先让我们考虑根据表建立列表。

• 根据表建立列表

首先，如前面部分那样创建列表项。如果需要，在可以用 Other Values属性的List Element对话框内输入标签和值(参看前面的“有效的列表”部分)。

遗憾的是，这个值不能是NULL值。如果对单个硬代码列表的元素使用 NULL值，那么在准备从表创建新的动态列表时，就不能利用 Clear_List清除它，而会得到一个错误，且列表的加载失败。其原因是明显给出在版本 1中出现的问题，即不希望在下拉列表中出现空白行。这个列表不允许删除NULL元素，也就不需要把它加进去产生不希望的空白行。这就意味着，必须为缺省值从数据库中选择一个值。例如，在 Ledger表单的Person列表中使用“ General Store ”这个值作为缺省列表元素。当从表加载这个列表时，这个值将消失，但 Other Values属性中将仍然有这个值。于是这个值出现在从数据库加载的列表以及在硬代码的版本中。

这个要求的后果之一是，不能重新使用目标库中的标准列表 SmartClass。必须采用定义列表元素和Other Values属性的方法来规定列表项和它的数据库值。这也意味着如果选择的值在数据库中被改变了，将必须在硬代码的列表的 Elements属性中改变它。这是很遗憾的，但必须这样做。

一旦建立了 Other Values属性，就可以用通常的方法为表单创建一个记录组对象。在Designer中显示一个对话框，询问是根据值的列表还是用查询创建记录组。缺省设置是用查询，用单选按钮选择查询，并在文本框中输入一个查询。这个查询选择与标签和值对应的数据，并且按它们排序。如果值是一个数字或日期，那么利用 To_Chat函数把这个值转换成字符串。利用ORDER BY子句按照所希望在显示列表中呈现的次序进行排序。

例如，如果希望为Ledger表的Action列创建一个列表，那么应该使用以下的查询：

```
SELECT DISTINCT Action, Action
  FROM Ledger
ORDER BY 1
```

注意 在ORDER BY子句中使用数字1可以按照标签排序而忽略存在有两个数据列使用相同名字的事实。

这个查询利用 Ledger表中的 Action的唯一值产生一个标签和值均相同的记录组。利用GROUP BY子句得到这些唯一的值。 GROUP BY也按照标签将这些值排序，但是无论如何，加ORDER BY子句是一个很好的编码方法，使得在以后需要改变 GROUP BY时不致于混淆。

现在在表单中或在附加库中创建以下的程序。这对整个系统来说都是一个好的候选方法，

因为可以在任何表单中重用这个程序。

```
PROCEDURE Set_Up_List (pList in VARCHAR2, pGroup in VARCHAR2) IS
  vErrFlag NUMBER := 0;
  eListPopulationProblem EXCEPTION;
BEGIN
  vErrFlag := Populate_Group(pGroup);
  IF vErrFlag = 0 THEN
    Clear_List(pList);  -- remove anything already there
    Populate_list(pList, pGroup);
  ELSIF vErrFlag = 1403 THEN  -- no list elements found
    null;  -- do nothing and ignore the problem
  ELSE
    RAISE eListPopulationProblem;
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    Message('Exception:  Could not populate '||pList
            ||' with query in group '||pGroup||'.');
END;
```

注意 如果对同一个列表调用这个程序若干次，只需要清除这个列表。如果不清除这个列表，那么只是把已经不在表单上的值加到列表单上去。还要仔细检查从 Populate_Group返回的代码，因为返回的代码指出SQL查询中的某些问题。还要查看Clear_List属性、List Element属性和Other Value属性之间关系的注释。

现在，需要选择一个触发器来建立这个列表。可以在通过 When-New-Form-Instance触发器打开表单时建立这个列表，或者可以等待到准备显示这个列表时为止，也许在这个列表出现的块的When-New-Block-Instance触发器中。关于这一点，可以有相当大的创造性使得在加载表单时节省一些处理时间。现在在触发器中输入以下代码：

```
Set_Up_List('Action', 'LedgerActionGroup');
```

这段程序用它的内部查询填写LedgerActionGroup，然后，使用这个组来填充这个列表。

注意 也许想重新建立一个列表在应用程序中单独起作用。例如，在一个子表单中要修改出现在列表中的一组值，那么需要再一次运行建立列表的代码，重新设置列表，让它包含这些新的值。遗憾的是，这要求完成所有现有的使列表项出现在列表中的块的事务。做这项工作的一个好方法是在重新建立列表的代码中发送一个 Clear_Block或Clear_Form语句。

6.2.2 单选按钮

通常，如果希望用户在一系列可能的值中作选择，那么应该提供一个供选择的列表。单选按钮提供了这样一种列表的方法。单选按钮组是一些无线电按钮的集合。无线电按钮是带有圆圈的小按钮 (准确的格式取决于图形用户接口平台)，当用户单击其中一个按钮时， Oracle Developer离开当前的按钮而选择被选中的按钮。因此，每次只能选择一个按钮。无线电按钮这个名字出自收音机中选台按钮的特性，它们是按下一个按钮则弹起另一个按钮 (或现代无线电的电子按钮)。在图6-12中，在每一种情况下只有一个选择是起作用的。

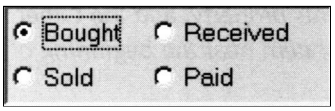

图6-12 单选按钮列表

当有相对小数目的相互不相容的选择时，通常这些选择是两个或三个不随时间变化的选择，使用单选按钮。在上例情况中，有四种可能的动作类型是非常稳定的。

注意 单选按钮组结合应用程序的结构来构造对数据库的值的约束。因此，动态列表通常是更可取的接口。因为用户可以扩展它而不改变应用程序的代码。如果列表是静态的而不是动态的，或者，如果值实际上不随时间变化，那么单选按钮组在显示和容易使用等方面有某些优点。

单选按钮组是对应于数据库表中的列的块项，单选按钮对应于列的可能值。单选按钮在单个单选按钮组的项目下聚集在一起。当用户在按钮上单击时，这个按钮对象的 Value属性就变成了这个单选按钮组的值。

这个逻辑使这些对象所具有的一组属性互相有牵连，即影响所有按钮的任何属性都是按钮组的属性。比如缺省值与数据库有关的属性、导航的属性就是如此。只影响单个按钮的属性是按钮的属性，比如按钮值、按钮的标签和按钮的物理位置等。要在 Oracle Developer中设置那些为选择新记录而显示的初始化按钮，可以把单选按钮组的 Default Value属性设置为按钮的值(比如在这个例子中的“ Bought ”)。

注意 在块中创建记录之前，这个块的任何项的Default Value属性不起作用。即使是可以看见这个块，在导航到它上面以前，实际上也并没有在这个块上创建记录。在实际在块上创建新记录以前，缺省值并不出现。

为了要建立单独的按钮，以通常的方法利用 Create工具创建，然后改变Label属性和Radio Button Value属性，Label是Oracle Developer作为提示而显示的文本， Radio Button Value是当用户单击按钮时， Oracle Developer赋予单选按钮的实际值。

至于列表，如果用一个与任何单选按钮值不对应的值从数据库中取回一行，那么就会发生问题， Oracle Developer将悄悄地丢掉这个记录。单选按钮组的 Other Value属性工作起来就像有同样名字的列表项的属性一样，它让你为了显示的目的把和按钮不对应的值变换为一个按钮值。

根据用户界面设计的类型，通常用某种可视化方法区分单选按钮组比较好。前面的例子利用样板图形的长方形包围着四个单选按钮，把它们和画布上其他的块项分开。在这种情况下，长方形的Beveled属性设置为Lowered ，可以显现出三维的效果。

6.2.3 计算项

创建表单的一个共同任务是创建表示某个计算值的项。例如，可以把几个数据库的项组合成单独的连接起来的字符串，而这些项在基础表中不是一一对应的。在 Ledger表单中，可以用Quantity(数量)乘Rat(价格)的计算域来代替数据库的 Amount (总计)域。另外一个普通的例子是当创建Total域时，在另一个块中计算记录总数。

在Oracle Developer以前的版本中，通过利用各种触发器来计算这些值，并通过把值赋予项来显示它们从而达到这些效果。通常是利用 When-Validate-Item和Post-Query触发器达到这些效果。例如，要计算 Amount (总数)，就把Base Table Item属性设置为False，通过改变项的类型使项成为显示项，然后对 Quantity (数量) 和Rate (价格) 创建 When-Validate-Item触发器，每一个触发器有这样的代码：

```
:Ledger.Amount := :Ledger.Quantity * :Ledger.Rate;
```

这个处理插入和更新了两个数据库的项。还应该用同样的代码在 Ledger块创建一个Post-Query触发器来处理从数据库返回的数据。

利用Calculation属性组，可以非常容易地创建计算域。通过设置项属性 Database Item为NO，创建一个不在基础表中的显示项。不创建触发器，而是设置 Calculation组的属性。把Calculation Mode (计算模式)设置为Formula (公式)产生计算特性，然后输入公式，就像前面见到的输入 PL/SOL代码那样：

```
:Ledger.Quantity * :Ledger.Rate
```

每当记录中发生某种变化时， Oracle Developer再计算这个域并显示新的值。

在Oracle Developer中进行总计统计是很容易的。在想要总计的块中创建总计项，必须把这个块的Advanced Database 组的Precompute Summaries (预先计算累加) 属性设置为Yes，这就告诉Oracle Developer，在做查询以前先计算总和。因为通常这个块会显示多个记录，然而总计时并不想对每个记录显示一个总计，所以把要显示的总计项的属性数目设置为 1(通常它为零，意味着项的数目和在块中显示的记录数目相同)。

注意 把Query All Records in Records group(查询记录组中所有的记录)设置为Yes，而不是把Precompute Summaries (预计计算总和)设置为Yes，可以达到同样的效果。与其分别计算总和，不如Oracle Developer查询所有的记录而不是在需要时提取它们。这就可以从存贮器中的记录来计算总和。对于有大的结果集的块，这可能会引起一些存贮器问题。预先计算也会慢下来，但不像Query All Records(查询所有的记录)那么严重。用域的性能测试进行实验，决定使用哪种方法。

现在，设置项的四种属性，Calculation Mode (计算模式)设置为Summary (求和)，决定了项的求和计算。把Summary Function设置为Sum，告诉Oracle Developer要添加记录的值。可以利用任何标准的总计功能，比如 Count、Average、Min、Max、Stddev或Variance。Summarized Block和Summarized Item下拉列表分别显示所有的块和所选择的块中的所有的项。例如，要创建一个Total (总计)项来总计Ledger块的Amount (数量)项，就把Summary Function (总计功能)设置为 Sum (求和)，对Ledger块求和 (Summarized Block)，而对 Amount (数目)项求和 (Summarized Hem)。Oracle Developer自动地在画布的独立的域中显示对块的总计，并处理对求和项所有的计算的变化，包括清除或除去记录。

6.2.4 日期项和时间项及格式掩码

和计算机打交道的大多数人现在必定认识到了“ 2000年 ”问题。当时钟在1999年的最后一天的夜里敲响时，所有两位数字的年域将变为“ 00 ”，而且根据媒体的说法，这个结果将是混沌的(Chaos)。这本书现在还是“ 温顺的 ” (Compliant)！在2000年1月1日以后你仍然可以读。

这个问题是讲得过分了，但是的确要注意日期域和时间域。为了确保没有 2000年的问题，应该把年号规定和显示为四位数字值。对于 Oracle Developer的日期项，用某种“ YYYY . ”似的变化转换成格式掩码。日期格式掩码可以处理成世界各地普通的日期格式，所以不必要书写大量的触发器或程序来格式化日期。

大多数日期和时间问题发生在进入另一个不完整的日期和时间场所时。处理时间的问题

可能相当严重，除非你记住了有关Oracle Developer和Oracle时间的几个基本事实。

生活中最主要的事实是，时间和日期是Siamese孪生子，它们不能够摆脱地连在一起。对Oracle Developer的项和域，PL/SOL变量和Oracle的列，日期类型DATE是共同的，并包括有日期和时间两部分。大多数问题发生在企图单独处理日期和时间的时候。例如，有可能想用单独的可输入日期和时间的域呈现给用户，让他们设置或查看时间。如果不在 Pre-Insert、Pre-Update、Pre-Delete和Post-Query触发器中进行PL/SOL编码，利用To_Char和To_Date转换功能组合和分开日期元素，就不能这么做。为了避免不必要的压力，在 Oracle Developer中，日期项使用格式掩码，例如，“MM/DD/YYYY”“HH：MI”，而不要为了“可用性”而将它们分开。

注意 Oracle Developer Forms为了和前面版本的表单兼容，也有DATETIME和TIME数据类型。不要使用这些数据类型，因为转换为Oracle DATE可能产生意外的结果。

使用Oracle Developer的格式掩码并不影响实际日期和时间的正确性，它会总是保持相同。在屏幕上看到的并不是进入数据库得到的，因此，对于这个问题没有任何事情要做。例如，在一个表单的日期项中没有看到时间，这并不意味着没有时间或时间是 00：00，虽然这可能是由缺省得到的结果。可以用PL/SOL代码设置特定的时间：

```
:Ledger.ActionDate := To_Date(To_Char(:Ledger.ActionDate,
                                      'MM/DD/YYYY')||' 12:00',
                              'MM/DD/YYYY HH:MI');
```

这个代码指定，通过Ledger.ActionDate插入的所有项的日期的缺省值为中午 12点，而不是半夜的12点。

表6-3显示了和年份有关的Oracle Developer格式掩码，可以组合成日期-时间格式掩码。

表6-4显示了可以在Oracle Developer格式掩码中使用的和时间有关的各种元素。

表6-3 与年份有关的格式掩码元素

元 素	说 明
A.D.	对B.C.日期显示“B.C.”，对A.D.日期显示“A.D.”
AD.	对B.C.日期显示“BC”，对A.D.日期显示“AD”。
B.C	对B.C.日期显示“B.C.”，对A.D.日期显示“A.D.”
BC	对B.C.日期显示“BC”，对A.D.日期显示“A.D.”
RR	两位数字的年份，世纪为缺省最好使用四位数字而不使用两位数字
RRRR	四位数字的年份，与YYYY相同
SYYYY	带有数字的年份和一个负号的B.C.日期
Y	一位数字的年份(不推荐)
Y，YYY	四位数字和一个千位分隔逗号的年份
YY	两位数字的年份(不推荐)
YYY	三位数字的年份(不推荐)
YYYY	四位数字的年份(推荐)

表6-4 与小时有关的格式掩码元素

元 素	说 明
A.M.	对ante-meridian时间为“A.M.”，对post-meridian时间为“PM.”
AM	对ante-meridian时间为“AM”，对post-meridian时间为“PM”
HH	一天中的小时用12小时格式(1-12)
HH12	一天中的小时用12小时格式(1-12)

(续)

元 素	说 明
HH24	一天中的小时用24小时格式(0-23)
MI	小时的分钟(0~59)
P.M.	对ante-meridian时间为“ A.M. ”，对Post-meridian时间为“ P.M. ”
PM	对ante-meridian时间为“ AM ”，对Post-meridian时间为“ PM ”
SS	分钟的秒(0~59)。
SSSSS	自午夜后的秒数(0~86399)

表6-5展示了和日期及月份有关的元素，可以组合格式掩码。

表6-5 与日子和月份有关的格式表征码元素

元 素	说 明
D	一周的数字日期(1~7，1为星期天，除非当前的NLS语言把它定义为别的东西，比如星期一)
DAY	固定长度，九个字符的日子名(“ SUNDAY ”、“ MONDAY ”等等)
DD	一个月中的日期(1~31)
DDD	一年中的日期(1 366)
DY	三个字母的日期名(SUN、MON、TUE等等)
J	自公元前4712年1月1日以来的天数(“ Julian ”)只在专业的排序应用程序中使用，Oracle公司反对使用这种格式，所以不要使用它
MM	数字格式的月份(1~12)
MON	三个字母的月份名(JAN、FEB，等等)
MONTH	固定长度，九个字符的月份名(“ JANUARY ”、“ FEBRURY ”等等)

可以把大多数标点符号放在格式掩码的位置，而 Oracle Developer在显示字符串中重新创建它。可以把在双引号内的任何文本串嵌入掩码来显示它。例如，掩码“ MONTH ”“ DD ”，“ YYYY ”显示字符串“ APRIL 18，1999 ”。

注意固定长度的MONTH(月份)，表征码元素使月份名字中产生了额外的空格，可以利用在字串行前加 FM，消除前面的空格“ FMMONTH ”“ DD ”，“ YYYY ”，则字符串显示为：“ APRIL 18，1999 ”。可以在格式元素中规定字体，根据自己的喜好使用“ MONTH ”或“ Month ”。

定界符是在日期中分开年、月、日的标点符号标签，通常是破折号或斜线，例如 18-APR-1999或4/18/1999。当把它们作为格式掩码的一部分时，用户可能会输入不同的标点符号 (例如4.18.1999)，掩码则会适当地转换它们。为了强制使用特定的标点符号和强制用户提供日期或时间中的数字和字符，可以使用 FX前缀。例如格式掩码FXDD-MON-YYYY HH：MM就是强制用户用标准的格式输入：01-JAN-1999 12：00，包括前面的0和破折号。

可以同时使用FX和FM前缀，使FX只用于限定符。例如， FMFXDD-MON-YYYY将允许用户输入1-JAN-1999，而不能输入1.JAN.1999。

6.2.5 过程的封装

EXECUTE特权给出一个有趣的方法可以给对象授予广泛的特权：程序封装。任意接近用户、角色和特权的方法的缺限是基于服务器的，这些特权的应用忽略了用来会话的机构。就是说，当通过应用程序注册时，可以得到修改数据的特权，就像通过 SQL*PLus或另外一个程序注册时得到的特权一样。如果某些或全部的确认进程只以 Oracle Developer的代码存在，那

么由于缺少确认，在通过其他的程序时可能会引起数据库问题。如果依赖于应用程序层的安全性，那么问题可能更严重，因为服务器并不实施这种安全性测量。

如果真想要在高度安全环境下限制用户的能力，可以把所允许的特性封装在存贮的程序中，然后在程序中给角色或用户授予 EXECUTE(执行)特权。用户并不需要基础对象的特权来执行这个程序，只需要创建这个程序本身的那些特权。这样，通过运行程序的应用程序注册的用户，就可以做他需要做的事情。但是通过 SQL*PLus注册的用户不能用 SQL直接做这些事情(当然，不是运行带有隐含限定的同一个程序)。

注意 这个讨论也适用于触发器以及程序。

Oracle Developer有能力使用存贮在作为数据源的数据块中的程序。首先，考虑数据块和作为数据库对象类的基础表，需要作什么能够与这些作为类的对象相联系呢？

- 选择(Select)：选择一些数目的数据库对象并作为数据块的记录显示它们。
- 插入(Insert)：用数据块的记录插入数据库对象。
- 更新(Update)：用数据块中被修改的记录去改变数据库对象。
- 删除(Delete)：从在数据块中选择的记录中删除数据库对象。
- 锁定(Lock)：锁定与块记录对应的数据库对象。

这五件最基本的事情就是 Oracle Developer在数据库中做的大部分的事情。如果为每一件事情都建立一个存贮的程序，然后把它们和数据块连接，那么就可以享受到对 Oracle Developer应用程序的所有程序封装的优点。任何人用这些程序上的 EXECUTE(执行)特权来运行应用程序，都将会看到应用程序的运行就像是使用标准的数据块一样。而没有这些特权的任何人将不能在应用程序中做任何事情。还有，如果把广泛的表查找作为数据库确认的顺序的一部分的话，那么把所有这些移到服务器作为存贮程序的一部分，可以改善性能。

要用这种方法使用程序封装，必须要为作为 Oracle Developer应用程序的数据块基础的每一个表，建立一个专门的程序包。例如，对 Skill表，可以定义这个程序包，将它连接到拥有 Skill的用户上：

```
CREATE OR REPLACE PACKAGE SkillPackage AS
    -- A record type with all the columns in the Skill table
  TYPE tSkill IS RECORD (
    rSkill Skill.Skill%TYPE,
    rDescription Skill.Description%TYPE);
  -- A record type with the primary key column in the Skill table
TYPE tSkillKey IS RECORD (rSkill Skill.Skill%TYPE);
  -- A ref cursor for a set of Skill rows
TYPE tSkillCursor IS REF CURSOR RETURN tSkill;
  -- A table type for a table of Skills
TYPE tSkillTable IS TABLE OF tSkill INDEX BY BINARY_INTEGER;
  -- A table type for a table of primary keys;
TYPE tSkillKeyTable IS TABLE OF tSkillKey INDEX BY BINARY_INTEGER;
  -- An exception for the row-locked error (Oracle error -54)
eRowLocked EXCEPTION;
PRAGMA EXCEPTION_INIT(eRowLocked, -54);
  -- Two query procedures, one by cursor and one by table
PROCEDURE SelectCursor (pSkillCursor IN OUT tSkillCursor,
                        pDescription IN VARCHAR2);
PROCEDURE SelectRows (pSkillTable IN OUT tSkillTable,
                      pDescription IN VARCHAR2);
```

```
    -- Data manipulation procedures
  PROCEDURE InsertRows (pSkillTable IN tSkillTable);
  PROCEDURE UpdateRows (pSkillTable IN tSkillTable);
  PROCEDURE DeleteRows (pKeyTable IN tSkillKeyTable);
  PROCEDURE LockRows (pKeyTable IN tSkillKeyTable);
END SkillPackage;
```

这个程序包是抽象数据类型的一个例子——由一组特性伴随的一组数据结构，这些特性对这组数据结构进行操作。它类似面向对象的类，但它没有继承性。

注意 要了解PL/SQL的意思，需要参考与PL/SQL有关的第11章和第12章。这些章节讨论在程序包技术条件中看到的各种元素。第 12章还包含关于结构的细节和与 Oracle Developer一起的PL/SQL程序包的使用。

程序体可以包含所需要的任何的 PL/SQL代码，虽然它们通常必须在 Oracle Developer期望的程序方向上运行。

```
CREATE OR REPLACE PACKAGE BODY SkillPackage AS
    --
    -- Query by reference cursor
  PROCEDURE SelectCursor (pSkillCursor IN OUT tSkillCursor,
                          pDescription IN VARCHAR2) IS
    vDescription VARCHAR2(80) := pDescription;
BEGIN
  IF vDescription IS NULL THEN  -- select all skills for NULL
    vDescription := '%';
  END IF;
  OPEN pSkillCursor FOR
    SELECT Skill, Description
      FROM Skill
     WHERE Description LIKE vDescription
     ORDER BY Skill;
END SelectCursor;
  --
  -- Query into table of records
PROCEDURE SelectRows (pSkillTable IN OUT tSkillTable,
                      pDescription IN VARCHAR2) IS
  vIndex NUMBER := 1;
  vDescription VARCHAR2(80) := Nvl(pDescription, '%');
  CURSOR cSkill IS
    SELECT Skill, Description
      FROM Skill
     WHERE Description LIKE pDescription
     ORDER BY Skill;
BEGIN
  OPEN cSkill;
  LOOP
    FETCH cSkill INTO pSkillTable(vIndex).rSkill,
                      pSkillTable(vIndex).rDescription;
    EXIT WHEN cSkill%NOTFOUND;
    vIndex := vIndex + 1;
  END LOOP;
END SelectRows;
  --
  -- Insert rows from a record table into database table
PROCEDURE InsertRows (pSkillTable IN tSkillTable) IS
  vIndex NUMBER := 1;
```

```
  vCount NUMBER := pSkillTable.Count;
BEGIN
  FOR vIndex IN 1..vCount LOOP
    INSERT INTO Skill (Skill, Description)
    VALUES (pSkillTable(vIndex).rSkill,
            pSkillTable(vIndex).rDescription);
  END LOOP;
END InsertRows;
  --
  -- Update rows from a record table into database table
PROCEDURE UpdateRows (pSkillTable IN tSkillTable) IS
  vIndex NUMBER;
  vCount NUMBER := pSkillTable.Count;
BEGIN
    FOR vIndex IN 1..vCount LOOP
      UPDATE Skill
         SET Description = pSkillTable(vIndex).rDescription
       WHERE Skill = pSkillTable(vIndex).rSkill;
    END LOOP;
  END UpdateRows;
    --
    -- Delete rows from a table of keys
  PROCEDURE DeleteRows (pKeyTable IN tSkillKeyTable) IS
    vIndex NUMBER := 1;
    vCount NUMBER := pKeyTable.Count;
  BEGIN
    FOR vIndex IN 1..vCount LOOP
      DELETE FROM Skill
            WHERE Skill = pKeyTable(vIndex).rSkill;
    END LOOP;
  END DeleteRows;
    --
    -- Lock rows from a table of keys
  PROCEDURE LockRows (pKeyTable IN tSkillKeyTable) IS
    vIndex NUMBER := 1;
    vCount NUMBER := pKeyTable.Count;
    vDummy VARCHAR2(1);
  BEGIN
    FOR vIndex IN 1..vCount LOOP
      SELECT 'X'
        INTO vDummy
        FROM Skill
       WHERE Skill = pKeyTable(vIndex).rSkill
      FOR UPDATE NOWAIT;
    END LOOP;
  END LockRows;
END SkillPackage;
```

注意 这种逻辑的有效使用，使得不能使用标准的Oracle Developer的query-by-example机制。你必须通过把变量传递到选择程序，自己建立查询。在刚给出的例子中，两个选择程序进行了说明，说明哪一个是可以查询的唯一列。程序体中的WHERE子句假设你想在这个值的输入中做一个LIKE表达式。你可以做得比这复杂得多，但这种技术超出了这本书的范围。

1) 编译作为程序包拥有者注册的程序包 (例如Talbot)。

2) 给需要调用这个程序包的程序的每一个角色授予在这个程序包上的 EXECUTE特权(例如HRSkillRole)。

```
GRANT EXECUTE ON SkillPackage TO HRSkillRole;
```

3) 作为DBA用户或作为有CREATE PUBLIC SYNONYM特权的用户注册，并为这个程序创建一个公共的同义词：

```
CREATE PUBLIC SYNONYM SkillPackage FOR Talbot.SkillPackage;
```

注意 必须为能够通过应用程序访问包装程序的用户授予EXECUTE特权和创建公共的同义词，除非你把拥有者的用户名编码作为程序名的一部分，例如像 TALBOT.SKILL PACKAGE、SELECURSOR。通常，这并不被认为是好的编码方法，因为它把应用程序编码和特定用户连接起来了。

现在准备创建数据块。

4) 从Tools | Data Block Wizard中启动Data Block Wizard(见图6-13)。

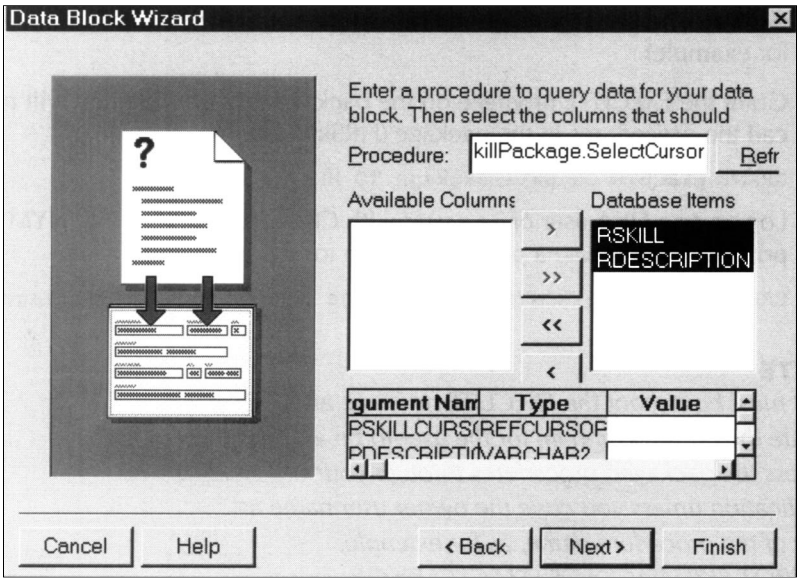

图6-13 Data Block Wizard对话框

5) 在第一个对话框中，选择Stored Procedure单选按钮，而不要选择Table按钮。这就告诉Wizard是程序的信息，而不是标准的表和列的信息。

6) 单击Next按钮，开始定义块。

7) 下一个对话框询问查询程序，输入程序包名和游标程序名。例如“ SKILLPACKAGE.SELECTCURSOR ”，遗憾的是这个版本不让你为了这个信息而浏览数据词典。

8) 在Refresh(刷新)按钮上单击，显示类型列的名字 (RSKILL和RDESCRIPTION)。

9) 用>或>>按钮来调整数据块所需要的列。注意，位于对话框底部的变量列表显示的是程序的变量。不要改变这些变量。

注意 确信你把程序包的名字加到了程序名字上，否则Data Block Wizard将找不到这个程序，在插图的对话框中程序包的名字滚动到了项域的左边。

10) 单击Next，移到下一个对话框，提示插入程序。

11) 输入程序包名字和插入的程序名：

“ SKILLPACKAGE.INSERTROWS ”。

12) 在Refresh按钮上再次单击，显示类型列和程序变量，虽然，在这里没有任何它们可以做的事情(见图6-14)。

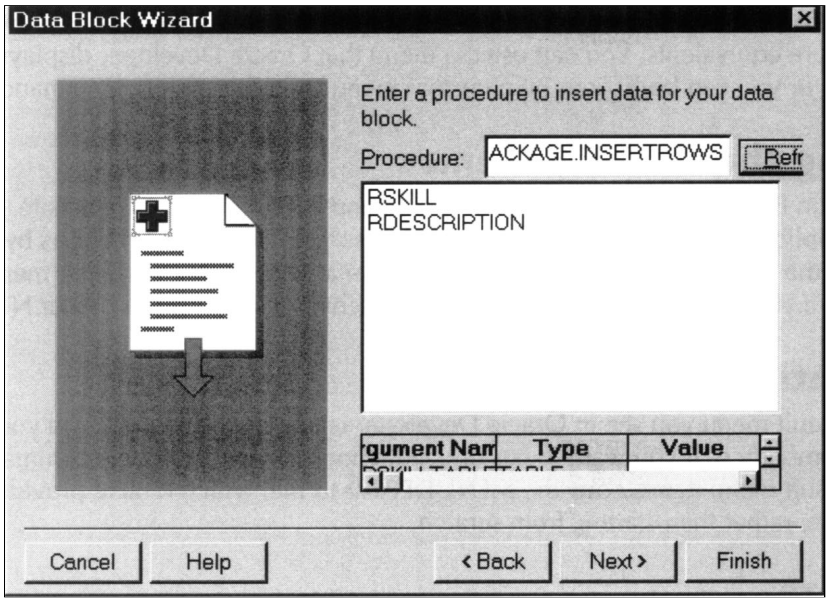

图6-14 单击Refresh，显示类型列和程序变量

13) 在下面的三个对话框中重复更新、删除和锁定这三个步骤 (分别是UPDATEROWS，DELETEROWS和LOCKROWS)。如果数据块结构使得有可能的话，会看到出现标准的主 -从关系定义对话框。这个窗口让你在新的块和先前定义的其他块之间定义任何主 -从关系。

14) 像往常一样，现在可以前进到Layout Wizard中。

利用程序封装，可以有限制地访问封装在 PL/SQL程序中的明确定义的动作特性，不容许广泛的SQL访问对象。

6.3 菜单

Form Builder的菜单系统让你执行运行时系统的不同的功能，比如 Next_Record，Enter_Query或Previous_Item。这些命令的大多数直接与按键对应，而且大多数有等效的内部程序，可以利用Oracle Developer显示缺省的菜单，或者可以建立自己定制的菜单以增加或改变命令。

6.3.1 利用缺省菜单

Form Builder系统有一个内部的命令菜单，在执行表单应用程序时，如果不定义菜单，那么系统显示缺省菜单，如图 6-15中的菜单。如果是在MDI环境下，比如在Windows 95或NT下运行，表单还显示缺省的菜单工具条。

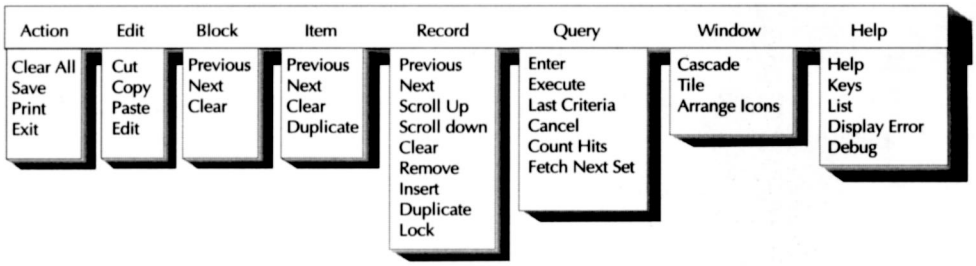

图6-15 Forms Runtime缺省的菜单结构

6.3.2 创建定制菜单

在Oracle Developer中看见的缺省菜单是在代码中生成的，而且不能修改它。如果想加一些命令到缺省的菜单项上来改制一个菜单，那么就拷贝 MENUDEF.MMB文件。Oracle Developer提供它作为样板，而不是从暂存区开始。

注意 提供自己菜单模块的一种结果是菜单工具条消失。像前面的“创建工具条”部分讨论的那样，这个工具条提供一些标准的操作，比如Enter Query和Next Record等。你不能通过定制的菜单来显示这个缺省的工具条。必须利用菜单的 Horizontal Menu Toolbar属性或Vertical Menu Toolbar属性的Visible值来建立自己的工具条。

1. 创建菜单、项和子菜单

在菜单模块中有两种对象：

- 菜单：主菜单、个别的菜单(比如Action或Record)和子菜单(从菜单项弹出的分级菜单)。
- 菜单项：在每个个别菜单中或子菜单中的项，对应于 PL/SQL代码或命令文本，个别菜单或子菜单拥有菜单上的项。

每个菜单和菜单项都有一个名字，用这个名字可以在菜单模块中唯一地识别它。另外，菜单项有一个标签，在显示菜单时，字符串出现在菜单上。

在创建菜单结构时，应该遵循五条原则。

第一，任何个别菜单或子菜单不应有多于九个的菜单项。如果菜单比较短，则比较容易使用。如果需要，可以利用子菜单把几个菜单项收集入组。

第二，如果要使用子菜单，努力争取只在主菜单的个别菜单的下一层上使用它们。嵌套子菜单的层次越深，找命令就越困难。例如在Query菜单中用Report子菜单，它有两个菜单项，Ledger Summary和Mailing Labels。这就意味着系统中菜单级不大于三，用户可以容易地找到这些菜单命令(见图6-16)。

第三，不要考虑根据应用程序当前状态变化的动态菜单。这样不会将用户弄得莫名其妙。菜单应该是稳定和不变化的。

第四，为了帮助用户避免错误，在应用程序当前状态下把不使用的菜单项置为禁止状态(变成灰色)。例如，如果显示一个特殊的无模式窗口，这个窗口使用一些命令，但不使用其他的东西，那么禁止不用于它的或不适用整个应用程序的菜单项。可以采用 Disable_Item内部子程序来做这项工作，这个子程序要给定菜单名和项名：

```
Disable_Item('Block', 'Previous');
```

这个例子禁止Block | Previous菜单项。要允许这个项，使用 Enable_Item，采用同样的参

数变量。

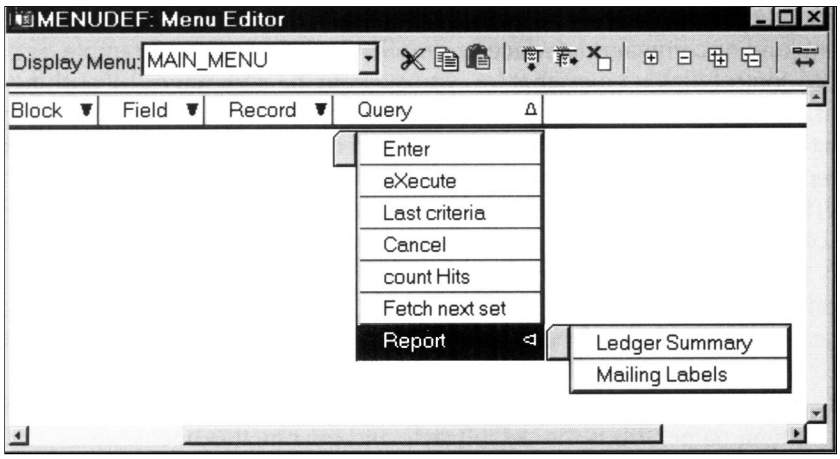

图6-16 嵌套子菜单

注意 通过使用Set_Menu_Item_Property内部子程序或通过分别在 Enable_Item和Disable_Item内部子程序中使用菜单名字MAIN_MENU，可以对主菜单上的项进行允许和禁止。

如果想显示菜单工具条，那么菜单项的次序就确定了工具条上图标的次序。因此，必须平衡菜单的透视次序和工具条的透视次序。另一方面，可以显示你自己的工具条，把它作为应用程序中的单独的画布。

如果有某些理由使你不能坚持这些原则，那么应该考虑用不同种类的命令接口来简化菜单。例如，建立一组控制对话框，通过按钮来执行命令，或建立一些对话框，采用输入选择作为选项而不是提供每一条命令作为菜单上的单个命令。不用列出作为菜单项的所有报表，可以有单个的菜单项，如Query | Run Report，有一个对话框用一系列单选按钮或复选框来选择运行报表。甚至可以从表示命令结构的图形显示中得到好处。发挥创造性，使你的应用程序更容易使用。

有三种类型的菜单，本书只详细讨论其中的一种：

- 下拉菜单：采用在单独的菜单名字上单击方式的菜单，这是缺省的类型，本书使用这种菜单。
- 全屏幕(Full-screen)菜单：占据了字符模式的整个屏幕的菜单，通过选择项与显示新屏幕来定位命令。这些菜单只在字符模式和块模式的应用程序中工作。
- 菜单条类型(Bar-style)菜单：这是这样一种菜单，当在它的上面单击时，用单独的菜单项代替主菜单。这种菜单只占据菜单条的空间，而且决不扩展到应用程序的内容区域，这种类型对于字符模式的应用程序比GUI更好。

如果想知道更多的有关全屏幕或工具条类型菜单的信息，请参看联机表单资料。

在菜单可以使用以前，必须把菜单文件编译为 MMX文件。在试图运行使用这个文件的任何表单之前，利用Form Designer(表单设计程序)中的File | Administration | Compile File来产生MMX文件。在进行某些修改以后，还必须重新编译这个 MMX文件，因为在运行表单时，不会自动做这些事情。

注意 对开发者来说，在运行表单之前忘记编译菜单是非常普遍的。每当在菜单上工作时，应该养成习惯，利用与File | Administration | Compile或Compile File对应的按钮，比如Windows中的CTRL-T自动产生表单。

2. 编辑菜单

Menu Editor让你直接看到正在建立的菜单系统。必须了解它的几个特性，才能充分编辑菜单和菜单项。在每个菜单左上角的小标签是选择标签。用在这些标签上单击的方式来选择菜单对象，也可以通过单击标签并把它拖到新的位置来移动菜单。 CTRL-dragging是拷贝菜单而不是移动它。在任何菜单项上单击即选择了这个项。像在表单中那样通过选择 Tools | property palette项或通过在Object Navigator上的对象的图标上双击，就可以看到在对象的Property Palette上的任何对象的属性。在 Menu Editor的菜单上双击，则可以编辑它的标签。可以用Delete 工具删除一个被选择的对象。

编辑器工具条上的两个专门的工具，可以用它添加菜单项和子菜单。用 Create Down工具可以在所选择的菜单项下加一个菜单项到菜单中。用 Create Right工具可以在选择的菜单项右边创建一个子菜单。这些工具对应于 Menu | Create Down和Menu | Create Right菜单项。

可以用通常的方法改变Object Navigator或Property Palette中的菜单项的名字。改变项的名字并不改变项的标签，反之亦然。如果改变名字，在编辑器看来，菜单是同样的，因为并没有改变标签。在项上双击或在 Property Palette中用Label属性来改变标签。

注意 界面设计原则经常说，要在标签后面放一个省略符号(三个圆点)：如“Options... ”。像大多数这些原则指出的那样去做，但是许多用户界面的设计却失败了。只有当显示一个对话框要求更多的信息执行命令时才应该这么做。例如对子菜单或对通过菜单项调用文档窗口就不用这么做。

3. 编制菜单项代码

用菜单项的Command Type(命令类型)属性来指定那些通过选择项目来执行的命令的种类。参看表6-6。

表6-6 菜单项命令类型属性值

值	说 明
Null	不做任何事情，可以使用这个值创建分隔符项，或使一个项为空，直到准备要它某些事情为止
Menu	调用子菜单，命令文本是子菜单的名字
PL/SQL	执行命令文本中的一块PL/SQL代码，这个代码不能直接引用表单模块的变量，而必须使用Name-In和Copy间接引用它们
Plus、Current Forms、Macro	不要使用这些兼容的选择，使用Host或Run_Product来PL/SQL内部程序运行SQL*Plus、表单，或操作系统命令

利用在圆点分开语法menu_name.item_name中的菜单名和项名来引用 PL/SQL中的菜单项，可以利用这个语法把命令连接到菜单项、初始化项、改变启动代码，或使能 /禁止菜单项，利用语法main_menu.item_mane来引用主菜单上的项，比如Block或Query。

要创建编码，在菜单项的属性设置板的 Command Text属性上双击，并把编码输入到PL/SQL编辑器中，编辑它然后关闭编辑器。

注意 如果要执行一个简单、标准的与某种按钮对应的命令，那么使用 DO_Key内部

程序：

```
Do_Key('COMMIT_FORM');
```

这个PL/SQL代码执行Save项，利用DO_Key联合命令键和菜单项，使你不必重复工作。

如果要使用Oracle Developer Server，那么可以利用内部子程序Web.show_Document来运行报表并在 Web浏览器中显示它。如果使用一个大的应用程序，那么可以用内部程序Run_Product运行Oracle产品。例如Ledger Summary菜单项运行Ledger Summary报表，这个菜单项的代码看起来像这样：

```
Run_Product(REPORTS, 'ledger', SYNCHRONOUS, RUNTIME, FILESYSTEM,
            NULL, NULL);
```

这个命令用文件系统中的 Ledger.RDF文件运行 Runform程序。它立即运行这个报表(RUNTIME)而不是以批处理方式，而且它同步运行这个报表，这意味着必须等待报表完成打印后，才能继续应用程序的工作，这里没有参数也没有图形显示。

注意 结尾的两个NULL值是被要求的。它们表示这个特殊的Run_Product调用没有使用变量。没有这两个NULL值，PL/SQL将不认识这个调用，因为变量的数目无效。

可以利用Run_Report_Object内部子程序更直接地运行报表。首先，必须在表单中定义一个报表对象，在你的表单下找出报表的标题，用通常的方法单击 Create工具。出现New Report对话框，它让你从Scratch或通过从文件中读入一个报表模块来创建一个新报表。第 7章详细介绍利用Report Editor在表单中创建报表。一旦有了报表对象，就可以利用 Property Palette给它取一个名字并设置Dev/2000/Integration属性组的Execution Mode属性和Communication Mode属性(例如同步和运行时)。还可以为报表设置系统参数，比如 Destination Type、Name和Format。

一旦建立了报表对象，重新产生表单，现在可以在调用 Run_Report_Object内部程序中通过名字来引用报表对象了。例如如果利用 Ledger.RDF报表文件创建了Ledger报表对象，那么把这个代码放在Query/Ledger Summary菜单项的PL/SQL块中 ：

```
DECLARE
  vReturn VARCHAR2(100);
BEGIN
  vReturn := Run_Report_Object('Ledger');
  IF vReturn IS NOT NULL THEN
    Message('Ledger Summary Report: '||vReturn);
    Synchronize;
  END IF;
END;
```

要引用附加到菜单上的表单的项值，可使用 Name_In内部函数。例如，要使WORKER块的NAME项的值与一个特定的名字比较，应该使用以下代码：

```
IF Name_In('WORKER.NAME') = 'Adah Talbot' THEN
```

类似地，用Copy内部程序指定一个值：

```
Copy('Adah Talbot', 'WORKER.NAME');
```

如果PL/SQL的复杂程度适中，那么最好把这个代码放入和菜单模块结合的程序库的单独的程序单元中。例如刚才给出的运行报表的代码可以是一个要求报表的名字的程序，如以下所示：

```
PROCEDURE Run_Form_Report(pReport IN VARCHAR2) IS
  vReturn VARCHAR2(100) := Run_Report_Object(pReport);
BEGIN
  IF vReturn IS NOT NULL THEN
    Message(pReport||' Report: '||vReturn);
    Synchronize;
  END IF;
END;
```

这不仅消除了单独维护程序的复杂性，而且使报表的错误报告机制标准化。

这是菜单模块的 Startup Code(启动代码)属性。在启动加载菜单的表单时， Oracle Developer执行在这个属性中的任何代码。这个代码可以在某些少有的情况下，即在显示表单以前做某些与菜单有关的事情的情况下，完成各种建立任务。在可能的情况下应该依赖缺省的属性，但有时，必须根据平台或某些类似的事情给不同的菜单项赋能。

4. 利用特殊的菜单项

Menu Item Type属性用来改变要显示的菜单项的种类。有五种类型的菜单项，如表 6-7所示。

表6-7 Menu Item Type属性值

值	说 明
Plain	标签缺省类型
Check	显示标签检查记号；设置项目状态为TRUE或FALSE,可通过具有CHECKED选项的GET_MENU_ITEM_PROPERTY内部函数对其进行测试并通过 SET_MENU_ITEM_PROPERTY进行设置
Radio	制作项目为单选按钮组中项目集之一，从此选择一个为 TRUE的项目集并将其余没为FALSE；然后可通过GET_MENU_ITEM_PROPERTY测试状态并通过SET_MENU_ITEM_PROPERTY进行设置
Separator	制作项目分隔符行以建立可视项目组
Magic	几种特殊菜单项目之一：Cut、Copy、Paste、Clear、Undo、About、Help、Quit和Window

可以将一个命令与检查和无线电菜单项相结合，但这样使用的大多数将产生不直观的作用，所以应该避免这样做。在检查一个复选框时，检查菜单项应该呈现相同的特性。接通和断开一个选项，不是启动一个进程而是设置一个属性。大多数用户界面比如 Windows界面的指南强调这些类型菜单项的特性： An Application Design Guide(应用程序设计指南))微软出版社1992)。

魔术般的Cut(剪切)、Copy(复制)、Paste(粘贴)、Clear(清除)、Quit(退出)和Window在Oracle Developer中都有内部功能性，这些功能也用适合于平台的方式 (位置和类型)处理它们的显示。必须为另外一些项建立命令文本或子菜单，比如 Help、About和Undo。

5. 在表单中建立菜单

要把一个菜单粘贴到表单上，首先编译这个菜单 (File | Administration | Compile或Compile File)，在Object Navigator中选择表单，然后在这个表单的 Menu Module(菜单模块)属性中输入菜单模块名。

当使用Call_Form程序启动表单时，可以告诉 Oracle Developer，使用当前的菜单而不是使用粘贴到要启动的表单中的菜单。这就让不同的表单共享应用程序中的单个菜单。

注意 还可以为菜单建立基于角色的安全性。有关细节请参考第9章的内容。

6. 按键分配

在表单应用程序中可以使用三种按键分配：标准功能键，菜单访问键和加速键。

前面部分提到过标准功能，比如 Execute_Query(执行查询)或Next_Record(下一个记录)。这些功能有缺省的键设置。可以利用 Oracle Terminal实用程序来改变它们，缺省设置是在菜单的PL/SQL命令文本中使用DO_Key，可以确保菜单命令和功能键有相同的作用。

大多数的GUI平台定义了一组菜单访问键。菜单访问键选择菜单项的按钮顺序 (ALT-F-G, CTRL-D，COMMAND-Q等等)，使用户不直接经过菜单。这样菜单访问键的作用就像用菜单项执行命令的键盘捷径。菜单通常根据平台的标准，利用专门的符号显示菜单访问键。例如，在Windows中，菜单访问键呈现为有下划线的大写字母，通过用ALT开头的字符顺序来执行它，并通过菜单条和子菜单上的访问键导航。例如， Ledger Summary报表菜单项是 ALI-Q-R-L (Query菜单，Report子菜单，Ledger Summary项)。

不用专门去做任何事情就可以得到这些菜单访问键。 Oracle Developer利用标签中的第一个大写字母或者第一个字母(如果设有大写字母的话)来作为菜单访问键。

如果上面的规则会对菜单标签产生两个或多个相同的菜单访问健，那么只要改变标签，告诉Oracle Developer要使用哪个键。可以用大写字母书写要使用的特定字符，如写成“rEport”而不是“Report”。如果不喜欢有趣的大写，那么在字符的前面放一个 &字符，例如“R&eport”。Oracle Developer除去这个&字符并使紧跟它的字母为菜单访问键。如果希望的话，还可以用两个&字符把一个文字“&”放入菜单项中。

Oracle Developer还提供加速键。加速键是一个键序列，可以把它和它的菜单访问键一起分配给菜单项。通常这样做是为了缩短键访问序列或把标准的键序列粘贴到某种标准的功能上。

Oracle Developer提供五种逻辑加速键：Accelerator1到Accelerator5。用Oracle Terminal来建立(第15章讲述有关细节)。在任何给定的菜单模块中，只能指定五个加速键。要指定某个键，在Object Navigator中找出菜单项，并显示它们的属性设置板。把逻辑名，比如 Accelerator3放入Accelerator属性中，然后利用Oracle Terminal，把这个逻辑键和物理键序列联系起来，比如CTRL-X或SHIFT-CTRL-Y。

本章介绍了在Form Builder中可以利用的高级图形设计技术。利用触发器、在第 5章中的其他控制特性、窗口、在本章学习的定制的菜单，可以建立高级的 GUI应用程序。以下两章把你的能力扩展到报表和图表。后面的章节给出一些专门的工具，使设计更安全、更耐用和可再利用。