第5章 数据库管理与维护

数据库管理系统(DBMS)是一种操纵和管理数据库的系统软件,用于建立、使用、管理和维护数据库。一旦数据库创建以后,DBMS 便对数据库中的数据进行统一的管理和控制,以确保用户在共享环境中能合法访问数据,防止数据出错、意外丢失,以及数据库遭到破坏后能迅速恢复正常。因此 DBMS 必须提供数据库管理与维护功能,以保证数据库中数据的正确有效和安全可靠。数据库管理与维护包括数据库的安全性管理、数据库中数据的并发控制和数据库的备份及恢复管理。

5.1 安全性管理


5 安全性管理

数据库的安全性管理是对数据库采取的一种保护措施。安全性管理是指保护数据库,防止非法使用,以避免非法用户对其进行窃取数据、篡改数据、删除数据和破坏数据库结构等操作。SQL Server 提供了强大的、内置的安全性和数据保护,以防止非法用户对数据库进行操作,保证数据库的安全。

5.1.1 SQL Server 数据库的安全机制

SQL Server 整个安全体系中包括认证和授权两大部分。当用户要访问 SQL Server 数据库中的数据时,必须通过三个级别的认证。首先是第一个级别认证——服务器级别的认证,即身份验证,需要通过登录 SQL Server 的登录账户和密码来验证该用户是否具有连接 SQL Server 数据库服务器的权限,在身份验证时,SQL Server 和 Windows 是组合在一起的,因此 SQL Server 提供了两种确认用户身份的验证模式,即 Windows 验证模式和混合验证模式。接着是第二个级别认证——数据库级别的认证,当用户访问数据库时,必须具备对具体数据库的访问权限,即验证用户是否是数据库的合法用户。最后是第三个级别认证——数据库对象级别的认证,当用户操作数据库中的数据对象时,必须具备相应的操作权,即验证用户是否具有操作数据对象的权限。后面章节将进行详细探讨。

5.1.2 服务器登录名

服务器登录名是服务器级别的认证,主要是进行身份验证,也指登录账号的验证。登录 SQL Server 访问数据库的用户,必须要有一个能登录 SQL Server 服务器的账号和密码,只有以该账号和密码通过 SQL Server 数据库服务器验证后才能连接上服务器,然后进行数据库访问,否则服务器将拒绝用户登录,从而确保系统的安全性。SQL Server 提供了 Windows 验证和混合验证两种身份验证模式,每一种身份验证都有一个不同类型的登录名。

1. 身份验证模式

Windows 验证模式会启用 Windows 身份验证并禁用 SQL Server 身份验证，而混合验证模式会同时启用 Windows 身份验证和 SQL Server 身份验证。因此，Windows 验证始终可用，并无法禁用。

1）Windows 验证模式

SQL Server 数据库管理系统通常运行在 Windows 上，而 Windows 本身就能够管理登录、验证用户的合法性，因此 SQL Server 的 Windows 验证模式正是利用了这一用户安全性和账号管理的机制。只要能够访问操作系统，Windows 验证模式就认为用户合法，因此这种模式只适用于能够提供有效身份验证的 Windows 操作系统。该模式下的 Windows 用户不需要独立的 SQL Server 账号和密码就可以访问数据库，但用户需要遵从 Windows 本身安全模式的所有规则，还可以用这种模式去锁定账户、审核登录和迫使用户周期性地更改登录密码。

Windows 验证模式是一种默认且比较安全的身份验证模式。由 Windows 系统提供身份验证而完成的连接也称为信任连接。Windows 验证模式有如下优点。

（1）由于 Windows 系统完成对用户账号的管理，因此数据库管理员可以集中进行数据库管理。

（2）Windows 系统拥有较强的用户账号管理工具，包括账户锁定、密码期限等。如果不通过定制来扩展 SQL Server，SQL Server 则不具备这些功能。

（3）Windows 拥有用户组管理策略，可以通过 Windows 对用户进行集中管理，针对一组用户同时设置访问 SQL Server 的权限。

2）混合验证模式

混合验证模式是指用户可以使用 Windows 身份验证和 SQL Server 身份验证进行服务器连接。如果不是 Windows 操作系统的用户或者是 Windows 客户端操作系统的用户使用 SQL Server，则应该选择混合验证模式。当采用混合验证模式时，SQL Server 首先确定用户的连接是否使用了有效的 SQL Server 用户账户和正确的密码，如果是有效登录，则接受用户的连接，如果用户使用有效的登录账号，但不是正确的密码，则拒绝用户的连接。当且仅当用户没有有效的登录时，SQL Server 才检查 Windows 账户的信息，在这种情况下，SQL Server 就会确定 Windows 账户是否有连接到服务器的权限，如果账号有权限，连接被接受，否则连接被拒绝。

使用 SQL Server 创建的登录账号和密码用于 SQL Server 身份验证，这些信息将存储在 SQL Server 中。通过混合验证模式进行连接的用户每次连接时必须提供登录账号和密码。混合验证模式有如下优点。

（1）是在 Windows 系统之上创建的另一个安全层次。

（2）支持除了 Windows 用户以外的更大范围的用户连接数据库服务器。

（3）一个应用程序可使用单独的 SQL Server 登录账号或密码。

2. 设置身份验证模式

在安装 SQL Server 数据库管理系统时，必须为数据库引擎选择身份验证模式：Windows 验证模式或混合验证模式。如果在安装时选择混合身份验证模式，则必须为名为 sa 的内置 SQL Server 系统管理员账号提供一个密码并确认该密码，以后就可以通过 sa 账

号进行 SQL Server 身份验证来连接服务器。在打开 SQL Server 连接服务器时，需要指定验证模式，这与 SQL Server 安装时所选择的验证模式有关。对于已经指定验证模式的 SQL Server 服务器，在 SQL Server 数据库管理系统中可以进行修改。

在 SQL Server Management Studio 中设置验证模式的步骤如下。

(1) 打开 SQL Server Management Studio，在“对象资源管理器”中的目标服务器上右击，在弹出的快捷菜单中，选择“属性”命令。

(2) 出现“服务器属性”界面，选择“选择页”中的“安全性”选项，进入安全性设置页面，如图 5-1 所示。

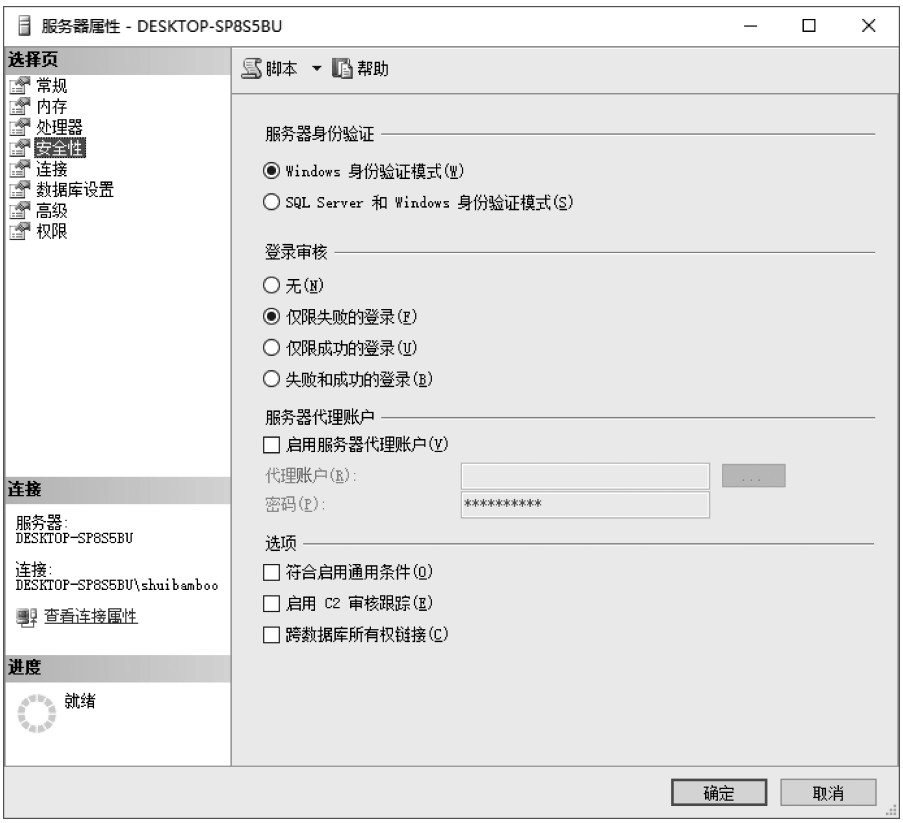


图 5-1 “服务器属性”界面

(3) 在“服务器身份验证”选项中，选中需要的验证模式。可以在“登录审核”选项中设置需要的审核方式。审核方式取决于安全性要求。这 4 种审核级别的含义如下。

① 无：不使用登录审核。

② 仅限失败的登录：记录所有的失败登录。

③ 仅限成功的登录：记录所有的成功登录。

④ 失败和成功的登录：记录所有的登录。

(4) 单击“确定”按钮，完成登录验证模式的设置。

3. 登录账号

SQL Server 提供两种方式进行身份验证，登录账号的管理也有两种方式：一种基于

Windows 用户或组来管理，另一种是使用 SQL Server 来管理。这里仅介绍 SQL Server 管理登录账号。

1）查看登录账号

可以使用 SQL Server Management Studio，通过对象资源管理器查看登录账号，也可以使用存储过程和查询视图来查看。

（1）使用 SQL Server Management Studio。

打开 SQL Server Management Studio，在“对象资源管理器”中选择数据库服务器，再选择“安全性”→“登录名”，可看到现有的登录账号，如图 5-2 所示。双击某个具体的登录账号，显示登录属性页面，可看到该登录账号的基本信息，也可以单击“服务器角色”查看是否是某个服务器角色成员，单击“用户映射”可查看是否有相应的数据库用户，单击“状态”可查看该用户是否允许连接数据库等信息。

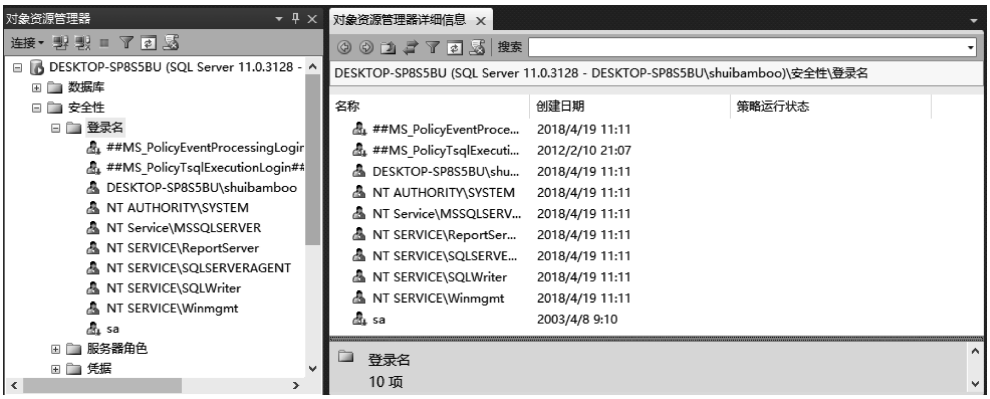


图 5-2　查看登录账号

（2）使用存储过程。

存储过程 sp_helplogins 可以查看登录账号，其语法格式如下。

```
EXECUTE sp_helplogins ['login']
```

login 指登录账号名，如果没有指定具体的登录账号，则查询所有登录账号。

（3）查询视图。

查询视图 sys.syslogins 可以查看登录账号。

```
SELECT * FROM sys.syslogins;
```

2）创建登录账号

可以使用 SQL Server Management Studio 和 SQL 语句来创建登录账号。

（1）使用 SQL Server Management Studio。

打开 SQL Server Management Studio，在“对象资源管理器”中选择数据库服务器，然后展开“安全性”→“登录名”，右击选择“新建登录名”，如图 5-3 所示。

在弹出的界面（图 5-4）中输入登录账号信息。“登录名”处输入新建登录名，如果选择“SQL Server 身份验证”，输入密码和确认密码，然后单击“默认数据库”下拉列表框来设置登录账号默认访问的数据库。如果选择“Windows 身份验证”，“登录名”处需要填写系统登

图 5-3　创建登录账号

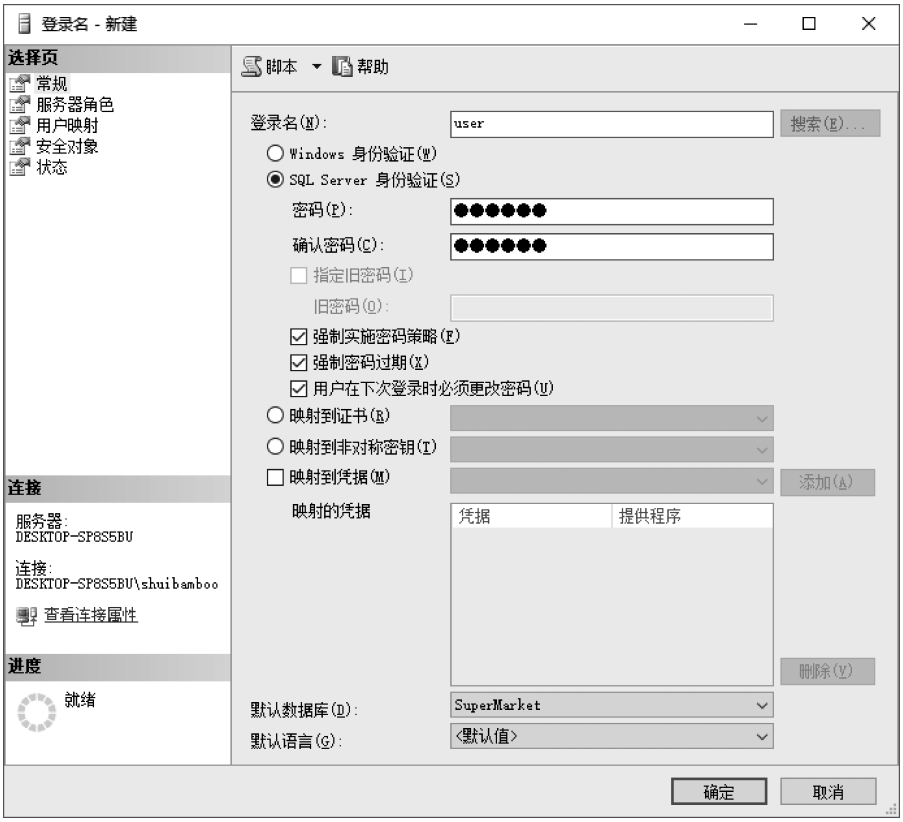


图 5-4　设置登录账号

录用户名，这时不需要设置登录密码，但仍需选择默认数据库。

（2）使用 SQL 语句。

用 SQL 语句创建登录账号的语法格式如下。

```
CREATE LOGIN login_name
FROM WINDOWS | WITH PASSWORD = 'password'[MUST_CHANGE]
[, DEFAULT_DATABASE  = database];
```

各参数说明如下。

FROM WINDOWS：指定创建的登录账号是 Windows 身份验证，login_name 是系统的登录用户名或用户组。

WIHT PASSWORD：指定登录账号的密码，这是 SQL Server 身份验证。

MUST_CHANGE：指定首次使用时需修改密码。

DEFAULT_DATABASE：指定默认数据库。

【例 5-1】 创建一个登录账户 user1，密码为 user1，默认数据库 SuperMarket。

```
CREATE LOGIN user1
WITH PASSWORD = 'user1', DEFAULT_DATABASE = SuperMarket;
```

3）修改登录账号

修改登录账号主要是对登录密码、默认数据库、启用和禁用登录、登录账号和解锁登录进行修改。可以使用 SQL Server Management Studio 的图形界面进行修改，操作方式与创建登录账号类似，这里不再赘述。下面介绍使用 SQL 语句来进行修改，其语法格式如下。

```
ALTER LOGIN login_name
WITH PASSWORD = 'password'| WITH DEFAULT_DATABASE = database |
ENABLE | DISABLE | NAME = login_name | UNLOCK
```

各参数说明如下。

ENABLE：启用登录账号。

DISABLE：禁用登录账号。

UNLOCK：解锁登录账号。

【例 5-2】 修改登录账户 user1，密码为 password1。

```
ALTER LOGIN user1
WITH
PASSWORD = ' password1';
```

4）删除登录账号

当不再需要某个登录账号时，可以将其删除。删除登录账号时，数据库中与登录账号对应的数据库用户不会被删除。

删除登录账号可直接在 SQL Server Management Studio 的对象资源管理器中，右击需要删除的登录账号，选择“删除”命令，如图 5-5 所示。

也可以使用 SQL 语句来进行删除，其语法格式如下。

```
DROP LOGIN login_name
```

【例 5-3】 删除登录账户 user1。

```
DROP LOGIN user1;
```

图 5-5　删除登录账号

5.1.3　数据库用户

数据库用户是数据库级别的认证。数据库的安全性主要是通过数据库用户账户进行控制，要想访问一个数据库，必须拥有该数据库的一个用户账户身份，通常该用户账户是在登录服务器时通过登录账号进行映射的。因此，连接上 SQL Server 服务器后，用户还需要以一个数据库用户账户及对应的权限访问该数据库中的数据。

在例 5-1 中创建的登录名“user1”，它可以通过身份验证连接到 SQL Server 数据库服务器上，但该登录账户还不具备访问数据库的条件，除非为该登录账号映射了相应的数据库用户。

服务器通过数据库用户对数据库访问权限进行设置。数据库系统管理员可以自定义数据库用户，并可以设置权限。数据库用户是一个或多个登录对象在数据库中的映射，可以对数据库用户对象进行授权，以便为登录对象提供对数据库的访问权限。用户定义信息存放在每一个数据库的 sysuser 表中。一个服务器登录账号可以被授予访问多个数据库（多个数据库用户），但一个登录名在每个数据库中只能映射一次。如果未对一个登录账号指定数据库用户，则登录时系统将试图将该登录账号映射成 guest 用户，如果还是失败的话，该用户将无法访问数据库。

1. 默认数据库用户

SQL Server 系统中有 dbo 用户、sys 用户和 guest 用户等默认数据库用户。

1）dbo 用户

dbo 全称 Database Owner，是每个数据库的默认用户，它具有在数据库中执行所有活动的权限。一般来说，创建数据库的用户就是创建数据库的所有者。可以通过 dbo 将它拥有的权限授予其他用户。因为“sysadmin”服务器角色的成员被自动映射为 dbo 用户，所以 sysadmin 角色登录可执行 dbo 能执行的任何任务。在 SQL Server 数据库中创建的对象也是所有者，这些所有者是指数据库对象所有者。通过 sysadmin 服务器角色成员创建的对象自动属于 dbo 用户。通过非 sysadmin 服务器角色成员创建的对象属于创建对象的用户，当

其他用户引用它们时必须以用户的名称来限定。例如，如果 ROLE1 是 sysadmin 服务器角色成员，并创建了一个名为 sale 的表，则 sale 表属于 dbo，所以用 dbo. sale 来限定，或者简化为 sale。但如果 ROLE1 不是 sysadmin 服务器角色成员，创建了一个名为 sale 的表，则 sale 表属于 ROLE1，所以用 ROLE1. sale 来限定。

2）sys 用户

sys 用户是包含系统对象的架构。事实上，所有系统对象包含在 sys 或 information_schema 的架构中。这是构建在每一个数据库中的两个特殊架构，它们仅在 Master 数据库中可见。相关的 sys 和 information_schema 架构的视图提供存储在数据库里所有数据对象的元数据的内部系统视图。这些视图被 sys 和 information_schema 用户所引用。

3）guest 用户

guest 用户是一个允许具有有效 SQL Server 登录的任何用户访问数据库的一个特殊用户。以 guest 账户访问数据库的用户被认为是拥有 guest 用户的身份并继承了 guest 账户的所有权限和许可。在默认情况下，guest 用户存放在 Model 数据库中，并且被授予 guest 账户的权限。由于 Model 是创建所有数据库的模板，所以所有新的数据库都包含 guset 用户，并且该用户被授予 guest 账户的权限。需要注意的是，只能在 Master 和 Tempdb 之外的所有数据库中添加或删除 guest 用户。

2. 查看数据库用户

可以使用 SQL Server Management Studio，通过对象资源管理器查看数据库用户，也可以使用存储过程和查询视图来查看。

1）使用 SQL Server Management Studio

打开 SQL Server Management Studio，在“对象资源管理器”中选择数据库服务器，然后选择“数据库”，在数据库列表中选择要查看的数据库，然后选择要查看数据库下的“安全性”→“用户”命令就可以看到当前数据库中的用户，如图 5-6 所示。

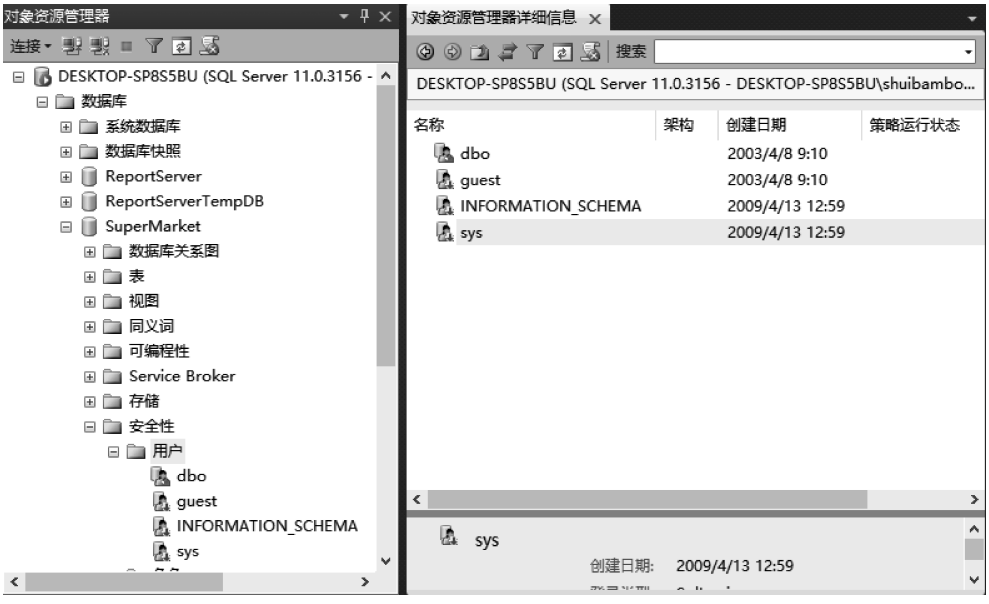


图 5-6　SuperMarket 的数据库用户

2）使用存储过程

存储过程 sp_helpuser 可以查看数据库用户，其语法格式如下。

```
EXECUTE sp_helpuser ['user_name']
```

user_name 指数据库用户名，如果没有指定具体的数据库用户，则查询所有数据库用户。

3）查询视图

查询视图 sys.sysusers 可以查看数据库用户。

```
SELECT * FROM sys.sysusers;
```

3. 创建数据库用户

可以使用 SQL Server Management Studio 和 SQL 语句来创建数据库用户。

1）使用 SQL Server Management Studio

打开 SQL Server Management Studio，在“对象资源管理器”中选择数据库服务器，然后选择“数据库”，在具体数据库下选择“安全性”→“用户”，右击选择“新建用户”命令，弹出如图 5-7 所示的页面，在该页面中设置数据库用户信息。“用户名”处输入新建用户名，用户名可以和登录名不同，接着选择对应的登录名，还需要给用户选择一个默认的架构（架构指数据库对象的集合。SQL Server 2012 中架构是独立的，和用户没有对应关系。因此可先创建一个架构，或直接选择一个已存在的架构），最后单击“确定”按钮，完成数据库用户的创建。

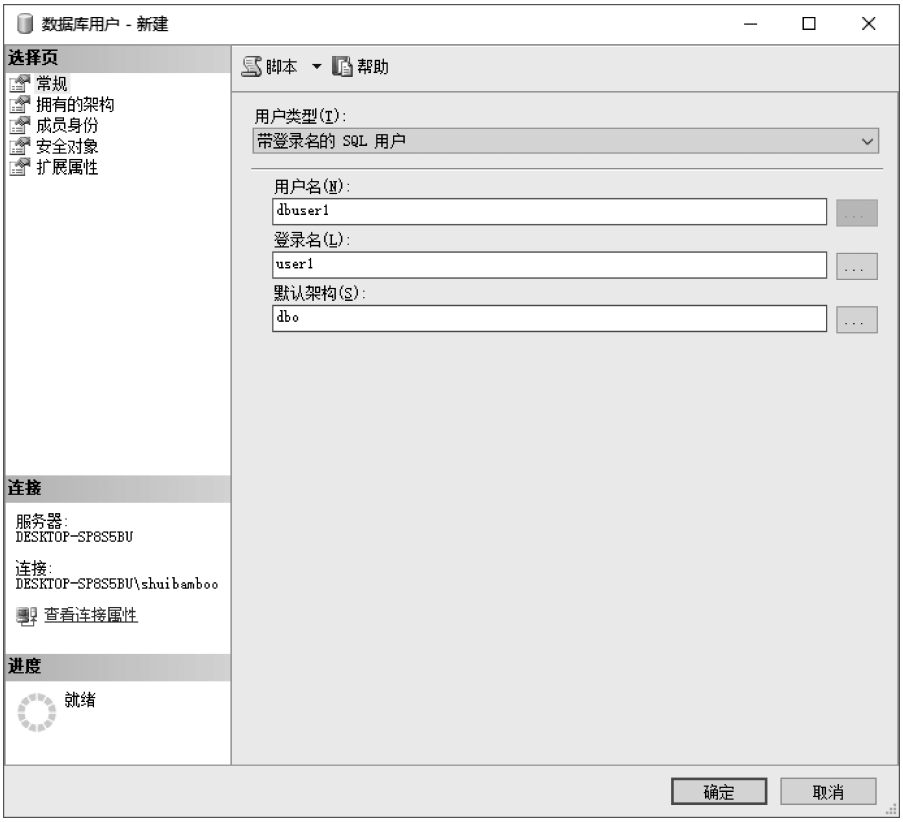


图 5-7　创建数据库用户

2）使用 SQL 语句

用 SQL 语句创建数据库用户的语法格式如下。

```
CREATE USER user_name
    FOR | FROM  LOGIN login_name
    [WITH DEFAULT_SCHEMA = schema_name];
```

各参数说明如下。

FOR | FROM LOGIN：指定数据库用户对应的登录账号。

WIHT DEFAULT_SCHEMA：指定数据库用户使用的架构。

【例 5-4】 为数据库 SuperMarket 创建一个数据库用户 dbuser2，对应的登录账号为 user1，架构为 dbo。

```
USE SuperMarket
GO
CREATE USER dbuser2
FROM LOGIN user1
WITH DEFAULT_SCHEMA = dbo;
```

4. 删除数据库用户

当不再需要某个数据库用户时，可将其删除。但删除数据库用户时，其对应的架构不会删除，即用户创建过的对象会得以保存。

删除数据库用户可直接在 SQL Server Management Studio 的“对象资源管理器”中，展开对应“数据库”→“安全性”→“用户”，在用户列表中找到要删除的数据库用户，在该用户上右击选择“删除”命令，便可完成删除操作，如图 5-8 所示。

图 5-8　删除数据库用户

也可以使用命令 DROP USER 语句进行删除，其语法格式如下。

```
DROP USER user_name
```

【例 5-5】 删除数据库用户 dbuser2。

```
DROP USER dbuser2;
```

5.1.4 角色管理

角色是 SQL Server 用来集中管理服务器或数据库的权限，用于为用户组分配权限。若用户被加入到某一个角色中，该用户就具备该角色的所有权限。所以，SQL Server 数据库管理员只对角色进行权限设置便可以实现对所有用户权限的设置，大大减少了管理员的工作量。SQL Server 提供了两类角色：服务器角色和数据库角色。

1. 服务器角色

服务器角色具有授予服务器管理的能力。用户创建了一个角色成员的登录，用户用这个登录能执行这个角色许可的任何任务。服务器角色属于服务器级别，因此其权限影响整个服务器。服务器角色是预先定义的，不能被添加、修改或删除，所以服务器角色又称为“固定服务器角色”或“预定义服务器角色”。

固定服务器角色独立于各个数据库，具有固定的权限，不能修改。固定服务器角色的权限范围是实例，不是具体的数据库，角色成员是登录账号。例如，如果某登录账号是 sysadmin 角色成员，则该登录账号可管理任何数据库，并自动映射到数据库的 dbo 用户。

1）常用的固定服务器角色

常用的固定服务器角色见表 5-1。

表 5-1 常用的固定服务器角色

角 色 名	角 色 权 限
sysadmin	系统管理员，可以在服务器中执行任何活动
serveradmin	服务器管理员，有设置和关闭对象服务器的权限
setupadmin	设置管理员，可以添加、删除和配置连接服务器，并能执行某些系统存储过程
securityadmin	安全管理员，可以管理登录账号、密码等
diskadmin	可以管理系统磁盘文件
processadmin	进程管理员，可以管理运行的进程
dbcreator	数据库创建者，可以创建、更改、删除或还原任何数据库
bulkadmin	可以执行批量插入语句 BULK INSERT 语句

2）为登录账号添加或删除固定服务器角色

通常可以通过界面和存储过程两种方法来为登录账号添加或删除固定服务器角色。使用界面时又可以通过两种方法来操作：一种是通过修改登录账号属性，选择服务器角色；另一种是打开服务器角色属性，然后选择成员。

打开 SQL Server Management Studio，在“对象资源管理器”中选择数据库服务器，然后选择“安全性”，在“登录名”下双击登录账号，在“登录属性”窗口中单击“选择页”的“服务器角色”，接着在右侧的“服务器角色”列表中选择角色，如图 5-9 所示。

展开“安全性”→“服务器角色”，双击要添加成员的角色，在角色成员列表下方通过“添加”或“删除”按钮来调整服务器角色的成员，如图 5-10 所示。

使用存储过程也可以完成以上操作。添加成员的语法如下。

```
EXECUTE sp_addsrvrolemember [@loginname] 'login',[@rolename] 'role'
```

删除成员的语法如下。

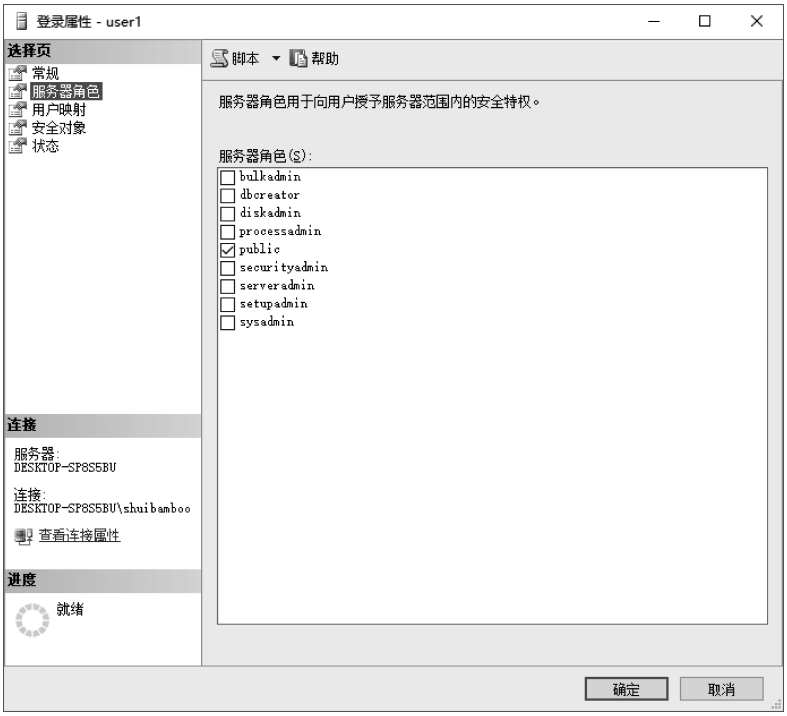


图 5-9 “登录属性”窗口设置固定服务器角色

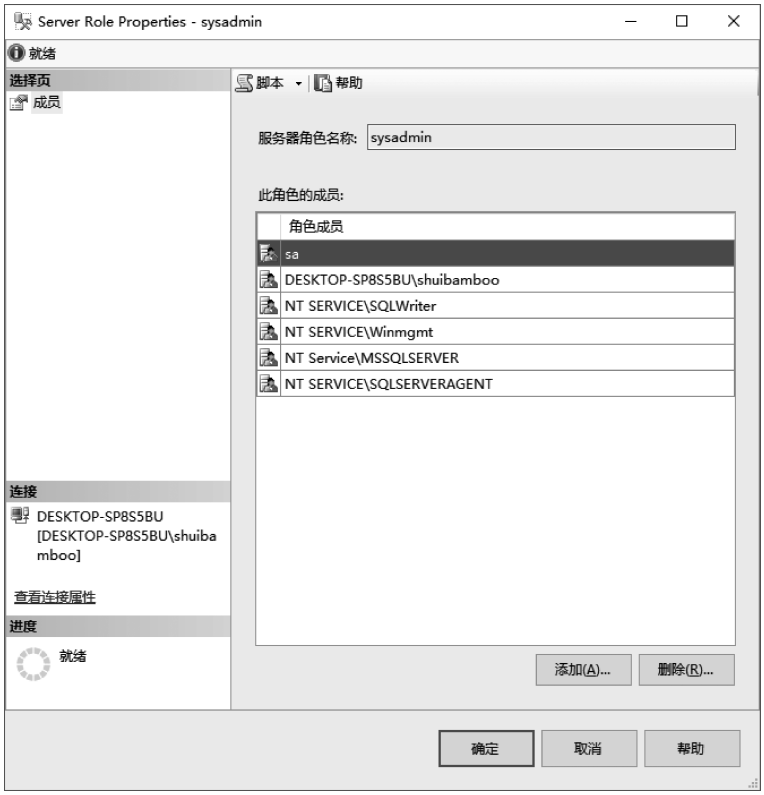


图 5-10 服务器角色属性窗口设置固定服务器角色成员

```
EXECUTE sp_dropsvrrolemember [@loginname] 'login',[@rolename] 'role'
```

参数说明如下。

login：指定登录账号。

role：指定服务器角色。

3）查看固定服务器角色信息

查看固定服务器角色列表：EXECUTE sp_helpsrvrole['role']

查看固定服务器角色权限：EXECUTE sp_srvrolepermission['role']

查看固定服务器角色成员：EXECUTE sp_helpsrvrolememeber['role']

2. 数据库角色

一旦创建了数据库用户，接下来需要管理这些用户的权限。数据库角色是某一个用户或一组用户授予不同级别的管理或访问数据库以及数据库对象的权限，这些权限是数据库专用的，并且可以使一个数据库用户具有属于同一数据库的多个角色。SQL Server 提供了两种类型的数据库角色：固定数据库角色和用户自定义数据库角色。

1）固定数据库角色

固定数据库角色是指 SQL Server 已经定义了这些角色所具有的管理、访问数据库的权限，而且 SQL Server 数据库管理员不能对其所具有的权限进行任何修改。SQL Server 每一个数据库中都有一组固定的数据库角色，在数据库中使用固定的数据库角色可以将不同级别的数据库管理工作分配给不同的角色，从而有效地实现工作权限的传递。

（1）常用的固定数据库角色。

SQL Server 提供了 10 种常用的固定数据库角色来授予数据库用户权限，具体内容见表 5-2。

表 5-2　常用的固定数据库角色

角　色　名	角 色 权 限
db_owner	数据库所有者，可以执行数据库的所有配置和维护活动
db _accessadmin	数据库访问权限管理者，可以增加或删除数据库用户、工作组和角色
db _ddladmin	数据库 DDL 管理员，可以在数据库中运行任何数据定义语言命令
db _securityadmin	可以修改角色成员身份和管理权限
db _backupoperator	可以备份和恢复数据库
db _datareader	仅能对数据库中任何表执行 select 操作，从而读取所有数据表的信息
db _datawriter	能够增加、修改和删除表中的数据，但不能进行 select 操作
db _denydatareader	不能读取数据库中任何表的数据
db _denydatawriter	不能对数据库中任何表执行增加、修改和删除数据的操作
public	每个数据库用户都属于 public 数据库角色，当尚未对某个用户授予或拒绝对安全对象的特定权限时，则该用户将继续授予该安全独享的 public 角色的权限。不能将用户从 public 角色中移除

（2）为数据库用户添加或删除固定数据库角色。

通常可以通过界面和存储过程两种方法来为数据库用户添加或删除固定数据库角色。使用界面时又可以通过两种方法来操作：一种是通过修改数据库用户属性，选择数据库角色。另一种是打开数据库角色属性，然后选择成员。

打开 SQL Server Management Studio，在“对象资源管理器”中展开数据库，选择具体数据库下的“安全性”，在展开项中双击“用户”下的数据库用户，在“数据库用户”窗口的“选择页”中单击“成员身份”，然后对数据库角色进行选择，如图 5-11 所示。

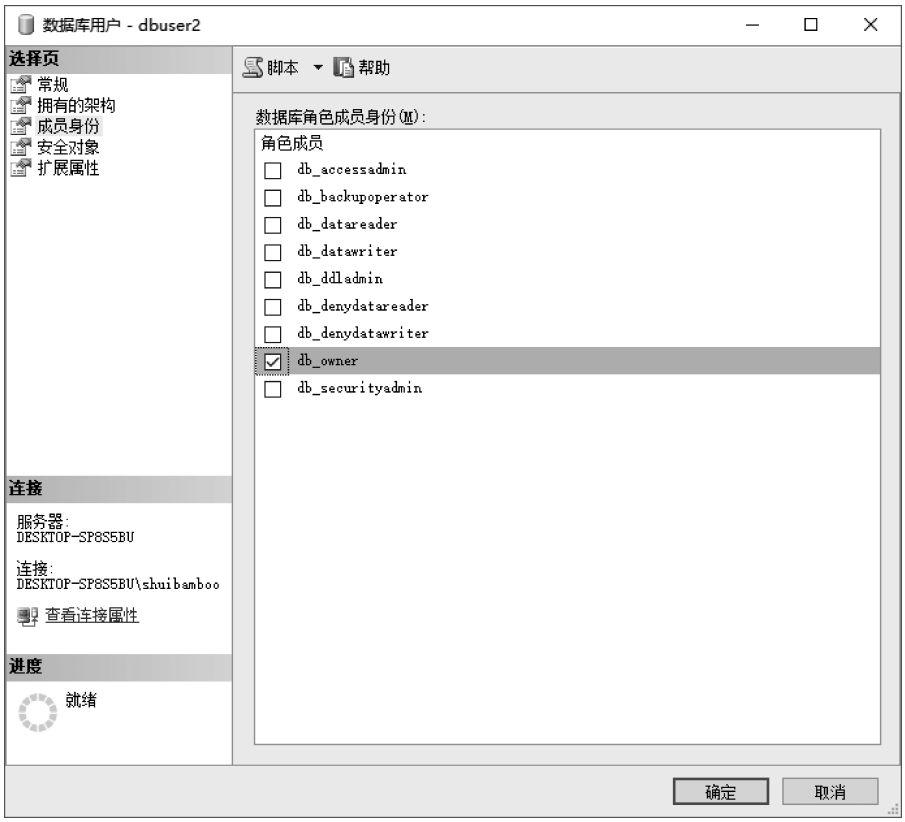


图 5-11　数据库用户属性窗口设置固定数据库角色

在具体数据库下的“安全性”下展开“角色”选项，找到“数据库角色”，双击数据库角色，在“数据库角色属性”界面中通过“添加”或“删除”按钮来调整数据库角色的成员，如图 5-12 所示。

使用存储过程也可以完成以上操作。添加成员的语法如下。

```
EXECUTE sp_addrolemember [@rolename] 'role',[@loginname] 'user'
```

删除成员的语法如下。

```
EXECUTE sp_droprolemember [@rolename] 'role',[@loginname] 'user'
```

各参数说明如下。

role：指定数据库角色。

user：指定数据库用户。

(3) 查看固定数据库角色信息。

查看固定数据库角色列表：EXECUTE sp_helpdbfixedrole

查看固定数据库角色权限：EXECUTE sp_dbfixedrolepermission['role']

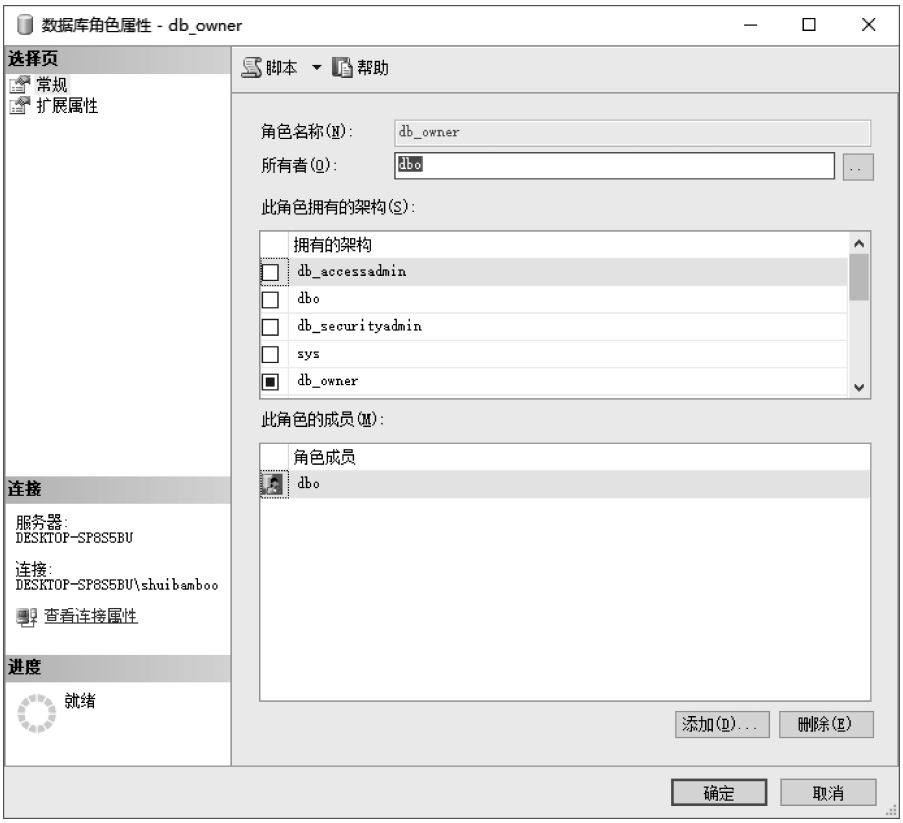


图 5-12 “数据库角色属性”窗口设置固定数据库角色成员

查看固定数据库角色成员：EXECUTE sp_helprolememeber['role']

2）用户自定义数据库角色

由于固定数据库角色不能进行权限修改，有时可能不能满足用户的需要，可以创建用户自定义数据库角色来设置权限。SQL Server 提供了 SQL Server Management Studio 和 SQL 语句两种方式来创建用户自定义数据库角色。

（1）使用 SQL Server Management Studio 方式。

首先，打开 SQL Server Management Studio，连接到目标服务器。

接着，在“对象资源管理器”窗口中展开“服务器”，“数据库”→展开具体的数据库→单击“安全性”→“角色”→“数据库角色”。

然后，右击“数据库角色”，选择“新建数据库角色”命令，打开“数据库角色-新建”窗口，如图 5-13 所示。

在新建窗口中，设置角色名称，确定所有者，单击“添加”按钮，选择数据库用户。

最后，单击“确定”按钮完成自定义数据库角色的创建。

（2）使用 SQL 语句

使用 SQL 语句创建用户自定义数据库角色的命令是 CREATE ROLE，其格式如下。

```
CREATE ROLE role_name
```

当然也可以使用 DROP ROLE 来删除数据库角色。

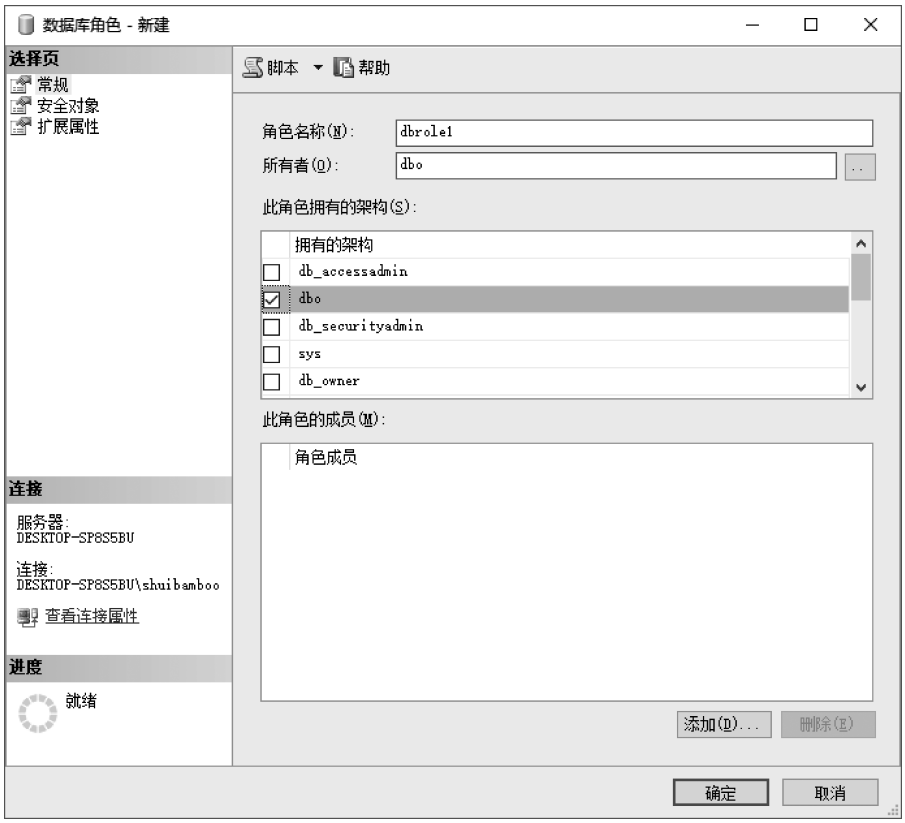


图 5-13　新建用户自定义数据库角色

5.1.5　架构

架构是指一组数据库对象的集合，是数据库内部数据库对象的组织方式，可将架构看成是对象容器。任何用户都可拥有架构，一个数据库用户可以使用多个架构，一个架构也可以被多个数据库用户使用。如果架构的所属是角色，则该角色的成员可以使用同一个架构，当数据库用户删除时，架构中的表会依然存在。如果架构的所属是用户，则删除数据库用户，如果架构中有表，则该用户将不允许删除，直到将用户使用的架构的所属修改为其他用户或角色。

1. 查看架构

SQL Server 数据库管理系统提供了两种方式查看架构，一种是使用 SQL Server Management Studio，另一种是使用查询视图。

1）使用 SQL Server Management Studio

在 SQL Server Management Studio 中选择某个具体数据库下的“安全性”，再展开“架构”，可看到该数据库中的所有架构。如果想查看具体架构的信息，只需双击架构名即可，如图 5-14 所示。

2）使用查询视图

```
SELECT * FROM sys.schemas;
```

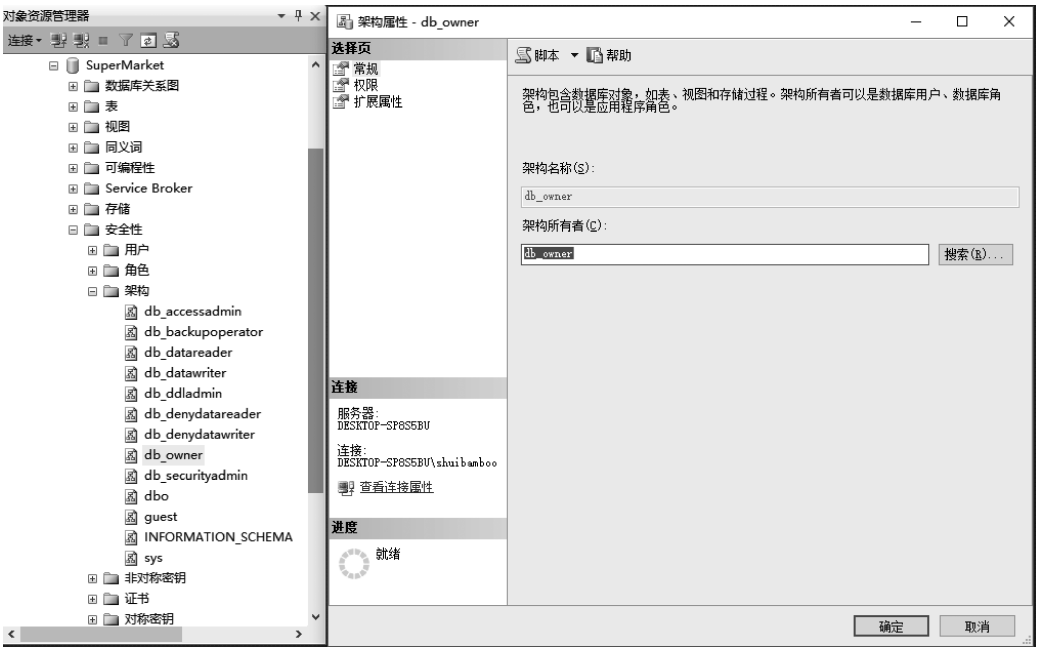


图 5-14　查看架构

运行结果如图 5-15 所示。

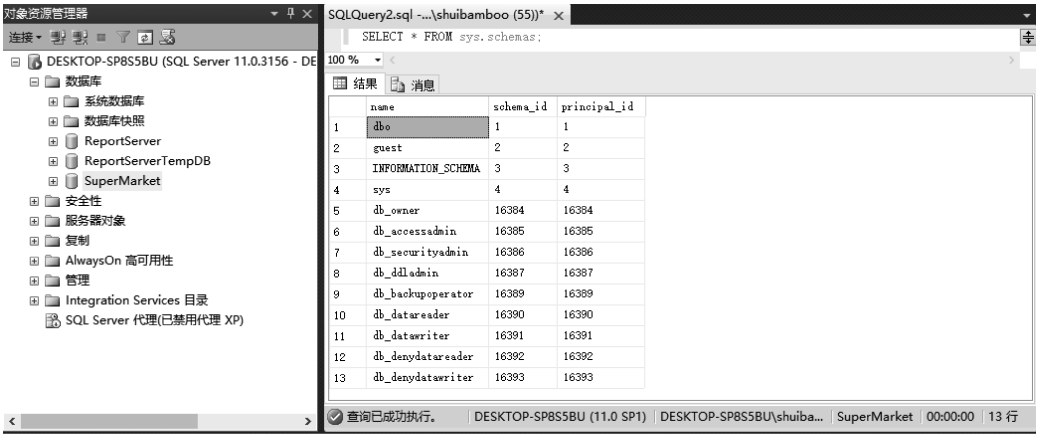


图 5-15　使用查询视图查看架构

2. 创建架构

SQL Server 数据库管理系统提供了两种方式创建架构，一种是使用 SQL Server Management Studio，另一种是使用 SQL。

1）使用 SQL Server Management Studio

在 SQL Server Management Studio 中选择具体数据库下的“安全性”，再选择“架构”，右击选择“新建架构”命令，弹出如图 5-16 所示的界面，输入要新建的架构名称和架构的所有者。架构的所有者可以是数据库用户或数据库角色，无论是数据库用户还是角色，都必须先于架构存在。

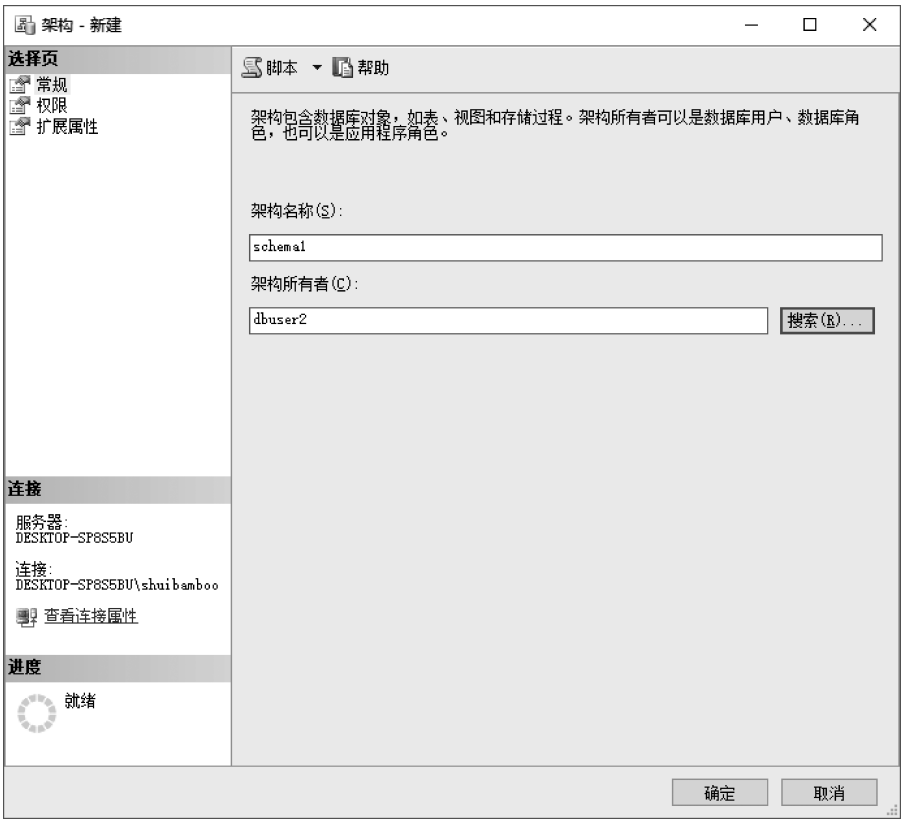


图 5-16　新建架构

2）使用 SQL 语句

使用 SQL 语句创建架构的语法格式如下。

```
CREATE SCHEMA schema_name
AUTHORIZATION owner_name;
```

其参数说明如下。

schema_name：指定架构名称。

AUTHORIZATION owner_name：指定架构的所有者，可以是用户或角色。

【例 5-6】 创建一个架构 schema2，所有者是数据库用户 dbuser2。

注意：*数据库用户 dbuser2 必须事先创建完成。*

```
CREATE SCHEMA schema2 AUTHORIZATION dbuser2
```

3. 删除架构

SQL Server 数据库管理系统提供了两种方式删除架构，一种是使用 SQL Server Management Studio，另一种是使用 SQL。

1）使用 SQL Server Management Studio

在 SQL Server Management Studio 中选择数据库下的“安全性”→“架构”，右击需要删除的架构，选择“删除”，如图 5-17 所示。

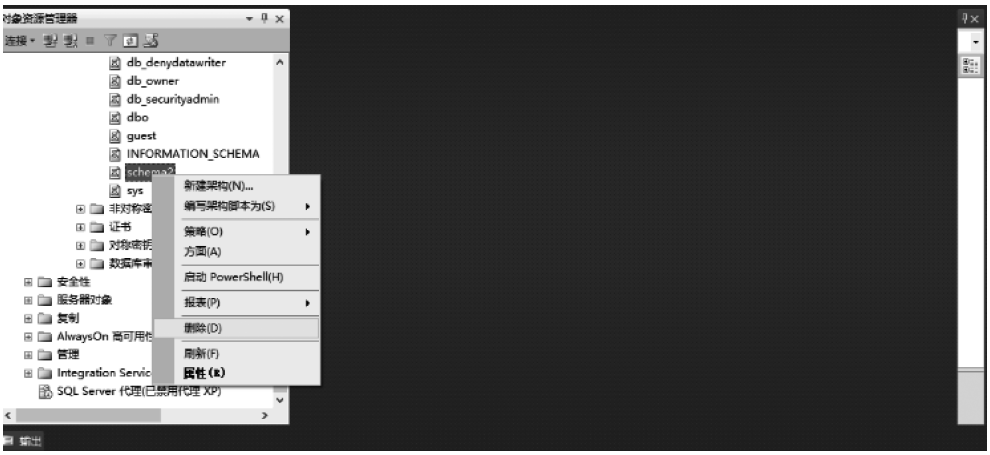


图 5-17　删除架构

2）使用 SQL 语句

使用 SQL 语句删除架构的语法格式如下。

```
DROP SCHEMA schema_name;
```

【例 5-7】 删除架构 schema2。

```
DROP SCHEMA schema2;
```

5.1.6　权限管理

权限是数据库对象级别的认证。权限用于控制对数据库对象的访问以及指定用户对数据库可以执行的操作，用户在登录到 SQL Server 数据库之后，其用户账户所归属的 Windows 组或角色被赋予的权限决定了该用户能对哪些数据库对象执行哪种操作以及能够访问、修改哪些数据。

通常情况下，只有数据库的所有者才可以在该数据库下进行操作。当一个非数据库所有者想访问数据库里的对象时，必须事先由数据库的所有者赋予该用户对指定对象执行特定操作的权限。

1. 权限分类

SQL Server 中，可以按照不同的方式把权限分成不同的类型。比如预定义权限和自定义权限，针对所有对象的权限和针对特殊对象的权限。

预定义权限是指在完成 SQL Server 安装后，不必授权就拥有的权限，比如固定服务器角色和固定数据库角色都属于预定义权限。自定义权限是指那些需要经过授权或继承才能得到的权限。

针对所有对象的权限是指某些权限对所有 SQL Server 中的对象起作用，比如 CONTROL 权限是所有对象都具有的权限。针对特殊对象的权限是指某些权限只能在指定的对象上起作用，比如 DELETE 只能用作表的权限，不可以是存储过程的权限；而 EXECUTE 只能用作存储过程的权限，不能作为表的权限等。

常用的权限类型是对象权限、语句权限和隐式权限三类。权限管理的主要任务是管理

语句权限和对象权限。

1）对象权限

对象权限用于用户对数据库对象执行操作的权利，即处理数据或执行存储过程所需要的权限。SQL Server 中所有对象权限是可以授予的。数据库用户可以为特定对象、特定类型的所有对象和所有属于特定架构的对象管理权限。这些数据库对象包括表、视图、存储过程、表值函数、标量函数和列等，具体见表 5-3。

表 5-3　对象权限

对　　象	操　　作
表	SELECT，INSERT，UPDATE，DELETE，REFERENCES
视图	SELECT，INSERT，UPDATE，DELETE，REFREENCES
存储过程	EXECUTE，SYNONYM
表值函数	SELECT，INSERT，UPDATE，DELETE，REFREENCES
标量函数	EXECUTE，REFERENCES
列	SELECT，UPDATE

2）语句权限

语句权限是用户是否具有权限来执行某一语句，用于控制创建数据库或数据库中的对象而涉及的权限。例如，某用户要在数据库中创建视图，则应该向该用户授予 CREATE VIEW 语句权限。只有 sysadmin、db_owner 和 db_securityadmin 角色的成员才能授予用户语句权限。SQL Server 中可以授予、拒绝或撤销的语句权限见表 5-4。

表 5-4　语句权限

语 句 权 限	权 限 描 述
CREATE DATABASE	创建数据库的权限
CAEATE TABLE	在数据库中创建表的权限
CREATE VIEW	在数据库中通过创建视图的权限
CREATE DEFAULT	在数据库中创建默认对象的权限
CREATE PROCEDURE	在数据库中创建存储过程的权限
CREATE RULE	在数据库中创建规则的权限
CREATE FUNCTION	在数据库中创建函数的权限
BACKUP DATABASE	备份数据库的权限
BACKUP LOG	备份日志的权限

3）隐式权限

隐式权限是系统预定义而不需要授权就有的权限，包括固定服务器角色、固定数据库角色和数据库对象所有者所拥有的权限。通常只有预定义系统角色的成员或数据库和数据库对象所有者具有隐式权限。所有角色的隐式权限不能修改，而且可以让角色具有相关的隐式权限。

2. 权限操作

权限操作包括授权权限、撤销权限和禁止权限。SQL Server 提供了两种方式操作权限：使用命令的方式和使用 SQL Server Management Studio 的方式。

1）使用 SQL Server Management Studio 进行权限操作

在 SQL Server 中可以使用 SQL Server Management Studio 实现对语句权限和对象权限的操作，从而实现对用户或角色权限的设定。这里给出语句权限操作的具体步骤。

首先，打开 SQL Server Management Studio，连接到目标服务器。

接着，在“对象资源管理器”窗口中打开“服务器”→“数据库”，选择要操作的具体数据库，右击选择“数据库属性”→单击“选择页”中的“权限”。

然后，选择用户，在下面的权限列表中对权限进行授予或拒绝，如图 5-18 所示。

最后，单击“确定”按钮完成权限操作。

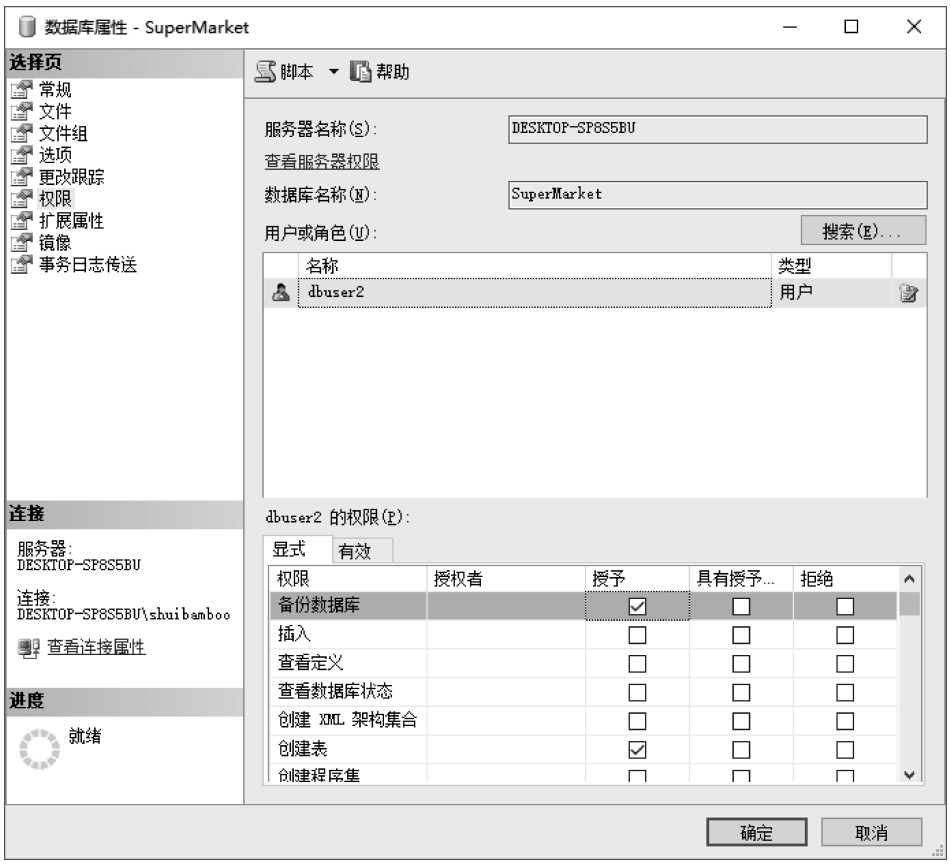


图 5-18　权限操作窗口

2）使用命令进行权限操作

（1）授予权限。

授予权限是将权限赋予某一数据库用户或角色执行所授权指定的操作，使用 GRANT 语句来完成。

授予对象权限的语法格式如下。

```
GRANT
ALL [PRIVILEGES] | PERMISSION [, … ]
[(column [, … n])] ON table | view | stored_procedure
TO security_account [, … n]
```

```
[WITH GRANT OPTION]
[AS group|role]
```

授予语句权限的语法格式如下。

```
GRANT
ALL| STATEMENT [, … n]
TO security_account [, … n]
[WITH GRANT OPTION]
```

各参数说明如下。

ALL [PRIVILEGES]：所有可授予的权限。

PERMISSION：表示在对象上可执行的具体权限，如对象权限表上的 INSERT。

column：在表或视图上允许用户将权限局限到某些列上，column 表示列的名字。

TO：指定被授予者。

WITH GRANT OPTION：表示被授权者是否可以把获得的权限授予其他用户。

security_account：定义被授予权限的用户。可以是 SQL Server 的数据库用户，也可以是 SQL Server 角色，还可以是 Windows 的用户或工作组。

STATEMENT：表示可以授予的语句权限。

【例 5-8】 授予数据库用户 dbuser2 查询和修改 Goods 表的权限。

```
GRANT SELECT, UPDATE ON Goods TO dbuser2
```

【例 5-9】 授予数据库用户 dbuser2 创建表和创建视图的权限。

```
GRANT CREATE TABLE,CREATE VIEW TO dbuser2
```

(2) 禁止权限。

禁止权限是指拒绝给某一数据库用户或角色特定权限的操作，同时阻止它们从其他角色中继承这个权限。使用 DENY 语句来完成。

DENY 语法格式与 GRANT 语法格式一样。

【例 5-10】 禁止 guest 用户对 Goods 表进行查询、添加、修改和删除操作。

```
DENY SELECT, INSERT, UPDATE, DELETE ON Goods TO guest
```

(3) 撤销权限。

撤销权限是撤销某一数据库用户或角色先前被赋予或禁止权限的操作。使用 REVOKE 语句来完成。REVOKE 语法格式与 GRANT 语法格式一样，只是将 TO 改成 FROM。

【例 5-11】 撤销数据库用户 dbuser2 对表 Goods 的查询和修改权限。

```
REVOKE SELECT, UPDATE ON Goods FROM dbuser2
```

3. 权限查看

可以使用 SQL Server Management Studio 进行权限查看，方法与权限操作的一致。这里介绍使用存储过程来查看当前数据库中某对象的对象权限或语句权限的信息。其语法格式如下。

```
EXECUTEsp_helprotect [[@name = ]'object_statement']
                     [, [@username = ]'security_account']
                     [,[@grantorname = ]'grantor']
```

其中，

[@name=]'object_statement'：对象名或授权语句名。

[@username=]'security_account'：被授权的用户账号名。

[@grantorname=]'grantor'：授权的用户账号名。

【例 5-12】 查看数据库用户 dbuser2 拥有的权限。

```
EXECUTE sp_helprotect @username = 'dbuser2'
```

执行结果见图 5-19。

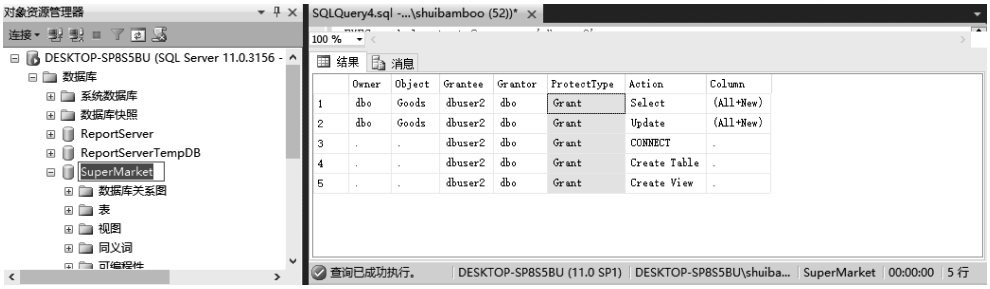


图 5-19　存储过程查看 dbuser2 拥有的权限

【例 5-13】 查看获得 CREATE VIEW 权限的用户信息。

```
EXECUTE sp_helprotect @name = 'CREATE VIEW'
```

执行结果见图 5-20。

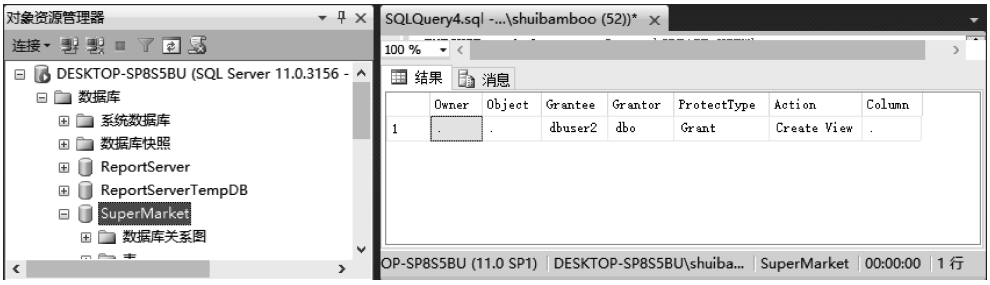


图 5-20　存储过程查看获得 CREATE VIEW 权限的用户信息

5.2　并 发 控 制

数据库是一个多用户的共享数据集合，在多个用户同时执行某些操作时，由于操作间的互相干扰，有可能产生错误的结果。即使这些操作在单独执行时都是正确的，但是在并发执行时有可能存取不正确的数据，破坏数据的一致性。因此，数据库管理系统必须提供并发控制机制以保证数据的正确性。

5.2.1 事务概述

1. 事务的概念

事务是由用户定义的一系列数据操作语句构成的，这些操作语句要么全部执行，要么全部不执行，是数据库运行的最小的、不可分割的工作单位。所有对数据库的操作都要以事务为一个整体单位来执行或撤销，同时事务也是保证数据一致性的基本手段。无论什么情况下，DBMS 都应该保证事务能正确、完整地执行。在关系数据库中，一个事务可以是一条 SQL 语句、一组 SQL 语句或整个程序。

2. 事务的特性

事务由有限的数据库操作序列组成，但不是任意的操作序列都能成为事务，它必须同时满足以下四个特性：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)和持续性(Durability)。这四个特性简称为 ACID 特性。

1) 原子性

一个事务对于数据的所有操作都是不可分割的整体，这些操作要么全部执行，要么全部不执行。原子性是事务概念本质的体现和基本要求。

2) 一致性

事务执行完成后，数据库中的内容必须全部更新，确保事务执行后使数据库从一个一致性状态变成另一个一致性状态，此时数据库中的数据具备正确性和完整性。当数据库只包含事务成功提交的结果，则说明数据库处于一致性状态；如果数据库系统运行过程中发生了故障，有些事务尚未完成就被迫中断，这些未完成事务对数据库所做的更新操作有一部分已写入物理数据库，这时数据库就处于一种不正确的状态，或者说不一致的状态，为了保证一致性，系统会对事务中对数据库的所有已完成的操作全部撤销，回滚到事务开始时的一致性状态。

3) 隔离性

隔离性也称独立性，表明一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对其他并发事务是隔离的，并发执行的各个事务之间不能互相干扰。

4) 持续性

持续性也称永久性，表明一个事务一旦提交，它对数据库中的数据的改变就应该是永久的，接下来的其他操作或故障不应该对其执行结果有任何影响。

事务是并发控制的基本单位，保证事务的 ACID 特性是事务处理的重要任务。事务的 ACID 特性可能遭到破坏的因素一般有以下两种。

(1) 多个事务并行运行时，不同事务的操作交叉执行。此时 DBMS 必须保证多个事务的交叉运行不影响这些事务的原子性。

(2) 事务在运行过程中被强行停止。此时 DBMS 必须保证被强行停止的事务对数据库和其他事务没有任何影响。

3. 事务的处理模型

SQL Server 事务处理模型有两种：一种是显式事务，是指事务有显式的开始和结束标记；另一种是隐式事务，是指每一条数据操作语句都自动成为一个事务。对于显式事务，不同的数据库管理系统有不同的形式。一类是采用国际标准化组织制定的事务处理模型；另

一类是采用 T-SQL 的事务处理模型。

1）ISO 事务处理模型

ISO 事务处理模型中事务的开始是隐式的，而事务的结束有明确标记。在这种事务处理模型中，程序的首条 SQL 语句或事务结束符后的第一条语句自动作为事务的开始；而程序的正常结束或 COMMIT/ROLLBACK 语句作为事务的终止。

【例 5-14】 向 Goods 表中插入数据。

```
INSERT INTO Goods(GoodsNO, GoodsName) values('G011','松子');
INSERT INTO Goods(GoodsNO, GoodsName) values('G012','瓜子');
INSERT INTO Goods(GoodsNO, GoodsName) values('G013','花生');
INSERT INTO Goods(GoodsNO, GoodsName) values('G014','开心果');
COMMIT
```

2）T-SQL 事务处理模型

T-SQL 事务处理模型对每个事务都有显式的开始标记 BEGIN TRANSACT 和结束标记 COMMIT 或 ROLLBACK。

【例 5-15】 向 Goods 表中插入数据。

```
BEGIN TRANSACT
INSERT INTO Goods(GoodsNO, GoodsName) values('G015','无花果');
INSERT INTO Goods(GoodsNO, GoodsName) values('G016','杨梅');
INSERT INTO Goods(GoodsNO, GoodsName) values('G017','话梅');
INSERT INTO Goods(GoodsNO, GoodsName) values('G018','葡萄干');
COMMIT
```

5.2.2 并发控制概述

数据库系统是多用户共享数据库资源，尤其是多个用户可以同时存取相同数据，如银行系统数据库、超市管理数据库等都是多个用户共享的数据库系统。在这些系统中，同一时间可同时运行数百个事务。若对多用户的并发操作不加以控制，就会造成数据存取错误，破坏数据库的一致性和完整性。

在 DBMS 运行多个事务时，如果一个事务完成以后，再开始另一个事务，这种执行方式为事务的串行执行。如果 DBMS 可以同时接受多个事务，并且这些事务在时间上可以重叠执行，这种执行方式为事务的并发执行。并发执行能提高系统资源的利用率，改善短事务的响应时间等。但并发执行可能会破坏事务的 ACID 特性。下面举例说明并发执行带来的数据不一致的问题。

设有两个校园超市收银台 A 和 B，其中，A 和 B 同时收取同一商品（洗衣粉）的费用，修改洗衣粉的库存数量。其操作过程及顺序如下。

收银台 A（事务 A）读出目前洗衣粉的库存数量，假设为 20 袋。

收银台 B（事务 B）读出目前洗衣粉的库存数量也为 20 袋。

收银台 A 此时要卖出 3 袋洗衣粉，则修改库存数量 20－3＝17，并将 17 写回数据库中。

收银台 B 此时也要卖出 4 袋洗衣粉，则修改库存数量 20－4＝16，并将 16 写回数据库中。

从上述操作可以看出，事务 B 覆盖了事务 A 对数据库的修改，使数据库中的数据不可

信，这种情况称为数据的不一致性。这种不一致性就是由并发执行引起的。由于在并发执行下 DBMS 对事务 A 和 B 操作序列的调度是随机的，会产生数据不一致，而这种不一致性是致命的，且在现实生活中是绝对不允许发生的。因此数据库管理员必须想办法避免这种情况，这就是数据库管理系统在并发控制中要解决的问题。

5 并发操作导致的问题

1. 并发操作导致的问题

数据库的并发操作会导致 4 种问题：丢失更新、读“脏”数据、不可重复读和产生“幽灵”数据。下面分别介绍这 4 种问题。

1）丢失更新

丢失更新是指当两个或两个以上的事务选择同一数据值，在更新最初的读取值时，会发生丢失更新的问题。两个事务 T1 和 T2 从数据库读取同一数据并进行更新，T1 执行更新后提交，T2 在 T1 更新后也对该数据进行了更新，此时 T2 提交的结果就破坏了 T1 提交的结果，导致 T1 的修改被 T2 覆盖掉，这样 T1 的更新就被丢失了。这是由于每个事务都不知道其他事务的存在，最后的更新将重写由其他事务所做的更新，这将导致数据丢失。

丢失更新是由于多个事务对同一数据并发进行写入操作引起的。前面例子中收银台 A 和收银台 B 同时对洗衣粉数量进行更新时，最后进行的更新数量必将替代第一个更新的数量，得到错误的结果 16 袋。如果收银台 A 完成收费以后，收银台 B 再收费就可避免这样的问题发生。

2）读“脏”数据

读“脏”数据是指一个事务读取了另一个事务失败运行过程中的数据。也就是说，事务 T1 更新了某一数据，并将更新结果写入磁盘，然后事务 T2 读取了这一数据（T1 更新后的数据）。过了一段时间，由于某种原因 T1 撤销了更新操作，T1 修改过的数据又恢复为原值，此时 T2 读取的数值与数据库中实际数据值不一致。这种数据就是“脏”数据。

前面例子中，收银台 A、B 同时修改洗衣粉数量，收银台 A 修改洗衣粉数量为 17，未做提交操作，这时收银台 B 将修改后的数量 17 读取出来，之后收银台 A 执行回滚操作，数量恢复为原值 20，而收银台 B 仍然在使用已回滚的数量 17。这种修改了但未提交随后又被回滚的数据就是“脏”数据。

3）不可重复读

不可重复读是指事务 T1 读取数据后，事务 T2 对该数据进行读取并执行更新操作，修改了 T1 读取的数据，T1 操作完数据后，又重新读取这个数据，但这次读取后，当 T1 再对这些数据进行相同操作时，所得到的结果与前一次不一样。

前面例子中，收银台 A、B 同时修改洗衣粉数量，收银台 A 在某一时刻读取的数量是 20 袋，过了一段时间，收银台 B 卖出 4 袋将数量更新为 16，此时收银台 A 读取的值不再是最初的 20 了。

4）产生“幽灵”数据

“幽灵”数据实际是不可重复读的一种特殊情况。它是指当事务 T1 按一定条件从数据库中读取某些记录后，事务 T2 在其中插入或删除数据，当 T1 再次按相同条件读取数据时，发现数据库中多出了一些数据或者之前的数据消失了，这些数据对于 T1 来说就是“幽灵”数据。

并发操作破坏了事务的隔离性从而导致出现以上 4 种问题。并发控制是用某种方法来

执行并发操作，使一个事务的执行不受其他事务的干扰，避免造成数据的不一致。

2. 并发控制的方法


5 并发控制的方法

实现并发控制的主要方法是使用封锁机制。锁可以防止事务的并发问题，在多个事务并发执行时能够保证数据库的完整性和一致性。封锁是指一个事务 T 在对某个数据对象操作之前，先向系统发出请求，对其加锁。加锁后事务 T 对该数据对象有一定的控制，在事务结束之后释放锁。而在事务 T 释放锁之前，其他事务不能更新此数据对象，以保证数据操作的正确性和一致性。封锁是一种并发控制技术，用来调整对数据库中共享数据进行并行存取的技术。前面超市的例子中，当收银台 A 要修改洗衣粉数量，在读取出数量前先封锁数量，再对数量进行读取和修改操作，这时收银台 B 就不能读取和修改，直到收银台 A 完成操作，将修改后的数量重新写回数据库，并释放对数量的封锁后，收银台 B 才可以读取和修改，这样就不会导致数据不一致的问题。

具体的控制由封锁的类型决定。基本的封锁类型有两种：排他锁(Exclusive Locks，X 锁)和共享锁(Share Locks，S 锁)。

1) 排他锁

排他锁又称写锁，可以防止并发事务对数据进行访问，其他事务不能读取或更新锁定的数据。如果事务 T 对数据对象 R 加上 X 锁，则只允许事务 T 读取和更新 R，其他任何事务不能再对 R 加任何类型的锁，直到事务 T 释放 R 上的锁。这就保证了其他事务在 T 释放 R 上的锁之前不能再读取和更新 R。由此可见，X 锁采用的方法是禁止并发操作。

2) 共享锁

共享锁又称读锁，允许并发事务读取数据。若事务 T 对数据对象 R 加上 S 锁，则事务 T 读取 R 但不能修改 R，其他任何事务只能再对 R 加 S 锁，而不能加 X 锁，直到事务 T 释放 R 上的 S 锁。这保证了其他事务可以读取 R，而不能再释放 R 上的 S 锁之前对 R 进行修改操作。

对数据库中数据进行读取操作不会破坏数据的完整性，而更新操作才会破坏数据的完整性。加锁的真正目的在于防止更新操作对数据一致性的破坏。S 锁只允许多个事务同时读取同一数据，不能对数据进行更新操作；X 锁只允许一个事务对同一数据进行读取和更新操作，其他事务只能等待 X 锁的释放，才能对该数据进行相应的操作。

排他锁和共享锁的控制可以用如表 5-5 所示的锁的兼容性来表示。

表 5-5　锁的兼容性

T1 / T2	排他锁(X 锁)	共享锁(S 锁)	--(没有锁)
排他锁(X 锁)	否	否	是
共享锁(S 锁)	否	是	是
--(没有锁)	是	是	是

在表 5-5 锁的兼容性内容中，最上面一行是事务 T1 已经获取的数据对象上的锁类型，其中，“--”表示没有加锁。最左侧一列是事务 T2 针对同一数据对象发出的封锁请求，该请求是否被满足，则用“是”和“否”在表格中表示出来。“是”表示事务 T2 的封锁请求与 T1 所获取的锁兼容，可以满足请求；“否”表示事务 T2 的封锁请求与 T1 的锁不兼容，请求被拒绝。

5.2.3 SQL Server 的封锁技术

1. 封锁协议

5 三级封锁协议

在使用排他锁和共享锁对数据对象进行加锁时，还需要约定一些规则：何时申请锁、持锁时间、何时释放锁等。这些规则称为封锁协议。对封锁方式规定不同的规则，就形成了不同级别的封锁协议，不同级别的协议能达到的数据一致性级别也不同。下面介绍三种封锁协议。

1）一级封锁协议

一级封锁协议是指事务 T 在修改数据对象之前必须先对其加 X 锁，直到事务结束（包括正常结束和非正常结束）时才释放锁。一级封锁协议可以防止丢失更新问题的发生。

在一级封锁协议中，如果事务仅仅是读数据而不是更新数据，则不需要加锁。所以一级封锁协议不能保证可重复读和读“脏”数据。

2）二级封锁协议

二级封锁协议是指在一级封锁协议基础上，加上事务 T 对要读取的数据之前必须先对其加 S 锁，读取完后立即释放 S 锁。二级封锁协议可以防止数据丢失更新问题，还可以防止读“脏”数据。

在二级封锁协议中，由于事务 T 读取完数据后立即释放了 S 锁，所以不能保证可重复读数据。

3）三级封锁协议

三级封锁协议是指在一级封锁协议基础上，加上事务 T 在读取数据之前必须先对其加 S 锁，读取完后并不释放 S 锁，直到事务 T 结束才释放。三级封锁协议除可以防止丢失更新和不读“脏”数据外，还可以防止不可重复读。

3 个封锁协议均规定对数据对象的更新必须加 X 锁，而它们的主要区别在于读取操作是否需要申请封锁，何时释放锁。3 个级别的封锁协议的主要规则及能解决的问题如表 5-6 所示。

表 5-6　不同级别的封锁协议

封锁协议	排他锁（X 锁）	共享锁（S 锁）	不丢失更新	不读脏数据	可重复读
一级封锁协议	必须加锁，直到事务结束才释放	不加锁	是		
二级封锁协议	必须加锁，直到事务结束才释放	加锁，读取完后立即释放锁	是	是	
三级封锁协议	必须加锁，直到事务结束才释放	必须加锁，直到事务结束才释放	是	是	是

2. 死锁和活锁

5 死锁和活锁

封锁技术可以有效地解决并发操作的一致性问题，但也会带来一些新的问题：活锁和死锁等问题。

1）活锁

当两个或多个事务请求对同一数据进行封锁时，可能会存在某个事务处于永远等待锁

的情况，这种现象称为活锁。比如事务 T1 封锁了数据对象 R 后，事务 T2 也申请封锁 R，于是 T2 等待；接着事务 T3 也申请封锁 R。当 T1 释放了 R 上的封锁后，系统首先批准了 T3 的请求，T2 仍然等待。这时事务 T4 又申请封锁 R，当 T3 释放了 R 上的封锁后，系统又批准了 T4 的请求，这样依次类推，T2 有可能永远等待，这就是活锁。

避免活锁最简单的方法就是采用先来先服务的策略。当多个事务请求封锁同一数据对象时，封锁子系统按申请封锁的先后顺序对事务进行排队，数据对象上的锁一旦释放就批准申请队列中的第一个事务获得锁。

2）死锁

在同时处于等待状态的两个或多个事务中，其中每一个事务又在等待其他事务释放封锁后才能继续执行，这样出现多个事务彼此相互等待的状态就称为死锁。比如事务 T1 封锁了数据对象 R1，事务 T2 封锁了数据对象 R2。之后 T1 又申请封锁数据对象 R2，由于 T2 已经封锁了 R2，于是 T1 的申请被拒绝只能等待，直到 T2 释放 R2 上的锁。接着 T2 又申请封锁 R1，由于 R1 已经被 T1 封锁，于是 T2 的申请被拒绝只能等待，直到 T1 释放 R1。这样就出现了 T1 在等待 T2，而 T2 又在等待 T1 的局面，T1 和 T2 两个事务永远不能结束，形成死锁。

目前在数据库中解决死锁问题的方法主要有两类：一类是采取一定的措施来预防死锁的发生；另一类是允许死锁的发生，但需采取一定的手段定期诊断系统中有无死锁，若有则解除它。

（1）死锁的预防。

预防死锁就是要破坏产生死锁的条件，通常有如下两种方法。

① 一次性封锁法。一次性封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行。比如针对前面死锁中的例子，事务 T1 将需要的数据对象 R1 和 R2 一次加锁，T1 就可以执行，而事务 T2 等待。当 T1 执行完后释放 R1、R2 上的锁，T2 就获得 R1 和 R2 上的锁，继续执行。这样就不会发生死锁。一次性封锁法虽然可以有效地防止死锁的发生，但也存在不足：将事务以后要用的全部数据对象加锁，扩大了封锁的范围，降低了系统的并发度，从而影响了系统的效率；另外，需要事先精确地确定每个事务所要封锁的所有数据对象，这对于不断变化的数据库来讲是很困难的，因此只能扩大封锁范围，将事务可能需要用到的数据进行加锁，这会进一步降低并发度。

② 顺序封锁法。顺序封锁法是要求所有事务必须按照一个预先约定的封锁顺序对所要用到的数据对象进行封锁。比如规定事务封锁数据对象 R1、R2 的顺序依次是 R1、R2，则事务 T1 和 T2 必须先封锁 R1 再封锁 R2，当 T2 请求 R1 的封锁时，由于 T1 已经封锁了 R1，则 T2 就只能等待，T1 释放 R1 和 R2 的锁之后，T2 就继续执行，这样就不会发生死锁。顺序封锁在一定程度上可以有效地防止死锁，但仍然存在不足：很难预先确定所有数据对象的加锁顺序；当封锁的数据对象很多时，随着数据的不断更新，维护数据对象的顺序也很困难。

因此，预防死锁策略难以实施，在解决数据库死锁的问题上，DBMS 普遍采用诊断并解除死锁的方法。

（2）死锁的诊断与解除。

死锁的解除是指允许产生死锁，在死锁发生后通过一定手段予以解除。一般使用超时法或事务等待图法。

① 超时法。是指对每个锁设定一个时限，如果某个事务的等待时间超过了该时限，就认为发生了死锁，此时调用解锁程序，以解除死锁。超时法实现简单，但存在明显不足：时限难于设置，若设置太长，则会导致死锁发生后不能及时发现；有可能误判死锁，事务可能因为其他原因使等待超时，系统会误认为发生了死锁。

② 事务等待图法。事务等待图是一个特殊的有向图 $G=(T,U)$。T 为结点的集合，每个结点表示正在运行的事务；U 为边的集合，每条边表示事务等待的情况。若 T1 等待 T2，则 T1、T2 之间画一条有向边，从 T1 指向 T2。建立事务等待图之后，诊断死锁的问题就变成了判断有向图 G 中是否存在回路的问题。事务等待图动态地反映了所有事务的等待情况，并发控制子系统周期性地生成事务等待图，并进行检测，如果图中没有回路，则没有发生死锁，反之则说明发生了死锁。

一旦检测到系统存在死锁，DBMS 就要设法解除。通常采用的方法是选择一个处理死锁代价最小的事务，将其撤销，释放该事务持有的所有锁，使其他事务得以继续运行下去。当然，为了保证数据的一致性，对撤销事务所执行的数据更新操作必须加以恢复。

3. 并发调度的可串行性

5 并发调度的可串行性

数据库管理系统对并发事务中的操作调度是随机的，不同的调度会产生不同的结果。什么样的调度是正确的呢？显然串行调度是正确的。一般来讲，如果多个事务在某个调度下的执行结果与这些事务在某个串行调度下的执行结果相同，那么这个调度也是正确的。虽然以不同顺序串行执行事务可能会产生不同的结果，但不会将数据库置于不一致的状态，因此这个调度是正确的。

多个事务的并发执行是正确的，当且仅当结果与按某一顺序串行地执行这些事务时的结果相同，则称这种调度策略为可串行化的调度。

可串行性是并发事务正确调度的准则。按这个准则规定，一个给定的并发调度，当且仅当它可串行化时，才认为它是正确的调度。为保证并发操作的正确性，数据库管理系统的并发控制机制必须提供一定的手段来保证调度是可串行化的。

【例 5-16】 假设有两个事务 T1 和 T2，分别包含下列操作。

事务 T1：读取 B；A＝B－3；写回 A。

事务 T2：读取 A；B＝A－3；写回 B。

假设 A、B 的初值均为 20，若按 T1→T2 的顺序执行后，其结果 A＝17，B＝14；若按 T2→T1 的顺序执行后，其结果 A＝14，B＝17。当并发调度时，如果执行的结果是这两者之一，则认为都是正确的并发调度策略。图 5-21 给出了这个两个事务的 4 种调度策略。

图 5-21 中(1)和(2)是不同的串行调度策略，虽然执行结果不同，但它们都是正确的调度。(3)虽不是串行调度，但其执行的结果与串行调度的结果相同，所以该调度是正确的。(4)的执行结果与前两个串行调度的结果都不同，所以是错误的调度。

4. 两段锁协议

5 两段锁协议

为保证并发调度的正确性，数据库管理系统的并发控制机制必须提供一定的手段来保证调度的可串行化。目前，数据库管理系统普遍采用两段锁协议来实现并发调度的可串行化，从而保证调度的正确性。

两段锁协议是最常用的一种封锁协议。它是指所有的事务必须分为两个阶段对数据对象进行加锁和解锁。具体包括两个方面的内容：在对任何数据进行读写操作之前，要先申

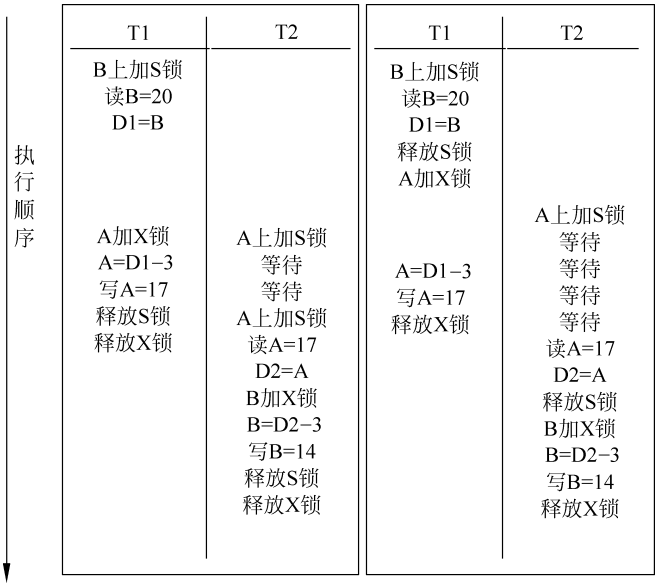
(1) 串行调度　(2) 串行调度　(3) 可串行化调度　(4) 不可串行化调度

图 5-21　并发事务的不同调度策略

请并获得对该数据的封锁；在释放一个封锁之后，事务不再申请和获得对该数据的封锁。

所谓“两段”锁就是事务分为两个阶段：第一阶段是申请封锁，在这个阶段，事务可以申请获得任何数据对象上的任何类型的锁，但是不允许释放任何锁；第二个阶段是释放封锁，在这个阶段，事务可以释放任何数据对象上的任何类型的锁，但不允许申请任何锁。如果并发执行的所有事务都遵守两段锁协议，则这些事务的任何并发调度策略都是可串行化的。

事务遵守两段封锁协议是可串行化调度的充分条件，而不是必要条件。也就是说，如果并发事务都遵守两段锁协议，则对这些事务的任何并发调度策略都是可串行化的。反之，若对并发事务的调度是可串行化的，并不意味着这些事务都符合两段锁协议。如图 5-22 所示，(1)遵守两段锁协议，(2)不遵守两段锁协议，但它们都是可串行化的调度。



(1) 遵守两段协议　(2) 不遵守两段协议

图 5-22　可串行化调度

5.3 备份及恢复管理

尽管数据库管理系统采用了许多措施来保证数据库的安全性和完整性，但故障仍不可避免，这会影响甚至破坏数据库，造成数据错误或丢失。通过备份和恢复数据库，可以防止因为各种原因而造成的数据破坏和丢失，并使数据库继续正常工作。


5 数据备份

5.3.1 备份与恢复概述

1. 数据备份

数据备份是指定期或不定期地对数据库及其相关信息进行复制，在本地机器或其他机器上创建数据库的副本。数据库备份记录了在进行备份这一操作时数据库中所有数据的状态，当数据库因意外被损坏时，副本就可在数据库恢复时用来恢复数据库。因此，数据备份是保证系统安全的一项重要措施。

1）备份类型

根据备份数据的大小，可以把备份分成以下四种。

（1）完全备份。

完全备份又称完全数据库备份，是数据备份常用的方式之一。完全备份将备份整个数据库，不仅包括用户表、系统表、索引、视图、存储过程等所有数据库对象，还包括事务日志部分。完全备份代表备份完成时的数据库，通过包括在备份中的事务日志可以使用备份恢复到备份完成时的数据库。

完全备份操作简单，便于使用。通常情况对于规模较小的数据库而言，可以快速完成完全备份，但随着数据库规模不断增大，进行一次完全备份，需要花费更多的时间和空间。因此，需要根据备份计划安排完全备份，对于大型数据库可以使用差分备份来补充完全备份。

（2）差分备份。

差分备份也称增量备份，与完全备份不同，它仅备份自上次完全备份以来对数据进行改变的内容。差分备份相比完全备份而言备份速度更快，空间更节省，简化了数据备份操作，减少丢失数据的可能性。为了减少还原频繁修改数据库的时间，可以执行差分备份。

如果数据库中的部分对象频繁更改，差分备份特别有用。在这种情况下，使用差分备份可以频繁地执行备份，并且不会产生完全备份的开销。

对于规模大的数据库来讲，完全备份需要大量的磁盘空间。为了节省备份时间和存储空间，可以在一次完全备份后安排多次差分备份。

（3）事务日志备份。

数据库事务日志是单独的文件，它记录了数据库的改变。事务日志备份是对事务日志进行备份，备份时复制自上次备份以来对数据库所做的改变，仅需要很少的时间，因此建议频繁备份事务日志，从而减少丢失数据的可能性。

用户可以使用事务日志备份将数据库恢复到特定的即时点或恢复到故障点。

事务日志备份和差分备份有所不同。差分备份无法将数据库恢复到出现故障前某一个指定的时刻，它只能将数据库恢复到上一次差分备份结束的时刻。

(4) 文件或文件组备份。

数据库由磁盘上的许多文件构成。如果数据库非常大,执行完全备份是不可行的,则可以使用文件备份或文件组备份来备份数据库的一部分。

2) 备份设备

在创建备份时,必须选择存放备份数据库的备份设备。数据备份是可以将数据库备份到磁盘设备或磁带设备上。磁盘备份设备就是硬盘或其他磁盘上的文件,可以像操作系统文件一样进行管理,也可以将数据库备份到远程计算机的磁盘上。

SQL Server 使用物理设备名称或逻辑设备名称标识备份设备。物理备份设备是操作系统用来标识备份设备的名称。SQL Server 使用两种方式建立备份设备。

(1) 使用 Microsoft SQL Server Management Studio 建立备份设备。

【例 5-17】 建立备份设备:其物理备份设备名为"E:\Database\SuperMarket_full.bak",逻辑备份设备名为"SuperMarket_bakdevice"。

建立步骤如下。

① 启动 Microsoft SQL Server Management Studio,并连接到目标服务器。

② 打开"对象资源管理器"窗口,展开"服务器"→"服务器对象",再到"备份设备"。

③ 右击"备份设备",选择"新建备份设备",打开"备份设备"窗口,如图 5-23 所示。

④ 在"设备名称"文本框中输入"SuperMarket_bakdevice";在不存在磁带机的情况下,"目标"选项自动选中"文件"单选按钮,在与之对应的文本框中输入文件路径和名称"E:\Database\SuperMarket_full.bak"。

⑤ 最后,单击"确定"按钮,完成备份设备创建。

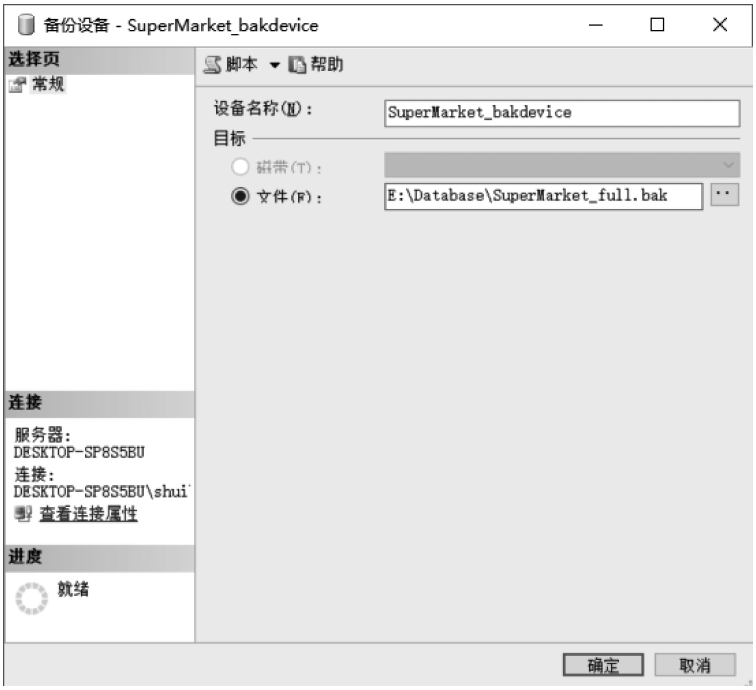


图 5-23 "备份设备"窗口

(2) 使用系统存储过程建立备份设备。

SQL Server 使用系统存储过程 sp_addumpdevice 添加物理备份设备。其语法格式如下。

```
sp_addumpdevice [@ devtype = ]'device_type'
                ,[@logicalname = ]'logical_name'
                ,[@physicalname = ]'physical_name'
```

各参数说明如下。

[@ devtype=]'device_type': 备份设备的类型。device_type 的数据类型是 varchar(20),无默认值,可取 disk,表示硬盘文件作为备份设备; 取 tape,表示 Windows 支持的任何磁盘设备。

[@logicalname=]'logical_name': 在备份和恢复语句中使用的备份设备的逻辑名称。logical_name 的数据类型为 sysname,无默认值,且不能为 NULL。

[@physicalname=]'physical_name': 备份设备的物理名称。物理名称必须遵从操作系统文件名规则或网络设备的命名约定,并且必须包括完整的路径。physical_name 的数据类型为 nvarchar(260),无默认值,且不能为 NULL。

【例 5-18】 利用系统存储过程创建例 5-17 的备份设备。

```
sp_addumpdevice @devtype = 'disk'
                ,@logicalname = 'SuperMarket_bakdevice'
                ,@physicalname = 'E:\Database\SuperMarket_full.bak'
```

在 SQL Server 中可以使用系统存储过程 sp_dropdevice 删除数据库设备或备份设备,并从 master.dbo.sysdevice 中删除相应的项。其语法格式如下。

```
sp_dropdevice [@logicalname = ]'device'
              [,[@delfile = ]'delfile']
```

各参数说明如下。

[@logicalname=]'device': 在 master.dbo.sysdevice 中列出的数据库设备或备份设备的逻辑名称。device 的数据类型为 sysname,无默认值。

[@delfile=]'delfile': 指定物理备份设备文件是否应删除。delfile 的数据类型为 varchar(7)。如果指定为 delfile,则删除物理备份设备磁盘文件。

【例 5-19】 删除备份设备 SuperMarket_bakdevice,并删除相关的物理文件。

```
sp_dropdevice 'SuperMarket_bakdevice', 'delfile'
```

3) 备份计划

创建备份的目的是为了恢复已损坏的数据库。但是,备份和恢复数据需要使用一定的资源,在特定的环境中进行。因此,在备份数据库之前需要对备份内容、备份频率以及数据备份存储介质等进行合理的计划。

(1) 备份内容。

备份数据库应备份数据库中的表、数据库用户、用户定义的数据库对象及数据库中的全部数据。表包括系统表、用户定义的表,还应该备份数据库日志等内容。

(2) 备份频率。

确定备份频率需要考虑的因素: 存储介质出现故障时,允许丢失的数据量的大小; 数

据库的事务类型，以及事故发生的频率。

不同的数据库备份频率通常不一样。一般情况下，数据库可以每周备份一次，事务日志可以每日备份一次。对于一些重要的联机数据库，数据库可以每日备份一次，事务日志甚至可以每隔数小时备份一次。

（3）备份存储介质。

常用的备份存储介质包括硬盘、磁带和命令管道等。具体使用哪一种介质，要考虑用户的成本承受能力、数据的重要程度、用户的现有资源等因素。在备份中使用的介质确定以后，一定要保持介质的持续性，一般不要轻易地改变。

2. 数据恢复

5 数据恢复

数据恢复是指当系统运行过程中发生故障时，利用数据库的备份副本和日志文件将数据库恢复到故障前的某个一致性状态。不同故障有不同的恢复策略和恢复方法。

1）事务故障的恢复

事务故障是指事务在运行到正常终止点前被中止，这时可以利用事务操作的日志文件撤销该事务对数据库进行的修改。事务故障恢复的步骤如下。

（1）反向扫描事务操作的日志文件，查找该事务的更新操作。

（2）对事务的更新操作执行反向操作。也就是对已经插入的新记录执行删除操作；对已经删除的记录执行插入操作；对已经修改的数据恢复旧值。

（3）这样从后到前逐个扫描该事务的所有更新操作，按同样的方式进行处理，直到扫描到该事务的开始标记为止，事务故障就恢复完毕。

事务故障的恢复工作由数据库管理系统自动完成，不需要用户干预。

2）系统故障的恢复

系统故障造成数据库数据不一致状态有两种情况：一是未完成事务对数据库的更新可能已写入数据库，这种情况需要强行撤销所有未完成的事务并清除事务对数据库所做的修改；二是已提交事务对数据库的更新可能还留在缓冲区，没有来得及写入磁盘上的物理数据库中，这种情况应将事务提交的更新结果重新写入数据库。因此系统故障恢复步骤如下。

（1）先正向扫描日志文件，找出在故障发生前已提交的事务，将其事务标记记入重做队列，同时找出故障发生时未完成的事务，将该事务标记记入撤销队列。

（2）接着对撤销队列中的各个事务进行撤销处理，其方法同事务故障恢复一致，也就是对已经插入的新记录执行删除操作；对已经删除的记录执行插入操作；对已经修改的数据恢复旧值。

（3）最后对重做队列中的各个事务进行重做处理，方法是正向扫描日志文件，按照日志文件中所登记的操作内容重新执行事务操作，使数据库恢复到最近的某个可用状态。

系统故障恢复仍由数据库管理系统自动完成，不需要用户干预。

3）介质故障的恢复

发生介质故障后，磁盘上的物理数据和日志文件被破坏，这是最严重的一种故障，可能会造成数据无法恢复。其恢复方法是重装数据库，然后重做已完成的事务。具体步骤如下。

（1）装入最新的数据库备份副本，使数据库恢复到最近一次存储时的一致性状态。

（2）装入最新的日志文件副本，根据日志文件中的内容重做已完成的事务。

介质故障恢复需要数据库管理员来操作，但数据库管理员只需重装最近存储的数据库

副本和有关的日志文件副本，然后执行系统提供的恢复命令即可，其余的恢复操作仍由DBMS自动完成。

除了上述针对系统故障的恢复外，数据库还有其他恢复技术，如检查点恢复技术、数据库镜像技术等。

5.3.2 SQL Server 数据库备份操作

SQL Server提供两种方式进行数据库备份：Microsoft SQL Server Management Studio和T-SQL语句。

1. 使用 Microsoft SQL Server Management Studio 方式备份数据库

使用Microsoft SQL Server Management Studio方式备份数据库使用以下步骤进行。

(1) 打开Microsoft SQL Server Management Studio，并连接到目标服务器。

(2) 在“对象资源管理器”窗口中展开“服务器”，打开“数据库”，右击要备份的数据库，在弹出的快捷菜单中选择“任务”，在弹出的子菜单中单击“备份”命令，打开“备份数据库”窗口，如图5-24所示。

(3) 在“备份类型”中设置“完整”或“差分”或“事务日志”备份类型。

(4) 在“备份集”→“名称”文本框中输入备份集名称。在“说明”中输入对备份集的描述(可选)。

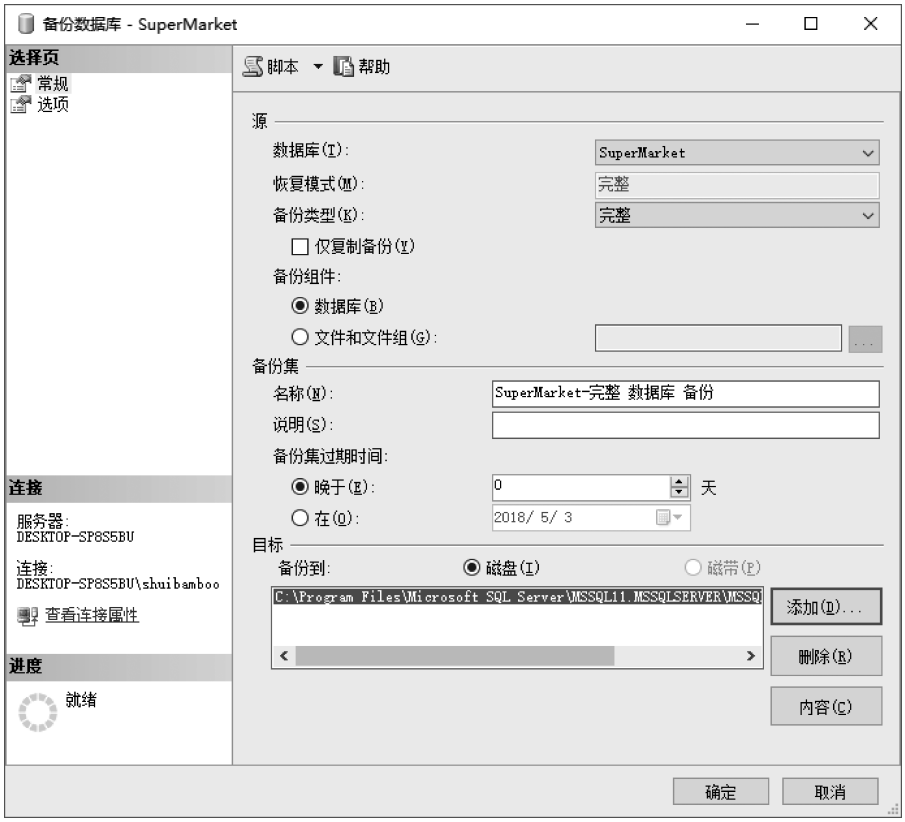


图5-24 “备份数据库”窗口

（5）在“目标”选项下的“备份到”一栏中选择“磁盘”。如果没有出现备份目的地，则单击“添加”按钮以添加到现有的目的地或创建新的目的地。

（6）在“备份数据库”窗口的“选择页”中单击“选项”，如图 5-25 所示，可进行备份介质选项设置。

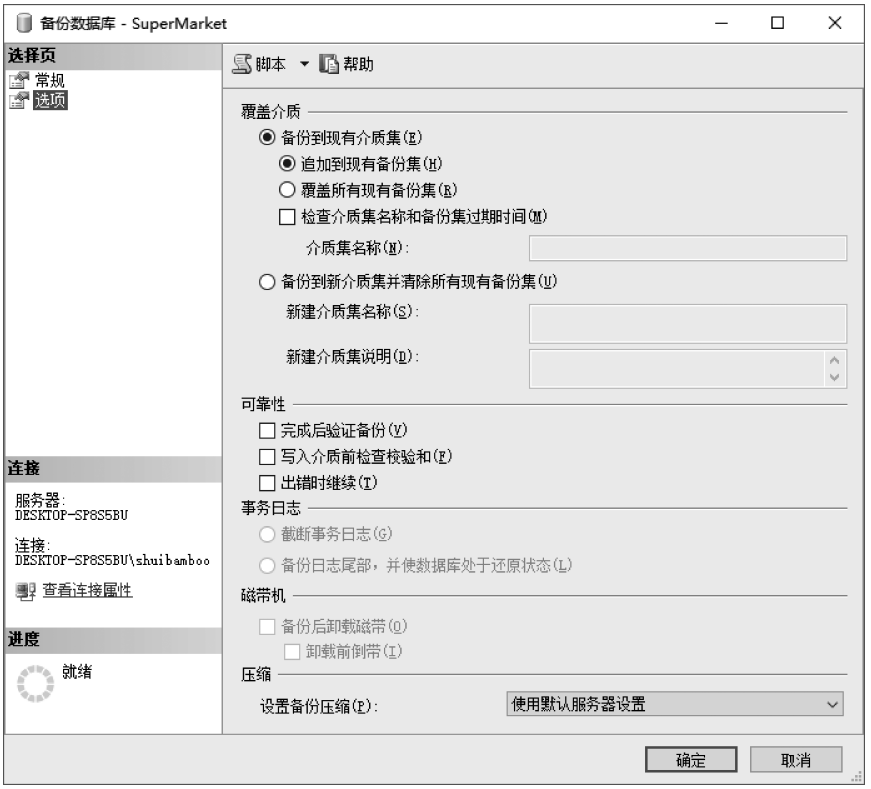


图 5-25　备份数据库选项

（7）单击“确定”按钮完成数据库备份。

2. 使用 T-SQL 语句备份数据库

不同的数据库备份类型，备份数据库的 T-SQL 语句略有不同。

1）完全备份

完全备份数据库的语法格式如下。

```
BACKUP DATABASE database_name
TO DISK = 'backup_device'
[WITH
    DIFFERENTIAL |
    COMPRESSION | NO_COMPRESSION |
    DESCRIPTION = 'text' |
    NAME = 'backup_set_name'
    EXPIREDATE = 'date '|
    RETAINDAYS = days |
    NO_CHECKSUM| CHECKSUM
]
```

各参数说明如下。

database_name：指定一个数据库名，用于对该数据库进行完全备份。

TO DISK＝'backup_device'：指定备份到磁盘或磁带设备上，并指定物理路径。

DIFFERENTIAL：只能与 BACKUP DATABASE 一起使用，指定数据库备份或文件备份应该只包含上次完全备份后更改的数据库或文件备份。差异备份通常比完全备份占用的空间更少。

COMPRESSION | NO_COMPRESSION：适用于 SQL Server 2008 Enterprise 及更高版本，指定是否对此设备执行备份压缩，优于服务器级默认设置。安装时默认行为是不进行备份压缩，但此默认设置可通过设置 backup compression default 服务器配置选项进行更改。COMPRESSION 是显式启用备份压缩，默认情况下，压缩备份时将执行校验和以检测是否存在媒体损坏的情况；NO_COMPRESSION 是显式禁止备份压缩。

DESCRIPTION＝'text'：指定说明备份集的自由格式文本。该字符串最长可以有 255 个字符。

NAME＝'backup_set_name'：指定备份集的名称。名称最长可达 128 个字符。如果不指定 NAME，它将为空。

EXPIREDATE＝'date'：指定备份集到期和允许被覆盖的日期。

RETAINDAYS＝days：指定需要经过多少天才可以覆盖该备份集。如果同时使用这个和 EXPIREDATE 选项，则它的优先级高于 EXPIREDATE。

NO_CHECKSUM|CHECKSUM：控制是否使用备份校验和。NO_CHECKSUM 是显式禁用备份校验和的生成，是默认行为，但压缩备份除外；CHECKSUM 是启用备份校验和。

【例 5-20】 备份 SuperMarket 数据库，备份集名为 SuperMarket_full_20180615，保留 7 天。

```
BACKUP DATABASE SuperMarket
TO DISK = 'E:\Database\supermarket.bak'
WITH
    NAME = 'SuperMarket_full_20180615',
    DESCRIPTION = '数据库完全备份',
    RETAINDAYS = 7
```

2）事务日志备份

数据库备份以后，可以通过备份事务日志来备份数据库备份后的数据库变化，其语法格式如下。

```
BACKUP LOG database_name
TO DISK = 'backup_device'
[WITH
    DESCRIPTION = 'text' |
    NAME = 'backup_set_name' |
    NO_TRUNCATE
]
```

各参数说明如下。

LOG：指定仅备份事务日志。该日志是从上一次成功执行的日志备份到当前日志的末尾。必须创建完全备份，才能创建第一个日志备份。

database_name：指定要备份日志的数据库名。

TO DISK＝'backup_device'：指定备份到磁盘或磁带设备上，并指定物理路径。

NO_TRUNCATE：指定不截断日志，并使数据库引擎尝试执行备份，而不考虑数据库的状态。因此，使用 NO_TRUNCATE 执行的备份可能具有不完整性的元数据。该选项允许在数据库损坏时备份日志；如果不使用 NO_TRUNCATE 选项，则数据库必须联机。

【例 5-21】 备份 SuperMarket 数据库的事务日志，备份集名为 SuperMarket_log_20180615，保留 7 天。

```
BACKUP LOG SuperMarket
TO DISK = 'E:\Database\supermarket.bak'
WITH
    NAME = 'SuperMarket_log_20180615',
    DESCRIPTION = '日志备份',
    RETAINDAYS = 7
```

3）文件和文件组备份

使用 BACKUP DATABASE 语句来实现文件和文件组备份，需要指定某个数据库文件或文件组包含在文件备份中。其语法格式如下。

```
BACKUP DATABASE database_name
FILE = 'logical_file_name' | FILEGROUP = 'logical_filegroup_name'
TO DISK = 'backup_device'
[WITH
    DIFFERENTIAL |
    COMPRESSION | NO_COMPRESSION |
    NAME = 'backup_set_name'
    EXPIREDATE = 'date'|
    RETAINDAYS = days |
    NO_CHECKSUM | CHECKSUM
]
```

各参数说明如下。

FILE＝'logical_file_name'：文件或变量的逻辑名称，其值等于要包含在备份中的文件的逻辑名称。

FILEGROUP＝'logical_filegroup_name'：文件组或变量的逻辑名称，其值等于要包含在备份中的文件组的逻辑名称。在简单回复模式下，只允许对只读文件组执行文件组备份。

【例 5-22】 备份 SuperMarket 数据库文件组 primary，备份集名为 SuperMarket_filegroup_20180615，保留 7 天。

```
BACKUP DATABASE SuperMarket
FILEGROUP = 'primary'
TO DISK = 'E:\Database\supermarket.bak'
WITH
    NAME = 'SuperMarket_filegroup_20180615',
    DESCRIPTION = '文件组备份',
    RETAINDAYS = 7
```

5.3.3 SQL Server 数据库恢复操作

数据库一旦出现故障，如果存在数据库备份，就可以使用备份文件来恢复数据库。SQL Server 提供两种方式进行数据库恢复：Microsoft SQL Server Management Studio 和 T-SQL 语句。

1. 使用 Microsoft SQL Server Management Studio 方式恢复数据库

使用 Microsoft SQL Server Management Studio 方式恢复数据库使用以下步骤进行。

(1) 打开 Microsoft SQL Server Management Studio，并连接到目标服务器。

(2) 在“对象资源管理器”窗口中展开“服务器”，打开“数据库”，右击要恢复的数据库，在弹出的快捷菜单中选择“任务”，在弹出的子菜单中单击“还原”命令，在弹出的子菜单中选择“数据库”命令，打开“还原数据库”窗口，如图 5-26 所示。

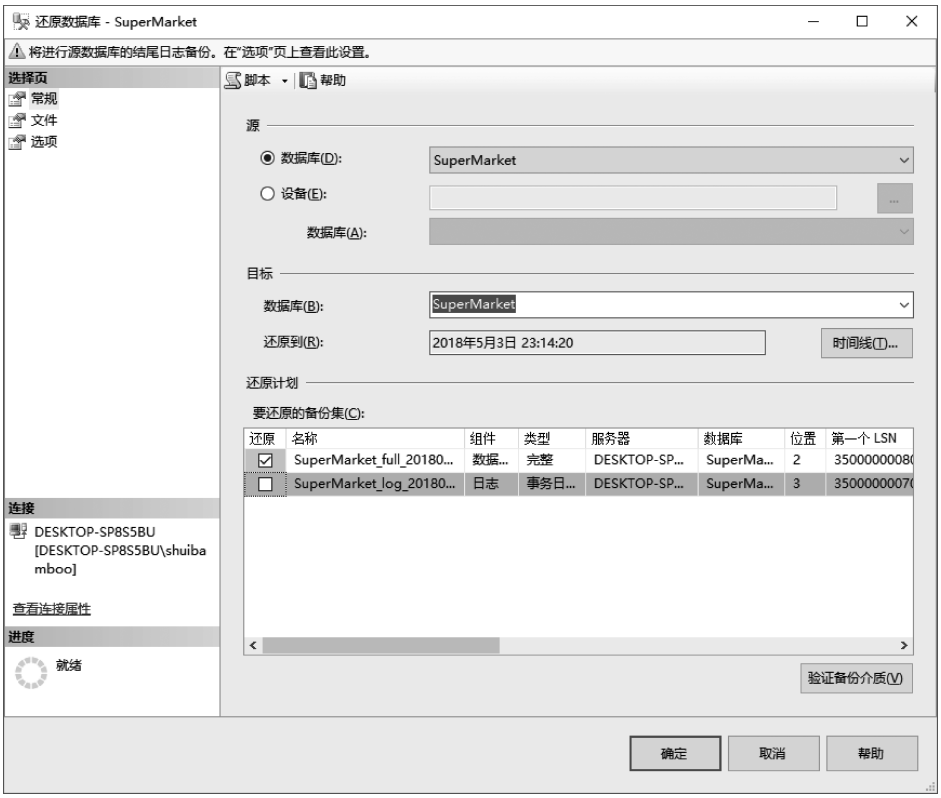


图 5-26 “还原数据库”窗口

(3) 在打开的“还原数据库”窗口中，列出了可用于还原的备份集，选择需要还原的备份集，单击“确定”按钮即可完成数据库还原。

(4) 如果没有列出当前可用的备份集，可选择“源”→“设备”，单击右侧的按钮，打开“选择备份设备”窗口，如图 5-27 所示。

(5) 在“备份介质类型”中选择“文件”或“备份设备”，单击“添加”按钮，定位磁盘文件或备份设备。单击“确定”按钮返回“还原数据库”窗口，选择需要还原的备份集，单击“确定”按钮即可完成数据库的恢复。

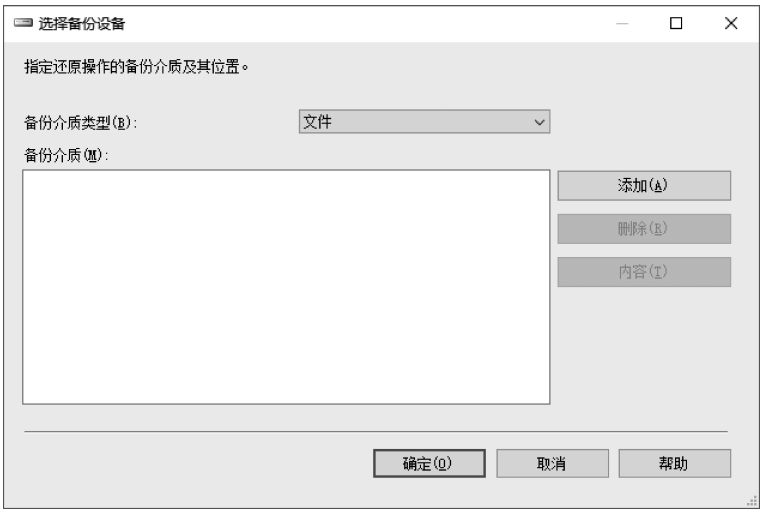


图 5-27 “选择备份设备”窗口

2. 使用 T-SQL 语句恢复数据库

恢复数据库的 T-SQL 语句与备份的 T-SQL 语句一一对应。

1）恢复数据库

语法格式如下。

```
RESTORE DATABASE database_name
FROM DISK = 'backup_device'
[WITH
    RECOVERY | NORECOVERY
    MOVE 'logical_file_name_in_backup' TO 'operating_system_file_name'[,…]|
    FILE = backup_set_file_number |
    REPLACE |
    CHECKSUM | NO_CHECKSUM
]
```

各参数说明如下。

database_name：要恢复的数据库名。

FROM DISK＝'backup_device'：指定要从哪些备份设备恢复。

RECOVERY | NORECOVERY：指示恢复操作是否回滚所有未曾提交的事务。默认的选项是 RECOVERY。RECOVERY 指示恢复操作回滚任何未提交的事务，在恢复过程后即可随时使用数据库。NORECOVERY 指示恢复操作不回滚任何未提交的事务。如果稍后必须应用另一个事务日志，则应指定 NORECOVERY。

MOVE 'logical_file_name_in_backup' TO 'operating_system_file_name'[,…]：指定对于逻辑名称由'logical_file_name_in_backup'指定的数据或日志文件，应当通过将其恢复到'operating_system_file_name'所指定的位置来对其进行移动。

REPLACE：会覆盖所有现有数据库以及相关文件，包括已存在同名的其他数据库或文件。强制还原。

FILE ＝ backup_set_file_number：标识要恢复的备份集。

CHECKSUM | NO_CHECKSUM：默认行为是在存在校验和时验证校验和，在不存在校验和时不进行验证并继续执行操作。CHECKSUM 指定必须验证备份校验和，在备份缺少备份校验和的情况下，该选项会导致恢复操作失败，并发出一条消息表明校验和不存在。默认情况下，当遇到无效校验和时，RESTORE 会报告校验和错误并停止。然而，如果指定了 CONTINUE_AFTER_ERROR，RESTORE 会在返回校验和错误以及包含无效校验和的页面编号之后继续。NO_CHECKSUM 是禁用恢复操作的校验和验证功能。

2）恢复事务日志

语法格式如下。

```
RESTORE LOG database_name
FROM DISK = 'backup_device'
[WITH
    RECOVERY | NORECOVERY
    MOVE 'logical_file_name_in_backup' TO 'operating_system_file_name'[,…]|
    FILE = backup_set_file_number
}
```

3）恢复文件或文件组

语法格式如下。

```
RESTORE DATABASE database_name
FILE = 'logical_file_name' | FILEGROUP = 'logical_filegroup_name'
FROM DISK = 'backup_device'
[WITH
    RECOVERY | NORECOVERY
    MOVE 'logical_file_name_in_backup' TO 'operating_system_file_name'[,…]|
    FILE = backup_set_file_number |
    REPLACE |
    CHECKSUM | NO_CHECKSUM
}
```

【例 5-23】 使用 SuperMarket 数据库的完整数据库备份例 5-20 进行恢复。

```
RESTORE DATABASE SuperMarket
FROM DISK = 'E:\Database\supermarket.bak'
WITH REPLACE, NORECOVERY
```

小　　结

数据库管理与维护功能用以保证数据库中数据的正确有效和安全可靠。本章从数据库的安全性管理、并发控制和数据库的备份及恢复管理三个方面进行了阐述。

数据库的安全性管理是数据库管理系统中非常重要的部分，安全性管理的好坏直接影响数据库数据的安全。本章介绍了 SQL Server 数据库的安全机制，它通过三个级别的认证：第一步验证用户是否是合法的服务器登录名，第二步验证用户是否是要访问的数据库的合法用户，第三步验证用户是否具有适当的操作权限。服务器登录名主要是身份验证，SQL Server 提供了 Windows 验证和混合验证两种验证模式。连接上 SQL Server 服务器

后，用户还需要以一个数据库用户账户及对应的权限访问该数据库中的数据。为了方便对用户和权限的管理，SQL Server 采用角色的概念来管理具有相同权限的一组用户，除了可以根据实际操作情况创建用户定义的角色外，系统还提供了一些预定义好的角色，包括管理服务器一级设置的固定的服务器角色和在数据库一级进行操作的固定的数据库角色。架构是指一组数据库对象的集合，一个数据库用户可以使用多个架构，架构也可以被多个数据库用户使用。权限包括对象权限、语句权限和隐式权限三类，可以为用户授予的权限有两种：一种是对数据进行操作的对象权限，即对数据的增加、删除、修改和查询的权限，另一种是创建对象的语句权限，包括创建表、视图、存储过程等对象的权限。利用 SQL Server 提供的 SSMS 工具和 T-SQL 语句，可以很方便地实现数据库的安全性管理。

本章接着介绍了事务和并发控制的概念。事务在数据库中是非常重要的一个概念，它是保证数据并发控制的基础。事务的特点是事务中的操作是一个完整的工作单元，这些操作，要么全部执行成功，要么全部执行不成功。只要数据库管理系统能够保证系统中一切事务的 ACID 特性，即事务的原子性、一致性、隔离性和持续性，也就保证了数据库处于一致状态。并发控制是当同时执行多个事务时，为了保证一个事务的执行不受其他事务的干扰所采取的措施。并发控制的主要方法是封锁，根据对数据操作的不同，锁可以分为共享锁和排他锁两种，当只对数据做读取操作时，加共享锁，当需要对数据进行修改操作时，需要加排他锁。在一个数据对象上可以同时存在多个共享锁，但只能同时存在一个排他锁。本章介绍了最常用的封锁方法和三级封锁协议。不同的封锁和不同级别的封锁协议所提供的系统一致性保证是不同的。对数据对象施加封锁会带来活锁和死锁问题，数据库一般采用先来先服务、死锁诊断和解除等技术来防止活锁和死锁的发生。为了保证并发执行的事务是正确的，一般要求事务遵守两阶段锁协议，即在一个事务中明显地划分锁申请期和释放期，它是保证事务是可并发执行的充分条件。

本章还介绍了数据库管理与维护中很重要的工作：备份和恢复数据库。SQL Server 支持四种备份方式：完全备份、差分备份、事务日志备份、文件或文件组备份。完全备份是备份整个数据库，不仅包括用户表、系统表、索引、视图、存储过程等所有数据库对象，还包括事务日志部分。差分备份仅备份自上次完全备份以来对数据进行改变的内容。事务日志备份对事务日志进行备份，备份时复制自上次备份以来对数据库所做的改变，仅需要很少的时间。文件或文件组备份是备份磁盘上跟数据库相关的文件。数据库的备份地点可以是磁盘，也可以是磁带。在备份数据库时可以将数据库备份到备份设备上，也可以直接备份在磁盘文件上。数据库的恢复通常是先从完全备份开始，然后恢复最近的差分备份，然后再按备份的顺序恢复后续的日志备份。SQL Server 支持在备份的同时允许用户访问数据库，在恢复数据库过程中是不允许用户访问数据库的。利用 SQL Server 提供的 SSMS 工具和 T-SQL 语句，可以很方便地实现数据库的备份和恢复管理。

习　　题

一、单项选择题

1. SQL Server 的 GRANT 和 REVOKE 语句主要用来维护数据库的(　　)。

A. 可靠性　　B. 一致性　　C. 安全性　　D. 完整性

2. 数据库的(　　)是指数据的正确性和相容性。

A. 并发控制　　B. 完整性　　C. 安全性　　D. 恢复

3. 一个事务执行过程中,其正在访问的数据被其他事务修改,导致处理结果不一致,这是由于违背了事务(　　)特性引起的。

A. 一致性　　B. 原子性　　C. 隔离性　　D. 持久性

4. 如果事务 T 对数据 R 已加 S 锁,则对数据 R(　　)。

A. 不能加 S 锁可加 X 锁　　B. 可加 S 锁不能加 X 锁

C. 可加 S 和 X 锁　　D. 不能加任何锁

5. 数据库中的封锁机制是(　　)的主要方法。

A. 完整性　　B. 并发控制　　C. 安全性　　D. 恢复

6. (　　)可以防止丢失修改,读“脏”数据和不可重复读。

A. 一级封锁协议　　B. 二级封锁协议

C. 三级封锁协议　　D. 两段锁协议

7. 如果对并发操作不加以控制,可能会带来(　　)问题。

A. 死锁　　B. 死机　　C. 不安全　　D. 不一致

8. 在 SQL Server 中提供的四种数据库备份方式,其中(　　)是指将从最近一次完全备份结束以来所有改变的数据备份到数据库。

A. 完全备份　　B. 差分备份

C. 事务日志备份　　D. 文件或文件组备份

9. 下面的 SQL 命令中,用于实现数据控制命令的是(　　)。

A. COMMIT　　B. UPDATE

C. GRANT　　D. SELECT

10. SQL Server 的(　　)权限主要管理用户对数据库对象的访问,例如,这个用户能否进行查询、删除、修改和插入一个表中的行,能否执行一个存储过程。

A. 语句权限　　B. 对象权限

C. 隐式权限　　D. 以上三种权限

11. SQL Server 的(　　)权限是指用户执行数据库操作的权限,即用户执行某些 T-SQL 语句的权力,例如创建和删除对象、备份和恢复数据库。

A. 语句权限　　B. 对象权限

C. 隐式权限　　D. 以上三种权限

12. 系统管理员需要让 Windows 的用户和非 Windows 的用户都能够访问 SQL Server,应该使用(　　)安全模式。

A. Windows 验证模式　　B. 混合验证模式

C. 哪种模式均可　　D. 哪种模式都不能满足要求

13. 一个用户试图连接到一个 SQL Server 上。服务器使用的是混合验证模式,且该用户不是 Windows 的用户(即没有登录 Windows),用户需如何填写登录名和口令框中的内容才能成功连接服务器?(　　)

A. 什么也不用填　　B. 用户的 SQL Server 账号和口令

C. 用户的 Windows 账号和口令　　D. 以上的选项都可以

14. 在 SQL Server 中，角色有服务器角色和数据库角色两种。其中，用户可以创建和删除(　　)。

A. 服务器角色　　B. 数据库角色

C. 服务器角色和数据库角色　　D. 两种角色都不行

15. 角色是一些系统定义好操作权限的用户组，其中的成员是登录账号。(　　)角色不能被增加或删除，只能对其中的成员进行修改。

A. 服务器角色　　B. 数据库角色

C. 操作员角色　　D. 应用程序角色

16. (　　)可以防止一个用户的工作不适当地影响另一个用户的工作。

A. 完整性控制　　B. 并发控制　　C. 安全性控制　　D. 访问控制

17. 下列不属于并发操作带来的问题是(　　)。

A. 不可重复读　　B. 读“脏”数据　　C. 死锁　　D. 丢失修改

18. 数据库管理系统普遍采用(　　)方法来保证调度的正确性。

A. 授权　　B. 封锁　　C. 索引　　D. 日志

19. 如果事务 T 获得了对数据 D 上的排他锁，则 T 对 D(　　)。

A. 既能读又能写　　B. 只能读不能写

C. 只能写不能读　　D. 不能读也不能写

20. 如果有两个事务，同时对数据库中同一数据进行操作，不会引起冲突的操作是(　　)。

A. 两个都是 UPDATE　　B. 一个是 SELECT，一个是 DELETE

C. 两个都是 SELECT　　D. 一个是 DELETE，一个是 SELECT

21. 假设事务 T1 和 T2 对数据库中的数据 D 进行操作，可能有如下几种情况，其中(　　)不会发生冲突。

A. T1 正在写 D，T2 也要写 D　　B. T1 正在写 D，T2 要读 D

C. T1 正在读 D，T2 要写 D　　D. T1 正在读 D，T2 也要读 D

22. 在 SQL Server 中，用户进行数据备份时，应备份(　　)内容。

A. 记录用户数据的所有用户数据库

B. 记录系统信息的系统数据库

C. 记录数据库改变的事务日志

D. 以上所有内容

23. 在 SQL Server 中提供的四种数据库备份方式，其中(　　)是备份制作数据库中所有内容的一个副本。

A. 完全备份　　B. 差分备份

C. 事务日志备份　　D. 文件或文件组备份

24. 在 SQL Server 中提供的四种数据库备份方式，其中(　　)是指将从最近一次日志备份以来所有的事务日志备份到备份设备中。

A. 完全备份　　B. 差分备份

C. 事务日志备份　　D. 文件或文件组备份

25. 在 SQL Server 中提供四种数据库备份方式，其中(　　)是对数据库中的部分文件或文件组进行备份。

A. 完全备份　　B. 差分备份

C. 事务日志备份　　D. 文件或文件组备份

二、简答题

1. SQL Server 的两种身份验证模式的优点分别是什么？

2. 简述数据库角色的作用。

3. 简述用户自定义角色的作用。

4. 什么是事务？事务有哪些特征？

5. 简述数据库中进行并发控制的原因。

6. 简述锁的机制及锁的类型，各类锁之间的兼容性。

7. 简述死锁及其解决办法。

8. 第一次对数据库进行备份时，必须使用哪种备份方式？

9. 差分备份方式备份的是哪段时间的哪些内容？

10. 什么是数据备份？数据备份的类型有哪些？

11. SQL Server 的备份设备是一个独立的物理设备吗？

12. 简述进行数据库备份时，应备份哪些内容？

13. 数据库恢复中 RECOVERY | NORECOVERY 选项的含义是什么？分别在什么时候使用？

14. 什么是数据库的安全性管理？

15. 什么是活锁？简述活锁产生的原因和解决方法。

16. 什么样的并发调度是正确的调度？

17. 根据不同的故障，给出对应的恢复策略和方法。

18. 简述 SQL Server 中常用的三类权限。

19. 简述两段锁协议的概念。

20. 并发操作可能产生哪几类数据不一致？用什么方法能避免各种不一致的情况？

21. 使用封锁技术进行并发操作的控制会带来什么问题？如何解决？

三、编程题

1. 使用 SQL 语句建立一个 Windows 身份验证的登录账号，登录名为 win_login。

2. 使用 SQL 语句建立一个 SQL Server 身份验证的登录账号，登录名为 SQL_login，密码为 123456。

3. 删除 Windows 身份验证的登录账号 win_login。

4. 建立一个数据库用户，用户名为 SQL_dbuser，对应的登录名为 SQL Server 身份验证的 SQL _login。

5. 将 SQL Server 身份验证的 SQL_user 登录名添加到系统管理员角色中。

6. 创建一个 SQL Server 账号 SQL_user2，并将该账号创建为 SuperMarket 数据库的用户。再授予 SQL_user2 用户查询 Category 表的权限；授予 SQL_user2 用户修改 Goods 表中 SalePrice 的权限。

7. 创建一个事务，将所有啤酒类商品的售价增加 2 元，将所有毛巾类的售价降低 1 元，并提交。

8. 分别实现数据库 SuperMarket 的备份和恢复操作。

实　　验

一、实验目的

熟悉和掌握数据库安全性管理的方法。

掌握事务机制，会创建事务。

熟悉和掌握数据库备份和恢复的方法。

二、实验平台

操作系统：Windows XP/7/8/10。

数据库管理系统：SQL Server 2012。

三、实验内容

在超市管理数据库 SuperMarket 的基础上进行实验。要求使用 SSMS 工具和 SQL 语句两种方式进行操作。

1. 设置数据库的身份验证模式(Windows 验证和混合验证)。

2. 建立登录账户，修改登录账户属性。

(1) 使用 Microsoft SQL Server Management Studio 方式创建、修改登录账号。

(2) 使用 SQL 语句创建、修改登录账号。

3. 建立数据库用户。

(1) 使用 Microsoft SQL Server Management Studio 方式创建、删除数据库用户。

(2) 使用 SQL 语句创建、删除数据库用户。

4. 权限管理。

(1) 使用 Microsoft SQL Server Management Studio 方式进行权限管理。

(2) 使用 SQL 语句进行权限管理。

5. 定义数据库角色。

(1) 使用 Microsoft SQL Server Management Studio 方式创建用户自定义数据库角色。

(2) 使用 SQL 语句创建用户自定义数据库角色。

6. 事务编写。

(1) 编写一个事务处理：某学生买 5 袋薯片，如中间出现故障则回滚事务。

(2) 编写一个事务，当学生购买商品时，插入购买明细到 SaleBill 中，并修改 Goods 表以保持数据一致性，如中间出现故障则回滚事务。

(3) 编写一个事务，当撤销某个学生购买明细时，删除 SaleBill 中的记录，然后修改 Goods 表以保持数据一致性，如中间出现故障则回滚事务。

7. 备份数据库。

(1) 使用 Microsoft SQL Server Management Studio 方式进行数据备份。

(2) 使用 T-SQL 语句进行数据备份。

8. 恢复数据库。

(1) 使用 Microsoft SQL Server Management Studio 方式进行数据恢复。

(2) 使用 T-SQL 语句进行数据恢复。