第9章 数据库完整性和安全性

9.1 引言

9.2 数据库的完整性

9.3 数据库的安全性

9.1 引言

- 数据库是长期保存在计算机内的有组织的可共享数据集合
- 在投入运行后，一方面需要利用数据库系统统一管理和共享数据
- 另一方面需要提供必要的措施保证其中数据的完整性和安全性

9.1 引言

- 数据库中发生数据错误，归纳起来主要有以下四个方面的原因：
  1. 系统发生软、硬件故障，数据遭到破坏
  2. 事务并发执行引起数据的不一致性
  3. 数据库的更新操作有误，如输入本身是错误的数据、不正确的操作或程序产生的不一致数据等。
  4. 自然的或人为的破坏，例如意外事故（失火、失窃等）、恶意的攻击篡改数据等等

9.1 引言

- 针对这四种情况，DBMS必须提供以下几项数据控制功能：
  1. 数据库恢复：将数据库从错误状态恢复到某一已知的一致状态的功能
  2. 并发控制：协调并发事务的执行，并维护数据的一致性
  3. 保持完整性：数据库中数据始终保证是正确的和一致的
  4. 保证安全性：防止非法使用数据库，造成数据泄露、篡改和破坏

9.2 数据库完整性

- 数据库的完整性是指数据的正确性、有效性和相容性
- 数据库是否具备完整性关系到数据库系统能否真实地反映现实世界，因此维护数据库的完整性是非常重要的
- 为维护数据库的完整性，DBMS必须提供一种机制来检查数据库中的数据，看其是否满足语义规定的条件

完整性约束条件

- 完整性检查是围绕完整性约束条件进行的，因此，完整性约束条件是完整性控制机制的核心
- 我们将加在数据库之上的语义约束条件称为数据库完整性约束条件，它们作为模式的一部分存入数据库中
- 而DBMS中检查数据是否满足完整性条件的机制称为完整性检查

完整性约束条件

- 完整性约束条件作用的对象可以是关系、元组、列三种
- 其中列约束主要是列的类型、取值范围、精度、排序等的约束条件
- 元组的约束是元组中各个字段间的联系的约束
- 关系的约束是若干元组间、关系集合上以及关系之间的联系的约束

完整性约束条件

- 完整性约束条件涉及的这三类对象，其状态可以是静态的，也可以是动态的
- 所谓静态约束是指数据库每一确定状态时的数据对象所应满足的约束条件，它是反映数据库状态合理性的约束，这是最重要的一类完整性约束
- 动态约束是指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，它是反映数据库状态变迁的约束
- 综合上述两个方面，我们可以将完整性约束条件分为6类

完整性约束条件

1. 静态列级约束：是对一个列的取值域的说明，这是最常用也最容易实现的一类完整性约束，包括以下几方面：
   1) 对数据类型的约束，包括数据的类型、长度、单位、精度等；
   2) 对数据格式的约束；
   3) 对取值范围或取值集合的约束；
   4) 对空值(NULL)的约束;
   5) 其他约束

完整性约束条件

2. 静态元组约束：一个元组是由若干个列值组成的，静态元组约束就是规定元组的各个列之间的约束关系

完整性约束条件

3. 静态关系约束:在一个关系的各个元组之间或者若干关系之间常常存在各种联系或约束。常见的静态关系约束有：
   1) 实体完整性约束；
   2) 参照完整性约束；
      实体完整性约束和参照完整性约束是关系模型的两个极其重要的约束，称为关系的两个不变性。
   3) 函数依赖约束。大部分函数依赖约束都在关系模式中定义。
   4) 统计约束。即字段值与关系中多个元组的统计值之间的约束关系。

完整性约束条件

4. 动态列级约束：是修改列定义或列值时应满足的约束条件，包括下面两方面：

   1) 修改列定义时的约束

      例如，将允许空值的列改为不允许空值时，如果该列目前已存在空值，则拒绝这种修改

   2) 修改列值时的约束

      修改列值有时需要参照其旧值，并且新旧值之间需要满足某种约束条件。例如，职工工资调整不得低于其原来工资，学生年龄只能增长等等

完整性约束条件

5. 动态元组约束：动态元组约束是指修改元组的值时元组中各个字段间需要满足某种约束条件。

   例如职工工资调整时新工资不得低于 原工资+工龄*5，等等。

6. 动态关系约束：动态关系约束是加在关系变化前后状态上的限制条件，例如事务一致性、原子性等约束条件

完整性控制

- DBMS的完整性控制机制应具有三个方面的功能：
  1. 定义功能，提供定义完整性约束条件的机制
  2. 检查功能，检查用户发出的操作请求是否违背了完整性约束条件
  3. 如果发现用户的操作请求使数据违背了完整性约束条件，则采取一定的动作来保证数据的完整性

完整性控制

- 完整性约束条件包括有6类，约束条件可能非常简单，也可能极为复杂
- 一个完善的完整性控制机制应该允许用户定义所有这六类完整性约束条件

完整性控制

- 检查是否违背完整性约束的时机通常是在一条语句执行完后立即检查，我们称这类约束为立即执行约束（**Immediate constraints**）
- 有时完整性检查需要延迟到整个事务执行结束后再进行，检查正确方可提交，我们称这类约束为延迟执行约束（**Deferred constraints**）
- 在关系系统中，最重要的完整性约束是实体完整性和参照完整性，其他完整性约束条件可以归入用户定义的完整性
- 下面详细讨论实现参照完整性要考虑的几个问题

完整性控制

1) 外键能否接受空值问题

- 在实现参照完整性时，系统除了应该提供定义外键的机制，还应提供定义外键列是否允许空值的机制。

2) 在被引用关系中删除元组的问题

- 一般地，当删除被引用关系的某个元组，而引用关系存在若干元组，其外键值与被引用关系删除元组的主键值相同，这时可有三种不同的策略：

(1) 级联删除（CASCADE）

(2) 受限删除（RESTRICT）

- 仅当引用关系中没有任何元组的外键值与被引用关系中要删除元组的主键值相同时，系统才执行删除操作，否则拒绝此删除操作。

(3) 置空值删除（SET NULL）

- 删除被引用关系的元组，并将引用关系中相应元组的外键值置空值。

完整性控制

- 例如要删除被引用关系Student中学号等于2170500166的学生元组，而在引用关系SC中存在学号等于2170500166的选课元组。
- (1) 级联删除（CASCADE）
- (2) 受限删除（RESTRICT）
- (3) 置空值删除（SET NULL）

完整性控制

3) 在引用关系中插入元组时的问题

- 一般地，当引用关系插入某个元组，而被引用关系不存在相应的元组，其主键值与引用关系插入元组的外键值相同，这时可有以下策略：
  （1）受限插入
  （2）递归插入

完整性控制

- 例如将一个新的选课元组（学号：2170500166；课程号：CS-01；成绩：88）插入引用关系SC中，而在被引用关系Student中暂时不存在学号等于2170500166的学生元组。
- (1) CASCADE
- (2) RESTRICT

完整性控制

4) 修改关系中主键的问题

(1) 不允许修改主键

(2) 允许修改主键

- 在有些RDBMS中，允许修改关系主键，但必须保证主键的唯一性和非空，否则拒绝修改（CASCADE、RESTRICT、SET NULL）

- 从上面的讨论我们看到DBMS在实现参照完整性时，除了要提供定义主键、外键的机制外，还需要提供不同的策略供用户选择。选择哪种策略，都要根据应用环境的要求确定

完整性约束的说明

- 完整性约束的说明一般分为隐含约束和显式约束两类
- 隐含约束的说明可用于实现域完整性约束、实体完整性约束以及参照完整性约束等，如定义主关键字、外关键字；在设计关系数据模式时，还可定义属性的类型、长度、单位、精度等

完整性约束的说明

- DBMS根据这些定义对数据库的更新操作自动检查，维护数据的完整性，这是数据库中最基本、最普遍的约束
- 显式约束的说明可实现一些隐含约束无法表达的约束，这些约束依赖于数据的语义和应用，本身比较复杂
- 用户根据不同DBMS的支持程度，一般选择使用4种可能的方法

完整性约束的说明

1．CHECK子句

- 在基表定义中，加入一个**CHECK**子句，利用此子句说明各列的值应满足的约束条件
- 这种约束是对单个关系的元组值进行约束，条件中也可以涉及本关系的其它元组或其它关系的元组
- 此时，**CHECK**子句只对定义它的关系起约束作用，对于条件中提到的其它关系没有约束作用

完整性约束的说明

2. 断言

- 每个断言就是一个谓词，指定了数据库状态必须满足的逻辑条件
- 利用断言表达数据库完整性约束，形成一系列断言的集合
- **DBMS**对每个更新事务，用集合中的断言对它进行检查。只有不破坏断言的更新操作才允许，否则，就撤销事务

完整性约束的说明

2．断言

- SQL中定义断言用CREATE语句，形式如下：

  CREATE ASSERTION <断言名>

  CHECK <逻辑条件>;

- 这里<逻辑条件>和SQL中查询语句的查询条件一样
- 撤销断言用DROP语句，形式如下：

  DROP ASSERTION <断言名>;

完整性约束的说明

2. 断言

- 例：在产品定单(Order)中，可以通过下面的断言限定其销售价格(sale_price)不得小于相关产品报价(list_price)的80%：

```
CREATE  ASSERTION  price-constraint CHECK
(NOT  EXISTS
(SELECT  *
 FROM   Order O
 WHERE  sale_price  <
   (SELECT 0.8 * list_price
    FROM Product P
    WHERE O.PNO=P.PNO
)
);
```

完整性约束的说明

2. 断言

- 断言的引入减少了编程的麻烦，本身也比较容易维护
- 但是如果断言较复杂，则给创建和检查会带来相当大的开销，同时降低了数据库更新操作的性能
- 因此，对于断言的使用应该权衡考虑

完整性约束的说明

3．存储过程

- 存储过程是指将常用的访问数据库的程序，作为一个过程进行编译，然后存储在数据库中，供用户调用
- 这样做既改善性能，又方便用户，还扩充了功能
- 将存储过程应用到完整性约束，就是在应用程序中插入一些存储过程说明完整性约束，在更新数据库时执行，完成数据库完整性的检查
- 如果违反给定的约束，则撤销事务，或者修改数据使数据库保持完整性

完整性约束的说明

4. 触发器

- 触发器是一条在对数据库执行更新操作时系统自动执行的语句，又称主动规则或ECA（事件—条件—动作）规则
- 即当发生某一事件时，如果满足给定的条件，则执行相应的动作
- 利用触发器可以表示约束，以某种更新数据库的操作作为事件，以违反约束作为条件，以违反约束的处理作为动作

完整性约束的说明

4. 触发器

- 在插入、删除、修改等更新操作发生时，触发器开始工作
- 先检查完整性约束条件是否满足，如果满足就执行相应的动作
- 动作可以有很多形式，如撤销事务，或传递一个消息给用户，或完成某些对数据库的操作，以保持数据库的完整性
- 触发器的用途不止这一点，它还可以用在监视数据库的状态等

完整性约束的说明

4. 触发器

- 例：在产品定单(Order)中，也可以通过下面的触发器限定其销售价格(sale_price)不得小于相关产品报价(list_price)的80%：

```
CREATE  TRIGGER price-constraint
BEFORE  INSERT  ON  Order
REFERENCING  NEW  AS  N
FOR  EACH  ROW
WHEN (N. sale_price<
(SELECT 0.8 * list_price
    FROM Product P
    WHERE N.PNO=P.PNO )
ROLLBACK;
```

9.3 数据库系统安全性

- 在一般计算机系统中，安全措施是一级一级层层设置的。例如可以有如下的模型：

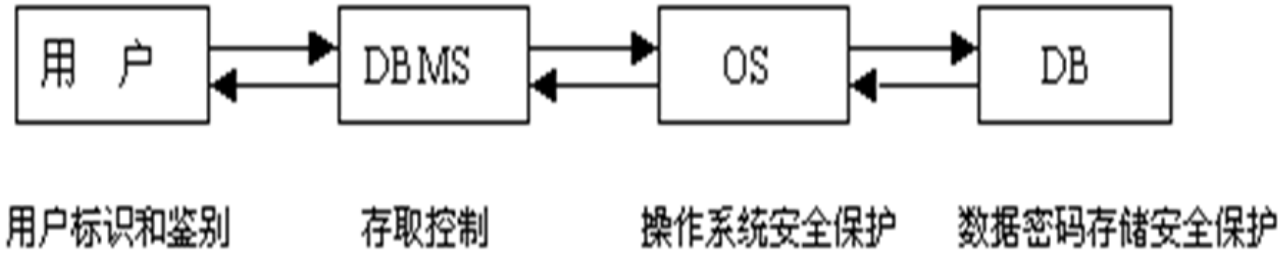


图 9.1 计算机系统的安全模型

用户标识和鉴别

- 用户标识和鉴别是系统提供的最外层安全保护措施
- 其方法是由系统提供一定的方式让用户标识自己的名字或身份
- 每次用户要求进入系统时，由系统进行核对，通过鉴定后才提供机器使用权
- 例如，使用用户名和口令

存取控制

- 数据库安全性所关心的主要是DBMS的存取控制机制
- 数据库安全最重要的一点就是确保只授权给有资格的用户访问数据库的权限，同时保证所有未被授权的人员无法接近数据，这主要通过数据库系统的存取控制机制实现

数据库系统的存取控制机制

- 存取控制机制主要包括两部分：
  - 定义用户权限，并将用户权限登记到数据目录中
  - 合法权限检查，每当用户发出存取数据库的操作请求后（请求一般应包括操作类型、操作对象和操作用户等信息），DBMS查找数据目录，根据安全规则进行合法权限检查，若用户的操作请求超出了定义的权限，系统将拒绝执行此操作
- 用户权限定义和合法权检查机制一起组成了DBMS的安全子系统

数据库系统的存取控制机制

- 当前大型的DBMS一般都支持C2级中的自主存取控制 （DAC）
- 有些DBMS同时还支持B1级中的强制存取控制(MAC)
- 在强制存取控制中，每一个数据对象被标以一定的“密级”，每一个用户也被授予某一个级别的“许可证”
- 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，强制存取控制相对自主存取控制更加严格

自主存取控制

- 大型数据库管理系统几乎都支持自主存取控制，目前的SQL标准也对自主存取控制提供支持，这主要通过SQL 的GRANT语句和REVOKE语句来实现
- 用户权限是由两个要素组成的：数据对象和操作类型
- 定义一个用户的存取权限就是要定义这个用户可以在哪些数据对象上进行哪些类型的操作

自主存取控制

- 在SQL中，有两种授权的方式。一种是在数据库一级，由DBA负责授予和收回用户需要得到的必要特权，只有取得这种授权，才能成为数据库用户；另一种是把自身拥有的权限转授给其他用户的授权，可以由DBA或者数据对象的创建者授予对某些数据对象进行某些操作的特权。

自主存取控制

- 第一种授权使用GRANT语句的语句格式如下
- GRANT<特权类型>[{，<特权类型〉}] TO <用户名>[IDENTIFIED BY ＜口令>]；
- 收回权限使用REVOKE语句的语句格式如下
- REVOKE<特权类型>[{，<特权类型〉}] FROM <用户名>;
- <特权类型>::＝CONNECT|RESOURCE|DBA

自主存取控制

- 例9-1 授予用户USER1在数据库上的CONNECT特权。
- GRANT CONNECT TO USER1 IDENTIFIED BY 1234;
- 收回用户USER1在数据库上的RESOURCE特权。
- REVOKE RESOURCE FROM USER1;

自主存取控制

- 第二种授权使用GRANT语句的语句格式如下。
- GRANT<特权>ON<数据对象>TO<受权者>[{,<受权者>}][WITH GRANT OPTION];
- <特权>::＝ALL PRIVILIEGES|<操作>[{,<操作>}]
- <操作>::＝SELECT|INSERT|UPDATE|DELETE[(<准许修改的属性表>)]
- <准许修改的属性表>::＝<属性名>[{,<属性名>}]
- <受权者>::＝PUBLIC|<用户名>
- 收回特权可用下面的REVOKE语句：
- REVOKE<特权>ON<数据对象>FROM<受权者>[{,<受权者>}][RESTRICT|CASCADE];

自主存取控制

- 例9-2 授予用户USER1在Shop表上的SELECT和INSERT特权。
- GRANT SELECT,INSERT ON TABLE Shop TO USER1;
- 授予用户USER1在Customer表上的所有特权。
- GRANT ALL PRIVILIEGES ON TABLE Customer TO USER1;
- 授予用户USER1、USER2在SC表上的SELECT特权和对属性Item、Quantity的UPDATE权限。
- GRANT SELECT,UPDATE(Item,Quantity) ON TABLE SC TO USER1,USER2;

自主存取控制

- 收回前面所授予的权限。
- **REVOKE SELECT,INSERT ON TABLE Shop FROM USER1;**
- **REVOKE ALL PRIVILIEGES ON TABLE Customer FROM USER1;**
- **REVOKE SELECT,UPDATE(Item,Quantity) ON TABLE SC FROM USER1,USER2;**

自主存取控制

- 在数据库系统中，定义存取权限称为授权**(Authorization)**
- 用户权限定义中数据对象范围越小授权子系统就越灵活。
- 例如，上面的授权定义可精细到字段级，而有的系统只能对关系授权。授权粒度越细，授权子系统就越灵活，但系统定义与检查权限的开销也会相应地增大

自主存取控制

- 衡量授权子系统精巧程度的另一个尺度是能否提供与数据值有关的授权
- 如上面的授权定义是独立于数据值的，即用户能否对某类数据对象执行的操作与数据值无关，完全由数据名决定
- 反之，若授权依赖于数据对象的内容，则称为是与数据值有关的授权。
- 有的系统还允许存取谓词中引用系统变量，如一天中的某个时刻，某台终端设备号,这样用户只能在某台终端、某段时间内存取有关数据，这就是与时间和地点有关的存取权限

自主存取控制

- 另外，我们还可以在存取谓词中引用系统变量，如终端设备号，系统时钟等，这就是与时间地点有关的存取权限，这样用户只能在某段时间内，某台终端上存取有关数据
- 自主存取控制能够通过授权机制有效地控制其他用户对敏感数据的存取
- 但是由于用户对数据的存取权限是“自主”的，用户可以自由地决定将数据的存取权限授予何人、决定是否也将“授权”的权限授予别人
- 在这种授权机制下，仍可能存在数据的“无意泄露”

强制存取控制

- 所谓强制存取控制是指系统为保证更高程度的安全性，按照TDI/TCSEC标准中安全策略的要求，所采取的强制存取检查手段
- 强制存取控制不是用户能直接感知或进行控制的
- 强制存取控制适用于那些对数据有严格而固定密级分类的部门，例如军事部门或政府部门
- 在强制存取控制中，DBMS所管理的全部实体被分为主体和客体两大类

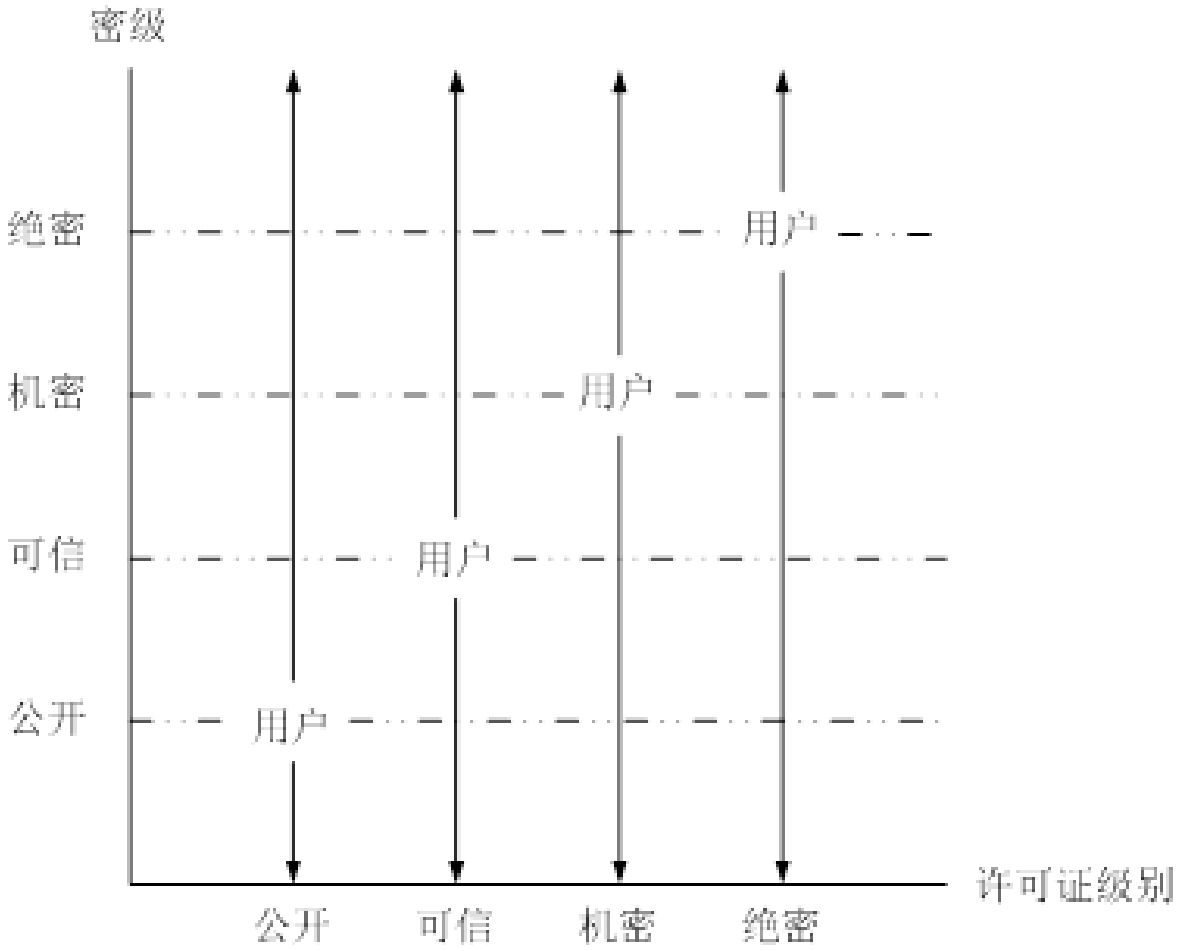
强制存取控制

- 主体是系统中的活动实体，既包括DBMS所管理的实际用户，也包括代表用户的各进程
- 客体是系统中的被动实体，是受主体操纵的，包括文件、基表、索引、视图等等
- 对于主体和客体，DBMS为它们每个实例（值）指派一个敏感度标记（Label）

强制存取控制

- 敏感度标记被分成若干级别，例如绝密(Top Secret)、机密(Secret)、可信(Confidential)、公开(Public)等
- 主体的敏感度标记称为许可证级别(Clearance Level)，客体的敏感度标记称为密级（Classification Level）
- MAC机制就是通过对比主体的Label和客体的Label，最终确定主体是否能够存取客体

强制存取控制



强制存取控制

- 当某一用户（或一主体）以标记label注册入系统时，系统要求他对任何客体的存取必须遵循如下规则：
  1. 仅当主体的许可证级别大于或等于客体的密级时，该主体才能读取相应的客体
  2. 仅当主体的许可证级别等于客体的密级时，该主体才能写相应的客体

强制存取控制

- 规则(1)的意义是明显的。而规则(2)需要解释一下
- 在某些系统中，第(2)条规则与这里的规则有些差别。这些系统规定：仅当主体的许可证级别小于或等于客体的密级时，该主体才能写相应的客体，即用户可以为写入的数据对象赋予高于自己的许可证级别的密级
- 这样一旦数据被写入，该用户自己也不能再读该数据对象了
- 这两种规则的共同点在于它们均禁止了拥有高许可证级别的主体更新低密级的数据对象，从而防止了敏感数据的泄漏

强制存取控制

- 强制存取控制(MAC)是对数据本身进行密级标记，无论数据如何复制，标记与数据是一个不可分的整体，只有符合密级标记要求的用户才可以操纵数据，从而提供了更高级别的安全性
- 前面已经提到，较高安全性级别提供的安全保护要包含较低级别的所有保护，因此在实现MAC时要首先实现DAC，即DAC与MAC共同构成DBMS的安全机制
- 系统首先进行DAC检查, 对通过DAC检查的允许存取的数据对象再由系统自动进行MAC检查，只有通过MAC检查的数据对象方可存取

视图机制

- 进行存取权限控制时我们可以为不同的用户定义不同的视图，把数据对象限制在一定的范围内，也就是说，通过视图机制把要保密的数据对无权存取的用户隐藏起来，从而自动地对数据提供一定程度的安全保护
- 视图机制间接地实现了支持存取谓词的用户权限定义
- 在不直接支持存取谓词的系统中，可以先建立视图，然后在视图上进一步定义存取权限

审计

- 前面讲的用户标识与鉴别、存取控制仅是安全性标准的一个重要方面，但不是全部
- 因为，任何系统的安全保护措施都不是完美无缺的，蓄意盗窃、破坏数据的人总是想方设法打破控制
- 为了使DBMS达到一定的安全级别，还需要在其它方面提供相应的支持
- 例如按照TDI/TCSEC标准中安全策略的要求，“审计”功能就是DBMS达到C2以上安全级别必不可少的一项指标

审计

- 审计功能把用户对数据库的所有操作自动记录下来放入审计日志（Audit Log）中
- DBA可以利用审计跟踪的信息，重现导致数据库现有状况的一系列事件，找出非法存取数据的人、时间和内容等
- 审计通常是很费时间和空间的，所以DBMS往往都将其作为可选特征，允许DBA根据应用对安全性的要求，灵活地打开或关闭审计功能
- 审计功能一般主要用于安全性要求较高的部门

数据加密

- 对于高度敏感性数据，例如财务数据、军事数据、国家机密，除以上安全性措施外，还可以采用数据加密技术
- 数据加密是防止数据库中数据在存储和传输中失密的有效手段
- 加密的基本思想是根据一定的算法将原始数据(术语为明文，**Plain text**)变换为不可直接识别的格式(术语为密文，**Cipher text**)，从而使得不知道解密算法的人无法获知数据的内容

数据加密

- 加密方法主要有两种，一种是替换方法，该方法使用密钥（Encryption Key）将明文中的每一个字符转换为密文中的一个字符
- 另一种是置换方法，该方法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。采用这种结合算法的例子是美国1977年制定的官方加密标准，数据加密标准（Data Encryption Standard，简称DES）
- 有关DES秘密密钥加密技术及密钥管理问题等已超出本课程的范围，这里不再讨论