软件工程 数据库 CASE

计算机科学1994 Vol. 21 №. 6

63-68

CASE 的数据库需求*)

郭 江 周伯生
(北京航空航天大学软件工程研究所 北京 100083)

TP311.5

摘 要 In this paper, we have listed, explained and classified the CASE database requirements. We also consider the main data manager classes, and classify them according to how well they satisfy each of the requirements. The major conclusion is that the object-oriented and extended relational DBMS's currently satisfy well most of the CASE database requirements, and that they are evolving in the right direction so that they will provide all the power needed by CASE environments.

关键词 CASE, CASE environment, Data base.

1 引言

CASE(计算机辅助软件工程)环境会产生大量异质数据,包括手册、程序源代码、测试结果,以及功能说明。为了有效地管理这些数据,CASE 数据仓库必须满足几个数据管理的需求,包括大的存贮容量、永久性、共享、以及完整性。为了使数据模型成为数据管理系统的基础,CASE 环境设计人员可以选择现存的数据库管理系统(DBMS),也可以自己做一个。目前,大多数 CASE 环境倾向于后者,以满足 CASE 数据需求的一个有限子集。选择现存的 DBMS,就要确定不同的问题。例如,OMEGA[1]发现一个关系 DBMS 需要较强的对象标识,而 CIA[1]发现它缺少足够的性能要求。如果这些障碍被克服了,就能在 CASE 中使用一个通用的 DBMS,这样的 CASE 有下列好处:共享相同的数据仓库,CASE 和非 CASE 工具结合以及两者之间交换数据;广泛使用数据库工具(如报表生成器);而且具有无限制的查询能力。

现在 DBMS 和 CASE 领域中有一个研究热点:开发新的数据模型和技术,满足 CASE 的需求[1]。商业产品和研究原型也提供了各种数据模型和技术选择,已经扩大了数据管理系统的选择范围。现在正是评价数据管理系统类是否满足需求的时机,并讨论新系统怎样阐述 CASE 的需要。

有些研究人员已经汇集了 CASE 环境的数据库需求,但并没有正式给出需求清单,也没有进行充分解释或公正地评判,更没有联系到特定的数据管理系统来考虑其满足需求的程度。

本文汇集了来自不同方面的需求,消除了那些重复和不相关的内容,并增加了新的需求以支持 CASE 环境。此外,还解释了每个需求的目的,并根据其功能及是否与数据模型相关对需求进行了分类,进而对过去的和当前的数据管理系统进行了分类,然后对每个类评价了它满足每个需求的程度,据此,提出了用于设计新 CASE 环境的数据管理系统。

2 CASE 数据库需求

这部分列出了 CASE 环境的主要数据库需求,并将需求划分成两类:特定于数据模型的需求和涉及功能但不依赖于数据模型的需求。

2.1 数据模型

数据模型构成了数据管理系统的基础,确定了数据管理系统的实体,实体之间的关系,在实体上可进行的操作。下面讨论数据模型应具有的一些性质。

·可扩充性:模型中不可能包括所有用户想要的特征,可扩充性允许以不可预知的方式使用数据管理系统,也可用于新的领域。有几个其它特征在一个或几个方面提供了可扩充性,如 ADT(抽象数据类型)机制扩充了系统的数据处理,但是数据模型还需要用明确的可扩充策略来设计。

·视图:通过视图机制,系统将应用从低层的改动隔离到了模式上,并提供了相同数据的不同视图。从数据库的角度,视图既可以看作是一个不存在的

*) 本文受国家"八五"项目的资助。郭江 博士生,主要研究领域:软件工程环境、数据库等。周伯生 教授,博士生导师,主要研究领域:软件工程、软件工程环境。

虚关系，但可以由其它关系来定义。视图既可以存放在物理内存中，也可以不放在其中(不同的系统由于访问时间效率的原因可以选择具体实现一个视图)。因此视图提供了额外的功能，而又不需浪费空间或产生数据一致性的问题。

·对象封装：面向对象语言的一个重要特征就是对象必须包括数据和操作，所以提供了模块化的优势，用户可以使用其服务而不考虑其实现。

·明确的关系：如果关系(链)是数据模型中的一部分(如实体关系模型)，那么数据模型就更容易表示数据，其表达能力更强，实现也更有效。

·丰富的、动态的类型系统：这样类型可以有不同类型的值(称为联合类型)，而且其大小也可以改变(如，数据和 ASCII 域)。

·各种 ADT 机制：用户应能定义新类型、新操作符以及新类型上的索引，而且要能用这些新类型作为其它新类型的成员，或用于其它类型构造器中(如，定义了一个栈类型以后，那么就应能构造栈数组和数组栈)。

·类型继承：要允许将对象类型或类定义作一个层次结构的一部分，从而继承其祖先的特性(如，一个学生可以从人来继承属性，并增加新的属性"专业域")，或者对这些特性增加约束(如，"经理"可能有约束10,000<年薪<50,000)。

·数据库子过程：把子过程作为 DBMS 的一部分是一种简单而又有力的思想，它不仅提供了一种更有效和灵活的系统，而且允许进行更细的调整。例如，既可以将它和数据耦合在一起封装成对象语义，也允许在应用系统或 DBMS 中更合适的地方执行数据转换操作。

·过程程序设计：一个过程程序是确定在一个仓库实体上执行一系列操作的程序，描述了系统强加于开发和维护软件文档的各种软件过程。这种支持可以以过程程序设计语言的形式出现，也可以是其它特征的组合，如数据库子过程和规则系统。

·规则系统：这种能力是让用户确定有关数据和其它对象的一套规则，以提供知识管理能力。几种重要的仓库能力，如视图、调用完整性以及子过程均可由通用的规则系统来提供。因此，规则系统不仅提供了本质性的服务，而且还增加了重要的所必需的新能力。

·复杂对象：一个复杂对象是由不同子实体链成的一个集合。复杂对象在 CASE 中以总的形式出现(如，一个带属性的 AST 由与其它结点和属性链在一起的结点组成)。这种支持必须包括访问和操纵子对象的机制。

2.2 功能方面的属性

有许多特征并不是数据模型所固有的，这意味着这些特征可以用不同的数据模型结合到系统中，而且可以用不同的方法实现。

·永久性：数据管理系统的最基本特征就是保存数据、对象、以及支持所有其它实体，并管理由系统施加到实体上的结构和限制。永久性必须与所有其它特性正交。主要的问题是在所有存贮级上提供无缝连接。

·并发和分布式访问：并发性允许多个用户访问同一个数据库，而分布式则允许数据分布到最接近于使用者的位置。

·可扩充的访问方法：特殊的访问方法允许用户快速访问数据。为有效地支持异质数据和存贮介质(RAM，磁盘、数据驱动器)，就需要用户定义的访问方法。

·开放式的程序接口：计算机科学和商业界已经使用了各种程序设计语言(如 LISP 和 C)以及应用程序语言(如，Lotus 1-2-3)。通过使仓库程序设计语言中性化，就可以使更多的应用系统访问它。

·主动数据：这种能力是让用户确定在条件满足时执行特定的操作，有时称为触发器，可用于通知重要事件的出现。主动数据可以直接实现，也可以建立在通用规则系统上。

·通用的查询能力：包括导航式与关联式查询。通用的查询机制对方便用户访问想要的数据是非常重要的。关联访问是当代关系系统的主要贡献之一；导航式访问是 CODASYL 数据库的一个主要部分，对灵活性和性能的改善是必要的。在 CASE 环境这样一个复杂和大的系统中，对不同用户有不同的要求，组合各种数据的不可预料的查询是不可避免的，也是很重要的。

·版本和配置支持：CASE 文档是在若干段很长的时间中开发的。每个阶段通过一系列文档的小改动来达到一个高层的目标。为了在不同阶段编写文档和保存文档，就要创建和命名可在以后检索和查询的新版本。一个配置是一个复杂的实体，由不同对象的特定版本组成。由于版本和配置是软件生命周期中必不可少的部分，因而需要直接支持。

·访问控制：在复杂的大项目中，由许多人负责对不同文档的管理。为了管理文档，必须选择授予访问文档对象的权利，而且还必须允许有关人员和小

组在系统中存放私有数据。

·大块数据:CASE 应用需要有效地从仓库中移进和移出大量的数据,只能由直接支持大块数据的操作来实现,如同时以 DB 内部格式读取大量元组的能力。输入输出源必须包括文件和过程。

·事务:需要长短事务的支持。目前,可以有效地支持短事务而且事务处理系统工作得也很好。为了支持短事务,系统锁住事务处理期间接触的每个数据项,直至事务提交将锁释放为止。这种模型对 CASE 不很适用,因为用户可能一次花数天时间来检查一个代码模块,修改之,然后放回去。这样就不能拒绝其它用户对该模块的访问,Kaiser[2]提出了几种可供选择的方法。

·崩溃和失败恢复:在系统出问题时能保护敏感数据的不丢失和一致性。例如,如果系统在一个事务过程中崩溃,那么在系统初始化时就有一个恢复过程,即将数据库恢复到事务开始前的状态。

·Undo 能力:允许用户将一系列有效的操作恢复到以前的、有效的数据库状态。这对少量最新发出的操作来说非常重要,因为允许迅速恢复并激发人们尝试新的方法。

·细的和粗的对象:大对象和小对象在操作、访问方法、存贮组织、以及处理方面有很大的区别。由于 CASE 数据既有细粒度的(如文档作者),也有粗粒度的(如用户手册中的图),因而需要明确支持两种粒度对象。

·元模式支持:系统要方便访问数据、过程、规则、视图、实体和约束的定义与说明。由于定义能很容易地修改、增加和删除,这将有助于系统的发展以及扩充。

·层次结构:图在 CASE 中很常见,有效地对其操作和表示非常重要。一个丰富的类型系统和明确的链类型将提供有效的表示,而支持导航式访问和指针也会提供有效的操作。

·多级存贮:由于 DBMS 管理的数据量增长很快,有必要扩充数据存贮的层次结构,不限于内存和磁盘结构形式。软件系统不仅由许多不同文档组成,而且每个文档有许多不同的版本,以多种方式表示在数据库中。对以百万行代码计的软件系统而言,在系统开发的生命周期中就会生成大量数据。

另外,数据模型必须简单而有力。简单则易于理解和实现,而有力则有较大的适应范围。模型应提供简单、有效的基础,而复杂的操作则建立在此基础之上。最后,所有这些特征需要以好的性能加以实现,否则就不能使用了。

可以看出,这些需求多但不繁锁。在某些情况下,可以由一个或几个特征满足,例如过程程序设计可以由 DB 子过程和规则系统的组合来支持,而规则系统可以提供视图机制。根据所能阐述的功能类型这些需求也可以分成几类:数据、对象、过程和知识管理。下面详细考虑每种类型。

(1)数据管理:是当代关系系统所具有的特征,也是商品化 DBMS 所期望的。数据管理特征是:永久性、程序接口、视图、关联查询、访问控制、大量数据管理、短事务、失败恢复、Undo 能力、并发、分布式访问、元模式操纵。

(2)对象管理:是支持非传统 DBMS 应用(如 CASE 和 CAD)的复杂数据需求,它们不针对特定对象,其主要特征有:可扩充的数据模型、对象和操作的封装、明确的关系、类型(动态,ADT,继承)、数据库过程、可扩充的访问方法、导航式查询、版本和配置支持、长事务、对象(大的和复杂的)、多级存贮、层次结构。

(3)知识管理和过程程序设计:知识管理特征包括知识活动的表示和管理,如规则系统。过程管理特征是用于支持软件生命周期中不同活动的表示和执行。知识管理和过程程序设计特征包括:规则系统、主动数据、过程程序设计。

3 数据管理系统

这部分将讨论 CASE 数据管理系统的不同选择。主要考虑:1)手工建立一个用户 DBMS;2)用生成器建立一个用户 DBMS;3)使用永久性对象存贮系统;4)使用一个关系 DBMS;5)使用一个关系对象 Shell;6)使用一个面向对象的 DBMS;7)使用一个扩充的关系 DBMS。

还有其它的选择,例如,实体关系 DBMS 和 CAD 的 DBMS、很大程度上被关系系统代替了的较老的网状或 CODASYL DBMS、以及基于规则的 DBMS 如 LOGRES[3]和从知识表示语言导出的 LDL[3]。下面详细解释每一类。

1°)手工建立一个用户 DBMS:一个用户写的 DBMS 如 Cadre[3]和 Software BackDlane[2]似乎是 CASE 环境最需要的数据管理机制。这种方法不是利用已存在的系统来满足 CASE 的所有需求,而是用户设计和开发一个用户系统来满足之。实际上,许多 CASE 环境都使用用户 DBMS,满足所需特征的一个子集。其范围从只在 OS 和 CASE 工具之间加一个数据管理层,到提供数据模型和更高级的数据

管理能力。尽管如此，所提供的特征子集通常包括：复杂对象的支持，有限的查询能力、并发性和永久性存贮的支持。值得注意的是那些没有提供的特征：强有力的访问方法、通用查询能力、多视图的支持、主动数据、以及可靠数据的存贮。这些是由通用 DBMS 所提供服务的核心，被丢掉的原因是：这些特征是使系统大而复杂的根源。换句话说，它们要花费许多人年来开发、优化，并要高水平的开发人员来维护。用户 DBMS 方法的缺点是建立在此基础上的环境不能和其它 CASE 环境或非 CASE 应用系统共享数据或工具，而且建立在上面的应用系统几乎不可移植。

2）用生成器建立一个用户 DBMS：这样的系统如 EXODUS[4]和 Genesis[4]，允许设计人员按照说明生成一个全功能的 DBMS。其实现可以通过允许用户替代不同的模块或给程序生成器提供特定模块的用户描述。例如，Genesis 允许替换缓冲和恢复管理模块以实现不同的策略，而 EXODUS 优化器生成工具则采用用户的优化器描述，并作为用户 DBMS 的一部分而生成，这似乎是一个非常有吸引力的选择方案。但存在若干问题，需要应用开发人员来编写 DBMS 的关键部分，例如，为了用 EXODUS 实现用户的 DBMS，就要使用数据库工具集和永久性程序设计语言来提供一个数据模型、查询语言和分析器、目录、访问方法、数据存贮结构、查询优化和查询执行策略。另外，这种方法很难和其它应用领域共享和集成数据。

3）使用一个永久性对象存贮系统：这样的系统如 Kala[1]和 ObServer[1]都提供了永久性对象和对象调用以及其它几个复杂的特征。例如，Kala 提供了一个低层的、无类型的存贮管理系统，用这个基础来实现特殊的事务、访问控制、版本和配置模型。但它们缺少许多其它复杂的特征，如规则，子过程、查询语言等。

4）使用一个关系 DBMS：这样的系统如 Ingres[1]和 Oracle[1]有许多 CASE 数据管理所需的特征，包括那些通常被用户写的 DBMS 所丢掉的特征：并发、崩溃恢复、关联查询等。另外，关系模型是简单而又广泛使用的数据管理系统，并开发出了许多应用。尽管如此，关系 DBMS 还有若干问题使之不能用作 CASE 数据管理系统：性能不好，对层次结构支持较少，数据模型不可扩充，而且缺少触发器。这些问题迫使将 CASE 数据以一种特定的、不自然的方式映射到 DBMS 上。另外，关系系统对复杂的 CASE 操作提供的性能较差。例如，Omega[1]报告，对一个经特别调整的 DBMS 而言，从 DBMS 中取一个五行的程序需要花费7秒。

5）使用一个关系对象 Shell：这样的系统如 Ingres Object Management Extension[5]和 Penguin[5]在关系 DBMS 的顶部提供了对象管理。这种方法增加了新的功能，特别是对于对象的支持，但也继承了关系系统的弱点，如执行性能不够好。

6）使用一个面向对象（OO）的 DBMS：这样的系统如 Iris[5]和 Gemstone[5]已经设计成满足复杂数据需求的系统。这些特征使得 OODBMS 必须包括"is-a"层次结构（继承）、对象标识、封装，以及数据模型和程序设计语言的紧密集成。许多 OODB 设计人员从一个 OO 程序设计语言出发，通过增加传统的 DBMS 特征如对象永久性和查询能力来对其进行扩充。这种方法使体系结构的决策和数据模型的特征（如，与一个特定的 OO 程序设计语言如 C++密切相关）有别于扩充传统的关系系统。它们对导航式的 CODASYL DBMS 提供了极大支持，而用户需要了解和确定通向所有数据项的路径。新的 OODB，如 O_2[1]包括了一些特定系统所省略掉的特征，如关联查询，以及对传统业务数据应用的良好支持。

7）使用一个扩充的关系 DBMS：这样的系统如 POSTGRES[1]和 Starburst[1]很可靠地建立在关系模型基础之上，而且对之进行扩充以弥补在对象和知识管理方面的缺点，如对象封装、规则系统、性能调整机制（如快速路径）。和 OODBMS 相比，它是从 DBMS 的观点出发并对之用复杂的对象以及知识管理特征来进行扩充的。例如，POSTGRES 结合了关系模型的优点并扩充支持新的应用：主动数据（触发器）、系统的可扩充性（数据模型、访问方法等）、子过程数据类型、改进性能的机制（快速路径和预计算）、历史数据、对版本的支持等等。扩充的关系系统的其它优点包括：已有工具和应用系统的变体、商业数据共存于同一个仓库中（允许"混合"应用和查询）、以及一条通向已存在的许多关系型应用系统的转移路径。

针对用户的功能和可扩充性提供下列五种主要策略。

1）为某个特定特征提供低层基础，给用户提供一个机制，并允许他们实现自己的方法。这种策略目前没有广泛接受的领域，如 Kala 对事务的支持。这种策略有若干严重不足：(i)系统设计人员方面的工作太多（在设计和开发方面均如此）。(ii)这些方法的

实现如果是非标准的，则可能导致与其它系统的不兼容，甚至那些基础相同的系统也是如此。它的优点是没有明确建议什么方法是最好的，如事务支持和配置管理。

2)使用一个模块生成器策略。将一个定义好的子系统描述传给模块生成器，就能产生出所需要的行为代码。生成器策略的例子在不同领域有 Yacc、分析器生成工具，以及在 DBMS 领域中利用 EXODUS 优化器生成工具来生成优化代码。这种方法在较窄的领域对深入理解的算法(如分析器)工作得很好。它的缺点包括生成器中可能使用与系统的其它部分不同的语言，而且在生成的模块中加入特定的动作比较困难。

3)提供一个接口和机制以便允许用户集成自己的方法。如，要在 POSTGRES 中定义一个新的访问方法，系统设计人员给系统编写和提供了关键功能，然后采取必要的步骤将之组合到优化器中。也就是，POSTGRES 提供了一个访问方法，而且还提供了一个接口以便用户能容易地提供自己定义的访问方法。

4)以模块的方式来设计和实现系统。用户可以用一个模块代替另一个模块。这是 GENESIS 中使用的用于针对用户需求的缓冲区管理方法。也就是，GENESIS 提供了缓冲区管理策略的一个变体，每个都封装在一个模块中。在系统安装时，选择一个需要的策略即可。这种方法的好处是允许用户在若干标准策略之间进行选择，但是它不给用户提供定义好的策略。

表1 不同模型的数据管理特征

特　征	用户	用户生成	对象存贮	关　系	关系对象 Shell	OO	扩充关系
永久性	F	F	F	F	F	F	F
分布式访问	F	F	F	F	F	F	F
大量数据	F	F	P	F	F	F	F
短事务	L	F	P	F	F	F	F
访问控制	F	F	P	F	F	F	F
程序接口	L	P	P	F	F	F	F
视图	N	P	N	F	F	F	F
关联查询	N	P	N	F	F	F	F
Undo 能力	F	P	P	F	F	F	F
并发支持	F	F	F	F	F	F	F
崩溃恢复	L	P	N	F	F	F	F
元模式	L	P	N	F	F	F	F

表2 不同模型的对象管理特征

特　征	用户	用户生成	对象存贮	关系	关系对象 Shell	OO	扩充关系
可扩充性	L	P	P	N	L	L	F
对象封装	F	F	F	N	F	F	F
明确的关系	L	F	P	N	L	F	L
动态类型	L	P	N	N	F	F	F
ADT	F	P	N	N	F	F	F
继承	F	P	N	N	F	F	F
DB 子程序	F	L	N	N	F	F	F
复杂对象	F	P	F	N	F	F	F
导航式查询	F	P	N	N	F	F	F
版本控制	F	P	P	N	N	F	F
长事务	F	N	P	N	N	F	N
大对象	F	P	F	N	F	F	F
多级存贮	N	N	L	N	N	L	F

5)提供一个灵活的、开放式的接口，以便系统设计人员可以使用自己的代码。例如，ObServer 用接口提供了对象的永久性支持以及对象调用，这样系统设计人员就可以实现自己的查询语言。缺点是用户必须提供模块，甚至在他想要一个标准的实现策略时也要自己提供模块。这种方法如果能结合系统提供的标准模块进行选择就是一种很好的方法。

4 数据管理系统分类

这部分主要根据满足 CASE 仓库需求的程度来对数据管理系统进行分类，很难正确地指出哪类管理系统能满足功能方面的需求，因为每类管理系统有许多不同的实现方法，而且每个具体实现技术也各不同。

在表1、2和3中以数据管理系统的不同选择作列，以不同的特征作行。表1列出了数据管理特征，表2列出了对象管理特征，而表3列出了知识和过程管理特征。如果服务器类不提供这些特征，就填上 N(NO)；如果仅提供有限的支持，就填上 L(LIM)；如果提供了基础而允许用户用特定的实现方法来实现，就填上 P(PRIM)；结果得到了全面的支持就填上 F(FULL)。

表3 不同模型的知识和过程管理特征

特 征	用户	用户生成	对象存贮	关系	关系对象 Shell	OO	扩充关系
主动数据	F	L	N	N	N	L	F
规则系统	N	P	N	N	N	N	F
过程程序设计	F	P	N	N	N	L	P

考虑这三个表，可以发现一个趋势，除了对象存贮类(它提供了数据管理基础的一个子集)和用户类(它省去了实现困难的特征如关联查询)之外，几乎所有的类都满足数据管理需求。

扩充的关系和面向对象类可以很好地满足对象管理特征。关系对象 Shell 覆盖了大多数对象管理特征，而关系类则一个也没有覆盖。用户生成器类对大多数对象管理需求提供了间接和有限的支持，而用户类则对大多数需求提供了全面或有限的支持。对象存贮类对大约一半的对象管理特征提供了间接支持。

关系、关系对象 Shell、以及对象存贮类都没有提供过程程序设计以及知识管理的支持。用户 DBMS、用户生成器、以及面向对象类对这些特征提供了一些支持，但仅是扩充的关系类对这些特征提供了全面的支持。

从这种分类出发，可以看出，有两类数据管理系统能满足大多数需求，而且只需要很少量的工作就可以使用面向对象的以及扩充的关系 DBMS 类。还可以看到妨碍成功的主要障碍是它们在产品环境中的执行性能。但是，一般认为性能方面的问题可以通过慎重的系统体系结构选择以及明智的数据库设计来解决。

参考文献

[1] M. Luis, "The Design of a CASE Environment Architecture and the Performance Evaluation of Database Designs for Software Documents", Ph. D Dissertation, EECS, UC Berkeley, California, 1992.

[2] G. E. Kaiser, "Intelligent Assistance for Software Development and Maintenance", IEEE Software, May 1988.

[3] Chimenti, "The LDL System Prototype", IEEE Data and Knowledge Engineering, March 1990.

[4] M. Carey, "The EXODUS Extensible DBMS Project: An Overview", In Reading in Object-Oriented Databases, Morgan Kaufmann, 1990.

[5] Relational Technology Inc., "INGRES Object Management Extension User Guide", Relational Technology Inc., Berkeley California, Jan. 1990.