**新建项目**

**计算机学院（软件学院）实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **学生姓名** | 徐可可 | **学号** | 171530425 | **实验成绩** |  |
| **专业** | 软件工程 | **班级** | 软工四班 | **实验日期** | 19年12月24日 |
| **课程名称** | 数据库原理 | | | **任课教师** | 马丽 |
| **实验名称** | 事务控制 | | | **实验序号** | 8 |
| **实验地点** | S411 | **实验台号** | 25 | **指导教师** | 马丽 |
| 1. **实验目的及要求**   熟练使用DBMS提供的功能组件来实现事务的创建，了解DBMS提供的事务并发控制机制，进一步理解事务的概念及特性，了解事务是如何保证数据库的一致性。 | | | | | |
| 1. **实验内容（或实验原理、实验拓扑）**   1. 定义事务，理解事务的ACID特性。  2. 事务故障恢复，理解事务的ACID特性。  3. 事务的并发控制，理解事务的ACID特性。 | | | | | |
| 1. **实验设备与环境**   1. 阿里云服务器：Intel(R) Xeon(R) Platinum 8269CY CPU @ 2. 50GHz、2G内存、40G高效云盘、Windows Server 2019 数据中心版、Microsoft SQL Server 2008 R2  2. 高配笔记本：Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz  、16G内存、48TB分布式硬盘、Windows 10 1909政府版、SQL Server Management Studio 18.4  3. 实验室电脑：英特尔 酷睿2 双核 T7700 @ 2.40GHz、3G内存、128 G虚拟硬盘、Windows 7 旗舰版 32位 SP1、Microsoft SQL Server 2008 R2 | | | | | |
| 1. **实验设计方案（包括实验步骤、设计思想、算法描述或开发流程等）**   在学生-课程数据库上完成如下操作。  1. 事务的定义  （1）了解事务运行模式  在查询编辑器中输入如下语句并执行，最后语句中包含语义错误，查看前面语句执行情况。了解在没有显示定义事务的情况下，DBMS默认每个SQL语句就是一个事务。发生错误后，只回滚一个SQL语句。  --select \* from sc  --select \* from c  insert into C(CNO,CN) values('c21','rjgc')  insert into C(CNO,CN) values('c22','txyl')  insert into C(CNO,CN) values('c21','gmtjs')  （2）显示定义事务   1. 定义一事务，包含前面的三条插入语句，最后语句中包含语义错误。执行该事务，与前面的执行结果进行比较，分析结果。 2. 修改定义中的错误，以rollback和 commit两种方式结束事务。查看被更新表的数据，说明rollback的 commit不同。   2. 事务故障恢复  事务故障破坏事务的原子性。事务故障后，系统自动强行回滚（rollback）该事务。即利用日志撤销此事务已对数据库的更新，保持事务的原子性。对提供检测点的DBMS，事务的回滚与设置的检测点有关。  在“生—课程”学数据库上，执行下面的事务，分析结果，阐述设置存储点的作用。  begin tran t1  select \* from sc  insert into SC(SNO,CNO) values('s01','c05')  select \* from sc  save tran t1  update sc set grade=60 where sno=’s01’ and cno=’c05’  select \* from sc  rollback tran t1  select \* from sc  比较每次查询的结果，说明 save tran t1的功能。把rollback tran t1改为rollback看一下，结果如何？  3. 事务的并发控制  大多数DBMS为并发事务提供封锁请求，有共享锁和排它锁。SQL SERVER中，为了模拟并发环境，打开多个查询窗口即可。  在“学生—课程”数据库上完成如下操作。  （1）在一个查询窗口中执行事务T1  begin tran T1  SELECT \* from SC where SNO='s07'  （2）在另一个查询窗口中执行事务T2，此时事务T1还未结束。  begin tran T2  SELECT \* from SC where SNO='s07'  比较T2与T1的结果。  （3）返回事务T1继续运行事务，进行数据修改并查询，例如：  Update sc set grade=grade+2 where sno=’s07’  Select \* from where sno=’s07’  观察结果。  （4）回到事务T2，事务T2进行同样操作，进行数据修改并查询。  Update sc set grade=grade+2 where sno=’s07’  Select \* from sc where sno=’s07’  说明此时的状态。  （5）回到事务T1，提交事务。即执行commit。  （6）返回到事务T2，说明此时的状态和结果。  （7）然后强行关闭两个查询，即T1已提交，T2不提交。  （8）再执行Select \* from sc where sno=’s07’，解释结果。对发生了故障的事务T2，系统是如何做的？保证了事务的什么特性？  \*4. 多粒度封锁  SQL SERVER的封锁操作是在增、删、改、查等语句中添加WITH子句来完成的，封锁方式和粒度有以下几种。  TABLOCK:对表实施封锁，  HOLDLOCK,TABLOCK  TABLOCKX  UPDLOCK  NOLOCK | | | | | |

|  |
| --- |
| 1. **实验结果（包括设计效果、测试数据、运行结果等）**   **在学生-课程数据库上完成如下操作。**  **1.事务的定义**  **（1）了解事务运行模式**  **在查询编辑器中输入如下语句并执行，最后语句中包含语义错误，查看前面语句执行情况。了解在没有显示定义事务的情况下，DBMS默认每个SQL语句就是一个事务。发生错误后，只回滚一个SQL语句。**  --select \* from sc  --select \* from c  insert into C(CNO,CN) values('c21','rjgc')  insert into C(CNO,CN) values('c22','txyl')  insert into C(CNO,CN) values('c21','gmtjs')    **（2）显示定义事务**  **①　定义一事务，包含前面的三条插入语句，最后语句中包含语义错误。执行该事务，与前面的执行结果进行比较，分析结果。**  BEGIN TRANSACTION  insert into Course(CNO,CN) values('c22','txyl')  insert into Course(CNO,CN) values('c21','gmtjs')  insert into Course(CNO,CN) values('c21','rjgc')  ROLLBACK  **执行结果：**  select \* from course    创建一个事务，以ROLLBACK结尾时，只要事务中有一句不能执行，那么都不会执行。  **②　修改定义中的错误，以rollback和 commit两种方式结束事务。查看被更新表的数据，说明rollback的 commit不同。**  不同之处：  以rollback结尾时，不论成功与否，都会回滚；而以commit结尾时，只要能成功执行，就能进行更改。  BEGIN TRANSACTION  insert into Course(CNO,CN) values('D22','txyl')  insert into Course(CNO,CN) values('D21','gmtjs')  COMMIT    BEGIN TRANSACTION  insert into Course(CNO,CN) values('E22','txyl')  insert into Course(CNO,CN) values('E21','gmtjs')  ROLLBACK  select \* from course    **2．事务故障恢复**  **事务故障破坏事务的原子性。事务故障后，系统自动强行回滚（rollback）该事务。即利用日志撤销此事务已对数据库的更新，保持事务的原子性。对提供检测点的DBMS，事务的回滚与设置的检测点有关。**  **在“学生—课程”学数据库上，执行下面的事务，分析结果，阐述设置存储点的作用。**  begin tran t1  select \* from sc  insert into SC(SNO,CNO) values('s01','c05')  select \* from sc  save tran t1  update sc set grade=60 where sno=’s01’ and cno=’c05’  select \* from sc  rollback tran t1  select \* from sc  **比较每次查询的结果，说明 save tran t1的功能。把rollback tran t1改为rollback看一下，结果如何？**  第一次select    第二次select    第三次select    第四次select      Save tran t1的功能是将这条语句上面所执行的事务记录下来，执行rollback tran t1后，会回到刚才记录的那个状态。  **把rollback tran t1改为rollback，执行结果：**  第一次select    第二次select    第三次select    第四次select      若把rollback tran t1改为rollback，会发现回到最初事务没有执行的那个状态。  **3．事务的并发控制**  **大多数DBMS为并发事务提供封锁请求，有共享锁和排它锁。SQL SERVER中，为了模拟并发环境，打开多个查询窗口即可。**  **在“学生—课程”数据库上完成如下操作。**  **（1）在一个查询窗口中执行事务T1**  begin tran T1  SELECT \* from SC where SNO='s07'  **（2）在另一个查询窗口中执行事务T2，此时事务T1还未结束。**  begin tran T2  SELECT \* from SC where SNO='s07'  **比较T2与T1的结果。**  T1和T2的执行结果一样    **（3）返回事务T1继续运行事务，进行数据修改并查询，例如：**  Update sc set grade=grade+2 where sno=’s07’  Select \* from where sno=’s07’  **观察结果。**    **（4）回到事务T2，事务T2进行同样操作，进行数据修改并查询。**  Update sc set grade=grade+2 where sno=’s07’  Select \* from sc where sno=’s07’  **说明此时的状态。**  T2一直在执行中    **（5）回到事务T1，提交事务。即执行commit。**  **（6）返回到事务T2，说明此时的状态和结果。**  T2成功执行，并且在T1加2的基础上又再加上2。    **（7）然后强行关闭两个查询，即T1已提交，T2不提交。**  **（8）再执行Select \* from sc where sno=’s07’，解释结果。对发生了故障的事务T2，系统是如何做的？保证了事务的什么特性？**    因为事务有ACID特性，事务T1提交，而T2没有提交，所以查看到的内容是T1执行后的内容。  保证了事务的ACID特性。  **系统是如何做的？**  事务的ACID特性是由关系数据库管理系统来实现的。数据库管理系统采用日志来保证事务的原子性、一致性和持久性。日志记录了事务对数据库所做的更新，如果某个事务在执行过程中发生错误，就可以根据日志，撤销事务对数据库已做的更新，使数据库退回到执行事务前的初始状态。  数据库管理系统采用锁机制来实现事务的隔离性。当多个事务同时更新数据库中相同的数据时，只允许持有锁的事务能更新该数据，其他事务必须等待，直到前一个事务释放了锁，其他事务才有机会更新该数据。  **1. 事务的回滚与设置检查点的关系。**  事务回滚：是数据库返回到事务开始的状态：事务在运行过程中发生某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的更新操作全部撤销，使数据库回滚到事务开始时的状态。  检查点：出于性能方面的考虑，数据库引擎对内存（缓冲区缓存）中的数据库页进行修改，但在每次更改后不将这些页写入磁盘。 相反，数据库引擎定期发出对每个数据库的检查点命令。 “检查点”将当前内存中已修改的页和事务日志信息从内存写入磁盘，并记录有关事务日志的信息。对于自动、手动和内部检查点，在数据库恢复期间只有在最新检查点后所做的修改需要前滚。 这将减少恢复数据库所需的时间。 |
| 1. **实验小结（包括收获、心得体会、注意事项、存在问题及解决办法、建议等）**   **1．事务的概念( Transaction )**  所谓事务是用户定义的一个数据库操作序列，这些操作要么都做，要么都不做，是一个不可分割的工作单位。  关系数据库中，事务可以是一条SQL语句、一组SQL语句。  在SQL语言中，定义事务的语句有三条：  Begin Transaction 开始  Commit 结束  Rollback 回滚  **2．事务开始：BEGIN TRANSACTION**  标记一个显式本地事务的起始点。BEGIN TRANSACTION将 @@TRANCOUNT 加 1。  **语法结构：**  BEGIN TRAN [ SACTION ] [ transaction\_name | @tran\_name\_variable  [ WITH MARK [ 'description' ] ] ]  **参数说明：**  transaction\_name：是给事务分配的名称。transaction\_name 必须遵循标识符规则，但是不允许标识符多于 32 个字符。仅在嵌套的 BEGIN...COMMIT 或 BEGIN...ROLLBACK 语句的最外语句对上使用事务名。  @tran\_name\_variable：是用户定义的、含有有效事务名称的变量的名称。必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。  WITH MARK ['description']：指定在日志中标记事务。Description 是描述该标记的字符串。  如果使用了 WITH MARK，则必须指定事务名。WITH MARK 允许将事务日志还原到命名标记。  **3．事务提交：COMMIT TRANSACTION**  标志一个成功的隐性事务或用户定义事务的结束。如果 @@TRANCOUNT 为 1，COMMIT TRANSACTION 使得自从事务开始以来所执行的所有数据修改成为数据库的永久部分，释放连接占用的资源，并将 @@TRANCOUNT 减少到 0。如果 @@TRANCOUNT 大于 1，则 COMMIT TRANSACTION 使 @@TRANCOUNT 按 1 递减。  **语法结构：**  COMMIT [ TRAN [ SACTION ] [ transaction\_name | @tran\_name\_variable ] ]  **参数说明：**  transaction\_name：Microsoft SQL Serve 忽略该参数。transaction\_name 指定由前面的 BEGIN TRANSACTION 指派的事务名称。transaction\_name 必须遵循标识符的规则，但只使用事务名称的前 32 个字符。通过向程序员指明 COMMIT TRANSACTION 与哪些嵌套的 BEGIN TRANSACTION 相关联，transaction\_name 可作为帮助阅读的一种方法。  @tran\_name\_variable：是用户定义的、含有有效事务名称的变量的名称。必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。  **4．事务回滚：ROLLBACK TRANSACTION**  将显式事务或隐性事务回滚到事务的起点或事务内的某个保存点。  **语法结构：**  ROLLBACK [ TRAN [ SACTION ]  [ transaction\_name | @tran\_name\_variable  | savepoint\_name | @savepoint\_variable ] ]  **参数说明：**  transaction\_name：是给 BEGIN TRANSACTION 上的事务指派的名称。transaction\_name 必须符合标识符规则，但只使用事务名称的前 32 个字符。嵌套事务时，transaction\_name 必须是来自最远的 BEGIN TRANSACTION 语句的名称。  @tran\_name\_variable：是用户定义的、含有有效事务名称的变量的名称。必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。  savepoint\_name：是来自 SAVE TRANSACTION 语句的 savepoint\_name。savepoint\_name 必须符合标识符规则。当条件回滚只影响事务的一部分时使用 savepoint\_name。  @savepoint\_variable：是用户定义的、含有有效保存点名称的变量的名称。必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。 |

|  |
| --- |
| 1. **附录（包括作品、流程图、源程序及命令清单等）**  * **temp.sql**   -- 在学生-课程数据库上完成如下操作。  --  1.事务的定义  -- （1）了解事务运行模式  -- 在查询编辑器中输入如下语句并执行，最后语句中包含语义错误，查看前面语句执行情况。了解在没有显示定义事务的情况下，DBMS默认每个SQL语句就是一个事务。发生错误后，只回滚一个SQL语句。  --select \* from sc  --select \* from c  insert into C(CNO,CN) values('c21','rjgc')  insert into C(CNO,CN) values('c22','txyl')  insert into C(CNO,CN) values('c21','gmtjs')    -- （2）显示定义事务  -- ①　定义一事务，包含前面的三条插入语句，最后语句中包含语义错误。执行该事务，与前面的执行结果进行比较，分析结果。  BEGIN TRANSACTION  insert into Course(CNO,CN) values('c22','txyl')  insert into Course(CNO,CN) values('c21','gmtjs')  insert into Course(CNO,CN) values('c21','rjgc')  ROLLBACK  -- 执行结果：  select \* from course    -- 创建一个事务，以ROLLBACK结尾时，只要事务中有一句不能执行，那么都不会执行。  -- ②　修改定义中的错误，以rollback和 commit两种方式结束事务。查看被更新表的数据，说明rollback的 commit不同。  -- 不同之处：  -- 以rollback结尾时，不论成功与否，都会回滚；而以commit结尾时，只要能成功执行，就能进行更改。  BEGIN TRANSACTION  insert into Course(CNO,CN) values('D22','txyl')  insert into Course(CNO,CN) values('D21','gmtjs')  COMMIT    BEGIN TRANSACTION  insert into Course(CNO,CN) values('E22','txyl')  insert into Course(CNO,CN) values('E21','gmtjs')  ROLLBACK  select \* from course    -- 2．事务故障恢复  -- 事务故障破坏事务的原子性。事务故障后，系统自动强行回滚（rollback）该事务。即利用日志撤销此事务已对数据库的更新，保持事务的原子性。对提供检测点的DBMS，事务的回滚与设置的检测点有关。  -- 在“学生—课程”学数据库上，执行下面的事务，分析结果，阐述设置存储点的作用。  begin tran t1  select \* from sc  insert into SC(SNO,CNO) values('s01','c05')  select \* from sc  save tran t1  update sc set grade=60 where sno='s01' and cno='c05'  select \* from sc  rollback tran t1  select \* from sc  -- 比较每次查询的结果，说明 save tran t1的功能。把rollback tran t1改为rollback看一下，结果如何？    -- 第一次select       第二次select         第三次select       第四次select  -- Save tran t1的功能是将这条语句上面所执行的事务记录下来，执行rollback tran t1后，会回到刚才记录的那个状态。  -- 把rollback tran t1改为rollback，执行结果：    -- 第一次select       第二次select         第三次select       第四次select  -- 若把rollback tran t1改为rollback，会发现回到最初事务没有执行的那个状态。  -- 3．事务的并发控制  -- 大多数DBMS为并发事务提供封锁请求，有共享锁和排它锁。SQL SERVER中，为了模拟并发环境，打开多个查询窗口即可。  -- 在“学生—课程”数据库上完成如下操作。  -- （1）在一个查询窗口中执行事务T1  begin tran T1  SELECT \* from SC where SNO='s07'  -- （2）在另一个查询窗口中执行事务T2，此时事务T1还未结束。  begin tran T2  SELECT \* from SC where SNO='s07'  -- 比较T2与T1的结果。  -- T1和T2的执行结果一样    -- （3）返回事务T1继续运行事务，进行数据修改并查询，例如：  Update sc set grade=grade+2 where sno='s07'  Select \* from sc where sno='s07'  -- 观察结果。    -- （4）回到事务T2，事务T2进行同样操作，进行数据修改并查询。  Update sc set grade=grade+2 where sno='s07'  Select \* from  sc where sno='s07'  -- 说明此时的状态。  -- T2一直在执行中    -- （5）回到事务T1，提交事务。即执行commit。  -- （6）返回到事务T2，说明此时的状态和结果。  -- T2成功执行，并且在T1加2的基础上又再加上2。    -- （7）然后强行关闭两个查询，即T1已提交，T2不提交。  -- （8）再执行Select \* from sc where sno=’s07’，解释结果。对发生了故障的事务T2，系统是如何做的？保证了事务的什么特性？    -- 因为事务有ACID特性，事务T1提交，而T2没有提交，所以查看到的内容是T1执行后的内容。  -- 保证了事务的ACID特性。  -- 系统是如何做的？  -- 事务的ACID特性是由关系数据库管理系统来实现的。数据库管理系统采用日志来保证事务的原子性、一致性和持久性。日志记录了事务对数据库所做的更新，如果某个事务在执行过程中发生错误，就可以根据日志，撤销事务对数据库已做的更新，使数据库退回到执行事务前的初始状态。  -- 数据库管理系统采用锁机制来实现事务的隔离性。当多个事务同时更新数据库中相同的数据时，只允许持有锁的事务能更新该数据，其他事务必须等待，直到前一个事务释放了锁，其他事务才有机会更新该数据。  -- 1. 事务的回滚与设置检查点的关系。  -- 事务回滚：是数据库返回到事务开始的状态：事务在运行过程中发生某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的更新操作全部撤销，使数据库回滚到事务开始时的状态。  -- 检查点：出于性能方面的考虑，数据库引擎对内存（缓冲区缓存）中的数据库页进行修改，但在每次更改后不将这些页写入磁盘。 相反，数据库引擎定期发出对每个数据库的检查点命令。 “检查点”将当前内存中已修改的页和事务日志信息从内存写入磁盘，并记录有关事务日志的信息。对于自动、手动和内部检查点，在数据库恢复期间只有在最新检查点后所做的修改需要前滚。 这将减少恢复数据库所需的时间。 |