下篇的内容很多都会在工作中用到，尤其是可编程对象，那些年我们写过的存储过程，有木有？到目前为止很多大型传统企业仍然很依赖存储过程。这部分主要难理解的部分是事务和锁机制这块，本文会进行简单的阐述没。虽然很多SQL命令可以通过工具自动生成，但如果能通过使用记下的话速度会更快，那么留给自己思考的时间就越多。此外，由于锁这部分知识比较复杂，不同的数据库厂商的实现也有不同，SQL SERVER除了我们常见的共享锁、排它锁（包括表级、页级、行级），还有一种被称为自旋锁的内存锁，这部分内容会在之后的T-SQL深入解析部分再做介绍。言归正传，让我们回到T-SQL的世界咯，希望一天我能成为这个世界里的一只小小的功夫熊猫，写起SQL来，下笔如有神。



**熊二恭祝大家猴年猴赛雷！**

此外，刚仔细学习了下汤雪华大神关于12306领域模型设计的文章，对于思路是一种很好的启迪，推荐下<http://www.cnblogs.com/netfocus/p/5187241.html>，博主xuanbg在评论中提及的动态分裂的思路也是棒棒哒。

本节主要介绍常见的DML操作，一般的添删改查INSERT、UPDATE、DELETE(TRUNCATE),以及特殊一点的MERGE。其中T-SQL支持一下五种类型的INSERT，如下所示。

|  |  |
| --- | --- |
| 语句类型 | 解释与示例 |
| INSERT VALUES | 标准方式：INSERT INTO dbo.Orders(orderid, orderdate, empid, custid) VALUES(10001, '20160207', 3, 'A') |
| INSERT SELECT | 使用便捷：INSERT INTO dbo.Orders(orderid, orderdate, empid, custid)  SELECT 10001, '20120207', 2, 'B' UNION ALL SELECT 10002, '20120207', 3, 'C' |
| INSERT EXEC | INSERT INTO dbo.Orders(orderid, orderdate, empid, custid)  EXEC Sales.GetOrder @country = 'China' |
| SELECT INTO | SELECT courtry, region, city INTO dbo.locations FROM Sales.Customers EXCEPT  SELECT courtry, region, city FROM HR.Employees |
| BULK INSERT | 类似SSIS的导入功能  BULK INSERT dbo.Orders FROM 'C;\orders.txt'  WITH (DATAFILETYPE = 'char', FIELDTERMINATOR = ',', ROWTERMINATOR = '\n' ) |

接下来，介绍IDENTITY标识列的相关知识，以及T-SQL对sequence的支持，和ORACLE中序列一致。需要注意的是标识值无论操作是否成功都会自动增长，因此当需要真心连续的记录号时，需要自己的替代方案。新增的序列对象是标准的SQL功能，它与标识列属性不同，是一个不会绑定到特定表中列的对象，需要时查询获取即可。

|  |  |
| --- | --- |
| 操作类型 | 解释与示例 |
| 创建标识列 | CREATE TABLE dbo.T1 (keycol INT NOT NULL IDENTITY(1, 1) CONSTRAINT PK\_T1 PRIMARY KEY) |
| 插入数据（有标识列的情况下） | 注意忽略标识列，INSERT INTO dbo.T1(datacol) VALUES(‘AAAAA’) |
| 获取当前标识号 | SELECT SCOPE\_IDENTITY(), @@identity, IDENT\_CURRENT('Sales.Orders')  第一列获取当前作用域下的标识号，第二列获取会话生成的最后一个标识号（无论作用域），最后一个获取全局的标记号，与会话无关 |
| 显示插入标识 | SET IDENTITY\_INSERT表名 ON/OFF |
| 创建序列 | CREATE SEQUENCE dbo.SeqOrderIDS AS INT MIN VALUE 1 CYCLE; |
| 使用序列 | SELECT NEXT VALUE FOR dbo.SeqOrderIDS |

* 删除和更新操作

|  |  |
| --- | --- |
| 操作类型 | 解释与示例 |
| 一般删除 | DELETE FROM dbo.Orders WHERE orderdate < ‘20160207’ |
| TRANCATE语句 | TRANCATE TABLE dbo.T1，测试时经常使用，更高效，其实相当于先删除表再创建表，而不是delete那样按条删除。DELETE是用完全日志模式，TRANCATE使用最小日志模式 |
| 基于联接的DELETE | 场景：从Orders表中删除所有与Customers表中美国客户相关的行  **标准方式**：DELETE FROM dbo.Orders WHERE EXISTS (  SELECT \* FROM dbo.Customer AS WHERE Orders.Custid = C.Custid AND C.Country = ‘USA’)  联接DELETE：DELETE FROM O FROM dbo.Orders AS O  JOIN dbo.Customers AS C ON O.custid = C.custid WHERE C.country = ‘China’ |
| 一般更新 | UPDATE dbo.OrderDetails SET discount = discount + 0.05 WHERE productid = 50 |
| 基于联接的UPDATE | 场景：对客户1的所有订单明细增加5%折扣  **标准方式**：UPDATE dbo.OrderDetails SET discount += 0.05 WHERE EXISTS(  SELECT \* FROM dbo.Orders AS O WHERE O.orderid = OrderDetails.orderid AND O.custid = 1)  联接UPDATE：UPDATE OD SET discount += 0.05 FROM dbo.OrderDetails AS OD  JOIN dbo.Orders AS O ON OD.orderid = O.orderid WHERE O.custid =1 |
| 赋值UPDATE | DECLARE @nextval AS INT;  UPDATE dbo.Sequences SET @nextval = val+= 1 WHERE id = ‘SEQ1’ |

* 合并数据和OUTPUT字句

从2008版本开始，T-SQL新增了一个MERGE操作符，相当于其他DML操作的组合，此外为了减少查询次数，可以通过OUTPUT字句将更新的操作输出（类似于触发器的功能，包含inserted、deleted隐藏表），便于构建相应的流水表，不过实话实说通过业务来执行流水操作，比SQL组合更加合理，继续使用表格将相关应用表述出来。

|  |  |
| --- | --- |
| 操作类型 | 解释与示例 |
| 合并数据MERGE | MERGE INTO dbo.Customers AS TGT  USING dbo.CustomerStage AS SRC ON TGT.custid = SRC.custid  WHEN MATCHED THEN UPDATE SET TGT.company = SRC.companyname, TGT.phone = SRC.phone, TGT.address = SRC.address  WHEN NOT MATCHED THEN INSERT (custid, companyname, phone, address) VALUES (SRC.custid, SRC.companyname, SRC.phone, SRC.address) |
| 通过表表达式修改数据 | WITH C AS(  SELECT custid, OD.orderid, productid, discount, discount + 0.05 AS newdiscount  FROM dbo.OrderDetails AS OD JOIN dbo.Orders AS O ON OD.orderid = O.orderid WHERE O.custid = 1 )  UPDATE C SET discount = newdiscount  补充一点是，内部的查询操作支持TOP关键字哦 |
| OUTPUT字句 | 场景：从Orders表中删除所有与Customers表中美国客户相关的行  **标准方式**：DELETE FROM dbo.Orders WHERE EXISTS (  SELECT \* FROM dbo.Customer AS WHERE Orders.Custid = C.Custid AND C.Country = ‘USA’)  联接DELETE：DELETE FROM O FROM dbo.Orders AS O  JOIN dbo.Customers AS C ON O.custid = C.custid WHERE C.country = ‘China’ |
| INSERT OUTPUT | INSERT INTO dbo.T1(datacol) OUTPUT inserted.keycol, inserted.datacol  SELECT lastname FROM HR.Employees WHERE country = 'China'  其中datacol是需要返回的属性 |
| DELETE OUTPUT | DELETE FROM dbo.Orders OUTPUT deleted.orderid, deleted.orderid, deleted.empid, deleted.custid  WHERE orderdate < '20160101' |
| UPDATE OUTPUT | UPDATE dbo.OrderDetails SET discount += 0.05  OUTPUT inserted.productid, deleted.discount AS olddiscount, inserted.discount AS newdiscount WHERE productid = 51 |

事务的概念早已为大家所熟知，想提的一点是其也可以称之为工作单元，包含查询和修改数据的多种活动，UnitOfWork工作单元这个企业架构设计模式实际上也是其实现之一。实际中，最常见的是将插入订单和插入订单详细放入一个事务中，事务的ACID属性及简单事务应用示例如下。

原子性(Atom): 事务是一个原子的工作单元，一起提交或撤销。

一致性(Consistency): 其是一个主观概念，取决于应用程序的需求，指数据的状态，与之后数据库的隔离级别紧密关联。

隔离性(Isolation): 其实一种控制访问数据的机制，在T-SQL中，支持锁和行版本控制两种模式来处理隔离。

持久性(Duration): 数据修改在写入数据文件前，会先写入日志文件，但出现故障时，会通过重做和撤销来恢复数据。

|  |
| --- |
| DECLARE @neworderid AS INT  INSERT INTO Sales.Order(custid, orderdate) VALUES (34, ‘20160213’)  SET @neworderid = SCOPE\_IDENTITY()  INSERT INTO Sales.OrderDetail(ordered, productid, unitprice, qty, discount) VALUES (@@orderid, 11, 14.00, 12, 0.000) |

* 锁和阻塞

正如之前所提到的T-SQL支持两种模式来处理隔离，一种是**锁**，这是一种”**悲观式并发**“，在默认的READ COMMITED隔离级别下，一旦一个事务中修改数据，那么这个将不能被其他事务读取，因为会给该数据加上排它锁，而当读取数据时获取共享锁，其他事务可以并行读取；另一种是**行版本控制技术**，是一种“**乐观式并发**“，其默认的隔离级别为READ COMMITED SNAPSHOT，事务中修改数据时，其他事务时可以进行读取操作的。

接下来介绍数据库中可以锁定的资源，包括行、页、表(对象)、数据库，按序锁定的资源粒度越来越大。行驻留在页中，而是包含表或索引数据的物理数据块。更复杂的可锁定资源包括范围、分配单元、堆&B树等，这儿暂不深究。在SQL SERVER中，如果要获得某个资源类型的锁，首先要获得起对应更高粒度级别上的**意向锁**，例如获得一个行上排他锁，那么该事务需要获取行所在页的意向排它锁和一个拥有该页对象的意向排它锁，**意向锁的目的在于便于在更高粒度级别有效检测不相容的锁请求**，用一个简单的表格来描述锁模式情况下锁的兼容性。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 请求的锁模式 | 授予了排它锁(X) | 授予了共享锁(S) | 授予了意向排它锁(IX) | 授予了意向共享锁(IS) |
| 能否授予请求排它锁 | No | No | No | No |
| 能否授予请求共享锁 | No | Yes | No | Yes |
| 能否授予意向排它锁 | No | No | Yes | Yes |
| 能否授予意向排它锁 | No | Yes | Yes | Yes |

看到这儿不禁要问，为什么不都使用最小粒度的锁，这样的并发性不是更好么?实际上锁是需要消耗资源的，因此需要在时间和空间上折衷。在默认情况下，系统首先获取细粒度的锁，并在某些情况下，触发锁升级，例如一条语句中获取5000个行锁，那么将升级为页锁。此外，T-SQL支持ALTER TABLE语句设置LOCK\_ESCALATION控制锁升级行为，包括是否支持锁升级和发生升级时的粒度（如页升级为表或分区）。

一般情况下，阻塞的出现是正常的，比如一个读操作等待排他锁的释放，但有时阻塞时间过长，严重影响响应时需要排除阻塞。可以通过如下几种查询动态视图的方式来查看系统中当前的阻塞并排除相关阻塞。

|  |  |
| --- | --- |
| 操作类型 | 解释与示例 |
| 查询会话相关锁信息 | SELECT request\_session\_id AS spid, resource\_type AS restype, resource\_database\_id AS dbid,  DB\_NAME(resource\_database\_id) AS dbname, resource\_description AS res, resource\_associated\_entity\_id AS resid,  request\_mode AS mode, request\_status AS status  FROM **sys.dm\_tran\_locks**  其中spid表示进程ID，restype表示锁定的资源类型(KEY, PAGE, Database, object)，mode表示锁模式，status表示是否授予了锁 |
| 查询连接相关信息 | SELECT session\_id AS spid, connect\_time, last\_read, last\_write, most\_recent\_sql\_handle  FROM **sys.dm\_exec\_connections** WHERE session\_id IN (60, 61)  其中connect\_time表示连接时间，write&read\_time表示读写时间，most\_recent\_sql\_handle表示该连接中最近的批处理语句的句柄。接下来可以通过一个简单的APPLY表运算符获取相应的SQL语句  SELECT session\_id AS spid, text  FROM sys.dm\_exec\_connections CROSS APPLY **sys.dm\_exec\_sql\_text**(most\_recent\_sql\_handle) AS st  WHERE session\_id IN (60, 61) |
| 查询会话相关信息 | SELECT session\_id as spid, login\_time, host\_name, program\_name  ,login\_name, nt\_user\_name, last\_request\_start\_time, last\_request\_end\_time  FROM **sys.dm\_exec\_sessions** WHERE session\_id in (60, 61)  其中包括会话的登陆时间、主机名、程序名、登录名、WindowsNT用户名，最后的请求开始和结束时间等信息 |
| 查询请求相关信息 | SELECT session\_id AS spid, blocking\_session\_id, command, sql\_handle, database\_id,  wait\_type, wait\_time, wait\_resource  FROM **sys.dm\_exec\_requests** WHERE blocking\_session\_id > 0  其中包括阻塞该会话的某个会话ID、阻塞的毫秒数等，可以通过blocking\_session\_id > 0判断是否为阻塞会话 |
| 处理阻塞 | 可以通过kill <spid>方式关闭会话，此外还可以设置会话中锁的时间，包括0立即超时，-1无超时（默认值），和n>0超时毫秒数 |

* 隔离级别

数据库的隔离级别决定了并发用户读取和写入的行为，一般来说隔离级别越高，数据的一致性越好，并发性越弱，接下来首先锁机制下的隔离级别。

**READ UNCOMMITED**: 最低的隔离级别，读取时不需要请求共享锁，会出现**脏读**，在对数据一致性要求不高的情况下使用，在实际中通过WITH NOLOCK方式使用。

**READ COMMITED**: 系统默认的隔离级别，支持读取已提交的数据，通过要求读取者获取共享锁来防止未提交的读取，但由于其会在读取完成后释放锁，因而会存在在两次读取之间数据不一致的问题（也称之为**不可重复读**）。

**REPEATABLE READ**: 可重复读通过在事务中始终持有读共享锁的方式防止两次不同的读取。同时由于在该隔离级别下，共享锁会一直持有，因而无法获取排它锁，也防止了**丢失更新**的情况，比如在低级别的隔离级别下，两个事务中均修改某个值，那么后面一个修改会奏效。

**SERIALIZABLE**: 最高的隔离级别，其除了在读请求时一直持有读共享锁，同时还会限定查询筛选所限的key键的范围（之间提及的锁范围），用于阻止其他事务尝试添加新行(被限定情况下)，防止了出现**幻读**的情况。

接下来，介绍行版本模式（该模式通过tempdb存储已提交行的之前版本，之后的深入剖析文章中还会重点介绍tempdb）下的隔离级别，，包括SNAPSHOT和READ COMMITTED SNAPSHOT，分别对应锁模式下的SERIALIZABLE和READ COMMITTED，区别是行版本模式下不会发出读共享锁,所以请求的数据以排他方式锁定时不会等待，读取的性能会获得改善，在修改数据的操作DELETE和UPDATE中需要复制行的版本，因而会相对降低写的性能。

**SNAPSHOT**: 读取数据时会确保获得**事务启动时**最近提交的可用行版本，这儿需要强调事务启动时的概念，比如两个事务A、B先后开启，B事务中修改数据并提交，这个数据修改是不会反应到事务A的，因为事务A获取额是在其开启前的行版本。值得一提的是，该级别可以防止更新冲突且不会造成死锁，比如同时在事务A和B中修改数据，系统会抛出异常，快照隔离事务由于更新冲突而终止。可以通过语句SET TRANSACTION ISOLATION LEVEL SNAPSHOT设置事务的隔离级别为SNAPSHOT。

**READ COMMITTED SNAPSHOT**: 它与SNAPSHOT的区别是，获取的**“语句”启动时**可用的最后提交的行版本，也就是在查询发起时最后提交的可用行版本，最后通过一个表格综述之前介绍的6种不同的隔离级别。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 隔离级别 | 脏读 | 不可重复读 | 丢失更新 | 幻读 | 检测更新冲突 | 使用行版本控制 |
| READ UNCOMMITTED | Yes | Yes | Yes | Yes | No | No |
| READ COMMITTED | No | Yes | Yes | Yes | No | No |
| READ COMMITTED SNAPSHOT | No | Yes | Yes | Yes | No | Yes |
| REPEATABLE READ | No | No | No | Yes | No | No |
| SERIALIZABLE | No | No | No | No | No | No |
| SNAPSHOT | No | No | No | No | Yes | Yes |

这部分的最后补充一下数据库中死锁的概念，其和操作系统中学到的死锁改变一样，也是两个或多个进程相互阻塞的情况。在SQL SERVER中一旦出现死锁，系统会通过DEADLOCK\_PRIORITY的死锁优先级来决定先终止哪一个进程，由于终止进程涉及事务的回滚等操作，会消耗一定的性能，通过更好的设计来避免死锁是更好的选择。

可编程对象比较多，包括变量、批、流元素、游标和临时表、用户定义函数、存储过程、触发器、动态SQL等概念，部分内容使用的场景较少，通过表格简述之，但对将对临时表这一常见并较难理解的概念进行细致介绍。

|  |  |
| --- | --- |
| 对象 | 解释与示例 |
| 变量 | DECLARE @i AS INT; SET @i = 10; |
| 批 | 表示一个单元分析和执行的命令组，变量存在于批的生命周期中，并且一个批中只能包含一个DDL语句。 USE TSQL2012; GO |
| 语句块和流元素 | 相对于PL/SQL，T-SQL中语法相对简单，结构完整性要求没有那么高  语句块: BEGIN END  逻辑流：IF BEGIN XXX END ELSE IF BEGIN XXX END ELSE BEGIN XXX END  循环流: WHILE @i < 10 BEGIN XXX END，支持BREAK和CONTINUE |
| 游标 | 游标使用的步骤：1.基于查询声明游标;2.打开游标;3.从游标记录中提取属性值给变量;4.遍历游标记录并迭代;5.关闭游标;6.释放游标  DECLARE C CURSOR FAST\_FORWARD FOR  SELECT custid, ordermonth, qty, FROM Sales.CustOrders ORDER BY custid, ordermonth  OPEN C  FETCH NEXT FROM C INTO @custid, @ordermonth, @qty  SELECT @precustid = @custid, @runqty = 0;  WHILE @@FETCH\_STATUS = 0  BEGIN IF @custid <> @precustid SELECT @precustid = @custid, @runqty = 0  SET @runqty = @runqty + @qty  INSERT INTO @Result VALUES(@custid, @ordermonth, @qty, @runqty)  FETCH NEXT FROM C INTO @custid, @ordermonth, @qty  END  CLOSE C |
| 用户函数、存储过程 | 前者之前介绍内嵌表值函数时以有例子，这儿只介绍Store Procedure，场景为获取某客户指定日期内订单并返回记录数  CREATE PROCEDURE Sales.GetCustomerOrders  @custid AS INT, @fromdate AS DATETIME ='19010101', @todate AS DATETIME ='99991231', @numrows AS INT OUTPUT  AS  SET NOCOUNT NO  SELECT orderid, custid, empid, orderdate FROM Sales.Orders  WHERE custid = @custid AND orderdate > @fromdate AND orderdate < @todate  SET @numrows = @@rowcount  GO |
| 触发器 | CREATE TRIGGER trg\_T1 ON dbo.T1 AFTER INSERT  AS  INSERT INTO dbo.T1\_Audit(keycol, datacol) SELECT keycol, datacol FROM inserted  GO |
| 动态SQL | DECLARE @sql AS VARCHAR(100)  SET @sql = 'PRINT ''XIONGER'''  EXEC(@sql),此外为了防止SQL注入，还可以使用sp\_executesql来达到参数化存储过程数据参数的目的。 |
| 错误处理 | BEGIN TRY END TRY  BEGIN CATCH IF XXX ELSE THROW END CATCH |

* 临时表

T-SQL支持3中类型的临时表，分别是本地临时表、全局临时表和表变量。本地临时表仅对创建它的会话可见，全局临时表对所有会话可见，表变量仅对当前会话的当前批有效，粒度更小，在T-SQL它也是实际的表（易误解为只存在内存）。临时表对于大量数据时性能更好，而表变量是处理少量数据最好选择，构建方式如下所示。

|  |  |
| --- | --- |
| 对象 | 解释与示例 |
| 本地临时表 | IF OBJECT\_ID('tempdbo.dbo.#MYTemp') IS NOT NULL  CREATE TABLE #MYTemp(orderyear INT NOT NULL PRIMARY KEY) |
| 全局临时表 | 将本地临时表中的#换成##即可 |
| 表变量 | DECLARE @MyOrder TABLE(orderyear INT NOT NULL PRIMARY KEY) |

补充部分常见SQL操作

|  |  |
| --- | --- |
| 元数据查询类型 | 解释与示例 |
| 创建架构 | SET SCHEMA HR AUTHPRIZATION dbo |

最后附上英文原版参考书目（Microsoft SQL Server 2012 T-SQL Fundamentals）下载地址：http://pan.baidu.com/s/1eRbhnbk

非常感谢大家的阅读，有T-SQL方面的任何疑问请随时和在下联系。

参考资料：

1. (美)本咁. SQL Server 2012 T-SQL基础教程[M]. 北京:人民邮电出版社, 2013.