**江苏科技大学**

**课 设 报 告**

课 程： 数据库原理与安全课程设计

学 院： 计算机学院

专 业： 信息安全

班 级： 1822107101

学 号： 182210710119

姓 名： 陈四贵

指导老师： 江苏科技大学 刘噶琼

目 录

[一、设计目的 2](#_Toc90631897)

[二、设计项目1·数据完整性控制 2](#_Toc90631898)

[【设计内容】 2](#_Toc90631899)

[【学习要求】 2](#_Toc90631900)

[【设计原理】 2](#_Toc90631901)

[【设计步骤】 4](#_Toc90631902)

[三、设计项目2·数据库的恢复 8](#_Toc90631903)

[【设计内容】 8](#_Toc90631904)

[【学习要求】 8](#_Toc90631905)

[【设计原理】 8](#_Toc90631906)

[【设计步骤】 8](#_Toc90631907)

[四、设计项目3·并发性控制与封锁 12](#_Toc90631908)

[【设计内容】 12](#_Toc90631909)

[【学习要求】 12](#_Toc90631910)

[【设计原理】 12](#_Toc90631911)

[【设计步骤】 15](#_Toc90631912)

[五、设计小结 17](#_Toc90631913)

# 一、设计目的

1．安装SQL Server2008R2

2．熟练使用SQL Server2008R2常规操作

3．完成实验项目，记载步骤及相关截图

4．完成课程设计报告

# 二、设计项目1·数据完整性控制

## 【设计内容】

1．通过对数据库安全性含义的理解；实现安全性控制一般方法；

2．通过SQL Server安全性机制，理解SQL Server身份验证模式、SQL Server登录账号、SQL Server数据库账号的操作。

## 【学习要求】

1．理解据库安全性含义；安全性控制一般方法；

2．掌握SQL Server安全性机制；SQL Server身份验证模式。

## 【设计原理】

1．数据完整性控制一般方法

数据完整性是指数据的精确性和可靠性，主要应用于保证数据库中数据的质量。为了防止数据库中存在不符合语义规定的数据。

数据完整性分为四类：

（1）实体完整性（保证表中每一行数据在表中是唯一的）；

（2）域完整性（保证列的值的有效性和正确性（CHECK约束、DEFAULT约束、NOT NULL 约束））；

约束就是限制，定义约束就是定义限制条件。主要有主键约束（PRIMARY KEY）、唯一约束（UNIQUE）、外键约束（FOREIGN KEY）、检查约束（CHECK）、默认约束（DEFAULT）、非空约束（NOT NULL）。

（3）参照完整性（表之间，多个表数据的一致性与更新的同步性）；

（4）用户定义完整性。

2．安全性机制及身份验证模式

Sqlserver安全管理主要概念包括：登录名、服务器角色、数据库用户、数据库角色、架构。

（1）登录名

Windows身份登录：

A．可以为每个windows用户（组）映射一个windows账号，登录时候不需要sqlserver账号，会直接根据计算机名和所在的windows身份，映射到对应的账号，例如下面两个windows用户创建的sqlserver 的windows身份：

B．sqlserver创建windows登录名时，标准的格式：计算机名\windows用户（组）名，例如，登录账号tanglin\lenovo，前者tanglin是计算机名，后者lenovo是windows用户名。

而TANGLIN\WindowsUserGroup则是一个用户组，该用户组的所有windows成员都可以映射到该登录名，使用同一个账号访问。

混合登录（windows身份验证、sqlserver身份两者同时验证）

例如上面的A、B以及sa等登录名，都是混合登录，需要windows身份验证、sqlserver身份两者同时验证。

（2）服务器角色

帮助管理sqlserver服务器上的权限，sqlserver提供了这些角色。这些角色是用于对其他主体进行分组的安全主体。服务器级角色的权限作用域为服务器范围（“角色”类似于 Windows 操作系统中的“组”）。

SQL Server 提供了9种固定服务器角色，无法更改或授予固定服务器角色。（注意：从 SQL Server 2012 开始，您可以创建用户定义的服务器角色，并将服务器级权限添加到用户定义的服务器角色）。

|  |  |  |
| --- | --- | --- |
| 序号 | 角色名称 | 说明 |
| 1 | sysadmin | sysadmin 固定服务器角色的成员可以在服务器中执行任何活动。 |
| 2 | serveradmin | serveradmin 固定服务器角色的成员可以更改服务器范围内的配置选项并关闭服务器。 |
| 3 | securityadmin | securityadmin 固定服务器角色的成员管理登录名及其属性。 他们可以 GRANT、DENY 和 REVOKE 服务器级权限。 他们还可以 GRANT、DENY 和 REVOKE 数据库级权限（如果他们具有数据库的访问权限）。此外，他们还可以重置 SQL Server 登录名的密码。  **注意：**能够授予数据库引擎的访问权限和配置用户权限的能力使得安全管理员可以分配大多数服务器权限。 securityadmin 角色应视为与 sysadmin 角色等效。 |
| 4 | processadmin | processadmin 固定服务器角色的成员可以终止在 SQL Server 实例中运行的进程。 |
| 5 | setupadmin | setupadmin 固定服务器角色的成员可以通过使用 Transact-SQL 语句添加和删除链接服务器。（在使用 Management Studio 时需要 sysadmin 成员身份） |
| 6 | bulkadmin | bulkadmin 固定服务器角色的成员可以运行 BULK INSERT 语句。 |
| 7 | diskadmin | diskadmin 固定服务器角色用于管理磁盘文件。 |
| 8 | dbcreator | dbcreator 固定服务器角色的成员可以创建、更改、删除和还原任何数据库。 |
| 9 | public | 每个 SQL Server 登录名均属于 public 服务器角色。 如果未向某个服务器主体授予或拒绝对某个安全对象的特定权限，该用户将继承授予该对象的 public 角色的权限。当您希望该对象对所有用户可用时，只需对任何对象分配 public 权限即可。您无法更改 public 中的成员关系。  **注意：**public 的实现方式与其他角色不同。 但是，可以从 public 授予、拒绝或撤销权限。 |

注意：public 的实现方式与其他角色不同。 但是，可以从 public 授予、拒绝或撤销权限。只能向用户定义的服务器角色中添加服务器级权限。 若要列出服务器级别的权限，请执行下面的语句。

SELECT \* FROM sys.fn\_builtin \_permissions('SERVER') ORDER BY permission\_name;

（3）数据库用户

A．数据库用户名与sqlserver登录名一一对应

在一个数据库中，一个数据库用户名只能对应一个登录名，一个登录名也只能对应一个数据库用户名。登录名只有通过数据库的用户才能对数据库进行访问权限。一个登录名在每个数据库中只能有一个用户名，从而通过该用户名对该数据库进行访问。

B．数据库用户默认架构

架构目的是将不同的数据库对象分成不同的组，而默认架构是用户访问数据库对象（表，视图，函数等等）时，首先查找的架构对象，无需在对象前面加上架构名称（例如表dbo.table\_A中的dbo）。而其他不属于该架构的对象（除了dbo架构）访问时都要加上架构名进行标识。

C．数据库用户拥有的架构

数据库用户拥有的架构，对其有所有的使用权限（包括dll和dml权限），当然，架构拥有者获得了架构的dll权限，也要同时获得数据库的dll权限，才能创建该架构下的创建各种数据库对象，但dml权限无需获得数据库的dll权限。

D．数据库用户权限的来源

（1）包含的数据库角色身份的所有权限

数据库用户包含的 所有数据库角色的所有权限，都映射到该数据库用户，例如：拥有的架构（等同与该数据库用户拥有的架构）/其他形式赋予的权限等 数据库赋予的dml权限；

（2）拥有的架构的所有权限 / 其他架构赋予的权限；

（3）数据库对象赋予的权限；

（4）数据库用户或数据库角色 转移或授予的权限。

E．默认架构的对象权限和其他架构的权限

对于默认架构，使用该权限时直接用数据库对象名进行访问，而其他的架构（无论是拥有的架构还是非拥有的）权限的对象，都要用到架构名前面标识进行。

## 【设计步骤】

(1)建立一个数据表A，设置其主键为自增字段，保证行数据在数据表中的唯一性；

(2)在数据表A中设置各种约束；

|  |
| --- |
| create table A(  ID bigint primary key identity(1,1),  Sname varchar(10) not null,  sex char(2) default '男',  age int check(age<=20)  ) |

此处主键为ID，其中Sname属性不为空，sex默认为男，年龄检测需要小于等于20。



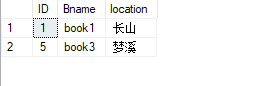
此处插入信息中FAN信息不满足age约束



由于第二次执行时FAN对应数据出错无法插入，导致编号3的CA数据被事务回滚删除，重新执行后记录编号变为5

(3)再建立一个数据表B，设置此表为表A的子表（类似一个订单，有订单信息以及订单详细物品明细），尝试建立一个触发器使得删除主表A中某数据时，子表B中所属的明细数据随之删除，保证两表之间数据的统一性；

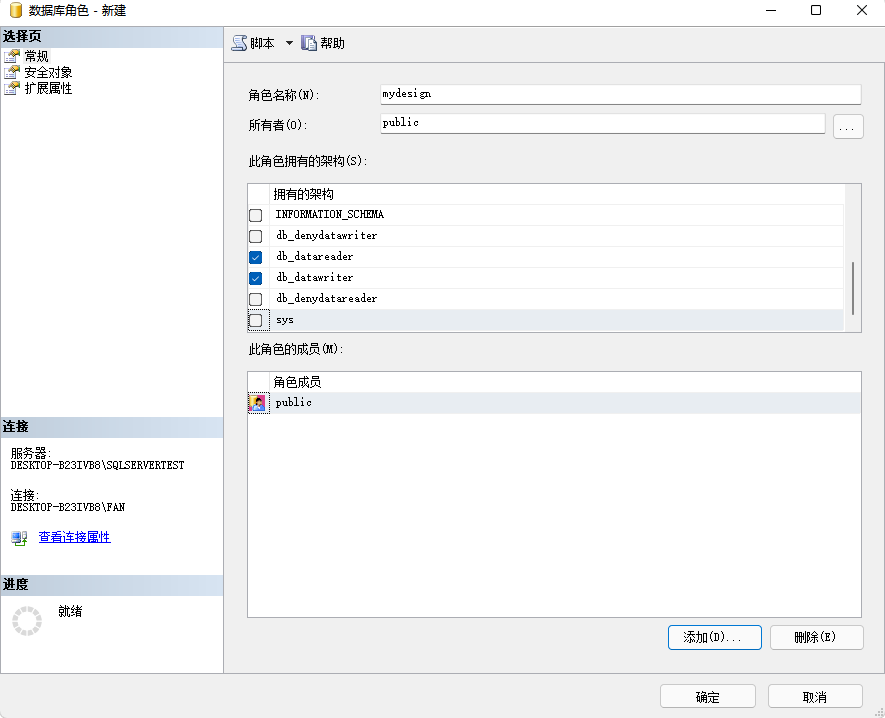
|  |
| --- |
| create table B(  ID bigint foreign key references A(ID),  Bname varchar(20),  location varchar(30)  )  insert into B (ID,Bname,location) values  (1,'book1','长山'),  (5,'book3','梦溪') |



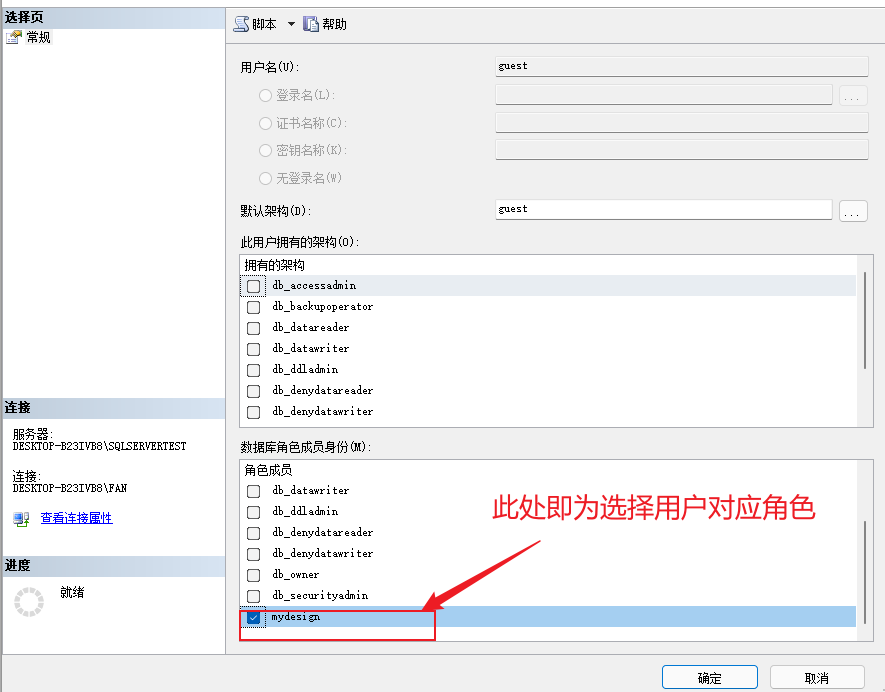
表B结构数据如图所示

|  |
| --- |
| create trigger A\_delete\_trigger on B  for delete  as  begin  delete B where ID IN (select ID from DELETED)  PRINT('B表中相应记录也被删除!')  end |

(4)尝试设置数据库用户自定义角色，设置数据库登录用户并配置其所属角色；



此处新建一个数据库角色



(5)学习设置数据库身份验证模式，观察设置不同模式后登录的不同之处。



此时为默认的windows身份验证方式



使用修改过密码的sa用户登录

windows身份验证表示由windows操作系统来验证登录的合法性。只要成功登录windows，登录SQL Server时就不需要再验证。  
SQL身份验证表示由SQL Server数据库系统来验证登录的合法性。登录SQL Server时需要提供用户名和密码。前者直接使用本机pc用户即可，后者需要根据之前在数据库服务中存储的用户信息进行登录。

# 三、设计项目2·数据库的恢复

## 【设计内容】

1．对数据库恢复问题，建立数据库恢复及其实现技术；

2．实现数据库的故障和恢复策略。

## 【学习要求】

1．理解数据库恢复的含义和实现技术；

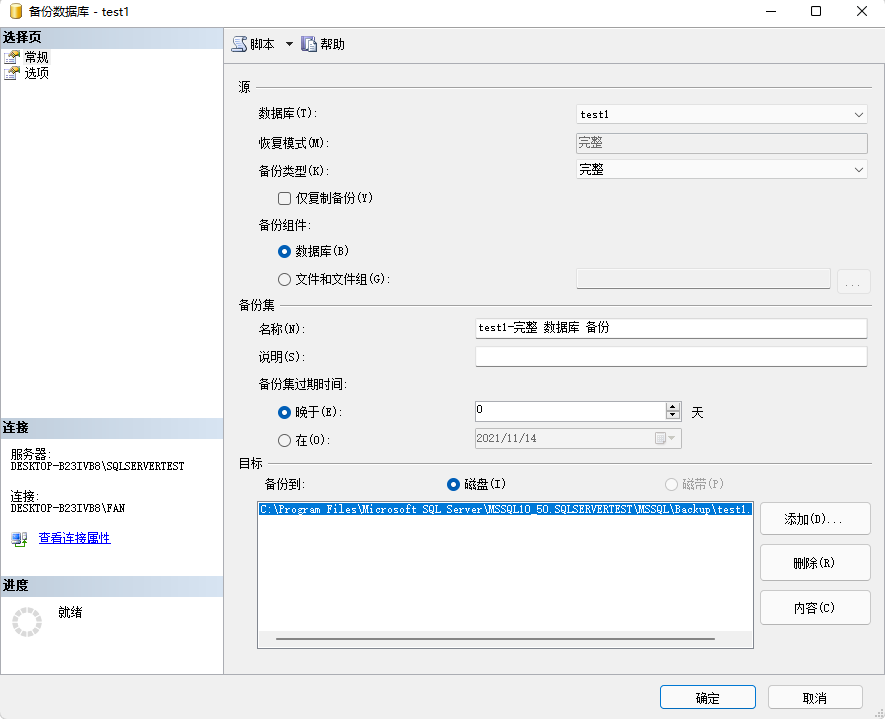
2．掌握数据库的故障和恢复的策略。

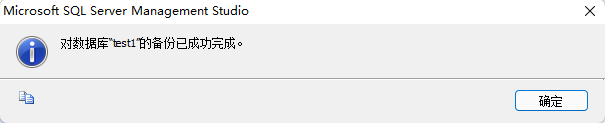
## 【设计原理】

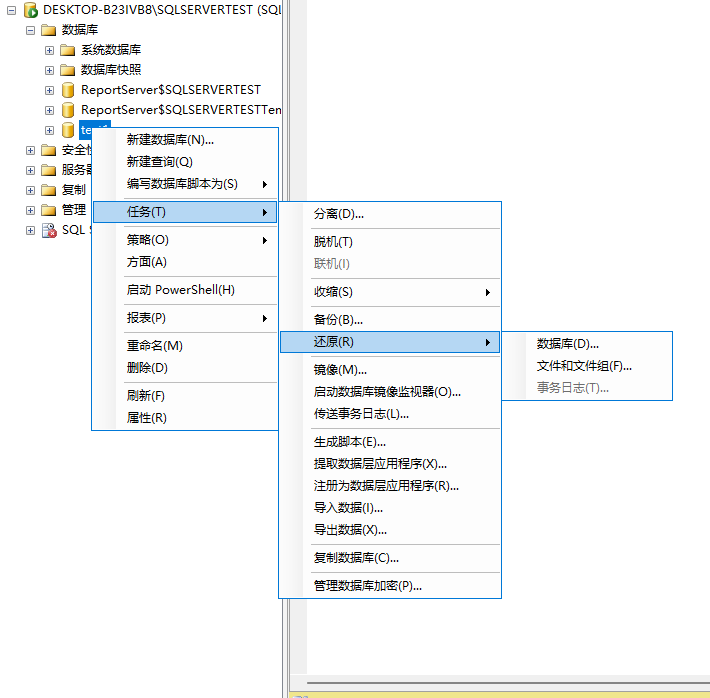
任何数据库都涉及数据的容灾处理，数据备份与恢复是数据库安全性的基本保障，数据库系统本身皆内置了备份、恢复功能，应掌握其使用。

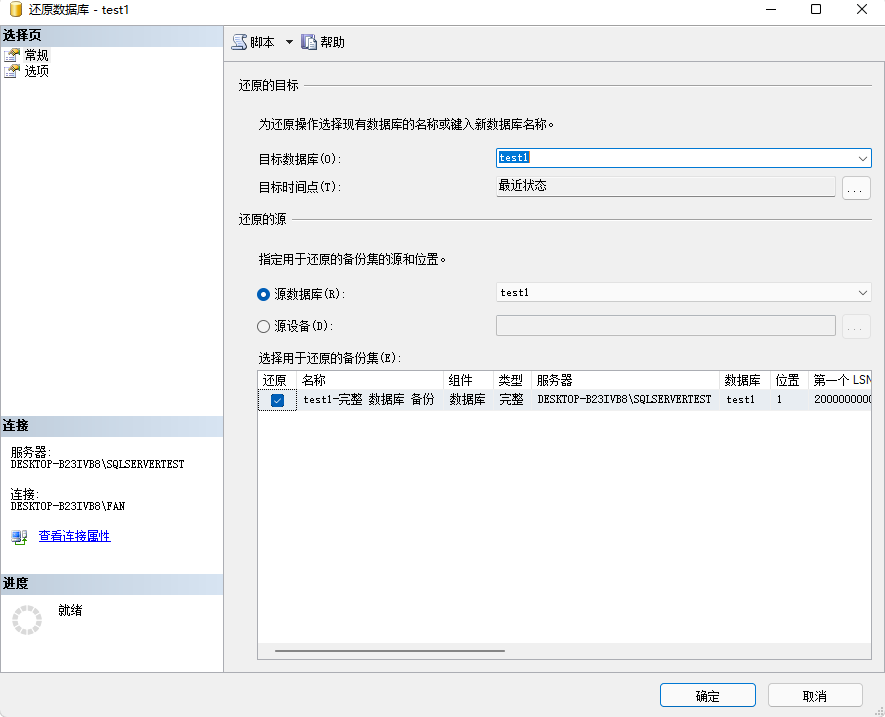
## 【设计步骤】

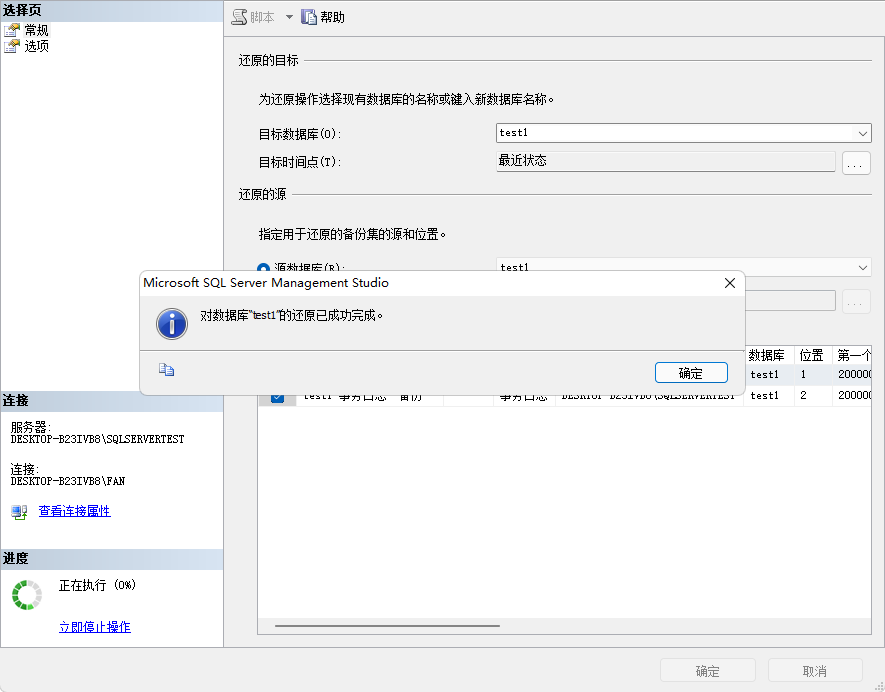
(1)建立一个数据库，右键菜单->任务->备份/还原，按照界面指引操作一遍；

此处可以针对备份类型进行详细设置，如选择完整、差异或日志备份等备份类型；选择备份的文件和文件组；选择备份过期时间等



在进行还原时同样可以选择恢复类型：





(2)查阅相关资料，简述SQL Server故障和恢复的策略。

SQL server故障类型：

事务故障：事务故障是单独一个事务出问题而不能执行下去，并不影响其他事务的执行；由系统自动完成，对用户是透明的。

②系统故障：系统故障是故障导致系统重启，当前运行中的事务及刚刚提交的事务会导致数据库不一致；

③介质故障：介质故障则是数据库文件的存储介质如硬盘发生故障导致数据丢失。是指外存故障，例如磁盘损坏、磁头碰撞，瞬时强磁场干扰等。

相应恢复策略:前两种可由DBMS自主进行，后一种需要进行数据库的备份和恢复，此处重点论述这方面内容。根据备份和恢复方案可分为四类：

①完全数据库备份与恢复：服务器备份整个数据库。系统发生故障时完全数据库备份就是恢复的基准。当用户数据库较小或者数据库很少进行修改时，可以只选择完全数据库备份，因为完全备份一个小数据库的时间是可以接受的或者少量的数据丢失是可接受的。  
②差异备份与恢复：服务器备份当前数据库与上一次完全备份数据库时相比更改的部分，以及备份过程中发生的活动、事务日志中所有未提交的事务。需要对经常修改的数据库进行完全数据库备份和日志备份。  
③事务日志备份与恢复：服务器备份上一次执行BACKUP LOG语句到现在执行 BACKUP LOG语句中间的这段事务日志。必须明确的是：在进行日志备份之前一定至少要完成一次完全数据库备份，否则在还原的时候将无法还原事务日志。当完全数据库和事务日志备份策略还原所消耗的时间用户不能接受时可以考虑使用差异备份策略，该策略可以减少还原的时间，因为它保留最后一次完全备份数据库的更改，而不必使用多个较大的日志文件。  
④数据库文件和文件组备份与恢复：在这种方式中管理员可以将数据库大卸八块，将多个数据文件或者文件组分别进行备份，而不是传统的备份整个数据文件。需要注意的是，执行这种备份方式时必须同时执行日志备份。数据库数量相当大完全备份可能需要几十个钟头这种情况下可以考虑分别对它的数据文件或者文件组进行备份以减少备份所消耗的时间。

# 四、设计项目3·并发性控制与封锁

## 【设计内容】

1．数据库并发性含义、事务性质、并发操作与数据不一致性。

## 【学习要求】

1．理解数据库并发性含义和事务性质；

2．掌握并发操作与数据不一致性的概念。

## 【设计原理】

1、防止并发问题

防止并发问题主要有两种途径，分别是使用锁和设置隔离级别。使用锁来处理并发问题更加精细，需要对每一个操作对象进行锁的设置，这期间难免会带来争用的问题以及其他性能问题；设置隔离级别的方法是对用户的一次连接进行的整体设置，粒度较粗但是可以避免锁有时会带来的争用问题。设计者可以根据设计需要自行选择这两种方式。

使用锁来解决什么问题？

归纳来说，锁主要解决4类主要问题：脏读（dirty read）、不可重复读（No-Repeatable Read）、幻影（phantom）、丢失更新。

例如：脏读

何为脏读呢？当事务读取一条记录，而该记录是另一事务尚未完成的一部分时，就会发生脏读。如果每一个事物都是正常完成了，那么这看上去像是不会存在问题。但是，如果该事务被回滚了呢？！看下面这个例子：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 事务1命令 | 事务2命令 | 逻辑数据库值 | 未提交数据库值 | 事务2显示内容 |
| BEGIN TRAN |  | 3 |  |  |
| UPDATE col=5 | BEGIN TRAN | 3 | 5 |  |
| SELECT anything | SELECT @var=col | 3 | 5 | 5 |
| ROLLBACK | UPDATE anything | 3 |  | 5 |
|  | SET whatever=@var |  |  |  |

这里就出问题了！

事务2现在正在使用一个已经无效了的值！如果想要试图返回去审核，想看看该数字是从哪里来的，会因为发现根本无法追踪，这是非常令人头疼的事。

针对这种脏读的情况，如果使用SQL Server默认的事务隔离级别（称作READ COMMITTED），就不会发生这种事了。

2、事务

SQL Server事务的使用是为了确保数据的一致性。例如银行转账，必须出和入作为一个事务来处理，否则会出现已出而未入的情况。通常写法：

begin tran

--sql 语句1

--sql 语句2

--sql 语句3

commit tran

上面写法存在隐患，当操作（增删改）是由null引发的错误时，事务会跳过错误继续执行正常的语句。例如：

--创建表Student

create table Student(Name nvarchar(20) not null)

--建立事务

begin tran

inserted into Student(Name) values (null)

inserted into Student(Name) values ('小札')

inserted into Student(Name) values (null)

commit tran

--由null引发的错误，insert，delete，update都会跳过错误继续执行

上面结果会多一条数据为“小札”。为了避免了这样的问题：

有三种方法：其中@@error，@@trancount是全局变量，只要发生错误，@@error不等于0，只要执行一次事务，@@trancount就+1，回滚会变为0。

方法一：xact\_abort on/off  on：开启，事务一旦出问题，全部回滚  off：关闭，不检查事务是否发生错误。

set xact\_abort on

begin tran

--sql语句1

--sql语句2

--sql语句3

Commit

方法二：每条操作语句后面判断是否回滚。

begin tran

--sql语句1

if @@error<>0

begin

rollback tran

return --这里除了return跳出，也可以使用goto+标签跳出事务

end

--sql语句2

if @@error<>0

begin

rollback tran

return

end

commit tran

方法三：try  catch

begin tran

begin try

--sql语句1

--sql语句2

--sql语句3

end try

begin catch

if @@trancount>0

rollback tran

end catch

if @@trancount>0

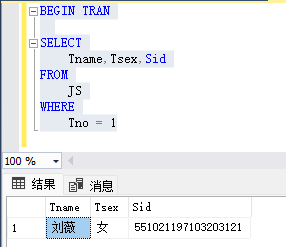
commit tran

## 【设计步骤】

（1）编写一个SQL存储过程，使用SQL Server默认的事务隔离级别执行针对相同表的两个事务操作，分析与不使用事务隔离级别的区别；

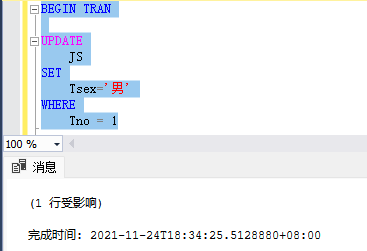
在SQL Server中，已提交读隔离级别是建立连接时的默认隔离级别。这个级别存在两种类型：已提交读和已提交读快照隔离级别。已提交读级别会在读数据之前等待，直到阻塞锁被释放。已提交读快照级别会在数据被其他事务阻塞时使用行版本控制来读数据最后一次提交的版本。

使用已提交读级别：



查询JS表中的Tname、Tsex、Sid

现在假设另一事务在事务打开状态下更改了Tsex。打开第二个查询窗口并执行以下批来UPDATE Tsex，但不提交事务：



更改JS表中的Tsex

 这个UPDATE语句会正常运行。一行受到了影响，即使数据在这个事务还没有运行完之前已被查询窗口1中的事务读取。因为已提交读级别并不会在事务结束前保持用于SELECT语句的共享锁。共享锁会在数据读取之后立即被SQL Server释放。

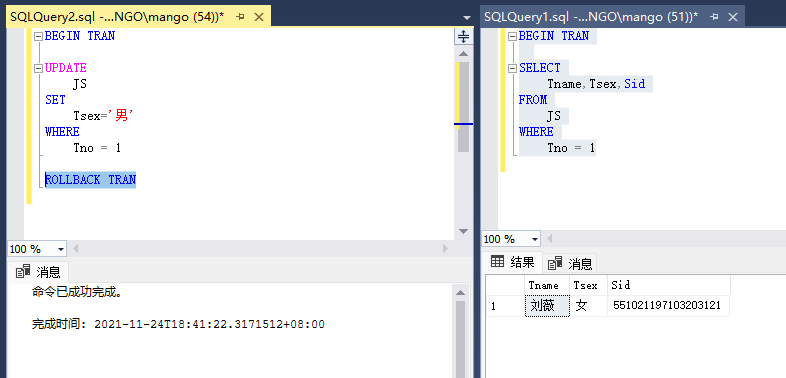
现在切换到查询窗口1并尝试再次读数据：

由于SELECT语句被阻塞，因此这个查询并没有结束。SQL Server会尝试在ContactID= 1的键上获取一个共享锁，但是由于在查询窗口2中的UPDATE语句对其有一个排他锁，因此这个操作不可能完成。虽然查询窗口2处于已提交读级别(由于您没有更改默认级别)，但排他锁依然存在。这个阻塞将持续存在，因为数据更改的排他锁会一直保持直到事务结束。

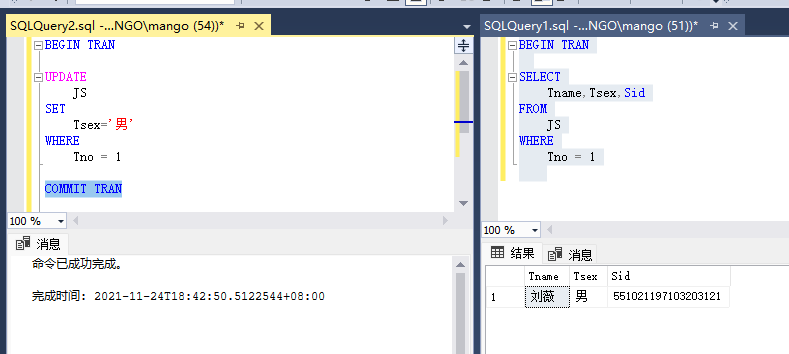


SELECT语句被阻塞

这时，在查询窗口2中执行一个ROLLBACK TRAN语句来回滚UPDATE语句。然后切换回查询窗口1。可以看到，查询窗口1中的查询完成了，并且其结果与以前的一样。查询窗口2中的事务结束的时候，锁被释放了，以至查询窗口1中的查询不再被阻塞。由于查询窗口2中的事务回滚，因此查询窗口1中得到的结果是原来的数据。如果查询窗口2中的事务被提交，则查询窗口1中会得到新的数据作为结果。



回滚后，SELECT语句执行结果

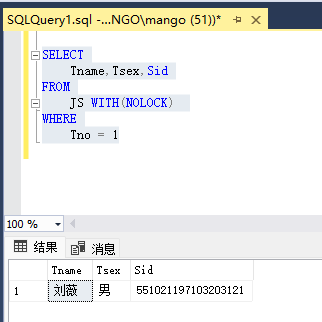


提交后，SELECT语句结果

可以看出，在(默认)已提交读级别中SQL Server会等到排他锁释放之后再进行读操作，以此来获取真正的提交数据。还可以看出，共享锁会持续到数据被读取之后，而排他锁会持续到事务提交之后。在许多事务几乎同时更改数据的时候这种行为可能会造成问题。在这些情况下，由于排他锁造成的阻塞，读数据会非常慢。

（2）存储过程中尝试sqlserver锁争用及nolock,rowlock的使用，查阅资料理解其原理。

若在(1)中的窗口1读取数据的话，可以使用这样的方法：



使用NOLOCK读取JS表中数据。nolock可以忽略锁，直接从数据库读取数据。此选项被选中时，SQL Server 在读取或修改数据时不加任何锁。 在这种情况下，用户有可能读取到未完成事务。rowlock使用行级锁，而不使用粒度更粗的页级锁和表级锁。

# 五、设计小结

设计数据库不应仅仅停留在创建表的使用上，在设计过程中还应考虑各种操作可能造成的影响，最大程度保证数据安全，保证数据库的正常使用。