第2章　数据库系统体系结构

2.1　引言

本章介绍数据库系统的体系结构。介绍体系结构的目的是给后续章节建立一个框架结构。这个框架结构用于描述一般数据库的概念，并解释特定数据库的结构——但不能说每个数据库系统都和这个框架结构完全相匹配，或者说这一特定的体系结构提供了唯一可能的框架结构。特别是，“小”系统（见第1章）将难以支持体系结构的各个方面。不过，此体系结构基本上能很好地适应大多数系统；而且，它基本上和 ANSI/SPARC DBMS研究组提出的数据库管理系统的体系结构（称作ANSI/SPARC体系结构——参见[2.1~2.2]）是相同的。但是，我们不在每个细节部分都采用ANSI/SPARC的术语。

注意：本章和第1章类似，对本章内容的理解有助于对现代数据库系统的结构和功能有一个较全面的认识，但本章的内容还是有些抽象和枯燥。因此和第 1章一样，读者可以先对这些内容“大致浏览”一下，待以后遇到直接相关的内容后再回过来看这部分内容。

2.2　三级体系结构

ANSI/SPARC 体系结构分为三层：即内模式、概念模式和外模式（见图 2-1）。广义地讲：

- 内模式（存储模式）是最接近物理存储的——也就是，数据的物理存储方式；
- 外模式（用户模式）是最接近用户的——也就是，用户所看到的数据视图；
- 概念模式（公共逻辑模式，或有时称逻辑模式）是介于前两者之间的间接的层次。

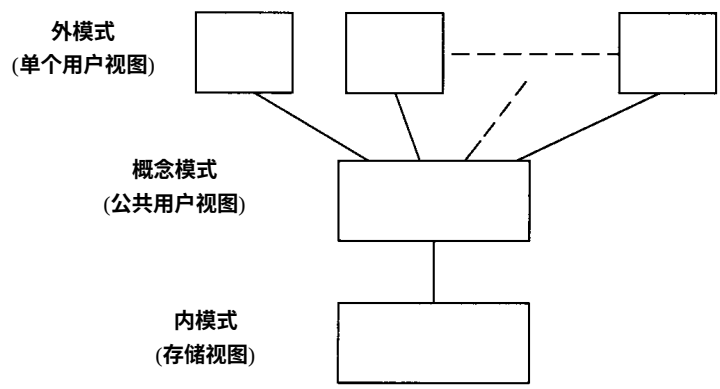

图2-1　三级体系结构

注意到外模式是单个用户的数据视图，而概念模式是一个部门或企业的数据视图。换句话说，“外部视图”（即外模式）会有许许多多，每一个都或多或少地抽象表示整个数据库的某一部分，而“概念视图”（概念模式）只有一个，它包含对现实世界数据库的抽象表示⊖。

⊖　这里所说的抽象表示是像记录和字段这些更加面向用户的概念，而不像位和字节那些面向机器的概念。

（大多数用户只对整个数据库的某一部分感兴趣）。同样，“内部视图”（即内模式）也只有一个，表示数据库的物理存储。

举例说明如下。图2-2给出了一个有关人事数据库的概念视图，对应的内部视图和两个对应的外部视图（一个为PL/I用户，一个为COBOL用户）。当然，例子完全是假设的——与任何实际系统无关——且忽略了许多无关的细节。说明如下：

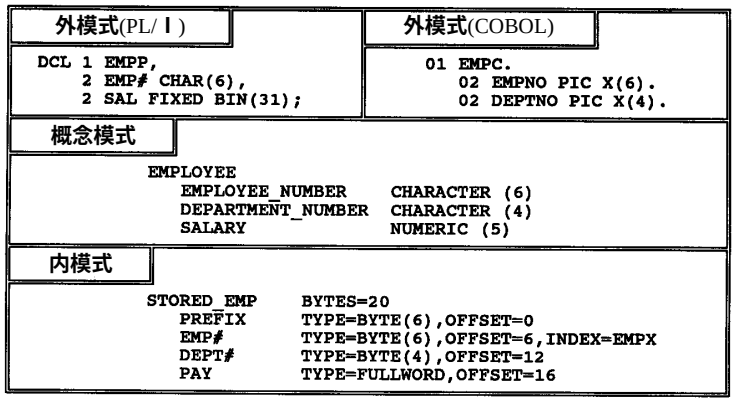

图2-2 三级结构举例

- 在概念模式中，数据库包含了EMPLOYEE（雇员）的实体类型的信息。每个雇员都有EMPLOYEE_NUMBER（6个字符），DEPARTMENT_NUMBER（4个字符）和SALARY（5位的十进制数）。
- 在内模式中，雇员由长度为20字节、名称为STORED_EMP的存储记录类型来表示。STORED_EMP包含四个字段：6字节的前缀（大概包含如标记或指针这样的控制信息），和相应于雇员的三个属性的三个数据字段。此外，STORED_EMP记录按雇员号字段进行索引，索引名为EMPX，索引的定义随后给出。
- PL/I用户对应一个数据库的外部视图，其中，每个雇员由一条包含两个字段的PL/I记录来表示（部门号对该用户没有意义，故已经省略）。记录类型是根据PL/I的规则由通常的PL/I结构声明来定义的。
- 类似地，COBOL用户也对应一个外部视图，其中，每个雇员也由包含两个字段的COBOL记录来表示（这次，工资记录被省略）。记录类型是根据COBOL的规则由通常的COBOL记录描述来定义的。

注意，在不同视图中相应的数据项可有不同的名字。例如，雇员号在PL/Ⅰ的视图中称为EMP#，而在COBOL的视图中称为EMPNO，在概念视图中称为EMPLOYEE_NUMBER，而在内部视图中又称为EMP#。当然，系统必须知道它们之间的关系；例如，要告知系统COBOL的EMPNO字段来自概念字段EMPLOYEE_NUMBER，而EMPLOYEE_NUMBER又来自于内部视图的存储字段EMP#。这种关系或映射，在图2-2中并未清楚地表现出来；可参见2.6节的进一步的讨论。

本章所谈论的内容对系统是不是关系的，差别不大。不过，简要说明一下关系系统中三级体系结构的情况，对具体理解这一概念不无裨益：

- 首先，关系系统的概念模式一定是关系的，在该层可见的实体是关系的表和关系的操作符（尤其包括第1章中提到的两个操作符选择RESTRICT和投影PROJECT）。

- 第二，外部视图也是关系的或接近关系的；例如，在图 2-2中，PL/I和COBOL的记录定义可以被当作与关系系统的关系表相似的 PL/I和COBOL的声明。
  注意：这里应指出“外部视图”一词（有时简称为视图），在关系系统中有其特定的含义，它与本章提到的含义有所不同。有关它在关系系统下的含义见第3章和第9章的解释。
- 第三，内模式不是关系的，因为该层的实体不是关系表的原样照搬。其实不管是什么系统，其内模式都是一样的（如存储记录，指针，索引，哈希表，等等）。事实上，关系模型与内模式无关，如第 1章指出的，它关心的是用户的数据视图。

现在我们从外模式开始进一步讨论三层结构的细节。整个的讨论都以图 2-3为基础。该图显示了体系结构的主要组成部分和它们之间的联系。

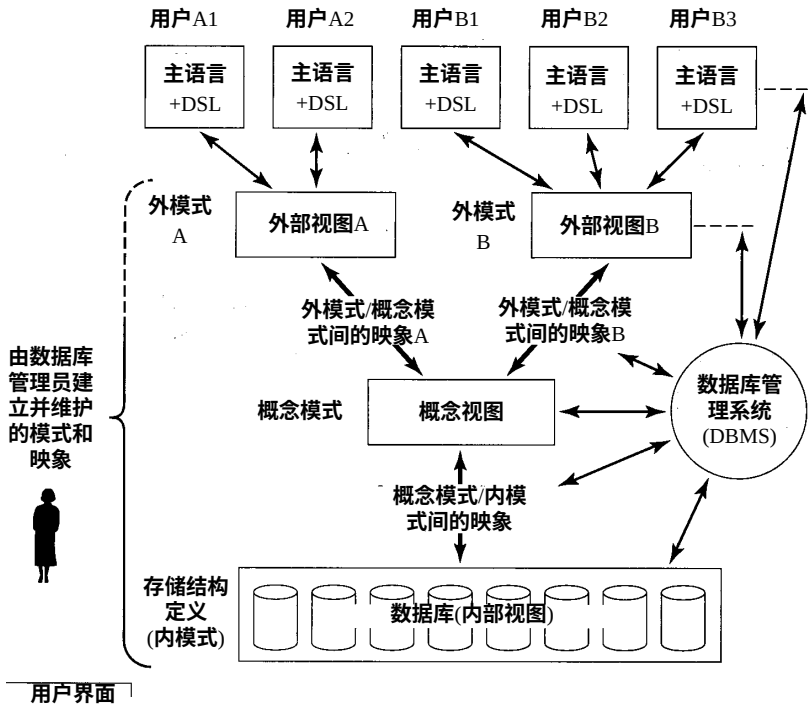

图2-3 系统体系结构

2.3 外模式

外模式就是单个用户的数据视图。如第 1章所述，一个用户可以是应用程序员或任一最终用户（这里DBA是一个特例，与其他用户不同， DBA还要了解概念模式和内模式。详见下面两节）。

每个用户都有其自己要使用的语言：

- 对于应用程序员，可能会使用常规的编程语言（如 PL/I、C++、Java）或其他专用语言。这些专用语言通常被称作“第四代”语言（ 4GL），这是由于（a）机器语言、汇编语言和像PL/I这样的语言被作为第一、二、三代语言；（ b）专用语言是对“第三代”语言(3GL)的发展，就像第三代语言对汇编语言的发展和汇编语言对机器语言的发展一样。
- 对于最终用户，其使用的语言或者是一种查询语言或是某一特定目的的语言，这些语言

可能是表格驱动的或菜单驱动的，可处理用户的请求并由在线的应用程序来支持。

重要的是所有这些语言都包含数据子语言 —— 即数据库对象和操作的整个语言的一个子集。数据子语言嵌入在相应的主语言中。主语言负责提供各种非数据库的功能，如局部变量、计算操作、逻辑分支，等等。一个指定系统可以支持多种主语言和多种数据库子语言；但是，目前大多数系统支持的特定的数据子语言是 SQL语言，它在第1章已经简要地介绍过。多数系统允许SQL 既可以作为独立的交互查询语言，又可嵌入到诸如 PL/I 或Java等主语言中（见第4章进一步讨论）。

尽管区分数据子语言和包含它的主语言对体系结构来说是方便的，但对用户来说，两者实际上没那么大区别；的确，从用户的观点来看，他们宁可两者没有区别。如果没有区别，或很困难才能区别出来，我们称两者是紧耦合。如果很容易清楚地区分，我们称两者是松耦合。然而只有少数的商业系统（尤其是对象系统 —— 见第24章）支持紧耦合，多数系统不支持；尤其是 SQL只支持松耦合（紧耦合提供给用户一套统一方便的功能，但很显然，系统开发者在这一部分花了很大功夫）。

理论上，任何特定的数据子语言至少包含两个子语言 —— 数据定义语言（DDL）和数据操纵语言（DML）。数据定义语言支持对数据库对象的定义或说明；数据操纵语言支持对这些对象的操作和处理。例如，在 2.2节的图2-2中的PL/I用户的情况。该用户的数据子语言包括用于与DBMS通信的那些PL/I的特征：

- 数据定义子语言部分包括 PL/I用于数据库对象的声明的构造 —— 声明（DECLARE）语句本身，某些PL/I数据类型，可能对PL/I的扩展以支持现有的PL/I不能处理的新对象。
- 数据操纵子语言部分包括 PL/I可执行的语句，这些语句完成与数据库的信息传递 —— 也可能包括特殊的新语句。

注意：准确地说，在编写本书时， PL/I实际上根本不包括任何特殊的数据库特征。典型情况下，DML语句只是PL/I 用于调用DBMS的CALL语句（尽管那些语句以某种方式掩盖了语法结构以使得对用户更友好 —— 见第4章嵌入式SQL）。

再来看体系结构：我们已经指出，单个用户只对整个数据库的某些部分感兴趣；而且，与数据的物理存储方式比较而言，其用户视图通常有些抽象。 ANSI/SPARC对用户视图的术语为外部视图。外部视图就是特定用户所看到的数据库的内容（即，对那些用户来说，外部视图就是数据库）。例如，人事部门的用户可能把部门和雇员记录值的集合作为数据库，而没有意识到采购部门的用户所看见的供应商和零件的记录值。

通常，外部视图包括许多外部记录类型的值（不必和存储记录一样）[⊖] 。用户数据子语言是根据外部记录来定义的；例如，数据操纵语言的检索操作将检索到外部数据记录值，而非存储记录的值。注意：在第 1章提到的“ 逻辑记录 ”一词实际上指的是外部记录。在以后的讨论中将会尽量避免使用“ 逻辑记录 ”一词。

每个外部视图都是通过外模式来定义的，外模式包括外部视图中的各种外部记录类型的基本定义（参见图 2-2的几个简单的例子）。外模式是使用用户数据子语言的 DDL部分来写的

⊖ 这里我们假定在外模式中所有的信息都是以记录的形式来表示的。但是，有些系统也允许以其他的方式来代替，或也用记录表示信息。对于使用其他方式的系统，本节给出的定义和解释会需要做相应的改变。类似的注意事项也适用于概念模式和内模式。这种情况的细节已超出本书这部分的范围；进一步的探讨见第13章（特别是“ 参考文献和简介 ”一节）和第 24章。

(因此有时DDL也当作外部数据定义子语言)。例如，雇员外部记录类型就可以定义为6个字符的雇员号字段加5位十进制数的工资字段等等。此外，在外模式和其下面的概念模式之间要定义映象（见下一节)。我们将在2.6节讨论映象。

2.4 概念模式

概念视图表示数据库的全部信息内容，其形式要比数据的物理存储方式抽象些。通常，它与任何特定用户观察数据的方式都很不同。广义上讲，概念视图更接近于实际数据，而不像某一用户所看到的数据，这些数据受到特定语言或可能使用的硬件限制。

概念视图由许多概念记录类型的值构成。例如，它可能包括部门记录值的集合，雇员记录值的集合，供应商记录值的集合，零件记录值的集合（等等)。概念记录既不和外部记录相同，也不和存储记录相同。

概念视图是由概念模式定义的。概念模式包括各种概念记录型的定义（见图 2-2中的简单例子)。概念模式是用另一种数据定义语言来写的，即概念数据定义语言。如果可以实现物理记录的独立性，那么概念视图根本不涉及物理表示和访问的技术 —— 它们只定义信息的内容。这样，在概念模式中不能涉及存储字段表示、存储记录队列、索引、哈希算法、指针或其他存储和访问的细节。如果概念视图以这种方式真正地实现数据独立性，那么根据这些概念模式定义的外模式也会有很强的独立性。

概念视图是整个数据库内容的视图，概念模式是该视图的定义。但如果把概念模式只理解为类似COBOL程序中简单的记录定义一样的一组定义，那是不准确的。在概念模式中的定义应包括许多额外的特征，诸如第 1章中提到的安全性和完整性约束。到目前为止，有些权威人士认为概念模式的根本目的是描述整个企业的情况 —— 不只是数据本身，而且还包括数据的使用情况：即数据在企业中的流动情况，在每一部门的用处，以及对它实行的审计和其他控制，等等 [2.3]。但必须强调的是，目前的系统实际上还不能支持这种程度的概念模式；目前大多数系统支持的“概念模式”实际上只不过是把单个的外模式合并起来，再加上了一些安全性约束和完整性约束。但是将来的系统很可能在支持概念模式上会更加复杂。

2.5 内模式

体系结构的第三层是内模式。内部视图是整个数据库的低层表示；它由许多内部记录型中每一类型的许多值组成。“内部记录”是ANSI/SPARC对存储记录的称谓（我们继续使用后者)。内部视图与物理层仍然不同，因为它并不涉及物理记录的形式 —— 即物理块或页 —— 也不考虑具体设备的柱面或磁道大小。换句话说，内部视图假定了一个无限大的线性地址空间；地址空间到物理存储的映射细节是与特定系统有关的，它未反映在体系结构中。注意：块或页是输入/输出的单位 —— 也就是说，在一次输入/输出操作中，二级存储和主存之间传输的数据量。典型的页面大小是1K、2K或4K字节（K=1024)。

内部视图由内模式来描述，内模式不仅定义各种存储记录，而且也说明存在什么索引，存储记录怎么表示，存储记录是在什么物理队列中，等等（见图 2-2中的简单的例子)。

内模式是用另一种数据定义语言 —— 内部数据定义语言来写的。注意：本书中我们通常使用更直观的词“存储结构”或“存储的数据库”代替“内部视图”，用“存储结构定义”代

替“内模式”。

最后指出一点，在某些特殊的情况下，应用程序——尤其是实用程序——可能会直接操作内模式而不是外模式。当然，这种做法肯定是不妥的，这会带来一定的安全性和完整性隐患（即安全性约束和完整性约束会被绕过），并且应用程序会失去数据独立性。但有时为了保证功能或性能上的要求，又不得不这样做。这就像使用高级语言系统的用户，偶尔会使用汇编语言以满足特定的功能或性能上的要求一样。

2.6 映象

除了三级模式本身，图2-3中的体系结构还含有一定的映象关系——即概念模式/内模式间的映象和外模式/概念模式间的映象。一般地讲：

- 概念模式/内模式的映象定义了概念视图和存储的数据库的对应关系；它说明了概念记录和字段在内部层次怎样表示。如果数据库的存储结构改变了——也就是说，如果改变了存储结构的定义——那么概念模式/内模式的映象必须进行相应的改变，以便概念模式能够保持不变（当然，对这些变动的管理是数据库管理员的责任）。换句话说，为了保持数据的物理独立性，内模式变化所带来的影响必须与概念模式隔离开来。
- 外模式/概念模式间的映象定义了特定的外部视图和概念视图之间的对应关系。一般地讲，这两层之间存在的差异情况与概念视图和存储模式之间存在的差异情况是类似的。例如，字段可能有不同的数据类型；字段和记录名可以改变；几个概念字段能合成一个单一的字段，等等。可能同时存在多个外部视图；多个用户可共享一个特定的外部视图；不同的外部视图可能有交叉。

注意：很显然，就像概念模式/内模式的映象是物理独立性的关键，外模式/概念模式的映象是逻辑独立性的关键。如在第1章中所述，如果用户和用户的应用程序能相对于数据库物理结构的改变而保持不变，系统就提供了物理独立性[1.3]。同样地，如果用户和用户的应用程序对于数据库逻辑结构的改变（即在概念或公共逻辑层的改变）能保持不变，系统就提供了逻辑独立性[1.4]。在第3章和第9章还会就这一重要问题做进一步的说明。

此外，多数系统允许定义以其他形式表示的某些外部视图（通过外模式/外模式间的映象），而不总是要求一个明确的到概念层的映射定义。这是一个很有用的特征，特别是当几个外部视图相互非常类似时。关系系统一般支持提供这种能力。

2.7 数据库管理员

如第1章中所述，数据管理员是根据企业的数据制定策略和政策决策的人，数据库管理员对执行这些决定提供必要的技术支持。因此，数据库管理员负责在技术层的全局控制。我们可以更详细地描述数据库管理员的任务。通常，数据库管理员的任务至少包括下列内容：

- 定义概念模式　数据管理员的工作是决定数据库中存放的信息——换句话说，是确定对企业有用的实体和实体的相关信息。这一过程通常是指数据库的逻辑设计（有时也称概念设计）。一旦数据管理员在抽象的层次上决定了数据库的内容，数据库管理员就使用概念数据定义语言创建相应的概念模式。模式的目标形式（已编译的）由数据库管理系统在响应访问要求时使用。源形式（未编译的）作为系统用户的参考文档。

  注意：在实际中，事情不像前面所说的那样能够清楚地区分。某些情况下，数据管

理员可能直接创建概念模式。另外，数据库管理员也可以做逻辑设计。

- 定义内模式　DBA也决定数据库中数据表示的问题。这一过程通常叫做物理设计[㊀]。完成物理设计之后，DBA必须使用内部数据定义语言创建相应的存储结构定义（内模式）。此外，DBA必须定义相应的概念模式/内模式的映象。实际上，概念数据定义语言或内部数据定义语言——更多的可能是指前者——会包括定义映象的方法，但是两方面功能（创建模式，定义映象）要清楚地分开。像概念模式一样，每个内模式和相应的映象都会以源形式和目标形式存在。
- 与用户联络　为了确保用户需要的数据可用，并可以使用外部数据定义语言来编写（或帮助用户写）必要的外模式，DBA要负责与用户的联络（如前所述，一个指定的系统可以支持几个不同的外部数据定义语言）。此外，相应的外模式/概念模式的映象也需要定义。实际上，外部数据定义语言会包括说明映象的方法，但是，模式和映象要清楚地分开。每个外模式和相应的映象都以源形式和目标形式两种方式存在。

  与用户联络的其他方面还包括考虑应用程序的设计；提供技术培训；帮助决定问题和解决问题；及类似的专业服务。
- 定义安全性和完整性约束　如前所述，安全性和完整性约束可以当作概念模式的一部分。概念数据定义语言包括说明这些约束的功能。
- 定义转储和重载机制　一旦企业采用了数据库系统，它就非常依赖于对系统的正确操作。如果数据库的任何部分遭到破坏——人为引起的或硬件错误或系统错误——必须能够以最小的代价恢复数据，且尽量减小对系统的影响。例如，未被破坏数据的访问并不受影响等。DBA必须定义和执行一个恰当的破坏控制计划。这些计划典型地应包括预先的对数据库卸载或转储到备份存储设备上并且当需要时从最近的转储中恢复数据库。

  顺便提一下，快速恢复数据的需求是将整个数据分散到几个数据库中而不是全都保存在一个数据库中的主要原因；单个数据库可能是转储和重载的单元。在这个关系中，注意TB级系统[㊁]——即存储了吉字节数据的商业数据库——已经存在，而且未来的系统的数据量会更大。毫无疑问，这种超大规模数据库系统(VLDB)要求非常精细和复杂的管理，尤其是要求系统不间断运行的时候。然而，为了讨论上简便起见，我们仍只考虑单一数据库的情况。
- 监控系统性能并响应不断变化的请求　在第1章中提到，DBA负责组织系统，以得到对企业最佳的性能，并根据需求的改变来做相应的调整(或调节)。例如，可能要不时地对数据库进行重组织以确保其效率。任何对系统物理存储层的改变都要伴以相应概念模式/内模式的映象定义的改变，以便概念模式保持不变。

当然，以上并非详尽的描述——只是试图对DBA的职责范围和特点给出一些想法。

2.8 数据库管理系统

数据库管理系统(DBMS)是处理数据库访问的软件。从概念上说，它包括以下处理过程(参见图2-3)：

㊀ 注意顺序：首先决定想要的数据，然后决定数据的存储表示。物理设计应该在逻辑设计之后。

㊁ 1024字节=1KB，1024KB=1MB，1024MB=1GB，1024GB=1TB，1024TB=1PB, 1024PB=1EB。注意，不严格地讲，1GB(gigabyte)是十亿字节（有时也用BB来代替GB）。偶尔——与通常相反——gigabyte发音是开头字母g的发音。

1) 用户使用某数据子语言（如SQL）发出一个访问请求。

2) DBMS接受请求并分析。

3) DBMS接下来检查用户外模式(目标形式)、相应外模式/概念模式的映象、概念模式、概念模式/内模式的映象和存储结构定义。

4) DBMS执行对数据库的必要的操作。

通过一个例子来看一下检索一个特定外部记录值的过程。通常，要从几个概念记录值得到字段，而每个概念记录值反过来又需要来自几个存储记录值的字段。概念上， DBMS首先检索所有要求的存储记录的值，然后构造所要求的概念记录值，接着再构造所要求的外部记录值。在每个阶段可能需要数据类型或其他方面的转换。

当然，前面的描述非常简单；特别是，这暗示整个过程是解释性的，因为它表明分析请求的处理，检查各种模式等等都是在运行时做的。反过来，解释意味着执行效率低，因为运行时才解释。为此，实际中可能对访问请求在运行之前先进行预编译（尤其是，目前 SQL产品一般支持这一点——参见第4章[4.12]和[4.26]）。

下面将详细地解释一下DBMS的功能。这些功能至少支持下列内容：

- 数据定义 DBMS必须能接受数据定义的源形式，并把它们转化成相应的目标形式。换言之，DBMS必须包括支持各种数据定义语言(DDL)的DDL处理器或编译器。在某种意义上，DBMS必须理解DDL定义，例如，它理解EMPLOYEE外部记录包括SALARY字段；并能够利用这一知识来分析和响应数据操纵要求(例如，“找出薪金小于50000美元的雇员”)。
- 数据操纵 DBMS必须能够检索、更新或删除数据库中现有数据，或向数据库增加数据。换句话说，必须包括处理数据操纵语言（DML）的DML处理器或编译器。
- 通常，DML请求可以是“计划”的或“非计划”的：
  a) 一个计划的请求是指在请求执行前就能预见到有关的需求。 DBA可以据此调整物理数据库的设计以保证请求有好的执行性能。
  b) 相反，一个非计划的请求是指需求是不可预知的，是一种特殊查询。物理数据库的设计不一定能够真正适合处理这样的请求。

  使用第1章中的术语，计划的请求是“操作型”或“生产型”应用的特征。而非计划的请求是来自“决策支持”类应用。计划请求通常来自预先写好的应用程序，而根据定义，非计划的请求是通过某查询语言处理器交互发出的。

  注意：如第1章所述，查询语言处理器是一个内置的在线应用程序，而不是 DBMS本身的一部分。为了完整，我们把它包括进图2-4中。
- 优化和执行 计划的或非计划的数据操纵语言请求必须经过优化器部件的处理，优化器是用来决定有效执行请求的方式。有关优化问题将在第17章进行详细讨论。优化后的请求在运行管理器的控制下执行。注意：在实际操作中，运行管理器调用文件管理器来访问存储的数据。有关文件管理器的内容将在本节的末尾进行详细讨论。
- 数据安全性和完整性 DBMS要监控用户的请求，拒绝那些有破坏DBA定义的数据库安全性和完整性约束企图的请求。在编译时或运行时或两种情况同时存在时都会执行这些任务。
- 数据恢复和并发 DBMS——或其他一些相关的软件，通常称作事务管理器或事务处理

监控器——必须保证要有恢复和并发控制。该细节内容已经超出了本章的范围；详细讨论见本书的第五部分。注意：事务管理器没有在图 2-4中出现，因为它并不是 DBMS的组成部分。

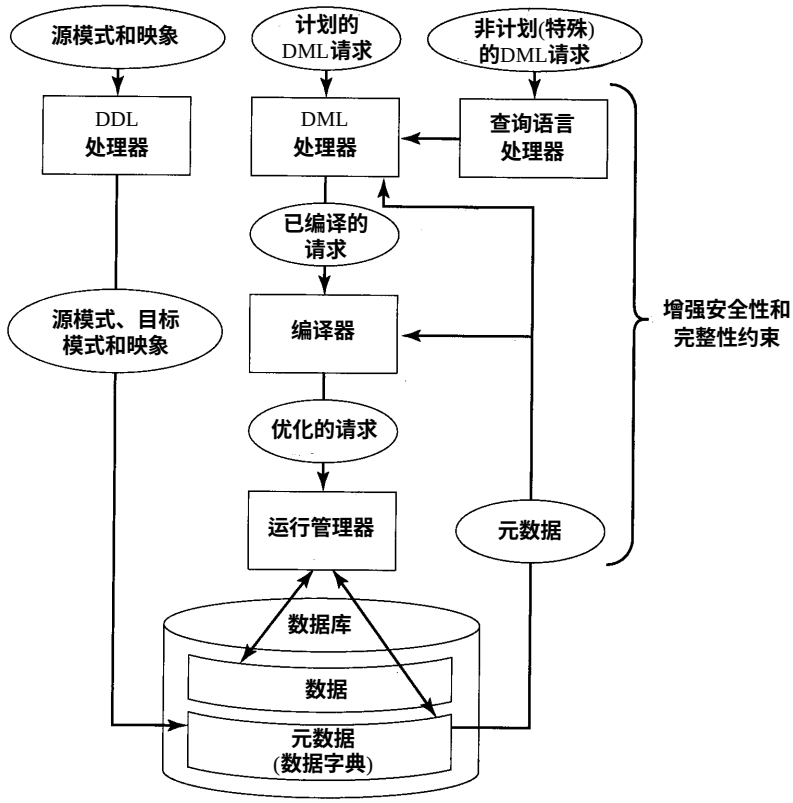

图2-4 DBMS的功能和组成

- **数据字典** DBMS包括数据字典。数据字典本身也可以看作是一个数据库（系统数据库而不是用户数据库）。字典是“数据的数据”（有时称为数据的描述或元数据）——即系统中其他实体的定义，而不只是“原始数据”。特别地，在数据字典中，各种模式和映象和数据的各种安全性和完整性约束都可以用源和目标两种形式来存储。一个易用的字典还包括许多其他信息，例如，给出哪个程序用数据库的哪个部分，哪个用户使用哪个报表，等等。数据字典甚至还可以统一于它定义的数据库中，因而包括它自己的定义。当然查数据字典和查其他数据库是相同的，因此，例如，可以分出哪个程序或哪个用户会受到系统改变的影响。详细讨论见第 3章。

  注意：在这一领域有许多术语相互混淆。有些人把数据字典称作目录或分类——隐含目录比真正的数据字典处于更内层——而用字典一词指某种应用开发工具。有时还用数据存储池(见第13章)或数据百科全书来指代后者。
- **性能** 毫无疑问，DBMS应尽可能高效地完成上述任务。

总而言之，DBMS的目的就是提供数据库的用户接口。用户接口可定义为系统的边界，其下的细节对用户来说是不可见的。因此，根据定义可知，用户接口是外模式。不过，在第 9

章中您将会注意到，有时外部视图与它下面的概念视图的相应部分不可能完全分清，至少目前的商用SQL产品还不能将外部视图完全从下面的层次中独立出来。

以下通过简单地对比数据库管理系统和文件管理系统（文件管理器或文件服务器）来小结本节。文件管理器是操作系统管理文件的部件；它比 DBMS更“接近磁盘”（事实上，DBMS通常是建立在某文件管理器之上的）。因此，文件管理系统的用户就可以创建或删除文件，并执行对这些文件中记录的简单检索和更新操作。与 DBMS相比：

- 文件管理器并不了解记录的内部结构，因而不能处理与结构相关的请求。一般很少提供或根本不支持安全性和完整性约束。
- 一般很少提供或根本不支持恢复和并发控制。
- 在文件管理层没有真正的数据字典的概念。
- 与DBMS相比，提供很少的数据独立性。
- 文件一般不像数据库那样具有“统一性”或“共享性”（文件通常是用户或应用程序专用的）。

2.9 数据通信管理器

本节简要讨论数据通信。最终用户的数据库请求实际上以消息的形式从用户工作站——在物理位置上，对于数据库系统本身来说，工作站可能是远程的——传递给一些在线的应用程序（嵌入的或其他），并传递给DBMS。同时，DBMS和在线的应用程序对用户工作站的响应也以消息的形式传递。所有这些消息通信都是由另一个软件部件——数据通信管理器来控制的。

数据通信管理器不是DBMS的部分，其本身是一个相对独立的系统。但由于要与 DBMS协同工作，有时两者在称为数据库 /数据通信系统(DB/DC系统)的较高层上被当作同等的部分，在该系统中DBMS处理数据库，而数据通信管理器处理传送给 DBMS和从DBMS发出的消息，更确切地说，数据通信管理器处理传送给使用 DBMS的应用程序和从应用程序发出的消息。但是，本书对消息处理介绍得较少（它本身是一个大课题）。2.12节简要地讨论了在不同系统间通信的问题（也就是，在像因特网那样的通信网络中的不同机器之间通信的问题），但实际上那又是一个独立的题目。

2.10 客户/服务器体系结构

到目前为止，本章已经从ANSI/SPARC体系结构的角度论述了数据库系统。特别地，在图2-3中给出了该体系结构的示意图。本节从一个稍微不同的角度来看数据库系统。当然，系统的全部目的是支持开发和执行数据库应用程序。从较高层来看，数据库系统可以看作是由两个非常简单的部分组成：一个服务器（也称为后端）和一组客户（也称为前端），见图2-5。说明如下：

- 服务器就是指DBMS本身，它支持2.8节讨论的所有DBMS的基本功能——数据定义、数据操纵、数据安全性和完整性，等等，尤其是提供了对所有在 2.3~2.6节讨论的内模式、概念模式和外模式的支持。在这样的上下文中，“服务器”一词就是DBMS的另一个称谓。
- 客户是指在 DBMS上运行的各种应用程序——用户编写的应用程序和嵌入的程序，即DBMS厂商或某第三方厂商所提供的应用程序。对服务器来说，用户编写的应用程序和

嵌入的程序之间没有什么不同——它们都使用相同的服务器接口，即在2.3节提到的外模式接口。

注意：某些特殊的“实用程序”会与前面的有些不同，因为它们有时会直接操作系统的内模式。这些实用程序最好作为DBMS的内部构件，而不是通常意义下的应用程序。这些在下一节将会详细讨论。

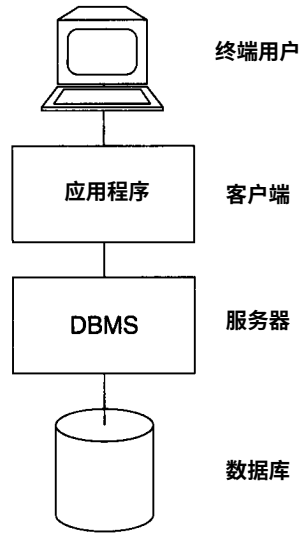

图2-5 客户/服务器体系结构

我们简要地阐述一下用户编写的应用程序和厂商提供的应用程序：

- 用户编写的应用程序基本上是规范的应用程序，或者采用第三代语言如C 或COBOL或采用一些专门的第四代语言来编写——尽管在两种情况下该语言都要有相应的数据子语言来与之匹配，如2.3节所述。
- 厂商提供的应用程序（常称为工具）的基本目的是支持创建和执行其他应用程序。所创建的应用程序是为某些特定任务定制的（它们可能不太像传统的应用程序；的确，工具的关键就是允许用户，特别是最终用户能够创建应用程序而且不必使用传统的程序设计语言来编写）。例如，厂商提供的一种工具是报表编写器，其目的是允许最终用户能够获得系统根据其请求提供的格式化报表。任何指定的报表请求可以被当作一个小的应用程序，它是用高层的（专用的)报表编写语言来写的。

  厂商提供的工具大致可以分成以下几类：

  a) 查询语言处理器；
  b) 报表编写器；
  c) 商用表格子系统；
  d) 电子制表软件；
  e) 自然语言处理器；
  f) 统计包；
  g) 复制管理或“数据提取”工具；
  h) 应用程序生成器（包括第四代语言处理器）；
  i) 其他应用程序开发工具，包括计算机辅助软件工程（CASE）产品。

  和其他许多软件。大多数工具软件的细节已经超出了本书的范围；但是，因为数据库系统的目的就是支持应用程序的创建和执行，可获得的工具软件的质量在“选择数据库”（即，选择恰当的数据库产品的过程）时当然是一个主要因素。换句话说，尽管DBMS是一个非常重要的因素，但DBMS本身不是要考虑的唯一因素。

根据以上所述可知，既然整个系统可以清楚地分成服务器和客户端两个部分，就可能出现将这两个部分分置于不同的机器上。换句话说，可能存在分布式处理。分布式处理是指通过通信网络的连接，使一个数据处理任务可以分散到网络中的不同机器上分别进行处理。事实上，这一可能性如此吸引人——有大量原因，主要是经济上的——以至于“客户/服务器”完全适用于的确需要客户和服务器在不同的机器上的情况。我们将在2.12节论述分布式处理。

2.11 工具

工具是设计用于帮助DBA处理各种管理任务的程序。前一节已经提到，一些实用程序运行于系统的外模式层，这样它们只不过是特定目的的应用程序；甚至有些不是 DBMS厂商提供的，而是由第三方提供的。但是，其他工具直接运行于系统内模式层，因而必须要由DBMS厂商提供。

下列是一些实际中需要的各种工具的一些实例：

- 载入例程，从一个或多个操作系统文件创建初始化数据库版本；
- 卸载/重载例程，卸载数据库或其一部分，备份数据库，并从这些备份的拷贝中重新装入数据（当然，“重载”工具和载入的工具基本上是一样的）；
- 重组织例程，由于各种原因要重新组织数据库中的数据——例如，将磁盘上的数据聚集起来，或回收废弃数据所占用的空间；
- 统计例程，计算各种性能统计数据，如文件大小或数值分布或输入/输出次数，等等；
- 分析例程，分析上述的统计数据；

等等。当然，以上提到的只是工具所提供功能范围中的小部分；还存在其他丰富的功能。

2.12 分布式处理

分布式处理是指不同机器可以通过通信网络如因特网连接起来，这样，一个数据处理任务可以分散到网络中的几台机器上进行分别处理（“并行处理”一词有时也用于同样的意义，除了不同的机器物理上连接成“并行”系统而不是这样的“分布”系统之外——例如，它们可以在地理上是分散的）。在不同机器之间的通信是由网络管理软件来处理的——可能是2.9节中提到的数据通信管理器的扩展，还可能是一个独立的软件部件。

在各种层次上都可能存在分布式处理，分布式处理也可能是各种各样的。2.10节已经提及，一个简单的例子是在一台机器上运行DBMS后端（服务器），而在另一台机器上运行应用程序前端（客户端），见图2-6。

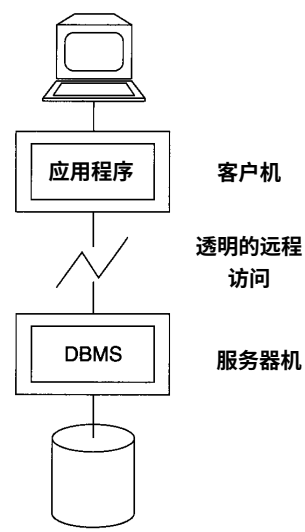

图2-6 运行在不同机器上的客户端和服务器

在2.10节的末尾已经提到过，尽管“客户/服务器”严格地说是体系结构术语，但它已经成为图2-6中图例方案的同义词，其中客户和服务器运行在不同的机器上。有许多理由支持这样一种方案：

- 首先，基本上是通常的并行处理的理由；即，许多处理部件都可用于整个任务，而且服务器和客户处理并行进行。响应时间和生产率就提高了。
- 而且，服务器机可以是为数据库管理系统功能而定制的机器（数据库机），这样就使数据库管理系统更高效。
- 同样，客户机可以是根据最终用户的需要定制个人工作站，这样就提供了更好的界面、高可用性、快速反应和全面提高用户的易用性。
- 几台不同的客户机可能访问同一台服务器。几个不同的

客户系统可以共享一个数据库（见图 2-7）。

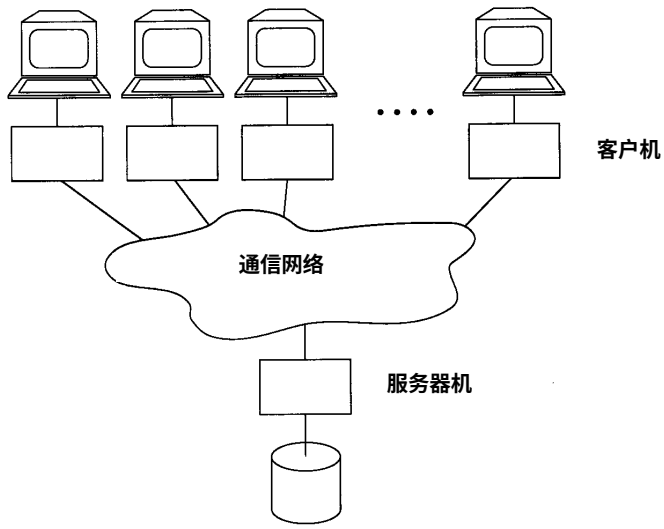

图2-7 一台服务器机，多台客户机

此外，在不同机器上运行客户和服务器符合企业的实际运作方式。一个企业 —— 例如银行 —— 运行许多计算机是很常见的，这样，企业就可以把一部分数据存在一台机器上，而另一部分数据存在另一台机器上。一台机器的用户偶尔会访问其他机器上的数据也是很平常的。继续看银行这个例子，一个分支机构的用户偶尔需要访问其他的数据。注意，客户机有自己的数据，服务器有自己的应用程序。一般地讲，每台机器都会作为一些用户的服务器和另一些用户的客户机（见图2-8）；换句话说，在本章前几节的意义下，每台机器都将支持整个数据库系统。

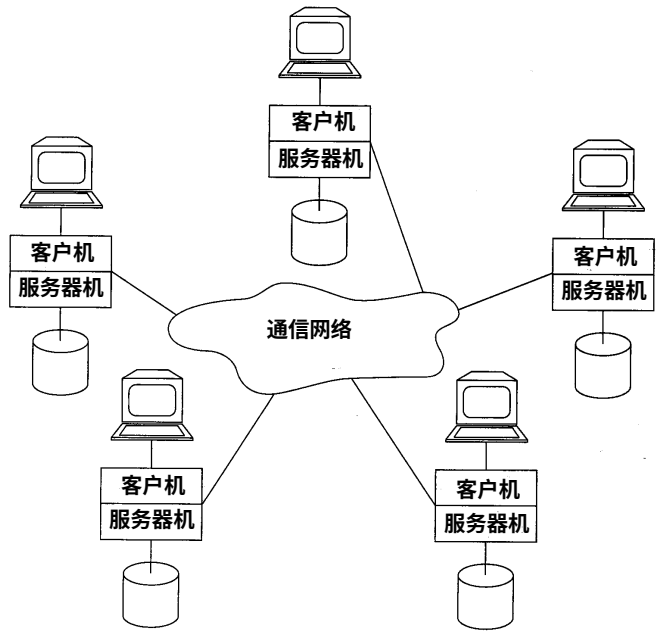

图2-8 每台机器既是服务器机又是客户机

最后一点是一台客户机可能访问几台不同的服务器机（与图 2-7中的例子的情况相反）。如前所述，企业非常需要这种功能，因为企业典型的运转方式是整个数据不是存储在单个的机器上，而是分散在许多不同的机器上，而且应用程序需要从不止一台机器访问数据。基本上，有两种方式可以提供这种访问：

- 一台客户机要访问任意数目的服务器，但是一次只能访问一个（即，每个单独的数据库请求只向一台服务器发出）。在该系统中，不可能在一个请求中将两个或更多的服务器的数据结合起来。系统中的用户必须了解哪台机器存有哪些数据。
- 客户机可以同时访问许多服务器（即，一个数据库请求可以将几个不同数据库的数据结合起来）。在这种情况下，对客户机来说，这些服务器 —— 从逻辑上看 —— 好像实际上只是一台服务器，系统的用户不必了解哪台机器存有哪些数据。

后一种情况就是通常所说的分布式数据库系统。分布式数据库本身是一个大课题，随之而来的逻辑结论是，完整的分布式数据库支持意味着，单个的应用程序应该可以“透明地”操纵数据，这些数据分布在各种不同的机器上，由不同的操作系统支持，由各种不同的通信网络连接起来 —— 这里，“透明地”的意思是指从逻辑上看，应用程序操纵数据，就像数据都由运行在同一台机器上的DBMS来管理。这种功能听起来有点儿悬！但是从实际的角度来看，这是非常有吸引力的，而厂商正在努力实现这样的系统。我们将在第 20章详细讨论这些系统。

2.13 小结

本章从体系结构的角度分析了数据库系统。首先，我们介绍了 ANSI/SPARC体系结构，它将数据库分成了内模式、外模式和概念模式三层。其中，内模式最接近物理存储（即，它要考虑数据的物理存储）；外模式最接近用户（即，它要考虑单个用户看待数据的方式）；而概念模式则是介于前两者之间的中间层（它提供数据的公共视图）。映象定义了外模式与概念模式之间的映象及概念模式与内模式之间的映象。映象是提供数据逻辑独立性和数据物理独立性的关键。

用户 —— 可分为最终用户和应用程序员，两者都操作外模式 —— 通过数据子语言与数据交互，数据子语言至少又分成两部分：数据定义语言和数据操纵语言。数据子语言嵌入在主语言中。注意：主语言和数据子语言间的界限和数据定义语言和数据操纵语言间的界限都是概念上的；在实际应用中，它们对用户来说是透明的。

我们进一步看到了DBA和DBMS的功能。DBA负责创建内模式（数据库物理设计）；相对地，创建概念模式（数据库的逻辑或概念设计）是由数据管理员来负责的。而 DBMS负责执行用户的数据定义语言和数据操纵语言的请求。 DBMS也负责提供某些数据字典的功能。

数据库系统也可以由一台服务器和一组客户机来构成。客户机和服务器能够在不同的机器上运行，这样提供了一种分布式处理。一般地讲，每台服务器能为几台客户机提供服务，而每台客户机也可以访问多个服务器。如果系统提供完全透明 —— 也就是每台客户机就好像在单机的单个服务器上操作，而不考虑物理连接状态 —— 那么就真正实现了分布式数据库系统。

练习

2.1 画出本章的数据库系统体系结构图（ANSI/SPARC体系结构）。

2.2 定义下列术语

后端	前端
客户端	主语言
概念数据定义语言，概念模式，概念视图	载入
概念模式/内模式映象	数据库的逻辑设计
数据定义语言	内部数据定义语言，内模式，内部视图
数据字典	数据库的物理设计
数据操纵语言	计划请求
数据子语言	重组织
数据库/数据通信系统	服务器
数据通信管理器	存储结构定义
分布式数据库	卸载/重载
分布式处理	非计划请求
外部数据定义语言，外模式，外部视图	用户接口
外模式/概念模式映象	工具

2.3 解释检索一个外部记录值的步骤。

2.4 列出DBMS的主要功能。

2.5 辨别数据的物理独立性和逻辑独立性。

2.6 如何理解元数据？

2.7 列出DBA执行的主要功能。

2.8 辨别DBMS和文件管理系统。

2.9 给出几个厂商提供的工具的例子。

2.10 给出数据库的工具的几个例子。

2.11 观察一个数据库系统。试用 ANSI/SPARC体系结构来看系统。它支持三层体系结构吗？两层之间的映象是如何定义的？各种数据定义语言（外部的，概念的，内部的）的功能是什么？系统支持什么数据子语言？主语言是什么？谁履行 DBA的职能？有安全性和完整性机制吗？有字典吗？字典是自描述的吗？系统支持什么厂商提供的应用程序？什么工具？有没有独立的数据通信管理器？有没有分布式处理的能力？

参考文献和简介

下面的参考文献目前来说比较陈旧（除了最后一个），但是与本章介绍的概念相关。

2.1 ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report，*FDT*(bulletin of ACM SIGMOD)*7*，No.2 (1975)

2.2 Dionysios C. Tsichritzis and Anthony Klug (eds.): “The ANSI/X3/SPARC DBMS Framework: Report of Study Group on Data Base Management Systems，” *Information System 3* (1978).

参考文献[2.1~2.2]是ANSI/SPARC研究组的中期和终期报告。有关数据库管理系统的ANSI/X3/SPARC研究组成立于1972年，由美国国家标准协会的标准计划和请求委员会为计算机和信息处理而建立的（约 25年后，X3这一名字已经改为 NCITS —— 国家信息技术

标准委员会)。研究组的目的是决定数据库技术的哪些领域是符合标准的，并对每个这样的领域给出一组推荐方案。在为此目的工作时，研究组认为，数据库系统能够满足标准的唯一方面是接口，定义一个一般的数据库体系结构或框架，强调这些接口的作用。终期报告提供了体系结构的详细描述和 42种接口。中期报告是早期的文档；在一些领域还提供了额外的细节。

2.3 J. J. van Griethuysen (ed.): "Concepts and Terminology for the Conceptual Schema and the Information Base," International Organization for Standardization (ISO) Technical Report ISO/TR9007: 1987(E)(March 1982; revised July 1987).

这个文档是ISO工作组的报告，其目的包括"概念模式语言的概念的定义"。它介绍了对一组形式方法的三个竞争的候选方，并且把三者都应用到一个普通的例子，即假定的汽车注册授权活动。三组竞争者是（1)"实体-属性-联系"模式；（2)"二元关系"模式；（3)"解释的谓词逻辑"模式。报告还包括了概念模式的基本概念，提出了一些理论作为能够支持这一概念的系统实现的基础。对系统的概念层有兴趣的人来说，这是一篇重要的文档。

2.4 William Kent: *Data and Reality*. Amsterdam，Netherlands: North-Holland/New York，N.Y.: Elsevier Science(1978).

一场对信息的本质和概念模式的本质的讨论。"本书提出了哲学原理，生活和现实本质上是无定形的，无序的，矛盾的，不一致的，不合理的，非客观的"(除了最后一章)。这本书可以看作是真实生活问题的概述，即现存的数据库形式方法——尤其是基于习惯的记录型的结构，包括关系模型——在处理上有困难。推荐阅读。

2.5 Odysseas G.Tsatalos，Marvin H.Solomon，and Yannis E.Ioannidis: "The GMAP: A Versatile Tool for Physical Data Independence," Proc. 20th Int. Conf. on Very Large Data Bases， Santiago，Chile(September 1994).

GAMP表示通用多层访问路径。文章的作者注意到今天的数据库产品"强迫用户使用与物理结构紧密联系的逻辑模型形式的查询"，而这样数据的物理独立性非常差。在文章中，他们提出一种概念模式/内模式的映象语言，可用于说明比今天的产品所支持的多得多的映象。指定一个"逻辑模式"，语言允许许多不同的物理模式的说明，它们多来自于逻辑模式。物理模式包括垂直的和水平的部分（见第 20章)，任意多的物理访问路径，聚集和控制的冗余。

这篇论文还给出了一个算法，该算法将用户对逻辑模式的操作转化为等价的对物理模式的操作。一个原型系统显示数据库管理员能调整物理模式以获得比通常要更好的性能。