数据库系统讲义

苏州电子计算机厂

前　言

近年来，各种计算机的数量在国内猛增，特别是微型计算机。在企事业管理和办公室自动化的应用中，数据库管理系统，特别是关系数据库管理系统软件，dBASE Ⅱ、dBASE Ⅲ INGRES、INFORMIX等，由于它们的实用性和操作方便已为许多用户所乐于使用。

无论用过还是没有用过数据库管理系统的用户，因随机资料和适应范围的原因，都很希望有介绍数据库系统，特别是介绍关系数据库系统概念和原理的书或讲义。

1984年在苏州电子计算机厂讲数据库系统课，承蒙厂情报资料室的大力支持，决定付印这本《数据库系统讲义》。

这本讲义主要介绍关系数据库系统的概念和使用方法。讲义分两大部分：第一部分由三章组成，叙述数据库系统的基本概念和总体结构；第二部分为十一章，以一个典型的关系数据库管理系统——系统R为背景，详细地介绍了关系方法的各种功能。每章之后都有小结、练习和参考文献供复习、思考和进一步学习之用。

本讲义主要取材于C·J·DATE的《数据库系统引论》第三版一卷。

C·J·DATE是美国San Jose城（加州），IBM公司一般产品部门的顾问工程师。他从1967年开始一直从事培训和教学工作，1970年以来主要活跃在数据库领域内，在美国和其他国家，讲演过许多数据库的论题——特别是关系数据库。他的书无论是实用性还是系统性都有参考价值，许多国家都以这本书为数据库系统的教学参考书。1983年初该书第二卷已出版，更进一步讲述了数据库管理系统的许多具体内容，例如分布式数据库、恢复等等。

讲义的编辑和付印过程中的工作是由苏州电子计算机厂情报资料室张盛森和其他有关同志完成的，在此表示衷心感谢。

讲义中不妥之处，存在的问题希望读者批评指正。

编者1985.2.4

目　录

第一章　基本概念

1.1 什么是数据库系统

数据库技术号称为“计算机和信息科学领域中成长最快的科技之一”，作为一个领域，它算是相当年轻的，因为直到六十年代才有厂商开始提供数据库管理系统的产品（尽管，早期的某些软件包已包含了现在系统的某些功能）。尽管年轻，但它已在理论和实践上迅速地发展起来。现在交给数据库处理的数据总量，保守点说也有几十亿字节（byte）。诸如财政上的投资，可用相当庞大的指数来表示。毫不夸大，现在已有上千种机构正在完全依赖于数据库系统的正常连续运行。

那么，什么是数据库系统呢？基本上说来，**它不过是以计算机为主的记录保存系统**。就是说，**数据库系统的整个目的是记录和维护信息**。〔数据或资料(Data)和信息(information)在本书中被看作同义词。有些作者是区别这两个术语的：数据是指实际记录在数据库中的值，信息则指的是某用户所理解的值的意义。这种区别是重要的，但我们认为明确指明其区别要比用两个本质上相似的术语来区分似乎更好些〕。数据库系统所涉及的信息，对它服务的机构来说，可以是任何有意义的东西，即在该机构里组织管理工作中作决策所必须的任何东西。图1.1是数据库系统的最简单的轮廓。

图1.1表明了数据库系统包括四个主要成分：**数据、硬件、软件和用户**。下面简单介绍这四种成分。后面再详细讨论它们。

数据

存储在系统中的数据可划分成一个或多个数据库。为了便于说明，我们假设只有一个数据库，其中包含了系统中全部存储的数据。因为这样的简化假定，并不影响以后的讨论。正如我们将在以后看到的，有许多好的理由使得在实际上不必施加这一限制。

一个数据库可以说是存储数据的仓库。一般说来，这个仓库被看作是**整体的**和**共享的**（integrated and shared）。

整体的。意味着数据库可以认为是若干不同数据**文件的单一化**，在单一化过程中部分地或完全地消除**了文件中的数据冗余**。例如，一个数据库包含EMPLOYEE（雇员）记录和ENROLLMENT（注册）记录。前者包括姓名、所属部门、住址、工资等项目，后者则表示雇员参加培训某课程的注册。假设为了实行课程管理，需要知道每个已注册的学生的所属部门，那么显然不必把所属部门这一重复数据项包括在ENROLLEMET记录之中。因为只要找到对应员工的EMPLOYEE记录就可以查到该雇员的所属部门。

共享的。意味着数据库中的各个数据段，可被若干不同的用户共享，不同的用户可存取数据库的同一数据部分用于不同的目的。实质上数据库的共用性是整体性这一事实的结果。在上面的例子中（EMPLOYEE／EMROLLMENT），EMPLOYEE记录中的所属部门这一项目，就可视为人事部门和教育部门的用户共享。数据库是整体的这一事实的另一个结果，是每个用户都只涉及整个数据库的某些子集，而且，这些不同

用户的子集将会以许多不同的方式重迭。换句话说，一个给定的数据库可为不同的用户以各自的不同用法所理介。（甚至两个用户共享数据库的同一子集时，在对这部分数据细节的看法上也可能有相当的差异。

共享的含义除上面所说的以外，还可推广为并发共用（Concurrent Sharing）：若干不同的用户在同一时刻实际存取数据库同一个数据段的能力。（支持这种共享形式的数据系统，有时称为多用户系统）。

硬件（hardware）

硬件包括存储数据库的辅助存储器——磁盘、磁鼓以及相关的设备和控制单元、通道等等。（这里假设数据库很大，无法全部放在主存储器中）。基于下面几点理由，本书不涉及系统的硬件方面。首先，硬件本身是一个大的课题；第二，数据库系统所涉及的硬件问题并不是数据库所特有的；第三，有关问题已经有了很完整的探讨和文献可供参考。

软件

软件是指介于实际存储的数据本身（物理数据库）和系统的用户之间的界面（一组程序）通常称为数据库管理系统DBMS（database management system）。所有来自用户的，对数据库进行存取的要求，都由DBMS来处理。因此DBMS的一个一般性功能是把**数据库用户和硬件的细节隔开**。（就好象程序设计语言，例如COBOL，把程序员和硬件细节隔开一样）。换句话说，DBMS把对数据库的看法，提升到了硬件之上，并提供给用户一些高水准表达演算的操作。（例如**"取出员工Smith的EMPLOYEE记录"**）。有关DBMS的这些功能和其他一些功能，将在后面详细讨论。

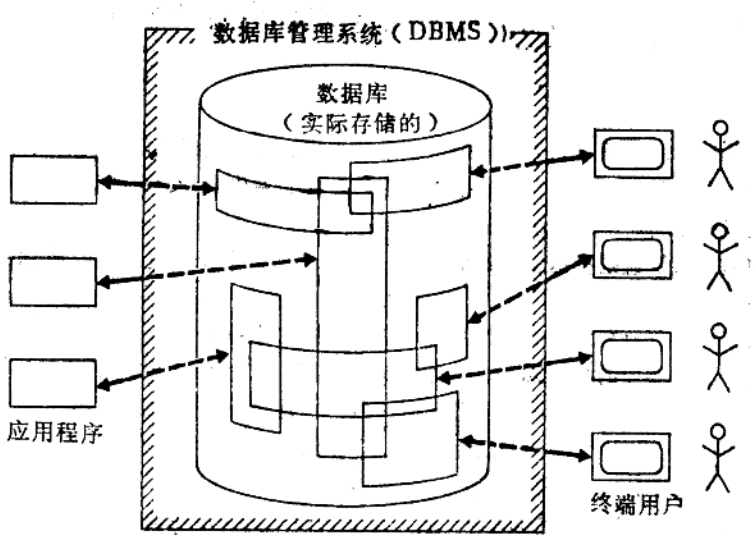

图1.1 数据库系统的简单轮廓

用户

考虑三大类用户。第一类是应用**程序员**（application programmer），负责编写使用数据**库的应用程序**，通常使用像COBOL，PL／1这类语言。写出的程序将按通常的方

式对数据进行操作：检索信息、生成新的信息、删除或变更已存的信息。（所有这些功能，都通过对DBMS发出适当要求来完成）。应用程序本身可以是通常的批量处理程序，也可以是供终端用户和系统进行对话的联机程序。

第二类用户是终端用户（end—user），从终端上存取数据。他们可以使用作为系统一个部分的查询语言（guery langnge），或者调用应用程序员写的应用程序，由其从终端上接受命令，根据命令的要求，向DBMS发出相应的要求。无论这两种方式的哪一种，一般说来，都能完成检索、生成、删除和修改的功能。不过检索是这类用户最常用的功能。

第三类用户是数据库管理员（data base administrator），或称DBA（图1.1上没有画出）。关于DBA的作用放到1.5专门讨论。

上面是数据库系统主要方面的一个完整概念。本章余下部分将稍详细地讨论它们。

1.2 经营用数据

早期，Engles出于明确起见把数据库中的数据称为"经营用数据"（Qperational data）以区别输入数据、输出数据和其他类型的数据。下面是修正Engles的定义后的数据库定义。

数据库是某特定企业所使用的存储了的经营用数据的集合。

这个定义要稍作解释。企业（enterprise）是一个总称，泛指传统上，任何由合理方式组成且自成体系的商业、科学、技术或其他方面的组织。例如：

制造公司
银行
医院
大学
政府部门

任何企业都必须维护一大堆为企业经营用的数据。这就是"经营用数据"。上列企业经营用的数据可能有如下几类：

产品数据
会计数据
病历数据
学生数据
计划数据

正如上面讲到的，经营用数据不包括输入、输出、工作队列及任何临时用数据。输入数据是指从外面（典型的例子是从卡片或终端上）输入的数据；这样的数据经过变换可以成为经营用数据，但它们本身并不是数据库的一部分。类似地，输出数据是指系统发出的信息或报告（打印或在终端上显示）。这类数据可以从经营用数据得出来，但它们本身并不是数据

库的一部分。

为了解释经营用数据的概念，我们拿制造公司作例子。这个公司要用到的数据为："计划"（projects）；该生产计划中要用到的"另件"（parts）；供应这些另件的"厂商"（suppliers）；存储另件的"仓库"（warehouses）；参与这些计划的"员工"（employees）等等。这些基本实体（entity）的数据要记录到数据库中去。（实体这个术语在数据库系统中是被广泛使用的，指要在数据库中表示的对象）见图1.2。

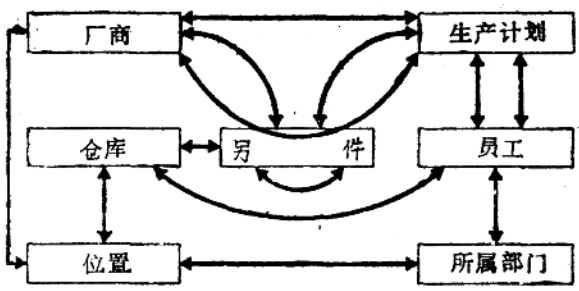

图1.2 经营用数据的例子

注意这样一点是重要的：一般说来，基本实体之间是有联系（associations）或关联（relationsInips）的。图1.2上这种关联用箭头表示。例如，厂商和另件之间的关联是，每个厂商供应某些另件，反过来每种另件为某些厂商所供应。类似地，另件用于生产计划，反过来生产计划又使用另件；另件存放在仓库中，仓库中存放着另件等等。要注意到这类关联都是双向的，即可以沿着两个方向来查询。例如对员工和所属部门之间的关联可用于回答下面两个问题：（a）给一个员工，求所属的部门；（b）给一个部门，求所有在该部门工作的员工。

如上所述（对照图1.2），很重要的一点是，如同关联着的实体一样，关联本身（关系）也是经营用数据的一部分。因此关联也就必须要在数据库中表示出来。后面我们将讨论表示关联的各种方法。

图1.2还说明了以下几点：

1、虽然图上的大多数关系都关联着两类实体，但并非都如此。图中就有一个箭头关联三个实体的情况（厂商——另件——计划）。它表明了这样的事实：某些厂商对某些生产计划供应另件。一般说来，这样的关联并不是"厂商——另件"和"另件——计划"的组合。例如"**厂商S2供应另件P4用于计划J3**"告诉我们的内容要比下面的两条关联的组合更多一些："**厂商S2供应另件P4**"和"**另件P4用于计划J3**"。我们知道后两条未必可以得出第一条。（但我们知道第一条，可推出后两条）。更明确地说，若已知S2供应P4和P4用于J3，那么可以得出S2供应P4用于计划Jx，同样可得出某厂商Sy供应P4用于J3。但是由此并不能推出Jx就是J3或Sy就是S2。像这类错误的推论有时被称为连接陷井（Connection trap）。

2、例子中的箭头仅涉及一类实体（另件）的关联。它表明了一个另件和另一个另件之间的关系：一种另件可以是另一种另件的部件例如螺丝钉是铰链的部件，螺丝钉是另件，而铰链也可以看作是一种另件。

3、一般说来，同样的实体之间还可以有若干种关联。图1.2中计划和员工之间就有两

种关联。一个关联表示“工作於”（员工分配于计划中工作），另一个关联可表示“是计划的负责人”（员工是这项计划的管理者）。

许多书（和系统）把实体（entity）和关联（relationship）看作两种基本的不同对象。但是实体之间的关联本身也可看作是一种实体。我们定义实体为：**“实体是希望记录信息的对象”**。显然关联也是符合实体这一定义的。例如“另件P8存放在仓库W8”中，是一个关联，我们可以把它看作是一个记录信息的实体，例如记录的信息为另件数量（另件P8在仓库W8中的数量）。本书的观点趋向于把关联看作特定的实体类型。

1.3 为什么要数据库？

为什么一个企业要把它的经营用数据存储到数据库中去呢？简单的回答是，数据库系统提供了一个企业可以对它的经营用数据进行集中控制。正如读者目前认识到的，企业的经营用数据对一个企业来说是最有价值的资产之一。目前典型使用经营用数据的状况是每个应用户都使用它自己的文件（自己掌握住的磁带或磁盘组）。经营用数据相当分散，因而，控制起来也就相当困难了。有了数据库之后，就和这种情况形成了一个明显的对照。

在数据库系统中，因为数据是集中控制的，因而也就必然要有一个相当适合的人来集中和管理经营用数据。这个人就是1.1节所讲的数据库管理员DBA。后面再讨论DBA的作用。这里仅强调两点：

这个人不仅要有高度的技术专长，而且要有能理解和解释来自上下层的对经营用数据的管理要求（事实上DBA不是一个人，而是一个班子）。这里强调一下DBA在一个企业中的地位是相当重要的。

现在让我们来看一下数据集中控制而带来的好处。

减少了重复数据

在非数据库系统里，每个应用户都有它自己的私有文件，这就常常导致了存储数据的很多重复，其结果是浪费了存储空间。例如，在人事应用和教学应用的文件中，就可能都包含着有关员工所属部门的资料。正如我们在1.1节所建议的，如果DBA知道这两个应用户的要求，那么这两个文件就可被整体化，以去掉重复。

并不是说，数据库里所有的重复数据都应该去掉。有时出于经营和技术的需要反而要维护同样数据的多份复本。但是，不管怎么说，数据库中的数据重复应该受到控制。就是说，系统应该知道这些重复，并且应该负责其传递更新。（传递更新的意思见下一段）。

避免（某种程度的）数据不一致性

这一点，实际上是前面一点的推论。假定有这样个一事实，比方说，员工E8在部门D8中工作，数据库中有了两个不同的登录，而系统又不知道这个重复（换句话说这个重复没有被控制），那么就会在某些情况下出现这两个登录项目的不一致（即仅更新了其中之一），这时，我们便说，数据库的数据是不一致的（inconsistent）。无疑，一个数据库处于不一致的状态，就有可能提供出错误的或相矛盾的资料。

显然，所给的事实，在数据库中若只登录在一处，便消除了重复，那么这种不一致性就不会出现。或者，假定不消除这样的重复，通过让系统知道的办法来加以控制，那么系统就可以保证数据库中的这种重复数据在用户看来是一致的。其方法是，无论对这两个登录项的

哪一个进行更新时，系统都自动地对另一个作同样的更新这就是传递更新（propagation updates）。更新一词，这里泛指生成、删除和修改操作。（注意，能做到传递更新的系统还很少，就是说，目前的大多数系统都没有控制重复数据的功能）。

数据可被共用

1.1节中，谈到这一点，这里再强调一下，是因为它很重要。它不仅意味着应用户可以共用数据库中的数据，而且对同一个数据库中的数据还可以开发新的应用户。换句话说，可以不必去生成任何新的存储文件就可以满足新应用户的数据要求。

可实行标准化

由于数据库中数据的集中控制，DBA就有可能保证在数据的表示上遵循所有可应用的标准。可应用的标准是指任何公司、设备、部门、工业、国际的标准。对于系统间的数据变更和迁移来说，存储数据的标准化是特别重要的。

可采用保密性限制

完成了整个经营用数据的裁决以后，DBA就可能（a）保证通过适当的通道，使得对数据库存取数据的方法唯一。这样一来也就可能（b）定义用户使用数据的权限，以检验存取权力之内的数据。对数据库中的数据片段的各种类型的存取（检索、修改、删除等等，）还可建立不同的可用性检查过程。注意，若没有对数据保密性及可用性的检验，数据库系统的存取情况，可能比通常的文件系统更不安全。

完整性（Integrity）可得到维护

完整性的问题是保证数据库中数据是否正确的问题。表示同一事实的两个登录项的不一致就是缺乏完整性的例子（这里当然是假定了数据库存在重复）。即使消除了重复数据，数据库中也仍然可以包含不正确的数据。例如，一个员厂的周工作日出现了200小时，一个部门的员工号表中，出现了不存在的员工号。数据的集中控制，在一定的范围内有助于避免这类情况。其方法是由DBA去定义一个检查过程，以测试任何数据更新是否在允许范围内。（这里的更新亦指：修改、生成和删除操作）。由于数据库数据可被共用的原因，在数据库中的数据的完整性可能比私用文件的环境更显得重要。因为如果没有适当的检查过程，由于程序的错误在数据库中生成了不良的数据，那么，应用该数据的其他程序就无辜的受到了影响。

互相冲突的要求可得到调和

一个企业的全体经营要求很可能和下面各个应用部门的要求发生冲突。这时DBA就有可能构筑出一个“对整个企业经营最佳的”数据库系统的服务来。例如，选择存储数据的表示时，可以选择，对最重要的应用户，存取速度可以很快，而对某些其他的应用，性能则不很佳的方案。

上面列出的优点都是相当明显的。还有一点则不那么明显(虽然它为上述优点所隐含)，那便是数据库系统可提供数据的独立性（Data independence）。严格说，这是数据库系统的目标不是优点。这一点如此重要，以致于值得我们用专门一节来叙述

1.4 数据独立性

要理解数据的独立性，最简单的方法是先考虑其反面。目前大多数它们都是数据相关的（data—dependent）。所谓数据相关，是指数据在辅存储器中的组织和存取方法完全取决于应用的要求。而且，这种数据组织和存取技术的知识也都基于应用程序的逻辑结构。例如由于效率的原因，它可能决定特定文件要以顺序索引的形式存储，其应用程序也就必须知道该文件的索引存在和文件的顺序是由索引定义的事实，这样应用程序的内部结构也就围绕着这些知识设计了。同时应用程序中的各种存取的精确格式和意外情况的测试过程等，势必要紧密地依赖于索引顺序文件组织的软件提供细节。

像上面这种应用，我们所以说它数据相关，是因为在变更存储结构（数据怎样被实际地存储）或存取策略（怎样存取数据）时，应用程序必然要作相应的变更，甚至是相当大的变更。例如，上面所讲的应用程序就不可能在不作重大修改的前提下，用杂凑寻址方法（hash—addressing）来代替原来的索引顺序文件。顺便提一下，非常有趣的是在上述情况下，应用程序要变更的部分都是与文件处理软件打交道的部分，这种变更的困难却与应用程序中解决问题的部分完全无关。由此可见**数据相关性完全是由于文件处理的软件界面的结构所造成的。**

在数据库系统中，极不希望应用程序对数据相关，其理由致少有下面二点。

1、对同样的数据，不同的应用会有不同的看法。例如，假设在企业导入整体化的数据库之前，有两个应用A和B，每个应用在自己的文件中都包含有字段“顾客余额”。在A的文件中以十进制记录这个字段值，B则以二进制记录。只要**DBMS**在选定的存储表示和用户希望的形式之间进行必要的转换，那么就可能把A和B的文件整体化以后存入数据库并消除重复。比方说，数据选定了以十进制形式存储于数据库中，那么每当应用B存取这个字段时，**DBMS**都进行十——二转换工作。

这是一个应用对数据的看法和数据在数据库中实际存储之间存在着差异的简单例子。许多其他可能的差异后面再讨论。

2、**DBA**必须在有变更要求时，方能自由地变更存储结构和存取策略（或两者同时）而不影响已存在的应用。例如，企业接受新的标准；应用户的优先权发生变更，利用新式存储设备等等。如果在这种情况下，应用是数据相关的，那么上述的变化就会导致相应的应用程序的变更，这是非常费事的程序设计工作，不然的话，程序员就可以把更多的精力用于新应用的开发。例如，一个大的计算站就将近要用25%的程序设计工作量来做这样的维护工作。显然这是在浪费极有价值的资源。

由此可见提供数据独立性是数据库系统的主要目标。我们可以定义**数据独立性为应用对存储结构和存取策略变更的免疫性**（immunity）。这意味着，应用程序与任何特定的存储结构和存取策略无关。1.5节将介绍数据库系统的总体结构，以提供达成该目标的基础。在此以前我们要较详细地考虑一下**DBA**可能作的变更类型（即应用程序要免疫的内容）。

首先给出一些定义。

存储字段（stored field）：是数据库中存储数据的最小命名单位。一般说来，数据库将包含许多不同类型的存储字段，每种类型的存储字段，都有许多现实值（occurrence instances）存储在数据库中。例如，数据库中包含着有关另件的资料，另件的资料中有称为“另件号”的存储字段。那么每个不同另件的另件号就是这个存储字段的现实值。（这里讲的另件一词实际上是指的另件类型而不是指个别的另件）。

存储记录（stored record）：是命名了的相关联的存储字段的集合。这里同样要区分类型和现实值的差别。一个存储记录的现实值是由一组相关的存储字段现实值组组成的。（这组现实值还表达出了各字段不同现实值之间的关联）。比方说，一个存储记录的现实值可由存储字段另件号、另件名、另件颜色、另件重量的每一个现实值组成。显然，这些存储字段现实值之间的关联，代表了某个特定的另件的特征。由于数据库中包含着多种另件的资料所以我们可以说，数据库中包含着另件这种存储记录类型的多个现实值。（这里再一次指出，每个不同另件有一个现实值）。在大多数系统中，存储记录现实值是对数据库进行存取的单位——由DBMS在一次存取中从数据库取出或存入的数据单位（见第二章）。

注意，资料中通常"类型"和"现实值"被省略，那么究竟指的是类型还是现实值，可以从上下文区别其意义。这样做，偶尔会产生混淆，不过在今后，为了简单叙述起见，有时仍要被省略。

存储文件（stored file）：它也是一个命名的单位，代表某种存储记录类型全部现实值的集合。（这里是为简单起见而略去了一个文件包含多种存储记录类型情况的介绍，但这种简略不影响今后的讨论）。

目前在大多数系统中，应用的逻辑记录和存储记录是相同的，而数据库系统不是这样。DBA可以对存储结构作变更——变更存储字段和存储记录——同时要求对应的逻辑字段和逻辑记录不作变更。例如上述文件的"另件重量"字段为了节省存储空间可以以二进制来存储其现实值。但COBOL的应用程序则可把它描述成字符行类型的PICTURE。这样的差别（在每次存取时必须由DBMS作数据类型转换）。一般说来，应用中所看到的数据和实际存储的数据之间可能相当不同。这个事实可用下面几段中有关存储结构变化的情况来说明。（更完整的说明可参考Engles的书）。在所列的每种情况下，读者可以考虑DBMS应该做些什么才能保护应用不受其变化的影响。这样的保护工作一定能做到吗？

数值数据的表示法

一个数值字段可以存储成内部算术运算的形式（即压缩十进制形式），或者是字符行形式。DBA还可选择其基底（十进制或二进制）、表示法（定点或浮点）、类型（实数或复数）和精确度（数字位数）。各点都可因改进性能、适应新标准或其他原因而变更。

字符数据的表示法

一个字符行字段可要求以若干种字符编码的一种来存储。（例如，EBCDIC，ASCII等等）。

数值数据的单位

在数值字段中的数值单位可以变更。例如，在测量结果的处理中，可能要求从英寸变为厘米。

数据代码

在某些情况下，可能要用代码来表示存储的数据。例如，对另件颜色字段，从应用的观点看来是字符行（'RED'（红）'BLUE,（蓝）或'GREEN'（绿）…），而存储时可以是一位十进制数字，解释为1 ='RED'，2 ='BLUE'等等。

数据的具体化

通常由应用所看到的逻辑字段对应着唯一的存储字段。（如我们上面谈到的，它们可能在数据类型、单位等方面有所差别）。在这种情况下，数值具体化的过程是，从对应存储字

段的现实值构作逻辑字段的现实值，并把它呈现给应用户。这个具体化可以说是直接的。然而有些情况，并不存在对应的存储字段；逻辑字段值的具体化要通过对若干存储字段的现实值的集合进行运算而得到。例如，像"总额"这样的逻辑字段的值可以通过累加各不同的数量值而具体化。"总额"是一个虚拟的字段对它的值的具体化过程可以说是间接的（注意，真实的虚拟字段其差别是不能直接地去生成或修正一个虚拟字段的现实值）。

存储记录的结构

两个已存在的存储记录类型可以结合成为一个存储记录。例如，记录型（另件号、颜色）和记录类型（另件号、重量）可以整体化为记录类型（另件号，颜色，重量）。在把建立数据库之前的应用加入到数据库系统中去时，常常会出现这种情况。这意味着，应用的逻辑记录可以是一个存储记录的子集。（即，存储记录中的某些字段对特定应用来说，就好像没有一样）。

反之，一个单一类型的记录也可被劈成两个记录类型。例如记录类型（另件号，颜色，重量）可分为另件号、颜色和另件号、重量两个记录类型。使用这种分隔的方法，就可以把不经常使用的数据存储到低速设备上去，这样，应用的逻辑记录就可能包含来自若干存储记录的字段。

存储文件的结构

给定的存储文件，可以通过各种各样的方法实际地存储到存储体中去。例如，它可以整个地存放在一个存储卷上（如一个磁盘组），也可以把它分配在几个不同类的存储卷上。它可以根据某个字段值的顺序来安排存储记录的现实值，也可以根据一些其它方法来安排记录顺序。例如根据一个或多个索引或根据嵌入数据现实值中的指引元。还可以通过杂凑计算的方法来存储。当然也可以不用上述的任何方式而采用其他方法来安排文件中的记录。存储的记录可以分块，也可以不分块存放。但是，**不管采用什么方式都应以不对应用有任何影响为前题**。

上面的叙述，意味着数据库是可以在不影响已存应用户的前题下增生（GROW）的。确实，提供数据独立性的主要理由也正是要在不损害已有应用户的前题下，使数据库能够增生。例如，对一个已有的存储记录类型应该可以通过增加一个新的字段类型而扩充（通常是对已有实体或关系类型增加新的信息，例如可把"单价"字段增加到另件存储记录中去）。增加的新的字段本身意味着它不被已有的应用所看见。类似地也可以把整个新的存储记录类型加入到数据库中去，而不影响已有的应用。这类记录通常都是表示了实体或关系的全新类型。（例如，"厂商"（Supplier）记录类型可以加入到另件数据库中）。

1.5 数据库系统的总体结构

我们现在对数据库系统的总体结构进行概要的介绍（图1.3，1.4，1.5）。介绍的目的是为以后各章提供一个大纲。这个大纲无论在描述一般数据库的概念或是解释各数据库的结构上都是特别有用的。但是我们并不要求每个数据库系统都和这个特定的大纲相符合，也不认为这个特定的总体结构就是提供的唯一可能的大纲。不过，这个总体结构的确很适合大多数系统，并且和ANSI/SPARC数据库管理系统研究组的提案相当一致。(不过，我们并不完全采用ANSI/SPARC的术语）。

这个总体结构分为三层：内部层、概念层和外部层（图1.3）。大体说来，内部层是最接近实际存储体的，就是说它涉及数据实际存储的方法；外部层最接近用户，它涉及各个用户对数据的看法。概念层则是一个介于上述两层之间的“中间层”。如果说，外部层是有关各个用户的看法，那么概念层可以想像为定义了用户大众的总看法。换句话说，有着许多“外部看法”，每个外部看法或多或少都可以看作数据库的某部分数据的抽象表示。同时还存在着一个“概念看法”，它可以看作是整个数据库的一个抽象的表示。（这里抽象的意思是指面向用户的结构，例如逻辑记录、逻辑字段等等，而不是指面向机器的结构，如字节、二进位等等）。说实在的，大多数用户并不对整个数据库感兴趣，而只是对它的某一局部感兴趣。类似地，还有一个“内部看法”，它代表了实际存储的整个数据库。

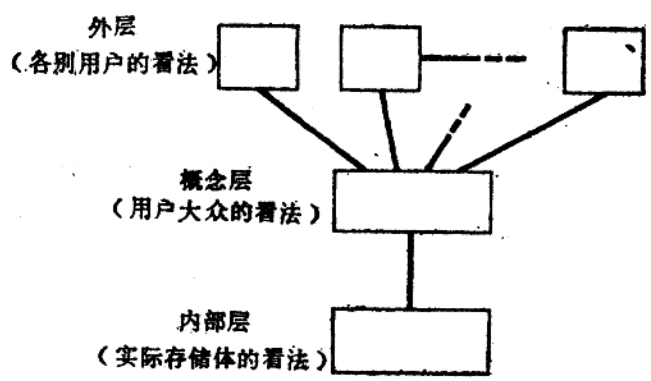

图1.3 数据库总体结构的三层

下述一个例子将帮助弄清这些思想。图1.4表明了一个简单的人事数据库的概念层对应的内部层我们和两个对应的外部层构造。（两个外部层，一个是PL/1的用户另一个是COBOL用户）。这个例子是完全假想的，它并不是某个真实系统的真实写照，许多无关的细节都已经有意省略掉了。

下面是图1.4及其说明：

External (PL/1)	External (COBOL)
DCL 1 EMPP, 2 EMP# CHAR(6), 2 SAL FIXED BIN(31);	01 EMPC. 02 EMPNO PIC X(6). 02 DEPTNO PIC X(4).

Conceptual

```
EMPLOYEE
   EMPLOYEE_NUMBER     CHARACTER  (6)
   DEPARTMENT_NUMBER   CHARACTER  (4)
   SALARY              NUMERIC    (5)
```

Internal

```
STORED_EMP  LENGTH=20
   PREFIX     TYPE=BYTE(6),OFFSET=0
   EMP#       TYPE=BYTE(6),OFFSET=6,INDEX=EMPX
   DEPT#      TYPE=BYTE(4),OFFSET=12
   PAY        TYPE=FULLWORD,FFSET=16
```

图1.4数据库三个层次的例子

概念层包含称为 **EMPLOYEE** 类型的实体数据库。每个 **EMPLOYEE**（员工）有一个 **EMPLOYEE—NUMBER**。员工号，长度6个字符，**DEPARTMENT—NUMBER**（所属部门号），长度4个字符和 **SALARY**（工资），4位数字。

在**内部层**，员工由称为**STORED—EMP** 的存储记录类型来表示。长度20个字节。**STORED—EMP** 包含四个存储字段类型：6个字节的前缀（prefix）(一般包含标记、指引之类的信息）和对应于员工性质的三个数据字段。此外 **STORED—EMP**记录还基于EMP#字段建立索引，这个索引称为EMPX。

PL/1用户对数据库的看法是：每个员工由两个字段的 PL/1 记录来表示。（由于所属部门号，对该用户不感兴趣，所以就省略掉了）。这个记录类型是借助于通常的 PL/1 结构说明规则定义的。

类似地，**COBOL** 用户的看法是用一个**COBOL**的记录来表示的，该记录也有两个字段（工资字段被省略了）。记录类型是根据通常的COBOL规则来描述的。

注意，对应的对象，在三个层次中可以各有不同的名字。例如，**COBOL**的员工号字段为 **EMPNO**，对应的存储字段则称为**EMP#**而对应的概念层则称为**EMPLOYEE—NUMBER**。这样的应对关系，系统必须知道。例如，系统必须能判明**COBOL**的字段**EMPNO**可以概念层的字段 **EMPLOYEE—NUMBER**导出，而概念层的**EMPLOYEE—NUMBE**又依次是由存储字段 **EMP#**来表示的。这样的对应叫做对映（mapping），图1.4未画出。

现在，稍详细地来讨论总体结构的某些方面。（图1.5）

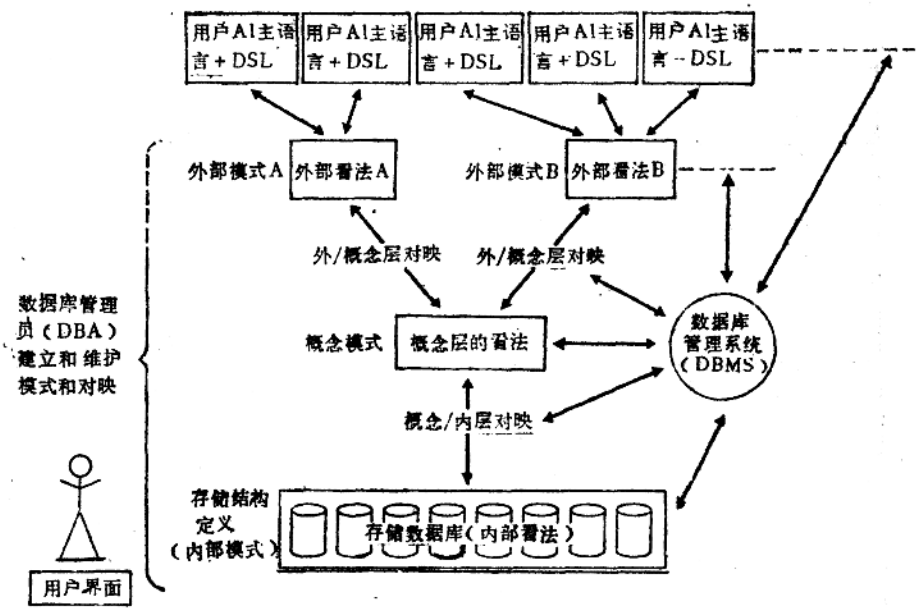

图1.5 数据库系统总体结构

用户可以是具有一定专业知识的应用程序员或终端联机用户。（DBA便是一个重要的特殊用户）。每个用户在使用权限内使用一种语言。对于应用程序员一般将使用通常的程序设计语言，例如COBOL，PL/1等；对于终端用户通常则使用查询语言或适应用户要求的由联机应用程序支援的特定语言。对数据库系统来说，重要的是用户语言要包括数据子语言（DSL—data Sublanguage）成分。数据子语言是整个语言的一个部分，这一部分仅与数据库有关（对数据库中的对象进行说明或操作）。我们谈到数据子语言时都是认为数据子语言是嵌在主语言中的。一个系统可以支援多种主语言和多种数据子语言。

原则上，任何给定的数据子语言都是两种语言的结合：由用户来定义或描述数据库的数据定义语言(DDL—data definition Langnage)和支援用户管理或处理定义了的数据的数对象据管理语言(DML—data manipulation Langnage)。例如图1.4的PL/1用户，使用了PL/1特色的数据子语言来和数据库通讯。PL/1特色的DDL，就是采用原PL/1中说明的结构，说明需要的数据库对象。这类DECLARE（DCL）说明，可用来说明某些原PL/1中没有的新数据类型，这些新对象也是原来的PL/1所不能处理的。（事实上，现行的PL/1并不包含任何数据库特色。PL/1的DML部分，是由那些从数据库或向数据库传送信息的可执行语句组成的。这些语句中也可能包含着新类型的语句。

上面所解释的数据子语言是假定和主语言（如PL/1）紧密融合在一起的，也就是说，对用户来说，没有是两个分离部分的感觉。可是当前的现实情况并非如此，至少所提到的程序设计语言并非如此：（a）定义完全在应用程序之外来做，使用的DDL也完全不象主语言。（b）数据管理则通过CALL标准子程序的方式来做。这些标准子程序作为DBMS的一部分。因而这样的DML实际上也是在语言的骨架之外的。换句话说，今天的大多数系统