第3章 数据库的安全性控制

数据库的一大特点是数据可以共享，但数据共享必然带来数据库的安全性问题，因为数据库系统中的数据共享不能是无条件的共享。数据库中数据的共享是在DBMS统一的严格的控制之下的共享，即只允许有合法使用权限的用户访问允许其存取的数据。因此，数据库系统的安全保护措施是否有效是数据库系统主要的性能指标之一。

数据库的安全性是指保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。对任何企业组织来说，数据的安全性最为重要。安全性主要是指允许那些具有相应的数据访问权限的用户能够登录DBMS并访问数据，以及对数据库对象实施各种权限范围内的操作，但是要拒绝所有的非授权用户的非法操作。因此，安全性管理与用户管理是密不可分的。

SQL Server提供了内置的安全性和数据保护。安全性管理建立在认证(Authentication)和访问许可(Permission)两种机制上。认证是指确定登录SQL Server的登录账号和密码是否正确，以此来验证其是否具有连接SQL Server的权限。但是，通过认证阶段并不代表能够访问SQL Server中的数据。只有在获取访问数据库的权限之后，才能够对服务器上的数据库进行权限许可下的各种操作，主要是针对数据库对象，如表、视图、存储过程等。这种访问数据库权限的设置是通过用户账号来实现的，同时在SQL Server中角色作为用户组的代替物大大地简化了安全性管理。所以，在SQL Server的安全模型中主要包括以下部分：

- SQL Server登录；
- 数据库用户；
- 权限；
- 角色。

3.1 SQL Server 身份验证

3.1.1 实验目的

本实验的目的是通过实验加深对数据安全性的理解，并掌握 SQL Server 中有关用户登录认证及管理方法。

3.1.2 原理解析

1. 身份验证模式

SQL Server 能在两种安全模式下运行：Windows 身份验证模式和混合模式。Windows 身份验证模式会启用 Windows 身份验证并禁用 SQL Server 身份验证。混合模式会同时启用 Windows 身份验证和 SQL Server 身份验证。Windows 身份验证始终可用，并且无法禁用。

1）Windows 身份验证模式

SQL Server 数据库系统通常运行在 Windows 平台上，而操作系统本身就具备安全管理、验证用户合法性的能力。所以，Windows 身份验证模式正是利用这一用户安全性和账号管理的机制，允许 SQL Server 可以使用 Windows 的用户名和口令，在该模式下只要通过 Windows 的认证就可连接到 SQL Server，SQL Server 不需要管理登录数据。

Windows 身份验证模式与 SQL Server 身份验证模式相比有许多优点，原因在于 Windows 身份验证模式集成了 Windows 平台的安全系统，并且 Windows 安全管理具有众多特征，如安全合法性、口令加密、对密码最小长度进行限制等，所以当用户试图登录 SQL Server 时，从 Windows 平台的网络安全属性中获取登录用户的账号与密码，并使用 Windows 平台验证账号和密码的机制来检验登录的合法性，从而提高了 SQL Server 的安全性。

当使用 Windows 认证时，一般总是把用户归入一定的 Windows 用户组，当新增加一个登录用户时，也把它归入某一 Windows 用户组，这种方法可以使用户更为方便地加入系统中，并消除了逐一为每个用户进行数据库访问权限设置而带来的不必要的工作量，简化了管理。

2）混合模式（Windows 和 SQL Server 身份验证模式）

在混合模式下，Windows 和 SQL Server 这两种身份验证模式都是可用的。连接 SQL Server 时必须提供登录名和登录密码，这些登录信息均通过 SQL Server 创建并存储在 SQL Server 的系统表 syslogins 中，与 Windows 的登录账号无关。SQL Server 自己执行认证处理，如果输入的登录信息与系统表 syslogins 中的某条记录相匹配，则表明登录成功。当使用 SQL Server 身份验证时，必须为所有 SQL Server 账户设置强密码。

如果在安装过程中选择 Windows 身份验证，则安装程序会为 SQL Server 身份验证创建 sa 账户，但会禁用该账户。如图 3.1.1 所示，对象资源管理器中，“安全”→“登录名”→sa 账户前面有一个叉号。如果稍后更改为混合模式身份验证并要使用 sa 账户，则必须启用该账户（右击 sa 账户，选择“属性”选项，在弹出的对话框中选择“状态”→“登录名”，选择“启用”单选按钮，参见图 3.1.2）。由于 sa 账户广为人知且经常成为恶意用户的攻击目标，因此除非应用程序需要使用 sa 账户，否则请勿启用该账户。切勿为 sa 账户设置空密码或弱密码。通常用户需要另外创建账号，赋予用户使用数据的权力。

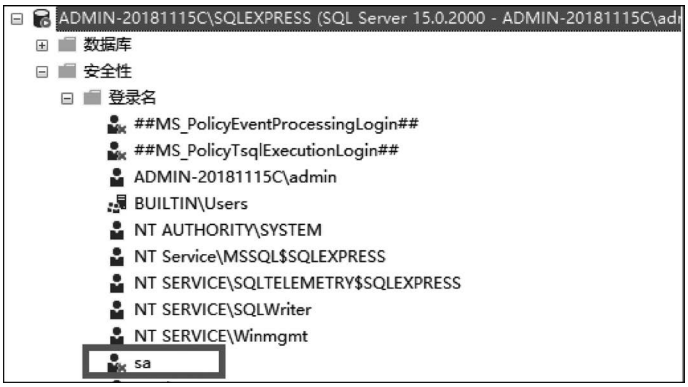


图　3.1.1

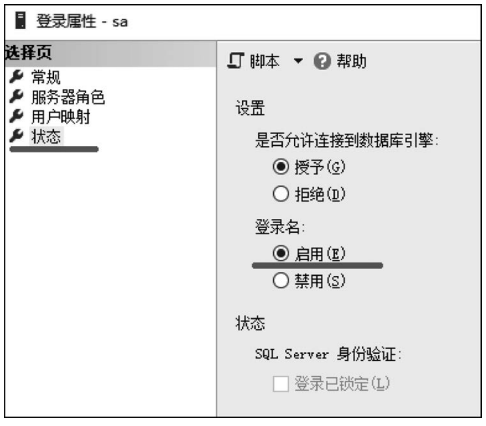


图　3.1.2

若选择“混合模式”，则 SQL Server 通过校验登录名和登录密码的方式进行安全验证。登录名和密码是系统提供的最外层安全保护措施。每次要求进入 SQL Server 时，由系统进行核对，通过鉴定后才提供机器使用权。

2. 数据库用户

数据库用户用来指出哪个人可以访问哪一个数据库。在一个数据库中，用户 ID 唯一标识一个用户，数据的访问权限以及数据库对象的所有关系都是通过用户账号来控制的。用户账号总是基于数据库的，即两个不同数据库中可以有两个相同的用户账号。

在数据库中，数据用户与登录名（登录用户）是两个不同的概念。一个合法的登录账号

只表明该账号通过了 Windows 认证或 SQL Server 认证，但不能表明其可以对数据库数据和数据对象进行某种或某些操作，所以一个登录账号总是与一个或多个数据库用户账号（这些账号必须分别存在相应的数据库中）这样才可以具有访问数据库的权限。例如，登录账号 sa 自动与系统自带的数据库中的数据库用户 dbo 相关联。右击 sa 账号，在弹出的快捷菜单中选择“属性”选项，在弹出的对话框中选择“用户映射”，可以看到右边 sa 映射到每一个数据库上的哪个用户，如图 3.1.3 所示。

图 3.1.3

3. 使用 Transact-SQL 管理 SQL Server 登录

1）管理登录用户及密码

使用 sp_addlogin 创建新的使用 SQL Server 身份验证模式的登录账号，其语法格式为：

```
sp_addlogin [ @loginame = ] 'login'
    [ , [ @passwd = ] 'password' ]
    [ , [ @defdb = ] 'database' ]
    [ , [ @deflanguage = ] 'language' ]
    [ , [ @sid = ] sid ]
    [ , [ @encryptopt = ] 'encryption_option' ]
```

其中，

@loginame：登录名。

@passwd：登录密码。

@defdb：登录时的默认数据库。

@deflanguage：登录时的默认语言。

@sid：安全标识码。

@encryptopt：将密码存储到系统表时是否对其进行加密，参数有三个选项。

- NULL 表示对密码进行加密；
- skip_encryption 表示对密码不加密；
- skip_encryption_old 只在 SQL Server 升级时使用表示旧版本已对密码加密。

可以通过 sp_droplogin 在 SQL Server 中删除登录账号，语法格式为：

```
sp_droplogin [ @loginame = ] 'login'
```

2）创建数据库用户

创建一个新的数据库用户，不仅要输入新创建的数据库用户名称，还要选择一个已经存在的登录账号与这个数据库用户建立映射。也就是说，除了 guest 数据库用户外，其他数据库用户必须与某一登录账号相匹配，所以，当使用系统过程来创建数据库用户时，必须指出其对应的登录账号。

（1）sp_grantdbaccess。系统过程 sp_grantdbaccess 就是被用来为 SQL Server 登录者或 Windows 用户或用户组建立一个相匹配的数据用户账号。其语法格式为：

```
sp_grantdbaccess [ @loginame = ] 'login'
[ , [ @name_in_db = ] 'name_in_db' [ OUTPUT ] ]
```

@loginame：表示 SQL Server 登录账号、Windows 用户或用户组。如果使用的是 Windows 用户或用户组，那么必须给出 Windows 主机名称或 Windows 网络域名。登录账号、Windows 用户或用户组必须存在。

@ name_in_db：表示登录账号相匹配的数据库用户账号。该数据库用户账号并不存在于当前数据库中，如果不给出该参数值，则 SQL Server 把登录名作为默认的用户名称。

（2）sp_revokedbaccess。系统过程 sp_revokedbaccess 用来将数据库用户从当前数据库中删除，其相匹配的登录者就无法使用该数据库。sp_revokedbaccess 的语法格式为：

```
sp_revokedbaccess [ @name_in_db = ] 'name'.
```

其中，@name_in_db 的含义参看 sp_revokedbaccess 的语法格式。

3.1.3　实验内容

（1）在 SQL Server Management Studio 中设置 SQL Server 的安全身份验证模式为混合模式。

（2）在 SQL Server 中建立一个名为 Li 的登录用户和 LiDB 的数据库用户。

（3）演示在 SQL Server 中取消 Li 这个用户。

3.1.4　实验步骤

（1）在 SQL Server Management Studio 中将所属的 SQL Server 服务器设置为 Windows 和 SQL Server 混合安全身份验证模式。其操作如下：在 SQL Server Management Studio（以后简称 SSMS）窗口中左部的对象资源管理器窗口中展开服务器组，右击需要设置的 SQL 服务器，在弹出的快捷菜单中选择“属性”选项，则弹出 SQL Server 服务器属性对话框，如图 3.1.4 所示。

在 SQL Server 服务器属性对话框中，选择“安全性”选择页，在“服务器身份验证”栏选择“SQL Server 和 Windows 身份验证模式”单选按钮。

（2）在 SQL Server Management Studio 中为自己建立一个服务器用户 Li 和数据库用户 LiDB。可以采用图形界面和命令行两种方法来创建。

方法 1：图形界面的方法。在对象资源管理器中展开服务器组，展开服务器，单击“安全

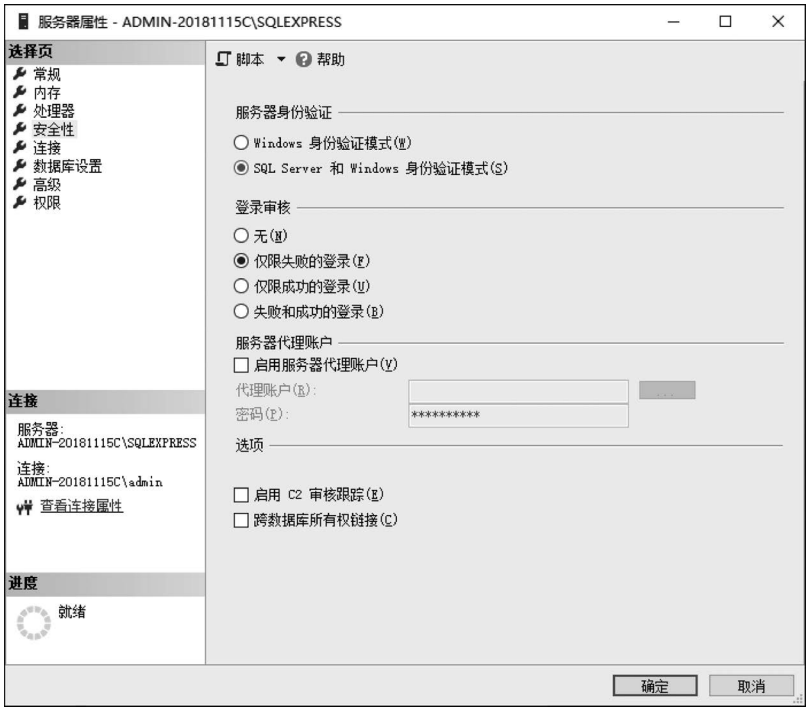


图 3.1.4

性”文件夹左侧的“＋”号，右击“登录名”，在弹出的快捷菜单中选择“新建登录名”选项，则弹出“登录名-新建”对话框，如图 3.1.5 所示。

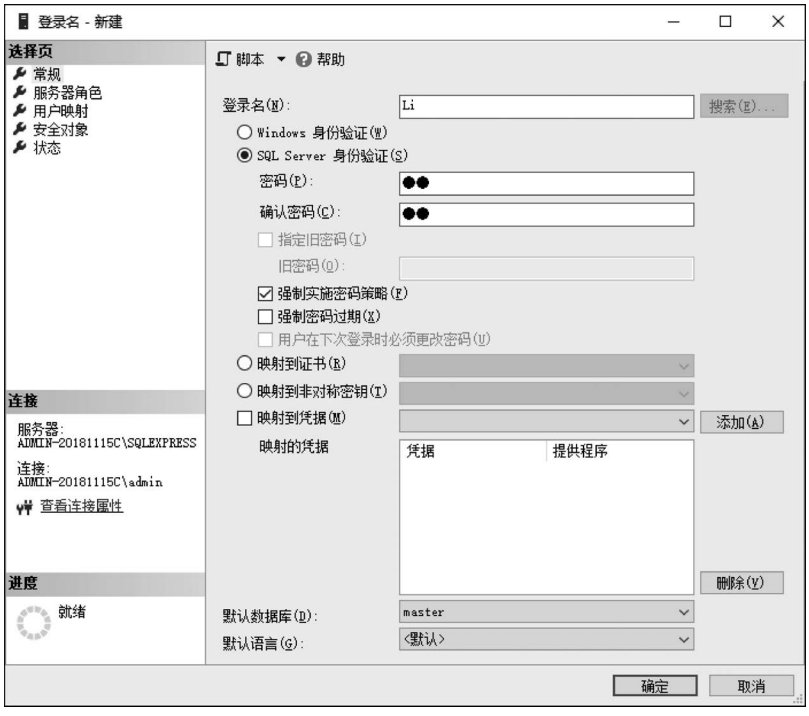


图 3.1.5

在“登录名-新建”对话框中可以通过“常规”“服务器角色”“用户映射”“安全对象”“状态”5 个选择页进行设置。

① 在“常规”选择页中，输入用户名(Li)，选择 SQL Server 身份验证，输入用户口令。

② 在“服务器角色”选择页中，需要确定用户所属的服务器角色，这里采用默认值。

③ 在“用户映射”选择页中，可以设置登录账号可访问的数据库，这里选择 school 作为默认数据库。

④ 在“安全对象”选择页中，可对不同类型的安全对象进行安全授予或拒绝(这里采用默认值)。

⑤ “状态”选择页中显示了用户登录的状态信息等。如该登录号是否被禁用。

单击“确定”按钮，即完成了创建登录用户的工作。

接下来创建数据库用户 LiDB，这里为了方便演示数据库用户和登录用户的不同，所以登录名和数据库用户名取了不同的名字来进行区分。后续授权的实验中可以发现授权是针对数据库用户名来进行的。如果希望 Li 能访问 school 数据库，则逐层展开对象资源管理器→school 数据库→“安全性”→“用户”，右击“用户”节点，在弹出的快捷菜单中选择“新建用户”选项，弹出“数据库用户-新建”对话框(见图 3.1.6)，输入用户名(数据库用户名)、登录名和默认架构。之后确认，就在 school 数据库中建立了数据库用户 LiDB，且是与登录用户 Li 映射的，也就是说登录用户 Li 一旦访问数据库 school，就自动映射到用户 LiDB，自动拥有 LiDB 的所有权限。默认架构设置为 dbo，默认架构是什么？在 3.5 节中再介绍。

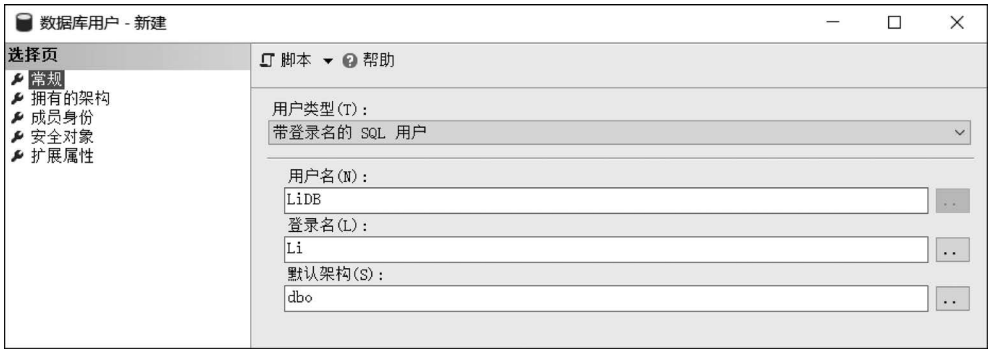


图　3.1.6

方法 2：命令行的方法。以系统管理员或 sa 账号登录 SSMS，单击“新建查询”按钮，在文本编辑框中输入建立登录账号的语句，然后执行即可，如代码 3.1.1 所示。

```
EXEC sp_addlogin 'Li','123456','school','English'
GO
USE school
GO
EXEC sp_grantdbaccess 'Li','LiDB'
```

代码 3.1.1

创建完新用户 LiDB 后需要赋予其成员身份为 db_owner，具体步骤如下。

在对象资源管理器中展开服务器组，展开服务器，单击“数据库”文件夹左侧的“＋”号，

继续展开 school→“安全性”→“用户”，右击用户 LiDB，在弹出的快捷菜单中选择“属性”选项，在弹出的对话框中选择数据库角色成员身份，并勾选 db_owner 复选框完成操作，如图 3.1.7 所示。

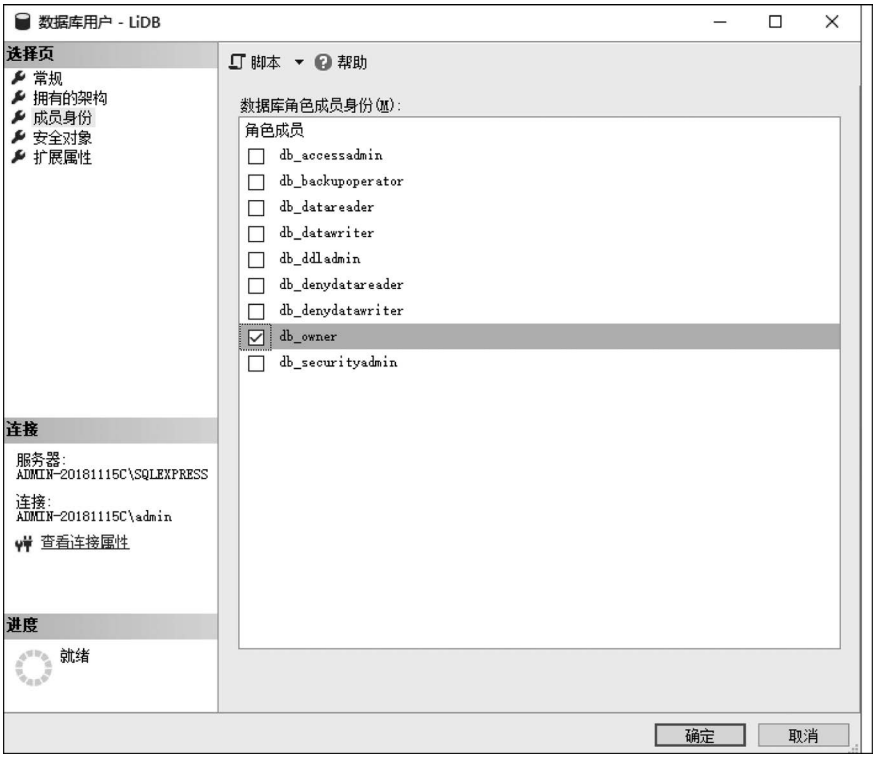


图 3.1.7

这时可以试试是否能用 Li 这个登录名登录服务器，如果希望成功地与服务器建立连接并登录成功，则还要确认一些配置，展开“开始”→“所有程序”→Microsoft SQL Server 2019→“SQL Server 2019 配置管理器”，在弹出的窗体中，找到“SQL Server 网络配置”，把“SQLEXPRESS 的协议”下的 Named Pipes 和 TCP/IP 启用，如图 3.1.8 所示。如果发生了设置的改变，则需要重新启动 SQL Server 或者重启计算机。

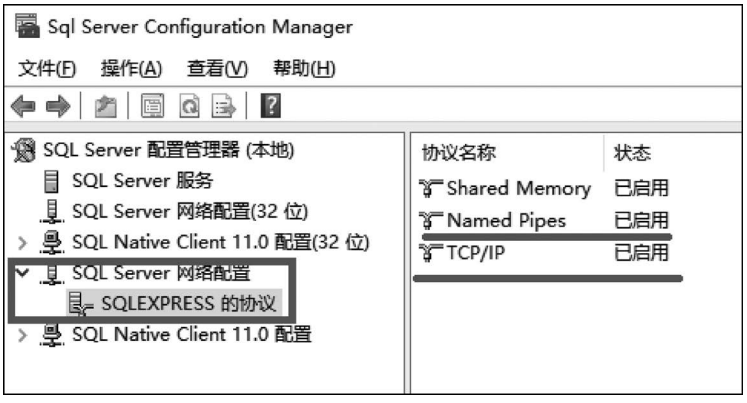


图 3.1.8

再回到对象资源管理器，右击服务器，在弹出的快捷菜单中选择“连接”选项，会出现图 3.1.9 中的界面，身份验证选择 SQL Server 身份验证，输入登录名 Li 和密码 123456，单击“连接”按钮就能使用 SQL Server 身份验证的方式连接到服务器了。

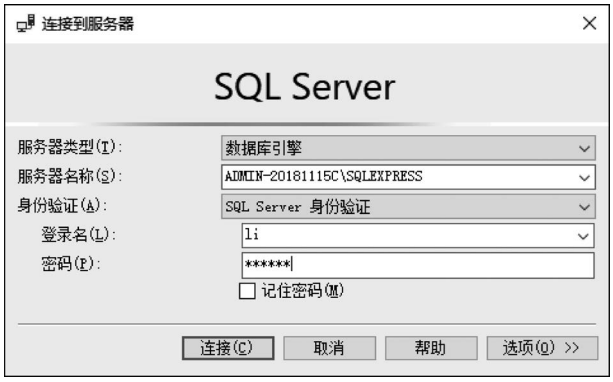


图　3.1.9

参见图 3.1.10 可以看到对象资源管理器中有两个服务器对象，一个是以 admin 结尾，另一个是以 Li 结尾，表示目前已经建立了两个和服务器的连接：一个是以 Windows 验证方式建立的；另一个是混合模式，是登录用户 Li 建立的。

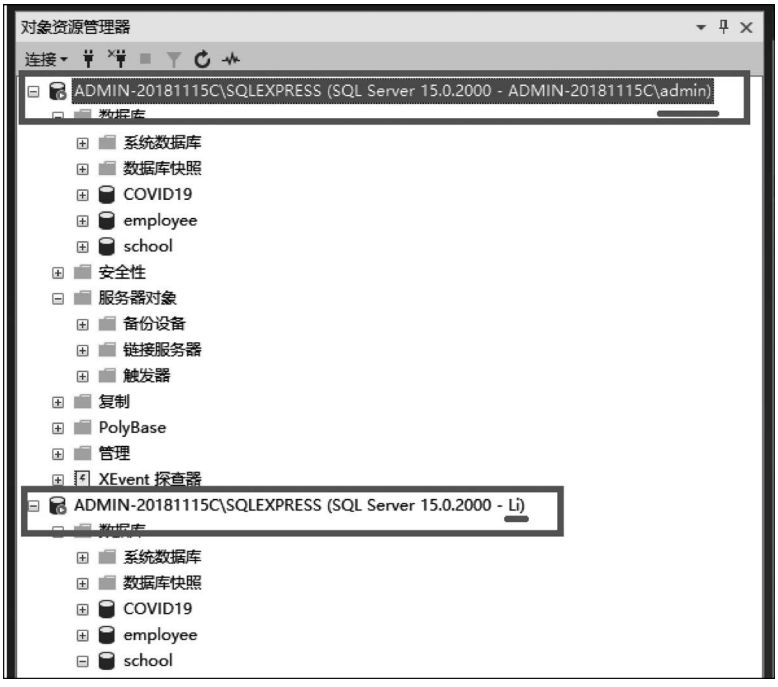


图　3.1.10

单击 admin 连接的服务器对象(见图 3.1.11 中步骤 1)，然后单击“新建查询”按钮(见图 3.1.11 中步骤 2)，这时界面的右下方出现了一个查询对话框，可以看到后缀是 admin(见图 3.1.11 中步骤 3)；同样地，单击 Li 连接的服务器对象，然后单击“新建查询”按钮，又出现了一个查询对话框，后缀是 admin(见图 3.1.11 中步骤 4)。通过这个步骤的观察可以来

判断某个查询是属于哪个连接。如果选择了相应的服务器对象，然后再打开某个 SQL 文件，也会发现同样的规律。例如图 3.1.12 是在不同连接下打开的 SQL 文件的查询窗口界面。可以观察到 3.2.2.sql 是 Li 这个连接中打开的文件，而 3.2.1.sql 则是 admin 这个连接中打开的文件。

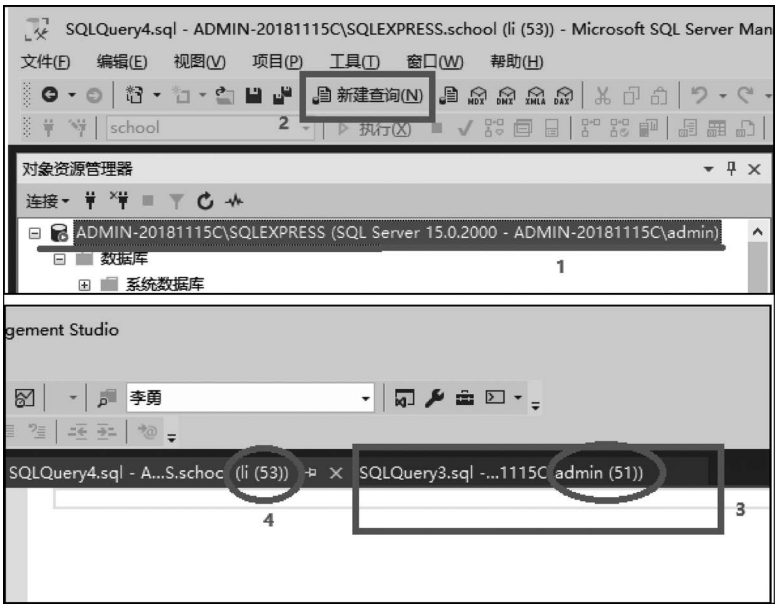


图 3.1.11

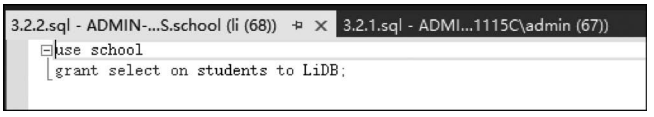


图 3.1.12

(3) 如果想用 SQL 语句删除数据库用户和登录用户，则以系统管理员或 sa 账号登录 SSMS，单击“新建查询”按钮。在文本编辑框中输入取消登录账号语句(由于 school 数据库已经授权给 LiDB，因此要先执行取消其数据库权限的语句)，然后执行即可，如代码 3.1.2 所示。

```
USE school
EXEC sp_revokedbaccess 'LiDB';
EXEC sp_droplogin 'Li';
```

代码 3.1.2

3.1.5 自我实践

(1) 在 school 数据库中创建账号“王二”，密码为 123，并在 school 数据库中创建数据库用户“王 DB”，绑定登录用户“王二”。

(2) 撤销“王二”这个账号。

3.2 自主存取控制

3.2.1　实验目的

通过实验加深对数据库存取控制机制的理解，通过自主存取控制(Discretionary Access Control，DAC)进行权限管理，熟悉 SQL Server 中的角色管理。

3.2.2　原理解析

1. 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机制实现。

某个用户对某类数据具有何种操作权限是个政策问题而不是技术问题。数据库管理系统的功能是保证这些决定的执行。为此 DBMS 必须具有以下功能。

(1) 把授权的决定告知 DBMS，这是由 SQL 的 GRANT 和 REVOKE 语句未完成的。

(2) 把授权的结果存入数据字典。

(3) 当用户提出操作请求时，DBMS 根据授权情况进行检查，以决定是否执行操作请求。

存取控制机制主要包括如下两部分。

(1) 定义用户权限，将用户权限登记到数据字典中。用户权限是指不同的用户对于不同的数据对象允许执行的操作权限。系统必须提供适当的语言定义用户权限，这些定义经过编译后存放在数据字典中，被称作安全规则或授权规则。

用户权限是由数据对象和操作类型两个要素组成的。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。定义存取权限也称为授权(Authorization)。

(2) 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机制一起组成了 DBMS 的安全子系统。

2. 自主存取控制

1) 自主存取方法

相对于强制存取方法，自主存取方法有如下特点。

- 同一用户对于不同的数据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。

• 还可将其拥有的存取权限转授给其他用户。

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 语句实现。

2）检查存取权限

对于获得上机权后又进一步发出存取数据库操作的用户，DBMS 查找数据字典，根据其存取权限对操作的合法性进行检查。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。

3）授权粒度

授权粒度是指可以定义的数据对象的范围，是衡量授权机制是否灵活的一个重要指标。授权定义中数据对象的粒度越细，即可以定义的数据对象的范围越小，授权子系统就越灵活。关系数据库中授权的数据对象粒度从大到小为数据库、表、属性列、元组。

4）自主存取控制的优缺点

优点：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。

缺点：可能存在数据的“无意泄露”，因为这种机制仅仅通过对数据的存取权限来进行安全控制，而数据本身并无安全性标记。一种解决方法是对系统控制下的所有主客体实施强制存取控制策略。

3. SQL Server 中的权限管理

在 SQL Server 中包括两种类型的权限，即对象权限和语句权限。

（1）对象权限总是针对表、视图、存储过程而言，决定了对表、视图、存储过程执行哪些操作（如 UPDATE、DELETE、INSERT、EXECUTE）。如果想要对某一对象进行操作，必须具有相应的操作权限。例如要成功修改表中数据，那么前提条件是已经被授予表的 UPDATE 权限。

不同类型的对象支持不同的操作，例如不能对表对象执行 EXECUTE 类型的操作。各种对象的可执行的操作如表 3.2.1 所示。

表 3.2.1　各种对象的可执行的操作

对　　象	可能的操作
表	SELECT、INSERT、UPDATE、DELETE、REFERENCE
视图	SELECT、UPDATE、INSERT、DELETE
存储过程	EXECUTE
列	SELECT、UPDATE

（2）语句权限主要指是否具有权限来执行某些语句，这些语句通常是一些具有管理性的操作，如创建数据库、表、存储过程等。这种语句（如 CREATE）虽然也包含有操作的对象，但这些对象在执行该语句之前并不存在于数据库中（如创建一个表，在 CREATE TABLE 语句未成功执行前数据库中没有该表），这一类属于语句权限范畴。

在 SQL Server 中使用 GRANT、REVOKE 和 DENY 三种命令来管理权限。

（1）GRANT：用来把权限授予某一用户，以允许执行针对该对象的操作（如 UPDATE、SELECT、DELETE、EXECUTE）或允许其运行某些语句（如 CREATE TABLE、CRETAE DATABASE）。

(2) REVOKE：取消对某一对象或语句的权限(这些权限是经过 GRANT 语句授予的)，不允许执行针对数据库对象的某些操作(如 UPDATE、SELECT、DELETE、EXECUTE)，或不允许其运行某些语句(如 CREATE TABLE、CREATE DATABASE)。

(3) DENY：用来禁止对某一对象或语句的权限，明确禁止某一用户对象执行某些操作(如 UPDATE、SELECT、DELETE、EXECUTE)或运行某些语句(如 CREATE TABLE、CREATE DATABASE)。

4. 角色和 SQL Sever 中的角色管理

(1) 除了给个别用户授予权限，DBMS 可能还提供以下授权功能。

- 给一个角色制定权限，然后把角色赋予用户。
- 将用户加入 DBMS 已经内建的权限组。

注意：*在有的 DBMS 中，角色就是组，或者组就是角色。*

(2) SQL Server 提供了两种数据库角色类型：预定义的数据库角色和用户自定义的数据库角色。

① 预定义的数据库角色。预定义数据库角色是指这些角色所有具有的管理、访问数据库权限已被 SQL Server 预先定义，并且 SQL Server 管理者不能对其所具有的权限进行任何修改。SQL Server 中的每一个数据库中都有一组预定义的数据库角色，在数据库中使用预定义的数据库角色，可以将不同级别的数据库管理工作分给不同的角色，从而很容易实现工作权限的传递。

② 用户自定义的数据库角色。当 DBA 打算为某些数据库用户设置相同的权限，但是这些权限不等同于预定义的数据库角色所具有的权限时，就可以定义新的数据库角色来满足这一要求，从而使这些用户能够在数据库中实现某一特定功能。用户自定义的数据库角色具有以下优点。

- SQL Server 数据库角色可以包含 Windows 用户组或用户。
- 在同一数据库中，用户可以具有多个不同的自定义角色，这种角色的组合是自由的，不仅仅是 Public 与其他某种角色的结合。
- 角色可以进行嵌套，从而在数据库实现不同级别的安全性。

5. 权限的授予和回收

DBMS 中允许用户之间的权限相互授予，如图 3.2.1 所示。

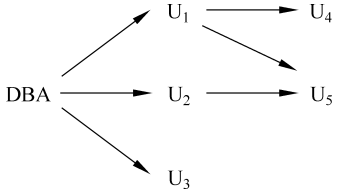


图　3.2.1

权限授予和回收有时候会使得用户的权限混淆不清，但必须牢记一点便可以厘清关系，那就是一个用户拥有权限的充分必要条件是在权限图中从根节点到该用户节点存在一条路径。如图 3.2.2 所示，当 DBA 回收了 U_2 用户的权限之后，但 U_2 通过 U_3 仍然存在路径到达根节点，当 DBA 继续回收了 U_3 用户的权限后，U_2 和 U_3 用户都不存在从 DBA 到它的一条授权路径，那么 DBMS 自动检查后发现，U_2 和 U_3 最终都不具有权限。

一对用户可能企图通过相互授权来破坏回收规则，所以要求授权图中所有边都是某条

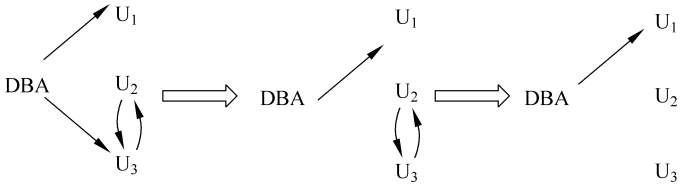


图 3.2.2

从 DBA 开始的路径的一部分，不要形成循环路径。当 DBA 回收了 U_3 的权限时，由于 DBA—U_2—U_3 仍存在一条路径，于是 U_3 仍然具有权限，如图 3.2.3 所示。

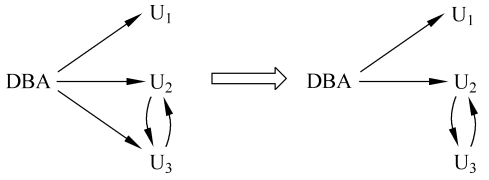


图 3.2.3

3.2.3 实验内容

(1) 分别通过 SSMS 和 SQL 的数据控制功能，设置和管理数据操作权限。对新建用户 Li 授予 school 数据库中 STUDENTS 表的 SELECT 权限。

(2) 通过 SSMS，实现对 SQL Server 的用户和角色管理。具体是创建一个数据库角色 OP_of_students，代表一个可以对 STUDENTS 表进行操作的操作员，对角色的权限进行设置，并将 Li 添加到这个角色中。该实验体现角色应用灵活高效的特点。

3.2.4 实验步骤

(1) 在 SQL Server 中建立一个名为 Li 的登录用户、LiDB 的 school 数据库用户，参见 3.1.4 节(2)。

(2) 使用用户名为 Li，输入用户口令登录到 SSMS(方法参见 3.1.4 节(2))，在 Li 的连接下新建 SQL 查询(或者打开代码文件 3.2.1.sql)。在“查询”的文本编辑器中输入 SQL 语句“SELECT * FROM STUDENTS”。运行后，得到消息“SELECT Permission denied on object 'student', database 'school', schema 'dbo'.”。可见用户 Li 没有对学生表的 SELECT 权限，如代码 3.2.1 所示。

```
SELECT * FROM STUDENTS
```

代码 3.2.1

结果如图 3.2.4 所示。

(3) 将 school 数据库的操作权限赋予数据库用户 LiDB，有两种方法。

方法 1：通过 SSMS 图形界面中提供的菜单。

在 SSMS 窗口中展开服务器，单击“数据库”文件夹左侧的“+”号，单击 school 数据库

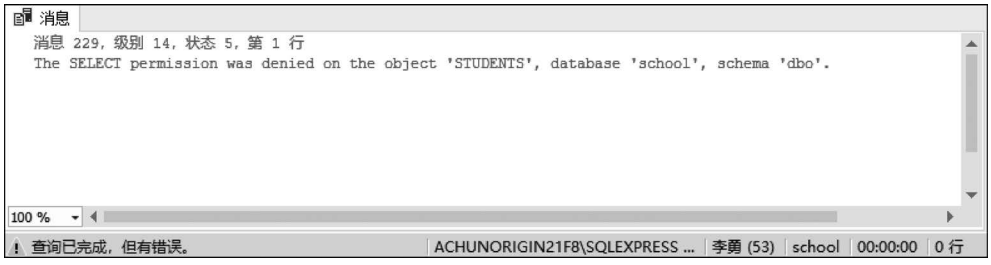


图　3.2.4

文件夹左侧的“＋”号，展开“安全性”→“用户”→LiDB，右击，在弹出的快捷菜单中选择“属性”选项，则弹出“数据库用户-LiDB”对话框，如图 3.2.5 所示（实际的界面和图 3.2.5 还有些区别，右下方的框里是空的）。

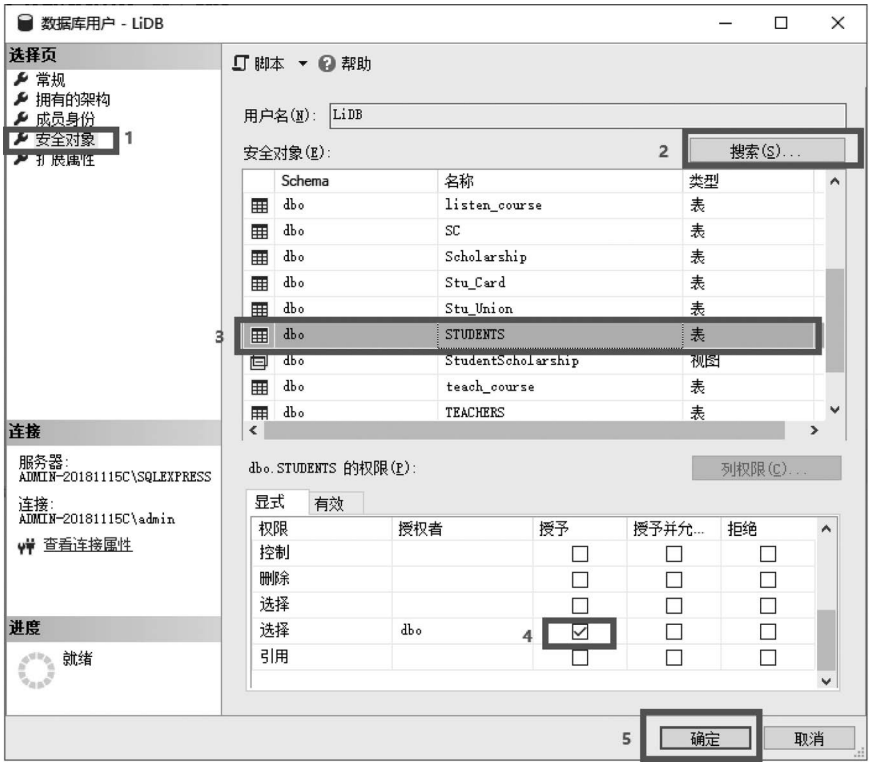


图　3.2.5

选择图 3.2.5 中的“安全对象”选择页，在出现的“安全对象”列表框中，单击“搜索”按钮，弹出如图 3.2.6 所示的对话框，选择“属于该架构的所有对象”单选按钮，“构架名称”选择 dbo，单击“确定”按钮后出现如图 3.2.5 所示的界面。选择表[dbo].[STUDENTS]，对话框的下面是有关数据库用户和角色所对应的权限表。这些权限均以复选框的形式表示。复选框有三种类型：“授予”“授予并允许”“拒绝”。可以对用户或角色的四种对象操作权限（SELECT INSERT UPDATE DELETE EXEC 和 DRI）进行授予或回收等。

在图 3.2.5 中找到 STUDENTS 表，授予 SELECT 权限，即让授予列与 SELECT 行交叉的复选框为“√”即可。

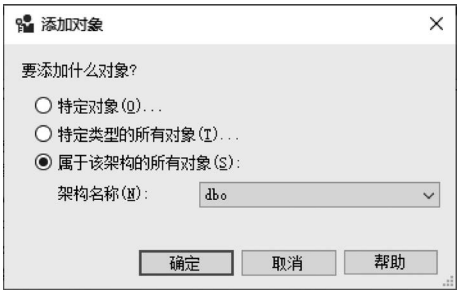


图 3.2.6

方法 2：通过 SQL 的数据控制功能。

对用户 Li 授权，必须是数据库对象拥有者以上用户授予。可以通过以系统管理员或 sa 账号登录进入 SSMS 中，选择 school 数据库，单击“新建查询”按钮，输入授权语句“GRANT SELECT ON STUDENTS TO LiDB;”，然后执行即可，如代码 3.2.2 所示。

```
USE school
GRANT SELECT ON STUDENTS TO LiDB;
```

代码 3.2.2

(4) 启动 SSMS 登录到指定的服务器，展开 school 数据库选中角色图标。展开“安全性”，右击“角色”选项，在弹出的快捷菜单中选择“新建数据库角色”选项，弹出“数据库角色-新建”对话框，如图 3.2.7 所示。

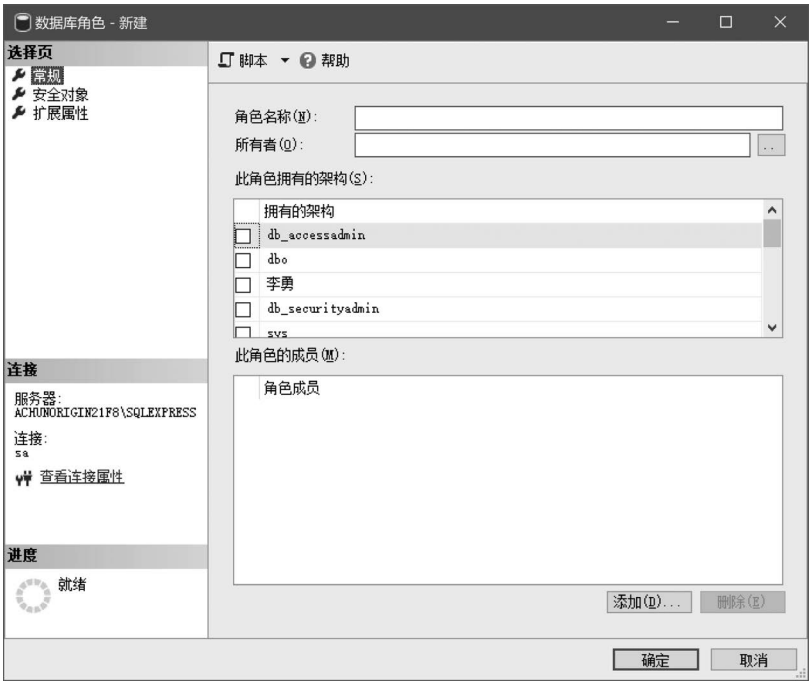


图 3.2.7

在“角色名称”文本框中输入 OP_of_students，表示是对 STUDENTS 表有权限的操作者，在“此角色的成员”列表框中，单击“添加”按钮添加 Li，单击“确定”按钮结束，如图 3.2.8 所示。

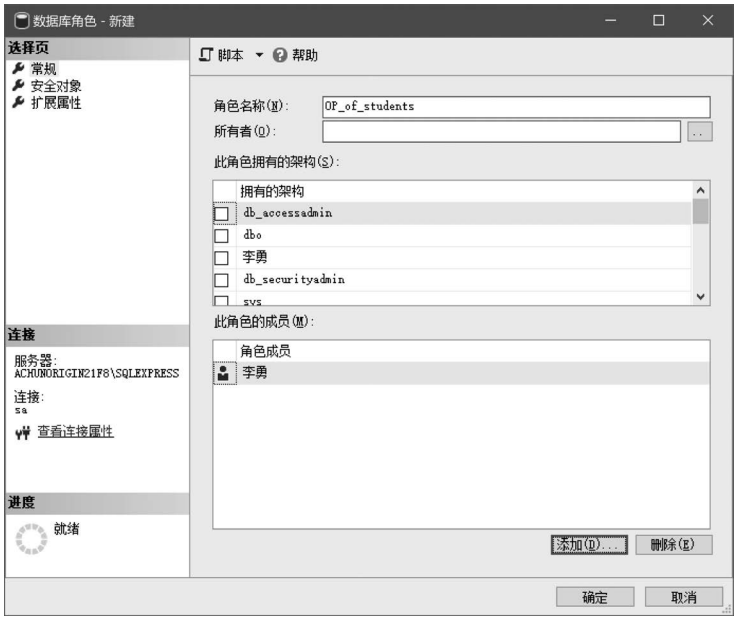


图　3.2.8

重复以上步骤，再次进入 school 数据库的角色目录，选择 OP_of_students 角色并右击，在弹出的快捷菜单中选择“属性”选项，进入对话框，选择“安全对象”选择页，在弹出的“安全对象”对话框中，单击“搜索”按钮，在弹出的“添加对象”对话框中选择“属于该架构的所有对象”单选按钮，构架名称选择 dbo，选择表[dbo].[STUDENTS]，此时便可以进入权限设置。把 STUDENTS 表上的 SELECT(选择)、UPDATE(更改)、DELECT(删除)、INSERT(插入)四种权限都赋予 OP_of_students 角色，如图 3.2.9 所示。

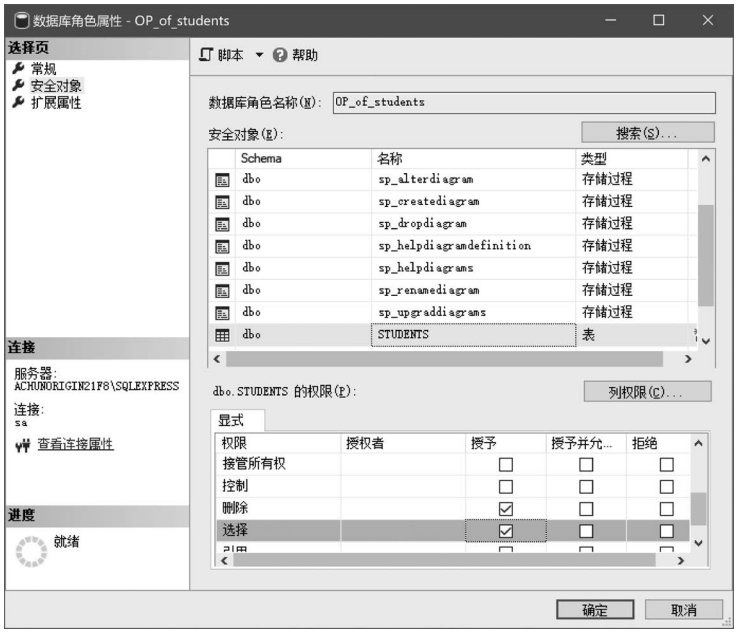


图　3.2.9

3.2.5 自我实践

(1) 以 sa 账号登录 SSMS,单击“新建查询”按钮,输入下列代码并执行。

第 1 行: EXEC sp_addlogin 'Li','123456';

第 2 行: USE school

第 3 行: EXEC sp_grantdbaccess'Li','happyrat';

第 4 行: GRANT SELECT,INSERT,UPDATE ON STUDENTS TO public;

第 5 行: GRANT ALL ON STUDENTS TO happyrat;

第 6 行: REVOKE SELECT ON STUDENTS TO happyrat;

第 7 行: DENY UPDATE ON STUDENTS TO happyrat;

(2) 回答下列问题:

第 1 行代码新建了一个名为 Li 的登录账户,123456 是什么? Li 登录账户将映射为数据库用户名 happyrat,为什么? 将是哪个数据库的用户?

解释第 4～7 行代码的作用。

若以账户 Li 登录服务器,能否对 school 数据库的表 STUDENTS 进行 SELECT 和 UPDATE 操作,为什么?

3.3 视图机制在自主存取控制上的应用

3.3.1 实验目的

通过实验加深对数据安全性的理解,熟悉视图机制在自主存取控制上的应用。

3.3.2 原理解析

1. 视图机制

为了说明视图机制的优点,先回顾一下授权粒度的定义: 授权粒度是指可以定义的数据对象的范围,是衡量授权机制是否灵活的一个重要指标。授权定义中数据对象的粒度越细,即可以定义的数据对象的范围越小,授权子系统就越灵活。关系数据库中授权的数据对象粒度从大到小为数据库、表、属性列、元组。

直接使用授权机制所能达到的数据对象的粒度最小只能是属性列,为了使数据粒度可以达到元组这一级,必须利用视图机制与授权机制配合使用。

视图机制与授权机制配合使用,首先可以利用视图机制屏蔽掉一部分保密数据,然后在视图上面再进一步定义存取权限,从而实现了水平子集和垂直子集上的安全,并且间接实现了支持存取谓词的用户权限定义。

例如，用户王平只能检索计算机系老师的信息，先建立计算机系老师的视图 CS_Teacher。

```
CREATE VIEW CS_Teacher
AS
SELECT
FROM Teacher
WHERE dept = 'CS';
```

在视图上进一步定义存取权限。

```
GRANT SELECT ON CS_Student TO 王平;
```

2. 视图的权限问题

如果创建了一个对象(关系/视图/角色)，则拥有对此基表的全部权限(包括授予别人权限的权限)。创建视图不需要授权，但是如果用户对基表没有任何权限，系统会拒绝创建视图。

3.3.3　实验内容

(1) 创建在选课表 CHOICES 上的视图 CS_View，授权给计算机系的讲授“计算机科学”这门课程(课程编号：10010)的数据库用户 Li，让其具有视图上的 SELECT 权限。

(2) 对视图上的 score 属性列的 UPDATE 权限授予用户 Li，可以修改学生的成绩，但是不能对学生的基本信息，如学号、选课号进行修改。

3.3.4　实验步骤

(1) 在数据库 school 上创建用户 Li，具体操作参见 3.1.4 节的(2)。

(2) 用 sa 账号登录数据库。新建查询，然后在 CHOICES 表上创建视图 CS_View(选课课程号为 10010)，如代码 3.3.1 所示。

```
USE school
GO
CREATE VIEW CS_View as
        SELECT * FROM CHOICES
        WHERE cid = '10010'
```

代码 3.3.1

结果如图 3.3.1 所示。

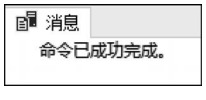


图　3.3.1

(3) 在视图 CS_View 上给用户 Li 授予 SELECT 的权限，如代码 3.3.2 所示。

```
USE school
GO
GRANT SELECT ON CS_View
    TO Li
```

代码 3.3.2

结果如图 3.3.2 所示。

(4) 将视图 CS_View 上 score 列的权限授予用户 Li,如代码 3.3.3 所示。

```
USE school
GO
GRANT UPDATE ON dbo.[CS_View]([score])
    TO Li
```

代码 3.3.3

结果如图 3.3.3 所示。

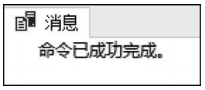


图 3.3.2

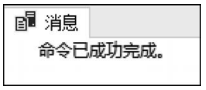


图 3.3.3

(5) 以用户 Li 登录 SSMS,然后新建查询,对 CS_View 进行查询操作,如代码 3.3.4 所示。

```
USE school
GO
SELECT * FROM CS_View
```

代码 3.3.4

结果如图 3.3.4 所示。

	no	sid	tid	cid	score
1	500000253	829348273	202560416	10010	87
2	500010915	894037661	200713929	10010	79
3	500023337	890644804	242635790	10010	88
4	500024940	829310417	221792985	10010	NULL
5	500048368	898738645	265304274	10010	77
6	500065154	821848893	254787674	10010	79
7	500077667	821494816	214751989	10010	98
8	500084641	881633930	250749054	10010	85
9	500100714	876820699	236991180	10010	88
10	500102772	817714690	238093990	10010	67

图 3.3.4

(6) 对 no 为 500024940 的学生的成绩进行修改,改为 90 分,如代码 3.3.5 所示。

```
USE school
GO
UPDATE CS_View SET score = 90
                WHERE no = 500024940
SELECT * FROM CS_View
```

代码 3.3.5

结果如图 3.3.5 所示。

	no	sid	tid	cid	score
1	500000253	829348273	202560416	10010	87
2	500010915	894037661	200713929	10010	79
3	500023337	890644804	242635790	10010	88
4	500024940	829310417	221792985	10010	90
5	500048368	898738645	265304274	10010	77
6	500065154	821848893	254787674	10010	79
7	500077667	821494816	214751989	10010	98
8	500084641	881633930	250749054	10010	85
9	500100714	876820699	236991180	10010	88
10	500102772	817714690	238093990	10010	67

图　3.3.5

3.3.5　自我实践

(1) 在数据库 school 上创建用户“王二”,具体操作参见 3.1.4 节中的(2)。在 STUDENTS 表上创建视图 grade2000,把年级为 2000 的学生元组放入视图。

(2) 授予用户王二在视图 grade2000 的 SELECT 权限。

3.4　Public 角色在安全性中的应用

3.4.1　实验目的

通过实验加深对 Public 角色的理解,特别是 Public 角色在安全性管理中的应用。

3.4.2　原理解析

Public 角色是一种特殊的固定数据库角色,数据库的每个合法用户都属于该角色。它为数据库中的用户提供了所有默认权限。这样就提供了一种机制,即给予那些没有适当权限的所有用户以一定的(通常是有限的)权限。

Public 角色为数据库中的所有用户都保留了默认的权限,因此是不能被删除的。一般情况下,Public 角色允许用户进行如下操作:使用某些系统过程查看并显示 master 数据库中的信息执行一些不需要权限的语句(如 PRINT)。

3.4.3　实验内容

(1) 在 SSMS 中新建查询,创建 test 登录用户以 Public 访问数据库,在 school 数据库

的 STUDENTS 表上授权查询操作给 Public，验证 test 用户是否可以查询 STUDENTS 表，再撤销 Public 权限，再验证是否可以查询。

(2) 在 school 数据库的 STUDENTS 表上授权查询操作给 Public，并授权给 test 用户，验证 test 用户是否可以查询 STUDENTS 表，先撤销 Public 权限，验证是否可以查询，再撤销 test 权限，再验证是否可以查询。

(3) 在 school 数据库的 STUDENTS 表上授权查询操作给 Public，并授权给 test 用户，验证 test 用户是否可以查询 STUDENTS 表，先撤销 test 权限，验证是否可以查询，再撤销 Public 权限，再验证是否可以查询。

3.4.4 实验步骤

(1) 以 sa 用户登录 SSMS，创建 test 登录用户以 Public 访问数据库，创建过程可参考 3.1 节，新建查询，在 school 数据库的 STUDENTS 表上授权查询操作给 Public，如代码 3.4.1 所示。

```
GRANT SELECT ON STUDENTS TO Public
```

代码 3.4.1

再以 test 用户登录 SSMS，新建查询，验证是否可以对 STUDENTS 表进行查询，如代码 3.4.2 所示。

```
SELECT * FROM STUDENTS
```

代码 3.4.2

查询结果如图 3.4.1 所示。

再以 sa 用户登录，撤销 Public 权限，如代码 3.4.3 所示。

```
REVOKE SELECT ON STUDENTS TO Public
```

代码 3.4.3

再以 test 用户登录，查询代码如代码 3.4.2 所示，查询结果如图 3.4.2 所示。

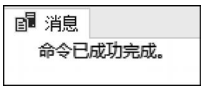


图 3.4.1

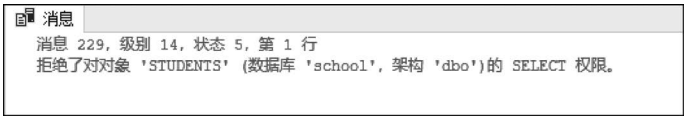


图 3.4.2

(2) 以 sa 用户登录 SSMS，新建查询，在 school 数据库的 STUDENTS 表上授权查询操作给 Public 和 test，如代码 3.4.4 所示。

```
GRANT SELECT ON STUDENTS TO Public
GRANT SELECT ON STUDENTS TO test
```

代码 3.4.4

再以 test 用户登录 SSMS，新建查询，验证是否可以对 STUDENTS 表进行查询，如代

码 3.4.2 所示，查询结果如图 3.4.1 所示。再以 sa 用户登录，先撤销 Public 权限，如代码 3.4.3 所示。以 test 用户登录，查询代码如代码 3.4.2 所示，查询结果如图 3.4.1 所示，再以 sa 用户登录，再撤销 test 权限，如代码 3.4.5 所示。

```
REVOKE SELECT ON STUDENTS TO test
```

代码 3.4.5

再以 test 用户登录，查询代码如代码 3.4.2 所示，查询结果如图 3.4.2 所示。

(3) 以 sa 用户登录 SSMS，新建查询，在 school 数据库的 STUDENTS 表上授权查询操作给 Public 和 test，如代码 3.4.6 所示。

```
GRANT SELECT ON STUDENTS TO Public
GRANT SELECT ON STUDENTS TO test
```

代码 3.4.6

再以 test 用户登录 SSMS，新建查询，验证是否可以对 STUDENTS 表进行查询，如代码 3.4.2 所示，查询结果如图 3.4.1 所示。再以 sa 用户登录，先撤销 test 权限，如代码 3.4.7 所示。以 test 用户登录，查询代码如代码 3.4.2 所示，查询结果如图 3.4.1 所示。再以 sa 用户登录，再撤销 Public 权限，如代码 3.4.3 所示。

```
REVOKE SELECT ON STUDENTS TO test
```

代码 3.4.7

再以 test 用户登录，查询代码如代码 3.4.2 所示，查询结果如图 3.4.2 所示。

3.4.5　自我实践

从实验中可以看出，授权给 Public 与授权给指定用户有什么区别？在实际应用中，哪个更安全一些？

3.5　理解架构的安全性管理

3.5.1　实验目的

通过实验加深对 SQL Server 中新增加的特性架构(Schema)的理解，并对自定义的架构进行权限的授予与撤销，掌握 SQL Server 中的架构管理。

3.5.2　原理解析

1. 架构的原理与特性

在 SQL Server 中，实现了 ANSI 中有关架构的概念。架构是被单个负责人(用户或者

角色)所拥有并构成唯一的命名空间,也可以看成是一种允许对数据库对象进行分组的容器对象,就是说可以在架构这个容器当中添加表、视图和存储过程等对象,然后将这个容器赋予用户。

其实架构这个概念在 SQL Server 2000 当中就有了,但当时用户和架构是隐含关联的,每个用户都拥有与其同名的架构。

在 SQL Server 中,一个数据库对象通过以下 4 个命名部分所组成的结构来引用。

<服务器>.<数据库>.<架构>.<对象>

但在 SQL Server 2000 中,一个数据库对象通过以下 4 个命名部分所组成的结构来引用。

<服务器>.<数据库>.<用户>.<对象>

如果想要把一个用户删除,必须先删除或者修改这个用户拥有的所有数据库对象。因此,如果有一个员工离职了,公司要想把其在数据库中所有的对象交接给另一个员工,则会十分麻烦。在实际应用中,会大量地使用 DBO 作为架构来赋予用户,这样,就会把用户的实际权限扩大化了。

在 SQL Server 中,架构与创建它的用户不再关联了,而是如上面所说的通过 4 个命名部分所组成的结构来引用。

通过用户与架构分离,可以得到许多好处。

(1) 多个用户可以通过角色或者成员关系来拥有同一个架构。

(2) 删除数据库用户变得更加简单方便。

(3) 删除数据库用户不需要重命名与用户名同名的架构所包含的对象,因此也无须对显式引用数据库对象的应用程序进行修改和测试。

(4) 多个用户可以共享同一个默认架构来统一命名。

(5) 共享默认架构使得开发人员可以为特定的应用程序创建特定的架构来存放对象,这比仅使用管理员架构(DBO schema)要好。

(6) 在架构和架构所包含的对象上设置权限(Permissions)比以前的版本拥有更高的可管理性。

在实际的应用中,架构有以下几个特点值得注意。

(1) 架构定义与用户分开。

(2) 在创建数据库用户时,可以指定该用户账号的默认架构。

(3) 若没有指定默认架构,则为 DBO,这样是为了向前兼容。

(4) 在应用中,如果通过 A.B 来引用一个对象 B,会先找与用户默认架构相同的架构下的对象,找不到则再找 DBO 对象。这是什么意思呢? 可以看一个例子,一个用户 Ray 在创建时指定默认的架构是 school。当 Ray 执行 SELECT * FROM TEACHERS 时,数据库系统会到 school 这个空间下,查找 TEACHERS 表这个对象。如果 school 空间真的有这个对象,则没有问题,可以找到。如果没有,则到 DBO 这个架构下找。

2. 与架构有关的语句

与架构有关的语句有 3 个,分别是:

```
CREATE SCHEMA -- 创建架构
ALTER SCHEMA -- 修改架构
DROP SCHEMA -- 删除架构
```

可以看出，这几个语句与修改数据库对象语句是差不多的。下面通过几个实例来学习如何使用这 3 个语句。

假设有一个数据库叫作 TEST。

```
USE TEST
GO
CREATE SCHEMA my_schema AUTHORIZATION Peter
CREATE TABLE A
(ID VARCHAR(20) NOT NULL UNIQUE,
NAME VARCHAR(20) NULL)
CREATE VIEW A_View
AS SELECT * FROM A
GRANT SELECT TO Ray
GRANT UPDATE TO Ray
```

在这个例子中，创建了一个名为 my_schema 的架构，该架构由 A 表和 A_View 视图组成。然后通过 AUTHORIZATION 选项将这个架构赋予了用户 Peter，这就是这个架构的主体(主体也可以拥有其他架构，并且可能不把当前架构当作默认架构)。最后两个语句最容易使人迷惑，这两句的意思是把 my_schema 这个架构当中的所有对象的 SELECT 权限和 UPDATE 权限赋予了 Ray。

下面列出几个 CREATE SCHEMA 语句需要注意的地方。

(1) CREATE SCHEMA 语句可以创建一个架构(该架构含有表和视图)，并能用单个语句来授予、取消或剥夺可保护(Securable)资源的权限(可保护资源是指 SQL Server 授权系统规则可以访问的资源)。这里有 3 个可获得资源的范围，即服务器、数据库以及架构，这些又包含了其他可保护资源，如 SQL Server 登录、数据库用户、表以及存储过程等。

(2) CREATE SCHEMA 语句是原子性的。换句话说，在 CREATE SCHEMA 语句运行期间如果有任何错误发生，那么在架构中指定的 Transact-SQL 语句都不会运行。

(3) 在 CREATE SCHEMA 语句中创建的数据库对象可以通过任何顺序进行指定，但是除了一个例外的情况：引用了另一个视图的某个视图必须在该被引用的视图后面进行指定。

(4) 数据库层次上的主体可以是数据库用户、角色，也可以是应用程序角色(角色和应用程序角色将在随后进行讨论)。在 CREATE SCHEMA 语句的 AUTHORIZATION 子句中指定的主体是架构内创建的所有对象的拥有者。架构含有对象的这种拥有关系通过 ALTER AUTHORIZATION 语句传递给其他数据库层次上的任何主体。

(5) 需要获取对数据库的 CREATE SCHEMA 权限，才能执行 CREATE SCHEMA 语句。同样，要想创建在 CREATE SCHEMA 语句内指定的对象，必须取得相应的 CREATE 权限。

ALTER SCHEMA 语句可以在相同数据库的不同架构之间转换某个对象。其语法格式如下：

```
ALTER SCHEMA schema_name TRANSFER object_name
```

看以下例子：

```
USE school
ALTER Schema Schema_A TRANSFER Person.Students
```

上面的例子是更改了 school 数据库中的一个名为 Schema_A 的架构，把同一个数据库中的 Person 架构下的 Students 表转换进来了。

ALTER SCHEMA 语句只能用来转换同一个数据库中不同架构之间的对象(架构内部的单个对象可以使用 ALTER TABLE 语句或者 ALTE RVIEW 语句进行更改)。

另外，还有 ALTER AUTHORIZATION 语句，可以用来修改架构的从属关系，例如 ALTER AUTHORIZATION ON SCHEMA::my_schema TO Jack 可以把 my_schema 架构的主体改为用户 Jack。

DROP SCHEMA 语句可以用来从数据库中删除一个模式。只有当架构没有包含任何对象时，才可以对某个架构成功地执行 DROP SCHEMA 语句。否则，系统将拒绝执行 DROP SCHEMA 语句。

3.5.3 实验内容

(1) 在 SSMS 中创建一个 user1 的登录用户，采用 SQL Server 身份验证，密码为 user1，允许其访问数据库 test，并加入 db_owner 的架构和角色成员，用 user1 登录 SSMS，创建表 a，并查询表 a，看执行语句是否正确。

(2) 在 SSMS 中以管理员账户登录，再次创建表 a，看执行是否正确，若正确，则看看是否出现两个表 a，若出现，执行一段插入语句 INSERT a values('a','aa')，在管理工具中看数据插入哪个表中。

(3) 分别在管理员账户和 user1 账户下新建查询，执行查询语句对表 a 执行查询，分别记录查询结果，并说明系统分别对哪个表 a 进行查询。

3.5.4 实验步骤

(1) 在 SSMS 中创建一个 user1 的登录用户，采用 SQL Server 身份验证，密码为 user1，允许其访问数据库 test，并加入 db_owner 的架构和角色成员，如图 3.5.1 所示。

用 user1 账户登录 SSMS，新建查询，执行代码 3.5.1。

```
CREATE TABLE a (a1 char(1),b1 char(2))
```

代码 3.5.1

执行结果如图 3.5.2 所示。

在 user1 用户下查看 test 数据库下的对象，如图 3.5.3 所示。

执行代码 3.5.2 进行查询。

```
SELECT * FROM a
```

代码 3.5.2

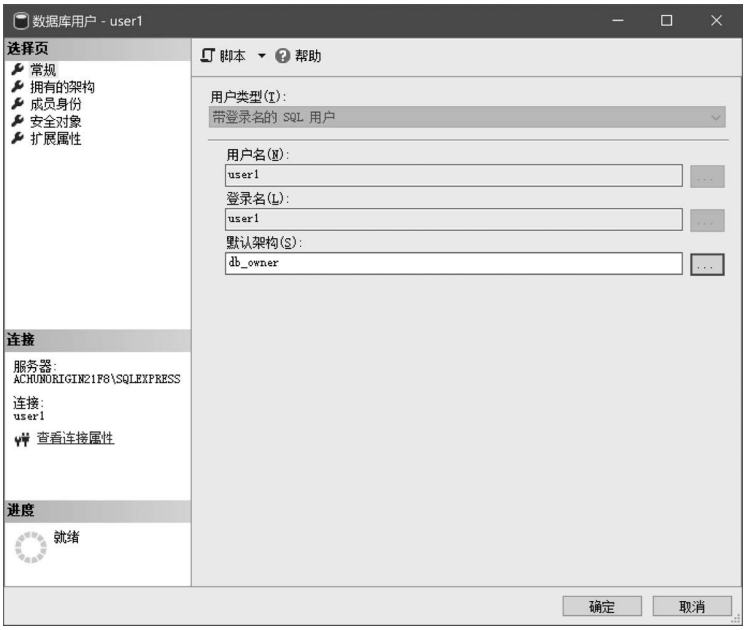


图　3.5.1

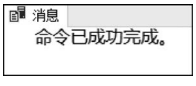


图　3.5.2

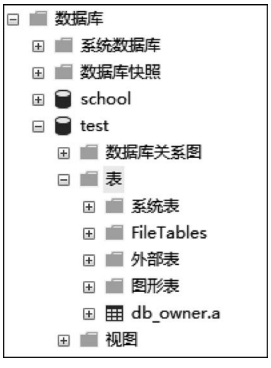


图　3.5.3

查询结果如图 3.5.4 所示。

(2) 在 SSMS 中以管理员账户登录,执行代码 3.5.1 创建表 a,执行结果如图 3.5.2 所示,此时查询数据库 test 下对象,发现有两个表 a,如图 3.5.5 所示。

图　3.5.4

图　3.5.5

执行查询代码 3.5.3。

```
INSERT INTO a values('a','aa')
```

代码 3.5.3

执行结果如图 3.5.6 所示。

可以发现数据捕入到表 dbo.a，而表 db_owner.a 中的数据依然为空，如图 3.5.7 所示。

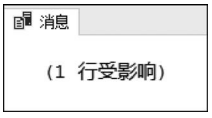


图 3.5.6

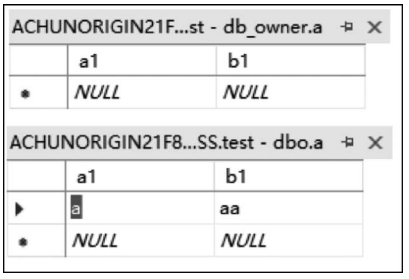


图 3.5.7

(3) 在 SSMS 中以管理员账户登录，新建查询，执行查询代码 3.5.2，查询结果如图 3.5.8 所示。

可以发现查询执行结果显示的是表 dbo.a 上的数据。

用 user1 账户登录 SSMS，新建查询，执行代码 3.5.2，查询结果如图 3.5.9 所示。

图 3.5.8

图 3.5.9

可以发现 user1 查询执行结果显示的是表 db_owner.a 上的数据。

3.5.5 自我实践

在实验中为什么能够成功地建立两个名称相同的表 a？如果对 user1 不分配 db_owner 的架构和角色成员，上述实验是否可以实现？请实验验证。

3.6 数据库中加密机制的安全管理

3.6.1 实验目的

通过实验理解 SQL Server 的加密体系，并掌握如何对数据库中的数据加密与解密。

3.6.2　原理解析

1. 数据加密及基本功能

(1) 数据加密就是按确定的加密变换方法(加密算法)对需要保护的数据(也称为明文,Plaintext)做处理,使其变换为难以识读的数据(密文,Ciphertext)。其逆过程即将密文按对应的解密变换方法(解密算法)恢复出现明文的过程,称为数据解密。

为了使加密算法能被许多人共用,在加密过程中又引入了一个可变量加密密钥。这样,不改变加密算法,只要按照需要改变密钥,也能将相同的明文加密成不同的密文。

(2) 数据加密的基本功能包括:

- 防止不速之客查看机密的数据文件;
- 防止机密数据被泄露或篡改;
- 防止特权用户(如系统管理员)查看私人数据文件;
- 使入侵者不能轻易地查找一个系统的文件。

(3) 与一般的数据加密和文件加密相比,由于数据库中数据有很强的相关性,并且数据量大,因此对其加密要比普通数据加密和文件加密有更大的难度,密钥管理更加困难。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。数据加密的过程实际上就是根据一定的算法将原始数据变换为不可直接识别的格式,从而使得不知道解密算法的人无法获知数据的内容,而仅允许经过授权的人员访问和读取数据,从而确保数据的保密性,是一种有助于保护数据的机制。

因此,数据库加密要求做到:

① 数据库中信息保存时间比较长,采用合适的加密方式,从根本上达到不可破译。

② 加密后,加密数据占用的存储空间不宜明显增大。

③ 加密/解密速度要快,尤其是解密速度,要使用户感觉不到加密/解密过程中产生的时延,以及系统性能的变化。

④ 授权机制要尽可能灵活。在多用户环境中使用数据库系统,每个用户只用到其中一小部分数据。所以,系统应有比较强的访问控制机制,再加上灵活的授权机制配合起来对数据库数据进行保护。这样既增加了系统的安全性,又方便了用户的使用。

⑤ 提供一套安全的、灵活的密钥管理机制。

⑥ 不影响数据库系统的原有功能,保持对数据库操作(如查询、检索、修改、更新)的灵活性和简便性。

⑦ 加密后仍能满足用户对数据库不同的粒度进行访问。

2. 数据加密的常用方法

(1) 替换方法。这种方法是制定一种规则,使用密钥(Encryption Key)将明文中的每个字母或每组字母替换为另一个或一组字母。例如,下面的这组字母对应关系就构成了一个替换加密器:

明文字母:A B C D E F ……

密文字母：K U P S W B ……

虽然说替换加密法比代码加密法应用的范围要广，但使用得多了，窃密者就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) 置换方法。变位加密法不隐藏原来明文的字符，而是将明文的字符按不同的顺序重新排列。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。按照密钥中数字指示的顺序，将原文重新抄写，就形成密文。

(3) 混合方法。将以上两种方法综合运用，进行有限次的复合与迭代，其中最著名的是美国 1977 年制定的官方加密标准：数据加密标准(Data Encryption Standard，DES)。

3. SQL Server 的数据加密技术

SQL Server 2005 是微软开始实施其“可信赖计算”计划以来的第一个主要的产品，提供了丰富的安全特性，为企业数据提供安全保障。对开发人员来说，最关注的是如何在程序设计过程中应用这些特性来保护数据库中的数据安全。本书将从应用程序开发者角度探讨基于数据加密特性的应用。

数据用数字方式存储在服务器中并非万无一失。尽管新的版本远比以前的版本安全，但攻击者还是有可能获得存储的数据。因此，数据加密成为更彻底的数据保护手段，即使攻击者得以存取数据，也不得不解密，因而对数据增加了一层保护。

SQL Server 2000 以前的版本没有内置数据加密功能，若要在 SQL Server 2000 中进行数据加密，不得不买第三家产品，然后在服务器外部作 COM 调用或者是数据送服务器之前在客户端的应用中执行加密。这意味着加密的密钥或证书不得不由加密者自己负责保护，而保护密钥是数据加密中最难的事，所以即使很多应用中数据已被很强地加密过，数据保护仍然很弱。

SQL Server 通过将数据加密作为数据库的内在特性解决了这个问题。除了提供多层次的密钥和丰富的加密算法外，最大的好处是可以选择数据服务器管理密钥。SQL Server 服务器支持的加密算法如下。

(1) 对称密钥加密(Symmetric Key Encryption)。对加密和解密使用相同的密钥。加密功能支持的最快加密模式是对称密钥加密。此模式适用于处理大量数据。可以通过证书、密码或其他对称密钥加密对称密钥。通常，这种加密方式在应用中难以实施，因为用同一种安全方式共享密钥很难。但当数据储存在 SQL Server 中时，这种方式很理想，可以让服务器管理它。SQL Server 支持多种对称密钥加密算法，包括 DES、Triple DES(3DES)、RC2、RC4、128 位 RC4、DESX、128 位 AES、192 位 AES 和 256 位 AES。这些算法使用 Windows 加密 API 实现。

(2) 非对称密钥加密(Asymmetric Key Encryption)。使用一组公共/私人密钥系统，加密时使用一种密钥，解密时使用另一种密钥。公共密钥可以广泛地共享。当需要用加密方式向服务器外部传送数据时，这种加密方式更方便。SQL Server 支持 RSA 加密算法以及 512 位、1024 位和 2048 位的密钥强度。

(3) 数字证书(Certificate)。一种非对称密钥加密，但是，一个组织可以使用证书并通过数字签名将一组公钥和私钥与其拥有者相关联。SQL Server 支持“因特网工程工作组”(IETF)X.509 版本 3(X.509v3)规范。一个组织可以对 SQL Server 使用外部生成的证书，

或者可以使用 SQL Server 生成的证书。

SQL Server 采用多级密钥来保护其内部的密钥和数据，如图 3.6.1 所示。

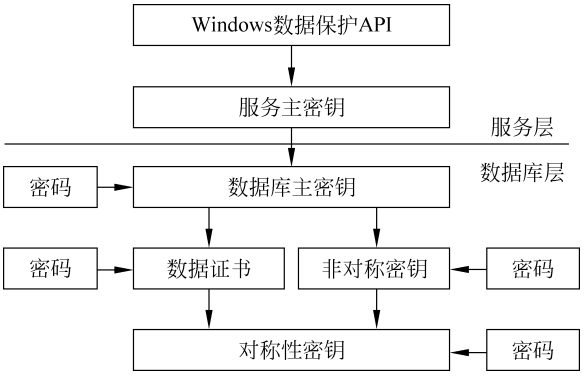


图　3.6.1

图 3.6.1 中引出箭头的密钥或服务用于保护箭头所指的密钥。所以服务主密钥(Service Master Key)保护数据库主密钥(Database Master Key)，而数据库主密钥又保护证书(Certificates)和非对称密钥(Asymmetric Keys)。而最底层的对称性密钥(Symmetric Keys)被证书、非对称密钥或其他的对称性密钥保护(箭头又指回其本身)。只需通过提供密码来保护这一系列的密钥。

图 3.6.1 中顶层的服务主密钥，安装 SQL Server 新实例时自动产生和安装，不能删除此密钥，但数据库管理员能对其进行基本的维护，如备份该密钥到一个加密文件，当其危及安全时可进行更新、恢复。

服务主密钥由 DPAPI(Data Protection API)管理。服务主密钥是首次需要它来加密其他密钥时自动生成的。默认情况下，服务主密钥使用 Windows 数据保护 API 和本地计算机密钥进行加密。只有创建服务主密钥的 Windows 服务账户或有权访问服务账户名称和密码的主体能够打开服务主密钥。服务主密钥本身是对称式加密，用来加密服务器中的数据库主密钥。

数据库主密钥与服务主密钥不同，在加密数据库数据之前，必须由数据库管理员创建数据库主密钥。通常管理员在产生该密钥时，提供一个口令，所以是用口令和服务主密钥来加密。如果有足够的权限，可以在需要时显式地或自动地打开该密钥。

每个数据库只有一个数据库主密钥。可以用 ALTER MASTR KEY 语句来删除加密，更改口令或删除数据库主密钥。通常这由数据库管理员来负责做这些。

有了数据库主密钥，就可以着手加密数据。Transact-SQL 有置于其内的加密支持。使用 CREATE 语句创建各种密码，使用 ALTER 语句进行修改。例如要创建对称式加密，可以通过一对函数 EncryptByKey 和 DecryptByKey 来完成。

3.6.3　实验内容

(1) 在数据库中创建主密钥 master key，在主密钥的基础上创建数字证书，使得主密钥对该数字证书加密，再创建对称密钥，使得数字证书对其加密。创建用户登录表 log_in 进

行加密的验证。

(2) 使用证书解开对称密钥,再利用对称密钥对表 log_in 的 pwd 字段进行加密处理,查询该表,验证 pwd 字段是否加密成功。

(3) 再一次使用证书解开对称密钥,对表 log_in 中 pwd 字段的数据进行解密处理,进行解密的查询,验证 pwd 字段是否解密成功。

3.6.4 实验步骤

(1) 在数据库中创建主密钥 master key,并对主密钥分配密码 p@ssw0rd,其 SQL 代码执行过程如代码 3.6.1 所示。

```
CREATE master key
ENCRYPTION BY password = 'p@ssw0rd'
```

代码 3.6.1

在主密钥的基础上创建数字证书 Mycert,使得主密钥对该数字证书加密,并设置证书终止实际时间,此时间必须设置为将来的时间,如代码 3.6.2 所示。

```
CREATE certificate Mycert
WITH
subject = 'My Certificate',
expiry_date = '2025 - 2 - 12 20:57:29'
```

代码 3.6.2

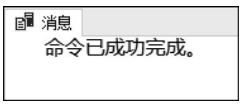


图 3.6.2

执行结果如图 3.6.2 所示。

再创建对称密钥 sym_my,使得数字证书对其加密,从 SQL Server 2016 (13.x) 开始,除 AES_128、AES_192 和 AES_256 以外的所有算法都已过时。若要使用旧算法(不推荐),必须将数据库的数据库兼容级别设置为 120 或更低。这里创建名为 AES_256 的对称密钥,然后使用证书 MyCert 对新密钥进行加密,执行代码如代码 3.6.3 所示。

```
CREATE symmetric key sym_my
WITH ALGORITHM = AES_256 encryption BY certificate MyCert;
```

代码 3.6.3

再创建个用户登录表 log_in 来验证下面的加密解密过程,如代码 3.6.4 所示。

```
CREATE TABLE dbo.log_in
(ID int identity primary key,[pwd] nvarchar(100))
```

代码 3.6.4

(2) 使用证书解开对称密钥,执行过程如代码 3.6.5 所示。

```
OPEN symmetric key sym_my decryption BY certificate MyCert
```

代码 3.6.5

再向 log_in 表中插入数据，对 pwd 字段数据进行加密处理，如代码 3.6.6 所示。

```
INSERT INTO log_in(pwd)values(encryptbykey(key_guid('sym_my'),N'mypassword1'))
INSERT INTO log_in(pwd)values(encryptbykey(key_guid('sym_my'),N'mypassword2'))
INSERT INTO log_in(pwd)values(encryptbykey(key_guid('sym_my'),N'mypassword3'))
```

代码 3.6.6

执行结果如图 3.6.3 所示。

完成后关闭对称密钥，如代码 3.6.7 所示。

```
CLOSE symmetric key sym_my
```

代码 3.6.7

对加密后的表 log_in 进行查询，如代码 3.6.8 所示。

```
SELECT * FROM dbo.log_in
```

代码 3.6.8

查询结果如图 3.6.4 所示。

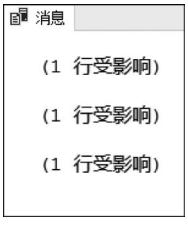


图　3.6.3

图　3.6.4

可以看出，对表 log_in 的 pwd 字段加密成功，看不到 pwd 字段这一列的实际内容。

(3) 再一次使用证书解开对称密钥，对表 log_in 中 pwd 字段的数据进行解密处理，如代码 3.6.9 所示。

```
OPEN symmetric key sym_my decryption BY certificate MyCert
```

代码 3.6.9

再对解密的表进行查询，执行过程如代码 3.6.10 所示。

```
SELECT ID,Cast (DecryptByKey(pwd) AS nvarchar) AS [pwd] FROM dbo.log_in
```

代码 3.6.10

执行结果如图 3.6.5 所示。

可以看出，对表 log_in 的 pwd 字段解密成功，结果返回真实存储的数据。

图　3.6.5

3.6.5　自我实践

在本次实验中，可否直接使用对称密钥对数据进行加密？这样的加密方式与本实验中采用的方式有何异同？

3.7 应用程序角色的安全性管理

3.7.1 实验目的

通过实验理解 SQL Server 中应用程序角色的作用，并掌握如何创建应用程序角色。

3.7.2 原理解析

应用程序角色是一个可提供对应用程序（而不是数据库角色或者用户）分配权限的方法。可以连接到数据库，激活应用程序角色以及采用授予应用程序的权限。授予应用程序角色的权限在连接期间有效。很多人会有疑问，应用程序角色是不是数据库角色的一种？为什么要特定设置这一种角色？主要作用是什么？下面慢慢解析这些问题。

1. 应用程序角色的作用

一般来说，SQL Server 中的安全系统在最低级别，即数据库本身上实现。无论使用什么应用程序与 SQL Server 通信，这都是控制用户活动的最佳方法。但是，有时候必须自定义安全控制以适应个别应用程序的特殊需要，尤其是当处理复杂数据库和含有大量的数据库时。也就是说，个别的应用程序有特殊的权限需要，例如访问一些平常用户不会访问或者没有访问权限的数据对象。

另外，可能希望限制用户只能通过特定的应用程序（如 SQL 查询分析器或 Excel 之类）来访问数据库或者防止用户直接访问数据。限制用户的这种访问方式将禁止用户使用应用程序连接到 SQL Server 实例并执行编写质量差的查询，以免对整个服务器的性能造成负面影响。

2. 应用程序角色的特点

应用程序与标准的角色不同，是特有的角色。

（1）应用程序角色不会包含成员。当通过特定的应用程序为用户连接激活应用角色时，将获得应用程序角色的权限。用户之所以与应用程序角色关联，是因为用户能够运行激活该角色的应用程序，而不是因为其是角色成员。

（2）默认情况下，应用程序是非活动的，需要用密码激活。当一个应用程序角色被该应用程序激活以用于连接时，连接会在连接期间永久地失去数据库中所有用来登录的权限、用户账户、其他组或数据库角色。连接获得与数据库的应用程序角色相关联的权限，应用程序角色存在于该数据库中。因为应用程序角色只能应用于所存在的数据库中，所以连接只能通过授予其他数据库中 guest 用户账户的权限，获得对另一个数据库的访问。因此，如果数据库中没有 guest 用户账户，则连接无法获得对该数据库的访问。如果 guest 用户账户确实

存在于数据库中，但是访问对象的权限没有显式地授予 guest，则无论是谁创建了对象，连接都不能访问该对象。从应用程序角色中获得的权限一直有效，直到连接从 SQL Server 退出为止。

(3) 若要确保可以执行应用程序的所有函数，连接必须在连接期间失去应用于登录和用户账户或所有数据库中的其他组或数据库角色的默认权限，并获得与应用程序角色相关联的权限。例如，如果应用程序必须访问通常拒绝用户访问的表，则应废除对该用户拒绝的访问权限，以使用户能够成功使用该应用程序。应用程序角色通过临时挂起用户的默认权限并只指派应用程序角色的权限而克服任何与用户的默认权限发生的冲突。

应用程序角色允许应用程序(而不是 SQL Server)接管验证用户身份的责任。但是，SQL Server 在应用程序访问数据库时仍需对其进行验证，因此应用程序必须提供密码，因为没有其他方法可以验证应用程序。

注意：应用程序角色是单向的。也就是说，对于一个指定的连接，一旦确定已经激活应用程序角色，则对那个连接来说，无法再回到用户自己的安全上下文。为了回到用户自己的安全上下文，必须终止这个连接，并再次登录。

3. 应用程序角色的作用过程

应用程序角色的作用过程类似下面这样：

(1) 用户登录(很可能使用应用程序提供的登录窗口)。

(2) 验证登录，获得访问权限。

(3) 应用程序执行名为 sp_setapprole 的系统存储过程，并提供角色名和密码。

(4) 验证应用程序角色，然后，连接被切换到应用程序角色的安全上下文(失去了用户拥有的所有权限，拥有的是应用程序角色的权限)。

(5) 在整个连接期间，将继续保持基于应用程序角色的访问权限，而非基于个人登录名的访问权限，不能回到自己的访问信息。

通常只会想把应用程序角色作为真正的应用程序情形的一部分来使用，并且，将直接在应用程序中构建设置应用程序角色的代码。另外，也将需要把密码编译到应用程序中，或者把这一信息存储在某个本地文件中，以便在需要时进行访问。

4. 应用程序角色的相关语句

1) 创建应用程序角色

要创建应用程序角色，可以使用一个新的称为 sp_addapprole 的系统存储过程。该存储过程是另一个使用起来相当简单的存储过程。其语法如下：

```
sp_addapprole[@rolename = ]<角色名>,[@password = ]<'密码'>
```

例如，要创建一个名为 MyApp 的应用程序角色，密码为 123，则命令为：

```
EXEC sp_addapprole MyApp,'123'
```

2) 向应用程序角色添加权限

向应用程序角色中添加许可权限与向任何其他事物中添加许可权限一样。只需在使用用户登录 ID 或者常规的服务器或数据库角色的地方，替换成应用程序角色的名字即可。

例如，要向前面创建的角色 MyApp 授权在 TableA 上的查询权限，语句为：

```
GRANT SELECT ON TableA TO MyApp
```

现在，应用程序角色 MyApp 有了在 TableA 查询的权限，除此之外，没有任何权限。

3）使用应用程序角色

应用程序角色的使用是这样一个过程：调用系统存储过程（sp_setapprole），并提供应用程序角色的名字和相应的密码。其语法如下：

```
sp_setapprole[@rolename = ]<角色名>,
            [@password = ]{Encrypt N'密码')|'密码'
            [,[@encrypt = ]'加密选项']
```

rolename 就是想要激活的应用程序角色的名字。

password 或者原样提供，或者使用 ODBC Encrypt 函数进行加密处理。如果准备加密密码，那么，需要在 Encrypt 关键字之后，用引号引住密码，并在前面放置一个大写的 N 向 SQL Server 表明，正在处理的是 Unicode 字符串，并且应当被如此对待。

注意：为加密参数使用的是花括号，而不是圆括号。如果不希望加密，则不必使用 Encrypt 关键字来提供密码。

仅当为密码参数选择了加密选项时，才需要使用 encryption style（加密类型）。如果对密码进行了加密，那么以 ODBC 作为加密类型。

下面举一个简单的例子来加深对如何使用应用程序角色的认识。

在数据库 A 中，UserA 不能访问 TableA 表，但是能够访问 TableB 表，执行以下语句：

```
SELECT * FROM TableA
SELECT * FROM TableB
```

很明显，第一个语句会发生错误，第二个则会返回 TableB 中的所有记录。

现在，要激活之前创建的应用程序角色 MyApp，UserA 输入：

```
sp_setapprole MyApp,{Encrypt N'123'},'odbc'
```

执行上述语句后，则会得到系统返回的激活应用角色成功的消息。然后再执行一次以下语句：

```
SELECT * FROM TableA
SELECT * FROM TableB
```

可以看到，现在能够访问 TableA（因为上面第二个例子中已经把 SELECT TableA 的权限授予了 UserA），但却不能访问 TableB（因为没有授予关于 TableB 的任何权限给 UserA）。

4）删除应用程序角色

当服务器上不再需要应用程序角色时，可以使用 sp_dropapprole 从系统中删除它。其语法如下：

```
sp_dropapprole[@rolename = ]<角色名>
```

例如，要删除 MyApp 这个应用程序角色，只需执行下面的语句：

```
EXEC sp_dropapprole MyApp
```

3.7.3　实验内容

1. 创建示例环境

首先使用下面的代码创建一个登录 1_test，并且为登录在数据库 school 中创建关联的用户账户 u_test，授予用户账户 u_test 对表 COURSES 的 SELECT 权限，然后再创建一个应用程序角色 r_p_test，授予该角色对表 STUDENTS 的 SELECT 权限。

2. 查看权限

激活应用程序角色 r_p_test 前，验证登录是否具有表 COURSES 的访问权，是否具有表 STUDENTS 的访问权。

3. 激活应用程序角色

用密码激活 r_p_test 应用程序角色，并且在将此密码发送到 SQL Server 之前对其加密；激活应用程序角色 r_p_test 后，登录再验证是否有对表 COURSES 的访问权，同样验证是否有对表 STUDENTS 的访问权。

3.7.4　实验步骤

1. 创建示例环境

首先对 school 数据库创建一个名为 1_test、密码为 1_test_1 的登录，为该登录在数据库 school 中创建关联的用户账户 u_test，并且授予用户账户 u_test 对表 COURSES 的 SELECT 权限。新建查询，执行 SQL 语句如代码 3.7.1 所示。

```
USE school
 -- 创建一个登录 1_test,密码为 1_test,默认数据库为 school
EXEC sp_addlogin'1_test','1_test_1','school'
 -- 为登录 1_test 在数据库 school 中添加安全账户 u_test
EXEC sp_grantdbaccess'1_test','u_test'
 -- 授予安全账户 u_test 对 COURSES 表的 SELECT 权限
GRANT SELECT ON COURSES TO u_test
```

代码 3.7.1

执行结果如图 3.7.1 所示。

创建一个应用程序角色 r_p_test，并授予 STUDENTS 表的 SELECT 权限，如代码 3.7.2 所示。

```
 -- 创建一个应用程序角色 r_p_test,密码为 abc
EXEC sp_addapprole 'r_p_test','abc'
 -- 授予角色 r_p_test 对 STUDENTS 表的 SELECT 权限
GRANT SELECT ON STUDENTS TO r_p_test
```

代码 3.7.2

执行结果同样如图 3.7.1 所示。

2. 查看权限

以 1_test 为账户名、密码为 1_test_1 登录到 SSMS，新建查询，分别查询表 COURSES 与表 STUDENTS，验证是否具有对这两个表的访问权限。

其中，查询 COURSES 表的 SQL 执行代码，如代码 3.7.3 所示。

```
SELECT courses_count = COUNT ( * ) FROM COURSES
```

代码 3.7.3

执行结果如图 3.7.2 所示。

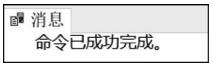


图 3.7.1

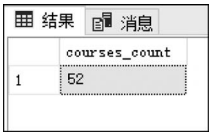


图 3.7.2

查询 STUDENTS 表的 SQL 执行，如代码 3.7.4 所示。

```
SELECT students_count = COUNT( * ) FROM STUDENTS
```

代码 3.7.4

执行结果如图 3.7.3 所示。

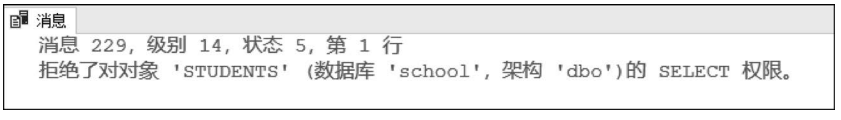


图 3.7.3

通过上面的验证可以发现，在激活应用程序角色之前，对 COURSES 表有访问权限但对表 STUDENTS 没有访问权限。

3. 激活应用程序角色

以 sa 账号登录 SSMS，新建查询，用密码激活 r_p_test 应用程序角色，并且在将此密码发送到 SQL Server 之前对其加密，执行代码如代码 3.7.5 所示。

```
DECLARE @cookie varbinary(8000);
EXEC sp_setapprole 'r_p_test','abc',@fCreateCookie = true,@cookie = @cookie OUTPUT;
```

代码 3.7.5

新建查询，分别查询表 COURSES 与表 STUDENTS，验证是否具有对这两个表的访问权限。其中，查询 COURSES 表的 SQL 执行代码，如代码 3.7.6 所示。

```
SELECT courses_count = COUNT ( * ) FROM COURSES
```

代码 3.7.6

查询结果如图 3.7.4 所示。

图　3.7.4

查询 STUDENTS 表的 SQL 执行代码，如代码 3.7.7 所示。

```
SELECT students _count = COUNT ( * ) FROM STUDENTS
```

代码 3.7.7

查询结果如图 3.7.5 所示。

再查询数据库用户名，如代码 3.7.8 所示。

```
SELECT USER_NAME();
```

代码 3.7.8

查询结果如图 3.7.6 所示。

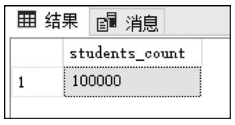


图　3.7.5

图　3.7.6

通过以上验证可以发现，在激活应用程序角色之后，实际的数据库用户名已变为 r_p_test，此时，对表 COURSES 没有了访问权限，而对表 STUDENTS 有了访问权限。

3.7.5　自我实践

在本实验中，如果在激活应用程序之前，利用 SELECT USERNAME 此时的数据库用户名，查询结果应该是怎样的？为什么？

3.8　行级安全

3.8.1　实验目的

本实验的目的是通过实验了解行级安全(Row-Level Security in SQL Server，RLS)，并掌握如何设置行级安全和使用的方法。

3.8.2　原理解释

行级安全是一种行级粒度的安全机制，它根据登录的用户授权的上下文限制该用户能

够访问的 SQL Server 表中的记录。DBMS 根据用户是谁以及用户可以访问哪些记录来显示表中的记录。这样是为了允许特定用户只访问他们自身的数据而不允许查看其他用户的数据。参见图 3.8.1,行级安全是 SQL Server 2016 开始才有的,之前的版本中要实现行级安全需要通过视图和自主控制相结合才能实现。

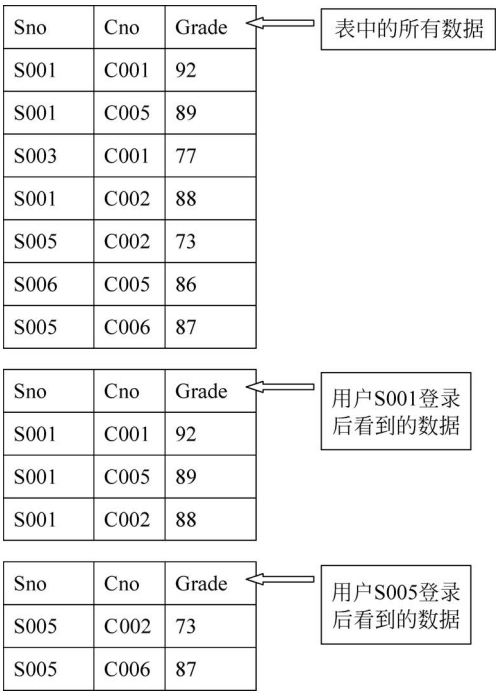


图 3.8.1

3.8.3 实验内容

(1) 创建 3 个用户账号和实验需的简单数据的表格 Grade,并授权给用户访问表的权限。
(2) 创建一个新架构和一个内联表值函数。
(3) 创建一个安全策略,将该内联表值函数添加为该安全策略的筛选器谓词。
(4) 将内联表值函数的查询权限赋予用户。
(5) 3 个用户对 Grade 表进行查询来测试行安全策略是否生效。

3.8.4 实验步骤

(1) 创建将演示不同访问功能的三个用户账户,如代码 3.8.1 所示。

```
CREATE USER Manager WITHOUT LOGIN;
CREATE USER S001 WITHOUT LOGIN;
CREATE USER S005 WITHOUT LOGIN;
GO
```

代码 3.8.1

结果如图 3.8.2 所示。

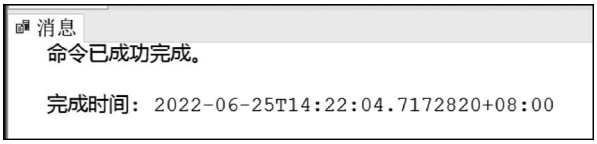


图　3.8.2

(2) 创建用于保留数据的表,如代码 3.8.2 所示。

```
CREATE SCHEMA Grade
CREATE TABLE Grade
    (Sno varchar(10),
    Cno varchar(10),
    Grade int);
```

代码 3.8.2

结果如图 3.8.3 所示。

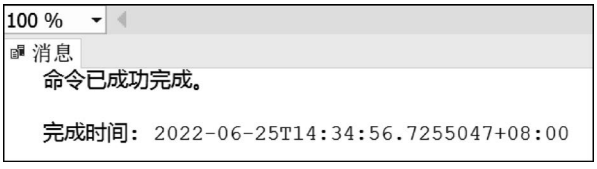


图　3.8.3

(3) 使用 7 行数据填充该表。

```
--插入信息
INSERT INTO Grade.Grade VALUES ('S001', 'C001', 92);
INSERT INTO Grade.Grade VALUES ('S001', 'C005', 89);
INSERT INTO Grade.Grade VALUES ('S003', 'C001', 77);
INSERT INTO Grade.Grade VALUES ('S001', 'C002', 88);
INSERT INTO Grade.Grade VALUES ('S005', 'C002', 73);
INSERT INTO Grade.Grade VALUES ('S006', 'C005', 86);
INSERT INTO Grade.Grade VALUES ('S005', 'C006', 87);
--查询表中所有数据
SELECT * FROM Grade.Grade ;
```

代码 3.8.3

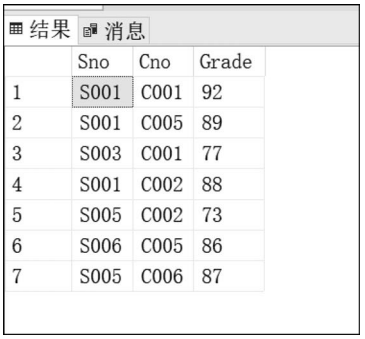
结果如图 3.8.4 所示。

(4) 向每个用户授予表的读取访问权限,如代码 3.8.4 所示。

```
GRANT SELECT ON Grade.Grade TO Manager;
GRANT SELECT ON Grade.Grade TO S001;
GRANT SELECT ON Grade.Grade TO S005;
GO
```

代码 3.8.4

结果如图 3.8.5 所示。



结果 消息

	Sno	Cno	Grade
1	S001	C001	92
2	S001	C005	89
3	S003	C001	77
4	S001	C002	88
5	S005	C002	73
6	S006	C005	86
7	S005	C006	87

图 3.8.4

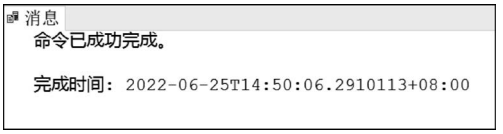


图 3.8.5

(5) 创建一个新架构和一个内联表值函数。该函数在 GradeRep 列中的行与执行查询的用户相同时(@GradeRep＝USER_NAME())或是在执行查询的用户是 Manager 用户(USER_NAME()＝'Manager')时返回 1,如代码 3.8.5 所示。

```
CREATE SCHEMA security;
GO
CREATE FUNCTION Security.tvf_security_predicate(@GradeRep AS nvarchar(50))
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN SELECT 1 AS tvf_security_predicate_result
WHERE @GradeRep = USER_NAME() OR USER_NAME() = 'Manager';
GO
```

代码 3.8.5

结果如图 3.8.6 所示。

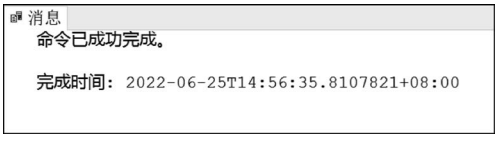


图 3.8.6

(6) 创建一个安全策略(将该函数添加为筛选器谓词)。状态必须设置为 ON 以启用该策略,如代码 3.8.6 所示。

```
CREATE SECURITY POLICY GradeFilter
ADD FILTER PREDICATE Security.tvf_security_predicate(Sno)
ON Grade.Grade.
WITH (STATE = ON);
GO
```

代码 3.8.6

结果如图 3.8.7 所示。

(7) 将 Security.tvf_security_predicate 的 SELECT 权限授予用户,如代码 3.8.7 所示。

```
GRANT SELECT ON Security.tvf_security_predicate TO Manager;
GRANT SELECT ON Security.tvf_security_predicate TO S001;
GRANT SELECT ON Security.tvf_security_predicate TO S005;
```

代码 3.8.7

结果如图 3.8.8 所示。

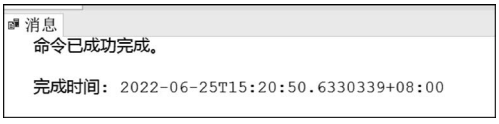


图　3.8.7

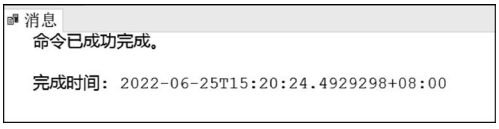


图　3.8.8

(8) 现在通过 3 个用户对 Grade 表进行查询来测试安全策略是否生效，如代码 3.8.8 所示。

```
EXECUTE AS USER = 'Manager';
SELECT * FROM Grade.Grade;
REVERT;

EXECUTE AS USER = '         S001';
SELECT * FROM Grade.Grade;
REVERT;

EXECUTE AS USER = 'S005';
SELECT * FROM Grade.Grade;
REVERT;
```

代码 3.8.8

管理员可以看到所有数据。S001 和 S005 用户应只能看到自己的成绩，如图 3.8.9 所示。

结果　消息

	Sno	Cno	Grade
1	S001	C001	92
2	S001	C005	89
3	S003	C001	77
4	S001	C002	88
5	S005	C002	73
6	S006	C005	86
7	S005	C006	87

	Sno	Cno	Grade
1	S001	C001	92
2	S001	C005	89
3	S001	C002	88

	Sno	Cno	Grade
1	S005	C002	73
2	S005	C006	87

图　3.8.9

3.9 综合案例

3.9.1 实验目的

通过完成一个综合案例的实验，加深对数据库安全性控制的理解，是对本章所涉及的数据库安全技术的一个复习和检测。

3.9.2 实验内容

问题：赵老师当了 2022 级电子商务班的班主任，他要能查到全校的课程信息以及本班学生的选课信息，如何让他有权查到这些信息？

主要内容如下：

1. 登录管理

为新老师创建登录账号 logzhao，验证该账号与数据库的连接访问是否正确。

2. 用户管理

对新老师的账号，创建用户 dbuserzhao，验证该用户与数据库的连接访问是否正确。

3. 对用户授权

问题 1：试解决赵老师能查询本年级学生的选课信息？

首先创建 2022 级学生选课信息的视图 scview，把访问该视图的权限授予赵老师，最后验证赵老师能否访问该视图。

问题 2：试解决让赵老师了解某课程的选课情况？

首先创建能查询指定课程选课信息的存储过程 scpro，把执行该存储过程的权限授予赵老师，最后验证赵老师能否执行存储过程。

补充内容：撤销赵老师查询某课程的选课情况，再验证赵老师能否执行存储过程。

4. 角色管理

问题：假如学校新增 10 个辅导员，都要在 STUDENTS 表中添加、修改和删除学生，要个个设置权限，这样方便吗？

可以考虑利用数据库的角色管理来实现：首先创建辅导员角色 m_role，然后授予该角色进行插入操作授权，再创建各个辅导员的登录以及对应的登录用户，使这些用户成为角色成员，再验证用户是否有插入操作的权限。

还可以考虑应用程序角色来实现：创建应用程序角色，激活该角色，对其进行插入操作

的授权，验证是否具有该操作的权限。

3.9.3　实验步骤

1. 登录管理

首先用管理员账号登录 SSMS，新建查询，创建新老师的登录账号，SQL 执行代码如代码 3.9.1 所示。

```
EXEC sp_addlogin 'logzhao','01'
```

代码 3.9.1

执行结果如图 3.9.1 所示。

验证账号是否可以登录，成功登录 SSMS 界面，如图 3.9.2 所示。

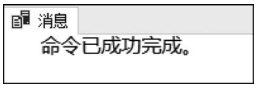


图　3.9.1

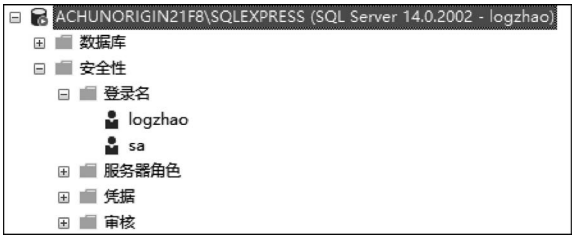


图　3.9.2

2. 用户管理

用管理员账号新建查询，创建新老师的登录用户名并映射到 logzhao 的登录名，在数据库 school 执行 SQL 代码，如代码 3.9.2 所示。

```
EXEC sp_grantdbaccess 'logzhao','dbuserzhao'
```

代码 3.9.2

执行结果同样如图 3.9.1 所示。

查看 logzhao 登录名的属性，选择“用户映射”选项页，可以在“映射到此登录名的用户”中找到 dbuserzhao 用户，用户创建成功，如图 3.9.3 所示。

3. 对用户授权

问题 1：解决赵老师能查询本年级学生的选课信息。

首先创建 2022 级学生选课信息的视图 scview，执行代码如代码 3.9.3 所示。

```
--在数据库增加 2022 级同学的信息
INSERT INTO STUDENTS VALUES ('202201001', 'Tom', 'Tom@163.com', 2022);
INSERT INTO STUDENTS VALUES ('202201002', 'Teddy', 'Teddy@163.com', 2022);
INSERT INTO STUDENTS VALUES ('202201003', 'Joda', 'Joda@126.com', 2022);
INSERT INTO STUDENTS VALUES ('202201004', 'Keddy', 'Keddy@sioza.com', 2022);
INSERT INTO STUDENTS VALUES ('202201005', 'Jerry', 'Jerry@163.com', 2022);
```

```
INSERT INTO CHOICES VALUES (50001, '202201001', '200003125', '10001', 63);
INSERT INTO CHOICES VALUES (50002, '202201001', '200005322', '10002', 74);
INSERT INTO CHOICES VALUES (50003, '202201002', '200003125', '10001', 83);
INSERT INTO CHOICES VALUES (50004, '202201002', '200005322', '10002', NULL);
INSERT INTO CHOICES VALUES (50005, '202201003', '200003125', '10001', NULL);
INSERT INTO CHOICES VALUES (50006, '202201003', '200005322', '10002', 83);
INSERT INTO CHOICES VALUES (50007, '202201004', '200003125', '10001', 61);
INSERT INTO CHOICES VALUES (50008, '202201004', '200005322', '10002', 99);
INSERT INTO CHOICES VALUES (50009, '202201005', '200003125, '10001', 77);
-- 新建查询,创建 2022 级学生选课信息的视图 scview
create view scview as
Select sc.sid,s.sname,sc.cid,s.grade from CHOICES sc,students s
Where sc.sid = s.sid and s.grade = '2022'
```

代码 3.9.3

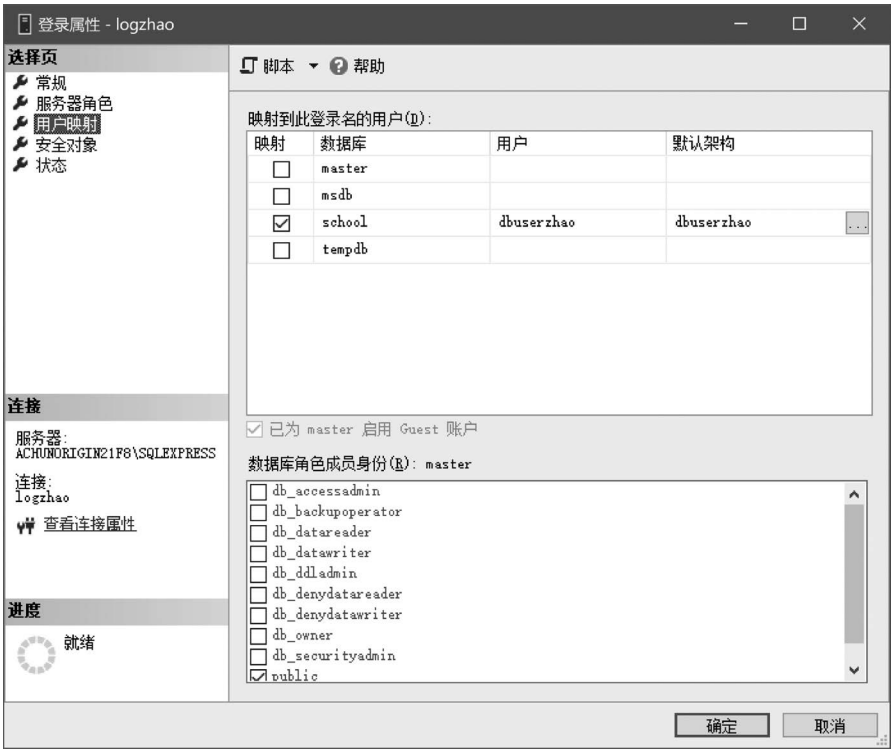


图 3.9.3

然后把访问该视图的权限授予赵老师,执行代码如代码 3.9.4 所示。

```
GRANT SELECT ON scview TO dbuserzhao
```

代码 3.9.4

授权成功结果如图 3.9.1 所示。

再用 logzhao 登录 SSMS,对 school 数据库新建查询,查询视图 scview,查询执行结果如图 3.9.4 所示。

问题 2:解决让赵老师了解某课程的选课情况。

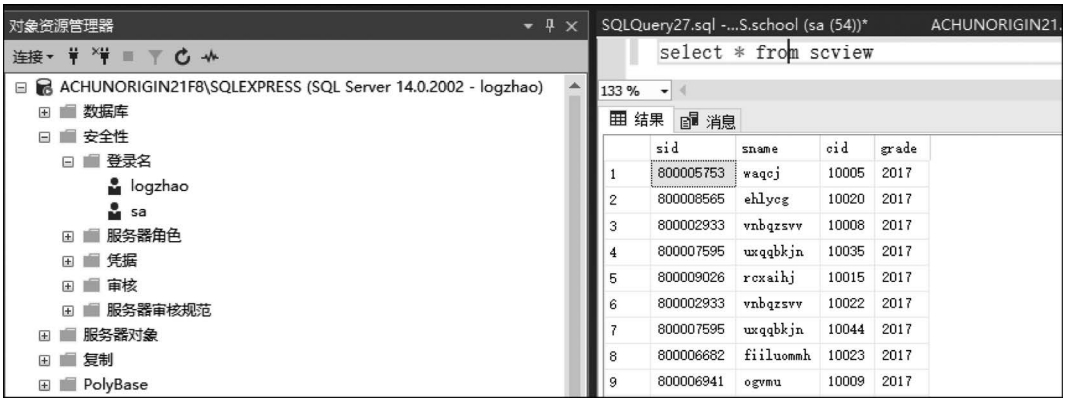


图　3.9.4

首先创建能查询指定课程选课信息的存储过程 scproc，SQL 执行代码如代码 3.9.5 所示。

```
CREATE PROC scproc @kc char(10) AS
SELECT sc.sid,sc.cid,courses.cname FROM CHOICES SC,COURSES
WHERE sc.cid = courses.cid AND courses.cname = @kc
```

代码 3.9.5

然后把执行该存储过程的权限授予赵老师，SQL 执行代码如代码 3.9.6 所示。

```
GRANT EXEC ON scproc TO dbuserzhao
```

代码 3.9.6

最后再用 logzhao 登录 SSMS，对 school 数据库新建查询，调用存储过程 scproc 进行查询，执行结果如图 3.9.5 所示。

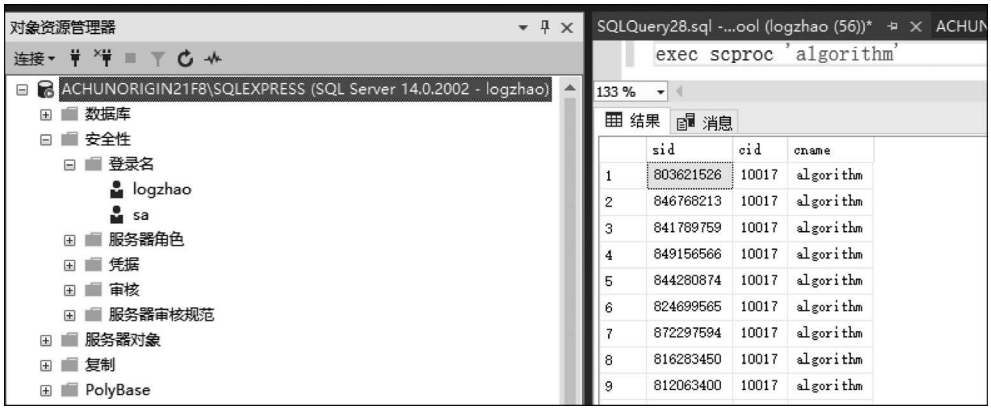


图　3.9.5

补充问题：撤销赵老师查询某课程的选课情况，SQL 执行代码如代码 3.9.7 所示。

```
REVOKE EXEC ON scproc FROM dbuserzhao
```

代码 3.9.7

最后再用 logzhao 登录 SSMS，对 school 数据库新建查询，调用存储过程 scproc 进行查

询，执行结果如图 3.9.6 所示。

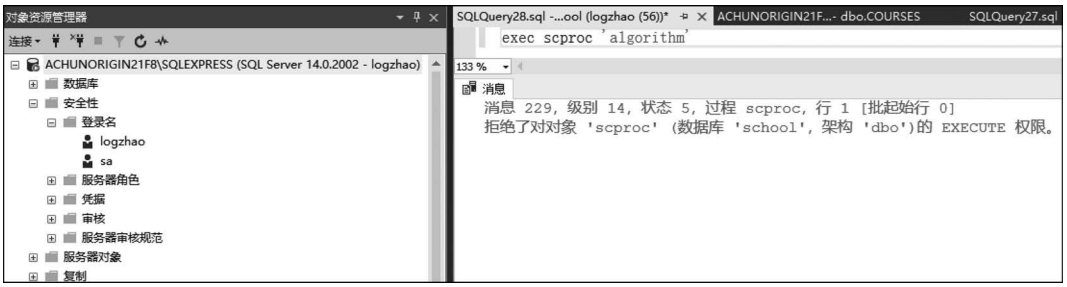


图 3.9.6

4. 角色管理

首先创建辅导员角色，SQL 语句如代码 3.9.8 所示。

```
EXEC sp_addrole 'm_role'
```

代码 3.9.8

再对 m_role 角色进行插入、修改、删除的授权，SQL 语句如代码 3.9.9 所示。

```
GRANT INSERT,UPDATE,DELETE ON STUDENTS TO m_role
```

代码 3.9.9

创建两个辅导员的登录 logteach1 和 logteach2 以及对应的用户 dbuser1 和 dbuser2，SQL 语句如代码 3.9.10 所示。

```
EXEC sp_addlogin 'logteach1','01';
EXEC sp_addlogin 'logteach2','02';
EXEC sp_grantdbaccess 'logteach1','dbuser1';
EXEC sp_grantdbaccess 'logteach2','dbuser2';
```

代码 3.9.10

将用户分配到辅导员的角色 m_role，SQL 语句如代码 3.9.11 所示。

```
EXEC sp_addrolemember'm_role','dbuser1';
EXEC sp_addrolemember'm_role','dbuser2';
```

代码 3.9.11

最后再用 logteach1 与 logteach2 分别登录 SSMS，对 school 数据库新建查询，插入测试数据，执行结果如图 3.9.7 与图 3.9.8 所示。

还可以通过创建应用程序角色，首先用管理员账号登录管理工具，新建查询，创建一个应用程序角色 r_p_test，密码为 abc，授予角色 r_p_test 对 STUDENTS 表的 SELECT 权限，SQL 语句如代码 3.9.12 所示。

```
EXEC sp_addapprole'r_p_test','abc';
GRANT SELECT ON STUDENTS TO r_p_test;
```

代码 3.9.12

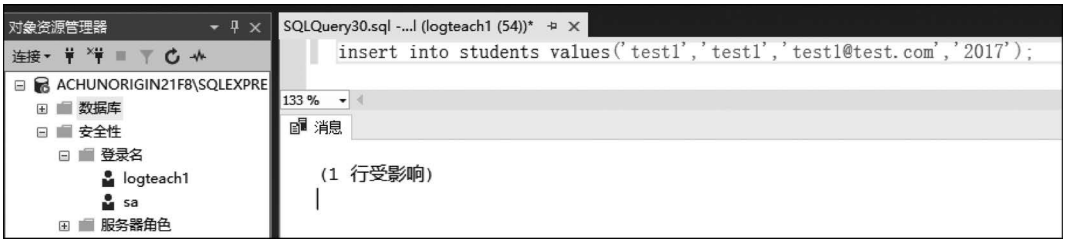


图　3.9.7

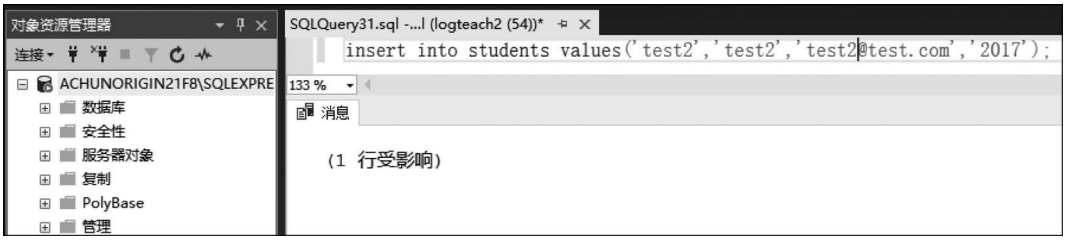


图　3.9.8

创建完毕用 logteach2 账号登录，由于先前该账号所属角色没有对表 STUDENTS 的查询权限，现在通过激活应用程序角色来使其获得对 STUDENTS 表的查询权限，激活与测试语句，如代码 3.9.13 所示。

```
--用密码 abc 激活 r_p_test 应用程序角色，并且在将此密码发送到 SQL Server 之前对其加密
DECLARE @cookie varbinary(8000);
EXEC sp_setapprole 'r_p_test','abc',@fCreateCookie = true,@cookie =
@cookie OUTPUT;
SELECT * FROM STUDENTS;
SELECT USER_NAME();
```

代码 3.9.13

执行结果如图 3.9.9 所示。

结果　消息

	sid	sname	email	grade
1	800001216	gfxrgs	hhce4@qhldj.gov	2017
2	800002933	vnbqzsvv	pvhxd4l@zqur.org	2017
3	800005753	waqcj	hlhq0h8@jdba.gov	2017
4	800006682	fiiluommh	ihzd6_k@krvft.gov	2017
5	800006941	ogvmu	62sfbd@lrt.gov	2017
6	800007595	uxqqbkjn	cr8g@zrvgt.edu	2017
7	800008565	ehlycg	nach10@uic.com	2017
8	800009026	rcxaihj	4ul4kqb@hko.edu	2017
9	800009099	zapyv	jqmqn8@iwaiu.org	2017
10	800009249	zyuoh	8enjrcu@upfw.org	1991
11	800010666	uwphrw	emb7k@ipp.com	1992

	(无列名)
1	r_p_test

图　3.9.9

可以发现，该账号已经可以对表 STUDENTS 进行查询操作。

3.10 本章自我实践参考答案

3.1.5 节自我实践参考答案

以系统管理员或 sa 用户登录进入查询分析器，执行下面的代码。

(1)
```
EXEC sp_addlogin '王二','123','school'
GO
USE school
GO
EXEC sp_grantdbaccess '王二'
```

(2)
```
USE school
EXEC sp_revokeaccess '王二';
EXEC sp_droplogin '王二'
```

3.2.5 节自我实践参考答案

123456 是新增用户的登录密码。Li 登录账户将映射为数据库用户名 happyrat，这是由第 3 行代码的存储过程决定的，将是 school 数据库的用户。

第 4 行的作用是把 STUDENTS 表上的 SELECT、INSERT、UPDATE 3 种权限授予全体用户。第 5 行的作用是把 STUDENTS 表的所有权限授予 happyrat 用户。第 6 行的作用是取消 happyrat 用户在 STUDENTS 表上的 SELECT 权限。第 7 行的作用是拒绝 happyrat 用户在 STUDENTS 表上的 UPDATE 操作。

若以账户 Li 登录数据库，则可以对 school 数据库的表 STUDENTS 进行 SELECT 操作，但是 UPDATE 操作不行，因为 UPDATE 操作被 DENY 了，而 happyrat 用户的 SELECT 权限虽然被 REVOKE，但作为 public 用户的 SELECT 权限仍在，所以可以进行 SELECT 操作。如果要使该用户不能进行 SELECT 操作，则可以用 DENY 或者对 public 的 SELECT 权限进行 REVOKE 操作。

3.3.5 节自我实践参考答案

(1) 打开查询分析器，用 sa 账号登录数据库。

```
USE school
GO
CREATE VIEW grade2000 AS
SELECT * FROM STUDENTS WHERE grade = '2000';
```

(2)

```
USE school
GO
GRANT SELECT ON grade2000 TO 王二
```

3.4.5 节自我实践参考答案

从实验中可以看出，授权给 public 是将对数据对象的操作权限授予所有用户，而授权给指定用户会更安全一些，指明了授权的对象，避免了某些操作不经意的授给所有其他用户。

3.5.5 节自我实践参考答案

由于系统采用了 Schema 架构，在建立账户 user1 的同时，为该账户分配了 db_owner 的架构和角色成员，而系统管理员默认为 dbo 的架构，所以可以成功的建立两个相同表 a，其中一个是 dbo.a，另一个是 db_owner.a。

如果不分配 db_owner，系统在建表时会拒绝操作，不可以建立两个相同名称的表 a。

3.6.5 节自我实践参考答案

本次实验中，可以直接采用对称密钥对数据进行加密，虽然直接加密相比借助数字证书的方法简单方便且系统开销小，但其安全性不是很高，像 pwdencrypt 语句对数据加密会产生不可逆的结果，给数据库的使用带来不便。

3.7.5 节自我实践参考答案

此时利用 SELECT USER_NAME 语句所查询的数据库用户名，查询结果应返回 u_test，因为在激活应用程序角色之前，“EXEC sp_grantdbaccess "l_test","u_test""语句为登录 l_test 在数据库 school 中添加默认的安全账户是 u_test。