第一部分

有关数据库的一些重要概念

第一章

公共知识

为快速开发出一个 ORACLE 应用程序并使之高效地运行，用户和开发者之间必须有一种共享的语言。并且他们还必须都对开发需求和 ORACLE 本身有着一致的较深刻的理解。

这种思想是应用程序开发中的一种新的思想!

过去，系统分析员在接到开发任务之后，先研究一下其开发需求，然后建立起一个应用程序来满足这种需求。在此过程中，用户的工作仅仅是描述一下自己的业务情况。在应用程序开发完毕之后或许再检查一下其功能。也仅此而已。

随着新的工具和方法的出现，在进行应用程序开发时，特别是用 ORACLE 开发时，应用程序是能够更紧密地满足业务的需求和业务人员的工作习惯的。但这只有在开发者和用户之间有着某种公共认识的情况下才有可能。

本书的目的就是要树立起这种认识，并为用户和开发者提供了大量的充分发挥 ORACLE潜力的方法。对于最终用户而言，他们对自己的业务情况有着详细的了解。但对开发者而言，一般他们对用户业务情况的了解是比较肤浅的，但他们对 ORACLE 的内部功能、特性以及计算机环境则有很深的认识。这些认识从技术上讲对最终用户而言是太专业了。但是，这些具有独占性质的专业知识领域，同用户和开发者在使用 ORACLE 可共享的知识比较起来，就显得比较次要了。

长期以来，“业务人员”和“系统人员”一直处于一种相互对立的状态。造成这种状况的原因是多方面的，如知识、文化、专业兴趣和目标的差异以及人们之间由于物理位置的分隔和造成的情感上的疏远等。但平心而论，这种情况并不是数据处理领域所特有的。在其他地方如会计部门、人事部门、管理部门都会发生这种情况。在那里，人们被分割在一个城市的不同的区域，或者是一幢建筑物的不同楼层。各团体之间人员之间的关系是那么地刻板，紧张，那么地不正常。不仅如此，还有其他一些人为地障碍，这些也都加剧了这种对立的状况。

有些人可能会讲：这不是很好吗？社会学家感兴趣的不正是这些吗？讨论这些跟 ORACLE 又有什么关系呢?

但我们应注意：ORACLE 并不是只有系统专业人员才能理解的一种秘密语言，事实上，ORACLE 从根本上改变了业务人员和系统人员之间关系的本质。任何人都可以理解 ORACLE，使用 ORACLE。现在，业务人员只需简单地输入一个英文查询命令，就可以立即访问那些由于系统中有其他人用来生成报表而受到访问限制的信息。在以前，这些信息是只有在释放以后才能被另外的用户访问的。

在使用 ORACLE 的地方，两大“阵营”人员之间的相互了解得以根本地改善。并

且他们都从对方学到了许多新的知识，他们之间的关系也得以正常化。这些也使得他们能开发出更优秀的应用软件，达到最佳的效果。

ORACLE 一直是建立在一种易于理解的关系模型（我们将在后面解释）的基础上的。所以，即使是非程序员也能容易地理解 ORACLE 能够完成的工作以及它是如何完成的，这使得 ORACLE 易于接近，也显得比较朴实。

并且，ORACLE 事实上能在各种类型的机器上运行，其功能也基本上相同。所以，不论用户的机器设备是哪个制造厂家生产的，都没有任何关系。这种特性为 ORACLE 的巨大成功作出了直接的贡献。

在当今计算机市场上充斥着大量经营“专有硬件”，“专用操作系统”，“专有数据库”，“专有应用程序”的计算机公司的情况下，ORACLE 给用户和系统部门提供了对他们的生活和未来的、新的控制能力。他们将不再局限于只能使用同一种硬件设备。在他们能拥有的几乎所有的计算机上，ORACLE 都能很好地运行。这是工作环境和应用程序开发领域的一场革命，其深远的意义是不言而喻的。

有些人或许还没有理解或接受这些观点。他们或许也还没有认识到由于时间和其他人为因素造成的“用户”和“系统”之间的障碍问题的严重性。但是“合作开发”技术的出现将极大地影响应用程序及其用途。

不幸的是，许多 ORACLE 应用程序开发者仍然在使用先前进行系统设计所积累起来的、现已过时或无益的设计方法。大多数这些方法（和限制）对前一代的系统设计是必不可少的，但现在用 ORACLE 进行设计时就完全没有必要再使用这些方法了。使用它们只会影响应用程序的效率。在解释 ORACLE 的过程中，我们首先将解除这些旧有方法和陈腐习惯造成的负担。事实上，ORACLE 中有其他更好的方法。

作者将用一种明白和简单的方式来解释 ORACLE，也即尽量让用户和开发者均能理解，并能共享这些知识。这将是我们贯穿全书的指导思想。另外过时的或不合适的设计和管理技术将被排除在外，取而代之的将是一些新的技术。

1.1 合作开发技术

ORACLE 是一种关系数据库，用这种观念来认识和管理业务处理中使用的各种数据是极为简单的。事实上，关系数据库无非是一些数据表的集合。表是我们在日常生活中经常会遇到的，如天气预报、股票行情、运动成绩等。所有这些都是一些表：带有列标题和在下面简单列出的各种信息。但即使是这样，关系的处理方法对于最复杂的业务处理也已经是足够了。

但不幸的是，事实上最能从关系数据库中受益的业务用户却经常对此了解极少。对于设计一个系统来满足用户工作需求的应用程序的开发者而言，他们也经常发现很难用简单的术语向用户解释关系数据库这个概念。所以，为使我们的合作开发技术能够成功，必须引入一种通用的语言。

本书的前两部分将以极易理解的术语解释关系数据库的含义，以及用户如何使用才

能达到最高的效率。这部分内容看起来好象只对用户有用，熟练的关系型应用程序设计者则可跳过这些章节而简单的把本书当成一个主要的 ORACLE 参考手册而使用。这种想法是比较有害的。因为这部分内容对熟练的设计者而言是简单一些，但它却给应用程序设计者提供了用于同用户讨论他们的需求和如何满足这些需求所必需的明白一致的术语。对于应用程序设计者而言，这种讨论可以帮助他们去掉不合适的设计习惯（也许他们自己还未认识到这些）。我们将在介绍关系型方法的过程中逐步指出这些问题。一定要认识到：在 ORACLE 应用程序开发过程中，使用一些只适合于非关系的开发的设计技术将严重地损害 ORACLE 的功能。

对于最终用户而言，在理解了有关关系数据库的一些基本概念之后，他们将能中肯地向应用程序开发者表明他们的需求，同时这些也将帮助他们理解这些需求将如何被满足。学完这些内容之后，一个普通的业务人员也能从初学者成为 ORACLE 专家。此时在使用 ORACLE 时，就可方便地获取并使用各种信息，象内行那样对报表和数据进行控制，并对应用程序所完成的操作和它如何完成这些操作有一个全面透彻的了解。ORACLE 将给用户提供对应用程序或专家级的查询实用程序的控制能力，并且在用户并没有充分使用所有可用的灵活性和功能时还将给出一些提示。

有了这些知识之后，用户还可以帮助程序设计者完成他们最不喜欢干的工作：生成新报表。在某些大的组织中，多达 95% 的编程工作是由新的报表请求造成的。由于用户在几分钟之内就可完成一个报表的生成，因此，他们也可能比较乐意完成这种工作。

1.2 数据的普遍性

图书馆中将会保存大量的号码，书名等。棒球卡片收集者则会保存运动员的名字，出生年月等内容。在业务处理中，必须保存客户、产品，价格、财政状况等信息片。所有上面列举到的信息片都将被称为“数据”。

信息专家们也许会认为只有把各种数据按照一种有意义的方式加以组织之后，这些数据才能称为是信息。按照这种理解，则可认为 ORACLE 也是一种能方便地把数据转换成信息的工具。ORACLE 将对各种数据进行排序和处理，以找出隐藏在这些数据之后尚未被发掘的其他有用的信息，如总和，购买趋势以及其他的一些关系。读者将学会如何完成这些操作。关键问题是要明确我们需要何种数据，然后才能完成以下的三个基本操作：获得数据、存储数据、从中获取有用信息。

完成这些基本操作之后，我们就可进行其他操作，如对数据的计算，把数据从一个地方移动到另一个地方，修改数据。这些操作就称为是数据处理。从根本上讲，数据处理包含有与上面相同的三个步骤，它们将影响信息的组织方式。

有时我们用一些简单的工具如纸和笔就能完成这些处理。但随着数据量的增大，工具也应随之而变化。此时我们也许会用到档案柜、计算器等。即使到了数据多得必须使用计算机的程度时，我们所需完成的任务仍然是相同的。

借助于关系数据库管理系统（简称为 RDBMS），如 ORACLE，我们可以用一种易

于理解和简便的方式来完成这些任务。ORACLE 完成的操作基本上是：

- 让用户输入数据。
- 保存用户输入的数据。
- 允许用户从其中取出数据并对之进行处理。

图 1－1 表明这种过程是何等简单。

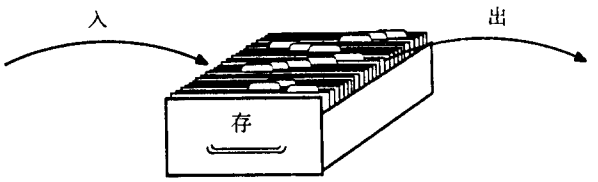

图 1-1 ORACLE 对数据的处理

ORACLE 支持这种“入-存-出”的方法，并提供了许多灵活的工具。借助于这些工具，我们可以完成数据的捕获、编辑、修改以及输入等操作，还可完成数据的保密，取出这些数据对之进行处理或用之生成报表。

1.3 ORACLE 语言

ORACLE 是以表的形式存储信息的，这一点与报纸上天气预报表很相近，如图1-2所示。

WEATHER

City	Temperature	Humidity	Condition
Athens	97	89	Sunny
Chicago	66	88	Rain
Lima	45	79	Rain
Manchester	66	98	Fog
Paris	81	62	Cloudy
Sparta	74	63	Cloudy
Sydney	29	12	Snow

图 1-2 报纸上的天气预报表

该表有四个竖直列：City（城市名），Temperature（温度），Humidity（湿度）以及 Condition（天气状况）。对于每一个城市的天气，则用一个行来表示。最后，该表还有一个表名：WEATHER。

我们见到的书面表大多数均有如下三个基本特征：列、行、表名。关系数据库中的表也是这样。每个人都可以理解 ORACLE 数据库中用于描述表的每一部分的词及其含义，这是因为这些词与日常生活中所用的是完全一样的，而并没有什么特殊的含义。引

用现在广泛流行的一句话来讲，就是所见即所得。

1.3.1 信息表

ORACLE 把信息都存储在表中，如图 1－3 所示。每个表均有一个或多个列。列的标题，如 City，Temperature，Humidity 和 Condition，描述了该列所存信息的类型。各信息是按城市一行一行存放的。每一个单一的数据集，如 Manchester（曼彻斯特）城的温度值，湿度值，以及天气状况都有其各自的行。

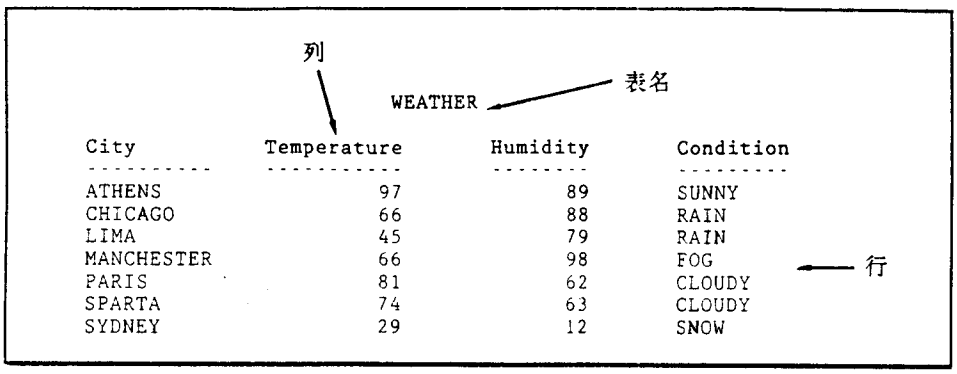

图 1-3 ORACLE 中的一个 WEATHER（天气）表

为使软件更易于让人接受和使用，ORACLE 尽量避免使用一些特殊的专业术语。在有关关系数据库理论的研究论文上，列名有时也称为是“attribute”（属性），行也通常被称为是“tuple”（元组）同 Couple 压韵，表则被称为是“entity”（实体）。但对最终用户而言，这些术语则可能会引起混乱。所以我们认为对于日常生活中已有的大家都能理解的名词，就没有必要再为之另取一个名字。既然 ORACLE 中使用了这种通用的语言，开发者当然也应能。我们必须对这种由于不必要的技术术语而引起的误解有清醒的认识。同 ORACLE 一样，本书中也将严格使用“表”，“列”，“行”这三个术语。

1.3.2 结构化查询语言

Oracle 是第一个发行使用基于英文的结构化查询语言（SQL）软件的厂商。这使得最终用户自己就可从数据中提取出有用的信息作成报表，而用不着让开发小组来形成每一个小报表。

ORACLE 的查询语言并不是没有任何结构的。它有其自己的语法和句法，但这些基本上与英文的正常语法是相同的，并且十分易于理解。

SQL 是一种令人吃惊的强大工具。后面我们将看到这一点。没有任何编程序经验的用户也可使用 SQL。

这里我们给出一个如何使用 SQL 的例子。假设现在我们有一张同上面一样的天气预报表，有人要我们从中选出湿度为 89 的城市，此时我们会马上回答“Athens”。若问温度为 66 华氏度的城市，我们也能立即回答出“Chicago 和 Manchester。”

ORACLE 也能如同人一样简单迅速地回答这些问题，并且这种简单查询的方式与

上面也是基本相同的。在 ORACLE 查询中要用到的几个关键字是 **Select**，**from**，**where** 和 **order by**。通过这几个词，ORACLE 就能正确地理解我们的要求并给出正确的回答。

1.3.3 简单的 ORACLE 查询

若在 ORACLE 系统中有 WEATHER，我们就可作如下简单的查询：

```
select City from WEATHER where Humidity = 89.
```

ORACLE 将作如下回答：

```
 City
 ---------
 ATHENS
```

然后我们还可继续查询：

```
select City from WEATHER where Temperature = 66.
```

对此查询，ORACLE 的回答可能是：

```
City
----------
MANCHESTER
CHICAGO
```

假定我们现在要按温度的高低顺序查看一下各城市的天气情况，怎么办呢？此时可以用 **order by**，如下所示：

```
select City, Temperature from WEATHER
  order by Temperature
```

此时 ORACLE 将立即回答如下：

```
City                 Temperature
-------------------- -----------
SYDNEY ..............  29
LIMA ................  45
MANCHESTER ..........  66
CHICAGO  ............  66
SPARTA ..............  74
PARIS  ..............  81
ATHENS ..............  97
```

从上面的显示结果，我们可以看到 ORACLE 按温度对表进行的重排。（这里最低的温度被最先列出来。后面我们将介绍如何指示 ORACLE 把最大的湿度显示在最前面。）

用 ORACLE 的查询实用程序，我们也可进行其他一些提问。这里举出这些例子只是为了说明从 ORACLE 的数据库中按对我们最为有用的形式获取信息是一件十分容易的事。我们还可以把一些简单的语句组合起来进行复杂的查询，完成这种操作的方法通常也是易于理解的。例如，**where** 和 **order by** 这两个语句本身都是十分简单的，我们可以将其组合起来使用，以指示 ORACLE 从数据库中提取出温度大于 80 的所有城市，并按温度增大的顺序将其显示出来。如下例所示：

```
select City, Temperature from WEATHER
Where Temperature>80              (>means greater than)
order by Temperature
```

ORACLE 将回答如下：

```
City                 Temperature
-------------------- -----------
PARIS                         81
ATHENS                        97
```

在更特殊的情况下，我们可以询问温度大于 80 而湿度小于 70 的城市：

```
select City, Temperature, Humidity from WEATHER
where Temperature > 80
      and Humidity < 70                  (< means less than)
order by Temperature
```

此时，ORACLE 回答如下：

```
City                 Temperature  Humidity
---------------      -----------  --------
PARIS                         81        62
```

1.3.4 关系数据库的含义

在 WEATHER 表中我们列出了来自几个不同国家的城市的天气状况，其中某些国家还有好几个城市在此表中。现在假设我们知道每个城市所在的国名，如图 1－4 所示。

WEATHER

City	Temperature	Humidity	Condition
ATHENS	97	89	SUNNY
CHICAGO	66	88	RAIN
LIMA	45	79	RAIN
MANCHESTER	66	98	FOG
PARIS	81	62	CLOUDY
SPARTA	74	63	CLOUDY
SYDNEY	29	12	SNOW

LOCATION

City	Country
ATHENS	GREECE
CHICAGO	UNITED STATES
CONAKRY	GUINEA
LIMA	PERU
MADRAS	INDIA
MADRID	SPAIN
MANCHESTER	ENGLAND
MOSCOW	U. S. S. R.
PARIS	FRANCE
ROME	ITALY
SHENYANG	CHINA
SPARTA	GREECE
SYDNEY	AUSTRALIA
TOKYO	JAPAN

图 1-4　WEATHER 表和 LOCATION 表

对于 WEATHER 表中的每一个城市，我们只需看一下 LOCATION 表，在 City 列

中找到该城市名，然后看一下同行的 Country 列的内容，即可知道该城市所在的国名。

从外面看起来，WEATHER 表和 LOCATION 表是两个完全独立的表。每一个表都有其各自的信息行和列。但是它们之间有一重要的相同之处：City 列。对于 WEATHER 表中的每一个 City 名，在 LOCATION 表中都有一个同样的 City 名。

例如，假定我们现在需要知道当前在 Australian（澳大利亚）的某个城市的天气情况，怎么找呢？现在请读者自己在上图中找一下，然后接着看下文。

读者的找法可能是：先在 LOCATION 表的 Country 列中找到一个 AUSTRALIA 项；由此知道该国的一个城市名 SYDNEY。然后用在 WEATHER 表的 City 列中查找 SYDNEY。找到之后，即可从同行的显示内容中找到其温度、湿度、以及天气状况信息：29，12 和 SNOW（下雪）。

所以，虽然这两个表是相互独立的，但我们也能一眼看出它们之间有着某种关系。这种关系就是一个表中的城市名是与另一表中的城市名相关的。这种关系就是关系数据库的基础，如图 1－5 所示。

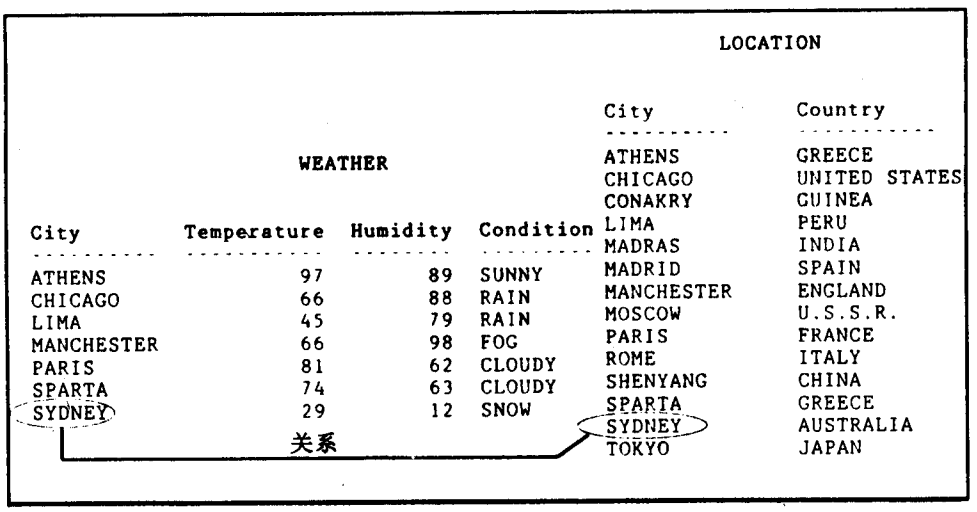

图 1-5　WEATHER 表和 LOCATION 表间的关系

这就是关系数据库的基本思想（有时也称为是关系模型）。数据是存储在表中的。表则有列，行和表名。若两表中都有包含相同信息类型的列，那么我们就可将这两个表连接起来。

由此我们可以看出，关系数据库还是十分简单的。

1.4　一些日常的例子

在我们理解了关系数据库的基本思想之后，我们就会发现，表，行，列是无处不在

的。这并不是我们以前从来没有见到过这些东西，只是那时候并不是按这种思考问题的方法来对待它们罢了。大多数我们这种习以为常的表均可存储在 ORACLE 中。然后我们就可以用这些表来回答其他一些方法需很长时间才能得到答案的问题。

在图 1－6 中，我们给出了一个典型的股票行情表。这里我们只是抽取了其中的一小部分。假设我们现在需要知道哪种股票交易所占份额最大，以及哪种股票价格变动百分点最多，是上扬还是下跌。这些均可借助于 ORACLE 中的查询而得到结果。这将比我们自己去搜索整个股票行情表要快得多。

Company	Close Yesterday	Close Today	Shares Traded
Ad Specialty	31.75	31.75	18, 333, 876
Apple Cannery	33.75	36.50	25, 787, 229
AT Space	46.75	48.00	11, 398, 323
August Enterprises	15.00	15.00	12, 221, 711
Brandon Ellipsis	32.75	33.50	25, 789, 769
General Entropy	64.25	66.00	7, 598, 562
Geneva Rocketry	22.75	27.25	22, 533, 944
Hayward Antiseptic	104.25	106.00	3, 358, 561
IDK	95.00	95.25	9, 443, 523
India Cosmetics	30.75	30.75	8, 134, 878
Isaiah James Storage	13.25	13.75	22, 112, 171
KDK Airlines	80.00	85.25	7, 481, 566
Kentgen Biophysics	18.25	19.50	6, 636, 863
LaVay Cosmetics	21.50	22.00	3, 341, 542
Local Development	26.75	27.25	2, 596, 934
Maxtide	8.25	8.00	2, 836, 893
MBK Communications	43.25	41.00	10, 022, 980
Memory Graphics	15.50	14.25	4, 557, 992
Micro Token	77.00	76.50	25, 205, 667
Nancy Lee Features	13.50	14.25	14, 222, 692
Northern Boreal	26.75	28.00	1, 348, 323
Ockham Systems	21.50	22.00	7, 052, 990
Oscar Coal Drayage	87.00	88.50	25, 798, 992
Robert James Apparel	23.25	24.00	19, 032, 481
Soup Sensations	16.25	16.75	22, 574, 879
Wonder Labs	5.00	5.00	2, 553, 712

图 1-6　股票行情表

图 1－7 列出了一些曲棍球队对阵的综合情况。在我们需要知道各队的相对排名情况或那个对战绩最为辉煌时，也可用 ORACLE 简单的英文查询。

图 1－8 是一新闻报纸的索引。如果要知道 F 部分都有些什么内容，或者是在我们从头至尾阅读该报表时将会以何种顺序读到这些文章，都可以借助于 ORACLE 的查询

来完成。我们将解释如何进行这些查询工作，如何创建一些表来存储这些信息。

OVERALL STANDINGS

Team	Won	Lost	Tied
Boston	17	13	3
Buffalo	21	9	4
Calgary	14	11	9
Chicago	19	13	2
Detroit	10	18	5
Edmonton	16	11	7
Hartford	16	17	1
Los Angeles	16	14	3
Minnesota	17	15	2
Montreal	20	13	4
New Jersey	15	15	3
N. Y. Rangers	15	14	5
N. Y. Islanders	11	20	4
Philadelphia	16	14	4
Pittsburgh	13	16	3
Quebec	6	23	4
St Louis	14	12	6
Toronto	16	18	0
Vancouver	11	16	6
Washington	13	15	4
Winnipeg	14	13	5

图 1-7　曲棍球比赛结果

Feature	Sections	Page
Births	F	7
Bridge	B	2
Business	E	1
Classified	F	8
Comics	C	4
Doctor's In	F	6
Editorials	A	12
Modern Life	B	1
Movies	B	4
National News	A	1
Obituaries	F	6
Sports	D	1
Television	B	7
Weather	C	2

图 1-8　报纸索引表

1.5 一个古老的例子

图 1－9 给出一个属于“G.B.Talbot”的分类帐本的一部分。G.B.Talbot 是从 1896 年开始使用该帐本的。

在帐本中有多不胜数的类似图中的各项目，一直记到 1905 年为止。Dora Talbot 每天付给她的工人一个美元，并在该帐本上记录下来。George B. Talbot（或许是 Dora Talbot 的儿子）将该帐本保存了起来。从中我们可以看到，只有极少数的几个工人出现了多次，其他的都只出现那么一两次而已。

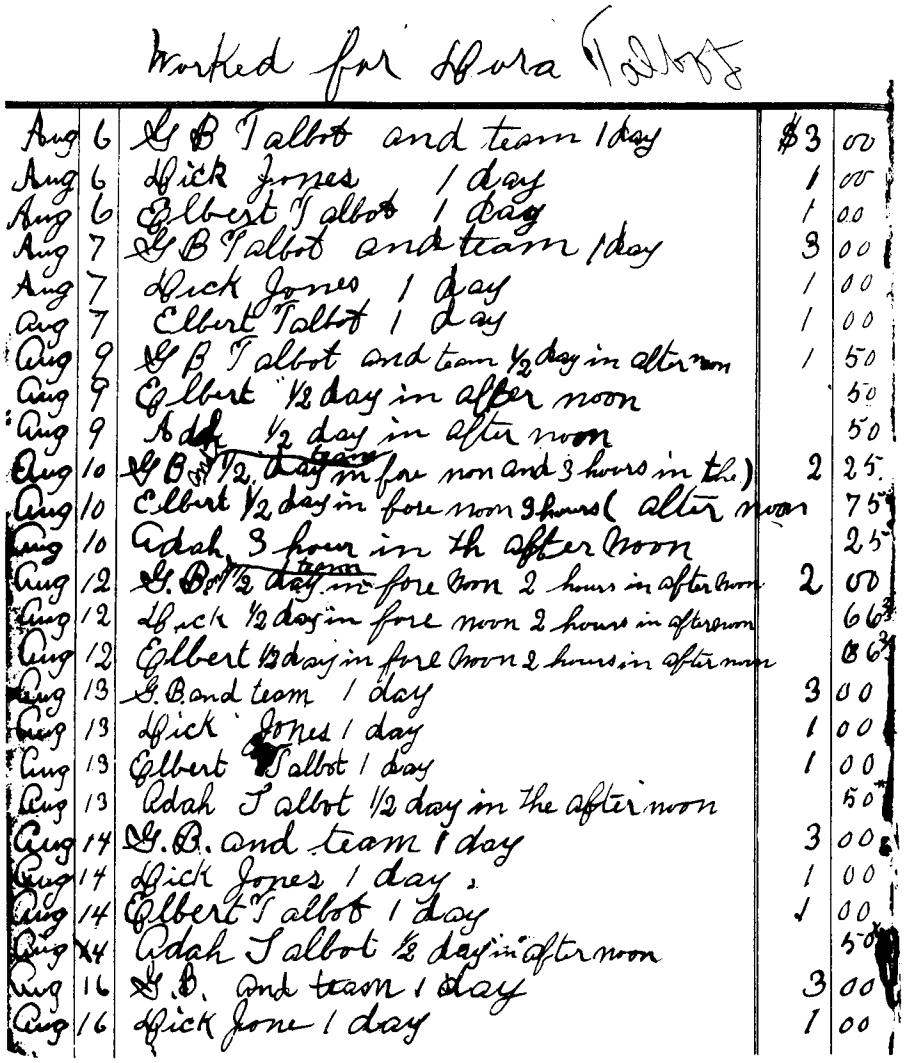

图 1-9 G.B.Talbot 的帐本

George 在另外一些页上记录了工人的名字和住址，如图 1－10 所示。我们将会看

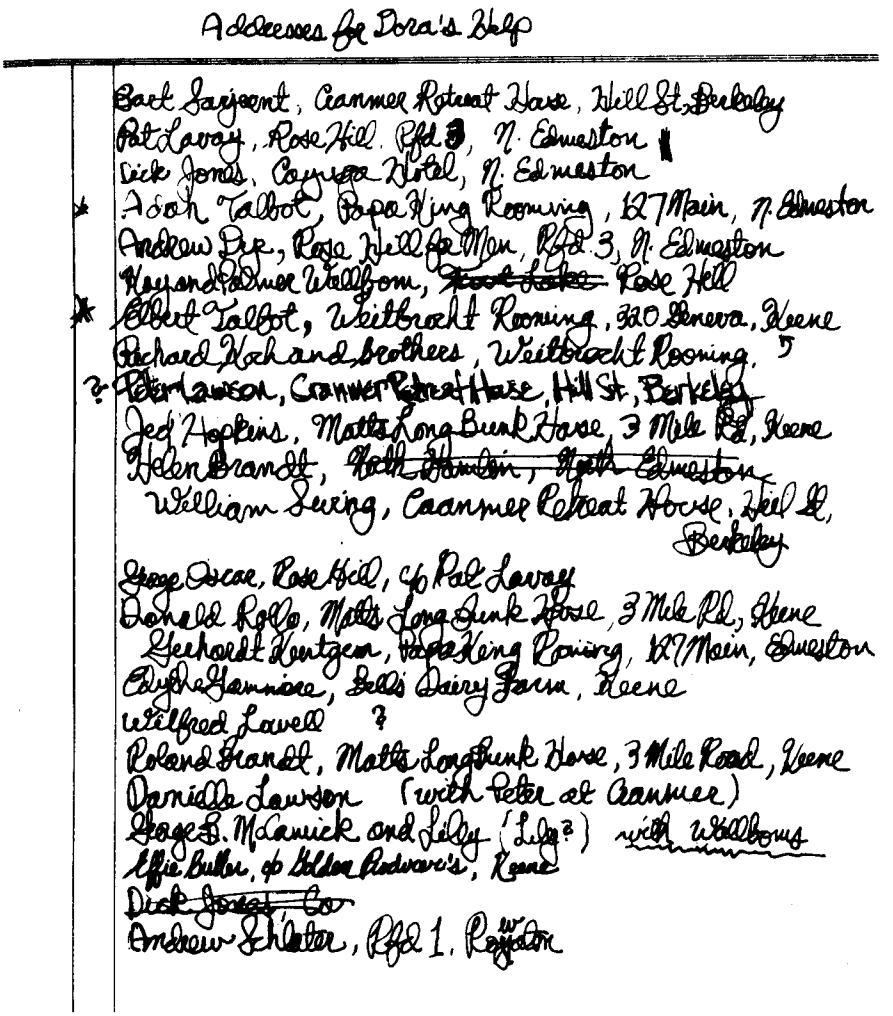

图 1-10　Talbot 帐本中工人的住址

到，可以用工人的名字将这两个表联系起来。设想一下，假如有月末 George 想派一个人将装着现金的信封送到每一个工人那儿去，他将怎么办呢？首先，他肯定会将付给某个工人的所有工资都加起来，然后把这些钱放到信封中，并在上面写上工人的名字。然后他将翻到他的帐本的第二部分，找到每一个工人的住址，在信封上写上地址之后再派人将信封送走。

G.B.Talbot 的这个帐本就是一个地地道道的关系数据库，只不过他的信息是用纸和墨水记录下来的，而不是计算机的磁盘。尽管这些表间的关联是用他自己的手，思维而不是 CPU 建立起来的，但他拥有的确实是一个合法的关系数据库。甚至可以讲他的这个关系数据库是符合规范化规则的（这条规则是关系数据库应用程序设计者所必需遵

循的)。规范化就是要求数据进行自然地分组。在 George.B.Talbot 的帐本中就没有把每天的报酬和地址混合在同一个部分（表）中。

该帐本的所有项都可以方便地转化为 ORACLE 的表。这样 G.B.Talbot 所面临的问题就会简单得多。本书的前面的部分将介绍如何完成这些操作，其中我们将用到当前的一些例子和 Talbot 的帐本以展示 ORACLE 的强大功能。

第二章

关系数据库中的危险

与任何新的技术或新的探索一样，我们不应只考虑它的益处和它所提供的机会，还要考虑它的代价和风险。在相对较新的技术如关系数据库中，多数公司还没有花费足够的时间让它们周围的“老手”知道应该避免什么以及如何避免。

ORACLE 把关系数据库和一系列有力而且易于使用的工具组合起来，于是由于其简单性而导致灾难的可能性成为现实。

本章讨论开发者和用户都需要考虑的一些危险。第三部分将更深入地讨论它们以及另一些附加的问题，尤其是开发者在其建立有适应能力的高效的应用程序的任务中感兴趣的问题。

2.1 真像他们所说的那么容易吗?

按照有关的销售者—工业福音师—的说法，用关系数据库及其相关的“第四代”工具开发一个应用程序要比传统的系统开发快 20 倍。而且这种开发非常容易，最终，所需要的程序设计者和系统分析者将越来越少，终端用户可以完全控制他们自己的命运。

但对有关方法的评论警告说，关系系统的固有速度比其它系统要慢，如果不采用更为传统的方法，一个能控制查询和写报表的用户可能会使计算机不知所措，公司将损失荣誉及财富。报刊也引证一些事例；某些大型应用程序投入生产时，其运行完全失败。

所以真实情况到底是怎样的？实际上竞争的规则已经发生变化。和传统的开发相比，第四代关系的开发对公司和管理提出了非常不同的要求。这些问题和风险很新，而且不明显。但一旦认清并懂得了这些，则风险不会更大，甚至可能比传统的开发更小。

2.2 风险是什么?

主要的风险是：它和他们所说的一样容易。理解表、列、行这些概念是不困难的。两个表之间的关系在概念上也很容易。即使“规范化”这一概念，它指对一个公司数据的各元素间的固有或“规范”的关系的分折过程，也是很容易理解的。

不幸的是，这种情况立即造就了“专家”，他们充满了自信和天真，但对于建立实际的、生产性的关系应用程序却几乎没有经验。对于走向市场的小型数据库，或者编制家庭财产目录，这样的应用程序，它不会带来很大的麻烦。错误将及时地暴露出来，这

样可以吸取教训，避免下次再发生错误。但在重要的应用程序中，它必定会带来灾难。报刊所报导的大型项目的失败后面通常隐藏着开发者的缺乏经验。

旧的开发方法通常较慢。部分是由于为进行检查和质量确认而增加了项目控制，但主要是因为旧的方法的任务——编码，为编译提交作业，连接，以及测试——导致了较慢的速度。这一循环，尤其是在主机上的循环，常常是冗长的，以致于程序设计者要花费大量的时间进行手工检查以避免由于代码中的一个错误而导致另一个完全循环的延迟。

第四代工具诱使开发人员仓促地进入生产。修改能够很快地实现，致使人们忽视了测试。实际上的对所有手工检查的消除带来了问题。当鼓励手工检查的反面激励（长的循环）消失时，手工检查也随之消失。许多看法是这样的："如果应用程序不是非常正确，我们能很快地修复它。如果数据有误，我们能很快地更新以修改它。如果速度不够快，我们能调节到很快的速度。我们提前得到它，并且展示我们做成的东西。"

这一问题由一个有趣的社会学现象弄得更糟：很多关系应用程序的开发者是刚毕业的大学生。他们在学校中学习了关系理论和设计，并且乐于作出一些成绩。更多的短期开发者，如一个班，还没有学习新的技术：他们忙于维持和加强他们已知的技术，这些技术支持他们的公司的当前的信息系统。结果是，和那些已经开发了几个完整的应用程序的人相比，没有经验的开发者倾向于结束有关项目，他们常常不愿意测试，而且对失败的后果不敏感。

一个重要的 ORACLE 项目中的测试循环应该比传统项目更长而且更完善。即使项目控制是正确的，或者是短期项目管理者在指导项目时也应该这样，这是由于越来越少的手工检查以及固有的过份的自负。这些测试必须检查数据入口屏幕和报表的正确性，数据加载和更新的正确性，数据的完整性以及并发性的正确性，尤其是峰值负荷时事务和存储容量的正确性。

由于实际上正如他们所说的一样容易，用 ORACLE 的工具开发应用程序是相当快的。但是它自动地减少了作为开发的规范部分的测试量，因此必须有意识地加强有计划的测试及质量确认以进行弥补。通常，那些对 ORACLE 或第四代工具比较陌生的人不能预见到这一点，但是你必须在你的项目计划中加入它。

2.3 新的图象的重要性

我们很多人都希望有一天，能象 Kirk（或 Picard）船长一样，简单地用"计算机…，"来用英语进行查询，并且立即得到答案。对家庭，我们希望有一天能够用英语输入"自然"语言查询，并且几秒内就从屏幕上得到答案。

我们比自己意识到的更希望达到这些目标。限制因素不再是技术，而是我们在应用程序设计中的僵化的思想。ORACLE 能够直接建立基于英语的系统，这些系统很容易被缺乏经验的用户理解和开发。ORACLE 数据库和工具中有潜力可用，但只有很少的人懂得并使用它。