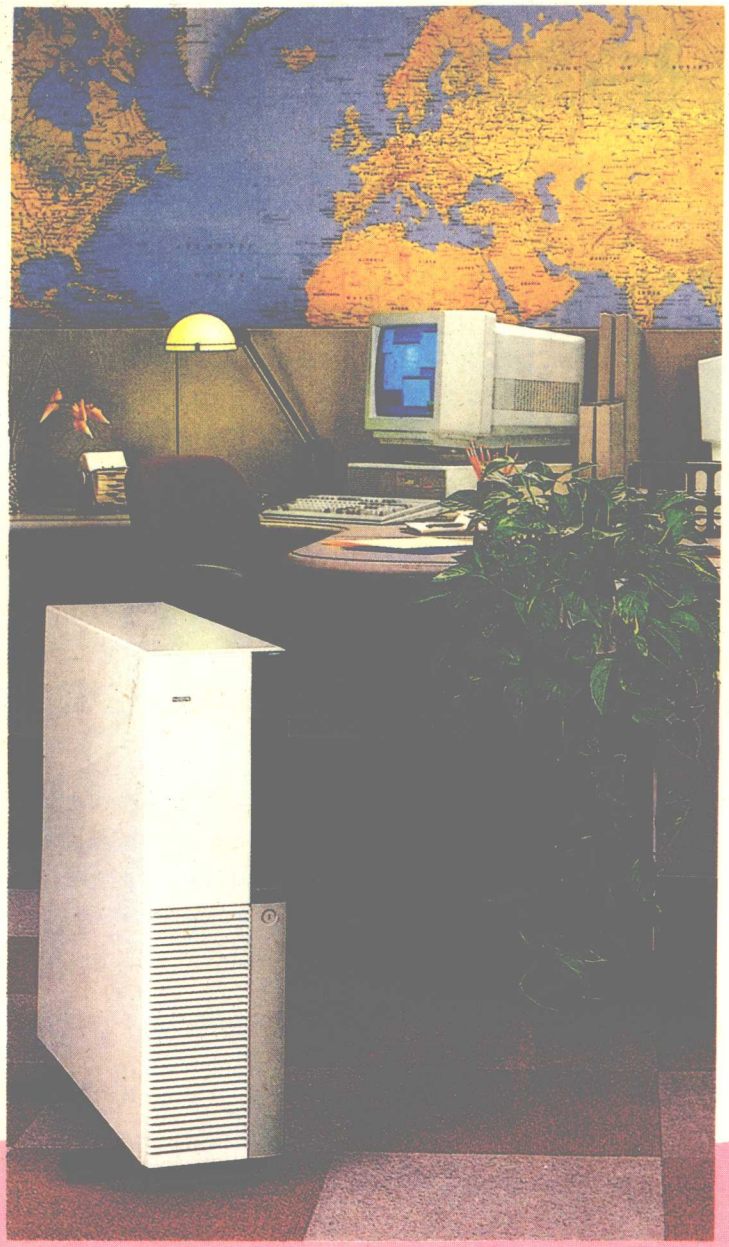
UNIX 系统 DBMS 实用指南



纪成 编译

北京科海培训中心

UNIX 系统 DBMS

纪 成 编译

北京科海培训中心

前　言

《UNIX 环境 DBMS》一书向读者提供了一个深入阐述 UNIX 环境下数据库管理系统的实用指南。它讨论了如何分析评估一个数据库、如何使用一个数据库以及如何根据自己的需求来选择一种合适的数据库管理系统。

除了对数据库管理系统的一般知识进行阐述之外，本书着重讨论了四种主要的 UNIX DBMS 产品的选择、设计、开发以及应用测试。这四种主要的 UNIX DBMS 产品是：Informix、Ingres、Oracle 和 Accell（Unify）。

本书分为五个部分。第一部分是“理论基础”，对 DBMS 的基础理论给出了概括性的介绍，特别对关系方法给予了特别说明，因为大多数 UNIX DBMS 是以关系模型为基础的。第二部分是“UNIX 与 DBMS 应用”，讨论了 DBMS 与操作系统间的交互作用，这部分内容对应用程序的设计者、开发者和终端用户来说是极为重要的。第三部分“四种 UNIX DBMS”详细讨论了四种主要的 UNIX DBMS 产品：Informix、Ingres、Oracle 和 Accell（Unify）。第四部分“选择一种 UNIX DBMS”讨论了如何进行需求分析以选择一种合适的 DBMS 产品。第五部分“未来的发展方向”，阐述了数据库管理系统近年来的发展动向及今后的发展趋势。

本书的问世，首先要感谢北京科海培训中心对我们的热情鼓励和支持，感谢夏非彼编辑对我们的帮助。

由于时间紧迫，加之作者水平有限，书中错误之处在所难免，敬请指正。

目　　录

第一部分　理论基础

第二部分　UNIX 和 DBMS 应用

第五部分 未来的发展方向

第一部分　理论基础

这部分内容是对应用于数据库管理系统技术中的理论的概述。其主要目的在于向读者介绍在数据库领域中较为常用的术语，而不是向读者提供关于 DBMS 的系统教程。因此，在这部分中，我们只是以非正规的格式来介绍一下 DBMS 的理论的粗浅知识。

下面几章的主要目的是向读者提供足够的背景知识，以便能够顺利地阅读本书后面技术性更强的一些章节。然而，我们仍对 DBMS 可对开发技术产生那些影响给出了非常重要的讨论并着重说明了我们为什么要使用数据库管理系统。本部分中的第一章讨论了当使用非 DBMS 数据存储技术时我们所面临的一些问题。

第二章论述了在关系数据库管理系统中使用的基本术语以及一些实用的数据设计技术。本部分中的其余两章主要说明一些商品化软件（关于 DBMS 的）中常用技术的类型。其目的在于建立一些关于 UNIX 是如何左右这些技术的基本原则，这些常用的实用技术均在本书所讨论的商品化 DBMS 软件中有所体现，而且，它们极有可能成为未来开发技术的主导方向。

第一章　什么是 DBMS

每个人都了解什么是数据库管理系统，真的是这样吗？但数据库和数据库管理系统的区别何在呢？我们经常听到这些说法，甚至一些人还每天在使用数据库或数据库管理系统。我们并不总是很清楚地了解，是什么使数据库和数据库管理系统区别了开来。这恰恰就是本章将要讨论的问题。

§1.1 历史回顾

自从人类开始保存书写记录以来，我们就拥有了所谓的数据库。无论是刻在石板上、写在纸上并保存在文件袋中，还是以信息方式存储在计算机中，简单地说，数据库就是数据的集合。当然，在计算机时代早期，我们使用的是术语——文件。由于早期计算机硬件的限制，大多数文件只能存储在卡片或纸带上。随着磁存储介质的开发利用，我们逐步使用磁带来存储文件，到今天，我们是作用磁盘来存储文件的。

术语数据库大约出现在我们开始同时保存越来越多的文件的时代。它指的是以某种方式相联系的数据文件的集合。例如，我们常常讲到的财务数据库、市场数据库或个人数据库等等。每一个数据库均可能包含几个文件，如：通用总目、分目等，在财务数据库中，还会有顾客文件。

因此，如果我们已这样长时间地使用数据库来为我们服务，那么为什么不使用一种数据库管理系统呢？答案在于过去一些年中软件开发方法提高开发效率是如何变化的。我们需要考察一下，在没有数据库管理系统的情况下，系统是如何建立的，并考察为什么 DBMS 能够有助改进这类系统的开发。

最早的软件是以机器语言用“0”、“1”代码的形式编写的。后来，人们又开发出了汇编语言，使得编写软件较以前容易了。我们称机器代码程序的时代为第一代语言，称汇编语言为第二代语言。使用汇编语言，我们可以比使用机器代码编写软件要快许多。换句话说，我们提高了软件开发的效率。

后来，又出现了诸如 COBOL、FORTRAN、PASCAL 等高级语言。我们称这些语言为第三代程序设计语言。高级语言出现后，人们再一次在量的方面提高了软件开发的顺序，因为这种语言和汇编语言比起来，程序设计工作要轻松得多。此外，和学习汇编语言比起来，更多的人则更愿意学习高级语言。然而，数据库管理系统（DBMS）通过将这些发展带入到下一个逻辑步骤而再次提高了开发效率。同第三代程序设计语言（如 COBOL）相比，其第四代语言提供了更高级别的开发界面。

数据库管理系统提供了许多工具以加速开发屏幕界面，如菜单和需要几页第三代程序设计语言代码的交互式界面。通过提供标准化工具，数据库管理系统不仅减少了在程序中的代码表述，而且增强了用户界面中的一致性。数据库管理系统向用户提供了开发典型的交互式程序的较为容易和迅速的手段。类似地，DBMS 还向用户提供了报表生成器，

用户可用来使用一些命令生成典型的报表，而无需在高级语言中所必须编写的多页代码。正如高级语言比起汇编语言来大大提高了软件开发效率一样，这些 DBMS 实用工具通过给定用户一个较高级别的开发接口，也同样提高了用户的开发效率。

数据库管理系统改变了用户使用查询工具访问数据的观念。非技术人员经过一些培训之后，可以使用诸如 ad__hoc 这样的查询工具。然而，用户仍然需要程序员来编写程序以得到较为完美的报表，或者仍然要那些经常使用报表的人予以指导。但如果用户马上需要数据且对报表格式不熟悉的话，则 ad__hoc 查询工具可使用户自己完成这项任务。

纵然在开发语言领域内取得了许多进展，但操作系统仍使用户脱离开了开发代码来访问存储数据和外围设备。操作系统用来管理硬件设备和程序的运行。这意味着，在数据管理领域中，我们再也不必关心特定文件起始和结束于那一个磁盘块了。通过有意义的各字并指定文件中记录的大小，我们就可以对文件进行引用了。操作系统的文件管理部分将这些文件各和记录大小设置翻译成磁盘上的块。

后来开发的操作系统的文件管理部分提供了不同的文件类型：顺序存取、索引存取和随机存取文件。利用这些类型的文件，对文件中记录的快速存取就变得很容易了，因为我们可以使用关键值对索引进行检索而不使用对整个文件的顺序检索。这样，我们就再次提高了程序员的工作效率（不幸的是，UNIX 仅开发了一种类型的文件：字节流文件。它所支持的唯一的记录类型是使用 ASCII NL（换行）字符作为文件中记录终结的任意的转换，文件中包含有可打印的字符。由于文件中包含二进制代码，因此不存在这样的转换）。

即使利用所有这些发展，程序员仍然面临许多耗费时间的任务。例如，操作系统的文件管理部分并不真正了解记录包含何种类型的数据。他们简单地将记录作为一个字节集合来处理，最多可以到存取程序一级以在记录中每个数据项（字段）开始和结束的地方进行解释。

数据库管理系统可以对个别的数据项（字段）进行标识。这样，就比操作系统的文件管理提供了更高一级的抽象方式。这样一来的结果是，我们再也不必访问那些由字节组成的记录并将它们翻译成有意义的字段了。从而记录中字段的顺序和位置就显得不是很重要了，在多数数据库管理系统的商品化软件中，它们都是互无关联的。

在使用普通的操作系统文件时，即使程序仅使用记录中的些字段，它也必须得读取整个的记录。每个程序必须要知道该记录中每个字段的确切位置。假定我们要在文件记录中增加一个字段（在任何应用中，这是很常见的），这个增加的字段很可能要增加记录的大小。因此，我们就不得不修改所有访问这条记录的程序，以便能够“接收”这个新的字段，甚至在程序不使用这条记录时，也得对程序进行修改。这样，我们就被迫要修改比实际需要多得多的程序，当然，修改之后，还必须对它们进行调试。所有这些工作消耗了我们程序设计时许多宝贵的时间。

数据库管理系统为我们提供了避免不必要修改的方法。其用户界面允许我们对字段进行精确的描述，而不对整个记录进行说明，在每一个程序中仅仅需要的是这些字段。这样，在记录中加入一个新的字段就不影响访问该记录的程序了，我们需要修改的仅是那些使用了这个新字段的程序。

使用普通操作系统文件的开发者还面临着其它问题。例如，考察一下存储在诸如日历

这样的商业应用中的通用数据类型。我们经常在应用数据库中存储日期，如订货日期、租凭日期、投资日期等等。我们也已开发了一些例行程序来对日期进行解释、操纵并将它们用在报表中：即一定时间形成一个报表。不幸的是我们常常需要重复地开发这些程序，有时甚至在同一个应用中也需重复地编写这些程序。

数据库管理系统提供了标准的实用工具，以描述这种常常使用的数据。它将一种数据类型归结为一个字段，以便它可以一种可靠的方式操纵诸如日期和款项这类的数据。这个通用工具集允许用户指定数据的检索条件，如“今天”和“本星期”等。这样，我们就再也不需要开发（和维护）实现这些常用功能的用户例行程序了。

提供一个工具集合来执行常用功能的原理很类似于已开发操作系统的方式。操作系统提供了一个工具集来访问常用的外围设备，这是使用文件的通用概念。数据库管理系统在更高的级别上也提供了这样的工具：逻辑地访问数据而不是访问记录中的字节位置的工具和使用格式与报表操纵数据的工具。在第三章中，我们将详细讨论这些工具所提供的功能的各种类型。

§1.2 DBMS 的目标

在§1.1 中，我们说明了自计算机早期以来随着开发方法的改进，数据库管理系统是如何取得逐步发展的。在这一节中，我们将主要说明它要达到的目标。

DBMS 技术有三个主要的目标，形成了数据库管理系统技术的基石。这三个目标均是基于独立于硬设备基础上的。其最终的目的是把信息系统的开发与人们通常工作的方式拉得更近一此些，从而使人们避免同计算机的实际工作打交道。

- 物理独立性：主要目的之一是使我们摆脱对物理配置的依赖。从某种角度讲，操作系统使用户从专门的低级存储机构（如磁盘块和柱面）中解脱了出来。在 UNIX 文件系统机构中，UNIX 提供了更高级别的接口进行存储。然而如果我们打算使用 UNIS 的原始磁盘，则仍不得不借助于块或字符在磁盘中管理文件的起始和结束。这种类型的开发工作是相当复杂的，因此很容易出错。对程序员来说，了解逻辑文件和磁盘上的某些块之间的关系是相当困难的。

  在普通文件使用中固有的另一类物理独立性是指将一条记录拆散放入数据字段中。字段必须总是以某一特定的顺序进行存储的。例如，字段 A 从记录的开头开始占用 10 个字节，字段 B 占用字节 11 和 12，等等。程序开发人员不得不仔细地匹配这种位置序列以便获取正确的数据。即便是一个字节的误差，也会造成极为严重的错误。达到这种独立性，可使程序员从令人乏味的工作中解脱出来。

  其它类型的物理独立性可以包括对指定类型终端或打印机的处理。当开发一个应用程序时，程序员必须要了解所支持的物理设备，以便在程序中编写进适当的处理程序。数据库管理系统的格式与报表开发实用程序已经考虑进了这些方面。

- 存取独立性：在使用每种不同类型的文件时，如顺序文件、索引文件和随机文件，在每种文件中，存取程序必须指定如何得到它所需要的记录。例如，假定用户要显示客户信息和特定客户的定单，则完成此项工作的程序首先要打开客户文

件，然后再顺序读取每一个索引记录以确定客户号码，读取记录并对数据进行解释。类似地，它还必须包含要查找的代码并访问代表该客户的客户订单记录。

最关键的一点是，每个程序都必须利用必要的条件来设置检索和存取所需文件的访问路径。适合用户使用某些其它字段（如客户名字）来访问相同数据的程序必须指定一条不同的存取路径。数据库管理系统的目标就是要摆脱这样的存取路径。例如，查询语言就不需要用户说明如何访问一条指定的记录。它们只需要用户说明希望显示什么样的数据即可。

DBMS的这个目标的作用在典型的办公室环境中体现得尤为明显。经理只对秘书提出要查询 J.Smith 在三月十二日的备忘录，他根本没必要详细说明到档案袋中去找等内容。数据库管理系统的目标就是要达到经理提供给秘书这样简单的指令级。

- 数据独立性：数据库管理系统的主要目的就是要减少由于对数据库结构的改变所造成的麻烦，在任何应用中，这是常常发生的事情。在文档数据库中，经理没有了解到填充方法的改变，因此，他发布给秘书的命令也没有任何改变。当用户出于自己使用的需要而在一种格式中加入了一个数据项“接收数据”时，用户不必通知使用该格式的每一个人。事实上，那些对该数据项无兴趣的用户完全可以简单地略过它们即可。

  这就是基于数据库的计算机为什么如此“刻板”以至于存取格式的每一个程序都应在类似环境中进行修改的原因。不幸的是，当数据库存储在普通文件中时，就发生这种情况。

  这样，数据库管理系统的目标之一，就是要向用户或程序隐藏对应用数据库的变化，当然，对那些需要新的数据项的用户和程序例外。其主要目的是要摆脱对存取数据的特定方法的依赖。数据库管理系统的目标就是要向秘书执行经理的命令那样去工作，而不管更多的细节。用户或程序仅引用那些需要的数据项，而忽略那些存在于数据库中的不需要的那些数据项。

数据库管理系统对开发策略的影响还有许多。对数据库管理系统较有益处的一个影响是数据冗余的去除。DBMS 向用户提供了删除冗作的方法，虽然对有的用户而言，这可能有些高深。此外，数据库管理系统并没有这种“智力”来检查并解决在几个地方的数据重复现象。

若要设计一个由多个不同应用共享的数据库，则必须设置一个数据库管理员(DBA)。DBA 负责组织所有应用均可以共享的那些数据。这些职责包括了解整个公司的需要，而不仅仅是针对个别的应用。作为数据库管理员，其责任是很重要的，他负责划分各个应用之间的界限。

由于数据库管理系统可以提高工作效率，因此，读者可能会认为，用户可以管理少一些的程序员了。在实践中，数据库管理系统真正意味着的是，应用开发工作可以在系统测试期间减少一些技术上的困难并在程序修改方面做较少的工作。在一家公司中，侧重点可能会是较短时间的设计和较长一些的代码。数据库管理系统所带来的真正益处并不是针对几个人的，而更多的是针对多个利用相同人数的应用任务的。事实上，用户很可能接受这

样的开发任务，在此之前，没有一个人能够想象得到，如此简单的任务会占用这样长的时间。

如何看待数据库管理系统的价值呢？其真正价值在于节省了信息系统的许多时间，这是它的直接作用。例如，假定一个特定的小型应用任务需要占用 12 人月的时间来开发，而从任务开始，最快也得 6 个月才能完成。我们还假定，使用数据库管理系统，可在三个月时间内完成同样的系统。这样，带给一家公司的益处就是，较以前相比，可以提前三个月利用所需要的信息。在某些情况下，这就意味着提高了竞争能力，可以较其它公司更早地得到重要的信息。

§1.3 DBMS 模型

每一种商品化数据库管理系统都是基于某种数据模型的，数据模型决定了数据库管理系统的基本功能。多数现行的系统，尤其是 UNIX 上的系统，均是基于关系模型的。较早一些的系统使用的数据模型有层次模型和网状模型，现在，这两种数据模型在它们的产品中提供了类关系型的接口。不久的将来，系统还可能使用语义模型或其它的数据模型。我们并不打算深入地讨论关于每种数据模型的理论。我们的做法是，介绍与这些数据模型相关的术语。

数据模型是描述数据项之间关系的一种简单方式，认识到这一点是相当重要的。对用户或程序员来说，它们仅是一个表示出来的视图而已。虽然在数据的内部存储与表示给用户的逻辑视图之间常常存在着某关系，但它们并不是产品用来内部存储数据的必要手段。在本章的剩余部分，我们将说明，如何将个应用数据库模型化为任何一种数据模型，虽然在复杂程度上它们有所不同。这一章的目的是让读者在不经过深入比较的情况下了解它们的区别。

我们的讨论仅限于前面提到过的四种数据模型。出于以下几个原因，我们在本章中没有讨论“反列”（inverted list）数据模型。首先，这是一个相当陈旧的数据模型，诸如 ADABAS、DATACOM / DB 和 MODEL 204 等较为成熟数据库管理系统采用的就是这种数据模型。第二点，虽然采用这种数据模型的数据库管理系统可能会在将来问世，但在 UNIX 下的商品化数据库管理系统没有一种实现了这种数据模型。这些产品所提供的接口是相当低级的。例如，在许多种情况下的索引对程序员是可见的。最后，在“反列”数据模型和许多关系型数据库管理系统的内部实现之间，有许多相似之处。因此，对较高级别的数据视图来讲，对这种数据模型的讨论无太大意义。

在下面的几节内容中，我们将使用图 1–1 所示的示例客户订单系统。下面给出关于这个数据库的一些说明，以了解在这个系统中的一些假定。一个客户可以放置多个订单，每个订单均包含一个订单项目号。对每个订单，我们将发送一个发货清单。客户可以有支持发货清单上的货物总量，因此，我们可能会发送多项发货清单。因此，每个订单都可以有多项发货清单，且每个发货清单均可以有多项支付项目。我们将以所讨论的每种数据模型表达该系统所需的数据库，以说明这些数据模型之间的区别。

重要的是要认识到，特定数据管理系统的数据模型同用户分析用于应用或整个公司的数据之间是毫无关系的。分析工作要涉及对数据的检查以决定它与其它数据项之间如何相关，也就是说，要了解数据。我们所想象的特定的数据库管理系统所执行的分析工作很可

能忽略不适合其数据模型的部分。由于被数据库管理系统的这种特性所迷惑，用户很可能同时漏掉这样的部分数据。在数据库和应用设计中，最好为这些数据标识其模型为找到其工作环境。在数据库设计阶段，程序设计者很可能需要有关特定数据库管理系统更为广泛的知识。在第二章中将要介绍的数据标准化的概念在数据分析和设计阶段均是很有用途的。这些概念不仅可应用于关系数据库模型，还可以应用于所有的其它数据模型。

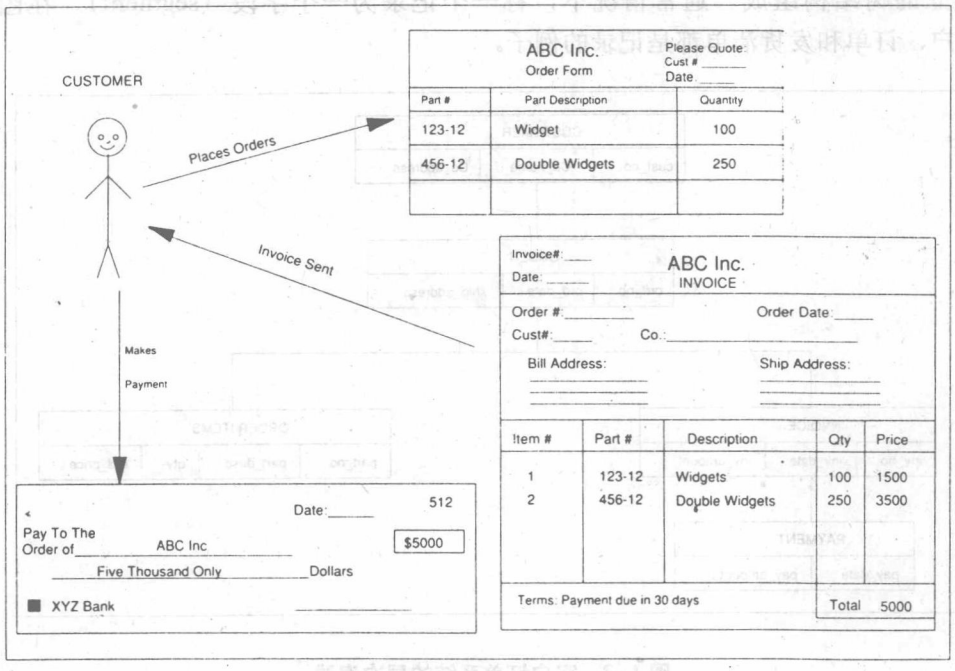


图 1–1　ABC__Inc 的客户订单系统

有许多工具来帮助程序设计者进行数据分析和设计。这些工具均可以归结到计算机辅助软件工程（CASE）工具中去。许多工具将同用户所喜欢的符号一起工作并提供用户理解数据的图示表达方法。我们应该认识到，为搜集第一手的信息，我们仍需要较为熟练的分析技巧。然而，图示则是消除数据分析人员和课题专家之间误解的良好沟通工具。程序设计者将发现这些工具是很实用的，其特性就是可由数据模型产生出的数据库模式。一些工具可以产生用于几种不同数据库管理系统产品的适当文字化的模式生成语句。

这些 CASE 工具的其它方面具有和应用设计中相类似的使用，尤其是当使用数据库管理系统的第四代开发技术时。然而，应当引起我们注意的是，CASE 工具的多数代码生成成份只生成第三代程序设计语言代码，如 COBOL。这些典型的工具不包括对格式驱动程序、报表生成器或第四代语言的接口。由于许多数据库管理系统销售商进入了 CASE 领域，我们很可能在不久的将来看到更好的工具出现。

§1.4　层次模型

这种模型是首先得到实现的，但很可能也是最有限的。这种数据模型的最著各的实现是 IBM 公司的 IMS 数据库管理系统。在 UNIX 环境中，只有那些从它们最初的

Mainframe 环境中分离出来的 DBMS 产品才使用该数据模型。主要的例子是 Information Builders Inc.的 FOCUS。当然，这样的产品提供了关系接口的某些格式（通常是在查询语言中）。

顾各思义，这种数据模型要求将数据安排成层次型的，就好象树的结构或组织机构流程图一样。图 1–2 给出了以层次方式安排示例应用数据库的一种方式。一个数据库可由多个独立的树结构组成。通常情况下，称一个记录为一个字段（segment）。在图 1–2 中，客户、订单和发货清单都是记录的例子。

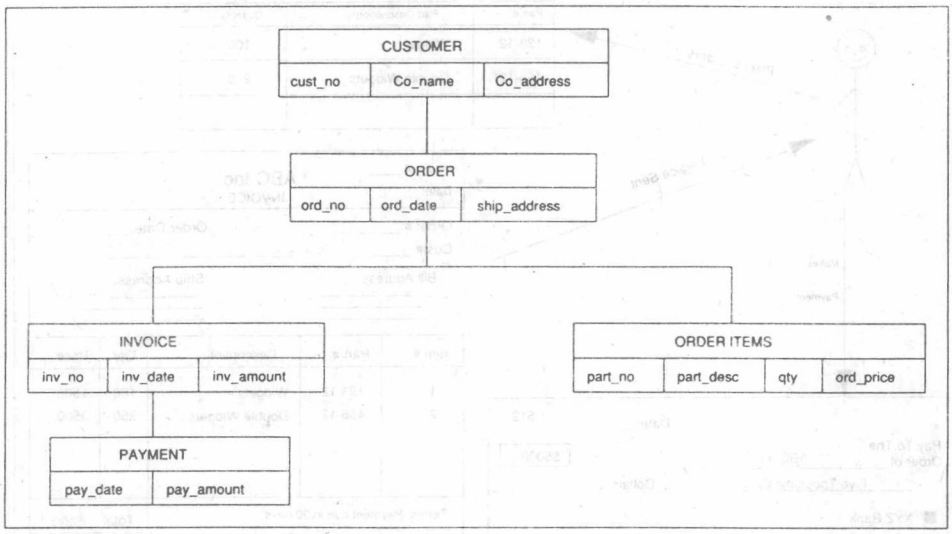


图 1–2　客户订单系统的层次表述

图 1–2 中，将字段链接到了一种“父子”关系中，这种关系是由连接字段的线来表示的。这样，在我们的例子中，字段 CUSTOMER 为子字段 ORDER 的“父子”字段。这种链接通常表达了一种“一对多”的联接关系。也就是说，对每一个客户均可以有多个订单，但对任何一个订单，则只能有一个客户。很清楚，也可以很容易地表达一对一的联接。一个字段的所有“例化”（instance）均作为一个“已定订单”的集合加以存储，只能以指定的顺序对它们进行访问。在层次结构的数据库管理系统中，联线和字段的顺序对于存取方法而言，起着重要的作用。

根据对这个例子的分析，我们应当清楚，多对多的关系是很难以这种模型表述出来的。假定在我们的例子中，一各客户支付了些发货清单的帐目，则在目前的数据结构中，我们还不可能接收这样的支付，因为这已将“支付”和“发货清单”之间的关系改变成了“多对多”的关系。显然，我们不可能将支票返回，因为我们的数据库还没有设计能够处理这种情况。一些软件产品，如 IMS，允许子字段有两个父字段以便能够描述这种情况。但这样仅仅是延缓了问题的发生，并没有解决什么。例如，如果我们还需要包括帐户中的支付这一应用，又怎么办呢？当然，这种情况是指一个帐户可以拥有许多次支付。

请注意在片段 order__item 中字段 ord__no 标识的缺省。这种识别实际上是由链本身来完成的，所以我们不必保存这个标识值了。这样，用户可以使特定的订单项目与一个订

单产生联系的唯一途径就是通过这条链了。链信息是当用户加入订单项目的例化时由数据库管理系统产生的。这样，当用户加入子片段的例化时，必须具有适当的父片段例化来作为当前的父片段。由于子片段的例化是有一定顺序的，所以还必须选择适当的点来指明子例化的位置以插入一个新的例化。

在层次结构中的存取方式是面向记录的，也就是说，通过查找第一个记录开始，然后顺序访问每一个后续的记录。诸如 FOCUS 的“修改成分”这样的最终用户工具在处理时隐藏了这个记录，但程序设计接口显然是面向记录的。在层次结构的数据库中，可以通过三种方式存取记录：直接地、顺序地或在当前父记录下的顺序存取。

直接存取方法涉及到对记录关键字使用选择条件。例如，在“发货清单”片段中，可以检索特定的发货清单号。顺序存取方法则只需简单地从第一条记录开始并使用下一条记录命令顺序访问下一个记录。在当前父片段下的顺序存取方法则利用了在一个父片段中子记录中的次序。这种方法对于存取一个父片段的特定例化的所有子片段均是可用的，例如，在一个特定订单中的所有订单项目。

在当前父片段下的顺序访问方法的概念还引入了“祖先”的概念，即父片段的父片段。例如，我们可以通过使用在当前父片段下的顺序存取方法来访问特定客户订下的特定货物处的所有例化，虽然这个客户字段是订单项字段的祖先。

§1.5 网状模型

网状数据模型可认为是层次数据模型的一个变种。网状数据模型优于层次数持模型的地方在于，其结构并不一定必须是树形结构。这种数据模型的 UNIX 实现实例是 AT&T TUXEDO 产品。虽然本书中所讨论的四种 UNIX 数据库管理系统无一使用了网状模型，但 UNIFY 却在其 explicit reference 机构中提供了类似的概念。关于网状数据模型的实现，较好的一个例子是 Cullinet 的 IDMS 产品。

图 1-3 将我们的示例数据库表述成了网状数据模型。联接（在网状模型中，术语是集合）可存在于任何两个记录之间。在这种情况下，记录之间的联接表达了一种“成员”关系，同层次数据模型中的“父子”关系一样，这种“成员”关系也是网状数据模型中必不可少的。关于两个记录的定义集合的限制如下：

- 在一个集合中的成员记录的一个例化只能在该集合的一次出现中出现0次或1次。
- 在一个集合中的上层记录的一个例化只能在该集合的一次出现中出现一次。

在实践中，这些限制意味着，一个记录的单一例化不能作为一个上层记录的多于一个例化的成员。这样，就象在层次模型中一样，我们不得不为两个包含同样货物号的订单增加一个订单项目记录的区别性例化。通过定义集合，我们可以表述一对一和一对多的关系。表达两个记录之间多对多的关系需要创建第三个记录并定义一个该记录类型和先前两个记录的每个记录之间的集合。

有三种访问记录的基本方式：直接存取、使用集合定义的顺序存取和简单的顺序存取。直接存取依赖的是在数据库模式中的 calc 码定义。这种码经常转换成哈希存取方法（将在第三章中进行讨论）。使用一个集合定义的顺序存取称为 Via，它基本上是一种导航

式存取方法。它依赖于同在成员记录的例化中隐含的次序一起的两个记录之间联接的使用，这些成员记录是属于上层记录的当前例化的。在我们的示例数据库中，我们可能要使用 calc 码 cust_no 来生成当前的所需客户记录，然后再使用客户一订单集合来逐个访问用于该客户的订单，对每个订单使用订单项目集合来顺序访问订单项目。

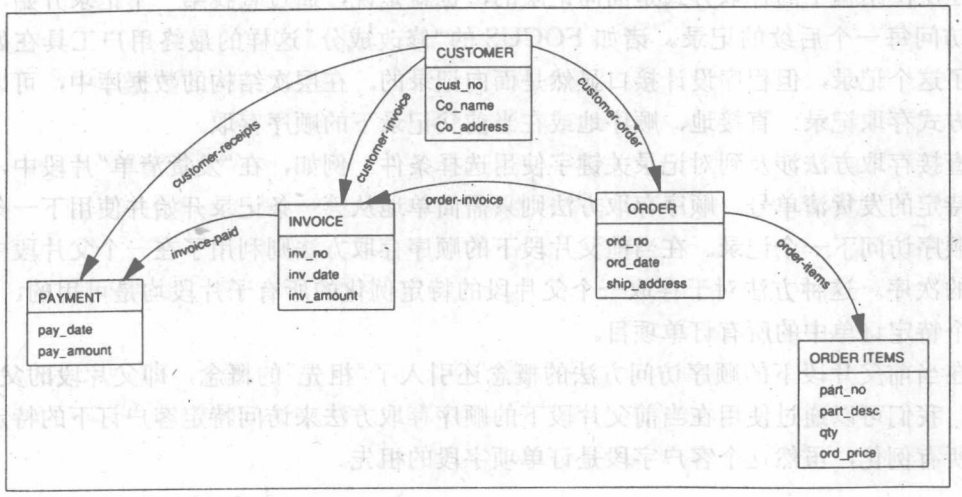


图 1-3　客户订单系统的网状模型表达

当对一个记录加入一个新的例化时，数据库管理系统通常小心地更新与该记录有关的集合。当然，重要的是要维护成员例化的顺序。顺序可能是基于按字段值排序的、先进后出的、后进先出的或简单地留给应用程序来处理。在最后一种方法中，应用程序必须过程性地定义使新的例化连接到的点。

决定那一个集合是必要的以及决定对每个记录类型的存取方法是一个复杂的过程。在各种应用的需要之间常常发生冲突且对数据库的重新配置也决不是简单直接的。当然，解决冲突这一较为繁重的任务就落到了数据库管理员的身上，应用程序员常常希望摆脱这个繁重的任务。

同层次模型不同的是，网状模型是由 CODASYL 数据库任务组（DBTG）于 1971 年提出的。根据这些提出的定义，在七十年代，网状模型得到了较大的发展。

§1.6　关系模型

这种数据模型是最易于理解的，在八十年代，这曾经是最为流行的一种数据库数据模型。本书中将要讨论的所有 DBMS 产品均是基于关系模型的，只是在实现上的级别的所变化。关系数据模型的理论是基于 E.F.Codd 博士在七十年代和八十年代初期的研究工作的基础上提出来的。数据库管理系统产品的商业化实现很可能是最适于 UNIX 环境的。

关于这种数据模型，目前已有多种专著来对此加以讨论和研究，最著各的是由 C.J.DATE 编著的《An Introduction to Database System》。在第二章中，对在关系数据库环境中使用的专门术语进行了更加详尽的论述。这一节仅向读者介绍一个关系数据模型的大概情况。本章中的其余部分，我们将说明我们的示例数据库的关系型表达方式，