<<数据库原理>>

图书基本信息

书名：<<数据库原理>>

13位ISBN编号：9787302184454

10位ISBN编号：7302184453

出版时间：2008-9

出版时间：清华大学出版社

作者：(美) (克罗恩克Kroenke) (D.M.) (美) (

页数：437

前言

Colin Johnson是西雅图一家小型制造厂的产品管理员。
几年前，Colin计划构建一个数据库来跟踪产品包中的产品。
开始时，他运用电子制表软件来完成这一工作，却无法通过该表获取所需的报表。
Colin听说了Microsoft Access，希望该软件能解决问题。
经过几天的尝试，他发现无法掌握Access的使用方法，于是购买了一些流行的Access书籍并努力学习。
不过最终他还是放弃了，后来他聘请了一个顾问，该顾问创建了一个大致可满足Colin要求的应用程序。
一段时间以后，Colin想对此应用程序作一些改动，但他却不敢进行这样的尝试。
Colin是个成功的商人，他能够主动地去实现他的目标。
作为一个老练的Windows用户，他可以通过自学掌握Excel、PowerPoint以及大量面向产品的应用程序。
但他在使用Access来解决问题时却停滞不前。
Colin认为“我确信我可以做到，但我没有更多的时间”。
这样的事件非常引人注目，因为在过去的10年内上演了无数次。
Microsoft Corporation、Oracle和其他数据库管理系统(DBMS)厂商都意识到了这一情况，他们投入了数百万美元来创建更好的图形用户界面、数以百计的多窗格向导，以及众多示例应用程序。
遗憾的是，这样的努力无法从根本上解决问题。
事实上，许多用户并不清楚向导可以实现哪些功能。
一旦这些用户需要修改数据结构或是组件(例如，窗体和查询)，就会陷入麻烦中，这令他们措手不及。
如果不了解底层结构，这些用户就只能绞尽脑汁，却徒劳无功。
最终也只能得到一些设计糟糕的数据库和应用程序，而无法满足用户的要求。
那么为什么像Colin这样的用户可以学会使用文字处理软件或电子数据表这样的产品，却无法学会使用DBMS产品呢？
其中一个主要原因就是许多人都不熟悉数据库的基本概念。
每个人可能都知道段落和边距，却无法理解关系的概念。
其次，他们可能觉得使用DBMS产品一定比了解数据库概念本身更容易。
“我们想做的只是跟踪一些内容，为什么实现起来却这么困难？
”如果不了解关系模型，在存储数据之前将一张销售发票分成5个单独的表就可能使业务用户感到迷惑。
基本概念　　当今的技术特点是如果不学习基本概念，那么就不可能成功地利用DBMS。
凭借多年来为业务用户开发数据库的经验，我认为数据库的基本概念主要包括以下内容：　　● 关系模型的基本概念　　● 结构化查询语言(SQL)　　● 数据建模　　● 数据库设计　　● 数据库管理
由于当前Internet和World Wide Web的广泛使用，因此可以增加一个更为基本的概念：　　● Web数据库处理　　像Colin这样的用户(或是将接受类似工作的学生)不需要像信息系统专家那样深入地了解这些主题。
因此，《数据库原理》只介绍了一些基本的概念——这些内容对于像Colin这样的用户(创建和使用小型数据库的用户)来说至关重要。
我已经在Database Processing：Fundamentals，Designs，and Implementation一书中重写、简化和删除了一些内容1。
不过在《数据库原理》中，也力求讨论准确，不会产生误导。
即使学生们已学习过更高级的数据库课程，也仍然可从《数据库原理》中获益。
独立于DBMS产品的概念　　《数据库原理》假设学生没有使用过任何特定的DBMS产品。
《数据库原理》通过Microsoft Access、SQL Server 2005 Express Edition和MySQL 5.0举例说明了数据库概念，从而使学生可以将这些产品作为工具使用，并且实际地尝试练习书中的内容，但是，所有的概念

都适合于DBMS产品。
通过这种方式，学生们可以理解所有数据库的基础知识——从小型的Access数据库到大型的Oracle或DB2数据库。
　　而且这一方法也避免了一个常见的问题。
在同时介绍概念和产品时，学生容易将概念与产品特性、功能相混淆。
例如参照完整性约束。
在讲授基础理论时，学生都知道在某些情况下，一个表中的列值必须总是由另一个表中的列值提供。
学生也将知道这一约束如何出现在关系定义的上下文中，以及DBMS或应用程序如何强制执行这一约束。
如果结合具体的DBMS教学，如Access，那么学生将学到的知识就可能是：在某些情况下选取复选框，而在其他情况下不选取的具体操作。
这样很容易导致在介绍产品特性的同时，使学生们淡忘了数据库的基本理论。
　　然而这并不是说在《数据库原理》中将不使用DBMS。
相反，学生们可以通过使用企业版DBMS产品来熟悉这些基本概念。
《数据库原理》的这一版本包括了关于Access、SQL Server 2005 Express Edition和MySQL 5.0的充分的基本信息，从而使您可以在不借助其他书籍或资料的情况在课堂上介绍这些产品。
《数据库原理》深入介绍了Access，因为它是在个人数据库方面非常流行的产品(并且包括在Microsoft Office Professional应用程序套件中)。
通过学习这些Access相关知识，学生可以完成书中介绍的所有数据库任务，并且可能胜任一些课程的学习。
然而，如果希望深入介绍特定的DBMS或使用《数据库原理》中没有介绍的DBMS产品，则需要提供额外的书籍或资料。
Prentice-Hall提供了有关Microsoft Access 2003和其他DBMS产品的大量辅助读物，可以结合《数据库原理》一起学习。
　　Access工作台　　《数据库原理》前面的版本在附录中介绍了Access。
Access广泛用于初级数据库课程，因此这一版本介绍了如何使用Access的更多信息。
每一章都带有一个“Access工作台”部分，其中使用Access举例说明了本章的概念和技术。
“Access工作台”的第一部分介绍了如何创建数据库和简单的表，到最后的第七部分则介绍了针对Access数据库的Web数据库处理。
“Access工作台”的作用不是全面地介绍Access，而是介绍包括了所有必要的基础Access主题，从而使学生可以有效地构建并使用Access数据库。
　　复习题、练习题和实践项目　　学生能否学会最终运用所学的知识非常重要，因此每章都提供了一些复习题、练习题(包括针对“Access工作台”的练习题)和3个贯穿《数据库原理》始终的项目。
如果学生阅读并理解了每一章的内容，就应该能知道复习题的答案。
练习题要求学生将每章所讲的概念应用到具体的小问题或任务中。
　　第一个项目Garden Glory介绍了一个向个人或企业提供园艺服务的合伙公司的数据库，并讨论了该数据库的开发和使用。
第二个项目James River珠宝行分析了一家零售店为支持针对的购买者而设计的程序的数据库需求。
第三个项目Queen Anne Curiosity商店介绍了零售商店的销售和库存需求。
《数据库原理》的所有章节和附录C中都包括了这3个项目。
在每个实例中，都要求学生将各章中学到的知识运用到项目中去。
教师们可以填写《数据库原理》最后的调查表向培生出版集团驻北京代表处领取教学支持资料，其中包括与《数据库原理》相关的一些数据库和示例数据。
在过去30多年中，我们已经发现数据库和数据库应用程序的开发是一项令人愉快和有益的活动。
我们相信：数据库的数量、大小和重要性会在将来不断增加，并且该领域将越来越显示其突出的地位。
我们希望《数据库原理》中介绍的概念、知识和技术将帮助学生成功地参与到现在和多年以后的数据

库项目构建中。

对第2版的改进　　主要的改动如下：　　● 介绍了SQL视图(附录C)。

● 介绍了子类型/子类型实体(第4章和第5章)。

● 使用了IE Crow ’ s Foot E-R图而不是UML E-R模型以方便用户使用，并且与Database Processing的第10版保持一致(第4章和后面的章节)。

● 重新安排了第6章和第7章中的主题——在第6章中作为数据库管理的一部分介绍了分布式数据库和面向对象的数据库。

● 附带介绍了基于Web的数据库处理，包括创建Web页面的特定步骤和代码示例，这些Web页面显示存储在数据库中的数据(第7章)。

● 介绍了商业智能(BI)系统的概念(第7章)。

● 在每章和附录C中使用“ Access工作台 ”介绍Access的基础知识。

● 介绍SQL Server 2005 Express Edition(附录A)和MySQL 5.0(附录B)的使用。

此外，使用示例数据集充分地开发了在《数据库原理》各个部分中使用的3个示例数据库——在每一章中使用的Wedgewood Pacific Corporation和Heather Sweeney Designs，以及在“ Access工作台 ”中使用的Wallingford Motors。

这些数据库的使用提供了全书各个章节之间概念示例的连续性，并且使学生可以创建实际的数据库以试验每章中讨论的主题。

此外，还有一个没有改动的方面需要提及，即这一版中保留了第2版中增加的较为有效的规范化讨论，并且使用说明性的过程来规范化关系。

《数据库原理》的第1版介绍了规范化的基本原理，让学生自己应用这些基本原理。

第2版的第2章介绍了学生可以用于运用规范化的4步过程。

这一改动不仅简化了规范化任务，而且使规范化的基本原理更加易于理解。

因此，当前版本中沿用了这一方法。

对于需要深入了解范式的教师，《数据库原理》第5章中在讨论范式时介绍了2NF和3NF的简短定义。

主要内容　　《数据库原理》共包括7章和3个附录。

第1章解释了使用数据库的原因、数据库的组成以及开发数据库的方法。

学生们将学习数据库及其应用程序的用途、数据库相对于电子表格列表的差别和优势。

第2章介绍了关系模型，定义了基本的关系模型术语，同时还介绍了规范化原则的基本概念，并介绍了规范化过程。

第3章讲述了基本的SQL语句。

介绍了定义数据的基本SQL语句，如SQL SELECT和数据修改语句。

《数据库原理》并不介绍高级的SQL语句，只讲述一些核心的语句。

附录C介绍了SQL视图。

接下来的3章讨论了数据库设计和管理。

第4章使用实体-关系(E-R)模型解决数据建模问题，其中包括对数据建模的需求、基本的E-R术语和概念，还提供了一个简短的E-R建模示例应用程序(Heather Sweeney Designs)。

第5章讲述了数据库设计，解释了规范化的基本概念。

第4章示例中的数据模型在第5章中则被转换为关系设计。

第6章讨论了数据库管理。

构建了一个作为功能数据库的示例数据库，并且将其用作讨论数据库管理需求的示例。

本章概述了并发控制、安全性和备份及恢复技术。

数据库管理主题对所有的数据库都很重要，对个人的单用户数据库也是如此。

事实上，这些主题在某些方面对于私人数据库更为重要，因为它们没有专业的数据库管理员来确保关键任务的执行。

第6章也讨论了分布式数据库和面向对象的数据库。

最后，第7章介绍了使用基于Web的数据库处理，包括开放数据库连接(ODBC)、活动数据对象(ADO)、活动服务器页面(ASP)的讨论。

本章也讨论了可扩展标记语言(XML)的出现和基本概念，并且介绍了商业智能(BI)系统。

附录A提供了SQL Server 2005 Express Edition的简短介绍，附录B则提供了MySQL 5.0的类似介绍。在每章和附录C的“Access工作台”中都包括了对Microsoft Access的介绍。

<<数据库原理>>

内容概要

本书是数据库初学者和初级开发人员不可多得的数据库宝典，其中融入了作者对数据库深入透彻的理解和丰富的实际操作经验。
与第2版一样，本版也深入浅出地描绘了数据库原理及其应用。
但本版技术更新、实用性更强，新增的内容包括子类型/子类型实体、IE Crows Foot E-R图、基于Wed的数据库处理、商业智能系统、SQL视图、SQL Server 2005 Express Edition和My SQL 5.0的使用等。

书中介绍了在成功管理数据库系统的基本概念，包括：·关系模型的基本原理·结构化查询语言（SQL）·数据建模·数据库设计·数据库管理·Web数据库处理

作者简介

David M．Kroenke，在1967年作为Rand Corporation公司的实习生时进入了计算行业，在此之后，他的职业生涯涉及教育、工业、顾问和出版等领域。

Kroenke曾经在科罗拉多州立大学、西雅图大学教学，目前在华盛顿大学教学。
在多年的教学生涯中，他组织了数十次由大学教授参加的教学研讨会。
在1991年，International Association ofInformation System授予他“年度计算机教育家”的荣誉称号。

在工业方面，Kroenke曾经为美国空军和波音计算机服务工作，并且负责创立了个公司。
他也曾经是Microrim公司负责产品销售和开发的副主席，并且是Wall Data公司在数据库划分方面的首席技术专家。
Kroenke是语义对象数据模型的创始者，他所拥有的咨询客户包括mM公司、Microsoft、Computer Sciences公司，以及许多其他的公司和组织。

Kroenke的著作Database Processing最初出版于1977年，现在已经是第10版。
Kroenke也出版了其他许多书籍，包括经典的Business Compu~~r Systems（198i）。
他最近编写的书籍是UsingMIS的第1版。
出于对成为海员的渴望，Kroenke也编写了KnowYour Boat：The Guide tOEverything ThatMakes YourBoat Work。
Kroenke现居住在华盛顿州的西雅图市，他结过婚，有两个孩子和两个外孙。

David J．Auer目前是西华盛顿大学的College of Business and Economics（CBE）信息系统和技术服务的主管，并且是CBE的决策科学部门的讲师。
他从1981年开始在CBE中任教，教授的课程包括Quantitative Methods、Production and Operations Management、Statistics、Finance and Management Information Systems。
在1994年，他受雇于目前的CBE职位。
除了管理CBE的计算机、网络和其他技术资源之外，他还教授Management Information System课程。
Auer负责教授Principles ofManagement Information Systems and Business Database Development课程，并且负责拓展CBE的网络基础结构课程，包括计算机硬件和操作系统、远程通信和网络管理。
Auer已经和其他人合作编写了一些与MIS相关的书籍。

Auer在华盛顿大学获得了英语文学学士学位，在西华盛顿大学获得了数学和经济学学士学位，并且在西华盛顿大学获得经济学硕士学位和辅导心理学硕士学位。
Auer是美国空军军官，他还作为组织开发专家和治疗专家为Employee Assistance Program（EAP）工作。
Auer和妻子Donna居住在华盛顿州的贝灵汉市，他是当地计划委员会（Planning Commission）的成员，并且积极参与社团成长和发展的相关问题。
他有两个孩子和3个外孙。
~

<<数据库原理>>

书籍目录

章节摘录

6.1 Heather Sweeney Designs公司的数据库 图6-1给出了使用SQL Server 2005语法创建Heather Sweeney Designs公司(称作HSD)数据库的SQL语句。
该SQL语句是从图5-24中HSD数据库设计中得到的，列的约束来自于图4-21中的属性说明，参照完整性约束如图5-25所示。
图6-2给出了用于填充HSD数据库的SQL语句，同样也是用SQL Server 2005语法编写的。
最后，最终得到的HSD数据库如图6-3中SQL Server 2005 Express Edition中的图形所示。
图6-1 创建HSD数据库的SQL 语句 图6-1 (续) 图6-2 用来填充HSD数据库的SQL 语句
图6-2 (续) 图6-2 (续) 图6-2 (续) 图6-3 SQL Server 2005 Express Edition中的HSD数据库 控制、安全和可靠的必要性 从单用户数据库到大型的跨机构数据库(如航空订票系统)，不同的数据库在大小和涉及领域方面存在巨大差异。
如图6-4所示，它们在处理方式上也有所不同。
图6-4 数据库处理环境 《数据库原理》将在第7章中讨论数据库处理应用程序时详细定义和讨论图6-4中所给出的各种环境。
至此，只需要知道图6-4中所有的应用程序元素都可以在同一时间内运行即可。
当Web页面ASP(Active Server Pages)和JSP(Java Server Pages)访问数据库时，可能会生成查询、表单和报表，也可能会执行存储过程。
在COBOL、C#以及其他程序设计语言中运行的传统应用程序也可能正在处理数据库的事务。
所有这些操作都可能会导致编程代码存储到DBMS中——这是将在《数据库原理》第7章中讨论的触发器和存储过程。
同时，当所有这些程序都在运行时，必须执行这些约束(如参照完整性约束)。
最后，数据库的用户可能会达到上百甚至上千，他们可能需要一周7天、一天24小时不间断地处理数据库。
所以，有必要引入3个数据库管理功能，以管理有可能出现的混乱。
首先，必须控制并发用户的操作，确保其操作所产生的结果与期望的结果一致。
第二，必须在适当的时候执行安全措施，在适当的时候只有经过授权的用户才可以执行授权的操作。
最后，在数据库出现故障时，必须运行备份和恢复技术和程序来保护数据库，从而在必要时能快速而精确地恢复它。
我们将依次考虑以上这些情况，在第7章中还将看到在使用Web应用程序访问数据库的时候已经开始使用这些管理功能了。
6.2 并发控制 并发控制的目的是保证一个用户的工作不会对另一个用户的工作产生不合理的影响。
在某些情况下，这些措施保证了当该用户和其他用户一起操作时，所得到的结果和他单独操作时的结果相同。
在另一些情况下，这表示用户的工作按预定的方式受到其他用户的影响。
例如，在一个订货系统中，用户应该能够访问一个订单，无论此时是否有其他用户访问该订单，或者同时有上百个用户在访问，所得到的结果都相同。
另一方面，如果用户希望打印一份最新的存货清单，他就可能需要得到其他用户目前正在改动的数据，即使这些改动以后可能会被取消。
遗憾的是，还没有一种并发控制技术或机制可以处理所有的情况。
它们都有利有弊。
例如，用户可以通过锁定整个数据库而得到非常严格的并发控制，但是在这个用户进行处理时，其他任何用户都不能执行任何操作了。
这是严格的保护措施，但是这样做的代价也非常高。
正如您将看到的，也可以采取其他措施，可能这些措施更难通过编程的方式实现或执行，但是可以得到更多的吞吐量。

当然，还有其他一些扩大吞吐量的措施，但也只是对低层次的并发控制而言。
设计多用户数据库应用程序时，开发人员必须权衡这些利弊。
6.2.1 使用原子事务的必要性 在大多数数据库应用中，用户都以事务的形式提交工作，事务也称为逻辑工作单元(logical units of work，LUW)。
事务(或LUW)是对数据库执行的一系列操作，这一系列操作或者全部成功运行，或者是不执行其中任何一个操作。
在后一种情况下，数据库保持原样。
因为这些事务是以单元的形式执行的，所以有时把这样的事务称为原子(atomic)事务。
在记录新的订单时，请考虑以下的一系列数据库操作： (1) 修改客户记录，增加Amount Owed值。
(2) 修改销售员的记录，增加Commission Due值。
(3) 将新的订单记录插入到数据库中。
假设最后一步可能由于文件空间不足而失败。
如果前两个改动都得以执行，而第三步却失败了，设想这样可能引起的混乱。
客户可能被要求付款，而他从来没收到过这个订单；销售人员可能会收到关于这个订单的佣金，而订单并没有送给客户。
很明显，这3个操作需要作为整体来执行：或者全部执行，或者都不执行。
图6-5比较了下面两种情况的结果：将这些操作作为一系列独立步骤来执行(见图6-5(a))，以及作为原子事务来执行(见图6-5(b))。
注意，当自动执行这些步骤而其中一个步骤失败时，则表明没有对数据库做任何改动。
同时，还应注意应用程序必须使用命令Start Transaction、Commit Transaction和Rollback Transaction等来标记事务逻辑的边界。
对于不同的DBMS产品，这些命令的形式有所不同。
图6-5 应用一系列操作事务和多个步骤事务的结果比较 6.2.2 并发事务处理 如果一个数据库同时处理两个事务，则称这两个事务为并发事务。
尽管可能在用户看来好像是同时处理这些并发事务的，但实际情况并不是这样，因为处理数据库的机器的中央处理器(CPU)一次只能执行一个指令，因此通常这些事务是交叉执行的，也就是说，操作系统在不同任务之间来回切换CPU服务，这样在给定时间间隔内执行其中一个事务的某个部分。
任务之间的切换非常快速，所以如果有两个人并排坐在浏览器前，同时对同一个数据库进行处理，可以相信他们的事务是同时完成的。
但是，事实上这两个事务是交叉执行的。
图6-6显示了两个并发的事务。
用户A的事务是读取并修改Item 100，再将其重新写入数据库中。
用户B执行相同的操作，但对象是Item 200。
CPU处理用户A的事务，直到它必须等待读取或写入操作完成，或者等待其他的操作完成。
然后，操作系统切换到用户B的控制上。
然后，CPU开始处理用户B的事务，直到发生类似的对事务处理的中断，这时操作系统又将控制权转回到用户A上。
同样的，对用户来说，这一处理过程看起来是同步的，但它实际上是交叉执行的，或者说是并发的。
图6-6 两个用户的任务并发处理的示例 6.2.3 丢失更新问题 因为用户在处理不同的数据，所以图6-6所示的并发处理不会出现什么问题。
但是假设两个用户同时都希望对Item 100进行操作。
例如，用户A希望订购5套Item 100，而用户B则希望订购3套。
图6-7描述了这个问题。
用户A读取Item 100的记录，该记录传送到用户的工作区。
根据记录，还有10套存货。
然后，用户B读取Item 100的记录，这些数据又传送到该用户的工作区。
同样，根据记录，还有10套存货。

现在用户A提取其中的5套，在他的用户工作区中将存货数目减为5，并将记录重新写入Item 100。
用户B提取了3套，在他的用户工作区中将存货数目减为7，并将记录重新写入Item 100。
这样数据库将出现错误，显示还有7套Item 100库存。
回顾该流程，开始库存是10，用户A提取了5套，用户B提取了3套，数据库最后显示库存中还剩7套。
显然，这是错误的。
　　在两个用户刚获得数据时，这些数据是正确的。
但是，当用户B读取记录时，用户A已经有了需要更新的副本。
这种情况称为丢失更新问题(lost update problem)或并发更新问题(concurrent update problem)。
另一个类似的问题称为非一致读取问题(inconsistent read problem)。
在这种情况下，用户A读取的数据已经被用户B的部分事务处理过。
其结果是用户A读取了错误的数据。
　　图6-7　丢失更新问题的示例　　6.2.4　并发问题：脏读取、不可重复读取和幻象读取　　并发处理有可能引起一些问题，这些问题都有其标准化的名字：脏读取、不可重复读取和幻象读取。
当一个事务读取了一个尚未提交到数据库但已经过修改的记录时，这种情况称为脏读取(dirty read)。
比如，在下面的情况中就会发生脏读取：如果一个事务读取一行数据，而这行数据被另一个事务修改过，但是后来第二个事务又取消了修改。
如果事务重新读取以前读取过的数据，并且发现另一个事务对其进行了修改和删除，这称为不可重复读取(nonrepeatable read)。
当事务重新读取数据，但发现在读取该数据后已有另一事务插入了新的数据行，这就称为幻象读取(phantom read)。
　　6.2.5　资源锁定　　可以通过一种办法来弥补由并发处理引起的不一致问题，即在将要修改某些数据行或表时禁止多个应用程序同时获取这些行或表的副本。
这个方法称为资源锁定(resource locking)。
该方法是锁定将要更新的数据，从而禁止共享，这样就防止了并发处理问题的发生。
图6-8给出了使用锁定命令的处理步骤。
　　因为数据被锁定，用户B的事务必须等待，直到用户A处理完Item 100的数据为止。
在运用该策略后，只有在用户A完成对Item 100的修改后，用户B才可以读取Item 100的记录。
在这种情况下，最终保存在数据库中的Item 100的数目是2，这才是正确的数字(一开始的数据是10，然后A取走5，B取走3，最后剩下2)。
　　图6-8　带有显式锁定的并发处理示例　　锁定可以由DBMS自动执行，也可以由应用程序或查询用户发给DBMS的命令执行。
DBMS执行的锁定称为隐式锁定(implicit lock)；而命令执行的锁定称为显式锁定(explicit lock)。
　　在前面的示例中，数据行被锁定。
但是，不是所有的锁定都应用在这个级别上。
一些DBMS产品的锁定针对的是页面级，有些针对表，有些则针对数据库。
锁定的规模称为锁定粒度(lock granularity)。
对于粒度较大的锁定，DBMS容易管理，但是经常引发冲突。
而较小粒度的锁定则不易管理(DBMS需要跟踪和检查许多细节内容)，但是较少引发冲突的情况。
　　锁定也因类型的不同而各异。
排它锁定(exclusive lock)可以锁定对产品项的任何类型的访问。
其他任何事务都不可以读取或修改数据。
共享锁定(shared lock)锁定的是正在修改的产品项，但是不拒绝对数据的读取。
这就是说，其他事务可以读取被锁定的产品项，只是不能修改数据。
　　6.2.6　串行化事务　　并发处理两个或多个事务时，所得数据库的结果在逻辑上应该和事务以任意串行方式处理后所得到的结果保持一致。
以这种方式处理并发事务的模式称为串行模式。
　　实现串行化的方式有许多种。

一种方式是使用二段锁定(two-phased locking)处理事务。
采用这种策略时，事务可以根据需要决定是否使用锁定，但是一旦释放了第一个锁定，就不可以再使用其他任何锁定。
因此，事务就有一个增生阶段(growing phase)，即获取锁定的阶段，还有一个收缩阶段(shrinking phase)，即释放锁定的阶段。
许多DBMS产品使用一种特殊的二段锁定。
如果使用了这类锁定时，那么在事务的执行过程中都可以进行锁定，但是直到发出COMMIT或ROLLBACK命令时才能释放锁定。
这种策略比二段锁定的要求更严格，但更容易实现。
考虑一个订货事务，该事务涉及处理CUSTOMER表、SALESPERSON表和OREDER表中的数据。
为了确保数据库不会因为并发行为而产生异常，订货事务在需要时可以锁定CUSTOMER表、SALESPERSON表和ORDER表，修改所有的数据库，然后释放所有的锁定。
6.2.7　死锁　　尽管锁定解决了一个问题，但同时也产生了另一个问题。
考虑两个用户都希望从库存中订购两个产品项时可能发生的情况。
假设用户A希望订购一些纸张，如果她可以得到纸张，则可能还需要再订购一些铅笔。
同样假设用户B希望订购一些铅笔，如果他可以得到铅笔，也可能需要再订购一些纸张。
处理过程如图6-9所示。
在图6-9中，用户A和用户B都被锁定的状态称为死锁(deadlock)，有时称为致命包含(deadly embrace)。
其中每个用户都在等待另一个用户已经锁定的资源。
解决这个问题通常有两种方法：阻止死锁发生，或者允许死锁发生，然后打开该死锁。
图6-9　死锁示例　　有多种方式可以防止死锁的发生。
一种方法是允许用户一次只发出一个锁定请求；事实上，用户必须立刻锁定当前所需的所有资源。
如果示例中的用户A一开始就同时锁定了纸张和铅笔记录，则死锁无论如何不会发生。
第二种防止死锁的方法是规定所有应用程序锁定资源的顺序必须完全相同。
几乎每一个DBMS都有检测死锁的算法。
当死锁发生时，一般的解决方法是回滚其中的一个事务并取消它对数据库所做的改动。
6.2.8　乐观锁定和悲观锁定　　激活锁定的方式一般有两种。
没有冲突发生时采用乐观锁定(optimistic locking)。
读取数据，处理事务，执行更新，然后检查是否发生冲突。
如果不存在冲突，事务就可以完成。
如果存在冲突，则重复事务，直到没有冲突。
采用悲观锁定(pessimistic locking)时，假定存在冲突。
首先使用一个锁定，处理事务，然后释放锁定。
图6-10和图6-11显示了使用这两种方式处理同一个示例的情况。
在这个示例中，事务将PRODUCT表中铅笔行的数量减少5。
图6-10显示了乐观锁定的情况。
首先读取数据，并且将铅笔的Quantity当前值存入OldQuantity变量。
然后处理事务，假设所有这些操作都得以顺利实现，则PRODUCT就得到一个锁。
这个锁可能只对铅笔行起作用，或者可作用于更高的粒度级别。
无论何种情况，都将执行一条更新铅笔行的SQL语句，其中的WHERE条件指定Quantity的当前值等于OldQuantity值。
如果没有其他的事务修改铅笔行的Quantity值，那么这个UPDATE语句就可成功执行。
如果有另一个事务修改了铅笔行的Quantity值，则UPDATE操作将会失败，需要重新执行事务。
图6-10　乐观锁定示例　　图6-11描述了对相同事务采用悲观锁定的逻辑。
在这种情况下，在所有操作开始运行前，PRODUCT(在某个粒度级别上)得到一个锁定。
然后读取数值，处理事务，执行UPDATE语句，最后再解除对PRODUCT的锁定。

乐观锁定的优点是当事务处理完后才获取锁定。
因此，锁定的持续时间比悲观锁定更短。
如果事务复杂或客户机的速度较慢(因为传输延迟或用户在执行其他工作、用户休息或者没有退出应用程序就关机等)，锁定持续的时间就会相对少很多。
如果锁定粒度大(如整个PRODUCT表)，这个优点就显得更为重要。
图6-11　悲观锁定的示例　　乐观锁定的缺点是，如果针对铅笔行同时有很多操作，事务就可能必须重复执行很多次。
因此，如果事务需要对指定行执行很多操作(比如购买一种热门股票)，就不适合使用乐观锁定。
6.2.9　声明锁定特征　　并发控制是一个复杂的问题；一些锁定的类型和策略必须经过反复试验和纠正之后才能决定。
因为这个以及其他原因，数据库应用程序一般不显式地使用锁定。
相反，程序标记事务边界，然后声明它们希望DBMS使用的锁定行为的类型。
通过这种方法，如果需要改变锁定行为，就不需要重新编写应用程序来锁定事务的不同区域，而只需修改锁定声明。
图6-12描述了一个铅笔事务，以BEGIN TRANSACTION、COMMIT TRANSATION和ROLLBACK TRANSACTION等语句标记事务的边界。
这些边界提供了DBMS执行不同锁定策略所需的基本信息。
如果现在开发人员声明(通过系统参数或类似途径)需要乐观锁定，那么DBMS将隐式地在适当的位置上设置锁定类型。
如果后来希望改为悲观锁定，那么DBMS会隐式地将悲观锁定设置在另一个位置。
图6-12　标记事务边界的示例　　6.2.10　一致事务　　有时候，会看到首字母缩略词ACID运用到事务上。
所谓ACID事务是一种原子性的(atomic)、一致的(consistent)、隔离的(isolated)和持久的(durable)事务。
原子性和持久性都容易定义。
原子事务是指或者执行事务中所有的数据库操作，或者不执行其中任何一个操作。
持久性事务是指事务中所有提交的修改都是永久的。
DBMS不会取消这些修改，即使在发生错误时也是如此。
如果事务是持久的，那么DBMS将在需要时提供辅助措施来恢复在所有提交操作中进行的修改。
术语一致性和隔离性不像原子性和持久性那样容易定义。
请考虑以下的SQL更新命令：　　UPDATE　　CUSTOMER　　SET　　AreaCode = 425　　WH
=98050　　假设在CUSTOMER表中有500000行，其中有500行的ZipCode值均为98050。
那么DBMS需要花费一些时间才能找到这所有的500行。
在这段时间内，允许其他事务更新CUSTOMER的AreaCode或ZipCode字段吗？
如果SQL语句是一致的，这样的更新就是不允许的。
如果在SQL语句执行时找到了匹配行，则将对这些行集进行更新。
这种一致性称为语句级一致性(statement level consistency)。
现在考虑包含两个SQL更新语句的事务：　　BEGIN TRANSACTION　　UPDATE CUSTOMER　　SET AreaCode=425　　WHERE ZipCode=98050　　. . .　　{other transaction work}　　. . .　　UPDATE CUSTOMER　　SET Discount=0.05　　WHERE AreaCode=425　　. . .　　{other transaction work}　　. . .　　COMMIT TRANSACTION　　在这个上下文中，“一致性”又表示何种意义呢？
语句级一致性表示不同的语句将独立处理一致的数据行，但是在这两个SQL语句执行的间隔中，允许其他用户对这些数据行进行修改。
事务级一致性(transaction level consistency)意味着，两个SQL语句作用到的所有数据行在整个事务运行期间都受到保护。
然而，也可以发现，在实现事务级一致性时，事务本身并不能发现自身的变化。
在上面的示例中，第二个SQL语句并不知道有些数据行已经被第一个SQL语句改变。
因此，当听到“一致性”这个术语时，需要进一步确定它属于哪种一致性。

此外，也需要小心处理事务级一致性的潜在问题。
隔离(isolated)的情况更为复杂，我们将在接下来的章节中讲解。
6.2.11 事务隔离级别 1992年，ANSI SQL标准定义了4个事务隔离级别(isolation level)，这种隔离标准指定了允许产生哪种并发控制问题。
图6-13展示了这4个隔离级别。
隔离级别 读取未提交 读取已提交 可重复读取 可串行化 问题 类型
脏读取 可能 不可能 不可能 不可能 不可重复读取 可能 可能 不可能 不可能 幻象读取 可能 可能 可能 不可能 图6-13 隔离级别汇总 使用这4个隔离级别的目的是让应用程序编程人员能够声明将使用的事务隔离级别，然后由DBMS通过管理锁定来实现相应的事务隔离级别。
如图6-13所示，“读取未提交”隔离级别允许“脏读取”、“不可重复读取”和“幻象读取”。
在“读取已提交”隔离级别中，不允许“脏读取”。
在“可重复读取”隔离级中，禁止“脏读取”和“不可重复读取”。
而在“可串行化”隔离级别中，禁止上面的3种操作。
虽然吞吐量也和工作负荷以及应用程序的编写方式息息相关，但从总体来说，隔离级别越高，吞吐量越小。
而且，不是所有的DBMS产品都会同时支持这4种隔离级别，产品对它们的支持方式和对应用程序开发人员的要求也各有不同。
6.3 游标类型 游标(cursor)是一种指针，指向从SQL SELECT语句得到的结果行集，它常常通过SELECT语句来定义。
比如，下面的语句定义了一个名为TransCursor的游标，该游标的操作对象是由SELECT语句指定的行集：
DECLARE CURSOR TransCursor AS SELECT * FROM [TRANSACTION] PurchasePrice > 10000; 在应用程序打开某个游标后，该程序可以把该游标放到结果集中的某个地。
最常见的情况是，把游标放到第一行或最后一行，但是也可能放在其他地方。
一个事务可以打开多个游标——可以依次打开，也可以同时打开。
此外，可能打开同一个表的两个或多个游标；可能直接打开关于该表的游标，也可能是通过关于该表的SQL视图来打开。
由于游标需要相当大的内存，因此同时打开很多游标(比如，为一千个当前事务打开游标)将消耗大量的内存。
减少游标所带来负担的一个方法是定义缩容(reduced-capability)游标，并在不需要满容量游标的时候使用它们。
图6-14给出了Windows环境中所使用的4种游标类型(其他系统中的游标类型也与此类似)。
最简单的游标是“只向前游标(forward only cursor)”。
通过该游标，应用程序只能向记录前面执行。
对于由事务中其他游标或其他事务所做的修改，只有当它们发生在游标所在行之前才是可见的。
另外3种游标类型又被称为“可滚动游标(scrollable cursor)”，因为应用程序可以向记录前面或后面滚动。
静态游标(static cursor)快速绘制并处理关系的图，使用这种游标进行的修改是可见的，而来自其他源的修改都是不可见的。
动态游标(dynamic cursor)是一种很有特色的游标。
对于动态游标而言，所有以行的顺序进行的插入、更新、删除和修改都是可见的。
除非事务的隔离级别是脏读取，否则只有提交的修改才是可见的。
键集游标(keyset cursor)结合了静态游标和动态游标的一些特点。
打开该游标后，将保存每一行的主键的值。
当应用程序把游标放在行中时，DBMS使用该键值来读取行的当前值。
由本事务或其他事务中的其他游标所插入的新行是不可见的。

如果应用程序发出一个对行的更新命令而该行已经被其他游标删掉了，那么DBMS将创建一个新行，该行保持旧的键值，并把更新的数值插入到新行中(假设所有必需的字段都存在)。
对于动态游标而言，除非事务的隔离级别是脏读取，否则只有已提交的更新和删除对于游标来说是可见的。
对于每种类型的游标来说，支持游标所需的系统开销和处理的数量也不相同。
一般来说，开销随着类型的下降(如图6-14所示)而增加。
因此，为了提高DBMS的性能，应用程序开发人员应该创建足以完成工作的游标即可。
同时，了解特定的DBMS如何实现游标以及游标是否位于服务器或客户机上也是非常重要的。
有些情况下，把动态游标放到客户机上可能比把静态游标放到服务器上要更好。
因为性能取决于DBMS产品以及应用程序需求所使用的实现方式，所以并没有规定一般规则。
游标类型 说明 注释 只向前游标 应用程序只能向记录前面移动 对于由事务中其他游标或其他事务所做的修改，只有当它们发生在游标所在行的前面时才是可见的 静态游标 应用程序在打开游标时才可以看到数据 由静态游标所做的修改是可见的。
来自其他源的修改是不可见的。
允许向前和向后的滚动 键集游标 当打开键集游标时，每行的主键都保存到了记录集中。
当应用程序访问行时，该键用来取回该行的当前值 来自任何源的更新都是可见的。
来自键集游标外部的源的插入是不可见的(因为在键集中没有这些源的键)。
来自键集游标的插入显示在该记录集的底部。
来自任何源的删除都是可见的。
而以行的顺序排列的修改都是不可见的。
如果隔离级别是脏读取，那么提交的更新和删除都是可见的；否则只有提交的更新和删除才是可见的
动态游标 任何类型的修改以及来自任何源的修改都是可见的 所有按记录集顺序进行的插入、更新、删除和修改操作都是可见的。
如果隔离级别是脏读取，那么未提交的修改是可见的。
否则，只有提交的修改才是可见的 图6-14 游标类型的汇总 需要小心的是：如果没有指定事务的隔离级别或没有指定被打开游标的类型，那么那么DBMS将使用默认的级别和类型。
这些默认的级别和类型对于您的应用程序来说可能非常适合，但是它们也可能成为麻烦。
因此，即使可以不考虑这些问题，但是这些问题所带来的后果也是无法回避的。
所以必须搞懂DBMS产品的功能。

6.4 数据库安全

数据库安全的目的就是确保只有授权用户才可以在授权时间内进行授权的操作。
这一目标通常分为两个部分：身份验证(即确保用户拥有首先使用该系统的基本权限)和授权(即授予用户对系统进行特定操作的具体权限和许可)。
如图6-15所示，用户验证要求用户通过密码(或其他明确的身份证明，如手指纹的生物仪器扫描等)登录到系统中，而用户授权则是通过授予用户DBMS特定的许可而达到的。
图6-15 数据库安全的身份验证与授权 注意，仅仅通过身份验证(用户登录到系统中)对于数据库的使用来说还是不够的——除非用户还获得了相应的许可，否则他(或她)就不可以访问数据库或执行使用数据库的操作。
因此，实现数据库安全的目标很难，但是无论执行哪些改进，数据库开发小组都必须确定(1)哪些用户可以使用数据库(即进行身份验证)；(2)每个用户的处理权限和责任。
这些安全需求可以通过DBMS的安全特性以及写入应用程序的附加安全特性来实现。

6.4.1 用户账户

比如，考虑示例Heather Sweeney Designs公司的数据库安全需求。
必须采取措施来控制访问数据库的员工。
其中一个方法是为每个员工创建用户账户。
图6-16展示了SQL Server 2005中在DBMS安全级别上创建用户登录HSD-User的情况。
图6-16 数据库服务器登录的创建 该步骤所创建的是DBMS(而不是特定的数据库)中最初的用户账户，相应赋予的密码是HSD-User+password，在第7章HSD Web页面中也需要这一密码。

注意，在Windows环境中，控制身份验证的方法有两种：第一种，使用Windows操作系统来控制身份验证；第二种，通过账户的登录名和密码来创建SQL Server内部用户账户。
对于其他DBMS产品而言，如果没有像SQL Server一样被操作系统指定，那么只能使用第二种方法，即使用内部用户账户。
应该小心管理用户账户和密码。
DBMS账户和密码安全的正确术语、特性和功能取决于所使用的DBMS产品。
6.4.2 处理权限和责任 所有主流的DBMS产品都提供了限制特定客户对特定对象进行特定操作的工具。
DBMS安全的一般模型如图6-17所示。
图6-17 DBMS安全模型 根据图6-17所示，可以赋予用户一个或多个角色(组)，一个角色可以拥有一个或多个用户。
用户、角色和对象(使用的是其通用的含义)拥有许多许可。
每个许可然后被赋予一个用户或角色以及一个对象。
一旦用户通过了DBMS的身份验证，DBMS将把该人的行为限制在用户确定的许可范围内，或限制在用户所赋予的角色的许可范围内。
现在，考虑Heather Sweeney Designs公司的用户身份验证问题。
该公司存在3类用户：administrative assistant(行政助理)，management(经理，即Heather)，systems administrator(系统管理员，即Heather的顾问)。
图6-18是Heather根据她的业务需要确定的处理权限。
行政助理可以读取、插入和修改所有表中的数据，但是只可以删除SEMINAR_CUSTOMER和LINE_ITEM中的数据。
这就意味着，他们有权利从培训课中删除客户，并可以从订单中删除产品项。
经理除了不能删除CUSTOMER数据外，可以对所有的表执行所有操作。
这里Heather考虑到，鉴于赢得客户的不易，她不愿因为不小心删除了CUSTOMER数据而失去客户。
最后，系统管理员可以修改数据库的结构，对其他用户进行授权(授予权限)，但不能对数据进行操作。
系统管理员不是用户，因此不允许其访问用户数据。
这一限制似乎很有限。
毕竟，如果系统管理员可以指定权限，他(或者她)可以通过改变权限来绕开安全系统，从而可以进行所需要的任何操作，修改数据，然后将权限再修改回去。
这些情况都有可能发生，但是会在DBMS日志中留下审计跟踪。
这些以及对安全系统进行改变的需求可以阻止系统管理员进行未授权的活动。
这明显优于允许系统管理员具有无效的用户数据访问权限。

<<数据库原理>>

编辑推荐

本书中介绍了在成功管理数据库系统的基本概念，包括：关系模型的基本原理、结构化查询语言（SQL）、数据建模、数据库设计、数据库管理、Web数据库处理。

版权说明