第7章　完整性与安全性

完整性和安全性是数据库系统在数据控制和数据保护方面提供的辅助功能。完整性保证授权用户对数据库进行修改时不会破坏数据的一致性，即防止对数据的意外破坏。安全性则防止保存在数据库中的数据未经授权的访问和恶意的破坏与修改。换句话说，安全性保证授权用户能做他们想做的事，而完整性则保证他们想做的事的正确性。

当然，完整性和安全性也有相似的地方：第一，系统都要给用户规定某些必须遵守的约束条件；第二，这些约束条件一般由DBA使用某种适当的语言确定下来，必须在系统词典中维护；第三，DBMS必须使用某种方式来监督用户，以确保他们遵守这些约束条件。本章将详细介绍完整性约束条件和安全性约束条件以及相应的实现机制。

7.1 域 约 束

数据库的完整性是指数据库的正确性和相容性。DBMS提供一种功能来保证数据库的完整性，这种功能称为完整性检查，即系统用一定的机制来检查数据库中的数据是否满足规定的条件，这种条件在数据库中称为完整性约束条件。数据库的完整性控制机制通常包括两部分内容，即对完整性约束条件的定义功能和对完整性约束条件的检查功能。

数据库的完整性约束条件又可以分为域约束和结构约束、静态约束和动态约束，以及立即执行约束和延迟执行约束。本节介绍的即是最基本的完整性约束——域约束。在数据库中，每个属性都必须对应一个所有可能取值构成的域，如SQL中定义的

整型、字符型、日期/时间型。声明一种属性属于某种具体的域就相当于约束它可以取的值，包括值的类型、范围、精度等等。每当有新数据项插入到数据库中，系统就能方便地进行域约束检测。

域约束的恰当定义不仅可以对插入数据库的值进行检测，而且可以对查询进行检测，以保证所做的查询是有意义的。例如，当多个属性有相同的域时，虽然从物理实现上没有区别，但是在逻辑上就可能出现语义错误。比如，对不同货币的值未经转换就直接进行比较，就会导致语义错误。属性域在原理上非常类似于编程语言中变量的类型。可以用CREATE DOMAIN子句来定义域。

例7.1 为不同的流通领域声明不同的域。

```
CREATE DOMAIN Yuan NUMERIC(12, 2)
CREATE DOMAIN Dollars NUMERIC(12, 2)
```

上面两条语句定义人民币和美元域为共有12位的十进制数字，小数点后有两位。虽然它们都是同一种数字类型，但是当把一个人民币类型的值赋值给美元类型的变量时会产生一个语法错误。这样的一个赋值很有可能是由于程序员忘了不同货币的差别而导致的编程错误。域约束可以帮助找到这样的错误。除了CREATE DOMAIN子句，还可以通过DROP DOMAIN和ALTER DOMAIN语句对以前创建的域进行修改和删除，同时，域中的值也可以通过CAST子句转换到另一个域中。

在SQL中，CHECK子句允许对域作强制约束，允许数据模式的设计者指定一个谓词，使得对类型属于该域的变量所赋的任意值都必须满足该谓词，如例7.2所示。

例7.2 用CHECK子句限制小时工资域的值必须大于最低工资。

```
CREATE DOMAIN HourlyWage NUMERIC (5, 2)
CONSTRAINT wage-value-test CHECK (value>=4.00)
```

其中，子句CONSTRAINT wage-value-test是可选的，它用来对该约束进行命名，可用于指出一个更新违犯了哪条约束。

例7.3 用CHECK子句限制域不允许取空值。

CREATE DOMAIN AccountNumber CHAR(10)

CONSTRAINT account-number-null-test CHECK (value NOT NULL)

例7.4 用IN子句限制域的取值范围。

CREATE DOMAIN AccountType CHAR (10)

CONSTRAINT account-type-test CHECK (value IN ('Checking', 'Saving'))

在一个元组插入或修改的时候可以很容易检测CHECK条件，但是，当CHECK条件中允许出现包含其他关系的子查询时，CHECK条件可能变得更复杂，检测它们的开销也会很大。因此，数据库需要提供更为有效的机制来保证数据库的完整性。

7.2 参照完整性

数据库的结构约束是关于数据之间联系的约束。由于数据库中同一关系的不同属性或者不同关系的属性之间都可以有一定的联系，因而它们也应满足一定的约束条件,包括属性之间的函数依赖、多值依赖、关系模型的实体完整性和参照完整性等。

7.2.1 基本概念

如果一个关系A中给定属性集t_1上的取值也在另一关系B的某一属性集t_2的取值中出现，那么称关系A中的t_1参照关系B中的t_2，关系A的表称为参照表，关系B的表称为被参照表。关系间的参照关系根据实体集的不同会出现不同的情况，下面逐一进行分析。

关系A中存在元组r，它的属性t_1的取值不在关系B的属性t_2的取值中出现，则元组r被称为悬挂元组。悬挂元组有时会破坏数据库的完整性，例如在连锁超市管理系统中，营业员的工作信息参照加盟店，如果营业员的工作加盟店编号不在加盟店表中存在，那么营业员就是在一个不存在的加盟店工作，显然，应该避免这种情况。

如果某个属性或属性集不是关系A的主码，但它是另一个关系B的主码，则该属性或属性集称为关系A的外码。如果存在外码，就可以避免悬空元组对数据完整性的破坏。假设将营业员的工作信息参照加盟店的主码，则营业员均在已有的加盟店工作。但是，当营业员的工作信息设为空值或者仅为部分加盟

店的外码时，可能存在加盟店没有营业员的情况，这在加盟店刚刚成立或者即将倒闭的时候是有可能的，因而这种悬挂元组是允许存在的。

以上两种情况的区别在于是否存在外码。在关系模型中，外码的取值或者为空值或者为参照关系中某个元组的主码值，那么这种约束称为参照完整性约束或子集依赖。参照完整性约束避免了数据库中悬挂元组的存在，因而保证了数据的正确性和相容性。

在对数据库进行修改时，数据库系统会采用自动的机制来保证参照完整性不被破坏。当向表中插入元组时，系统必须保证外码的取值来自被参照的表中主码的取值；当从表中删除元组时，则系统必须计算参照表中的元组集合。如果集合非空，则撤销删除命令或者进行级联删除，即删除具有参照关系表中的所有对应的元组。

在对数据库进行更新时，对参照关系表(A)的更新以及对被参照关系表(B)的更新要分别进行考虑：如果参照关系表(A)中的元组被更新，并且该更新修改外码的值，则与插入情况类似，新值的取值必须来自被参照表(B)中主码的取值；如果被参照关系表(B)中的元组被更新，并且该更新修改主码的值，则与删除情况类似，系统必须用旧值(更新前的值)计算参照关系中的元组的集合，如果该集合非空，则更新被拒绝，或者以类似删除的方式做级联更新。

7.2.2 E-R模型和SQL中的参照完整性

在E-R模型中，由关联集得到的每一个关系都有参照完整性约束，如图7.1所示。其中,关联集R的关系模式的属性包括每一个相关实体集的主码$K_1 \cup K_2 \cup \ldots \cup K_n$，对于每一个实体$i$，$R$模式中的$K_i$是参照$K_i$在由实体集$E_i$产生的关系模式中的外码。

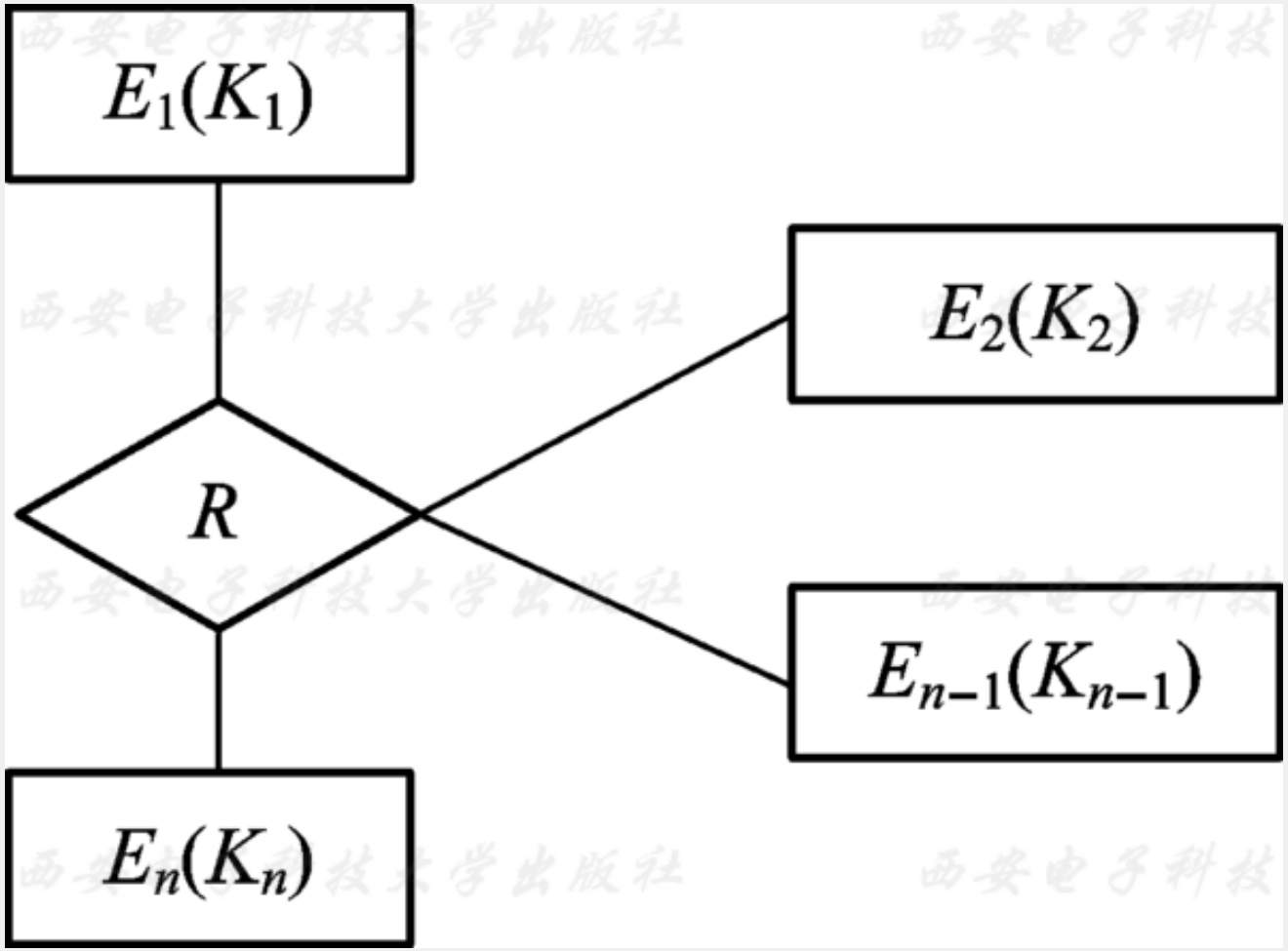

图7.1 n元关联集

另一种参照完整性约束的来源是弱实体集。弱实体集的关系模式必须包含它所依赖的实体集的主码。因此，每个弱实体集的关系模式包含一个导致参照完整性约束的外码。

在SQL规范中，外码可以用FOREIGN KEY子句指明，通过REFERENCE指明被参照的关系，作为CREATE TABLE语句的一部分。

例7.5 在连锁超市加盟店系统中，营业员的外码定义。

```
CREATE TABLE chain (...
chain_id CHAR(10), …)
CREATE TABLE employee (…
FOREIGN KEY (chain_id) REFERENCE chain, …)
```

默认状态下，一个外码参照被参照表中主码的属性。REFERENCE子句可以指明被参照关系的一组属性，这一组指定的属性可以定义为被参照关系的候选码。

当数据操作可能违反参照完整性约束时，通常的做法是拒绝执行导致破坏完整性的操作。但是，也可以在FOREIGN子句中指明：如果被参照关系上的一个删除或更新动作违反了约束，则采取一些步骤修改参照关系中的元组来恢复完整性约束，而不是拒绝这一动作。

例7.6 在连锁超市加盟店系统中，营业员的外码定义。

```
CREATE TABLE employee (…
FOREIGN KEY (chain_id) REFERENCE chain
ON DELETE CASCADE
ON UPDATE CASCADE, …)
```

在例7.6中，ON DELETE CASCADE子句指明如果删除chain中元组将导致参照完整性约束被违反，则对employee关系做“级联”删除，即删除参照了chain中被删除元组的那些元组。更新操作与此类似。

如果有涉及多个关系的外码依赖链，则在链一端做的删除或更新操作可能传至整个链。如果一个级联更新或删除导致的对约束的违反不能通过进一步的级联操作解决，则系统中止该事务。于是，该事务所做的所有改变及级联动作都将被撤销。

除了级联删除。还可以通过SET子句(代替CASCADE)将参照字段(chain_id)置为空(SET NULL)或置为域的默认值(SET DEFAULT)。

例7.7 在连锁超市加盟店系统中，营业员的外码定义。

```
CREATE TABLE employee (…
FOREIGN KEY (chain_id) REFERENCE chain
ON DELETE SET NULL, …)
```

允许NULL值的存在使SQL中参照完整性约束的语义变得很复杂，并且有时不一定能保证数据的正确性。为了避免这种复杂性，最好将外码的所有列声明为非空。一个事务也许包括几步，在一步之后完整性约束也许会暂时被破坏，但是后面一步也许就会去掉这个违反。

例7.8 将编号为“95001”的商品编号修改为“95012”，将供应关系表supply中的商品编号修改为“950012”。

```
BEGIN TRANSACTION
UPDATE commodities SET Cno="95012"
WHERE Cno="95001";
UPDATE supply SET Cno="950012"
WHERE Cno="95001";
COMMIT
```

如果立即执行每一条语句，由于供应表supply中的商品编号在第一条更新语句结束后以级联更新为“95012”，因此系统无法执行第二条更新语句。但是，如果在事务结束后执行完整性约束，则供应表supply中的商品编号可以更新为“950012”。因此，完整性约束不是在事务的中间步骤上检查，而是在一个事务结束的时候检查。

7.3 断　言

一个断言(Assertion)就是一个谓词，表示数据满足的条件。域约束和参照完整性约束是断言的特殊形式，它们易于检测并且适用于很多数据库应用。除此之外，断言还可以提供更为复杂的条件表达式，方便用户自定义完整性约束。SQL的断言采用CREATE ASSERTION子句定义断言的名称，用CHECK子句定义断言的内容。

例7.9 每个连锁超市加盟店的进货商品数量必须少于库存商品数量。

```
CREATE ASSERTION income-constraint CHECK
  (NOT EXISTS (SELECT * FROM chain
    WHERE (SELECT number FROM supply
      WHERE supply.chain_id=chain.chain_id)
        >= (SELECT number FROM commodity
          WHERE supply.cid=commodity.cid)))
```

在例7.9中，由于SQL不提供全称谓词，只好使用等价的“NOT EXISTS”结构，通过嵌套的SQL语句实现。

在创建断言时，系统会立即检测其有效性。如果断言有效，则以后只允许不破坏断言的数据库修改操作。如果断言较复杂，检测会带来相当大的开销。因此，使用断言应该特别小心。一些系统开发员省去了对一般性断言的支持，或只提供易于检测的特殊形式的断言，以减少断言的检测和维护代价。

7.4 触 发 器

域约束、参照完整性约束和断言都是静态约束，即对数据库每一确定状态的数据所应满足的约束条件。除此之外，数据库还应维护动态约束，即当数据库从一种状态转变为另一种状态时，新、旧值之间所应满足的约束条件。例如，更新职工的工资时要求新工资不能低于旧工资值。

动态约束又可分为立即执行约束和延迟执行约束。前者当事务中某一更新语句执行完后，马上对此数据所应满足的约束条件进行完整性检查；而后者是在整个事务执行结束后，再对约束条件进行完整性检查，结果正确才能提交事务。

触发器就是当对数据库进行修改时，自动执行的语句。数据库系统可以利用触发器机制对数据的完整性和安全性进行检查。对不同类型的数据对象，需要定义不同的触发器。一般触发器采用“事件—条件—动作”模型：

(1) 指明在什么条件下触发器被执行。它被分解为一个引起触发器被检测的事件和一个触发器执行必须满足的条件。

(2) 指明触发器执行时采取的动作。

触发器在数据库中作为存储过程被存储起来，可以被所有的数据库操作访问。只要指定的事件发生，相应的条件被满足，数据库系统就会执行对应的触发器。

对于示警或满足特定条件时自动执行的某项任务来说，触发器是非常有用的机制。例如，一个仓库希望每种商品存货保持一个最小量；当某种商品存货少于最小值的时候，自动发出一个订货单。触发器就可以这样执行：在更新某种商品存货的时候，触发器会比较这种商品的存货数量和它的最小值，如果存货数量等于或小于最小值，一个新的订单就会被追加到订单关系表中。

为了在数据库之外的其他系统中订货，还需另外创建一个持久运行的系统进程来周期性扫描订单关系表并订购产品。这个系统进程也要注意订单关系表中哪些元组已经被处理了以及每个订单什么时间订的货。这个进程时刻跟踪订单的交货情况，当意外发生时(如交货延迟了)负责警示处理。

7.4.1 SQL中的触发器

SQL中的触发器功能很强，不仅进行完整性检查，而且包含安全性控制的定义。这里，采用SQL 2003的触发器语法，举例说明触发器中的完整性定义和检查功能。

例7.10 重新订购商品的触发器。

```
CREATE TRIGGER recorder-trigger AFTER UPDATE
OF amount on inventoryREFERENCING old row AS orow, new
row AS nrow
    FOR EACH ROW
    WHEN nrow.level<=(SELECT level
        FROM minlevel
        WHERE minlevel.item=orow,item)
```

```
AND orow.level>(SELECT level
    FROM minlevel
    WHERE minlevel.item= orow.item)
BEGIN
    INSERT INTO orders
    (SELECT item, amount
    FROM reorder
    WHERE reorder.item = orow.item)
END
```

其中，CREATE TRIGGER子句声明了触发器的名称recorder-trigger，AFTER UPDATE OF指定触发器在任何一次对关系inventory上的属性amount更新之后都会开始执行。一个

SQL更新语句可以更新关系中的多个元组， FOR EACH ROW子句可以显式地在每一个更新行上迭代。REFERENCING old(new) row AS子句建立了临时变量orow(nrow)用来存储更新行更新前后的值。WHEN语句指定一个条件，只有满足条件的元组，系统才会执行余下的触发器过程。触发器过程由BEGIN…END子句将多个执行过程集成为一个复合语句。这里，仅当商品数量从大于最小量降到最小量以下的时候才下达一个订单。如果只检查更新后的新数值是否小于最小量，就可能在商品已经被重订购的时候错误地下达一个订单。

触发器事件和动作可以有很多形式，列出如下：

(1) 触发器事件可以为UPDATE、INSERT和DELETE。

(2) 触发器事件可以在事件后被激发(AFTER UPDATE/INSERT/DELETE OF)，也可以在事件前被激发(BEFORE UPDATE/INSERT/DELETE OF)。

(3) 触发器的执行可以对插入、删除或者更新的行执行操作，即行触发器，用FOR EACH ROW子句定义；也可以对引起插入、删除或者更新的SQL语句执行动作，即语句触发器，用FOR EACH STATEMENT子句定义。

(4) 触发器的临时变量既可以是行，用REFERENCE old(new) ROW AS来容纳被影响的元组；也可以是表，用REFERENCE old(new) TABLE AS来容纳所有被影响的行。但是，无论临时表是语句触发器还是行触发器，都不能用BEFORE触发器，但是可以用AFTER触发器。

这样，在过渡表的基础上，一个单独的SQL语句就可以用来执行多个动作。由于每一个数据库都有它们各自的触发器语法，因而很多数据库系统提供非标准的触发器接口，或者只实现某些触发器特性。例如，很多数据库系统不支持BEFORE子句，用关键字ON来代替AFTER；也可能不支持REFERENCING子句，而用关键字INSERTED或DELETED来表示临时过渡变量。

7.4.2 触发器的应用

触发器的优点有很多，一般来讲，触发器可以应用在以下几个方面：第一，在数据写入关系表前，进行强制约束检验或数据转换，以维护数据完整性；第二，用于示警，如果触发程序发生错误，执行结果会被撤销；第三，用于满足特定条件时自动执行某项任务。

部分数据库管理系统可以针对数据定义语言使用触发程序，称为数据定义语言触发器(DDL Trigger)。然而随着数据库技术的发展，触发器的好多用途都可以由新技术替代，下面列出了可以取代触发器的其他技术：

(1) 维护概要数据：例如，在插入或删除一个员工的信息时，可以利用触发器维护每一个部门的薪金总额。然而，今天的很多数据库系统支持物化视图，这使维护概要数据的方法更简单，因而可以不用触发器。

(2) 复制数据库：例如，在每一个关系的插入/删除/更新的操作上使用触发器，跟踪其改变并将变更记录下来，由一个独立的进程将这些变更拷贝到数据库的副本，系统对副本执行修改。然而，现代的数据库系统提供内置的数据库复制工具，使得复制在大多数情况下不必使用触发器。

(3) 很多触发器的应用，可以由SQL 2003中介绍的“封装”特性取代：例7.10中的重新订购触发器，可以通过一个特殊的过程，利用封装来完成对商品的订购。这个过程会轮流检查商品数量，执行重新订购触发器的动作。

触发器的应用多种多样，但是也有一定的缺点。因为在运行期间，一个触发器错误会导致该触发器的插入、删除或更新语句失败，并且，一个触发器的动作可以引发另一个触发器。在最坏的情况下，这甚至会导致一个无限的触发链，形成死锁。因而，写触发器时应格外小心。通常，数据库系统限制这种触发器链的长度，把超过限定长度的触发器链看成是一个错误。

7.5 安全性与授权

数据库的安全性是指保护数据库以防止不合法使用所造成的数据泄露、更改和破坏。系统安全保护措施是否有效是数据库系统的主要性能指标之一。这一节分别从数据库的授权、视图、角色、权限与审计等方面介绍数据库系统的安全机制，有关SQL中的安全性机制在下一节介绍。

7.5.1 安全性概述

安全性问题不是数据库系统所独有的，计算机系统都需要面对这个问题。在数据库系统中，由于数据集中存放，并且为许多用户直接共享，因而需要特别的安全机制保护数据不受恶意访问和破坏。绝对杜绝对数据库的恶意滥用是不可能的，但可以使那些企图在没有适当授权情况下访问数据库的代价足够高，以阻止未经授权就对数据库中的数据进行读取、修改和破坏。

在一般的计算机系统中，安全措施是分层设置的，在不同层面采用不同的安全措施，进而达到保护数据库的目的。图7.2描述了计算机系统不同层次上的安全模型。

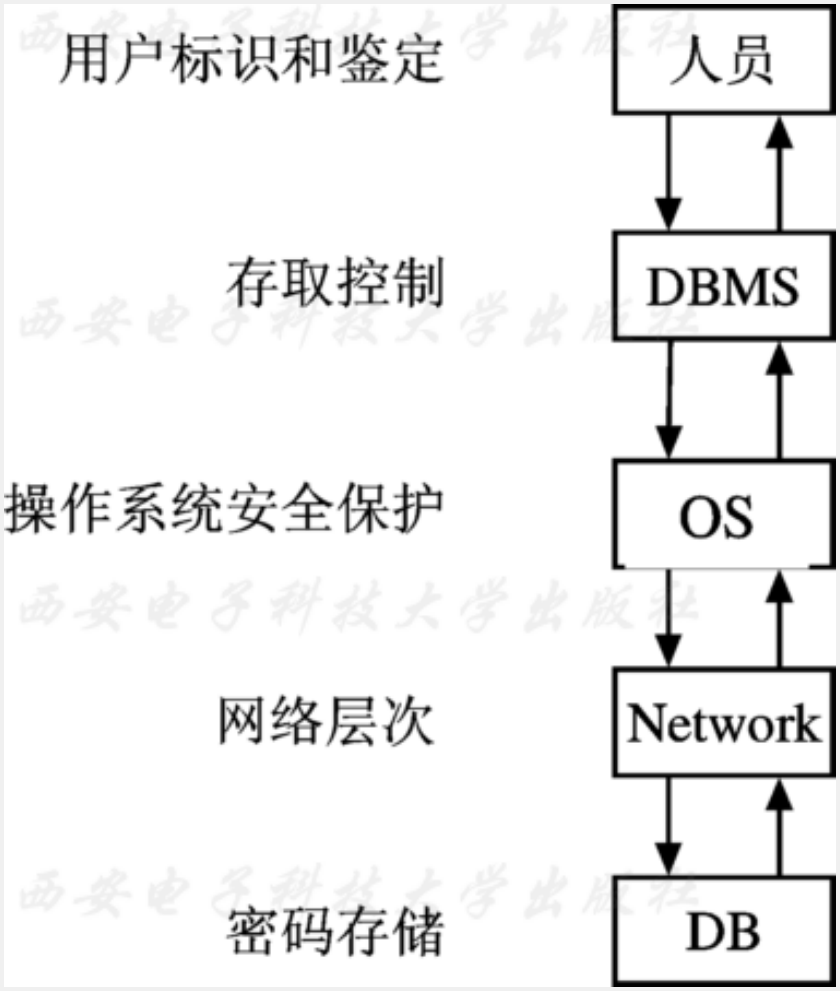

图7.2 计算机系统安全模型

(1) 数据库层次(也称物理层次)：物理层次的安全性主要针对强力逼迫透露口令、在通信线路上窃听、甚至盗窃物理存储设备等行为。因此数据以密码形式存储在数据库中，这样，即使窃密者以其他手段从数据库中获取了数据，也无法理解。

(2) 网络层次：由于几乎所有的数据库系统都允许通过终端或网络进行远程访问，网络软件的软件层安全性和物理安全性一样重要，不管在因特网上还是在私有的网络内。

(3) 操作系统层次：数据库系统是建立在操作系统之上的，操作系统的各种保护措施也可以对数据库起到保护作用。另一方面，操作系统在安全方面的漏洞也可能成为对数据库进行非法访问的一种手段。