第5章 数据库的安全性

数据或信息是现代信息社会的五大“经济要素”(人、财、物、信息、技术)之一,是与财物同等重要(有时甚至更重要)的资产。企业数据库中的数据对于企业是至关重要的,尤其是一些敏感性的数据,必须加以保护,以防止故意破坏或改变、未授权存取和非故意损害。其中,非故意损害属于数据完整性和一致性保护问题,故意破坏或改变、未授权存取属于数据库安全保护问题,本章将予以专门讨论。

5.1 数据库安全性概述

数据库的安全性(Security)是指保护数据库,防止不合法的使用,以免数据的泄露、更改或破坏。

数据库的“安全性”和“完整性”这两个概念听起来有些相似,有时容易混淆,但两者是完全不同的。

(1) 安全性:保护数据以防止非法用户故意造成的破坏,确保合法用户做其想做的事情。

(2) 完整性:保护数据以防止合法用户无意中造成的破坏,确保用户所做的事情是正确的。

两者不同的关键在于“合法”与“非法”和“故意”与“无意”。

为了保护数据库,防止故意的破坏,可以在从低到高的5个级别上设置各种安全措施。

(1) 物理控制:计算机系统的机房和设备应加以保护,通过加锁或专门监护等防止系统场地被非法进入,从而进行物理破坏。

(2) 法律保护:通过立法、规章制度防止授权用户以非法的形式将其访问数据库的权限转授给非法者。

(3) 操作系统(OS)支持:无论数据库系统是多么安全,操作系统的安全弱点均可能成为入侵数据库的手段,应防止未经授权的用户从OS处着手访问数据库。

(4) 网络管理:由于大多数DBMS都允许用户通过网络进行远程访问,所以网络软件内部的安全性是很重要的。

(5) DBMS实现:DBMS的安全机制的职责是检查用户的身份是否合法及使用数据库的权限是否正确。

实现数据库系统安全具体要考虑很多方面的问题,例如以下问题。

(1) 法律、道德伦理及社会问题:例如,请求者对其请求的数据的权利是否合法?

(2) 政策问题:例如,拥有系统的组织单位如何授予使用者对数据的存取权限?

(3) 可操作性问题:有关的安全性政策、策略与方案如何落实到系统实现?例如,若使用口令或密码,如何防止密码本身的泄露?若可以授权,如何防止被授权者再授权给不应被

授权的人?

(4) 设施有效性问题:例如,系统所在地的控制保护、硬/软件设备管理的安全特性等是否合适?

这些问题不属于本书讨论的范畴,我们只考虑数据库系统本身。如果要实现数据库安全,DBMS 必须提供下列支持。

(1) 安全策略说明:即安全性说明语言,例如支持授权的 SQL 语言。

(2) 安全策略管理:即安全约束目录的存储结构、存取控制方法和维护机制,例如自主存取控制方法和强制存取控制方法。

(3) 安全性检查:执行"授权"及其检验,认可"他能做他想做的事情吗"?

(4) 用户识别:即标识和确认用户,确定"他就是他说的那个人吗"?

现代 DBMS 一般采用"自主"(discretionary)和"强制"(mandatory)两种存取控制方法来解决安全性问题。在自主存取控制方法中,每个用户对各个数据对象被授予不同的存取权力(authority)或特权(privilege),哪些用户对哪些数据对象有哪些存取权力都按存取控制方案执行,但并不完全固定。在强制存取控制方法中,所有的数据对象被标定一个密级,所有的用户也被授予一个许可证级别。对于任一数据对象,凡具有相应许可证级别的用户就可以存取,否则不能。

5.2 数据库安全性控制

在一般计算机系统中,安全措施是一级级层层设置的,其安全控制模型如图 5-1 所示。

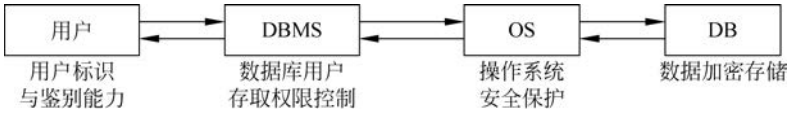


图 5-1 计算机系统的安全模型

(1) 当用户进入计算机系统时,系统首先根据输入的用户标识(例如用户名)进行身份的鉴定,只有合法的用户才准许进入系统。

(2) 对已进入计算机系统的用户,DBMS 还要进行存取控制,只允许用户在所授予的权限之内进行合法的操作。

(3) DBMS 是建立在操作系统之上的,安全的操作系统是数据库安全的前提。操作系统应能保证数据库中的数据必须由 DBMS 访问,而不允许用户越过 DBMS,直接通过操作系统或其他方式访问。

DBMS 与操作系统在安全上的关系可用一个现实生活中与安全有关的实例来形象地说明。2005 年在某市发生了一起特大虫草盗窃案,盗贼通过租用店铺,从店铺的沙发下秘密地挖掘了一条长 39 米的地道,通往街对面的一家虫草行库房,盗走了价值千万元的虫草。虫草行库房周围的物理防护坚固,但盗贼绕过了这些防护,从库房地面这个薄弱环节盗走虫草。

(4) 数据最后通过加密的方式存储到数据库中,即便非法者得到了已加密的数据,也无法识别数据内容。

本书对于操作系统这一级的安全措施不进行讨论,只讨论与数据库有关的用户标识和

鉴别、存取控制、视图、数据加密和审计等安全技术。

5.2.1 用户标识与鉴别

实现数据库的安全性包含两方面的工作：一是用户的标识与确认，即用什么来标识一个用户，又怎样去识别他；二是授权及其验证，即每个用户对各种数据对象的存取权力的表示和检查。这里只讨论第 1 个方面。

那么如何识别一个用户？常用的方法有以下 3 种。

(1) 用户的个人特征识别：例如用户的声音、指纹、签名等。

(2) 用户的特有东西识别：例如用户的磁卡、钥匙等。

(3) 用户的自定义识别：例如用户设置口令、密码和一组预定的问答等。

1. 用户的个人特征识别

使用每个人所具有的个人特征，如声音、指纹、签名等来识别用户是当前最有效的方法，但是有以下两个问题必须解决。

(1) 专门设备：要能准确地记录、存储和存取这些个人特征。

(2) 识别算法：要能较准确地识别出每个人的声音、指纹或签名。这里关键问题是要让“合法者被拒绝”和“非法者被接受”的误判率达到应用环境可接受的程度。百分之百正确(即误判率为零)几乎是不可能的。

另外，其实代价也不得不考虑，这不仅是经济上的代价，还包括识别算法执行的时间、空间代价。它影响整个安全子系统的代价/性能比。

2. 用户的特有东西识别

让每一用户持有一个他特有的物件，例如磁卡、钥匙等。在识别时，将其插入一个“阅读器”，它读取其面上的磁条中的信息。该方法是目前一些安全系统中较常用的一种方法，但用在数据库系统中要考虑以下两点。

(1) 需要专门的阅读装置。

(2) 要求有从阅读器抽取信息及与 DBMS 接口的软件。

该方法的优点是比个人特征识别更简单、有效，代价/性能比更好；缺点是用户容易忘记带磁卡或钥匙等，也可能丢失甚至被别人窃取。

3. 用户的自定义识别

使用只有用户自己知道的定义内容来识别用户是最常用的一种方法，一般用口令或密码，有时用只有用户自己能给出正确答案的一组问题，有时还可以两者兼用。

在使用这类方法时要注意以下几点。

(1) 标识的有效性：口令、密码或问题答案要尽可能准确地标识每一个用户。

(2) 内容的简易性：口令或密码要长短适中，问答过程不要太烦琐。

(3) 本身的安全性：为了防止口令、密码或问题答案的泄露或失窃，应经常改变。

实现这种方法需要专门的软件来进行用户名或用户 ID 及其口令的登记、维护与检验等，但它不需要专门的硬件设备，较之以上的方法这是其优点。其主要的缺点是口令、密码或问题答案容易被人窃取，因此还可以用更复杂的方法。例如，每个用户都预先约定好一个计算过程或函数，在鉴别用户身份时，系统提供一个随机数，用户根据自己预先约定的计算过程或函数进行计算，而系统根据用户的计算结果是否正确进一步鉴定用户身份。

例如,让用户记住一个表达式 T=XY+2X,系统告诉用户 X=1、Y=2,如果用户回答的是 T=4,则证明了该用户的身份是合法的。当然,这是一个简单的例子,在实际使用中还可以设计复杂的表达式,以使安全性更好。

5.2.2 存取控制策略

数据库安全性所关心的主要是 DBMS 的存取控制策略。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,同时令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策略来实现。

存取控制策略主要包括以下两部分。

1. 定义用户权限,并将用户权限登记到数据字典中

用户对某一数据对象的操作权力称为权限。某个用户应该具有何种权限是个管理问题和政策问题,而不是技术问题。DBMS 的功能就是保证这些决定的执行。为此,DBMS 系统必须提供适当的语言来定义用户权限,这些定义经过编译后存放在数据字典中,被称为安全规则或授权规则。

2. 合法权限检查

每当用户发出存取数据库的操作请求后,DBMS 查找数据字典,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限检查策略一起组成了 DBMS 的安全子系统。

当前,大多数 DBMS 所采取的存取控制策略主要有两种,即自主存取控制和强制存取控制。其中,自主存取控制的使用更为普遍。下面分别简要说明这两种方法。

(1) 自主存取控制(DAC):在自主存取控制方法中,用户对于不同的数据库对象有不同的存取权限,不同的用户对同一对象也有不同的权限,而且用户还可将其拥有的存取权限转授给其他用户。因此,自主存取控制非常灵活。

(2) 强制存取控制(MAC):在强制存取控制方法中,每一个数据库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,强制存取控制相对比较严格。

5.2.3 自主存取控制

用户使用数据库的方式称为“授权”(Authorization)。权限有两种,即访问数据的权限和修改数据库结构的权限。

访问数据的权限有 4 个。

(1) 读(Select)权限:允许用户读数据,但不能修改数据。

(2) 插入(Insert)权限:允许用户插入新的数据,但不能修改数据。

(3) 修改(Update)权限:允许用户修改数据,但不能删除数据。

(4) 删除(Delete)权限:允许用户删除数据。

根据需要,可以授予用户上述权限中的一个或多个,也可以不授予上述任何一个权限。

修改数据库结构的权限也有 4 个。

(1) 索引(Index)权限:允许用户创建和删除索引。

(2) 资源(Resource)权限:允许用户创建新的关系。

(3) 修改(Alteration)权限：允许用户在关系结构中加入或删除属性。

(4) 撤销(Drop)权限：允许用户撤销关系。

自主存取控制方式是通过授权和取消来实现的。下面介绍自主存取控制的权限类型，包括角色(Role)权限和数据库对象权限及各自的授权和取消方法。

1. 权限类型

自主存取控制的权限类型分为两种，即角色权限和数据库对象权限。

(1) 角色权限：给角色授权，并为用户分配角色，用户的权限为其角色权限之和。角色权限由 DBA 授予。

(2) 数据库对象权限：不同的数据库对象，可提供给用户不同的操作。该权限由 DBA 或该对象的拥有者(Owner)授予用户。

2. 角色的授权与取消

授权命令的语法如下：

```
GRANT <角色类型>[,<角色类型>] TO <用户> [IDENTIFIED BY <口令>]
<角色类型>:: = Connect|Resource|DBA
```

其中，Connect 表示该用户可连接到 DBMS；Resource 表示用户可访问数据库资源；DBA 表示该用户为数据库管理员；IDENTIFIED BY 用于为用户设置一个初始口令。

取消命令的语法如下：

```
REVOKE <角色管理>[,<角色管理>] FROM <用户>
```

3. 数据库对象的授权与取消

授权命令的语法如下：

```
GRANT <权限> ON <表名> TO <用户>[,<用户>]
    [WITH GRANT OPTION]
<权限>:: = ALL PRIVILEGES|SELECT|INSERT|DELETE|UPDATE[(<列名>[,<列名>])]
```

其中，WITH GRANT OPTION 表示得到授权的用户，可将其获得的权限转授给其他用户；ALL PRIVILEGES 表示所有的操作权限。

取消命令的语法如下：

```
REVOKE <权限> ON <表名> FROM <用户>[,<用户>]
```

说明：数据库对象除了表以外，还有其他对象，例如视图等，但由于表的授权最具典型意义，而且表的授权也最复杂，所以此处只以表的授权为例来说明数据库对象的授权语法，其他对象的授权语法与之类似，只是在权限上不同。

5.2.4 强制存取控制

自主存取控制能够通过授权机制有效地控制对敏感数据的存取，但它存在一个漏洞——一些别有用心的用户可以欺骗一个授权用户，采用一定的手段来获取敏感数据。例如，领导 Manager 是客户单 Customer 关系的物主，他将“读”权限授予用户 A，且 A 不能再将该权限转授他人，其目的是让 A 审查客户信息，看有无错误。现在 A 自己另外创建一个

新关系 A_customer,然后将自 Customer 读取的数据写入(即复制到)A_customer。这样,A 是 A_customer 的物主,他可以做任何事情,包括再将其权限转授给任何其他用户。

存在这种漏洞的根源在于,自主存取控制机制仅以授权将用户(主体)与被存取数据对象(客体)关联,通过控制权限实现安全要求,对用户和数据对象本身未做任何安全性标注。强制存取控制就能处理自主存取控制的这种漏洞。

强制存取控制方法的基本思想在于为每个数据对象(文件、记录或字段等)赋予一定的密级,级别从高到低为绝密级(Top Secret,TS)、机密级(Secret,S)、可信级(Confidential,C)、公用级(Public,P)。每个用户也具有相应的级别,称为许可证级别。密级和许可证级别都是严格有序的,例如,绝密>机密>可信>公用。

在系统运行时,采用以下两条简单规则:

(1) 用户 i 只能查看比它级别低或同级的数据。

(2) 用户 i 只能修改和它同级的数据。

强制存取控制是对数据本身进行密级标记,无论数据如何复制,标记与数据都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。

强制存取控制的优点是系统能执行“信息流控制”。前面介绍的授权方法,允许凡有权查看保密数据的用户就可以把这种数据复制到非保密的文件中,造成无权用户也可以接触保密的数据。强制存取控制可以避免这种非法的信息流动。

注意:这种方法在通用数据库系统中不是十分有用,它只在某些专用系统中有用,例如军事部门或政府部门。

5.3 视图机制

视图可以作为一种安全机制。通过视图,用户只能查看和修改他们所能看到的数据,其他数据库或关系既不可见也不可以访问。如果某一用户想要访问视图的结果集,必须授予其访问权限。

例如,假定李平老师具有检索和增/删/改“数据库”课程成绩信息的所有权限,学生王莎只能检索该科所有同学成绩的信息。那么可以先建立“数据库”课程成绩的视图 score_db,然后在视图上进一步定义存取权限。

(1) 建立视图 score_db。

```
CREATE  VIEW  score_db
AS
SELECT  *
FROM  score
WHERE  cname = '数据库';
```

(2) 为用户授予操作视图的权限。

```
GRANT  SELECT
ON  score_db
TO 王莎;
```

```
GRANT  ALL PRIVILEGES
ON  score_db
TO 李平;
```

5.4 安全级别与审计跟踪

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(安全策略方面),不是全部。为了使 DBMS 达到一定的安全级别,还需要在其他方面提供相应的支持。例如,按照 TCSEC/TDI 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。

5.4.1 安全级别**

美国国防部根据军用计算机系统的安全需要,于 1985 年制定了《可信计算机系统安全评估标准》(简称 TCSEC);1991 年,美国国家安全局的国家计算机安全中心发布 TCSEC 的可信数据库系统解释(简称 TDI),这就形成了最早的信息安全及数据库安全评估体系。TCSEC/TDI 将系统安全性分为 4 等 7 级,依次是 D——最小保护、C(包括 C1、C2)——自主保护、B(包括 B1、B2、B3)——强制保护、A(包括 A1)——验证保护,按系统可靠或可信程度逐渐增高。下面分别简单介绍。

(1) D 级:最低安全级别。将一切不符合更高标准的系统归于 D 级,例如,DOS 就是操作系统中安全标准为 D 的典型例子。

(2) C1 级:实现数据的所有权与使用权的分离,进行自主存取控制,保护或限制用户权限的传播。

(3) C2 级:提供受控的存取保护,即将 C1 级的自主存取控制进一步细化,通过身份注册、审计和资源隔离来支持“责任”说明。

(4) B1 级:标记安全保护,即对每一客体和主体分别标以一定的密级和安全证等级,实施强制存取控制以及审计等安全机制。

(5) B2 级:建立安全策略的形式化模型,并能识别和消除隐通道。

(6) B3 级:提供审计和系统恢复过程,且指定安全管理员(通常是 DBA)。

(7) A1 级:验证设计,在提供 B3 级保护的同时给出系统的形式化设计说明和验证,以确信各安全保护真正实现。即安全机制是可靠的,且足够支持对指定的安全策略给出严格的数学证明。

5.4.2 审计跟踪

由于任何系统的安全保护措施都不是完美无缺的,蓄意盗取、破坏数据的人总会想方设法打破控制。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,称为审计跟踪。DBA 可以利用审计跟踪信息,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,为分析攻击者线索提供依据。一般情况下,将审计跟踪和数据库日志记录结合起来会达到更好的安全审计效果。

DBMS 的审计主要分为语句审计、权限审计、模式对象审计和资源审计。其中,语句审

计是指监视一个或多个特定用户或者所有用户提交的数据库操作语句(即 SQL 语句);权限审计是指监视一个或多个特定用户或所有用户使用的系统权限;模式对象审计是指监视一个模式中在一个或多个对象上发生的行为;资源审计是指监视分配给每个用户的系统资源。

审计机制应该至少记录用户标识和认证、客体的存取、授权用户进行并影响系统安全的操作,以及其他与安全相关的事件。对于每个记录的事件,在审计记录中需要包括事件时间、时间类型、用户、事件数据和事件的成功/失败情况。对于标识和认证事件,必须记录事件源的终端 ID 和源地址等;对于访问和改变对象的事件,则需要记录对象的名称。

审计通常是很费时间和空间的,所以 DBMS 往往将其作为可选特征,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。

5.5 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的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(称为明文)变换为不可直接识别的格式(称为密文),从而使得不知道解密算法的人无法获知数据的内容。

加密方法主要有两种,一种是替换方法,另一种是转换方法。

(1) 替换加密法:这种方法是制定一种规则,将明文中的每个或每组字符替换成密文中的一个或一组字符。其缺点是使用得多了,窃密者可以从多次搜集的密文中发现其中的规律,破解加密方法。

(2) 转换加密法:这种方法不隐藏原来明文的字符,而是将字符重新排序。例如,加密方首先选择一个用数字表示的密钥,写成一行,然后把明文逐行写在数字下,再按照密钥中数字指示的顺序将原文重新抄写,就形成密文。

下面是一个示例。

① 密钥:6852491703。

② 明文:张三偷走了李四的钱包。

③ 密文:李的了三走钱张四包偷。

单独使用这两种方法的任意一种都是不安全的,但是将这两种方法结合起来就能提供相当高的安全程度。采用这种结合方法的例子是美国于 1977 年制定的官方加密标准——数据加密标准(DES)。

有关 DES 密钥加密技术及密钥管理问题在这里不再讨论。

目前有些数据库产品提供了数据加密例行程序,可根据用户的要求自动对存储和传输的数据进行加密处理。另一些数据库产品虽然本身未提供加密程序,但提供了接口,允许用户用其他厂商的加密程序对数据加密。

由于数据加密与解密也是比较费时的操作,而且数据加密与解密程序会占用大量的系统资源,所以数据加密功能通常也作为可选特征,允许用户自由选择,只对高度机密的数据加密。

5.6 统计数据库的安全性

有一类数据库称为“统计数据库”，例如人口调查数据库，它包含大量的记录，但其目的只是向公众提供统计、汇总信息，而不是提供单个记录的内容。也就是说，查询的仅仅是某些记录的统计值，例如求记录数、和、平均值等。在统计数据库中，虽然不允许用户查询单个记录的信息，但是用户可以通过处理足够多的汇总信息分析出单个记录的信息，这就给统计数据库的安全性带来严重的威胁。

统计数据库存在着提供隐通道的机会，因为可能从被允许查询的结果推导出受保护的信息。例如，设某单位有关于职工工资的一个统计数据库，它允许对职工的平均工资、各部门的最高工资等进行统计查询，但不允许查询关于单个职工工资的信息。

现在，假设职工刘五想知道职工张三的工资数目，他可以做以下工作。

(1) 用 SELECT 语句查询刘五自己和其他 n－1 个人(例如 30 岁的男职工)的工资总额 A。

(2) 用 SELECT 语句查找张三和上述同样 n－1 个人的工资总和 B。

随后，刘五可以很方便地通过下列算式求得张三的工资数：

B－A＋“刘五自己的工资数”

这样，刘五就窃取到了张三的工资数目。这两个查询都是允许的，统计数据库应防止上述问题的发生。刘五成功的关键在于他用的两个查询有很多相同的数据对象。为了防止这种情况发生，系统应对用户查询得到的记录个数加以控制。

在统计数据库中，对查询应做下列限制：

(1) 查询查到的记录个数至少是 n。

(2) 查询查到的记录的“交”数目最多是 m。

系统可以调整 n 和 m 的值，使得用户很难在统计数据库中获取其他个别记录的信息，但要做到完全杜绝是不可能的。我们应限制用户计算和、个数、平均值的能力。如果一个破坏者只知道他自己的数据，那么已经证明，他至少要花 1＋(n－2)/m 次查询才有可能获取其他个别记录的信息。因此，系统应限制用户查询的次数在 1＋(n－2)/m 次以内。但是这个方法还不能防止两个破坏者联手查询导致数据的泄露。

保证数据库安全性的另一个方法是“数据污染”，也就是在回答查询时提供一些偏离正确值的数据，以免数据泄露。当然，这个偏离要在不破坏统计数据的前提下进行。此时系统应该在准确性和安全性之间做出权衡。当安全性遭到威胁时，只能降低准确性的标准。

不管用什么样的安全策略与技术，都不能绝对保险，总存在旁通之途。所以好的安全机制不应是杜绝安全性问题，而是使企图破坏安全的手段不能实现或代价很高，这才是数据库安全机制的总体设计目标。

5.7 MySQL 的安全设置

MySQL 安全设置用于实现“正确的人”能够“正确地访问”“正确的数据库资源”。MySQL 通过两个模块实现数据库资源的安全访问控制，即身份认证模块和权限验证模块。

其中,身份认证模块用于实现数据库用户在某台登录主机的身份认证,只有在合法的登录主机通过身份认证的数据库用户才能成功连接到 MySQL 服务器,继而向 MySQL 服务器发送 MySQL 命令或者 SQL 语句;权限验证模块用于验证 MySQL 账户是否有权执行该 MySQL 命令或 SQL 语句,确保"数据库资源"被正确地访问或者执行。

在成功安装 MySQL 后,默认情况下,MySQL 会自动创建 root(超级管理员)账户,管理 MySQL 服务器的全部资源。但出于安全及工作方面的考虑,仅靠 root 账户不足以管理 MySQL 服务器的诸多资源,root 账户不得不创建多个 MySQL 账户共同管理各个数据库资源。下面依次介绍 MySQL 对用户账号、权限和角色的管理。

5.7.1 用户管理

MySQL 用户包括 root 用户和普通用户,root 用户是超级管理员,拥有所有的权限;而普通用户只拥有创建该用户时赋予它的权限。

在 MySQL 数据库中,为了防止非授权用户对数据库进行存取,DBA 可以创建登录用户、修改用户信息和删除用户。

1. 创建登录用户

在 MySQL 数据库中,为了防止非授权用户对数据库进行存取,DBA 可以创建登录用户。创建用户主要通过 CREATE USER 语句实现,在使用该语句创建用户时不赋予任何权限,还需要通过 GRANT 语句分配权限。CREATE USER 语句的语法形式如下:

```
CREATE USER 用户 [IDENTIFIED BY [PASSWORD] '密码']
[,用户 [IDENTIFIED BY [PASSWORD] '密码']]…;
```

说明:

(1) 用户的格式:用户名@主机名。其中,主机名指定了创建的用户使用 MySQL 连接的主机。另外,"%"表示一组主机,localhost 表示本地主机。

(2) IDENTIFIED BY 子句指定创建用户时的密码。如果密码是一个普通的字符串,则不需要使用 PASSWORD 关键字。

【例 5-1】 创建用户 tempuser,其口令为 temp。

```
CREATE USER tempuser@localhost IDENTIFIED BY 'temp';
```

创建的新用户的详细信息自动保存在系统数据库 mysql 的 user 表中,执行如下 SQL 语句,可查看数据库服务器的用户信息。

```
USE mysql;
SELECT * FROM user;
```

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv
localhost	root	Y	Y	Y	Y	Y
localhost	mysql.session	N	N	N	N	N
localhost	mysql.sys	N	N	N	N	N
localhost	tempuser	N	N	N	N	N

2. 修改用户密码

在创建用户后,允许对其进行修改,可以使用 SET PASSWORD 语句修改用户的登录

密码，其语法形式如下：

```
SET PASSWORD FOR 用户 = '新密码';
```

【例 5-2】 修改用户 tempuser 的密码为 root。

```
SET PASSWORD FOR tempuser@localhost = 'root';
```

【例 5-3】 修改超级用户 root 的密码为 root。

```
SET PASSWORD FOR root@localhost = 'root';
```

3. 修改用户名

修改已存在的用户名可以使用 RENAME USER 语句，其语法形式如下：

```
RENAME USER 旧用户名 TO 新用户名[,旧用户名 TO 新用户名][,…];
```

【例 5-4】 修改普通用户 tempuser 的用户名为 temp_U。

```
RENAME USER tempuser@localhost TO temp_U@localhost;

USE mysql;
SELECT * FROM user WHERE user = 'temp_U' and host = 'localhost';
```

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv
localhost	temp_U	N	N	N	N	N
NULL	NULL	NULL	NULL	NULL	NULL	NULL

4. 删除用户

使用 DROP USER 语句可删除一个或多个 MySQL 用户，并取消其权限。其语法形式如下：

```
DROP USER 用户[,…];
```

【例 5-5】 删除用户 temp_U。

```
DROP USER temp_U@localhost;

USE mysql;
SELECT * FROM user WHERE user = 'temp_U' and host = 'localhost';
```

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv
NULL	NULL	NULL	NULL	NULL	NULL	NULL

5.7.2 权限管理

权限管理主要是对登录到 MySQL 服务器的数据库用户进行权限验证。所有用户的权限都存储在 MySQL 的权限表中。合理的权限管理能够保证数据库系统的安全，不合理的权限设置会给数据库系统带来危害。

权限管理主要包括两个内容，即授予权限和撤销权限。

1. 授予权限

创建了用户,并不意味着用户就可以对数据库随心所欲地进行操作,用户对数据进行任何操作都需要具有相应的操作权限。

在 MySQL 中,针对不同的数据库资源,可以将权限分为 5 类,即 MySQL 字段级别权限、MySQL 表级别权限、MySQL 存储程序级别权限、MySQL 数据库级别权限和 MySQL 服务器管理员级别权限。

下面依次介绍每种权限级别具有的权限类型及为用户授予权限的语句。

1) 授予 MySQL 字段级别权限

在 MySQL 中,使用 GRANT 语句授予权限。授予 MySQL 字段级别权限的语法形式如下:

```
GRANT 权限名称(列名[,列名,…])[,权限名称(列名[,列名,…]),…]
 ON TABLE 数据库名.表名或视图名
 TO 用户[,用户,…]
[WITH GRAND OPTION];
```

拥有 MySQL 字段级别权限的用户可以对指定数据库的指定表中指定列执行所授予的权限操作。

系统数据库 MySQL 的系统表 Column_priv 中记录了用户 MySQL 字段级别权限的验证信息。Cloumn_priv 权限表提供的权限名称较少,如表 5-1 所示。MySQL 字段级别的用户仅允许对字段进行查询、插入及修改。

表 5-1 Column_priv 权限表提供的权限名称

权限名称	权限类型	说明
SELECT	Column_priv	查询数据库表中的记录
INSERT		向数据库表中插入记录
UPDATE		修改数据库表中的记录
REFERENCES		暂未使用
ALL PRIVILEGES	以上所有权限类型的和	Grant_priv 权限类型除外
USAGE	没有任何权限类型	仅用于登录

【例 5-6】 授予 MySQL 字段级别权限示例。

```
USE mysql;

CREATE USER column_user@localhost IDENTIFIED BY 'password';

GRANT SELECT(ename,sal,empno),UPDATE(sal)
   ON TABLE scott1.emp
   TO column_user@localhost
   WITH GRANT OPTION;

SELECT * FROM Column_priv;
```

Host	Db	User	Table_name	Column_name	Timestamp	Column_priv
localhost	scott1	column_user	emp	empno	0000-00-00 00:00:00	Select
localhost	scott1	column_user	emp	sal	0000-00-00 00:00:00	Select,Update
localhost	scott1	column_user	emp	ename	0000-00-00 00:00:00	Select

以 Column_user 用户连接 MySQL 服务器，执行如下语句。

```
SELECT ename,sal FROM scott1.emp;
```

ename	sal
SMITH	800.00
ALLEN	1600.00
WARD	1250.00
JONES	2975.00
MARTIN	1250.00
BLAKE	2850.00
CLARK	2450.00
SCOTT	3000.00

```
SELECT job,comm FROM scott1.emp;
```

2）授予 MySQL 表级别权限

授予 MySQL 表级别权限的语法形式如下：

```
GRANT 权限名称[, 权限名称,…]
 ON TABLE 数据库名.表名或视图名
 TO 用户[,用户,…]
[WITH GRAND OPTION];
```

拥有 MySQL 表级别权限的用户可以对指定数据库中的指定表执行所授予的权限操作。

系统数据库 MySQL 的系统表 Table_priv 中记录了用户 MySQL 表级别权限的验证信息。Table_priv 权限表提供的权限名称如表 5-2 所示。

表 5-2　Table_priv 权限表提供的权限名称

权限名称	权限类型	说　明
SELECT	Table_priv	查询数据库表中的记录
INSERT		向数据库表中插入记录
UPDATE		修改数据库表中的记录
DELETE		删除数据库表中的记录
CREATE		创建数据库表，但不允许创建索引和视图
DROP		删除数据库表以及视图的定义，但不能删除索引
GRANT		将自己的权限分享给其他 MySQL 用户
REFERENCES		暂未使用
INDEX		创建或删除索引
ALTER		执行 ALTER TABLE 修改表结构
CREATE VIEW		执行 CREATE VIEW 创建视图。在创建视图时，还需要持有基表的 SELECT 权限
SHOW VIEW		执行 SHOW CREATE VIEW 查看视图的定义
TRIGGER		创建、执行及删除触发器
ALL PRIVILEGES	以上所有权限类型的和	Grant_priv 权限类型除外
USAGE	没有任何权限类型	仅用于登录

【例 5-7】 授予 MySQL 表级别权限示例。

```
USE mysql;

CREATE USER table_user@localhost IDENTIFIED BY 'password';

GRANT ALTER,SELECT,INSERT(empno,ename)
 ON TABLE scott1.emp
 TO table_user@localhost;

SELECT * FROM tables_priv
 WHERE host = 'localhost' AND user = 'table_user';
```

Host	Db	User	Table_name	Grantor	Timestamp	Table_priv	Column_priv
localhost	scott1	table_user	emp	root@localhost	0000-00-00 00:00:00	Select,Alter	Insert

以 table_user 用户连接 MySQL 服务器,执行如下语句。

```
DESC scott1.emp;
```

Field	Type	Null	Key	Default	Extra
empno	decimal(4,0)	NO	PRI	NULL	
ename	varchar(10)	YES		NULL	
job	varchar(9)	YES		NULL	

```
ALTER TABLE scott1.emp
 MODIFY COLUMN empno INT;

DESC scott1.emp;
```

Field	Type	Null	Key	Default	Extra
empno	int(11)	NO	PRI	NULL	
ename	varchar(10)	YES		NULL	
job	varchar(9)	YES		NULL	

3) 授予 MySQL 存储程序级别权限

授予 MySQL 存储程序级别权限的语法形式如下:

```
GRANT  权限名称[,权限名称,…]
 ON  FUNCTION|PROCEDURE 数据库名.函数名|数据库名.存储过程名
 TO  用户[,用户,…]
[WITH GRAND OPTION];
```

拥有 MySQL 存储程序级别权限的用户可以对指定数据库中的存储过程或者存储函数执行所授予的权限操作。

系统数据库 MySQL 的系统表 Proc_priv 中记录了用户 MySQL 存储程序级别权限的验证信息。Proc_priv 权限表提供的权限名称如表 5-3 所示。

表 5-3 Proc_priv 权限表提供的权限名称

权限名称	权限类型	说明
GRANT	Proc_priv	将自己的权限分享给其他 MySQL 用户
EXECUTE		执行存储过程或函数
ALTER ROUTINE		修改、删除存储过程或函数
ALL PRIVILEGES	以上所有权限类型的和	Grant_priv 权限类型除外
USAGE	没有任何权限类型	仅用于登录

【例 5-8】 授予 MySQL 存储程序级别权限示例。

```
USE mysql;

CREATE USER proc_user@localhost IDENTIFIED BY 'password';

GRANT  EXECUTE  ON PROCEDURE scott1.emp_p
 TO  proc_user@localhost;

GRANT ALTER ROUTINE,EXECUTE ON FUNCTION scott1.sum_fn
 TO proc_user@localhost;

SELECT * FROM Proc_priv;
```

Host	Db	User	Routine_name	Routine_type	Grantor	Proc_priv	Timestamp
localhost	scott1	proc user	emp p	PROCEDURE	root@localhost	Execute	0000-00-00 00:00:00
localhost	scott1	proc user	sum fn	FUNCTION	root@localhost	Execute,Alter Routine	0000-00-00 00:00:00

以 proc_user 用户连接 MySQL 服务器,执行如下语句。

```
CALL scott1.emp_p;
```

v_ename	v_job
SCOTT	ANALYST

```
SELECT scott1.sum_fn(3);            ##sum_fn(n)返回 1～n 的和
```

scott1.sum_fn(3)
6

```
DROP FUNCTION scott1.sum_fn;
```

#	Time	Action	Message
1	10:56:44	DROP FUNCTION scott1.sum_fn	0 row(s) affected

4) 授予 MySQL 数据库级别权限

授予 MySQL 数据库级别权限的语法形式如下:

```
GRANT 权限名称[,权限名称,…]
 ON  数据库名.*
 TO 用户[,用户,…]
[WITH GRAND OPTION];
```

拥有 MySQL 数据库级别权限的用户可以对指定数据库中的对象执行所授予的权限操作。

系统数据库 MySQL 的系统表 db 中记录了用户 MySQL 数据库级别权限的验证信息。db 权限表提供的权限名称如表 5-4 所示。

表 5-4 db 权限表提供的权限名称

权限名称	权限类型	说明
SELECT	Select_priv	查询数据库表中的记录
INSERT	Insert_priv	向数据库表中插入记录
UPDATE	Update_priv	修改数据库表中的记录
DELETE	Delete_priv	删除数据库表中的记录
CREATE	Create_priv	创建数据库或者数据库表,但不允许创建索引和视图

续表

权限名称	权限类型	说　明
DROP	Drop_priv	删除数据库、数据库表以及视图的定义,但不能删除索引
WITH GRANT OPTION	Grant_priv	将自己的权限分享给其他 MySQL 账户
REFERENCES	References_priv	暂未使用
INDEX	Index_priv	创建或者删除索引
ALTER	Alter_priv	执行 ALTER TABLE 修改表结构。在修改表名时,还需要持有旧表的 DROP 权限以及新表的 CREATE、INSERT 权限
CREATE TEMPORARY TABLES	Create_tmp_table_priv	执行 CREATE TEMPORARY TABLES 命令创建临时表
LOCK TABLES	Lock_tables_priv	执行 LOCK TABLES 命令显式地加锁,执行 UNLOCK TABLES 命令显式地解锁
EXECUTE	Execute_priv	执行存储过程或者函数
CREATE VIEW	Create_view_priv	执行 CREATE VIEW 创建视图。在创建视图时,还需要持有基表的 SELECT 权限
SHOW VIEW	Show_view_priv	执行 SHOW CREATE VIEW 查看视图的定义
CREATE ROUTINE	Create_routine_priv	创建存储过程或者函数
ALTER ROUTINE	Alter_routine_priv	修改、删除存储过程或者函数
EVENT	Event_priv	创建、修改、删除以及查看事件
TRIGGER	Trigger_priv	创建、执行以及删除触发器
ALL PRIVILEGES	以上所有权限类型的和	Grant_priv 权限类型除外
USAGE	没有任何权限	仅用于登录

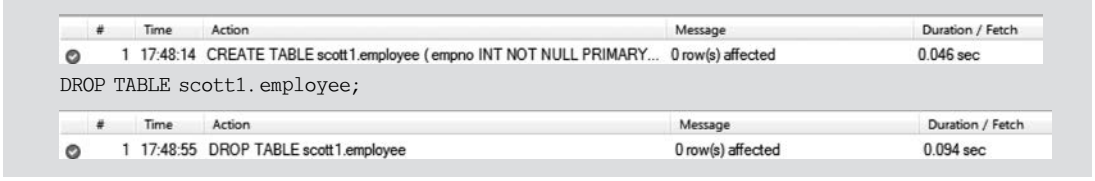
【例 5-9】 授予 MySQL 数据库级别权限示例。

```
USE mysql;

CREATE USER database_user@localhost IDENTIFIED BY 'password';

GRANT CREATE,SELECT,DROP
 ON scott1. *
 TO database_user@localhost;

SELECT * FROM db
 WHERE host = 'localhost' AND db = 'scott1';
```

Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Grant_priv	Re
localhost	scott1	database user	Y	N	N	N	Y	Y	N	N
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

以 database_user 用户连接 MySQL 服务器,执行如下语句。

```
CREATE TABLE scott1.employee
( empno INT NOT NULL PRIMARY KEY,
 ename VARCHAR(10)
);
```



5）授予 MySQL 服务器管理员级别权限

授予 MySQL 服务器管理员级别权限的语法形式如下：

```
GRANT 权限名称[,权限名称,…]
 ON *.*
 TO 用户[,用户,…]
[WITH GRAND OPTION];
```

拥有 MySQL 服务器管理员级别权限的用户可以对服务器上所有数据库中的所有对象执行所授予的权限操作。

系统数据库 MySQL 的系统表 user 中记录了用户 MySQL 服务器管理员级别权限的验证信息。user 权限表提供的权限名称不仅包含表 5-4 中数据库级别的所有权限类型，而且包含对整个 MySQL 服务器的管理权限，其权限名称如表 5-5 所示。

表 5-5　MySQL 服务器管理员的“管理”权限

权限名称	权限类型	说　明
RELOAD	Reload_priv	执行 FLUSH HOSTS、FLUSH LOGS、FLUSH PRIVILEGES、FLUSH STATUS、FLUSH TABLES、FLUSH THREADS、REFRESH 以及 RELOAD 等刷新命令
SHUTDOWN	Shutdown_priv	执行 mysqladmin 的 SHUTDOWN 命令，停止服务器的运行
PROCESS	Process_priv	执行 SHOW PROCESSLIST 显示 MySQL 服务器上正在执行的线程，还可以执行 KILL 命令杀死该线程
FILE	File_priv	执行 LOAD DATA INFILE、SELECT … INTO OUTFILE 命令或者执行 file()函数
SHOW DATABASES	Show_db_priv	执行 SHOW DATABASES
SUPER	Super_priv	执行 CHANGE MASTER TO、KILL、PURGE BINARY LOGS、SET GLOBAL 以及 mysqladmin 的 DEBUG 等命令
REPLICATION SLAVE	Repl_slave_priv	该权限应该授予通过从服务器连接主服务器的 MySQL 账户，没有该权限，从服务器将不能获取主服务器的更新
REPLICATION CLIENT	Repl_client_priv	执行 SHOW MASTER STATUS、SHOW SLAVE STATUS 命令
CREATE USER	Create_user_priv	执行 CREATE USER、DROP USER、RENAME USER、REVOKE ALL PRIVILEGES 命令
CREATE TABLESPACE	Create_tablespace_priv	创建、修改以及删除表空间或者日志文件组

【例 5-10】 授予 MySQL 服务器管理员级别权限示例。

```
USE mysql;

CREATE USER server_user@localhost IDENTIFIED BY 'password';

GRANT ALL PRIVILEGES
 ON *.*
 TO server_user@localhost;

SELECT * FROM user
 WHERE host = 'localhost' AND user = 'server_user';
```

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv	Process_priv	File_
localhost	server_user	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

以 server_user 用户连接 MySQL 服务器,执行如下语句。

```
CREATE DATABASE student;
```

#	Time	Action	Message	Duration / Fetch
1	19:10:32	CREATE DATABASE student	1 row(s) affected	0.000 sec

2. 撤销权限

撤销权限就是取消已经赋予用户的某些权限。撤销用户不必要的权限在一定程度上可以保证数据的安全性。在撤销权限后,用户账户的记录将从系统表 db、tables_priv、columns_priv 和 procs_priv 中删除,但是用户账户记录仍然在 user 表中保存。

使用 REVOKE 语句实现撤销权限,其语法格式有两种:一种是撤销用户的所有权限;另一种是撤销用户指定的权限。

1) 撤销所有权限

撤销用户所有权限的 REVOKE 语句的语法形式如下:

```
REVOKE ALL PRIVILEGES,GRANT OPTION
 FROM 用户[,用户,…];
```

【例 5-11】 撤销例 5-6 中用户 column_user@localhost 的所有权限。

```
SELECT * FROM mysql.columns_priv;
```

Host	Db	User	Table_name	Column_name	Timestamp	Column_priv
localhost	scott1	column_user	emp	empno	0000-00-00 00:00:00	Select
localhost	scott1	column_user	emp	sal	0000-00-00 00:00:00	Select,Update
localhost	scott1	column_user	emp	ename	0000-00-00 00:00:00	Select
localhost	scott1	table_user	emp	ename	0000-00-00 00:00:00	Insert
localhost	scott1	table_user	emp	empno	0000-00-00 00:00:00	Insert

```
REVOKE ALL PRIVILEGES,GRANT OPTION
 FROM column_user@localhost;

SELECT * FROM mysql.columns_priv;
```

Host	Db	User	Table_name	Column_name	Timestamp	Column_priv
localhost	scott1	table_user	emp	ename	0000-00-00 00:00:00	Insert
localhost	scott1	table_user	emp	empno	0000-00-00 00:00:00	Insert

```
SELECT * FROM mysql.user
```

```
WHERE host = 'localhost' AND user = 'column_user';
```

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdov
localhost	column user	N	N	N	N	N	N	N	N

2）撤销指定权限

撤销用户指定权限的 REVOKE 语句的语法形式如下：

```
REVOKE 权限名称[(列名[,列名,…])][,权限名称[(列名[,列名,…])],…]
 ON   *.*|数据库名.*|数据库名.表名或视图名
 FROM 用户[,用户,…];
```

【例 5-12】 撤销例 5-9 中用户 database_user@localhost 的 CREATE 和 DROP 权限。

```
SELECT * FROM db
 WHERE host = 'localhost' AND user = 'database_user';
```

Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Grant_priv
localhost	scott1	database user	Y	N	N	N	Y	Y	N

```
REVOKE CREATE,DROP
 ON scott1.*
 FROM database_user@localhost;

SELECT * FROM db
 WHERE host = 'localhost' AND user = 'database_user';
```

Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Grant_priv
localhost	scott1	database user	Y	N	N	N	N	N	N

5.7.3 角色管理

从前面的介绍可以看出，MySQL 的权限设置是非常复杂的，权限的类型也非常多，这就为 DBA 有效地管理数据库权限带来了困难。另外，数据库的用户通常有几十个、几百个，甚至成千上万个。如果管理员为每个用户授予或者撤销相应的权限，则这个工作量是非常大的。为了简化权限管理，MySQL 提供了角色的概念。

角色是具有名称的一组相关权限的组合，即将不同的权限集合在一起就形成了角色。可以使用角色为用户授权，同样也可以撤销角色。由于角色集合了多种权限，所以当为用户授予角色时，相当于为用户授予了多种权限。这样就避免了向用户逐一授权，从而简化了用户权限的管理。

下面以项目开发中常见的场景为例，应用程序需要读/写权限、运维人员需要完全访问数据库、部分开发人员需要读取权限、部分开发人员需要写权限，如果向多个用户授予相同的权限集，则应按创建角色→授予角色权限→授予用户角色的步骤来实现。

1. 创建角色

创建角色的语法形式如下：

```
CREATE  ROLE  角色;
```

说明：角色格式：'角色名'@'主机名'。

【例 5-13】 分别在本地主机上创建应用程序角色 app、运维人员角色 ops、开发人员读角色 dev_read、开发人员写角色 dev_write。

```
USE mysql;

CREATE ROLE 'app'@'localhost','ops'@'localhost',
 'dev_read'@'localhost','dev_write'@'localhost';

SELECT * FROM USER
 WHERE host = 'localhost'
    AND user IN('app','ops','dev_read','dev_write');
```

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv
localhost	app	N	N	N	N
localhost	dev_read	N	N	N	N
localhost	dev_write	N	N	N	N
localhost	ops	N	N	N	N

2. 授予角色权限

授予角色权限的语法格式类似于授予用户权限,只需将 GRANT 语句中 TO 后面的用户改为角色即可。

【例 5-14】 分别授予角色 app 数据读/写权限、角色 ops 访问数据库权限、角色 dev_read 读取权限、角色 dev_write 写权限。

```
GRANT SELECT,INSERT,UPDATE,DELETE
 ON SCOTT1.* TO 'app'@'localhost';

GRANT ALL PRIVILEGES
 ON SCOTT1.* TO 'ops'@'localhost';

GRANT SELECT
 ON SCOTT1.* TO 'dev_read'@'localhost';

GRANT INSERT,UPDATE,DELETE
 ON SCOTT1.* TO 'dev_write'@'localhost';

SELECT * FROM db
 WHERE host = 'localhost' AND
 user IN('app','ops','dev_read','dev_write');
```

Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Grant_priv	Re
localhost	scott1	app	Y	Y	Y	Y	N	N	N	N
localhost	scott1	dev_read	Y	N	N	N	N	N	N	N
localhost	scott1	dev_write	N	Y	Y	Y	N	N	N	N
localhost	scott1	ops	Y	Y	Y	Y	Y	Y	N	Y

3. 授予用户角色

授予用户角色的语法形式如下:

```
GRANT 角色[,角色,…] TO 用户[,用户,…];
```

【例 5-15】 分别将角色授予新用户 app01、ops01、dev01、dev02、dev03。

```
#创建新的用户账号
CREATE USER 'app01'@'localhost' IDENTIFIED BY '000000';

CREATE USER 'ops01'@'localhost' IDENTIFIED BY '000000';

CREATE USER 'dev01'@'localhost' IDENTIFIED BY '000000';

CREATE USER 'dev02'@'localhost' IDENTIFIED BY '000000';

CREATE USER 'dev03'@'localhost' IDENTIFIED BY '000000';

#给用户账号分配角色
GRANT 'app'@'localhost' TO 'app01'@'localhost';

GRANT 'ops'@'localhost' TO 'ops01'@'localhost';

GRANT 'dev_read'@'localhost' TO 'dev01'@'localhost';

GRANT 'dev_read'@'localhost','dev_write'@'localhost'
  TO 'dev02'@'localhost','dev03'@'localhost';

#验证角色是否正确分配,可使用 SHOW GRANTS 语句
SHOW GRANTS FOR 'dev01'@'localhost' USING 'dev_read'@'localhost';
```

Grants for dev01@localhost
GRANT USAGE ON *.* TO `dev01`@`localhost`
GRANT SELECT ON `scott1`.* TO `dev01`@`localhost`
GRANT `dev_read`@`localhost` TO `dev01`@`localhost`

注意：用户在使用角色权限前必须先激活角色，设置语句如下：

```
SET GLOBAL activate_all_roles_on_login = ON;
```

4. 撤销用户角色

撤销用户角色的语法形式如下：

```
REVOKE 角色[,角色,…] FROM 用户[,用户,…];
```

【例 5-16】 撤销用户 app01 的角色 app。

```
REVOKE 'app'@'localhost' FROM 'app01'@'localhost';

SHOW GRANTS FOR 'app01'@'localhost' USING 'app'@'localhost';
```

#	Time	Action	Message	Duration / Fetch
1	18:36:49	SHOW GRANTS FOR 'app01'@'localhost' USING 'app'@'localhost'	Error Code: 3530. 'app'@'localhost' is not granted to 'app01'@'localhost'	0.000 sec

5. 删除角色

删除角色的语法形式如下：

```
DROP ROLE 角色[,角色,…];
```

【例 5-17】 删除角色 app 和 ops。

```
DROP ROLE 'app'@'localhost','ops'@'localhost';
```

#	Time	Action	Message	Duration / Fetch
1	17:37:57	DROP ROLE 'app'@'localhost','ops'@'localhost'	0 row(s) affected	0.078 sec

5.8 小　　结

数据库的安全指的是保护数据,以防止非法使用造成的数据泄密、更改和破坏。数据库的安全管理涉及用户的访问权限问题,通过设置用户标识、用户的存取控制权限、定义视图、审计、数据加密技术等来保证数据不被非法使用。

实现数据库系统安全性的技术和方法有多种,最重要的是存取控制技术、视图技术和审计技术。自主存取控制功能一般是通过 SQL 的 GRANT 语句和 REVOKE 语句来实现的。对数据库模式的授权则由 DBA 在创建用户时通过 CREATE USER 语句实现。数据库角色是一组权限的集合。使用角色来管理数据库权限可以简化授权的过程。在 SQL 中用 CREATE ROLE 语句创建角色,用 GRANT 语句给角色授权。

习　题　5

一、选择题

1. 对用户访问数据库的权限加以限定是为了保护数据库的(　　)。

A. 安全性　　B. 完整性　　C. 一致性　　D. 并发性

2. 数据库的(　　)是指数据的正确性和相容性。

A. 完整性　　B. 安全性　　C. 并发控制　　D. 系统恢复

3. 在数据库系统中,定义用户可以对哪些数据对象进行的操作被称为(　　)。

A. 审计　　B. 授权　　C. 定义　　D. 视图

4. 某高校 5 个系的学生信息存放在同一个基本表中,采取(　　)的措施可使各系的管理员只能读取本系学生的信息。

A. 建立各系的列级视图,并将对该视图的读权限赋予该系的管理员

B. 建立各系的行级视图,并将对该视图的读权限赋予该系的管理员

C. 将学生信息表的部分列的读权限赋予各系的管理员

D. 将修改学生信息表的权限赋予各系的管理员

5. 下列关于 SQL 对象的操作权限的描述正确的是(　　)。

A. 权限的种类分为 INSERT、DELETE 和 UPDATE 3 种

B. 权限只能用于实表,不能用于视图

C. 使用 REVOKE 语句撤销权限

D. 使用 COMMIT 语句赋予权限

二、填空题

1. 对数据库_____性的保护是指要采取措施,防止库中数据被非法访问、修改,甚至被恶意破坏。

2. 安全性控制的一般方法有_____、_____、_____、_____和_____5种。

3. 在MySQL中,可以使用_____语句为数据库添加用户。

4. _____是具有名称的一组相关权限的组合。

5. 授予权限和撤销权限的命令依次是_____和_____。

三、操作题

1. 表DEPT的结构:DEPT(deptno,dname,loc)。

请用SQL的GRANT和REVOKE语句(加上视图机制)完成以下授权定义或存取控制功能:

(1) 创建本地用户账号test_user,其口令为test。

(2) 向用户test_user授予对象"SCOTT.DEPT"的SELECT权限。

(3) 向用户test_user授予对象"SCOTT.DEPT"的INSERT、DELETE权限,仅对loc字段具有更新的权限。

(4) 用户test_user具有对DEPT表的所有权限,并具有给其他用户授权的权力。

(5) 撤销用户test_user的所有权限。

(6) 用户test_user只有查看"10"号部门的权力,不能查看其他部门的信息。

(7) 建立角色ROLE1,并授予对SCOTT数据库的所有操作权限。

(8) 将ROLE1角色的权力授予用户test_user。

(9) 撤销用户test_user的ROLE1角色。

(10) 删除角色ROLE1。

2. 阅读下列说明,回答问题(1)~(3)。

说明:某工厂的仓库管理数据库的部分关系模式如下。

仓库(仓库号,面积,负责人,电话)

原材料(编号,名称,数量,储备量,仓库号)

要求一种原料只能存放在同一仓库中。"仓库"和"原材料"的关系实例分别如表5-6和表5-7所示。

表5-6 "仓库"关系

仓库号	面积	负责人	电话
01	500	李劲松	87654121
02	300	陈东明	87654122
03	300	郑爽	87654123
04	400	刘春来	87654125

表5-7 "原材料"关系

编号	名称	数量	储备量	仓库号
1001	小麦	100	50	01
2001	玉米	50	30	01
1002	大豆	20	10	02
2002	花生	30	50	02
3001	菜油	60	20	03

(1) 根据上述说明,用 SQL 定义“原材料”和“仓库”的关系模式如下。

```
CREATE  TABLE 仓库(仓库号  CHAR(4),  面积  INT,
        负责人  CHAR(8),
        电话  CHAR(8), _____①_____);        //主键定义

CREATE  TABLE 原材料(编号  CHAR(4)  _____②_____, //主键定义
        名称  CHAR(6),数量  INT,储备量  INT,
        仓库号 _____③_____,
        _____④_____);                        //外键定义
```

(2) 将下面的 SQL 语句补充完整,完成“查询存放原材料数量最多的仓库号”的功能。

```
SELECT  仓库号
FROM  _____①_____
_____②_____;
```

(3) 将下面的 SQL 语句补充完整,完成“01”号仓库所存储的原材料信息只能由管理员李劲松来维护,而采购员李强能够查询所有原材料的库存信息的功能。

```
CREATE  VIEW  raws_in_wh01
AS
  SELECT_____①_____  FROM  原材料  WHERE  仓库号='01';

GRANT_____②_____  ON  _____③_____  TO  李劲松;
GRANT_____④_____  ON  _____⑤_____  TO  李强;
```