第五部分　高 级 专 题

在本书第二部分我们曾说过，关系模型是现代数据库技术的基础。该部分所介绍的内容基本仅限于基础范畴。但是除了第二部分介绍的关系模式的内容以外，数据库技术还有很多其他的内容。对于学习数据库的学生和专业人员，要想充分掌握数据库这门学科，还有很多知识需要学习（从第三、四部分的讨论已明确表明了这一点）。下面我们将进一步讨论数据库领域的高级专题。这些内容包括：

- 安全性（第16章）
- 优化（第17章）
- 空缺信息（第18章）
- 类型继承（第19章）
- 分布数据库（第20章）
- 决策支持（第21章）
- 时态数据（第22章）
- 基于逻辑的数据库（第23章）

以上内容的次序并无定规，读者也可以有选择地阅读。

第16章　安　全　性

16.1　引言

数据安全性问题常与数据完整性问题混淆（至少在信息环境下），但是这两个概念实际上是不同的。安全性是指保护数据以防止不合法的使用造成数据泄漏、更改和破坏；完整性是指数据的准确性和有效性。通俗地讲：

- 安全性(security)保护数据以防止不合法用户故意造成的破坏；
- 完整性(integrity)保护数据以防止合法用户无意中造成的破坏；

或者简单地说就是：安全性确保用户被允许做其想做的事情；完整性确保用户所做的事情是正确的。

当然这两个概念也有相似点：系统应对用户不能违背的约束了如指掌；这些约束必须用合适的语言给出（通常由DBA），而且必须通过系统日志进行维护；DBMS必须监控用户的操作以确定约束发挥了效用。在本章中，我们专门讨论安全性问题（完整性已在第 8章以及其他章节进行了详细讨论）。

注：将这两个概念分两章讨论的主要原因在于：我们认为完整性是很基本的概念，而安全性是第二位的问题。

概述

安全性问题涉及许多方面，如：

- 法律、社会以及伦理问题（例如：请求者对其请求的信息是否具有合法权利？）；
- 物理控制（例如：计算机或终端所在房间是否上锁或受到保护？）；
- 政策问题（例如：拥有系统的企业如何控制使用者对数据的存取？）；
- 可操作性问题（例如：如果某个密码方案被采用，密码自身又如何保密？）；
- 硬件控制（例如：处理单元是否具备安全特性，如存储保护键或保护操作模式？）；
- 操作系统支持（例如：底层操作系统在退出时是否抹去了主存和磁盘上的内容？）；

最后还包括

- 数据库系统需专门考虑的问题（例如：数据库系统是否具有数据所有权的概念？）。

在本章中将只就上述的最后一类问题展开讨论（绝大部分）。

现代的DBMS通常采用自主存取控制和强制存取控制这两种方法来解决安全性问题，有的只提供其中的一种方法，有的两种都提供。无论采用哪种存取控制方法，需要保护的数据单元或数据对象包括从整个数据库到某个元组的某个部分。两者的区别为：

- 在自主存取控制方法(discretionary control)中，用户对不同的数据对象具有不同的存取权限(也称为优先权)，而且没有固有的关于哪些用户对哪些数据对象具有哪些存取权限的限制（例如：用户*U1*能看到*A*但看不到*B*，而用户*U2*能看到*B*但看不到*A*）。因此自主存取控制非常灵活。
- 在强制存取控制方法 (mandatory control)中，每一个数据对象被标以一定的密级(classification level)，每一个用户也被授予一个许可证级别(clearance level)。对于任意一个对象，只有具有合法许可证的用户才可以存取。因此，强制存取控制本质上具有分层特点，且相对比较严格（例如：如果用户*U1*能看到*A*但看不到*B*，这说明*B*的密级高于*A*，因此不存在用户*U2*能看到*B*但看不到*A*）。

我们将在16.2节讨论自主存取控制，在16.3节讨论强制存取控制。

不管我们采用的是自主存取控制方法还是强制存取控制方法，所有有关哪些用户可对哪些数据对象进行操作的决定都是政策决定，这显然超出了 DBMS的权限，DBMS只是实施这些决定。这就要求：

- 政策决定的结果（a）需为系统所知（这可通过某一特定的定义语言的语句来实现）；（b）需被系统记住（这可通过将它们保存在目录中实现）。
- 必须采用一定的方式检查存取请求是否违背目录中适用的安全性约束（通常用“存取请求”表示涉及的请求操作、请求对象以及请求用户），这主要通过DBMS的安全性子系统(security subsystem)或者授权子系统（authorization subsystem）实现。
- 为了决定哪些安全性约束适用于给定的存取请求，系统必须能识别请求的来源，也就是请求用户。因此，当用户登录系统时必须给出用户 ID以及口令(password)。口令只能为系统所知，并用以保证用户ID的合法性[⊖]。

关于最后一点，注意一定数量的不同用户可能共享同一 ID。这种情况下系统支持用户组

⊖ 检查用户是否就是其声称的用户称作授权。现在有比简单地检查密码复杂得多的授权技术，包括各种生物测定设置，如指纹阅读器、视网膜扫描器、手形图象器、声音确认器、签名识别器，等等。这些设备可有效地用于检查“别人无法窃取的个人特征”[16.3]。

(user group)，并允许组内每一用户共享对同一数据对象的相同存取权限。将用户加入用户组以及将用户从用户组删除的操作应与指定用户组对哪些数据对象拥有哪些权限的操作分离。但是注意记录用户组内有哪些用户的最佳位置还是目录（或为数据库自身）。参考文献[16.9]描述了一个用户组嵌套的系统，并提到“将用户分成有层次的用户组，提供了强有力的管理有着大量用户以及数据对象的大型系统的工具”。

16.2 自主存取控制

上一节已提到，大多数DBMS采用自主存取控制和强制存取控制这两种方法中的任意一种，或两者都采用。准确地说，应该是大多数系统支持自主存取控制，有些系统同时支持强制存取控制；实际系统中支持最多的是自主存取控制，因此我们首先对其进行讨论。

前面已经提到，需要支持（自主）安全性约束定义的语言，显然，说明允许的内容要比禁止的内容容易，因此定义语言通常支持的不是安全性约束的定义，而是有关授权(authority)的定义（授权实际上就是约束的对立面，即一旦被授权，就不允许施以约束）。我们首先介绍一种定义授权的语言[⊖]。下面是一简单的例子：

```
AUTHORITY SA3
   GRANT RETRIEVE (S#,SNAME,CITY),DELETE
   ON     S
   TO     Jim,Fred,Mary;
```

该例子说明了授权包括四个重要的部分：

1) 名字（例子中为SA3），授权将以该名字登记到目录；
2) 一种或多种权限（例子中为RETRIEVE —— 仅在某些属性上，以及DELETE），通过GRANT子句指定；
3) 关系变量（例子中为S）是授权施以的数据对象，由ON子句指定；
4) 一个或多个用户（准确地说应该是用户ID）对指定的关系变量拥有指定的权限，由TO子句指定。

该语句的语法如下：

```
AUTHORITY <authority name>
   GRANT <privilege commalist>
   ON    <relvar name>
   TO    <user ID commalist>;
```

解释：*<authority name>*、*<relvar name>*和*<user ID commalist>*不言自明，ALL作为合法“用户ID”，意味着所有的用户。*<privilege>*如下：

```
RETRIEVE [(<attribute name commalist>)]
INSERT   [(<attribute name commalist>)]
UPDATE   [(<attribute name commalist>)]
DELETE
ALL
```

RETRIEVE（无限制）、INSERT（无限制）、UPDATE（无限制）以及DELETE无须加以

⊖ Tutorial D不包括任何授权定义措施，本节假定的语言可看作反映了Tutorial D的精华。

说明 ㊀。如果RETRIEVE有属性名列表，则说明 RETRIEVE权限只适用于指定的属性，INSERT、UPDATE也具有类似的含义。 ALL意味着所有权限： RETRIEVE（所有属性）、INSERT（所有属性）、UPDATE（所有属性）以及DELETE。注意：为简便起见，在此我们没有讨论关于是否需要特权以执行一般的关系分配操作，另外我们讨论的范围也局限在数据操纵（data manipulation）操作上。实际上，合法权检查机制还要检查许多其他的操作，如定义与删除关系变量的操作，定义与删除授权自身的操作，等等。限于篇幅，我们忽略了对这些操作做细致的讨论。

如果用户尝试对某一数据对象进行某种操作，但不具有访问权限，将会出现怎样的情形呢？最简单的反应显然是拒绝该尝试（当然还应给出合适的诊断信息），这是实际系统中最基本的要求，所以我们不妨将之做为缺省处理。在更多的敏感情形下，其他的一些反应可能更合适，例如中止程序的运行或锁住用户的键盘。将这些尝试记录在特殊的日志（威胁监控(threat monitoring)）中或许更可取，这些信息可用以对破坏系统安全性的尝试进行分析，而且自身能抵制不合法的渗透（可参考本节最后关于审计追踪的讨论）。

当然，我们也需要一种方式来删除授权：

```
DROP AUTHORITY <authority name>;
```

例如：

```
DROP AUTHORITY SA3;
```

为简化起见，我们假设删除关系变量时将自动删除施用于该关系变量上的所有授权信息。

下面是一些关于授权的例子，大部分不言自明：

```
1) AUTHORITY EX1
     GRANT RETRIEVE(P#,PNAME,WEIGHT)
     ON    P
     TO    Jacques,Anne,Charley;
```

用户Jacques、Anne和Charley所看到的是基关系变量P的“垂直子集”，该例是值无关授权的例子。

```
2) AUTHORITY EX2
     GRANT RETRIEVE,UPDATE(SNAME,STATUS),DELETE
     ON    LS
     TO    Dan, Misha;
```

LS是一视图（见第9章的图9-4——“London供应商”）。用户Dan和Misha看到的是关系变量S的“水平子集”，该例是值依赖授权的例子。注意：虽然Dan和Misha能删除S的部分元组（通过视图LS），但不能INSERT元组，也不能对属性S#或CITY进行UPDATE。

```
3) VAR SSPPR VIEW
     (S JOIN SP JOIN (P WHERE CITY='Rome') {P#})
                                {ALL BUT P#,QTY};
   AUTHORITY EX3
     GRANT RETRIEVE
     ON    SSPPR
     TO    Giovanni;
```

㊀ 即使是提及相关对象如视图定义或完整性约束也需要RETRIEVE权根。

该例也是值依赖授权的例子：用户 Giovanni可以查询供应商的信息，但只能是供应存储在Rome的零件的供应商。

```
4) VAR SSQ VIEW
       SUMMARIZE SP PER S{S#} ADD SUM{QTY} AS SQ
   AUTHORITY EX4
      GRANT RETRIEVE
      ON    SSQ
      TO    Fidel;
```

用户Fidel可查看每个供应商的总发货量，但不能查看到每次发货量，用户只能查看到基于底层基本数据的统计汇总信息。

```
5) AUTHORITY EX5
      GRANT RETRIEVE,UPDATE (STATUS)
      ON    S
      WHEN  DAY() IN ('Mon','Tue','Wed','Thr','Fri')
            AND NOW()≥ TIME '09:00:00'
            AND NOW()≤ TIME '17:00:00'
      TO    Purchasing ;
```

这里我们对AUTHORITY的语法作了扩展，增加了 WHEN子句用以说明特定的“环境控制”；我们也假设系统支持两个无操作数的操作（niladic operator）：DAY（）和NOW（）。授权EX5保证了供应商的状态值只能由用户“ Purchasing”（假定是purchasing部门的任意一个职员）在工作日的工作时间更改。这通常称为环境依赖（context-dependent）授权，因为存取请求是否被准许依赖于环境。

其它系统应该支持且对环境依赖授权有用的内置操作的例子有：

```
TODAY()      —— 值为当前日期
USER()       —— 值为当前用户的ID
TERMINAL()   —— 值为当前请求的源终端的ID
```

现在你可能已意识到，所有的授权是“或”在一起使用的。换句话说，如果至少存在一个授权准许给定的存取请求，该请求就是可接受的。但是注意：举个例子来说，如果（ a）一个授权允许用户 Nancy检索零件的颜色；（b）另一授权允许她检索零件的重量；这并不意味着她能同时检索零件的颜色和重量（这种组合需要特别授权）。

最后提一点，用户只能做定义的授权明确允许的事情，任何未明确授权的事情都隐含是不合法的。

1. 修改请求

为了解释上述的概念，我们简单介绍 Ingres原型系统的安全机制以及查询语言 QUEL，因为其采用了一种巧妙的方法。基本原理就是系统对给定的 QUEL请求在其执行前进行自动地修改，使其不违背安全性约束。举个例子：假定用户 *U*只允许检索存储在London的零件：

```
DEFINE PERMIT RETRIEVE ON P TO U
       WHERE       P.CITY = "London"
```

假设用户 *U*发出如下的QUEL请求：

```
RETRIEVE(P.P#,P.WEIGHT)
WHERE     P.COLOR = "Red"
```

利用存储在目录中的允许对关系变量 P和用户 *U*存取的条件，系统将用户的请求自动修改

如下：

```
RETRIEVE(P.P#,P.WEIGHT)
WHERE    P.COLOR = "Red"
AND      P.CITY = "London"
```

显然，这种修改不可能违背安全性约束。注意，用户 *U*并不知道系统执行的语句与其请求并不完全相同，因为数据本身就是敏感的，用户 *U*无权知道非London地区的零件。

上述修改请求的过程实际上与视图实现 [8.20]以及完整性约束（尤其在 Ingres原型）的技术是一致的，因此该方案的优点在于实现简单 —— 许多必要的代码已存在于原来的系统；效率高 —— 安全措施的实施在编译时，而不是在运行时，至少部分如此；能满足某用户对同一关系变量的不同部分具有不同的存取权限的需求。

该方案的缺点在于不能处理所有安全性约束。举个简单的例子，假设用户 *U*根本就不具有存取关系变量P的权限，根据上述修改过程，不存在 RETRIEVE的修改形式能说明关系变量 P存在，因此应给出错误消息“ 你无权存取该关系变量 ”或“ 不存在该关系变量 ”。

下面是DEFINE PERMIT的语法：

```
DEFINE PERMIT <operation name commalist>
      ON    <relvar name>[(<attribute name commalist>)]
      TO    <user ID>
    [ AT    <terminal name commalist>]
    [ FROM  <time> TO <time>]
    [ ON    <day>  TO <day> ]
    [ WHERE <boolean expression>]
```

该语句与AUTHORITY非常类似，除了增加了对WHERE子句的支持。参考下面的例子：

```
DEFINE PERMIT APPEND,RETRIEVE,REPLACE
       ON   S(S#,CITY)
       TO   Joe
       AT   TTA4
       FROM 9:00 TO 17:00
       ON   Sat TO Sun
       WHERE  S.STATUS < 50
       AND  S.S# = SP.P#
       AND  SP.P# = P.P#
       AND  P.COLOR = "Red"
```

注意：QUEL中的APPEND和REPLACE分别类似于我们的INSERT和UPDATE。

2. 审计追踪

永远不要认为安全性系统是坚不可摧的，潜在的渗透者总能想方设法突破控制，尤其是当其因此能获得很高的利益时。如果数据具有很高的敏感性，数据处理过程很重要，审计追踪就非常必要了。如果你怀疑数据库中的数据遭到篡改，就可以通过审计追踪检查用户对数据库究竟执行了哪些操作，并可确认操作都受到了控制，或可帮助确定误操作的人员。

审计追踪本质上是一特殊的文件或数据库，系统可自动记录用户对数据执行的所有操作。在有些系统中，审计追踪物理上与恢复日志合二为一（参看第 14章），而有的系统中两者分别存放；无论哪种方式，用户都应该能利用规则的查询语言解释审计追踪（当然用户必须具有权限）。通常的审计追踪包含如下的信息：

- 请求（源文本）

- 发出操作调用的终端
- 发出操作调用的用户
- 操作日期和时间
- 操作作用的关系变量，元组，属性
- 旧值
- 新值

正如本节前面所提到的，审计追踪维护的信息必须足够充分以防止某些情形下的不怀好意的渗透者。

16.3 强制存取控制

军方和政府的数据具有很高的敏感性，通常具有静态的严格的分层结构（ classification structure），强制存取控制对于存放这样的数据的数据库非常适用。该方法的基本思想在于每个数据对象具有一定的密级（classification level），如：绝密、机密、秘密，等等；每个用户具有一定的许可证级别（clearence level）。密级和许可证级别都是严格有序的，如：绝密 >机密>秘密。Bell和La Padula[16.1]采用如下的简单规则：

1) 用户i可以查询对象j，当且仅当i的许可证级别大于或等于j的密级（简单规则）；

2) 用户i可以更新对象j，当且仅当i的许可证级别等于j的密级（星规则）。

规则1的意义是明显的，而规则 2需要一点说明。规则 2的另一种表述为用户 i所写的对象自动获得的密级等于其许可证级别，该规则是为了防止具有较高级别的用户将该级别的数据复制到较低级别的文件中。注意：如果只是纯粹的“写”（INSERT）操作，写规则可以弱化为：当且仅当i的许可证级别小于或等于 j的密级，用户 i可以“写”对象j，但是用户可能无权读所写的对象，因此，这样的弱化不很现实。

20世纪90年代早期强制存取控制引起了数据库领域的注意，因为美国国防部要求其所购买的所有系统都必须支持这样的控制，这就促使各大 DBMS厂商竞相提供这样的支持。美国国防部颁布的“桔皮书”[16.19]和“紫皮书”[16.20]对强制存取控制作了全面的描述和定义，“桔皮书”定义了任意“可信计算基”（Trusted Computing Base，简称TCB）应当遵从的一系列安全性要求；而“紫皮书”则定义了这些要求在数据库系统中相应的解释。

上述两份文献给出了通用的安全性分级模式，共定义了四类安全级别：D、C、B和A，由D类到A类级别依次增高。D类提供最小（minimal）保护，C类提供自主（discretionary）保护，B类提供强制（mandatory）保护，A类提供验证（verified）保护。

- 自主保护：C类分为两个子类C1和C2，C1安全级别低于C2。每个子类都支持自主存取控制，即存取权限由数据对象的所有者决定。
  1) C1子类对所有权与存取权限加以区分，虽然它允许用户拥有自己的私有数据，但仍然支持共享数据的概念。
  2) C2子类还要求通过注册、审计以及资源隔离以支持责任说明（ accountability）。
- 强制保护：B类分为三个子类B1、B2和B3，B1安全级别最低，B3最高。
  1) B1子类要求“标识化安全保护”，即要求每个数据对象都必须标以一定的密级，同时还要求安全策略的非形式化说明。
  2) B2子类要求安全策略的形式化（ formal）说明，能识别并消除隐蔽通道（ covert

channel)。隐蔽通道的例子有（a）从合法查询的结果中推断出不合法的查询的结果；（b）通过合法的计算推断出敏感信息。

3) B3子类要求审计和恢复支持以及指定的安全管理者。

- 验证保护：A类要求对安全机制是可靠的且足够支持指定的安全策略给出严格的数学证明。

有些DBMS产品现在提供B1级强制存取控制以及C2级自主存取控制。支持强制存取控制的DBMS也称为多级安全系统（multi-level secure system）[16.13,16.16,16.21]或可信系统（trusted system）[16.17,16.19~16.20]。

多级安全

假设要对关系变量S进行强制存取控制，为简化起见，假设要控制存取的数据单元是元组，则每个元组需标以密级，如图16-1所示（4=绝密，3=机密，2=秘密）。

S

S#	SNAME	STATUS	CITY	CLASS
S1	Smith	20	London	2
S2	Jones	10	Paris	3
S3	Blake	30	Paris	2
S4	Clark	20	London	4
S5	Adams	30	Athens	3

图16-1 具有密级的关系变量S（示例）

假设用户*U1*和*U2*的许可证级别分别为3和2，那么*U1*和*U2*看到的S是不一样的。若请求查询所有的供应商，*U1*能查得四个元组：S1、S2、S3和S5；*U2*只查得两个元组：S1和S3，*U1*和*U2*都无法查得元组S4。

可从修改请求（request modification）的角度考虑上述问题。考虑这样的查询“查找London的供应商”：

```
S WHERE CITY = 'London'
```

系统将请求修改为

```
S WHERE CITY = 'London' AND CLASS ≤ user clearance
```

更新操作类似，举个例子，用户 *U1*不知道元组S4存在，因此，对*U1*来说，下面的INSERT语句是合理的：

```
INSERT INTO S RELATION {TUPLE   {S# S# ('S4'),
                                SNAME  NAME('Baker'),
                                STATUS 25,
                                CITY   'Rome'}};
```

系统必须拒绝该INSERT请求，但这样做实际上使用户意识到S4在数据库中是存在的。因此系统接受该请求，但将之修改为：

```
INSERT INTO S RELATION {TUPLE{     S#   S#('S4'),
                                   SNAME  NAME('Baker'),
                                   STATUS 25,
                                   CITY   'Rome',
                                   CLASS CLASS(3)}};
```

这样，供应商的主码实际上不是{S#}，而是{S#，CLASS}。注意：这里假设只有一个候选码，因此可将之作为主码。

供应商关系变量可看作多级变量（multi-level relvar），“相同”数据对不同用户实际并不相同被称作多重实例（polyinstantiation），如查询供应商S4的请求将对具有绝密、机密以及秘密等不同安全级别的用户返回不同的结果。

UPDATE 和DELETE也是类似处理方式。问一个问题，你认为上面的处理方式是否违背了信息原则？

16.4 统计数据库

注意：本节和下节的大部分资料与参考文献[16.4]略有不同。

统计数据库允许用户查询聚集类型的信息（例如合计、平均值等），但是不允许查询单个记录信息。例如，查询“程序员的平均工资是多少？”是合法的，但是查询“程序员 Adams 的工资是多少？”就不允许。

在统计数据库中存在着特殊的安全性问题，即可能存在着隐蔽的信息通道，使得可以从合法的查询中推导出不合法的信息。正如参考文献 [16.6]指出的：“综合信息总是带有少量的原始信息，利用足够的综合信息就可能获得原始信息，这也称作机密信息的推断。”随着数据仓库应用得越来越广泛，这个问题也变得越来越严重。

假定数据库中只有一个关系变量 STATS（如图16-2所示），所有的属性都简单地定义在基本数据类型（数字和字符串）上；用户 *U*只具有执行统计信息查询的权限，并且打算查出 Alf 的工资；*U*知道Alf是程序员且为男性。

NAME	SEX	CHILDREN	OCCUPATION	SALARY	TAX	AUDITS
Alf	M	3	Programmer	50K	10K	3
Bea	F	2	Physician	130K	10K	0
Cary	F	0	Programmer	56K	18K	1
Dawn	F	2	Builder	60K	12K	1
Ed	M	2	Clerk	44K	4K	0
Fay	F	1	Artist	30K	0K	0
Guy	M	0	Lawyer	190K	0K	0
Hal	M	3	Homemaker	44K	2K	0
Ivy	F	4	Programmer	64K	10K	1
Joy	F	1	Programmer	60K	20K	1

图16-2 关系变量 STATS（样本值）

考虑查询1和查询2[⊖]：

```
1) WITH (STATS WHERE SEX = 'M' AND
         OCCUPATION = 'Programmer') AS X :
   COUNT (X)
```

结果：1。

```
2) WITH (STATS WHERE SEX = 'M' AND
         OCCUPATION = 'Programmer') AS X :
   SUM (X, SALARY)
```

结果：50K。

显然，尽管用户 *U*的统计信息查询都是合法的，数据库的安全仍受到了危害。从上例可

⊖ 本节的查询都以 Tutorial D 的简化格式表示。举个例子，查询 1中的表达式 COUNT (X) 用EXTEND TABLE_DEE ADD COUNT(X) AS RESULT表示更合适。

看出，如果用户能找到一个布尔表达式识别出某个个体，有关该个体的信息将不再安全。这表明当综合查询结果集的基数低于某个下限 b 时系统应拒绝响应。同样，当综合查询结果集的基数高于上限 $N-b$ 时系统也应拒绝响应（N 是关系的基数），从下面的查询 3~6 可看出，当基数很高时数据库的安全同样会受到危害。

```
3) COUNT (STATS)
```

结果：12。

```
4) WITH (STATS WHERE NOT (SEX = 'M' AND
        OCCUPATION = 'Programmer')) AS X:
   COUNT (X)
```

结果：11；12－11=1。

```
5) SUM (STATS, SALARY)
```

结果：728K。

```
6) WITH (STATS WHERE NOT (SEX = 'M' AND
        OCCUPATION = 'Programmer')) AS X:
   SUM (X, SALARY)
```

结果：678K；728K－678K=50K。

不幸的是，简单地将综合查询结果集的基数 c 限定在 $b \leq c \leq N-b$，并不足以避免数据库的安全受到危害。考虑图 16-2，假定 $b = 2$，当且仅当 $2 \leq c \leq 8$ 时查询请求才被响应，布尔表达式

```
SEX = 'M' AND OCCUPATION = 'Programmer'
```

不再合法。但是考虑查询 7~10：

```
7) WITH (STATS WHERE SEX = 'M') AS X:
   COUNT (X)
```

结果：4。

```
8) WITH (STATS WHERE SEX = 'M' AND NOT
                (OCCUPATION = 'Programmer')) AS X:
   COUNT (X)
```

结果：3。

从查询 7 和 8，用户 U 可推断出只存在一个男性程序员，其必定是 Alf，由此可通过下面的语句查出 Alf 的工资。

```
9) WITH (STATS WHERE SEX = 'M') AS X:
   SUM (X, SALARY)
```

结果：328K。

```
10) WITH (STATS WHERE SEX = 'M' AND NOT
                 (OCCUPATION = 'Programmer')):
    SUM (X, SALARY)
```

结果：278K；328K-278K=50K。

布尔表达式 SEX = 'M' AND NOT OCCUPATION = 'Programmer' 称作 Alf 的个体追踪者（individual tracker）[16.6]，这是因为通过它用户才追踪获得个体信息。一般地讲，如果用户知道识别某个体 I 的某布尔表达式 BE，并且 BE 可表示为 $BE1$ AND $BE2$ 这样的形式，那么 $BE1$

AND NOT *BE2*就是*I*的个体追踪者（假设*BE1*和*BE1* AND NOT *BE2*都是合法的，即两者的结果集基数都在上述的限定范围内）。原因在于：

```
set (BE) = set (BE1 AND BE2)
         = set (BE1) minus set(BE1 AND NOT BE2)
```

如图16-3所示。

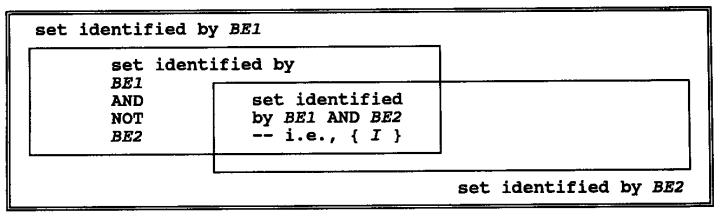

图16-3 个体追踪者BE1 AND NOT BE2

参考文献[16.6]概括了前面的内容，并指出对几乎任意一个统计数据库总能找到一个通用追踪者（general tracker），这是相对于个体追踪者而言的。通用追踪者是这样的布尔表达式，它能用于获得任意不合法查询（包含不合法表达式的查询）的结果，而个体追踪者只对某些特殊的不合法查询有效。实际上，任何结果集基数 c满足$2b \leq c \leq N-2b$的表达式都是通用追踪者，这里b必须小于$N/4$，通常适用于各种实际情况。一旦找到这样的追踪者，相应于不合法表达式*BE*的查询就可迎刃而解，下面的例子将对此进行解释。该例中假设 *BE*的结果集基数小于b，类似地，当*BE*的结果集基数大于$N-b$时也是如此处理。注意，从定义可看出，若 *T*是一通用追踪者，则NOT *T*也是通用追踪者。

例：仍假设$b=2$；则通用追踪者是任何结果集基数 c满足$4 \leq c \leq 6$的表达式。仍假设用户 *U*从外部资料获悉Alf是一个男性程序员，即不合法布尔表达式是（与前面一样）：

```
SEX = 'M' AND OCCUPATION = 'Programmer'
```

并且假设*U*试图查出Alf的工资。对通用追踪者使用两次，首先确定 *BE*实际上唯一标识Alf（步骤2~4），再确定Alf的工资（步骤5~7）。

步骤1：猜测一追踪者*T*。这里选择的*T*为表达式

```
AUDITS = 0
```

步骤2：利用表达式*T*以及NOT *T*获得数据库中个体的总数。

```
WITH (STATS WHERE AUDITS = 0) AS X:
COUNT (X)
```

结果：5。

```
WITH (STATS WHERE NOT(AUDITS = 0)) AS X:
COUNT (X)
```

结果：5；5+5=10。

显然，我们所猜测的*T*就是一通用追踪者。

步骤3：利用表达式*BE* OR *T*以及*BE* OR NOT *T*获得数据库中个体总数与满足不合法表达式*BE*的个体数目的总和。

```
WITH (STATS WHERE (SEX = 'M' AND
                   OCCUPATION = 'Programmer')
```

```
            OR AUDITS = 0) AS X:
COUNT (X)
```

结果：6。

```
WITH (STATS WHERE (SEX = 'M' AND
                   OCCUPATION = 'Programmer'
                OR NOT (AUDITS = 0)) AS X:
COUNT (X)
```

结果：5；6+5=11。

步骤4：从上面的结果可算出满足*BE*的个体数目是1（步骤3的结果减去步骤2的结果），显然，*BE*唯一标识了Alf。

在步骤5和6中重复步骤2和3的查询，但使用SUM取代COUNT。

步骤5：利用表达式*T*以及NOT *T*获得数据库中个体的工资总额。

```
WITH (STATS WHERE AUDITS > 0) AS X:
SUM (X, SALARY)
```

结果：438K。

```
WITH (STATS WHERE NOT (AUDITS = 0)) AS X:
SUM (X, SALARY)
```

结果：290K；438K+290K=728K。

步骤6：利用表达式*BE* OR *T*以及*BE* OR NOT *T*获得Alf的工资与工资总额的总和。

```
WITH (STATS WHERE (SEX = 'M' AND
                   OCCUPATION = 'Programmer'
             OR AUDITS = 0) AS X:
SUM (X, SALARY)
```

结果：488K。

```
WITH (STATS WHERE (SEX = 'M' AND
                   OCCUPATION = 'Programmer'
             OR NOT (AUDITS = 0)) AS X:
SUM (X, SALARY)
```

结果：290K；488K+290K=778K。

步骤7：从步骤6的结果中减去工资总额（步骤5的结果）即得Alf的工资。

结果：50K。

图16-4解释了通用追踪者：

```
set (BE) = (set (BE OR T) plus set(BE OR NOT T))
                          minus set(T OR NOT T)
```

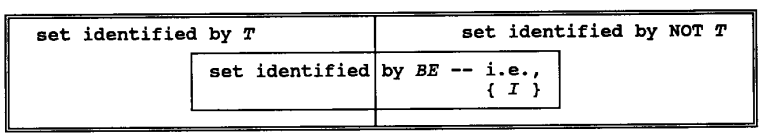

图16-4 通用追踪者T

如果最初的猜测是错的，也就是说，*T*实际上并不是一个通用追踪者，则表达式（*BE* OR *T*）与(*BE* OR NOT *T*)中至少有一个可能是不合法的。举个例子，如果*BE*和*T*的结果集基数分

别为p和q，满足$p < b$，$b \leq q \leq 2b$，则(BE OR T)的结果集基数（不可能超过$p+q$）小于$2b$（译者认为该例子不恰当，修改如下：如果BE和T的结果集基数分别为p和q，满足$p < b$，$b \leq q \leq 2b$，则(BE OR NOT T)的结果集基数（不可能超过$p+N-q$）大于$N-2b$而小于N；若大于$N-b$而小于N，则(BE OR NOT T)不合法。因此必须再猜测通用追踪者并检测。参考文献 [16.6]表明找到通用追踪者的过程并不困难。本例中，最初的猜测是正确的，T是一个通用追踪者（因为其结果集基数为5），因此步骤3的查询都是合法的。

总结：通用追踪者“几乎总是”存在的，而且通常容易找到并易于使用。事实上，通过猜测迅速找到追踪者是可能的 [16.6]。即使在通用追踪者不存在的情况下，[16.6]指出，对于特殊的查询通常也能找到特殊的追踪者。因此，在统计数据库中安全性确实是个问题。

已经有了一些解决办法，但没有一个能完全令人满意的。举个例子，一种方法是“数据交换”，即在元组间交换属性值，但不破坏整体的统计精确性。这样即使某个特定的值（如工资）被识别出来，仍然没有办法知道该值属于哪个个体。采用这种办法的困难在于确定能被交换的值的集合。其他方法采用了类似的限制形式。由此看来我们应同意 Denning的结论 [16.6]：“折衷的办法简单易行。对机密信息绝对保密的要求与对总体的任意子集采用精确的统计措施的要求是不可能同时满足的，要保证秘密可行至少要降低其中的一个要求。”

16.5 数据加密

前面的讨论都是认为恶意的攻击者将使用通常的系统设施存取数据库，现在考虑用户可能试图旁路系统的情况，如物理地取走数据库，在通讯线路上窃听。对这样的威胁最有效的解决方法就是数据加密，即以加密格式存储和传输敏感数据。

数据加密的术语有：明文，即原始的或未加密的数据。通过加密算法对其进行加密，加密算法的输入信息为明文和密钥；密文，明文加密后的格式，是加密算法的输出信息。加密算法是公开的，而密钥则是不公开的。密文，不应为无密钥的用户理解，用于数据的存储以及传输。

例：明文为字符串：

```
AS KINGFISHERS CATCH FIRE
```

（为简便起见，假定所处理的数据字符仅为大写字母和空格符）。假定密钥为字符串：

```
ELIOT
```

加密算法为：

1) 将明文划分成多个密钥字符串长度大小的块（空格符以“+”表示）

```
AS+KI NGFIS HERS+ CATCH +FIRE
```

2) 用00~26范围的整数取代明文的每个字符，空格符=00，A=01，...，Z=26：

```
0119001109 1407060919 0805181900 0301200308 0006091805
```

3) 与步骤2一样对密钥的每个字符进行取代：

```
0512091520
```

4) 对明文的每个块，将其每个字符用对应的整数编码与密钥中相应位置的字符的整数编码的和模27后的值取代：

```
0119001109  1407060919  0805181900  0301200308  0006091805
0512091520  0512091520  0512091520  0512091520  0512091520
```

0604092602 1919152412 1317000720 0813021801 0518180625

5) 将步骤4的结果中的整数编码再用其等价字符替换：

FDIZB SSOXL MQ+GT HMBRA ERRFY

如果给出密钥，该例的解密过程很简单（练习：对上面的密文进行解密）。问题是对于一个恶意攻击者来说，在不知道密钥的情况下，利用相匹配的明文和密文获得密钥究竟有多困难？对于上面的简单例子，答案是相当容易的，不是一般的容易，但是，复杂的加密模式同样很容易设计出。理想的情况是采用的加密模式使得攻击者为了破解所付出的代价应远远超过其所获得的利益。实际上，该目的适用于所有的安全性措施。这种加密模式的可接受的最终目标是：即使是该模式的发明者也无法通过相匹配的明文和密文获得密钥，从而也无法破解密文。

1. 数据加密标准

传统加密方法有两种，替换和置换。上面的例子采用的就是替换的方法：使用密钥将明文中的每一个字符转换为密文中的一个字符。而置换仅将明文的字符按不同的顺序重新排列。单独使用这两种方法的任意一种都是不够安全的，但是将这两种方法结合起来就能提供相当高的安全程度。数据加密标准（Data Encryption Standard，简称DES）就采用了这种结合算法，它由IBM制定，并在1977年成为美国官方加密标准[16.18]。

DES的工作原理为：将明文分割成许多64位大小的块，每个块用64位密钥进行加密，实际上，密钥由56位数据位和8位奇偶校验位组成，因此只有25^6个可能的密码而不是26^4个。每块先用初始置换方法进行加密，再连续进行16次复杂的替换，最后再对其施用初始置换的逆。第i步的替换并不是直接利用原始的密钥K，而是由K与i计算出的密钥Ki，详细算法请参看[16.18]。

DES具有这样的特性，其解密算法与加密算法相同，除了密钥 Ki的施加顺序相反以外。

2. 公开密钥加密

多年来，许多人都认为 DES并不是真的很安全。事实上，即使不采用智能的方法，随着快速、高度并行的处理器的出现，强制破解 DES也是可能的。“公开密钥”加密方法使得 DES以及类似的传统加密技术过时了。公开密钥加密方法中，加密算法和加密密钥都是公开的，任何人都可将明文转换成密文。但是相应的解密密钥是保密的（公开密钥方法包括两个密钥，分别用于加密和解密），而且无法从加密密钥推导出，因此，即使是加密者若未被授权也无法执行相应的解密。

公开密钥加密思想最初是由 Diffie和Hellman[16.7]提出的，最著名的是 Rivest、Shamir以及Adleman[16.15]提出的，现在通常称为RSA（以三个发明者的首位字母命名）的方法，该方法基于下面的两个事实：

1) 已有确定一个数是不是质数的快速算法；

2) 尚未找到确定一个合数的质因子的快速算法。

参考文献[16.10]给出了一个例子：确定一个 130位的数是否质数只花了大约 7分钟；而确定两个63位质数的乘积的两个质因子（在同一台机器）需要 40 000 000 000 000 000年 ㊀ 。

㊀ 即使如此，RSA方法仍有安全性问题。参考文献[16.10]在1977年就出现了。1990年，Arjen Lenstra和Mark Manasse成功地分解了155位的整数[16.22]，他们估计为此通过分布在各地的1000台计算机协同工作所涉及的计算量等同于在单台计算机上以每秒 100万条指令的速度执行需273年。这个155位的整数是第9个费马数$2^{512}+1$(注意，$512=2^9$)。也可看[16.12]，其介绍了另一个完全不同的、成功地破解了 RSA加密算法的方法。

RSA方法的工作原理如下：

1) 任意选取两个不同的大质数p和q，计算乘积$r=p*q$；

2) 任意选取一个大整数e，e与$(p-1)*(q-1)$互质，整数e用做加密密钥。注意：e的选取是很容易的，例如，所有大于p和q的质数都可用。

3) 确定解密密钥d：

$$d * e = 1 \text{ modulo}(p - 1) * (q - 1)$$

根据e、p和q可以容易地计算出d，算法可参看[16.15]。

4) 公开整数r和e，但是不公开d；

5) 将明文P (假设P是一个小于r的整数)加密为密文C，计算方法为：

$$C = Pe \text{ modulo } r$$

6) 将密文C解密为明文P，计算方法为：

$$P = Cd \text{ modulo } r$$

参考文献[16.15]对该方法的工作原理进行了证明，即用d确实可将密文C复原为明文P。然而只根据r和e（不是p和q）要计算出d是不可能的。因此，任何人都可对明文进行加密，但只有授权用户（知道d）才可对密文解密。

下面举一简单的例子对上述过程进行说明，显然我们只能选取很小的数字。

例：选取p=3, q=5，则r=15，（p-1）*（q-1）=8。选取e=11（大于p和q的质数），通过d * 11 = 1 modulo 8，计算出d =3。

假定明文为整数13。则密文C为

$$\begin{aligned} C &= Pe \text{ modulo } r \\ &= 1311 \text{ modulo } 15 \\ &= 1,792,160,394,037 \text{ modulo } 15 \\ &= 7 \end{aligned}$$

复原明文P为：

$$\begin{aligned} P &= Cd \text{ modulo } r \\ &= 73 \text{ modulo } 15 \\ &= 343 \text{ modulo } 15 \\ &= 13 \end{aligned}$$

因为e和d互逆，公开密钥加密方法也允许采用这样的方式对加密信息进行“签名”，以便接收方能确定签名不是伪造的。假设A和B希望通过公开密钥加密方法进行数据传输，A和B分别公开加密算法和相应的密钥，但不公开解密算法和相应的密钥。 A和B的加密算法分别是ECA和ECB，解密算法分别是DCA和DCB，ECA和DCA互逆，ECB和DCB互逆。

若A要向B发送明文P，不是简单地发送ECB（P），而是先对P施以其解密算法DCA，再用加密算法ECB对结果加密后发送出去。密文C为：

$$C = \text{ECB}(\text{DCA}(P))$$

B收到C后，先后施以其解密算法DCB和加密算法ECA，得到明文P：

$$\begin{aligned} &\text{ECA}(\text{DCB}(C)) \\ &\quad = \text{ECA}(\text{DCB}(\text{ECB}(\text{DCA}(P)))) \\ &\quad = \text{ECA}(\text{DCA}(P)) && /*\text{DCB和ECB相互抵消}*/ \\ &\quad = P && /*\text{DCB和ECB相互抵消}*/ \end{aligned}$$

这样*B*就确定报文确实是从*A*发出的，因为只有当加密过程利用了DCA算法，用ECA才能获得*P*，只有*A*才知道DCA算法，没有人，即使是*B*也不能伪造*A*的签名。

16.6 SQL的支持

目前的SQL标准只支持自主存取控制。与安全性相关的SQL性质有两个：视图机制，可用于对未授权的用户隐藏敏感数据；授权子系统，允许具有特定权限的用户有选择地动态地将这些权限授予其他用户，并在以后回收这些权限。这两个性质将在下面讨论：

1. 视图和安全性

为了说明SQL中用于安全性目的的视图的使用，给出16.2节例2~例4视图例子的对应的SQL表示，如下所示。

2)
```
CREATE VIEW LS AS
        SELECT S.S#,S.SNAME,S.STATUS,S.CITY
        FROM   S
        WHERE  S.CITY = 'London' ;
```

视图定义了将被授权的数据，而授权是通过GRANT语句实现的，如：

```
GRANT SELECT, UPDATE (SNAME, STATUS), DELETE
ON            LS
TO            Dan, Misha;
```

注意：在SQL中授权是通过GRANT语句而不是通过假定的“CREATE AUTHORITY”语句定义的，因此未对其命名，而完整性约束是必须命名的。

3)
```
CREATE VIEW SSPPR AS
        SELECT S.S#,S.SNAME,S.STATUS,S.CITY
        FROM   S
        WHERE  EXISTS
               (SELECT * FROM P
               WHERE  EXISTS
               (SELECT * FROM P
               WHERE  S.S# = SP.S#
               AND       SP.S# = P.P#
               AND       P.CITY = 'Rome'));
```

相应的GRANT语句：

```
GRANT SELECT ON SSPPR TO Giovanni ;
```

4)
```
CREATE VIEW SSQ AS
       SELECT S.S#,(SELECT SUM(SP.QTY)
                    FROM      SP
                    WHERE SP.S# = S.S#) AS SQ
       FROM   S ;
```

相应的GRANT语句：

```
GRANT SELECT ON SSQ TO Fidel ;
```

16.2节中的例5是关于环境依赖的授权。SQL支持各种无参数内置运算符，如CURRENT_USER、CURRENT_DATE、CURRENT_TIME等，这些运算符可用于定义环境依赖视图。注意：SQL不支持原例5的DAY()运算符。例：

```
CREATE VIEW S_NINE_TO_FIVE AS
       SELECT S.S#,S.SNAME,S.STATUS,S.CITY
       FROM      S
       WHERE     CURRENT_TIME≥ TIME '09:00:00'
       AND       CURRENT_TIME≤ TIME '17:00:00' ;
```

相应的GRANT语句：

```
GRANT SELECT, UPDATE (STATUS)
ON    S_NINE_TO_FIVE
TO    Purchasing ;
```

注意：S_NINE_TO_FIVE是一个很特别的视图！它的值随着时间的变化而变化，即使其基于的数据未改变。另外，包含内置操作符 CURRENT_USER的视图定义对于不同的用户值也不同。这样的“视图”与通常意义上的视图并不相同，事实上它们是参数化的视图。至少从概念上讲，允许用户定义自己的（潜在的参数化的）值依赖的函数更可取，像S_NINE_TO_FIVE这样的“视图”就可看作这类函数。

前面的例子说明了视图机制“免费”提供了一种重要的安全性方法，称其“免费”是因为该机制在系统中本用做其他用途。另外，许多授权检查，甚至是值依赖的检查，都可在编译时进行，而无须等到运行时，这对性能改善有着很重要的意义。然而，基于视图的安全性方法偶尔也会遇到轻微的麻烦，尤其是当用户在同一时间欲对同一个表的不同子集拥有不同的权限时。举个例子，考虑这样的应用，扫描并显示所有 London的零件，同时对其中红色的零件进行更新。

2. GRANT和REVOKE

视图机制用不同的方式将数据库划分为多个片段，使得未授权用户无法获取敏感信息，然而它并未说明授权用户在这些片段上能执行哪些操作。这个任务（前面例子中已涉及）将由GRANT语句来承担，这里将进行详细讨论。

首先注意任何一个对象的创建者都被自动授予了该对象上的所有权限。举个例子，基表 *T* 的创建者将被自动授予表 *T*上的SELECT、INSERT、UPDATE、DELETE以及REFERENCES权限（下面将对这些权限一一解释）。而且这些权限都被授予了“ with grant authority ”，这表明拥有这些权限的用户还可以将这些权限转授给其他用户。

GRANT语句的语法如下：

```
GRANT <privilege commlist>
      ON <object>
      TO <user ID commlist>
  [WITH GRANT OPTION] ;
```

解释：

- 合法的 <*privilege*>有USAGE、SELECT、INSERT、UPDATE、DELETE以及REFERENCES ㊀。USAGE权限用于指定特定的（SQL风格）域；REFERENCES权限用于指出完整性约束的特定的表；其它的权限不言自明。注意： INSERT、UPDATE以及REFERENCES权限（很奇怪，不包括 SELECT权限）可定义于列级。也可指定 ALL PRIVILEGES，但其语义不太直观[4.19]。

㊀ 当永久存储模块(PSM-Persistent Stored Module)特性在1996年加入到标准中时增加了EXECUTE权限。

- 合法的<*object*>为DOMAIN<*domain name*>以及TABLE<*table name*>。其中“TABLE”（实际是可选的）包括视图或基表。
- <*user ID commalist*>可用关键字PUBLIC替代，意味着系统中所有的用户。
- WITH GRANT OPTION，意味着将指定对象上的指定的权限授予指定的用户，同时带有授权权限，即指定的用户可将这些权限继续转授给其他用户。当然，要指定 WITH GRANT OPTION首先要求发出GRANT语句的用户必须具备授权权限。

若用户*A*向用户*B*授予了权限，则也可从用户*B*收回权限。收回权限可通过REVOKE语句实现，其语法为：

```
REVOKE [GRANT OPTION FOR] <privilege commalist>
                          ON <object>
                          FROM <user ID commalist>
                          <option> ;
```

其中：a）GRANT OPTION FOR意味着只有授权权限被收回；

b）<*privilege commalist*>、<*object*>以及<*user ID commalist*>与GRANT语句中的含义相同；

c）<*option*>或者是RESTRICT，或者是CASCADE。例：

```
1) REVOKE SELECT ON S FROM Jacques,Anne,Charley RESTRICT;
2) REVOKE SELECT,UPDATE(SNAME,STATUS),DELETE
          ON LS FROM Dan,Misha CASCADE;
3) REVOKE SELECT ON SSPPR FROM Giovanni RESTRICT;
4) REVOKE SELECT ON SSQ FROM Fidel RESTRICT;
```

RESTRICT与CASCADE: 假设*p*是某对象上的某权限，假设用户*A*将*p*授给用户*B*，*B*又将其转授给用户*C*。如果*A*从*B*回收权限*p*，将会发生什么事情呢？首先，假设REVOKE成功地执行了。那么用户*C*所拥有的权限*p*将不再有效，因为其由用户*B*转授，而*B*不再拥有权限*p*。RESTRICT与CASCADE选项就是为了避免发生这种情形，其中如果有可能导致上述情形时，RESTRICT将会使得REVOKE以失败告终；而CASCADE将会引起上述所有的权限都被回收。

删除一个域、基表、列或视图将自动从所有用户回收被删除对象上的所有权限。

16.7 小结

本章讨论了数据库安全性问题的不同方面。首先对安全性和完整性进行了比较：安全性确保用户被允许做其想做的事情；完整性确保用户所做的事情是正确的。换句话说，安全性即为保护数据以防止不合法的使用造成数据泄漏、更改和破坏。

安全性由DBMS的安全性子系统实施，安全性子系统将对所有的存取请求检查存储在系统目录中的安全性约束。首先讨论的是自主存取控制模式，在该模式下，对指定对象的存取将取决于该对象的所有者。自主模式下的每个授权都包括名字、权限集（RETRIEVE、UPDATE等）、相应的关系变量（即约束所施予的数据）以及用户集。这样的授权可被用于提供值依赖、值无关、统计汇总以及环境依赖控制。审计追踪可用于记录试图破坏安全性的行为。除此以外还简单讨论了请求修改这一用于自主模式的实现技术，该技术最早在Ingres原型系统中的QUEL语言中运用。

强制存取控制模式中每个对象具有密级，而每个用户具有许可证级别。我们解释了该模

式下的存取规则，对美国国防部定义的安全性分级模式 [16.19~16.20]进行了概要介绍,并简单地讨论了多级关系变量和多重实例的思想。

统计数据库有其特有的安全性问题。统计数据库中有着大量的有关个体的相关敏感信息，只向用户提供统计汇总信息，但通过追踪者的方式其安全性很容易受到破坏。事实上，由于数据仓库系统的出现，该问题应受到足够的重视。

本章介绍了数据加密的基本思想：替换和置换，解释了数据加密标准，并描述了公开密钥加密方法的工作原理，特别给出了 RSA（质数）方法的一个例子。另外还介绍了数字签名的概念。

另外还给出了SQL中描述安全性的方法，用于隐藏信息的视图的使用，以及利用 GRANT和REVOKE控制用户对数据对象拥有的权限（主要是基表和视图）。

最后必须强调一点，即使 DBMS提供了大量的安全性控制措施，如果能旁路，这些控制也是没用的。在 DB2中，数据库是以操作系统文件的形式物理存储在磁盘上的，如果有可能通过传统的操作系统服务从传统的程序存取那些文件， DB2的安全机制将失效。因此， DB2与它的伴随系统（即底层的操作系统）协同工作可保证整个系统是安全的。其细节本章不予讨论。

练习

16.1 假定关系变量STATS与16.4节中的定义相同，如下所示：

```
STATS{NAME,SEX,CHILDREN,OCCUPATION,SALARY,TAX,AUDITS}
      PRIMARY KEY{NAME}
```

采用16.2节中的语言定义下面给定的安全性约束：

a. 用户Ford对整个关系变量具有RETRIEVE权限；

b. 用户Smith对整个关系变量具有INSERT和DELETE权限；

c. 每个用户只对自己的元组具有RETRIEVE权限；

d. 用户Nash对整个关系变量具有 RETRIEVE权限，但只对 SALARY和TAX属性具有UPDATE权限；

e. 用户Todd只对NAME、SALARY和TAX属性具有RETRIEVE权限；

f. 用户Ward的RETRIEVE权限与Todd一样，而只对SALARY和TAX属性具有UPDATE权限；

g. 用户Pope具有对职业为preacher的元组的所有权限；

h. 用户Jones具有对职业为非特殊性质的元组的 DELETE权限，这里非特殊性质的职业为人数超过10个的职业；

i. 用户King具有对每个职业最高和最低工资的RETRIEVE权限。

16.2 若考虑对定义和删除关系变量、定义和删除视图以及定义和删除权限等操作的控制，应如何扩展AUTHORITY语句的语法？

16.3 再考虑图16-2，假设从外部资料获悉， Hal是一个至少有两个孩子的 homemaker。利用个体追踪者写出一系列统计查询以获得 Hal的tax数字。就像16.4节一样，假定系统对于集合基数小于2或大于9的查询不响应。

16.4 重复16.3的问题，只是不采用个体追踪者，而采用通用追踪者。

16.5 对下面的密文进行解密，密文采用与16.5节“AS KINGFISHES CATCH FIRE”例子类似的方式，但是利用的是5字节的加密密码：

```
F N W A L
J P V J C
F P E X E
A B W N E
A Y E I P
S U S V D
```

16.6 通过RSA公开密钥加密方法进行工作，给定 p=7、q=5、e=17以及明文P=3。

16.7 你能想出任何可能有加密引起的实现上的问题以及其他不利条件吗？

16.8 给出练习16.1的SQL解决方案。

16.9 写出将练习16.8的解决方案中所授予的权限删除的SQL语句。

参考文献和简介

16.1 D. E. Bell and L.J.La Padula: “Secure Computer Systems: Mathematical Foundation and Model,” MITRE Technical Report M74-244(May 1974).

16.2 Silvana Castano,Mariagrazia Fugini,Giancarlo Martella,and Pierangela Samarati: *Database Security*. New York,N.Y.: ACM Press/Reading,Mass.: Addison-Wesley(1995).

16.3 James Daly: “Fingerprinting a Computer Security Code,” *Computerworld* (July 27th,1992).

16.4 C.J.Date: “Security,” Chapter 4 of *An Introduction to Database Systems:Volume II*. Reading, Mass.:Addison-Wesley(1983).

16.5 Dorothy E.Denning: *Cryptography and Data Security*. Reading Mass.: Addison-Wesley(1983).

16.6 Dorothy E.Denning and Peter J.Denning: “Data Security,” *ACM Comp.Surv.11*, No.3 (Septemper 1979).

一本不错的入门书，包括自主存取控制、强制存取控制（此处称为流控）、数据加密以及推理控制（统计数据库的专有问题）。

16.7 W.Diffie and M.E.Hellman: “New Direction in Cryptography,” *IEEE Transactions on Information Theory* IT-22(November 1976).

16.8 Ronald Fagin: “On an Authorization Mechanism,” *ACM TODS 3*.No.3(September 1978).

该文章是对参考文献[16.11]的勘误。由于[16.11]的机制在一定的情况下将回收不应该回收的权限，这篇文章正是对此进行了纠正。

16.9 Roberto Gagliardi,George Lapis,and Burce Lindsay: “A Flexible and Efficient Database Authorization Facility,” IBM Research Report RJ6826(May 11th,1989).

16.10 Martin Gardner: “A New Kind of Cipher That Would Take Millions of Years to Break,” *Scientific American 237*, No.2(August 1977).

对公用密钥加密方法的非正式的介绍，[16.12,16.22]将有进一步的说明。

16.11 Patricia P.Griffiths and Bradford W.Wade: “An Authorization Mechanism for a Relational Data Base System,” *ACM TODS 1*, NO.3(September 1976).

描述了System R中最先采用的GRANT和REVOKE机制，该机制是SQL标准所采用的模式的基础，虽然在实现细节上并不相同。

16.12 Nigel Hawkes: “Breaking into the Internet,” *London Times* (March 18th,1996).

描述了一个计算机专家是怎样通过估计系统对消息解密花费的时间的方法破解了RSA模式，这其实是“通过观察一个人拨号以及每次拨号花费的时间的方法猜测锁的组合方式的电子等价物”。

16.13 Sushil Jajodia and Ravi Sandhu: “Toward a Multi-Level Secure Relational Data Model,” Proc.1991 ACM SIGMOD Int. Conf. on Management of Data, Denver, Colo.(June 1991).

正如16.3节中所指出的，安全性环境下的“多级”说明了系统支持强制存取控制。这篇文章暗示了此领域当前的工作很特殊，因为在基本的概念上尚未达成一致，文章提出了多级系统原理的形式化说明。

16.14 Abraham Lempel: “Cryptology in Transaction,” *ACM Comp.Surv.11*, No.4:Special Issue on Cryptology(December 1979).

关于加密及相关内容的入门书。

16.15 R.L.Rivest, A.Shamir,and L.Adleman: “A Method for Obtaining Digital Signatures and Public Key Cryptosystems,” *CACM 21*, No.2(February 1978).

16.16 Ken Smith and Marianne Winslett: “Entity Modeling in the MLS Relational Model,” Proc.18th Int. Conf. on Very Large Data Bases,Vancouver,Canada(Angust 1992).

论文中的“MLS”表示“多级安全”[16.13]。这篇论文着重讨论了MLS数据库的意义，并提出了新的BELIEVED BY子句用于查询和更新操作，使这些操作处于数据库理解或为特定用户“相信”的状态。该方法声称可解决用以前的方法解决的许多问题。参看[16.21]。

16.17 Bhavani Thuraisingham: “Current Status of R&D in Trusted Database Management System,” *ACM SIGMOD Record 21*, No.3(September 1992).

关于“可信”或多级系统（20世纪90年代早期）的综述以及大量的参考文献的集合。

16.18 U.S.Department of Commerce/National Bureau of Standands: *Data Encryption Standard*. Federal Information Processing Standards Publication 46(1977 January 15).

定义了官方的数据加密标准（DES），由联邦机构以及任何愿意采用该标准的个人使用。加密/解密算法（参看16.5节）适合在硬件芯片上实现，这意味着采用该算法的设备能以很高的数据速率操作。已有大量这样的设备可购得。

16.19 U.S.Department of Defense: *Trusted Computer System Evaluation Criteria* (the “Orange Book”),Document No.DoD 5200-28-STD.DoD National Computer Security Center(December 1985).

16.20 U.S.National Computer Security Center: *Trusted Database Management System Intepretation of Trusted Computer System Evaluation Criteria* (the “Lavender Book”),Document No.NCSC-TG-201,Version 1(April 1991).

16.21 Marianne Winslett,Kenneth Smith,and Xiaolei Qian: “Formal Query Languages for Secure Relational Databases,” *ACM TODS 19*, No.4(December 1994).

继续[16.16]的工作。

16.22 Ron Wolf: "How Safe Is Computer Data?A Lot of Factors Govern the Answer," *San Jose Mercury News* (July 5th,1990).

部分练习答案

16.1

a.
```
AUTHORITY AAA
    GRANT RETRIEVE ON STATS TO Ford ;
```

b.
```
AUTHORITY BBB
    GRANT INSERT,DELETE ON STATS TO Smith ;
```

c.
```
AUTHORITY CCC
    GRANT RETRIEVE
    ON    STATS
    WHEN  USER() = NAME
    TO    ALL ;
```

这里假定用户将自己的名字作为 ID。注意WHEN子句和无参数内置运算符 USER（）的使用。

d.
```
AUTHORITY DDD
    GRANT RETRIEVE,UPDATE(SALARY,TAX)
    ON  STATS
    TO  Nash ;
```

e.
```
AUTHORITY EEE
    GRANT RETRIEVE(NAME,SALARY,TAX)
    ON    STATS
    TO    Todd ;
```

f.
```
AUTHORITY FFF
   GRANT RETRIEVE(NAME,SALARY,TAX),
         UPDATE(SALARY,TAX)
   ON    STATS
   TO    Ward ;
```

g.
```
VAR PREACHERS VIEW
     STATS WHERE OCCUPATION = 'Preacher' ;

AUTHORITY GGG
    GRANT ALL
    ON    PREACHERS
    TO    Pope ;
```

注意该例中必须使用视图。

h.
```
VAR NONSPECIALIST VIEW
     WITH (STATS RENAME OCCUPATION AS X) AS T1,
          (EXTEND STATS
          ADD COUNT (T1 WHERE X = OCCUPATION) AS Y) AS T2,
          (T2 WHERE Y > 10) AS T3:
          T3 {ALL BUT Y}

AUTHORITY HHH
    GRANT DELETE
```

```
            ON NONSPECIALIST
            TO Jones ;
i. VAR JOBMAXMIN VIEW
   WITH(STATS RENAME OCCUPATION AS X) AS T1,
       (EXTEND STATS
        ADD MAX(T1 WHERE X = OCCUPATION,SALARY) AS MAXSAL,
            MIN(T1 WHERE X = OCCUPATION,SALARY) AS MINSAL)
                                  AS T2:
       T2{OCCUPATION,MAXSAL,MINSAL}

       AUTHORITY III
            GRANT RETRIEVE ON JOBMAXMIN TO King ;
```

16.2 **这里考虑一点：具有创建一个新的基变量权限、且事实上创建了一个基变量的用户被认为是该新变量的所有者。就像**SQL**中，基变量的所有者将被自动授予该变量上的所有可能的权限，不仅包括**RETRIEVE、INSERT、UPDATE**和**DELETE**权限，而且包括定义将该变量上的权限授予其他用户的授权的权限。**

16.3 Hal**的个体追踪者为**

```
CHILDREN > 1 AND NOT(OCCUPATION = 'Homemaker')
```

考虑如下的查询序列：

```
COUNT (STATS WHERE CHILDREN > 1
```

结果：6。

```
COUNT (STATS WHERE CHILDREN > 1 AND NOT
                (OCCUPATION = 'Homemaker'))
```

结果：5。

因此表达式

```
CHILDREN > 1 AND NOT(OCCUPATION = 'Homemaker')
```

唯一标识了Hal。

```
SUM (STATS WHERE CHILDREN > 1, TAX)
```

结果：48K。

```
SUM (STATS WHERE CHILDREN > 1 AND NOT
               (OCCUPATION = 'Homemaker'), TAX)
```

结果：46K。

因此Hal**的**tax**值为**2K。

16.4 **通用追踪者为：**SEX = 'F'。

```
SUM (STATS WHERE SEX = 'F', TAX)
```

结果：70K。

```
SUM (STATS WHERE NOT(SEX = 'F'), TAX)
```

结果：16K。

因此总的tax**为**86K。

```
SUM (STATS WHERE(CHILDREN > 1 AND
                 OCCUPATION = 'Homemaker')OR
                 SEX = 'F', TAX)
```

结果：72K。

```
SUM (STATS WHERE (CHILDREN > 1 AND
                  OCCUPATION = 'Homemaker' OR NOT
                  SEX = 'F', TAX)
```

结果：16K。

将上面两个结果相加并减去前面计算的结果，可得到 Hal**的**tax**值为**88K-86K=2K。

16.5 **明文是**

```
EYES I DARE NOT MEET IN DREAMS
```

加密密钥是什么？

16.7 **一个问题是，即使在支持加密的系统中，数据内部处理时还必须以明文的格式（如为了比较操作能正确执行），那么在应用程序执行的同时，或在内存数据转储到磁盘上的同时敏感数据可能被获取。而且索引已加密的数据以及对这样的数据进行日志维护都存在着严重的技术问题。**

16.8

a.
```
GRANT SELECT ON STATS TO Ford ;
```

b.
```
GRANT INSERT,DELETE ON STATS TO Smith ;
```

c.
```
CREATE VIEW MINE AS
          SELECT STATS.*
          FROM  STATS
          WHERE STATS.NAME = CURRENT_USER ;
   GRANT SELECT ON MINE TO PUBLIC ;
```

这里假定用户将自己的名字作为 ID**。注意无参数内置运算符** CURRENT_USER**的使用。**

d.
```
GRANT SELECT,UPDATE(SALARY,TAX)
       ON STATS
       TO Nash ;
```

e.
```
CREATE VIEW UST AS
         SELECT STATS.NAME,STATS.SALARY,STATS.TAX
         FROM  STATS ;
  GRANT SELECT ON UST TO Todd ;
```

SQL**必须使用视图，因为其并不支持列级** SELECT**权限。**

f.
```
GRANT SELECT,UPDATE(SALARY,TAX) ON UST TO Ward ;
```

利用前一道题中的同一视图。

g.
```
CREATE VIEW PREACHERS AS
        SELECT STATS.*
        FROM   STATS
        WHERE STATS.OCCUPATION = 'Preacher' ;
  GRANT  ALL PRIVILEGES ON PREACHERS TO Pope ;
```

注意该例中简写“ ALL PRIVILEGES **”的使用。但是，**ALL PRIVILEGES**并不像其字面上那样意味着所有的权限，而是指执行**GRANT**语句的用户对相关对象所能授予的所有权限。**

h.
```
CREATE VIEW NONSPECIALIST AS
        SELECT STX.*
        FROM   STATS AS STX
        WHERE  (SELECT COUNT(*)
                FROM   STATS AS STY
```

```
                    WHERE  STY.OCCUPATION = STX.OCCUPATION) >10 ;
    GRANT DELETE ON SPECIALISTS TO Jones ;
```

i.
```
CREATE VIEW JOBMAXMIN AS
        SELECT STATS.OCCUPATION,
            MAX(STATS.SALARY) AS MAXSAL,
            MIN(STATS.SALARY) AS MINSAL
        FROM  STATS
        GROUP BY STATS.OCCUPATION ;
GRANT SELECT ON JOBMAXMIN TO King ;
```

16.9

a. `REVOKE SELECT ON STATS FROM Ford RESTRICT ;`
b. `REVOKE INSERT,DELETE ON STATS FROM Smith RESTRICT ;`
c. `REVOKE SELECT ON MINE FROM PUBLIC RESTRICT ;`
d.
```
REVOKE SELECT,UPDATE(SALARY,TAX)
                    ON STATS FROM Nash RESTRICT ;
```
e. `REVOKE SELECT ON UST FROM Todd RESTRICT ;`
f.
```
REVOKE SELECT,UPDATE(SALARY,TAX)
             ON UST FROM Ward RESTRICT ;
```
g. `REVOKE ALL PRIVILEGES ON PREACHERS FROM Pope RESTRICT ;`
h. `REVOKE DELETE ON NONSPECIALIST FROM Jones RESTRICT`
i. `REVOKE ON SALS FROM King RESTRICT`