数据模型

在概述中我们谈及了数据管理系统的目的、作用和历史，也列举了一些评判系统设计好坏的标准。本章进入正题。我们开始具体讨论数据管理系统的功能及设计。

数据的组织方式

对应用程序而言，数据管理系统提供的最基本功能无非两个。第一，将数据保存到系统中。第二，在需要时，能方便地从系统中取出数据。当然，如前文所述，系统还应该保证数据的持久性、安全性、真实性，并提供数据更新和数据处理的能力。而这些功能都是建立在数据存取功能之上的。所以，我们首先需要了解数据存取功能及其接口。

显而易见，数据存取功能的对象是数据。在数据管理系统中，数据应该以什么样的方式来描述和组织呢？例如，我们有如下两条数据。每一条数据都刻画了一个人，包括这个人的姓名、生日、住址等。

> A man named Jason Chang, who was born on Jan 20, 2001, lives in Beijing, at 20 Yamaha Street.

> A woman named Jessie Li, who was born on Dec 4, 1992, lives in Shanghai, at 200 Sichuan Street.

一种简单的组织方式是将关于Jason Chang的数据打成一个包，一次性存入系统。需要时，我们提供“Jason Chang”这个名字，系统即可将他的数据取出来。同理，我们也可以将关于Jessie Li的数据打包存入系统。需要时，再通过“Jessie Li”这个名字将数据取出。为了实现这样的存取功能，数据被组织在如下的映射结构中：

> "Jason Chang" → "A man named Jason Chang, who was born on Jan 20, 2001, lives in Beijing, at 20 Yamaha Street."

> "Jessie Li" → "A woman named Jessie Li, who was born on Dec 4, 1992, lives in Shanghai, at 200 Sichuan Street."

这里的人名可以视为**键**（key），关于人的数据可以视为**值**（value）。也就是说，为了实现数据的存取，我们可以用如下的“**键值对**”结构对数据进行组织。

> *key → value*

在这个结构中，“键”被作为数据项的标识。我们使用“键”对数据进行存取。这样，似乎满足了数据的存取需求。

然而，情况可能会变得更加复杂。假设我们积累了很多人的数据，除了Jason Chang和Jessie Li，还包括其他人的数据。这个时候，我们想从系统中取出居住在地址为20 Yamaha Street的人的信息。事先我们并不知道这个人叫什么名字。在这个情况下，“键值对”结构就无法满足我们的数据获取需求了。虽然“20 Yamaha Street”的字眼藏在“键值对”的“值”中，但对计算机而言，这些值只是一个个无法理解的字符串。为了达到目的，我们只能将所有的键值对依次取出，然后逐个判断哪些人住在20 Yamaha Street。换句话说，“键值对”这种组织结构太简单。我们无法通过它向计算机表达某些数据获取需求，而只能将所有数据取出后进行人工判断。

如果我们将数据组织在下面的结构中，情况就有所改变。

> name: Jason Chang
> birthdate: Jan 20, 2001
> gender: male

> address: 20 Yamaha Street
> city: Beijing

> name: Jessie Li
> birthdate: Dec 4, 1992
> gender: female
> address: 200 Sichuan Street
> city: Shanghai

这种结构将每个人视为一个**实体**（entity），每个实体被赋予若干**属性**（attributes），比如姓名（name）、生日（birthdate）、住址（address）等。属性的**取值**（value）实现了对实体的刻画。基于这样的结构，我们可以将实体整体存入系统，也可以将它们整体取出来。不仅如此，系统还可以对单个属性进行操作，比如改动某个人的住址或名字。如果要查找居住在地址为20 Yamaha Street的人，我们可以要求系统找到在address属性上的取值为"20 Yamaha Street"的实体。计算机利用属性和取值的固定格式，可以轻易做到这一点。这种基于实体和属性的结构比"键值对"结构更精细更复杂，能表达更丰富的数据获取需求。

通过以上例子，我们看到，数据管理系统的数据存取接口很大程度上取决于数据的组织结构。当数据的组织结构很简单时，系统只能提供简单的数据存取方式，如"键值对"。当数据的组织结构变得更加精细时，系统就可以提供更多样化的数据存取方式，从而满足更广泛的数据存取需求，如通过属性描述的实体。数据的组织结构有一个更正式的名称，叫**数据模型**。数据模型是人和计算机共同描述数据的方式，是对数据进行抽象之后的具体表现形式。它严格定义了数据的组织结构，也严格定义了其组织结构的含义。有了数据模型，系统就可以根据数据的组织结构去处理数据，用户就可以根据它去表达自己的数据存取需求。

数据模型选择

不同的数据模型拥有不同的数据存取接口，服务于不同的存取需求。那么，一个数据管理系统到底应该使用简单的还是复杂的数据模型呢？一方面，我们希望数据管理系统的接口简单，这样不仅易于使用，也更符合软件模块化的原则。另一方面，我们也希望数据管理系统的功能强大，从而可以承载更多的数据处理功能，满足用户更多的数据存取需求，减轻应用程序的负担。也就是说，我们希望有一个足够简单的模型，但这个模型又具备尽可能强的"表达能力"（我们用"表达能力"来衡量用户能够通过模型表述多少种类的数据存取需求）。然而，事与愿违。正如我们的例子所展示的，一个简单模型的表达能力通常较弱 -- 使用"键值对"模型，我们就只能使用键对数据进行存取。如果我们要追求更强的表达能力，就不得不去选择更加复杂的模型。但过高的复杂性是不可接受的。最极端的例子就是将程序设计语言作为描述数据的模型。使用程序设计语言，我们可以表达任意的数据结构，并指挥计算机完成任何数据处理任务（因为大部分程序设计语言是图灵完备的）。但如果我们将程序设计语言作为数据管理系统的功能接口，数据管理系统存在的意义就没有了 -- 相当于所有的数据存取工作都由程序直接完成了。

这就又回到了系统设计的折衷问题。我们需要在简单模型和复杂模型之间取一个折衷，兼顾系统的易用性和功能性。在设计数据管理系统时，数据模型的选择是至关重要的，它决定了整个系统的使用方式和实现方式，也常常是争论的焦点。根据不同的数据模型可以将数据管理系统分为不同的类型，如关系型数据库系统、文档数据库系统、键值对数据库系统、图数据库系统、时序数据库系统等等。在本书接下来的内容中，我们将展示数据模型是如何影响数据管理系统的功能的。在介绍每一个数据管理系统时，我们都将首先讲解它的数据模型。

值得指出的是，数据模型的复杂度和表达能力并不是线性关系。当两个数据模型的复杂程度相当时，并不意味着二者具备同样的表达能力。模型后面还可以有一套推理系统。表达能力还取决于这套推理系统的推理能力。一个模型的推理能力越强，它的表达能力也会随之越强。简单的说，有的模型允许我们定义抽象的规则；系统通过在规则上的自动推理可以获得更强的数据处理能力和表达能力。比如，逻辑表达式就比程序设计语言具备更强的推理能力。使用逻辑表达式，我们常常可以更加简洁地表述我们的数据处理需求。例如，我们可以用以下两条规则定义祖先：（a）一个人的父母是这个人的祖先；（b）一个人的祖先的父母也是这个人的祖先。如

果系统具备逻辑推理能力，我们只需要告诉它这个规则，就可以直接要求它找到任何一个人的祖先。但如果它不具备逻辑推理能力，我们就只能用一大堆程序去实现祖先的查找。

探讨模型复杂度和表达能力之间的关系很有意义。但这个问题已经超出了本书的范围。感兴趣的读者可以查阅人工智能教材中关于谓词逻辑、一阶逻辑和高阶逻辑的内容。

上一页<< | **>>下一页**