 众所周知，SQL Server执行SQL语句的性能判定标准主要是IO读取数大小。本文在不违反这一原则情况下，同时来分析一下部分SQL语句执行时，SQL Server内存的变化情况。 首先简述一下SQL Server内存占用的特点。SQL Server所占用的内存除程序(即SQL Server引擎)外，主要包括缓存的数据(Buffer)和执行计划(Cache)。SQL Server以8KB大小的页为单位存储数据。这个和SQL Server数据在磁盘上的存储页大小相同。当SQL Server执行SQL 语句时，如果需要的数据已经在其内存中，则直接从内存缓冲区读取并进行必要的运算然后输出执行结果。如果数据还未在内存中，则首先将数据从磁盘上读入内存Buffer中。而我们通常评价SQL性能指标中的IO逻辑读取数对应的正是从内存缓冲区读取的页数，而IO物理读取数则对应数据从磁盘读取的页数。 注：以下的试验在多人共享的开发测试服务器上也可以进行，因为实际上可以分别看到某个表所占用的内存情况。但为了方便，笔者在做此试验时，在一个单独的、确认没有其它并发任务的数据库上进行，因此所看到的内存变化正是每一次所执行的SQL语句引起的。 我们首先来看一个简单的实例。创建下表：以下是引用片段：Create Table P_User (UserMobileStatus int NOT NULL, MobileNo int NOT NULL, LastOpTime DateTime Not NULL) 然后为该表插入一定的数据：以下是引用片段：Declare @i int Set @i=28000 WHILE @i BEGIN Insert Into P_User Select @i % 2,@i,GetUTCDate() Set @i=@i 1 END 然后我

们在查询分析器中首先执行:以下是引用片段：Set Statistics IO ON 并按下Ctrl M以显示实际的执行计划。 此时，可以开始进行我们的试验了。为了准确观察每一次SQL语句变化情况，在执行第一条SQL语句以前，我们首先清空SQL Server所占用的数据内存：以下是引用片段：CHECKPOINT GO DBCC DROPCLEANBUFFERS 这将清空SQL Server所占用的数据缓冲区(此语句在生产服务器上慎用，因为将导致一段时间内后续的SQL语句执行变慢)。 测试1：在没有索引的表上执行SQL语句 1.1 执行全表选取或者低选择性选取 Select * From P_User 从SQL执行计划可以看到，由于此时表中没有任何索引，因此将产生Table Scan。而IO统计结果如下： (1000 row(s) affected) 表P_User。扫描计数1，逻辑读取4 次，物理读取4 次，预读0 次，lob 逻辑读取0 次，lob 物理读取0 次，lob 预读0 次。 我们看一下数据库内存中的情况。 首先查询到我们所操作database的database_id：以下是引用片段：Select database_id From sys.databases Where name=TestGDB 然后使用该database_id从表中查看内存情况：以下是引用片段：SELECT * FROM sys.dm_os_buffer_descriptors bd WHERE database_id=5 order by allocation_unit_id,page_id 得到结果如下： 得到的结果中可以看到，除了必要的管理页(一个PFS_Page和一个IAM_Page)外，内存中总共出现了4个Data_Page页。这和刚才IO统计中看到的结果：逻辑读为4，物理读为4相同。由于是全表读取，表明P_User表全部数据所占用的数据页数也正是4，将这4个数据页的row_count数加起来也可以验证其总数据行=1000。 在上例中，如果不清空数据缓冲区，再执行一遍SQL，可以看到内存毫无变化，而逻辑读也不变，只

是物理读变为0，因为已经不需要再从磁盘读入数据。 1.2 执行高选择性选取 另外，在没有索引的情况下，如果将上例修改为：以下是引用片段：Select Top 1 * From P_Order 或者Select * From P_Order Where MobileNo=28502 可以看到，系统同样要读取全部的数据页到内存。 如果使用Select Top 1 * From P_Order Where MobileNo=28502这样的选取方式，有可能会出现只读取部分数据页到内存的情况。但由于在没有索引情况下，数据实际上是无序存放在堆上，所以结果很不稳定，也有可能发生读取所有的数据页到内存。 测试2：建立聚集索引情况下，执行SQL语句 2.1 执行全表选取或者低选择性选取 修改表结构，在MobileNo字段上建立聚集索引。然后再次执行刚才的SQL语句。得到的执行计划变为聚集索引扫描。IO统计消息为： (1000 row(s) affected) 表P_User。扫描计数1，逻辑读取6 次，物理读取1 次，预读4 次，lob 逻辑读取0 次，lob 物理读取0 次，lob 预读0 次。 这里的逻辑读取变为6次。 内存情况如下： 内存中的变化是增加了一个非叶级的聚集索引页，而叶级的聚集索引则会和数据放在一起。 另外，可以查看该表索引的级别：以下是引用片段： SELECT database_id,object_id,index_id,index_level,page_count,record_count FROM sys.dm_db_index_physical_stats (DB_ID(NTestGDB), OBJECT_ID(Ndbo.P_User), NULL, NULL , DETAILED). 从结果可以看到该表的聚集索引总共分2级。 因而逻辑读增加了2(由于发生Clustered Index Scan，除了根级别的聚集索引页占用1次外，从根级别聚集索引定位到叶级别的聚集索引也将额外占用1次逻辑读)。 另外一个变化是只发生了一次物理读，即读取根级别的聚集索引页，另外4个数据页则通过预读方式而

不是物理读从磁盘装入内存Buffer。这使得有聚集索引的情况下，执行SQL所直接花费的代价实际上更小。 2.2 执行高选择性选取 在建立聚集索引情况下，对性能有益的变化是： 对于Select Top 1 * From P_Order 或者Select * From P_Order Where MobileNo=28702这样的语句，在有聚集索引情况下，只会将最终记录所在的页读入内存。