第一部分　基 础 知 识

第一部分包括以下四章：

- 第1章介绍什么是数据库和为什么需要数据库系统。该章还简述了关系数据库系统和其他数据库系统之间的不同之处。
- 第2章介绍数据库系统的一般体系结构，即 ANSI/SPARC体系结构。该体系结构构成本书其他部分的基本框架。
- 第3章概括介绍了关系系统。其目的是为在第二部分和后续的章节中进一步的讨论奠定一个基础。该章还介绍了实例数据库供应商和零件数据库。
- 最后，第4章介绍了标准关系语言 SQL。

第1章　数据库管理概述

1.1　引言

数据库系统本质上是一个用计算机存储记录的系统。数据库本身可被看作为一种电子文件柜；也就是说，它是收集计算机数据文件的仓库或容器。系统用户可以对这些文件执行一系列的操作，例如：

- 向数据库中增加新的空文件；
- 向现有文件中插入数据；
- 从现有文件中检索数据；
- 更改现有文件的数据；
- 删除现有文件中的数据；
- 删除数据库中的现有文件。

图1-1显示了一个名为 CELLAR（酒窖）的小型数据库，它只包含一个文件。文件中依次包含各种酒的藏酒量情况。图 1-2显示了一个该数据库的检索操作，以及该操作返回的数据。注意：为清晰起见，在本书中，我们采用大写字母来表示数据库的操作名、文件名以及其他类似内容。实际使用中，采用小写字母输入这些内容会更方便些。多数系统其实对大小写是不区分的。

图1-3举例说明了在酒窖数据库中进行插入(insert)、修改(change)和删除(delete)操作的情况。插入和删除整个文件的例子在后面的第 3章、第4章、第5章及其他章节中给出。

从上面的例子中可以得出以下几点：

1) 像图1-1中CELLAR这样的计算机文件经常被称作表(table)（更准确地说，叫关系表(relational table) —— 参见1.3节及1.6节）。

BIN#	WINE	PRODUCER	YEAR	BOTTLES	READY
2	Chardonnay	Buena Vista	1997	1	1999
3	Chardonnay	Geyser Peak	1997	5	1999
6	Chardonnay	Simi	1996	4	1998
12	Joh. Riesling	Jekel	1998	1	1999
21	Fumé Blanc	Ch. St. Jean	1997	4	1999
22	Fumé Blanc	Robt. Mondavi	1996	2	1998
30	Gewurztraminer	Ch. St. Jean	1998	3	1999
43	Cab. Sauvignon	Windsor	1991	12	2000
45	Cab. Sauvignon	Geyser Peak	1994	12	2002
48	Cab. Sauvignon	Robt. Mondavi	1993	12	2004
50	Pinot Noir	Gary Farrell	1996	3	1999
51	Pinot Noir	Fetzer	1993	3	2000
52	Pinot Noir	Dehlinger	1995	2	1998
58	Merlot	Clos du Bois	1994	9	2000
64	Zinfandel	Cline	1994	9	2003
72	Zinfandel	Rafanelli	1995	2	2003

图1-1 酒窖数据库（文件名为CELLAR）

Retrieval:

```
SELECT WINE, BIN#, PRODUCER
FROM   CELLAR
WHERE  READY = 2000 ;
```

Result (as shown on, e.g., a display screen):

WINE	BIN#	PRODUCER
Cab. Sauvignon	43	Windsor
Pinot Noir	51	Fetzer
Merlot	58	Clos du Bois

图1-2 检索举例

Inserting new data:

```
INSERT
INTO   CELLAR ( BIN#, WINE, PRODUCER, YEAR, BOTTLES, READY )
VALUES ( 53, 'Pinot Noir', 'Saintsbury', 1997, 6, 2001 ) ;
```

Changing existing data:

```
UPDATE CELLAR
SET    BOTTLES = 4
WHERE  BIN# = 3 ;
```

Deleting existing data:

```
DELETE
FROM   CELLAR
WHERE  BIN# = 2 ;
```

图1-3 插入/修改/删除举例

2) 表中的行被看作是文件中的记录，表中的列被看作是这些记录的字段。本书中，通常在谈及数据库系统时我们会使用记录和字段这些术语（主要限于前两章）；而在谈及关系系统时，将采用行和列的叫法（见 1.3节及1.6节）。注意：实际上，在本书后续的更正式的讨论中，将会采用一些更为正式的术语。

3) 为了简化，在例子中我们默认列 WINE（酒名）和PRODUCER（生产商）为字符串，而其他的列为整数数据。我们将在第 3、4章，尤其在第5章中详细讨论有关列的数据类型问题。

4) BIN#为CELLAR表的主码（意思是表中没有两行包括同样的BIN#值），像在图1-1中一样，我们经常用双下划线来标识主码列。

5) 在图1-2和1-3中列出的操作或“语句”，如SELECT（查找）、INSERT（插入）、UPDATE（更新）和DELETE（删除）等，都是SQL语言的表达结果。SQL是关系数据库的标准交互语言，而且如今市场上几乎所有数据库产品都支持它。注意：“SQL”这一名字最初代表“结构化查询语言”，并且发音为“sequel”。现在SQL已经成为一种标准，其名字已根本不再有任何正式的字母缩写的含义，其发音更倾向于发“ess-cue-ell”。本书将采用后一种发音。

6) 注意SQL用关键字UPDATE来表示“修改”。这可能造成混乱，因为“update”这个词也经常用来概括INSERT、UPDATE和DELETE三种操作。在书中我们将通过用大小写来区分这两种含义，即小写表示增、删、改，而大写表示UPDATE操作符。

7) 正如大家所熟知的，目前使用的绝大部分数据库系统实际上都是关系型的（或者至少应该是关系型的——参见4.7节）。由于这一原因，书中的讨论重点也将针对这类系统。

最后提及一点：本章以及下一章内容是基础性的，它对全面正确认识当今数据库系统的特征及功能至关重要。但是，不能否认这两章内容有些抽象和枯燥，而且涉及大量新的概念与术语。在书中以后的部分——尤其是第3章和第4章——读者会发现其内容不再那么抽象，相应地也变得容易理解。因此可以先略读前两章，然后在以后遇到直接相关的问题时再仔细重读这些内容。

1.2 什么是数据库系统

如前所述，数据库系统是指一个计算机存储记录的系统，即，它是一个计算机系统，该系统的目标是存储信息并支持用户检索和更新所需要的信息。这里所讨论的信息可以是个人或企业所关心的任何信息，换句话说，它是指任何对个人或组织经营企业的一般处理过程有帮助的数据。

注意：本书中，术语“数据”和“信息”是同义词。有些作者更喜欢区分两者的含义，即用“数据”表示在数据库中实际存储的内容，而用“信息”表示一些用户对这些数据的理解。这一差异显然很重要，但不适合使用在两个本来同义的术语间生造出来的差异加以区别，我们更倾向于在适当的地方采用明确表述的方式。

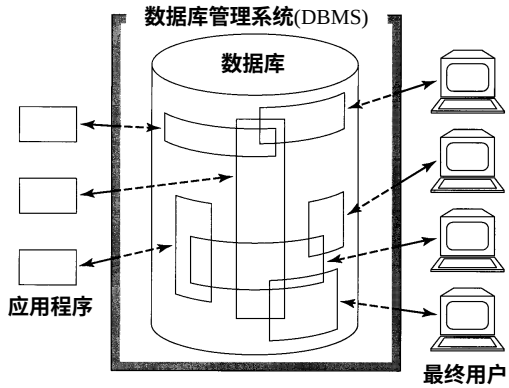

图1-4 数据库系统简图

图1-4是一个数据库系统的简图。它显示了数据库系统包括的四个主要部分：数据、硬件、软件和用户。下面将简略地介绍这四个部分。当然，以后会对每一部分进行更详细的讨论（除了硬件部分，硬件的细节内容已经超出了本书的范围）。

1. 数据

数据库系统可用于小至个人机、大到大型机的各种计算机。显然，任一系统所能提供的功能，某种程度上要决定于其运行的机器的大小与能力。尤其是大型机上的系统（“大型系

统”)趋向于多用户，而小型机上的系统(“小型系统”)趋向于单用户。一个单用户系统在任何时候最多只有一个用户访问数据库系统。多用户系统可以同时有多个用户访问数据库系统。如图1-4所示。为了不失一般性，本书中假定采用后一种系统，但实际上这一区别绝大多数用户是不关心的，因为通常的多用户系统的主要目的就是让每个用户感觉他或她像是在单用户系统上操作。多用户系统的问题主要是系统内部的问题，而不是那些用户可见的问题 (见本书第四部分，尤其是第5章)。

注意：为简单起见，不妨假定系统的全部数据都存储在一个数据库中，本书中即采用这一假定(因为这样本质上并不影响任何其他的讨论)。然而实际上，即使在小系统中，也经常需要将数据分散在不同数据库中存放。我们将在第2章及其他章中提到这其间的原因。

通常，数据库中的数据，至少在大型系统中，既是集成的，又是共享的。正如将在 1.4节所看到的，数据集成与数据共享代表着在大型环境中数据库系统的主要优点。其实数据集成在小型环境中也具有重要意义。当然，在小型环境中也还有很多其他的优点 (以后将讨论)。但现在首先解释集成与共享的含义。

- 集成，指的是数据库可以被当作几个不同文件的合并，数据库至少可以部分地消除文件之间的冗余。例如，一个指定的数据库可以包含一个 EMPLOYEE (雇员)文件和一个ENROLLMENT(注册)文件。EMPLOYEE文件中给出雇员名、地址、部门和工资，等等。ENROLLMENT文件表示在接受培训的雇员的登记 (参见图1-5)。现在假定，为了进行培训课程管理，需要清楚每个在训学员所在的部门。那么，显然没有必要在ENROLLMENT文件中重复有关的部门信息，因为需要的话，可以从 EMPLOYEE文件中找到。

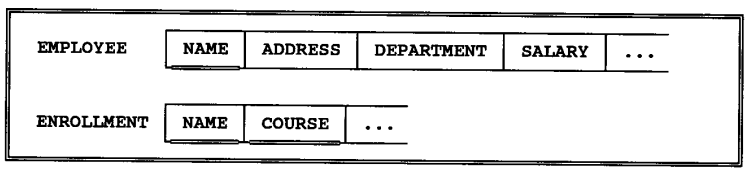

图1-5 EMPLOYEE 和ENROLLMENT文件

- 共享，指的是数据库中的每项数据可以被不同的用户共享。换句话说，每一个用户都可以因不同的目的而访问相同的数据。如上所述，不同的用户甚至可以同时访问同一数据(“并发访问”)。之所以有并发共享或其他方式的共享，部分是因为数据库是集成的。比如，在上述例子中的部门信息共享就很有代表性， EMPLOYEE文件中的部门信息既可以被人事部门的用户共享，也可以被教育部门的用户所共享，即这两个部门的用户因为不同的目的而使用这一共享信息。注意：如果数据库是非共享的，则它通常被称作个人数据库或专用数据库。

数据库集成和共享带来的另外一个结果，是任一用户都只关心整个数据库中的一小部分，而且不同用户所使用的数据会以各种方式重叠。换句话说，对于一个指定的数据库，不同的用户会以许多不同的方式来观察。实际上，即使两个用户共享数据库中同一块数据，在细节层上，他们看待数据的角度也会有所不同。这一点将在 1.5节，第2、3章，特别是在第9章中进行详细讨论。

我们将在1.3节中详细阐述数据库系统中数据部分的本质特征。

2. 硬件

系统的硬件部分包括：

- 二级存储设备，以及相关的I/O设备（磁盘驱动器等）、设备控制器、I/O通道等。二级存储设备（大部分为磁盘）用来存放数据。
- 硬件处理器和相应的主存。硬件处理器和相应的主存用于支持数据库系统软件的执行(见下一小节)。

本书将不对系统硬件部分作过多介绍。这主要基于以下几点原因：一是硬件部分内容庞杂，自成体系；二是硬件方面的问题不是数据库系统独有的；三是硬件方面的有关知识在其他资料上已经详细地阐述了。

3. 软件

在物理数据库（例如物理存储的数据）和数据库系统的用户之间有一层，即软件层，它通常被称作数据库管理器或数据库服务器，而其最通用的称法为数据库管理系统(DBMS)。所有访问数据库的请求都是由DBMS来处理的。DBMS提供了许多对数据操作的实用程序，如1.1节中的增加和删除文件或表，也可以在这些文件中检索或更新数据。DBMS提供的基本功能为数据库用户屏蔽掉了物理层的细节（就像程序设计语言系统为应用程序员屏蔽掉物理层细节一样）。换句话说，DBMS为用户提供了一种在硬件层之上观察数据库的高级别方式，并且支持用户以这种高级别方式表达操作请求（如在1.1节简要讨论的SQL操作）。本书将会详细讨论DBMS这方面的功能以及其他功能。

还有两点进一步的说明：

- 在整个系统中，DBMS是最重要的软件部分，但不是唯一的。其他软件包括实用程序、应用开发工具、设计辅助、报表书写器和事务管理器或事务处理监控器(TP monitor)。见第2、3章和第五部分有关这方面的更进一步的讨论。
- DBMS这一术语也通常用于指某个特定厂商的特定产品。例如，IBM的基于OS/390的“DB2 Universal Database”产品。术语DBMS实例(instance)有时用于指这些产品运行于某一特定的计算机设备上的特定拷贝。正如大家将会看到的，有时有必要区别这两个概念。

注意：有时企业的人员用数据库这一术语而实际上是指DBMS。以下是一个典型的例子：“X厂商的数据库与Y厂商的数据库的性能比是二比一”。这种用法是不恰当的，但是非常、非常普遍（当然，问题是如果我们称数据库管理系统为数据库，那么我们怎么称数据库？）。

4. 用户

我们考虑大致三类主要用户（相互间可能会有些重叠）：

- 首先是应用程序员。应用程序员负责编写数据库应用程序。他们使用某些程序设计语言，如COBOL、PL/I、C++、Java或某种高级的第四代语言（见第2章），来编写应用程序。这些程序通过向DBMS发出SQL语句请求来访问数据库。这些程序通常可以是批处理应用程序，或联机应用程序，目的是允许最终用户通过联机工作站或终端访问数据库。大多数当今的应用程序都是联机方式的。
- 第二类用户是最终用户。他们从联机工作站或终端与系统交互。最终用户可以通过在前一段提到的联机应用程序访问数据库，或者他或她可以使用数据库系统软件提供的接口。当然，这些由厂商提供的接口也可以被联机应用程序的方式所支持，但是这些应用程序是固有的(builtin)，而不是用户编写的。大多数数据库系统至少包括一种固有的应用程

序，即查询语言处理器，通过它用户可以交互地发出数据库请求（就是人们所熟知的语句或指令）给DBMS，诸如SELECT和INSERT。在1.1节中提到的SQL语言就是一种数据库查询语言的典型实例。

注意：“查询语言”一词虽很常用，但其实它有用词不当之嫌。因为自然语言的动词“查询”只表示查找数据，而查询语言通常（并不总是）还提供更新和其他操作。

大多数系统也提供其他固有的界面。通过这种界面，用户根本不发出像SELECT这样明确的数据库请求，而是代之以选择菜单中的一项或填充表格中的一栏。这样的菜单驱动或表格驱动界面对于并未在IT中受过正式训练的人来说更容易（IT即信息技术；IS是信息系统的缩写词）。然而，命令驱动界面——如查询语言——需要一定量的专业训练（很显然不像用COBOL语言编写一个应用程序那样麻烦）。同时，命令驱动式界面有时会比菜单驱动或表格驱动界面更灵活。在这类界面中，其查询语言中包含了某些特性，而它们可能不被其他界面所支持。

- 第三类用户没有在图1-4中显示。他们是数据库管理员或简称为DBA。有关数据库管理员职能的讨论，请参照1.4节和第2章的2.7节。

以上完成了我们对数据库系统主要方面的初步描述，在以后章节中我们还会更进一步讨论这方面的内容。

1.3 什么是数据库

1. 持久数据

数据库中的数据通常被认为是持久存储的（尽管事实上保存并非很久！）。对于持久性，直观上，是为了把数据库中的数据与其他的输入数据、输出数据、控制语句、工作队列、软件控制块和中间结果等临时数据相区分，而且通常任何数据本质上都是暂时的。更精确地说，我们说数据库中的数据是持久的，是因为一旦数据进入数据库被DBMS接受，就只有向DBMS提出某些明确的请求时，才能从数据库中删除数据。这有别于某些程序运行结束时产生的副产品。这种持久性的概念使得我们可以给数据库下一个更精确的定义：

- 数据库是一个持久数据的集合，这些数据用于某企业的应用系统中。

这里“企业”一词只是一个方便的通常的叫法，如任何独立的商业组织、科学组织、技术组织或其他组织。企业可能仅是个人（一个小的个人数据库），或者一个公司或类似的大型实体（有一个大型的共享数据库），或任何界于两者之间的单位。如以下的例子：

1) 制造公司
2) 银行
3) 医院
4) 大学
5) 政府部门

任何企业必须保存自身运作的大量数据。这些就是上面提到的持久数据。上述企业主要包含下列持久数据：

1) 生产数据
2) 会计数据
3) 病人数据

4) 学生数据

5) 计划数据

注意：本书的前几版使用“操作型数据”代替“持久性数据”。早期的术语反映了数据库的重点在操作型应用或生产型应用。例如，例行的、经常重复的应用，这些应用程序要重复地执行以支持企业的日常运行（例如，银行系统中支持存款和取款操作的应用程序）。联机事务处理用于指这种环境。但是，渐渐地，数据库也面向其他类型应用，例如，决策支持应用，这样，操作型数据的说法就不再合适了。事实上，目前的企业经常有两个独立的数据库：一个保存操作型数据；另一个称为数据仓库，保存决策支持数据。数据仓库通常包括概要信息（如，总计、均值），而这些概要信息来自于操作型数据库，也就是说，以一定时间间隔，一天一次或一周一次从操作型数据库中抽取这些信息。对决策支持数据库及其应用的进一步讨论见第21章。

2. 实体与联系

现在我们稍微仔细地分析一个制造企业 (名为“ KnowWare公司”)的例子。这种企业主要关心以下信息：眼下已有的工程；在这些工程中所使用的零件；提供零件的供应商；储存零件的仓库；在这些工程中工作的雇员，等等。工程、零件、供应商等构成了基本的实体，公司需要记录这些信息（在数据库中，实体一词通常指数据库中表示的任何可区分的事物）。如图1-6所示。

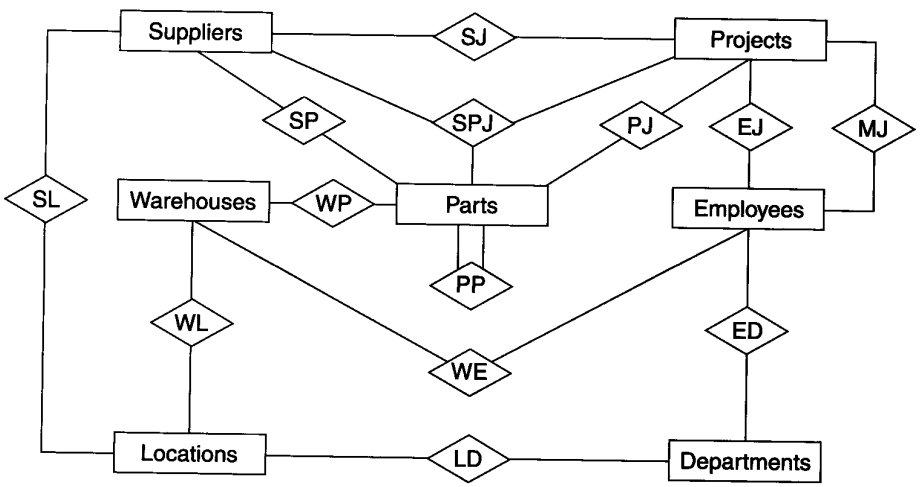

图1-6 KnowWare公司的实体/联系(E/R)图

除了基本实体本身（例子中的供应商、零件等），还有连接这些基本实体的联系。这些联系在图1-6中通过菱形和连线表示。例如，在供应商和零件之间有一个联系“ SP”：每个供应商供应某些零件；反过来，每个零件都由某些供应商提供（更准确地说，每个供应商供应某些种类的零件，每种零件都由某些供应商提供）。类似地，零件在工程中被使用；反过来，工程要使用零件（联系PJ）；零件被存放在仓库中，并且仓库中存放着零件（联系 WP）；如此等等。注意这些联系都是双向的，也就是说，每个方向都可调换。例如，在供应商和零件之间的联系SP可用来回答下面两个问题：

- 指定一个供应商，获得由该供应商提供的零件；

• 指定一种零件，获得供应这种零件的供应商。

重要的是这一联系（当然，以及图中表示的其他所有的联系）和基本实体一样作为数据的一部分。它们和基本实体一样都要在数据库中表示。注意：在一个关系系统中，如图 1-1所示，实体及联系都是用表来表示的。在第 3章中也可以见到。

上面已经说明图1-6是一个实体/联系图（E/R图）的例子。第13章将详细讨论这种图。

图1-6还说明了以下的几点重要内容：

1) 尽管该表中的大部分联系是涉及两个表的，也就是二元关系，但是这并不意味着所有的联系必须是二元的。例子中联系（“ SPJ ”）就涉及三个实体类型（供应商、零件和工程），它是一个三元关系。这个三元关系表示某些供应商为某些工程供应某些零件。注意，这个三元关系 (“ 供应商给工程提供零件 ”)和下面三个二元关系的联合是不同的，即“ 供应商供应零件 ”、“ 零件用于工程 ” 和 “ 工程由供应商供应 ”。例如，语句

a. 史密斯(Smith)给曼哈顿(Manhattan)工程提供活动扳手。

所提供的信息要多于下面三个语句 :

b. 史密斯供应活动扳手；

c. 活动扳手用于曼哈顿工程；

d. 曼哈顿工程由史密斯供应。

知道了b、c和d，我们不能有效地推断出 a。更精确地说，如果已知 b、c和d，我们就可以推断出史密斯给某项工程 (如工程Jz)供应活动扳手，某个供应商 (如供应商S*x*)供应活动扳手给曼哈顿工程，史密斯给曼哈顿工程提供某种零件 (如零件P*y*)。但是我们不能推出S*x*是史密斯，或者P*y*是活动扳手，或者Jz是曼哈顿工程。这样的错误推论有时被称作连接陷阱。

2) 图中还显示了只含一个实体（零件）类型的联系（ PP)。这一联系指出某种零件把其他零件作为直接的组成部件（也称作材料帐单联系）。例如，螺丝钉是铰链的一个组成部分，而铰链也被看作零件，或许也是某个更高层零件（如盖子）的组成部分。注意，这个联系仍然是二元的，只是它所连接的两个实体恰好是同一类型或同一个实体。

3) 通常，一组指定的实体类型可能由一些不同联系连接起来。在图 1-6的例子中，涉及工程和雇员的有两个不同的联系：一个（ EJ）表示雇员被分配到工程中这一事实；另一个（MJ）表示雇员管理工程这一事实。

现在来分析把联系本身看作一个实体的情况。若从实体的定义 (任何具有有用信息的对象) 来看，联系也符合这一定义。例如，“ 零件P4储存在仓库W8 ” 是一个实体，我们可以记录有关信息 —— 例如，相应的数量。而且，对于实体和联系不加以不必要的区分会有很多好处 (其讨论超出本章的范围)。因此，本书通常只把联系当作一种特殊的实体。

3. 属性

如前所述，实体是具有有用信息的对象。实体（包括联系）具有属性，这些属性记录实体的相应信息。例如，供应商有地址；零件有重量；工程有优先级；任务有开工期；如此等等。这样的属性必须在数据库中表示。例如，数据库可能包含表 S，表示供应商，表中有一列CITY表示供应商地址。

通常，属性可以简单，也可以很复杂。例如，“ 供应商地址 ” 这一属性可以设定得非常简

单，仅表示城市名，在数据库中只用一个简单的字符串表示即可。相反，数据仓库中可能会包含一个属性“建筑平面图”，这个属性就会很复杂，可能包括整个建筑图和相应的描述文字。在本书编写时，大多数据库产品才刚刚开始支持图像和文本这样复杂的属性。本书后面第 5章和第六部分会重点讨论这个问题。在此之前，仍假定属性都是简单类型，而且可以用简单的数据类型来表示。这些简单的数据类型包括数字、字符串、日期和时间等。

4. 数据和数据模型

对数据和数据库存在另外一种重要的认识方式。“数据”一词来自拉丁文“给，供给”。这样，数据就是指定的事实，从中可以推出另外的事实（从指定的事实中推出另外的事实恰恰是DBMS响应用户要求时所做的处理）。一个“指定的事实”要符合逻辑学家所说的真命题[⊖]；例如，陈述句“供应商 S1住在伦敦”就是这样的真命题。因此，数据库实际上就是这些真命题的集合[1.2]。

关系数据库系统之所以在工业界和理论界都能占主导地位的原因之一是，关系数据库系统直接支持对数据和数据库的解释。关系系统是基于一种称为关系数据模型的形式化理论。该理论指出：

- 数据通过表中的行来表示，并且这些行可以被直接解释为真命题。例如，在图 1-1中，BIN# 72这一行可以被解释为如下真命题：

“Bin 号码为72的行包含两瓶1995年的Rafanelli Zinfandel酒，这些酒到2003年才可饮用。”

- 所提供的操作符可以针对表的行进行，这些操作符直接支持从指定的命题推出另外的真命题的处理。举个简单的例子，关系投影操作符可以从前面列出的真命题推出下列的真命题：

“一些Zinfandel酒到2003 年才可饮用。”

（更精确地说，“在某个bin中的某些 Zinfandel酒，是由某厂商于某年生产的，到2003年才可饮用。”）

关系模型不是唯一的数据模型，还存在其他模型（见 1.6节）。这类模型与关系模型截然不同，一般不具有规范的形式逻辑基础。通常的数据模型是什么？这里给出如下的定义：

- 数据模型是对对象、操作等的一个抽象的、自包含的逻辑定义，这些定义合起来构成了一个面对用户的抽象机。其中对象可以用来建模数据结构。操作符用于建模一些行为。

通常，需要区分模型及其实现(implementation)，其定义为：

- 对指定的数据模型的实现是指在真实机器上的物理实现，一个真实机器是抽象机的组成部分，它们一起构成模型。

简而言之，模型是用户必须知道的；实现是用户不需要知道的。

注意：不难看出，模型与实现之间的区别就如同典型的逻辑与物理之间的区分一样。但是，今天的许多数据库系统(甚至一些关系系统)并没有分清这些区别。确实，目前尚缺乏对这一问题的认识。由此造成了在数据库原理 (即数据库应当怎样)和数据库实践(即数据库实际如何)之间经常存在鸿沟。本书中主要涉及原理，因此有必要提醒读者当你使用商业产品时，会经常遇到一些与原理有差异的情况。

在结束本节之际，还应该提到的是，实际上数据模型这个词在字面上就有两个十分不同的含义。一个是上面提到的；另一个是指作为某特定企业 (例如本节前面提到的制造公司

⊖ 逻辑中的命题是指可以明确判断真假的句子。例如，“威廉·莎士比亚写了《傲慢与偏见》”就是一个命题，该命题为假。

KnowWare公司) 的持久数据模型。两者之间的区别如下：

- 第一种含义下的数据模型就像编程语言 —— 虽然有点抽象 —— 它可被用来解决各种特定的问题，但是，它本身与特定的问题毫无联系。
- 第二种含义下的数据模型就像一个用上述语言编写的特定程序。换句话说，第二种含义下的数据模型利用第一种含义下的模型所提供的一些功能，用于解决一些特定的问题。因此它可以看作是第一种含义下的数据模型的特定应用。

本书中，在没有明确说明的情况下，数据模型一词都是指第一种含义。

1.4 为什么用数据库

为什么要用数据库呢？它有什么优点？在某种程度上，问题的答案依赖于所谈的系统是单用户的还是多用户的。更准确地说，多用户系统有更多的优点。先看单用户系统：

仍以酒窖的藏酒量为例(图1-1)，该图例示了单用户系统的情况。该数据库太小也太简单，不足以展示数据库的优点。但是设想一个大酒店有类似数据库，其库存可能有成千上万瓶酒，而且库存变化十分频繁；或者设想一家卖酒的商店，也有相当大的库存，并且库存周转率很高。在这种情况下，数据库系统与传统的、基于纸的记录保存方式相比，其优点可能就显而易见。以下列举了一些优点：

- 简洁：不需要成卷的大量文件。
- 快捷：机器对数据的变动和更新比手工快得多。尤其是，对于实时查询可以不要费时的手工查找而快速地给出答案。
- 省力：不再手工保存大量的文件，机械的任务可以由机器更好地完成。
- 方便：可以随时得到准确、最新的信息。

上述的好处在多用户环境能更好地体现出来。因为通常多用户环境下的数据库要比单用户的更大也更复杂。然而，多用户环境下还有一个很突出的优点，即，数据库系统保证了企业对数据的集中控制（对这一点，大家应意识到，它是最有价值的资产之一）。这种情况与没有数据库系统的企业相比，形成了鲜明的对照。在典型的没有数据库系统的企业，每个应用拥有各自的文件 —— 经常是各自的磁带和磁盘 —— 以致于难以用任何系统的方法来控制这些非常分散的数据。

1. 数据管理和数据库管理

前面简明扼要地描述了集中控制的概念。这一概念隐含着企业中要有某个可确认的人对数据有着核心的权力。这就是前面1.2节简要提到过的数据管理员（简称为DA）。假如数据是一个企业最有价值的资产，就一定要有人懂得这些数据以及企业对这些数据的需求，并且要处于企业的高级管理层。数据管理员就是指这样的人。因此，数据管理员的工作就是首先决定什么数据存储在数据库中，一旦存储了这些数据就要建立维护和处理这些数据的机制。例如，在什么情况下谁可以执行什么操作就是一种机制 —— 换句话说，它是数据安全机制（见下一小节）。

注意，数据管理员是管理者而不是技术人员（尽管他或她当然要在技术上对数据库系统的性能有所了解）。负责执行数据管理员的决定的技术人员就是数据库管理员（简称为DBA）。与数据管理员不同，DBA是信息技术（IT）方面的专业人员。数据库管理员的工作是创建实际的数据库以及执行需要实施各种决策的技术控制。数据库管理员也负责确保系统正确执行操作，并且提供各种其他技术服务。数据库管理员通常包括一些系统程序员和其他技术助理

（也就是说，数据库管理员的功能实际上由一组人员来承担，而不是一个人）；但是为了简化，通常假定数据库管理员只是一个个体。我们将在第 2章具体讨论数据库管理员的职能。

2. 数据库方法的优点

本小节给出一些特定的优点，这些优点来自前面所述的集中控制。

1) 数据共享。我们在1.2节讨论了这一点，但是为了完整，我们在这里再次提到。共享不仅指现有的应用程序可以共享数据库的数据，而且新的应用程序也能对这些数据进行操作。换句话说，不向数据库中添加任何新数据也可能满足新应用程序的数据要求。

2) 减少冗余。在非数据库系统中，每个应用程序都有自己的专用文件。这种情况经常导致在存储数据上有相当大的冗余，结果浪费存储空间。例如，一个有关人事的应用程序和一个有关教育的应用程序可能同时拥有包含职员的部门信息的文件。但是，如 1.2节所示，这两个文件可以集成起来消除冗余，只要数据管理员意识到两个应用程序的数据要求 —— 也就是说企业应有必要的全局控制。

3) 避免不一致（某种程度上）。这是前一点必然的结果。假定一种实际情况 —— 雇员E3在部门D8工作 —— 数据库中有两个不同的条目。还假定 DBMS也没有意识到冗余的存在（也就是对冗余失控）。则必然会有两个记录不一致的情况：即，当其中一个更新时，另一个不变。这种情况称为数据库不一致。显然，处于不一致状态的数据库可能提供给用户错误的或矛盾的信息。

当然，如果指定事实是由一条记录表示（也就是如果排除了冗余），那么这样的不一致就不会发生。另一种选择是，冗余没有排除但是受到控制（被 DBMS得知），那么数据库管理系统就可以保证对用户来说数据库总是一致的， DBMS确保两个记录中的任何一条改变会自动地应用到另一条。这一过程即为传播更新。

4) 提供事务支持。事务是一个逻辑工作单元，它包括一些数据库操作（特别是，一些更新操作）。常见的例子如从帐户 A到帐户 B转移一定的现金数。显然，这里要求两个更新操作：一个是从帐户 A提出现金；另一个是把现金数存入帐户 B。如果用户已经说明两个更新是同一事务的一部分，那么系统要确保两个操作要么都做，要么都不做 —— 即使在系统执行过程中出现故障（比如因为电源断）也应如此。

注意：刚刚举例说明的事务的原子性不是事务支持的唯一优点，但是与其他一些优点不同，它甚至可以应用到单用户的情况。事务支持的各种优点和怎样实现的全面描述见第14章和第15章。

5) 保持完整性。完整性的问题是确保数据库中的数据是正确的。同样事实的两条记录的不一致，就是缺少完整性的例子（见前一小节的讨论）；当然，只要在存储的数据中有冗余，就会引起这样的问题。即使没有冗余，数据库也可能包含错误的信息。例如，可能显示雇员一周工作了 400小时而不是40小时，或者属于一个不存在的部门。数据库的集中控制可以有效地避免此类问题。做法是通过支持数据管理员定义一些完整性约束（也称为商业规则），由DBA加以实施，完整性约束在任何操作执行时都得到有效的检验。

值得指出的是数据完整性在数据库中要比在各自独立的文件系统中重要得多，因为数据库中的数据是共享的。要是没有正确的控制，有可能一个用户错误地更新数据库而生成的错误数据，会殃及其他无辜的用户。目前，数据库厂商对数据库的完整性

约束的支持还相当不够（尽管最近这一方面的情况有所改善）。这一事实很不幸（见第8章），因为数据库完整性既基本又非常重要。

6) 增强安全性。数据库管理员可以确保访问数据库的唯一方式是通过正确的通道，因此可以定义安全性约束或规则。当试图访问敏感数据时，要检查这些安全性约束或规则。对于数据库的每条信息的不同类型的访问（修改、插入或删除等）可建立不同的约束。注意，没有这样的约束，数据的安全性可能比传统的文件系统更处于危险之中，也就是说，某种意义上数据库系统的集中性要求相称的、好的安全系统。

7) 平衡相互冲突的请求。数据库管理员了解企业的全局的需要，在他的指示下能建立系统的结构以提供对企业最佳的全局服务。例如，所选择的数据的物理表示应尽可能使重要的应用以最快的方式访问数据（可能会以降低其他某些应用的访问速度为代价）。

8) 加强标准化。数据库管理员对数据库集中控制（在数据管理员的指示下），可以确保所有表示数据的可用标准都可以观察到。可用标准可包括下面的任意一种或全部：部门标准、安装标准、社团标准、工业标准、国家标准和国际标准。标准化的数据表示可以很有效地支持数据交换或者两个系统间的数据移动（随着分布式系统的出现，这一点就越来越重要——见第2章和第20章）。同时，数据命名和文档标准也有效地支持了数据共享和易理解性。

以上列出的大多数优点都是比较显而易见的。但是，有一点则不然，这就是，对数据的独立性的支持（严格地说，数据独立性是数据库系统的客观目标，而不仅仅是一个必要的优点）。数据独立性的概念十分重要，以下专辟一节来深入讨论。

1.5 数据独立性

数据独立性包括两个方面：物理独立性和逻辑独立性 [1.3~1.4]。首先讨论数据的物理独立性。在未进一步说明之前，“数据独立性”应该理解为数据的物理独立性。我们将在第2章、第3章，尤其是第9章中讨论数据的逻辑独立性。注意，应该说“数据独立性”一词用的不是很恰当（起码它没有抓住问题的本质）；但是，由于传统上一直这么用，本书中仍采用该术语。

要理解数据独立性的含义，最好的方法是搞清什么是非数据独立的。在旧的系统中——关系系统之前的和数据库系统之前的系统——实现的应用程序常常是数据依赖的。这也就意味着，在二级存储中，数据的物理表示方式和有关的存取技术都是应用设计中要考虑的，而且，有关物理表示的知识和访问技术直接体现在应用程序的代码中。

- 例子：假定有一个应用程序使用了图1-5中的雇员文件，还假定文件在雇员姓名字段进行索引。在旧的系统中，该应用程序肯定知道存在索引，也知道记录顺序是根据索引定的，应用程序的内部结构是基于这些知识而设计的。特别地，各种数据访问的准确形式和应用程序的异常检验程序都很大程度上依赖于数据管理软件提供给应用程序的接口细节。

我们称这个例子中的应用程序是数据依赖的，因为一旦改变数据的物理表示就会对应用程序产生非常强的影响。例如，用哈希算法来对例子重建索引后，对应用程序不做大的修改是不可能的。而且，这种情况下应用程序修改的部分恰恰是与数据管理软件密切联系的部分。这其中的困难与应用程序最初所要解决的问题毫不相关，而是由数据管理接口的特点所引起的。

数据库系统中，应尽可能避免应用程序依赖于数据的情况。这至少有以下两条原因：

1) 不同的应用程序对相同的数据会从不同角度来看。例如，假定在企业建立统一的数据库之前有两个应用程序A和B。每一个都拥有包括客户余额的专有文件。假定A是以十进制存储的，而B是以二进制存储的。这时有可能要消除冗余，并把两文件统一起来。条件是DBMS可以而且能够执行以下必要的转换，即存储格式（可能是十进制或二进制或者其他的）和每个应用程序所采用的格式之间的转换。例如，如果决定以十进制存储数据，每次对B的访问都要转换成二进制。

这是个非常细小的例子，数据库系统中应用程序所看到的数据和物理存储的数据之间可能是不同类型的。本节后面部分还会考虑其他许多可能的不同情况。

2) DBA必须有权改变物理表示和访问技术以适应变化的需要，而不必改变现有的应用程序。例如，新类型的数据可能加入到数据库中；有可能采纳新的标准；应用程序的优先级（因此相关的执行需求）可能改变；系统要添加新的存储设备，等等。如果应用程序是数据依赖的，这些改变会要求程序做相应的改变，这种维护的代价无异于创建一个新的应用。类似的情况甚至在今天都并不少见，如典型的Y2K问题，这对充分利用稀缺宝贵的资源是极其不利的。

总之，数据独立性的提出主要是数据库系统的客观要求。数据独立性可以定义成应用程序不会因物理表示和访问技术的改变而改变。当然，这意味着应用程序不应依赖于任何特定的物理表示和访问技术。在第2章中，描述了支持以上基本要求的数据库系统的结构。在此之前，我们还是先讨论一下发生改变的具体情况，即DBA通常都有哪些改变上的要求，进而使应用程序尽量免受这方面的影响。

首先给出三个术语：存储字段、存储记录和存储文件（见图1-7）。

- 存储字段，简言之，就是存储数据的最小单位。数据库中对每一种类型的存储字段都包含许多具体值（或实例）。例如，包含不同类型零件信息的数据库可能包括称为零件数目的存储字段类型，那么对每种零件（如螺丝钉、铰链、盖子等），即有一个该存储字段的具体值。

  注意：实际中，通常不再明确指出是类型还是值，而是依据上下文来确定其含义。尽管有可能带来一些混淆，但实际中是方便的，本书中仍不时地采用这种方式（以上说明也同时适用于存储记录——见下一段要讨论的内容）。
- 存储记录是相关的存储字段的集合。我们仍区分类型与值。一条存储记录的值由一组相关的存储字段的值组成。例如，在零件数据库中的一条存储记录的值可能由下列每个存储字段的值组成。这些

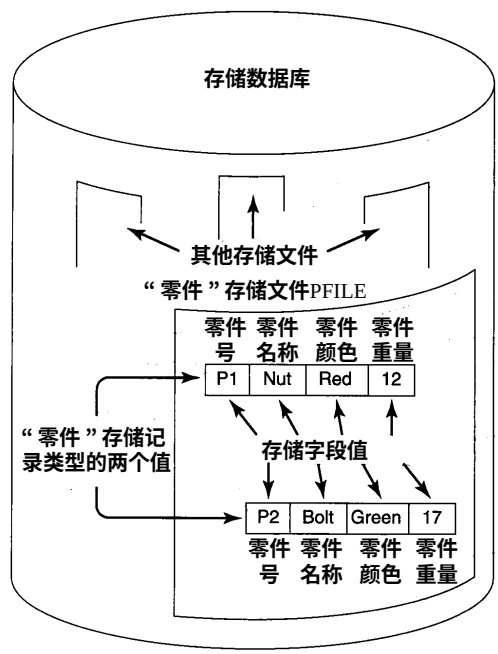

图1-7 存储字段、记录和文件

字段包括：零件号、零件名称、零件颜色和零件重量。而数据库是由零件存储记录类型的许多值（每种零件一个值）组成。

- 最后，存储文件是由现存的一种类型的存储记录的值组成。注意：为简单起见，假定任一存储文件值包含一种类型的存储记录。这种简化并不影响下面的论述。

现在，在非数据库系统中，常常是应用程序所处理的任一逻辑记录都是和相应的存储记录相同的。然而，我们已经看到，在数据库系统中这种情况是不必要的，因为 DBA可能需要对存储的数据表示进行改变 —— 即对存储字段、存储记录和存储文件 —— 可是从应用程序的角度看数据还是不变的。例如，在 EMPLOYEE文件中的DEPARTMENT字段可能为了节省存储空间而存成二进制格式，而一指定的 COBOL应用程序可能把它作为字符串来处理。随后DBA会由于某种原因改变数据的存储形式，即从二进制改为十进制，但对 COBOL应用程序而言，仍以字符串的形式对待。

如前所述，像这种在一次访问中出现某字段的数据类型变化的情况相对比较少。但是，应用程序中的数据和实际所存储的数据之间的差异会相当大。为明确说明这一点，我们给出下列可能要改变的各种存储表示。每种情况都应该考虑 DBMS应怎么做才能使应用程序保持不变（即是否可以达到数据独立性）。

- 数字数据的表示。一个数字字段可能存成内部算术形式，或作为一个字符串。对每一种方式，DBA必须选择恰当的数制（例如，二进制或十进制）、范围（固定的或浮点）、方式（实数或复数），以及精度（小数的位数）。其中任一方面都可能需要改变以提高执行效率或符合某一新标准，或因其他原因而改变。
- 字符数据的表示。一个字符串字段可能使用了几个不同编码的字符集中的一种 —— 如ASCII、EBCDIC和Unicode。
- 数字数据的单位。数字字段的单位会改变 —— 例如，在实施公制度量的处理中从英寸转成厘米。
- 数据编码。有时以编码的形式来表示物理存储的数据是非常好的。例如，零件颜色字段，在应用程序中看作字符串（“红”、“蓝”或“绿”），存储时可以存成单个十进制数字，可根据1=“红”，2=“蓝”，如此等等这样的编码模式来解释。
- 数据具体化。实际中由应用程序所看到的逻辑字段经常与特定的存储字段相联系（尽管它们会在数据类型、编码等方面都不相同）。在这种情况下，数据实例化的处理 —— 也就是，从相应的存储字段的值构建逻辑字段的值，并提供给应用程序 —— 可以说是直截了当的。但是，有时，一个逻辑字段可能没有对应的存储值，它的值可以通过根据一些存储字段的值进行计算来具体化。例如，逻辑字段“总量”的值可以通过对各个单个存储数量来汇总而得到。这里的总量是个虚字段的例子，具体化的过程是间接的。注意，用户也会看到实字段与虚字段的不同，因为虚字段的值是不能更新的（至少，不能直接地更新）。
- 存储记录的结构。两个存储记录可以合成一个。例如，存储记录

零件号	零件颜色

和

零件号	零件重量

可以合成为

零件号	零件颜色	零件重量

的形式。当把现有的应用程序集成到数据库系统时会经常发生这种改变。这也暗示应用程序的逻辑记录由相应存储记录的恰当子集组成——也就是说，存储记录中某些字段对应用程序来说是不可见的。另一种情况是单个的存储记录被分成两个。将前面的例子反过来，存储记录

零件号	零件颜色	零件重量

可能被分裂成

零件号	零件颜色

和

零件号	零件重量

例如，这种分裂允许较少使用的原始记录部分存储在一个慢速设备上。这意味着应用程序的逻辑记录可由来自几个不同存储记录的字段组成——即，可能是某指定的存储记录的超集。

- 存储文件的结构。一指定的存储文件可以各种方式实现其存储。例如，它可以完全存储在单个的存储设备上（如，单个磁盘），或者存储在几个设备（可能在几个不同设备类型）上；可能根据一些存储字段的值按一定物理顺序来存储，或无序存储；存储顺序可能按某一种或某几种方式进行，例如，通过一个或多个索引，一个或多个嵌入的指针链，或者两者兼有；通过哈希算法可能访问到也可能访问不到；存储记录可能物理上被分块，也可能没有；如此等等。但是上述任何考虑都不会以任何方式影响应用程序（当然性能除外）。

以上所列基本概括了可能的存储数据形式的改变。这意味着数据库应该能够增长而并不削弱现存的应用程序的功能；的确，在保证数据库增长的同时而不削弱应用程序的功能，是提出数据独立性的重要原因之一。例如，必须有办法通过增加新的存储字段来扩展现有的存储记录，如对现存的实体类型的进一步追加信息（如，“单价”字段可能会加入到零件的存储记录中）。这样新的字段对原应用程序来说应是不可见的。同时，还可能增加全新的存储记录类型（这样就有新的存储文件），而这不会引起应用程序的改变。这样的记录代表新的实体类型（如，一个“供应商”记录类型可以加到“零件”的数据库中）。而且这些增加对应用程序来说也应是不可见的。

至此，大家应清楚把数据模型从其实现中分离出来的原因之一就是数据独立性的要求，就像在1.3节末尾所指出的那样。某种程度上，不做这种分离，就得不到数据的独立性。在目前不能正确做到这种分离的情况很严重，尤其是当今的SQL系统都做不到这一点，实在让人沮丧。注意，这并不意味着当今的SQL系统根本不支持数据的独立性，只是它们提供的要远远少于关系系统理论上要求达到的。换句话说，数据独立性不是绝对的（不同系统可提供不同程度的数据独立性，很少有系统根本不提供的）；SQL系统提供得要比其他的系统多一些，但还不是很好，这在后续章节中将会看到。

1.6 关系系统及其他

正如在1.3节末尾所提到的，基于关系数据模型（关系系统）的数据库管理系统在数据库市场上已经占据了主导地位。而且，过去30年中大量主要的数据库研究是基于此模型的。实际上，不可否认，1969~1970年间关系模型的建立，在整个数据库领域的历史中无疑是最重要的事件。由于这些原因，加上关系模型有坚实的逻辑和数学基础，使之因此成为数据库原理

的主要教学内容，本书的重点(如1.1所指出的)也主要针对关系系统。

那么究竟什么是关系系统呢？很显然本书在此还不能给一个完满的解答，但是可以先给出一个大致的定义，之后再给出更详尽的答案。简言之，关系系统是指：

1) 数据以表（而且只有表）的形式呈现给用户；且

2) 提供给用户的操作（如，检索）以表为操作对象，即操作是从旧表中生成新的表。例如，“选择”操作，是提取某指定的表的行子集，而“投影”操作是提取一个表的某些列的子集，表的行子集和列子集本身都可以看作一个表。

注意：这样的系统之所以称为“关系的”，是因为表的数学用语为关系；当然，关系和表这两个词至少在非正式的情况可以当作同义词（见第 3章和第5章进一步的讨论）。这里应指明以下对原因的解释是不成立的，即因为实体联系图中联系的数学用语为关系；实际上，在关系系统与这些图之间也没有多少直接的联系。参见第 13章。

这里重复一下，我们会在后面更详细地阐述这些定义，但这里只是开个头。图 1-8提供了一个例子。图中a部分的数据包含一个名为CELLAR的表（实际上，它是图1-1中的CELLAR表的一个简化版本，规模减小是为了便于管理）。图中b部分是两个查找的例子，一个包含选择或行子集操作，另一个包含投影或列子集操作。注意：两个查找都是用 SQL 来实现的。

现在我们可以这样区分关系系统和非关系系统。如前所述，关系系统的用户把数据看作表，而且只能是表。非关系系统的用户则把数据看作其他的数据结构，代替或者扩展关系系统中的表结构。这些结构需要相应的操作。例如，像IBM的 IMS这样的层次系统，展现给用户的数据是树结构（层次）的集合的形式，对这些结构的操作符包括对遍历指针的——即表示整个层次路径的指针操作符（与此对比，这是与关系系统相区分的重要特征，关系系统没有这样的指针）。

a. 给定表 CELLAR

WINE	YEAR	BOTTLES
Chardonnay	1996	4
Fumé Blanc	1996	2
Pinot Noir	1993	3
Zinfandel	1994	9

b. 操作符(举例)

1. 选择 结果

```
SELECT WINE, YEAR, BOTTLES
FROM   CELLAR
WHERE  YEAR > 1995 ;
```

WINE	YEAR	BOTTLES
Chardonnay	1996	4
Fumé Blanc	1996	2

2. 投影 结果

```
SELECT WINE, BOTTLES
FROM   CELLAR ;
```

WINE	BOTTLES
Chardonnay	4
Fumé Blanc	2
Pinot Noir	3
Zinfandel	9

图1-8 关系系统的数据结构和操作符（例子）

进一步说，根据数据结构和提供给用户的操作符，数据库系统实际上能够很方便地加以分类。根据这一点，以往系统（非关系的）可分为三大类，即倒排表、层次的和网状的系统[⊖]。本书中我们不详细讨论这些系统，因为至少从技术角度上已经是过时了（如果有兴趣，可参见 [1.5]，其中有对三种系统详细的描述）。但是，应当指出这里的网状(network)与通信上的网络(network)毫无联系，它指的是系统所支持的数据结构和操作符。

注意：网状系统又可称为CODASYL 系统或 DBTG系统，这是因为其核心是由数据库系统语言协会（CODASYL）下属的数据库任务组（DBTG）所提出的。可能这一系统的典型代

⊖ 与关系模型类似，本书的早期版本引用倒排表、层次的和网状的模型（许多文献也这样），但使用这些词语会误导读者，因为和关系模型不同，倒排表、层次的和网状的“模型”都是根据事实定义的；即商业的倒排表、层次的和网状系统的产品首先实现，而相应的“模型”是通过从那些现存的实际系统中归纳得出的定义。

表是Computer Associates International 公司的IDMS。和层次系统一样（与关系系统不同），这些系统都把指针提供给用户。

第一代关系产品出现于20世纪70年代末80年代初。在写本书之际，绝大多数数据库系统都是关系系统。它们可以在各种硬件和软件的平台上运行。最主要的产品（以字母顺序排列）包括：IBM 公司的DB2；Computer Associates International公司的 Ingres II ；Informix Software 公司 的 Informix Dynamic Server；微软公司的Microsoft SQL Server ；Oracle公司 的 Oracle 8i ；Sybase 公司的 Sybase Adaptive Server 。注意：本书中以后涉及到这些产品时，我们分别只提及缩写名字DB2、Ingres、Informix、SQL Server、Oracle、Sybase，等。

最近，一些对象和对象/关系型数据库产品推向市场[⊖]。对象关系系统表示（对大部分）某些初始的关系型数据库产品向上兼容的扩展，如 DB2和Informix；对象系统（面向对象）表示试图做一些彻底的改变，如 GemStone Systems公司的GemStone 和 Versant Object Technology 公司的 Versant ODBMS。我们将会在本书第六部分讨论这些新的系统。

除以上谈到的数据模型方法以外，近几年的研究还提出了一些新的方法，包括多维的方法和基于逻辑（也称演绎或专家）的方法。我们将在第 21章讨论多维系统，在第 23章讨论基于逻辑的系统。

1.7 小结

现在概括本章所讨论的主要问题。首先，数据库系统可以被看作计算机化的存储记录的系统。该系统包括数据（存储在数据库中）、硬件、软件（尤其是数据库管理系统）和 —— 最重要的 —— 用户。用户又可分成应用程序员、最终用户和数据库管理员 (DBA)。DBA负责根据数据管理员制定的政策来管理数据库和数据库系统。

数据库是集成和共享的；它们用于存储持久数据。这些数据通常可以表示为实体及实体间的联系 —— 尽管实际上联系只不过是一种特殊实体。我们简要介绍了实体/联系图。

数据库系统有许多优点。其中最重要的一点就是数据独立性。数据独立性的定义是指能使应用程序免于随着数据物理存储和访问方式的变化而变化。另外，数据独立性要求数据模型和它的实现分开（提醒大家关注数据模型一词，它可能有两种截然不同的意义）。

数据库系统通常支持事务，即工作的逻辑单位。事务的优点之一是，即便在事务执行中系统出现故障，也能确保事务原子性(即要么全做，要么全不做)。

最后，数据库系统以许多不同的方法为基础。特别地，关系系统以称为关系模型的形式化理论为基础。在关系模型中，数据表示为表中的行（解释为真命题），所提供的操作直接支持从已指定的真命题推断出其他真命题的处理过程。从经济的和理论的前景来看，关系系统很显然是最重要的（这种情况在可预见的将来是不可能改变的）。我们已经给出了几个SQL的例子（特别地，例如SQL的SELECT、INSERT、UPDATE和DELETE语句），SQL是关系系统的标准语言。本书将着重于关系系统的讨论，对 SQL则论述不多。

练习

1.1 定义下列术语

⊖ 这里对象一词具有特定含义，这一含义将在第六部分解释。在那之前，仍使用该词的通常含义，以免句子的含义混淆。

二元联系	完整性
命令驱动界面	菜单驱动界面
并发访问	多用户系统
数据管理	联机应用程序
数据库	持久数据
数据库系统	属性
数据独立性	查询语言
数据库管理系统	冗余
数据库管理员	安全性
实体	共享
实体/联系图	存储字段
表格驱动界面	存储文件
集成	事务

1.2 使用数据库系统的优点是什么？

1.3 使用数据库系统的缺点是什么？

1.4 什么是关系系统？区分关系和非关系系统。

1.5 什么是数据模型？解释数据模型与其实现的差别。为什么这个差别很重要？

1.6 给出下列对图1-1中CELLAR数据库的SQL检索操作的结果：

a.
```
SELECT WINE, PRODUCER
FROM   CELLAR
WHERE  BIN# = 72 ;
```

b.
```
SELECT WINE, PRODUCER
FROM   CELLAR
WHERE  YEAR > 1996 ;
```

c.
```
SELECT BIN#, WINE, YEAR
FROM   CELLAR
WHERE  READY < 1999 ;
```

d.
```
SELECT WINE, BIN#, YEAR
FROM   CELLAR
WHERE  PRODUCER = 'Robt. Mondavi'
AND    BOTTLES > 6 ;
```

1.7 从练习1.6的每个答案中选出一行，用自己的话解释成真命题。

1.8 给出下列对图1-1中酒窖数据库的SQL更新操作的结果：

a.
```
INSERT
INTO   CELLAR ( BIN#, WINE, PRODUCER, YEAR, BOTTLES, READY )
VALUES ( 80, 'Syrah', 'Meridian', 1994, 12, 1999 ) ;
```

b.
```
DELETE
FROM   CELLAR
WHERE  READY > 2000 ;
```

c.
```
UPDATE CELLAR
SET    BOTTLES = 5
WHERE  BIN# = 50 ;
```

d.
```
UPDATE CELLAR
SET    BOTTLES = BOTTLES + 2
WHERE  BIN# = 50 ;
```

1.9 写出对酒窖数据库执行下列操作的SQL语句：

(a) 找出所有Geyser Peak酒的bin号、酒的名字和瓶数。

(b) 找出存储量超过5瓶的酒的bin号和酒的名字。

(c) 找出所有红酒的bin号。

(d) 对bin号为30的加3瓶酒。

(e) 从库存中删除所有Chardonnay。

(f) 加入一条新记录：12瓶Gary Farrell Merlot：bin号55，1996年生产，2001年出厂。

1.10 假定有一些古典音乐的CD和/或LP和/或磁带，并且想要建立一个数据库来查找某作曲家（如：Sibelius），或某指挥家（如：Simon Rattle），或某独唱家(如：Arthur Grumiaux)，或某作品（如：贝多芬的第5交响曲），或某管弦乐队（the NYPO），或某种作品（如：violin concerto）或室内乐乐队(如：Kronos Quartet)的有关资料，为此数据库画出一个像图1-6那样的实体/联系图。

参考文献和简介

1.1 E.F.Codd :“ Data Models in Database Management，” Proc.Workshop on Data Abstraction，Database，and Conceptual Modelling， Pingree Park，Colo.(June 1980);*ACM SIGART Newsletter* No.74 (January 1981);*ACM SIGMOD Record 11*，No.2(February 1981);ACM *SIGPLAN Notice* 16，No.1(January 1981)。

Codd是关系模型的创始人，在参考文献[5.1]中，他首次描述了该模型。但是，并没有定义数据模型这个概念——不过现在的论文中包括了。提出的问题是：数据模型，尤其是关系模型，通常的目的是什么？想要怎样？并对这个声明继续提供证明，关系模型实际上是第一个定义的数据模型（换句话说，Codd声称是数据模型概念的创始人，同时也是关系数据模型的创始人）。

1.2 Hugh Darwen:“ What a Database Really Is: Predicates and Propositions，” in C.J.Date，Hugh Darwen，and David McGoveran，*Relational Database Writings 1994-1997*. Reading，Mass.: Addison-Wesley (1998).

在1.3节的末尾简要提到过，数据库最好被看作真命题的集合，本书给出了这一观点的非正式(但十分精确)的解释。

1.3 C.J.Date and P.Hopewell:“Storage Structure and Physical Data Independence，” Proc.1971 ACM SIGFIDET Workshop on Data Definition ，Access，and Control，San Diego，California (November 1971).

1.4 C.J.Date and P.Hopewell:“ File Definition and Logical Data Independence，” Proc.1971 ACM SIGFIDET Workshop on Data Definition ，Access，and Control，San Diego，California (November 1971).

参考文献[1.3~1.4]第一次定义和区分了数据的物理独立性和逻辑独立性。

1.5 C.J.Date: *Relational Database Writings 1991-1994*，Reading，Mass.: Addison-Wesley(1995).

部分练习答案

1.3 一些缺点如下：

- 折衷的安全性（没有很好控制）；
- 折衷的完整性（没有很好控制）；
- 需要额外的硬件；
- 执行开销非常大；
- 成功的操作至关重要（企业可能受攻击而操作失败）；
- 系统可能很复杂（尽管这些复杂性对用户是不可见的）。

1.6 a.

WINE	PRODUCER
Zinfandel	Rafanelli

b.

WINE	PRODUCER
Chardonnay	Buena Vista
Chardonnay	Geyser Peak
Joh. Riesling	Jekel
Fumé Blanc	Ch. St. Jean
Gewurztraminer	Ch. St. Jean

c.

BIN#	WINE	YEAR
6	Chardonnay	1996
22	Fumé Blanc	1996
52	Pinot Noir	1995

d.

WINE	BIN#	YEAR
Cab. Sauvignon	48	1993

1.7 只给出a部分的答案“Rafanelli是Zinfandel的一个生产商”，或更精确地说，“某个号包含一些Zinfandel酒，该酒是由Rafanelli在某年生产的，将在某年饮用”

1.8 (a) 在CELLAR表中加入Bin#为80的行。

(b) Bin#为45、48、64和72的行从CELLAR表中删除。

(c) Bin#为50的行的瓶数置为5。

(d) 和c相同。

顺便提一下，通过主码值找到一行是很方便的(CELLAR表的主码是{BIN#}——见第8章)。

1.9 (a)
```
SELECT BIN#WINE, BOTTLES
    FROM CELLAR
    WHERE PRODUCER='Geyser Peak';
```

(b)
```
SELECT BIN#WINE
    FROM CELLAR
    WHERE BOTTTLES>5;
```

(c)
```
SELECT BIN#
    FROM CELLAR
    WHERE WINE='Cab.Sauvignon'
    OR WINE='Pinot Noir'
    OR WINE='Zinfandel'
    OR WINE='Syrah'
    OR ......;
```

本题没有更简洁的答案，因为“酒的颜色”在数据库中并没有明确记录；因而，DBMS不知道(例如) Pinot Noir是红色的。

(d)
```
UPDATE CELLAR
    SET BOTTLES=BOTTLES+3
    WHERE BIN#=30;
```

(e)
```
DELETE
    FROM CELLAR
    WHERE WINE='Chardonay';
```

(f)
```
INSERT
    INTO CELLAR(BIN#,WINE,PRODUCER,YEAR,BOTTLES,READY)
    VALUES(55,'Merlot','Gary Farrell','1996,12,2001);
```