**养成写高效的SQL语句的习惯**

我们做软件开发的，大部分人都离不开跟数据库打交道，特别是erp开发的，跟数据库打交道更是频繁，存储过程动不动就是上千行，如果数据量大，人员流动大，那么我们还能保证下一段时间系统还能流畅的运行吗？我们还能保证下一个人能看懂我们的存储过程吗？

要知道sql语句，我想我们有必要知道sqlserver查询分析器怎么执行我么sql语句的，我么很多人会看执行计划，或者用 profile来监视和调优查询语句或者存储过程慢的原因，但是如果我们知道查询分析器的执行逻辑顺序，下手的时候就胸有成竹，那么下手是不是有把握点 呢？

**查询的逻辑执行顺序**

1. FROM < left\_table>
2. ON < join\_condition>
3. < join\_type> JOIN < right\_table>
4. WHERE < where\_condition>
5. GROUP BY < group\_by\_list>
6. WITH {cube | rollup}
7. HAVING < having\_condition>
8. SELECT
9. DISTINCT
10. ORDER BY < order\_by\_list>
11. < top\_specification> < select\_list>

标准的SQL 的解析顺序为:

1. .FROM 子句 组装来自不同数据源的数据
2. .WHERE 子句 基于指定的条件对记录进行筛选
3. .GROUP BY 子句 将数据划分为多个分组
4. .使用聚合函数进行计算
5. .使用HAVING子句筛选分组
6. .计算所有的表达式
7. .使用ORDER BY对结果集进行排序

**执行顺序**

1. FROM：对FROM子句中前两个表执行笛卡尔积生成虚拟表vt1
2. ON:对vt1表应用ON筛选器只有满足< join\_condition> 为真的行才被插入vt2
3. OUTER(join)：如果指定了 OUTER JOIN保留表(preserved table)中未找到的行将行作为外部行添加到vt2 生成t3如果from包含两个以上表则对上一个联结生成的结果表和下一个表重复执行步骤和步骤直接结束
4. WHERE：对vt3应用 WHERE 筛选器只有使< where\_condition> 为true的行才被插入vt4
5. GROUP BY：按GROUP BY子句中的列列表对vt4中的行分组生成vt5
6. CUBE|ROLLUP：把超组(supergroups)插入vt6 生成vt6
7. HAVING：对vt6应用HAVING筛选器只有使< having\_condition> 为true的组才插入vt7
8. SELECT：处理select列表产生vt8
9. DISTINCT：将重复的行从vt8中去除产生vt9
10. ORDER BY：将vt9的行按order by子句中的列列表排序生成一个游标vc10
11. TOP：从vc10的开始处选择指定数量或比例的行生成vt11 并返回调用者

看到这里，那么用过linqtosql的语法有点相似啊？如果我们我们了解了sqlserver执行顺序，那么我们就接下来进一步养成日常 sql好习惯，也就是在实现功能同时有考虑性能的思想，数据库是能进行集合运算的工具，我们应该尽量的利用这个工具，所谓集合运算实际就是批量运算，就是 尽量减少在客户端进行大数据量的循环操作，而用SQL语句或者存储过程代替。

**只返回需要的数据**

返回数据到客户端至少需要数据库提取数据、网络传输数据、客户端接收数据以及客户端处理数据等环节，如果返回不需要的数据，就会增加服务器、网络和客户端的无效劳动，其害处是显而易见的，避免这类事件需要注意：

1. 横向来看：

1. 不要写SELECT \*的语句，而是选择你需要的字段。
2. 当在SQL语句中连接多个表时, 请使用表的别名并把别名前缀于每个Column上.这样一来,就可以减少解析的时间并减少那些由Column歧义引起的语法错误。

如有表table1（ID,col1）和table2 （ID,col2）

Select A.ID, A.col1, B.col2

-- Select A.ID, col1, col2 –不要这么写，不利于将来程序扩展

from table1 A inner join table2 B on A.ID=B.ID Where …

2. 纵向来看：

1. 合理写WHERE子句，不要写没有WHERE的SQL语句。
2. SELECT TOP N \* --没有WHERE条件的用此替代

**尽量少做重复的工作**

1. 控制同一语句的多次执行，特别是一些基础数据的多次执行是很多程序员很少注意的。
2. 减少多次的数据转换，也许需要数据转换是设计的问题，但是减少次数是程序员可以做到的。
3. 杜绝不必要的子查询和连接表，子查询在执行计划一般解释成外连接，多余的连接表带来额外的开销。
4. 合并对同一表同一条件的多次UPDATE，比如：

UPDATE EMPLOYEE SET FNAME='HAIWER'

WHERE EMP\_ID=' VPA30890F' UPDATE EMPLOYEE SET LNAME='YANG'

WHERE EMP\_ID=' VPA30890F'

这两个语句应该合并成以下一个语句

UPDATE EMPLOYEE SET FNAME='HAIWER',LNAME='YANG' WHERE EMP\_ID=' VPA30890F'

1. UPDATE操作不要拆成DELETE操作+INSERT操作的形式，虽然功能相同，但是性能差别是很大的。

**注意临时表和表变量的用法**

在复杂系统中，临时表和表变量很难避免，关于临时表和表变量的用法，需要注意：

1. 如果语句很复杂，连接太多，可以考虑用临时表和表变量分步完成。
2. 如果需要多次用到一个大表的同一部分数据，考虑用临时表和表变量暂存这部分数据。
3. 如果需要综合多个表的数据，形成一个结果，可以考虑用临时表和表变量分步汇总这多个表的数据。
4. 其他情况下，应该控制临时表和表变量的使用。
5. 关于临时表和表变量的选择，很多说法是表变量在内存，速度快，应该首选表变量，但是在实际使用中发现，主要考虑需要放在临时表的数据量，在数据量较多的情况下，临时表的速度反而更快。执行时间段与预计执行时间(多长)。
6. 关于临时表产生使用SELECT INTO和CREATE TABLE + INSERT INTO的选择，一般情况下，SELECT INTO会比CREATE TABLE + INSERT INTO的方法快很多，但是SELECT INTO会锁定TEMPDB的系统表SYSOBJECTS、SYSINDEXES、SYSCOLUMNS，在多用户并发环境下，容易阻塞其他进程，所以我 的建议是，在并发系统中，尽量使用CREATE TABLE + INSERT INTO，而大数据量的单个语句使用中，使用SELECT INTO。

**子查询的用法**

子查询是一个 SELECT 查询，它嵌套在 SELECT、INSERT、UPDATE、DELETE 语句或其它子查询中。任何允许使用表达式的地方都可以使用子查询，子查询可以使我们的编程灵活多样，可以用来实现一些特殊的功能。但是在性能上，往往一个 不合适的子查询用法会形成一个性能瓶颈。如果子查询的条件中使用了其外层的表的字段，这种子查询就叫作相关子查询。相关子查询可以用IN、NOT IN、EXISTS、NOT EXISTS引入。 关于相关子查询，应该注意：

1. NOT IN、NOT EXISTS的相关子查询可以改用LEFT JOIN代替写法。

比如：

SELECT PUB\_NAME FROM PUBLISHERS WHERE PUB\_ID NOT IN (SELECT PUB\_ID FROM TITLES WHERE TYPE = 'BUSINESS')

可以改写成：

SELECT A.PUB\_NAME FROM PUBLISHERS A LEFT JOIN TITLES B ON B.TYPE = 'BUSINESS' AND A.PUB\_ID=B. PUB\_ID WHERE B.PUB\_ID IS NULL

又比如：

SELECT TITLE FROM TITLES

WHERE NOT EXISTS

(SELECT TITLE\_ID FROM SALES

WHERE TITLE\_ID = TITLES.TITLE\_ID)

可以改写成：

SELECT TITLE

FROM TITLES LEFT JOIN SALES

ON SALES.TITLE\_ID = TITLES.TITLE\_ID

WHERE SALES.TITLE\_ID IS NULL

2. 如果保证子查询没有重复 ，IN、EXISTS的相关子查询可以用INNER JOIN 代替。比如：

SELECT PUB\_NAME

FROM PUBLISHERS

WHERE PUB\_ID IN

(SELECT PUB\_ID

FROM TITLES

WHERE TYPE = 'BUSINESS')

可以改写成：

SELECT A.PUB\_NAME --SELECT DISTINCT A.PUB\_NAME

FROM PUBLISHERS A INNER JOIN TITLES B

ON B.TYPE = 'BUSINESS' AND

A.PUB\_ID=B. PUB\_ID

3. IN的相关子查询用EXISTS代替，比如

SELECT PUB\_NAME FROM PUBLISHERS

WHERE PUB\_ID IN

(SELECT PUB\_ID FROM TITLES WHERE TYPE = 'BUSINESS')

可以用下面语句代替：

SELECT PUB\_NAME FROM PUBLISHERS WHERE EXISTS

(SELECT 1 FROM TITLES WHERE TYPE = 'BUSINESS' AND

PUB\_ID= PUBLISHERS.PUB\_ID)

4. 不要用COUNT(\*)的子查询判断是否存在记录，最好用LEFT JOIN或者EXISTS，比如有人写这样的语句：

SELECT JOB\_DESC FROM JOBS

WHERE (SELECT COUNT(\*) FROM EMPLOYEE WHERE JOB\_ID=JOBS.JOB\_ID)=0

应该写成：

SELECT JOBS.JOB\_DESC FROM JOBS LEFT JOIN EMPLOYEE

ON EMPLOYEE.JOB\_ID=JOBS.JOB\_ID

WHERE EMPLOYEE.EMP\_ID IS NULL

还有

SELECT JOB\_DESC FROM JOBS

WHERE (SELECT COUNT(\*) FROM EMPLOYEE WHERE JOB\_ID=JOBS.JOB\_ID)<>0

应该写成：

SELECT JOB\_DESC FROM JOBS

WHERE EXISTS (SELECT 1 FROM EMPLOYEE WHERE JOB\_ID=JOBS.JOB\_ID)

**尽量使用索引**

建立索引后，并不是每个查询都会使用索引，在使用索引的情况下，索引的使用效率也会有很大的差别。只要我们在查询语句中没有强制指定索引，索 引的选择和使用方法是SQLSERVER的优化器自动作的选择，而它选择的根据是查询语句的条件以及相关表的统计信息，这就要求我们在写SQL。

语句的时候尽量使得优化器可以使用索引。为了使得优化器能高效使用索引，写语句的时候应该注意：

A、不要对索引字段进行运算，而要想办法做变换，比如

SELECT ID FROM T WHERE NUM/2=100

应改为:

SELECT ID FROM T WHERE NUM=100\*2

SELECT ID FROM T WHERE NUM/2=NUM1

如果NUM有索引应改为:

SELECT ID FROM T WHERE NUM=NUM1\*2

如果NUM1有索引则不应该改。

发现过这样的语句：

SELECT 年,月,金额 FROM 结余表 WHERE 100\*年+月=2010\*100+10

应该改为：

SELECT 年,月,金额 FROM 结余表 WHERE 年=2010 AND月=10

B、 不要对索引字段进行格式转换

日期字段的例子：

WHERE CONVERT(VARCHAR(10), 日期字段,120)='2010-07-15'

应该改为

WHERE日期字段〉='2010-07-15' AND 日期字段<'2010-07-16'

ISNULL转换的例子：

WHERE ISNULL(字段,'')<>''应改为:WHERE字段<>''

WHERE ISNULL(字段,'')=''不应修改

WHERE ISNULL(字段,'F') ='T'应改为: WHERE字段='T'

WHERE ISNULL(字段,'F')<>'T'不应修改

C、 不要对索引字段使用函数

WHERE LEFT(NAME, 3)='ABC' 或者WHERE SUBSTRING(NAME,1, 3)='ABC'

应改为: WHERE NAME LIKE 'ABC%'

日期查询的例子：

WHERE DATEDIFF(DAY, 日期,'2010-06-30')=0

应改为:WHERE 日期>='2010-06-30' AND 日期 <'2010-07-01'

WHERE DATEDIFF(DAY, 日期,'2010-06-30')>0

应改为:WHERE 日期 <'2010-06-30'

WHERE DATEDIFF(DAY, 日期,'2010-06-30')>=0

应改为:WHERE 日期 <'2010-07-01'

WHERE DATEDIFF(DAY, 日期,'2010-06-30')<0

应改为:WHERE 日期>='2010-07-01'

WHERE DATEDIFF(DAY, 日期,'2010-06-30')<=0

应改为:WHERE 日期>='2010-06-30'

4. 不要对索引字段进行多字段连接

比如：

WHERE FAME+ '. '+LNAME='HAIWEI.YANG'

应改为:

WHERE FNAME='HAIWEI' AND LNAME='YANG'

**多表连接的连接条件**

多表连接的连接条件对索引的选择有着重要的意义，所以我们在写连接条件条件的时候需要特别注意。

1. 多表连接的时候，连接条件必须写全，宁可重复，不要缺漏。
2. 连接条件尽量使用聚集索引
3. 注意ON、WHERE和HAVING部分条件的区别

ON是最先执行， WHERE次之，HAVING最后，因为ON是先把不符合条件的记录过滤后才进行统计，它就可以减少中间运算要处理的数据，按理说应该速度是最快 的，WHERE也应该比 HAVING快点的，因为它过滤数据后才进行SUM，在两个表联接时才用ON的，所以在一个表的时候，就剩下WHERE跟HAVING比较了

考虑联接优先顺序：

1. INNER JOIN
2. LEFT JOIN (注：RIGHT JOIN 用 LEFT JOIN 替代)
3. CROSS JOIN

其它注意和了解的地方有：

1. 在IN后面值的列表中，将出现最频繁的值放在最前面，出现得最少的放在最后面，减少判断的次数
2. 注意UNION和UNION ALL的区别。--允许重复数据用UNION ALL好
3. 注意使用DISTINCT，在没有必要时不要用
4. TRUNCATE TABLE 与 DELETE 区别
5. 减少访问数据库的次数

还有就是我们写存储过程，如果比较长的话，最后用标记符标开，因为这样可读性很好，即使语句写的不怎么样但是语句工整，C# 有region，sql我比较喜欢用的就是：

--startof 查询在职人数

sql语句

--end of

正式机器上我们一般不能随便调试程序，但是很多时候程序在我们本机上没问题，但是进正式系统就有问题，但是我们又不能随便在正式机器上操作，那么怎么办呢？我们可以用回滚来调试我们的存储过程或者是sql语句，从而排错。

BEGIN TRAN

UPDATE a SET 字段=''

ROLLBACK

作业存储过程我一般会加上下面这段，这样检查错误可以放在存储过程，如果执行错误回滚操作，但是如果程序里面已经有了事务回滚，那么存储过程就不 要写事务了，这样会导致事务回滚嵌套降低执行效率，但是我们很多时候可以把检查放在存储过程里，这样有利于我们解读这个存储过程，和排错。

BEGIN TRANSACTION

--事务回滚开始

--检查报错

IF ( @@ERROR > 0 )

BEGIN

--回滚操作

ROLLBACK TRANSACTION

RAISERROR('删除工作报告错误', 16, 3)

RETURN

END

--结束事务

COMMIT TRANSACTION

大概就写这么多了，有错误的地方欢迎大家拍砖,希望交流和共享。