第一篇　基础篇

这一篇简要地阐述数据库系统的基本概念，作为读者学习数据库技术的入门知识。第二篇重点介绍当前微机上流行的关系数据库管理系统 FoxBASE＋、FoxPro。当然在微机上常用的关系数据库管理系统还有 Access、Clipper、dBASEⅢplus、dBASEⅣ等。国外微机上用的关系数据库管理系统就更多了。如 Paradox 就用得十分普遍，而国内却用得很少。这本书限于篇幅，并按照丛书的宗旨——《精编》，即挑选当前微机上 最常用的关系数据库管理系统，因此只能给读者介绍这两个关系数据库管理系统。

第一章　数据库系统概论

为了给读者学习这本书及有关数据库技术类的著作打下基础，特别加上了基础篇。这一章给出了数据库系统的一般概念和知识，而不仅仅限于 FoxBASE＋、FoxPro。

1.1　绪　　论

1.1.1　数据、数据库、数据库系统、数据库管理系统

在系统地介绍数据库的基本概念之前，有必要首先介绍一些数据库最常用的概念。

1)数据(DATA)

数据在大多数人的头脑中第一个反应就是数字。其实数字只是最简单的一种数据，它并不是数据库技术主要的研究对象。数据的种类很多，在日常生活中数据无处不在：文字、图形、图像、声音、学生的档案记录、货物的运输情况……，这些都是数据。

为了了解世界，交流信息，人们需要描述这些事物。在日常生活中我们直接用自然语言(如汉语)描述。在计算机中，为了存储和处理这些事物，需要抽出对这些事物感兴趣的特征组成一个记录来进行描述。例如：在学生档案中，如果人们最感兴趣的是学生的姓名、性别、年龄、出生年月、籍贯、所在系别、入学时间，那么可以这样描述一个学生：

(李明、男、21、1972、江苏、计算机系、1990)

我们可以对数据作如下定义：描述事物的符号记录称为数据。因此上面的学生记录就是一个数据。对于上条学生记录，知情的人会得到下面的信息：李明是个大学生，1972 年出生，江苏人，1990 年考入计算机系，而不知情的人要说明以后才明白。可见，数据的形式还不能完全表达其内容，需要经过解释。所以数据和关于数据的解释是不可分的，即数据与其语义是不可分的。

2)数据库(DATABASE 简称 DB)

数据库，顾名思义，是存放数据的仓库。只不过这个仓库是在硬盘上，而且数据是按--

定的格式存放的。

所以数据库是长期储存在计算机内、有组织的、可共享的数据集合。数据库中的数据按一定的数据模型组织、描述和储存，具有较小的冗余度，较高的数据独立性和易扩展性，并可为各种用户共享。

人们总是千方百计地收集各种各样的数据，然后进行处理。目的是从这些数据中得到有用的信息。在科学技术飞速发展的今天，人们的视野越来越广，数据量急剧增加。过去人们手工处理数据，现在人们借助计算机科学地保存和管理复杂的大量的数据，以便人们能方便而充分地利用这些宝贵的信息资源。

3)数据库管理系统(DATABASE MANAGMENT SYSTEM 简称 DBMS)

有了数据，又有了数据库，下一个问题就是如何科学地组织和存储数据、如何高效地获取和处理数据了。完成这个任务的是一个软件系统——数据库管理系统 DBMS。

数据库管理系统是位于用户与操作系统之间的一层数据管理软件。

数据库在建立、运用和维护时由数据库管理系统(DBMS)统一管理、统一控制。用户能方便地定义数据和操纵数据，并保证数据的安全性、完整性、多用户对数据的并发使用及发生故障后的系统恢复。数据库是数据库系统的一个重要组成部分。

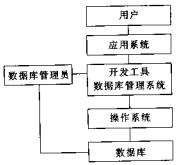

图 1-1

4)数据库系统(DATABASE SYSTEM 简称 DBS)

数据库系统是指在计算机系统中引入数据库后的系统构成，一般由数据库、DBMS(及其开发工具)、应用系统、数据库管理员和用户构成。应当注意的是，数据库的建立、使用和维护等工作只靠一个 DBMS 远远不够，还要有专门的人员来完成，这些人被称为数据库管理员。

在一般不引起混淆的情况下常常把数据库系统简称为数据库。

数据库系统可以用图 1-1 表示。

数据库技术是数据管理的最新技术。数据库系统是当代计算机系统的重要组成部分，对于它的研究方兴未艾。

1.1.2 从文件系统到数据库系统

数据库系统是在文件系统上发展起来的。

人们对数据的处理是指对数据进行收集、组织、存储、加工和传播的一系列活动的总和。但人们借助计算机进行数据处理是近二十年的事。研制计算机的初衷是利用它进行复杂的科学计算。随着计算机软件和硬件的发展，其应用远远地超出了这个范围。1954 年通用电器公司(GEC)研制出的第一台商业数据处理的电子计算机 UNIVACI，标志着计算机开始进行事务处理并存储信息文件，产生了电子信息系统。人们得益于计算机惊人的处理速度和大容量的存储器，从而解脱了在大堆文件中寻找数据之苦，这种基于计算机的信息处理系统也就迅速发展起来。

这种信息处理系统的作法是把数据组织成相互独立的数据文件，利用“按文件名访问，

按记录进行存取"的管理技术,并可以经常对文件进行修改、插入和删除的操作。当时这种信息处理系统是很专业化的。根据具体应用建立文件,并编写处理这些文件的程序。一旦应用有了变化,不得不重新创建文件并编写新的处理文件的程序。从常识上看,获得的数据有些经常在改变,数据之间会存在这样那样的联系。上述这种独立的文件管理系统在数据量小,数据变化少时尚可维持,但在数据量大,数据变化频繁时就显得力不从心了。60 年代中期,在美国诞生了第一个商品化的数据库系统—— IMS 系统(Information Managment System)。

从文件系统到数据库系统,标志着数据管理技术的飞跃。80 年代后不仅在大型机上而且在多数微机上配置了 DBMS,数据库技术得到广泛的应用和普及。这本书介绍的 FoxBASE + 、FoxPro 就是配置在微机上的关系数据库管理系统(RDBMS)。

这种利用数据库管理数据的方法即"数据库方法",概括地讲是指把某系统要用到的各种数据综合整理为一组相互关联的文件(而不是文件系统中根据具体应用组织成相互独立的文件)实现数据共享。

1.1.3 数据库系统的优点

与文件系统相比,数据库系统的优点是很突出的:

(1) 数据由 DBMS 统一管理

DBMS 既管理数据的物理结构,也管理数据的逻辑结构;既考虑数据,也考虑数据之间以及文件之间的数据联系;DBMS 管理的是结构化的数据。

(2) 数据结构化

在文件系统中,相互独立的文件的记录内部是有结构的。传统文件的最简单形式是等长同格式的记录集合。例如:一个学生人事记录文件,每个记录都有如图 1-2 的记录格式:

学生人事记录

学号	姓名	性别	系别	年龄	政治面貌	家庭出身	籍贯	家庭成员	奖惩情况

图 1-2

前八项是任何学生必须具有的而且是等长的,而后两项信息量的多少变化较大。为了建立完整的学生档案文件,每个学生记录的长度必须等于信息量最多的记录长度。这样对于信息量较少的学生记录将在空间上造成很大的浪费。因此我们可以用变长记录,即用主记录与详细记录相结合的形式建立文件。将学生人事记录的前八项作为主记录,后两项作为详细记录,则每个记录有如图 1-3 记录格式。

则某个学生记录为图 1-4。

这样可以节省许多存储空间,灵活性也有相对提高。

但这样建立的文件还有局限性,因为这种灵活性只对一个应用而言。一个学校或一个组织涉及许多应用,在数据库系统中,不仅要考虑某个应用的数据结构,还要考虑整个组织的数据结构。例如一个学校的管理信息系统中不仅要考虑的学生的人事管理,还要考虑学籍管理、选课管理等等,可按图 1-5 方式为该校的信息管理系统组织学生数据。

这种数据组织方式为各种管理提供必要的记录,使学校的学生数据结构化了。这就要

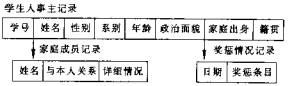

图 1-3

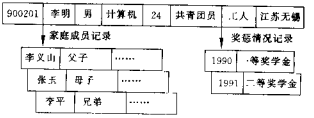

图 1-4

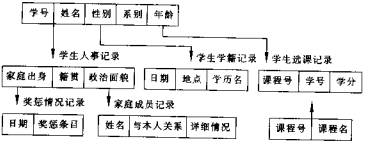

图 1-5

求在描述数据时不仅要描述数据，还要描述数据之间的联系。文件系统尽管其记录内部已有了某些结构，但记录之间没有联系。数据库系统实现整体数据的结构化，是数据库的主要特征之一，也是数据库系统与文件系统的本质区别。

(3) 数据冗余度小

在文件系统中，一个应用面对自己专用的一个或几个数据文件，会有许多数据相互重复。上例中一个学生的人事记录文件可能被诸如学生人事管理、学生学籍管理、学生选课管理等几个应用使用，将被重复几次包括在这几个应用的专用数据文件中，产生了很大的数据重复，即数据冗余。

数据库系统从整体角度看待和描述数据，数据不再面向某个应用面是面向整个系统。上例中学生基本记录就可以被多个应用共享使用。从而大大减少数据冗余，节约了存储空间，避免了数据之间的不一致性。

数据的不一致性是指同一数据不同拷贝的值不一样。这是由于文件系统中的数据重复

存储，不同的应用使用和修改不同的拷贝造成的。

(4) 具有较高的数据独立性

数据独立性既有物理独立性，又有逻辑独立性

物理独立性是指当数据存储结构(或称物理结构)改变时，数据的逻辑结构不变，则用户编写的应用程序不变。逻辑独立性是指当数据的总体逻辑结构改变时，数据的局部逻辑结构不变，应用程序是依据数据的局部逻辑结构编写的，所以应用程序不变。

(5) 数据的共享性好

由于数据由数据库统一管理，而且所管理的是有结构的数据，在使用数据时就有很灵活的方式，可以适应各种用户的要求。而且数据易于扩展，以满足新的应用要求。数据库中数据共享的意义是多种应用、多种语言、多种用户相互覆盖的使用数据集合。为了适应共享数据的环境，DBMS 提供了以下四个方面的数据控制功能：

①数据的安全性保护

数据的安全性是指保护数据，以防止不合法的使用造成的数据的泄密和破坏。使每个用户只能按规定，对某些数据以某些方式进行使用和处理。

②数据的完整性

数据的完整性指数据的正确性、有效性和相容性。即控制数据在一定的范围内有效，或要求数据之间满足一定的关系。

③并发控制

当多个用户的并发进程同时存取、修改数据库时，可能会发生相互干扰而得到错误的结果并使得数据库的完整性遭到破坏，因此必须对多用户的并发操作加以控制和协调。

④数据库恢复

计算机系统的硬件故障、软件故障、操作员的失误以及故意的破坏都会影响数据库中数据的正确性，甚至造成数据库部分或全部数据的丢失。DBMS 必须具有将数据库从错误状态恢复到某一已知的正确状态(亦称为完整状态或一致状态)的功能，这就是数据库的恢复。

综上所述，数据库是长期存储在计算机内有组织的大量的共享的数据集合。它可以供各种用户共享、具有最小冗余度和较高的数据独立性。DBMS 在数据库建立、运用和维护时对数据库进行统一控制，以并保证数据的完整性、安全性，并在多用户对数据的并发使用时进行并发控制和发生故障后的系统恢复。

1.1.4 数据库技术的研究范围

概括地讲，数据库学科主要的研究范围有以下三个领域：

(1) 数据库管理系统软件的研制

DBMS 是数据库系统的基础。DBMS 的研制包括：DBMS 以及以 DBMS 为核心的一组相互联系的软件系统。研制的目标是扩大功能、提高性能和提高用户的生产率。

(2) 数据库设计

数据库设计的主要任务是在 DBMS 的支持下，按照应用的要求，为某一部门或组织设计一个结构合理、使用方便、效率较高的数据库及其应用系统。其中主要的研究方向是数据库设计方法学和设计工具的研究和开发。

(3) 数据库理论

数据库理论的研究主要集中于关系的规范化理论,关系数据理论。近年来,随着人工智能与数据库理论的结合,数据库逻辑演绎和知识推理等理论、演绎数据 库系统、知识库系统的研制都已成为新的研究方向。

1.2 三种常用的数据库系统

1.2.1 数据模型

模型,特别是具体模型,人们并不陌生。一张地图、一组建筑设计沙盘、一架精致的航模飞机都是具体的模型。只要一眼望去就会联想到真实生活中的事物。这里即将谈到的数据模型(Data Model)也是一种模型,但它看不见也摸不着。它是数据特征的抽象,描述的是数据的共性。

数据库是某个企业、组织或部门所涉及到的数据的综合。计算机不可能直接处理现实世界中的具体事物,需要把具体事物转换成计算机能够处理的数据。在数据库中用数据模型这个工具来抽象、表示和处理现实世界中的数据和信息;现实世界中的实体对应于机器世界中的数据,现实世界中实体之间的联系在机器世界中就是数据之间的联系。通俗地讲数据模型就是现实世界的模拟。

数据模型应满足三方面要求:一是能比较真实地模拟现实世界;二是容易为人所理解;三是便于在计算机上实现。一种数据模型要很好地满足这三方面的要求在目前尚很困难。在数据库系统中应针对不同的使用对象和应用目的,采用不同的数据模型。

实体(Entity)在这里是指客观存在并可相互区分的事物。它可以是具体的人、事、物,也可以是抽象的概念或联系,如学生的一次选课,老师与系的工作关系(即某位老师在某系工作)等。实体概念的关键是"可相互区分"这个特性。计算机这个没有想象力的家伙是无法处理模棱两可的事物或联系的。

不同的数据模型具有不同的数据结构形式。目前最常用的数据模型有层次模型(Hierachical Model)、网状模型(Network Model)和关系模型(Relational Model)。其中层次模型和网状模型统称为非关系模型。

80年代以来,面向对象的方法和技术在计算机各个领域,包括程序设计语言、软件工程、信息系统设计、计算机硬件设计等各方面都产生了深远的影响,也促进数据库中面向对象数据模型的研究和发展。

数据库按数据模型来分,可分为层次数据库、网状数据库、关系数据库。

在非关系模型中,实体用记录表示,实体之间的联系转换成记录之间的两两联系。

非关系模型中数据结构的基本单位是基本层次联系。

两个记录以及它们之间的一对多(包括一对一)的联系称为基本层次联系,用图 1-6 表示如下。

图中 Ri 位于联系 Lij 的始点,称为双亲结点(Parent),Rj 位于联系 Lij 的终点,称为子女结点(Child)。

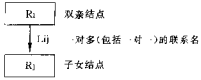

图 1-6 基本层次联系

1.2.2 层次数据库

层次数据库采用层次模型作为数据的组织方式。

在数据库中,把满足以下两个条件的基本层次联系的集合称为层次模型:

(1) 有且仅有一个结点无双亲,这个结点称为根结点;

(2) 其他结点有且仅有一个双亲。

在层次模型中,同一双亲的子女结点称为兄弟结点(Twin 或 sibling),没有子女结点的结点称为叶结点。图 1-7 是层次模型的一个例子:

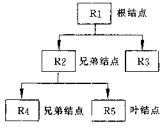

图 1-7 一个普通的层次模型

其中 R1 为根结点;R2 和 R3 为兄弟结点,是 R1 的子女结点;R4 和 R5 为兄弟结点,是 R2 的子女结点;R3、R4 和 R5 为叶结点。

从图上可以看出层次模型是一棵倒立的树,结点的双亲是唯一的。

一个典型的层次数据库管理系统是 1968 年 IBM 公司研制出的 IMS 系统。

我们已经知道,层次模型中一个具有双亲与子女关系的记录反映的是现实世界的一个一对多的联系。我们以学生与系的关系为例,看一看在层次数据库中怎样用层次模型来组织数据(图 1-8、图 1-9)。

可见用层次模型对具有一对多的层次关系的部门描述得非常自然、直观,容易理解。这是层次数据库的突出优点。

但是层次数据库的双亲记录与子女记录之间只能反映一对多的联系,而现实世界中记录之间普遍存在着多对多的联系。这就需要用网状模型来表示了。

图 1-8　系别/学生层次数据库模式

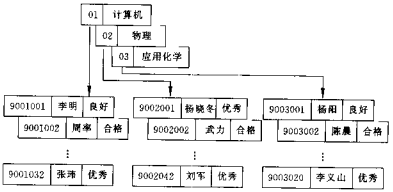

图 1-9　系别/学生层次数据库实例

1.2.3　网状数据库

网状数据库采用网状模型作为数据的组织方式。

在数据库中,我们把满足以下两个条件的基本层次联系集合称为网状模型:

(1) 允许一个以上的结点无双亲;

(2) 一个结点可以有多于一个的双亲。

从定义可以看出,层次模型中子女结点与双亲结点的联系是唯一的,而在网状模型中这种联系可以不唯一。因此,要为每个联系命名,并指出与该联系有关的双亲记录和子女记录。网状模型如图 1-10:

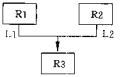

图 1-10　一个简单的网状模型

R1 与 R3 之间的联系被命名为 L1,R2 与 R3 之间的联系命名为 L2。其中 R1、R 2 为 R3 的双亲结点。

图 1-11 中都是网状模型的例子。

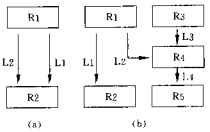

图 1-11　网状模型的例子

网状数据库的典型代表是 DBTG 系统，亦称 CODASYL 系统。这是 70 年代数据系统语言研究会 CODASYL(Conference On Data Systems Language)下属的数据库任务组(Data Base Task Group 简称 DBTG)提出的一个系统方案。

DBTC 系统使用的是网状模型。

网状模型允许结点无双亲，或有一个以上的双亲，从而构成了比层次结构复杂的网络结构。以学生选课为例，看一看网状数据库是怎样用网状模型来组织数据(图 1-12、图 1-13)。

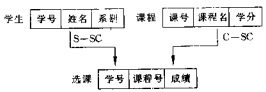

图 1-12　学生/选课/课程的网状数据库模式

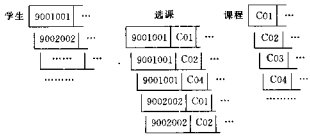

图 1-13　学生/选课/课程的网状数据库实例

1.2.4　关系数据库

关系数据库采用关系模型作为数据的组织方式。

关系模型是上述三种数据模型中最重要的模型。美国 IBM 公司的研究员 E.F. Codd 于 1970 年发表题为“大型共享系统的关系数据库的关系模型”的论文，文中首次提出了数据库

系统的关系模型。80年代以来，计算机厂商新研制出的数据库管理系统(DBMS)几乎都支持关系模型，非关系系统的产品也大都加上了关系接口。数据库领域当前的研究工作都是以关系数据库为基础。

在用户看来，一个关系模型的逻辑结构是一张二维表，它由行和列组成。

现以学生的人事记录(图 1-14)为例，介绍关系模型及其主要术语。

学号	姓名	性别	系别	年龄	籍贯
9001001	李明	男	计算机	22	江苏无锡
9001002	周琴	女	计算机	22	上海
9001032	张玮	女	计算机	22	北京
：	：	：	：	：	：
9002001	杨晓冬	男	物理	21	山西大同
：	：	：	：	：	：
9003020	李义山	男	应用化学	20	山东烟台

图 1-14　学生人事记录表

关系：对应通常说的表，如上面的这张学生人事记录表(假设三个系有 94 位学生)；

元组：表中的一行即为一个元组，如上表有 94 行，也就有 94 个元组(假设，三个系有 94 位学生)；

属性：表中的一列即为一个属性，如上表有六列，对应六个属性(学号，姓名，性别，系别，年龄和籍贯)；

码(Key)：表中的某个属性组，它可以唯一确定一个元组，如上表的学号，按照学生学号的编排，每个学生的学号都不相同，所以它可以唯一确定一个学生，也就成为本关系的码；

域(Domain)：属性的取值范围，如人的年龄一般在 1 岁～150 岁之间。上表中学生年龄这个属性的域应是(14～35)，性别的域是(男，女)系别的域是一个学校所有系名的集合；

分量：元组中的一个属性值；

关系模式：对关系的描述，一般表示为：关系名(属性 1，属性 2，…，属性 n)，例如上面的关系描述为　学生(学号，姓名，性别，系别，年龄，籍贯)。

关系模型与以往的模型不同，它是建立在严格的数学概念的基础上的。此外，关系模型的概念简单、清晰，用户易懂易用，简化了程序员的工作和数据库开发建立的工作，因面关系数据模型诞生以后发展迅速，深受用户的喜爱。在后面章节介绍的 FoxBASE＋、FoxPro 就是关系数据库管理系统。

1.3　数据库模式

1.3.1　模式和实例

在数据模型中有“型”(TYPE)和“值”(VALUE)的概念。型是指对某一类数据的结构和

属性的说明，值是型的一个具体赋值。例如：学生人事记录定义为——(学号，姓名，性别，系别，年龄，籍贯)这样的记录型，而(900201，李明，男，计算机，22，江苏无锡)则是该记录型的一个记录值。

模式是数据库中全体数据的逻辑结构和特征的描述，它仅仅涉及到型的描述，不涉及到具体的值。而实例则是模式的一个具体值，同一个模式可以有很多的实例。模式是相对稳定的，而实例是相对变动的。模式反映的是数据的结构及其关系，而实例反映的是数据库某一时刻的状态。

1.3.2 三级模式

三级模式是数据库系统的体系结构特征。数据库三级模式是：

外模式：是用户见到的那一部分数据的逻辑结构和特征的描述，是数据库用户见到的数据视图，亦称子模式或用户模式。每个用户的外模式不一定相同。

逻辑模式：是数据库中全体数据的逻辑结构和特征的描述，是所有用户的数据视图，亦称模式。

内模式：是数据物理结构和存储结构的描述，亦称存储模式。是数据库的内部表示。

三级模式都和 DBMS 有关，是数据库系统中实现的模式。模式描述的是数据的全局逻辑结构，外模式描述的是数据的局部逻辑结构。数据库的三级模式如图 1-15：

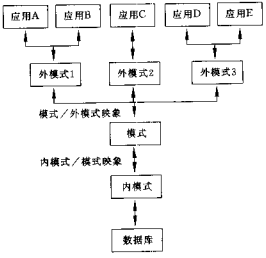

图 1-15 数据库系统的三级模式

1.3.3 数据独立性

数据独立性概念比较抽象，举个简单的例子读者可以实际体会。

在 C 语言程序(其它高级语言程序类似)中，通常是把数据说明和数据类型定义与程序放在同一个.C 文件中。编写过类似程序的读者可以知道如果数据说明或数据类型定义稍有改变，程序必须做相应改变才能运行，否则编译时会出错，这时数据紧紧依赖于程序，便

称数据和程序不具有独立性。前面已经提及数据库系统的优点之一是具有较高的数据独立性。这是由于数据库系统的三级模式和两级映象的体系结构。

下面给出数据独立性的定义。首先要了解数据库系统中映象的概念。

在数据库系统中有两种映象功能(或称为转换功能):模式到外模式的映象、内模式到模式的映象。

第一种映象的功能使得当数据存储结构(或称物理结构)改变时,数据的逻辑结构不变,则用户编写的应用程序不变。这称为数据与程序的物理独立性,简称数据的物理独立性。

编写应用程序用到的数据是总体数据的子集,数据库系统为这些局部数据提供数据说明功能。数据库系统利用第二种映象使得当数据的总体逻辑结构改变时,数据的局部逻辑结构不变,则用户编写的应用程序不变,这称为数据与程序的逻辑独立性,简称数据的逻辑独立性。

实现这两种映象的根本是把数据说明和数据定义从应用程序中抽出。具体的映象工作由 DBMS 完成,从而减化了编制应用程序的工作量,减少了应用程序的维护和修改。

1.4 数据库语言

1.4.1 数据定义语言(DDL)

在 1.3.2 中介绍了数据库的三级模式,在数据库中数据库模式是用数据定义语言(Data Definition Language 简称 DDL)来描述的,它是作为 DBMS 的一部分提供的。这个语言分为三个方面:模式 DDL、子模式 DDL 和定义数据库物理结构的物理 DDL。在微机数据库管理系统中一般只有模式 DDL,例如 FoxBASE + 中利用 CREATE [〈数据库文件名〉]、MODIFY STRUCTURE 等命令定义和修改数据库模式。

1.4.2 数据操纵语言(DML)

数据操纵语言(Date Manipulation Language 简称 DML)是 DBMS 提供的用来检索、插入、修改和删除数据库数据的语言。DML 有两种基本类型:过程化 DML 和非过程化 DML。过程化 DML 不仅要求用户指出所需的数据是什么,还要指出如何存取这些数据;非过程化 DML 只要求用户指出所需的数据而不必指出存取这些数据的过程。可见非过程化 DML 比过程化 DML 容易理解和使用。层次、网状数据库系统中 DML 一般是过程化的;关系数据库系统中的 DML 一般是非过程化的。FoxBASE + 提供的就是一种非过程化的 DML。它利用 LOCATE 语句查询数据,用 INSERT[BLANK] [BEFORE]、DELETE[〈范围〉] [WHILE〈条件〉] [FOR〈条件〉]、UPDATE 等命令操纵数据库数据。

如要查询年龄在 22 岁以上的学生:

```
.use student
.locate for age > 22
```

然后用

```
.disp
```

语句将满足条件的一条记录显示出来。但值得注意的是，满足条件的记录会不止一个，这时在记录显示后用

```
.continue
```

语句让系统继续寻找满足条件的记录，然后再用 disp 语句，把记录显示出来。如果没有找到这样的记录，系统会这样显示：

```
End of Locate scope.
```

如要在学生表 student 中删除第三条记录，用 FoxBASE + 命令：

```
.use student
.delete record 3
```

如果要插入记录可以这样写：

```
.insert before
学号  900200
姓名  王峰
性别  男
系别  计算机
年龄  22
籍贯  北京
```

这时该条记录被插入在当前记录之前。

在上述对数据库操作中用户输入的命令只是告诉系统所要做的操作和要查询的数据。语句简单，适合于终端用户使用。

另一种广泛使用的关系数据库语言是：SQL(Structured Query Language)语言。它已经成为关系数据库语言的工业标准，在大型 DBS 中广泛采用。

1.5 数据库保护

为了保护数据的安全可靠和正确有效，DBMS 必须提供统一的数据保护功能或称为数据控制功能。主要有数据的安全性、完整性、并发控制和恢复。

1.5.1 数据库的安全性控制

数据库的安全性是指保护数据库以防止不合法的使用所造成的数据泄漏、更改或破坏。

数据库系统中集中存放着大量的数据，并为许多用户直接共享，是宝贵的信息资源。系统的安全保护是否有效是数据库系统的主要性能指标之一。

数据库安全性的保密措施有系统处理的和物理的两种。物理的是指对于用武力强迫透露口令、在通讯线路上窃听、以至盗窃数据的物理存储设备等行为采取的将数据编成密码、加强警卫、保护存储设备等。这不在书中所讨论范围之内。这里只讨论计算机系统提供的保护措施。

计算机系统的安全措施是一级一级层层设置的，可以用图 1-16 表示。

用户进入数据库系统前，由系统提供一定的方式让用户标识自己的身份或名字，系统

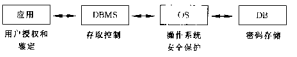

图 1-16

进行核实,通过鉴定后才能授予用户使用机器的权力。

对于获得上机权的用户还要根据预先定义好的用户权限进行存取控制,保证用户只能存取他有权存取的数据。所谓用户权限是指不同的用户对于不同的数据对象允许执行的操作权限。例如,用户甲可以读学生表,但不能修改它;用户乙可以读,也可以修改学生表;用户丙可以读所有的数据库表,但不能修改也不能建立新表。

不同的数据库系统有自己的安全性控制方法。

数据库安全保密显然是很重要的。任何一个数据管理系统都必须有一定的安全性保护措施,但是它们都不是绝对安全的。

1.5.2 完整性定义和检查

数据库完整性是指数据的正确性和相容性。例如,学生的年龄必须是整数,取值范围为 14——35;学生的学号一定是唯一的;学生所在的系必须是学校开设的系……等。DBMS 必须提供一种功能来保证数据库数据的完整性,这种功能亦称为完整性检查。

数据的完整性与安全性是两个不同的概念。前者是指为了防止数据库中存在不合语义的数据、防止错误的输入和输出所造成的无效操作和错误结果;后者是指保护数据库不被恶意的破坏和非法的存取。

DBMS 以一定的机制来检查数据库中的数据是否满足规定的条件。实现数据库完整性应做到:(1)系统要提供定义完整性约束条件的机制;(2)提供检查是否违背完整性约束条件的方法。若发现用户的操作使数据违背了完整性约束条件,DBMS 就采取一定的动作,如拒绝用户执行该操作,来保证数据的完整性。

和安全性保护相同,对于完整性的控制方式也是因系统的不同而不同。

1.5.3 并发控制

数据库是一个共享资源,可以由多个用户使用。这些用户可以一个一个地串行执行。每个时刻只有一个用户程序运行,执行对数据库的存取。其它用户必须等到该用户程序结束后才能对数据库存取。如果一个用户程序涉及大量数据 的输入/输出交换,则数据库系统的大部分时间将处于休闲状态。为了充分利用数据库资源,应该允许多个用户程序并行地存取数据库。这样就会产生多个用户程序并发地存取同一数据的情况。若对并发操作不加控制用户就可能存取和存储不正确的数据,破坏数据库的完整性(这里也称为一致性)。

一、基本概念

事务(Transaction)是并发控制的单位。事务是一个操作序列。这些操作要么都做,要么都不做,是一个不可分割的工作单位。事务通常以 BEGIN TRANSACTION 开始,以 COMMIT 或 ROLLBACK 操作结束。COMMIT 即提交,提交事务中所有的操作,事务正常结束。

ROLLBACK 即撤消已有的所有操作,滚回到事务开始时的状态。这里的操作指对数据库的更新操作。

事务(Transaction)和程序是两个概念,一般地讲,程序可包括多个事务。由于事务是并发控制的基本单位,所以下面的讨论均以事务为对象。

一个最常见的并发操作的例子是飞机定票系统中的定票操作。例如,在该系统中的一个活动序列:

(1) 甲售票员读出某航班的机票余额 A,设 A = 16

(2) 乙售票员读出同一航班的机票余额 A,也为 16

(3) 甲售票点买出一张机票,将机票余额修改 A←A - 1,所以 A = 15,把 A 写回数据库

(4) 乙售票点也买出一张机票,将机票余额修改 A←A - 1,所以 A = 15,把 A 写回数据库

结果明明买出两张机票,数据库中机票余额只减少 1。

这种情况称为数据库的不一致性。这种不一致性是由并发操作引起的。因为在并发操作情况下,对甲、乙两个事务的操作序列的调度是随机的。若按上面的调度序列执行,甲事务的修改就被丢失。这是由于第 4 步中乙事务修改 A 并写回后覆盖了甲事务的修改。

并发控制就是要用正确的方式调度并发操作,避免造成数据的不一致性,使一个用户事务的执行不受其它事务的干扰。

并发控制的主要方法是采用封锁机制。例如在飞机订票例子中,若甲事务要修改 A 时,在读出 A 前先封锁 A。这时其他事务就不能读取和修改 A,直到甲修改并写回 A 后解除了对 A 的封锁为止。这样,就不会丢失甲的修改。

二、封 锁(Locking)

封锁就是事务 T 可以向系统发出请求,对某个数据对象(最常用的是记录)加锁。于是事务 T 对这个数据对象就有一定的控制,其他事务不能更新此数据,直到 T 释放(Unlock)它的锁为止。确切的控制由封锁的类型决定。基本的封锁类型有两种:

排他锁(Exclusive locks 简记为 X 锁)和共享锁(Share locks 简记为 S 锁)。

若事务 T 对数据 R 加上 X 锁,则只允许 T 读取和修改 R。其他一切事务对 R 的任何封锁请求全不能成功,直到 T 释放 R 上的 X 锁。这就保证了其他事务不能再读取和修改 R,直至 T 释放 X 锁。

若事务 T 对数据 R 加上 S 锁,则只其他事务对 R 的 X 锁请求不能成功,而对 R 的 S 锁请求可以得到。这就保证了其他事务可以读取 R 但不能修改 R,直至 T 释放 S 锁。

可以用相容矩阵(图 1-17)来表示这些控制方式。

	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

Y = Yes,相容的请求; N = No,不相容的请求

图 1-17 封锁类型的相容矩阵

若事务 T1 已经获得了 R 上的某种锁,用左边列来表示这些锁(横线表示没有加锁)。另

一个事务 T2 发出了某种封锁请求，用最上面的一行来表示这些锁。则 Y 表示事务 T2 的请求可以满足，与 T1 的封锁相容。N 表示事务 T2 的请求被拒绝，与 T1 的封锁冲突。

利用封锁机制，可以解决飞机定票系统中所讲的丢失修改问题。

	T1	T2
(1)	请求封锁	
	Xlock A	
	读 A = 16	
(2)		请求封锁
		Xlock A
		等待
(3)	A < A - 1	等待
	写回 A = 15	等待
	Commit	等待
	Unlock X	
(4)		获得封锁
		Xlock A
		读 A = 15
(5)		A < - A - 1
		写回 A = 14
		Commit
		Unlock X

图 1-18　没有丢失修改

图 1-18 中用封锁机制执行并发控制时是基于这样的约定：

(1) 事务 T 在读，写数据对象 R 时首先要发出 Slock 或 Xlock 的请求，事务 T 获得所要的锁后才能读，写 R。

(2) 事务 T 结束时(无论是 COMMIT——正常结束或者 ROLLBACK——非正常结束)才释放锁。

运用 X 锁和 S 锁这两种基本封锁，还可以建立不同的约定，形成不同级别的封锁协议。

三、封锁的粒度(Granularity)

封锁的对象的大小称为封锁的粒度(Granularity)。封锁的对象以关系数据库为例，可以是这样一些逻辑单元：属性值，属性值的集合，元组，关系，某索引项，整个索引直至整个数据库。或者是这样一些物理单元：页(数据页或索引页)，块等。

选择多大的单元为封锁对象与系统的并发度和并发控制的开销有关。直观地看，封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，封锁机构简单，开销小。反之，封锁的粒度越小，并发度较高，但封锁机构复杂，开销大。

因此，需要同时考虑封锁机构和并发度这两个因素，适当选择封锁粒度以求得最优的效果。此外，对于一个仅处理少量数据的用户事务来讲，以元组为封锁单元就比较合适，若以

关系为封锁单元就太大。反之,对于一个处理大量数据的用户事务来讲,以关系为封锁单元就比较合适了。

如果在一个系统中同时存在不同大小的封锁单元供不同的事务选择使用是比较理想的。

四、活锁和死锁

和操作系统一样,封锁的方法可能引起活锁和死锁。例如事务 T1 封锁了数据 R,事务 T2 请求封锁 R,于是 T2 等待。T3 也请求封锁 R,当 T1 释放了 R 上的封锁之后系统首先批准了 T3 的请求,T2 仍然等待。然后 T4 又请求封锁 R,当 T3 释放了 R 上的封锁之后系统又批准了 T4 的请求,...,T2 有可能永远等待,这就是活锁的情形(图 1-19(a))。避免活锁的简单方法是采用先来先服务的策略。即让封锁子系统按请求封锁的先后次序对事务排队。数据 R 上的锁一旦释放就批准申请队列中第一个事务获得锁。

又如事务 T1 封锁了数据 R1,T2 封锁数据 R2。然后 T1 又请求封锁 R2,T2 请求封锁 R1,于是 T1 等待 T2 释放 R2 上的封锁而同时 T2 等待 T1 释放 R1 上的封锁。这就使得两个事务永远不能结束。出现了死锁的局面(图 1-19(b))。

T1	T2	T3	T4
lock R	.	.	.
.	.	.	.
.	lock R	.	.
.	等待	lock R	.
Unlock	等待	等待	lock R
.	等待	lock R	等待
.	.	.	.
		Unlock	等待
		.	lock R
		.	.
		.	.

(a) 活锁

T1	T2
lock R1	
	lock R2
lock R2	
等待	lock R1
等待	等待
.	.

(b) 死锁

图 1-19

死锁的问题在操作系统中已深入研究。在数据库中解决死锁常用的方法有:

(1) 要求每个事务一次就将所有要使用的数据全部加锁,否则就不能执行。例如,若在上面例子中事务 T1 将数据 R1,R2 一次加锁,T1 执行而 T2 等待。这样就不会发生死锁。但是这样势必造成封锁范围的扩大而降低并发度。

(2) 预先规定一个封锁顺序,所有的事务全必须按这个顺序对数据执行封锁。

(3) 不采取任何措施来预防死锁的发生,而是采用某种方法诊断系统中是否有死锁。如果发现死锁就设法解除。如选择一个处理死锁需花费代价最小的事务 T,把 T 撤消,释放所有被 T 持有的封锁,使其他事务得以继续运行。当然,对 T 所执行的数据修改操作必须加以恢复。