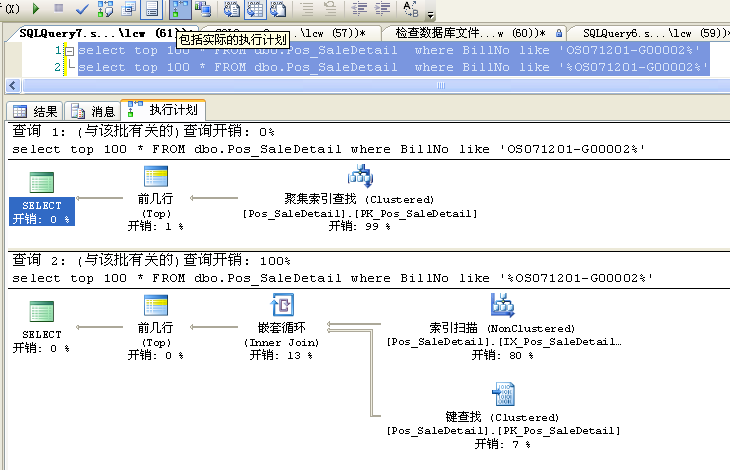
ERP系统优化分享

成功的软件系统背后一定有一套严谨的数据库。如果在数据库设计阶段没有考虑到性能因素，当客户运行一段时间，数据量增大，性能产生瓶颈时，而又必须要修改表结构才能解决性能问题，这个时候再去重新构建表结构，带来的消极影响将无法估量。本次讲解数据库调优中涉及到程式开发阶段需要关注的一些知识。

1. **优化工具以及参数设置。**为了更好的跟踪观察优化过程，需要了解以下的一些参数和设置。
2. 参数设置
3. SET STATISTICS TIME ON --显示分析、编译和执行各语句所需的毫秒数。
4. SET STATISTICS IO ON --使SQL Server 显示有关由Transact-SQL 语句生成的磁盘活动量的信息。
5. DBCC FREEPROCCACHE; --从过程缓存中删除所有元素。释放过程缓存将导致系统重新编译某些语句（例如，即席SQL 语句），而不重用缓存中的语句。
6. DBCC DROPCLEANBUFFERS --从缓冲池中删除所有清除缓冲区。若要从缓冲池中删除清除缓冲区，请首先使用CHECKPOINT 生成一个冷缓存。这可以强制将当前数据库的全部脏页写入磁盘，然后清除缓冲区。完成此操作后，便可发出DBCC DROPCLEANBUFFERS 命令来从缓冲池中删除所有缓冲区。
7. 执行计划，如图：



1. Performance Dashboard工具 ，该工具利用DMV 和DMF来获取以下数据：

1）CPU瓶颈(和什么查询最损耗CPU)

2）IO瓶颈(还有，什么查询执行最多的IO)

1. 查询优化器产生的索引建议(缺失索引)；
2. SQL Server profiler.数据库跟踪服务，适合跟踪执行语句，并回放。
3. 数据库引擎优化顾问。

**二、索引的知识。**

索引是与表或视图关联的磁盘上结构，可以加快从表或视图中检索行的速度。索引包含由表或视图中的一列或多列生成的键。这些键存储在一个结构（B 树）中，使 SQL Server 可以快速有效地查找与键值关联的行。

表或视图可以包含以下类型的索引：

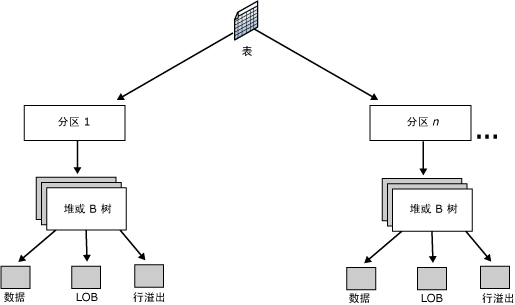
聚集

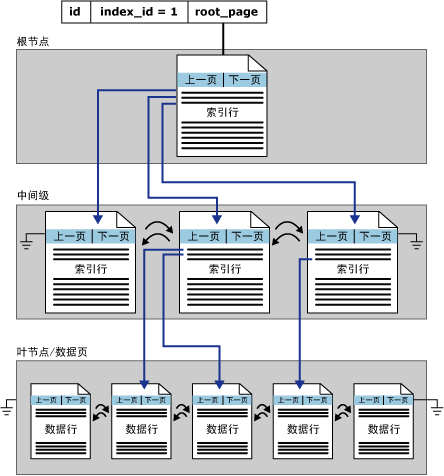
* + 聚集索引根据数据行的键值在表或视图中排序和存储这些数据行。索引定义中包含聚集索引列。每个表只能有一个聚集索引，因为数据行本身只能按一个顺序排序。
  + 只有当表包含聚集索引时，表中的数据行才按排序顺序存储。如果表具有聚集索引，则该表称为聚集表。如果表没有聚集索引，则其数据行存储在一个称为堆的无序结构中。
* 非聚集
  + 非聚集索引具有独立于数据行的结构。非聚集索引包含非聚集索引键值，并且每个键值项都有指向包含该键值的数据行的指针。
  + 从非聚集索引中的索引行指向数据行的指针称为行定位器。行定位器的结构取决于数据页是存储在堆中还是聚集表中。对于堆，行定位器是指向行的指针。对于聚集表，行定位器是聚集索引键。
  + 在 SQL Server 2005 中，可以向非聚集索引的叶级别添加非键列以跳过现有的索引键限制（900 字节和 16 键列），并执行完整范围内的索引查询。有关详细信息，请参阅具有包含性列的索引。

表和索引体系结构。

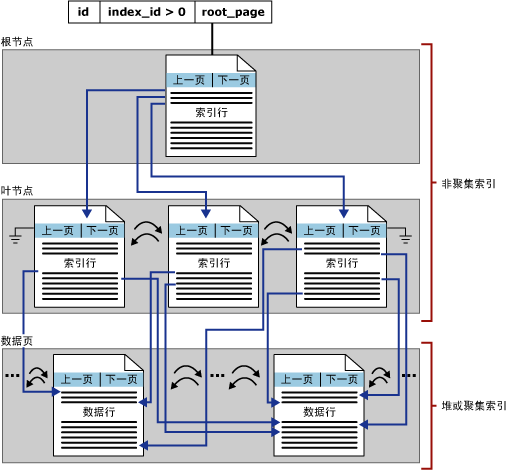
表和索引作为 8 KB 页的集合存储。本主题介绍表页和索引页的组织方式。

**ms-help://MS.SQLCC.v9/MS.SQLSVR.v9.zh-CHS/udb9/local/collapse.gif表组织**

下图显示了表的组织。表包含在一个或多个分区中，每个分区在一个堆或一个聚集索引结构包含数据行。堆页或聚集索引页在一个或多个分配单元中进行管理，具体的分配单元数取决于数据行中的列类型。

对于某个聚集索引，sys.system\_internals\_allocation\_units 中的 root\_page 列指向该聚集索引某个特定分区的顶部。SQL Server 将在索引中向下移动以查找与某个聚集索引键对应的行。为了查找键的范围，SQL Server 将在索引中移动以查找该范围的起始键值，然后用向前或向后指针在数据页中进行扫描。为了查找数据页链的首页，SQL Server 将从索引的根节点沿最左边的指针进行扫描。下图显式了聚集索引单个分区中的结构。

下图说明了单个分区中的非聚集索引结构。



聚集索引和非聚集索引都可以是唯一的。这意味着任何两行都不能有相同的索引键值。另外，索引也可以不是唯一的，即多行可以共享同一键值。有关详细信息，请参阅唯一索引设计指南。

**主键与索引的关系。**

区别： 1、主键只能有一个，索引可以有很多个。2、功能不一样，主键考虑的数据的完整和正确性、唯一、不可为null,不太在乎字段大小、顺序。而索引考虑主要是查询的效能，不太考虑数据是否唯一，以及索引字段大小、顺序，字段越小越好。

联系：有主键一定就会产生索引。如果表中已存在聚集索引，主键就会自动默认产生相同列的一个唯一非聚集索引。如果没有聚集索引，就会产生一个唯一聚集索引。

每当修改了表数据后，都会自动维护表或视图的索引。

1、 了解数据库本身的特征。例如，它是频繁修改数据的联机事务处理 (OLTP) 数据库，还是主要包含只读数据的决策支持系统 (DSS) 或数据仓库 (OLAP) 数据库？

2、 了解最常用的查询的特征。例如，了解到最常用的查询联接两个或多个表将有助于决定要使用的最佳索引类型。有关详细信息，请参阅常规索引设计指南。

3、 了解查询中使用的列的特征。例如，某个索引对于含有整数数据类型同时还是唯一的或非空的列是理想索引。

4、 确定哪些索引选项可在创建或维护索引时提高性能。例如，对现有某个大型表创建聚集索引将会受益于 ONLINE 索引选项。ONLINE 选项允许在创建索引或重新生成索引时继续对基础数据执行并发活动。有关详细信息，请参阅设置索引选项。

5、 确定索引的最佳存储位置。非聚集索引可以与基础表存储在同一个文件组中，也可以存储在不同的文件组中。索引的存储位置可通过提高磁盘 I/O 性能来提高查询性能。例如，将非聚集索引存储在表文件组所在磁盘以外的某个磁盘上的一个文件组中可以提高性能，因为可以同时读取多个磁盘。有关详细信息，请参阅在文件组上放置索引。或者，聚集索引和非聚集索引也可以使用跨越多个文件组的分区方案。在维护整个集合的完整性时，使用分区可以快速而有效地访问或管理数据子集，从而使大型表或索引更易于管理。有关详细信息，请参阅已分区表和已分区索引。在考虑分区时，应确定是否应对齐索引，即，是按实质上与表相同的方式进行分区，还是单独分区。有关详细信息，请参阅已分区索引的特殊指导原则。

6、 索引的设计。

设计索引时，应考虑以下数据库准则：

• 一个表如果建有大量索引会影响 INSERT、UPDATE 和 DELETE 语句的性能，因为在表中的数据更改时，所有索引都须进行适当的调整。

• 避免对经常更新的表进行过多的索引，并且索引应保持较窄，就是说，列要尽可能少。

• 使用多个索引可以提高更新少而数据量大的查询的性能。大量索引可以提高不修改数据的查询（例如 SELECT 语句）的性能，因为查询优化器有更多的索引可供选择，从而可以确定最快的访问方法。

• 对小表进行索引可能不会产生优化效果，因为查询优化器在遍历用于搜索数据的索引时，花费的时间可能比执行简单的表扫描还长。因此，小表的索引可能从来不用，但仍必须在表中的数据更改时进行维护。

• 视图包含聚合、表联接或聚合和联接的组合时，视图的索引可以显著地提升性能。若要使查询优化器使用视图，并不一定非要在查询中显式引用该视图。有关详细信息，请参阅设计索引视图。

• 使用数据库引擎优化顾问来分析数据库并生成索引建议。有关详细信息，请参阅数据库引擎优化顾问概述。

**定义聚集索引键时使用的列越少越好。考虑具有下列一个或多个属性的列：**

•唯一或包含许多不重复的值

如：Inv\_MoveMaster (BillNo,CompanyCode)

•按顺序被访问

如：Inv\_MoveMaster(OutCheckDate)

•经常用于对表中检索到的数据进行排序。按该列对表进行聚集（即物理排序）是一个好方法，它可以在每次查询该列时节省排序操作的成本。

**聚集索引不适用于具有下列属性的列：**

•频繁更改的列

这将导致整行移动，因为数据库引擎 必须按物理顺序保留行中的数据值。这一点要特别注意，因为在大容量事务处理系统中数据通常是可变的。

如：Inv\_MoveMaster(ModifyDTM)

•宽键

宽键是若干列或若干大型列的组合。所有非聚集索引将聚集索引中的键值用作查找键。为同一表定义的任何非聚集索引都将增大许多，这是因为非聚集索引项包含聚集键，同时也包含为此非聚集索引定义的键列。

如：Inv\_CarrySize

(CompanyCode,Sequence,StockCode,MaterialCode,SizeCode)

可以对表或索引视图创建多个非聚集索引。通常，设计非聚集索引是为改善经常使用的、没有建立聚集索引的查询的性能。

与使用书中索引的方式相似，查询优化器在搜索数据值时，先搜索非聚集索引以找到数据值在表中的位置，然后直接从该位置检索数据。这使非聚集索引成为完全匹配查询的最佳选择，因为索引包含说明查询所搜索的数据值在表中的精确位置的项。如：Inv\_MoveMaster (BillNo,CompanyCode)。查询优化器能快速找出索引中与指定 BillNo,CompanyCode匹配的所有项。每个索引项都指向表或聚集索引中准确的页和行，其中可以找到相应的数据。在查询优化器在索引中找到所有项之后，它可以直接转到准确的页和行进行数据检索。如果不唯一，将可能指向多个数据页中，从而降低了读取效能。

数据库注意事项

**设计非聚集索引时需要注意数据库的特征。**

• 更新要求较低但包含大量数据的数据库或表可以从许多非聚集索引中获益从而改善查询性能。

决策支持系统应用程序和主要包含只读数据的数据库可以从许多非聚集索引中获益。查询优化器具有更多可供选择的索引用来确定最快的访问方法，并且数据库的低更新特征意味着索引维护不会降低性能。

• 联机事务处理应用程序和包含大量更新表的数据库应避免使用过多的索引。此外，索引应该是窄的，即列越少越好。

一个表如果建有大量索引会影响 INSERT、UPDATE 和 DELETE 语句的性能，因为所有索引都必须随表中数据的更改进行相应的调整。

查询注意事项

在创建非聚集索引之前，应先了解访问数据的方式。

考虑对具有以下属性的查询使用非聚集索引：

• 使用 JOIN 或 GROUP BY 子句。

应为联接和分组操作中所涉及的列创建多个非聚集索引，为任何外键列创建一个聚集索引。

• 不返回大型结果集的查询。

• 包含经常包含在查询的搜索条件（例如返回完全匹配的 WHERE 子句）中的列。

示例对比：见脚本一

列注意事项

考虑具有以下一个或多个属性的列：

• 覆盖查询。

当索引包含查询中的所有列时，性能可以提升。查询优化器可以找到索引内的所有列值；不会访问表或聚集索引数据，这样就减少了磁盘 I/O 操作。使用具有包含列的索引来添加覆盖列，而不是创建宽索引键。有关详细信息，请参阅 具有包含性列的索引。

如果表有聚集索引，则该聚集索引中定义的列将自动追加到表上每个非聚集索引的末端。这可以生成覆盖查询，而不用在非聚集索引定义中指定聚集索引列。例如，如果一个表在 C 列上有聚集索引，则 B 和 A 列的非聚集索引将具有其自己的键值列 B、A 和 C。聚集索引不需要包含性的列，因为聚集索引就是数据行本身。

• 大量非重复值，如姓氏和名字的组合（前提是聚集索引被用于其他列）。 如果只有很少的非重复值，例如仅有 1 和 0，则大多数查询将不使用索引，因为此时表扫描通常更有效。

**三、索引使用规范：**

复合索引

Create index IX\_Test ON User\_Table(firstname, lastname)

Select address from User\_Table Where firstname = ‘张’ 该句能用到索引查找

Select address from User\_Table Where lastname = ‘张’ 该句能用到索引扫描

Like 通配符出现在字符串前面不能用到索引

例: 索引(Lastname)

Where Lastname Like ‘D%’ 该句能用到索引查找 快

Where Lastname Like ‘%D’ 该句能用到索引扫描 慢

尽量少在where子句中的“=”左边字段列进行函数、算术运算或其他表达式运算, 因为这样用不到索引

例： where convert(varchar(10),a.BillDate,120) >= @startDate and convert(varchar(10),a.BillDate,120) <= @endDate --优化前

where a.BillDate between convert(varchar(10), @startDate,120) and convert(varchar(10),@endDate,120) --优化上面语句用不到索引

CREATE VIEW [dbo].[VIEW\_FF\_LoadMaterial\_Size]

AS

SELECT --Upper(dbo.FF\_Material\_Size.MaterialCode) as MaterialCode,/\* Upper function don't use Index \*/

dbo.FF\_Material\_Size.MaterialCode,

dbo.FF\_Material\_Size.SizeCode, dbo.FF\_Material\_Size.BarCode

FROM dbo.FF\_Material\_Size INNER JOIN

dbo.FF\_Material ON dbo.FF\_Material\_Size.MaterialCode = dbo.FF\_Material.MaterialCode

WHERE (dbo.FF\_Material.AuditingState = '1')

--在FF\_Material\_Size表上用触发器来替代

尽量避免反复检索同一张或几张表（使用相同的检索条件），尤其是记录笔数很多的表，这种情况可以先根据条件提取所需要的数据到临时表，然后再做关联

**四、表结构的设计。**

1. 字符串长度如果相同，尽量使用char类型而非varchar类型。特别是在索引字段。

如：CompanyCode,BillNo等字段。

1. 建立适量Check约束，如：ALTER TABLE dbo.Sal\_OrderDetail ADD CONSTRAINT chkDiscount CHECK(Discount between 0.000 and 1.000);
2. 新增单据号获取方式变为存储过程获取，降低并发锁定。示例代码见附件：2。
3. OLTP 与OLAP分开，建立在线事务数据库和历史数据库。
4. 索引的适当建立。

**五、SQL 2005 新语法使用场景：**

CTE:

1. 分页：

；with MyCTE(ID, Name, RowID)

as

(

select EmployeeID as ID, FirstName + ' ' + LastName as Name,

Row\_Number() over (order by EmployeeID) as RowID

from HumanResources.vEmployee

)

select \* from MyCTE where RowID between 1 and 10

1. 递归查询：

USE tempdb

GO

-- 建立演示环境

CREATE TABLE Dept(

id int PRIMARY KEY,

parent\_id int,

name nvarchar(20))

INSERT Dept

SELECT 0, -1, N'<全部>' UNION ALL

SELECT 1, 0, N'财务部' UNION ALL

SELECT 2, 0, N'行政部' UNION ALL

SELECT 3, 0, N'业务部' UNION ALL

SELECT 4, 0, N'客服部' UNION ALL

SELECT 5, 4, N'销售部' UNION ALL

SELECT 6, 4, N'MIS' UNION ALL

SELECT 7, 6, N'UI' UNION ALL

SELECT 8, 6, N'程式开发' UNION ALL

SELECT 9, 8, N'财务开发' UNION ALL

SELECT 10, 8, N'进销存开发'

GO

-- 查询指定部门下面的所有部门, 并汇总各部门的下级部门数

DECLARE @Dept\_name nvarchar(20)

SET @Dept\_name = N'MIS'

;WITH

DEPTS AS( -- 查询指定部门及其下的所有子部门

-- 定位点成员

SELECT \* FROM Dept

WHERE name = @Dept\_name

UNION ALL

-- 递归成员, 通过引用CTE自身与Dept基表JOIN实现递归

SELECT A.\*

FROM Dept A, DEPTS B

WHERE A.parent\_id = B.id

),

DEPTCHILD AS( -- 引用第个CTE,查询其每条记录对应的部门下的所有子部门

SELECT

Dept\_id = P.id, C.id, C.parent\_id

FROM DEPTS P, Dept C

WHERE P.id = C.parent\_id

UNION ALL

SELECT

P.Dept\_id, C.id, C.parent\_id

FROM DEPTCHILD P, Dept C

WHERE P.id = C.parent\_id

),

DEPTCHILDCNT AS( -- 引用第个CTE, 汇总得到各部门下的子部门数

SELECT

Dept\_id, Cnt = COUNT(\*)

FROM DEPTCHILD

GROUP BY Dept\_id

)

-- 查询指定部门下面的所有部门, 并汇总各部门的下级部门数

SELECT -- JOIN第,3个CTE,得到最终的查询结果

D.\*,

ChildDeptCount = ISNULL(DS.Cnt, 0)

FROM DEPTS D

LEFT JOIN DEPTCHILDCNT DS

ON D.id = DS.Dept\_id

;WITH

DEPTS AS( -- 查询指定部门及其下的所有子部门

-- 定位点成员

SELECT \* FROM Dept

WHERE name = @Dept\_name

UNION ALL

-- 递归成员, 通过引用CTE自身与Dept基表JOIN实现递归

SELECT A.\*

FROM Dept A, DEPTS B

WHERE A.parent\_id = B.id

)

SELECT \* FROM DEPTS;

;WITH

DEPTS AS( -- 查询指定部门及其所有上级部门

-- 定位点成员

SELECT \* FROM Dept

WHERE name = @Dept\_name

UNION ALL

-- 递归成员, 通过引用CTE自身与Dept基表JOIN实现递归

SELECT A.\*

FROM Dept A, DEPTS B

WHERE b.parent\_id = a.id

)

SELECT \* FROM DEPTS;

go

-- 删除演示环境

DROP TABLE Dept

3. 用CTE生成行号，速度极快

;WITH t AS

(

SELECT 1 AS num

UNION ALL

SELECT num+1

FROM t

WHERE num<100000

)

SELECT \* FROM t

OPTION(MAXRECURSION 0)

更多见：<http://topic.csdn.net/u/20100330/23/b2f663b1-0edf-4847-857e-e75640c90c1a.html>

4. 字符串聚合函数

--> --> 生成测试数据

if not object\_id('Tab') is null

drop table Tab

Go

Create table Tab([Col1] int,[Col2] nvarchar(1))

Insert Tab

select 1,N'a' union all

select 1,N'b' union all

select 1,N'c' union all

select 2,N'd' union all

select 2,N'e' union all

select 3,N'f'

Go

--方法1:

--SQL2000用函数:

go

if object\_id('F\_Str') is not null

drop function F\_Str

go

create function F\_Str(@Col1 int)

returns nvarchar(100)

as

begin

declare @S nvarchar(100)

select @S=isnull(@S+',','')+Col2 from Tab where Col1=@Col1

return @S

end

go

Select distinct Col1,Col2=dbo.F\_Str(Col1) from Tab

Go

--方法2:

--SQL2005用XML:

select a.Col1,Col2=stuff(b.Col2.value('/R[1]','nvarchar(max)'),1,1,'')

from (select distinct COl1 from Tab) a

Cross apply

(select COl2=(select N','+Col2 from Tab where Col1=a.COl1 For XML PATH(''), ROOT('R'), TYPE))b

--方法3:

--SQL2005用CTE:

;with roy as(select Col1,Col2,row=row\_number()over(partition by COl1 order by COl1) from Tab)

,Roy2 as

(select COl1,cast(COl2 as nvarchar(100))COl2,row from Roy where row=1

union all

select a.Col1,cast(b.COl2+','+a.COl2 as nvarchar(100)),a.row from Roy a join Roy2 b on a.COl1=b.COl1 and a.row=b.row+1)

select Col1,Col2 from Roy2 a where row=(select max(row) from roy where Col1=a.COl1) order by Col1 option (MAXRECURSION 0)

--生成结果:

/\*

Col1 COl2

----------- ------------

1 a,b,c

2 d,e

3 f

(3 行受影响)

\*/

排名函数:

--drop table Dept

CREATE TABLE Dept( id int PRIMARY KEY, parent\_id int, name nvarchar(20))

INSERT Dept values

(0, -1, N'<全部>'),(1, 0, N'财务部'),(2, 0, N'行政部'),(3, 0, N'业务部'),

(4, 0, N'客服部'),(5, 4, N'销售部'),(6, 4, N'MIS'),(7, 6, N'UI'),

(8, 6, N'程式开发'),(9, 8, N'财务开发'),(10, 8, N'进销存开发'),(11, 8, N'进销存开发'),(12, 8, N'商场开发')

GO

select \* from dept

--RANK()

select ID,parent\_id,RANK()over(partition by parent\_id order by name)as [rank],name

from Dept order by parent\_id

--DENSE\_RANK()

select ID,parent\_id,DENSE\_RANK()over(partition by parent\_id order by name)as [DENSE\_RANK],name

from Dept order by parent\_id

--NTILE()

select ID,parent\_id,NTILE(3) over(order by name)as [DENSE\_RANK],name

from Dept order by name

--ROW\_NUMBER()

select ID,parent\_id,ROW\_NUMBER() over(order by name)as [DENSE\_RANK],name

from Dept order by name

行列转换函数：

IF OBJECT\_ID('TB') IS NOT NULL

DROP TABLE TB

CREATE TABLE TB(id INT,customer varchar(20),jd varchar(40),price INT) ;

INSERT INTO TB

values(1,'001','第一季度',1000),(2,'001','第二季度',14000)

,(3,'001','第三季度',15000),(4,'002','第一季度',100)

,(5,'002','第二季度',1100),(6,'002','第三季度',2100)

,(7,'002','第四季度',3100),(8,'003','第四季度',3100)

--select \* from TB

--SQL 2000

SELECT customer,

[第一季度] = SUM(CASE WHEN jd = '第一季度' THEN price ELSE 0 END),

[第二季度] = SUM(CASE WHEN jd = '第二季度' THEN price ELSE 0 END),

[第三季度] = SUM(CASE WHEN jd = '第三季度' THEN price ELSE 0 END),

[第四季度] = SUM(CASE WHEN jd = '第四季度' THEN price ELSE 0 END)

FROM TB GROUP BY customer;

--SQL 2005

SELECT customer,[第一季度],[第二季度],[第三季度],[第四季度] --INTO #UNPIVOT

FROM

(SELECT customer,jd ,price

FROM TB ) p

PIVOT

( SUM (price)

FOR jd IN ([第一季度],[第二季度],[第三季度],[第四季度])

)AS unpvt

SELECT \* FROM #UNPIVOT

SELECT customer,JD,price

FROM

(SELECT customer,[第一季度],[第二季度],[第三季度],[第四季度]

FROM #UNPIVOT) p

UNPIVOT

(price FOR JD IN

([第一季度],[第二季度],[第三季度],[第四季度])

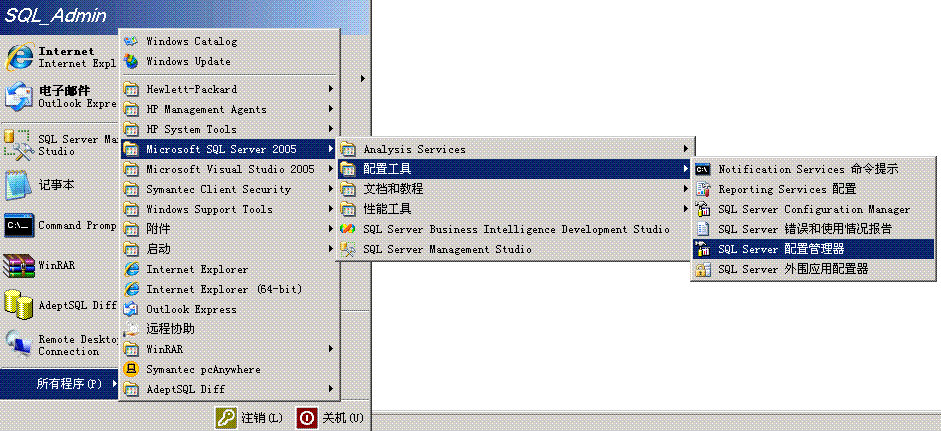
)AS unpvt

GO

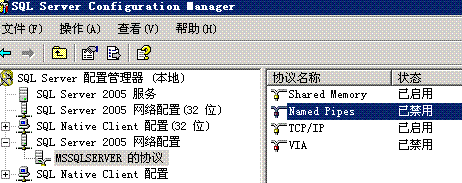
--DROP TABLE #UNPIVOT

附件1：数据库常见问题：数据库无法联接：

1. 防火墙屏蔽端口1433，可以通过：运行—》cmd--》telnet [IP地址] 1433 ，来判断端口是否打开
2. 命名管道关闭，如图：

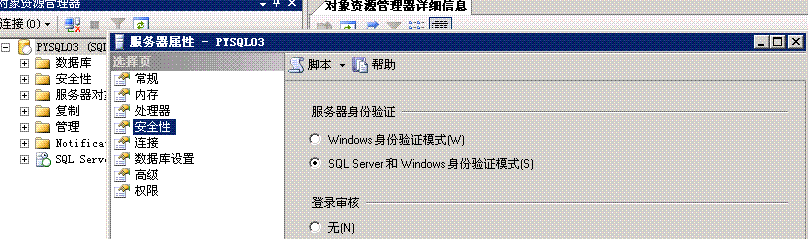


然后 选择Named Pipes 右键属性🡪启用.重新启动实例，即可。



1. 服务器验证模式，当出现集成账号可以访问，但SQL账号联接失败时，修改服务器验证模式。选择中实例—》右键属性—》安全性---》SQL SERVER 和Windows 身份认证模式—》实例重启，即可。

如下图：



1. 跨域链接：如果非默认实例，联接方式： IP\实例名 而不是 机器名\实例名

如果是默认实例，链接方式： IP 而不是 机器名

附件2：新增单据号获取方式

create table Sys\_BillType(ID int,Bill\_type varchar(10),Bill\_Code varchar(10),cnt int,modifydtm datetime ,remark varchar(10))

insert into Sys\_BillType(ID,Bill\_type,Bill\_Code,cnt)

VALUES(1,'POS','OS',1),(2,'INV','IM',1),(3,'Order','CK',1)

--SELECT \* FROM Sys\_BillType

go

create proc Get\_BillNo

(@Bill\_type varchar(10),

@Shop\_Code varchar(10),

@BillNo varchar(20) output)

as

begin

select @Billno=Bill\_Code+ --单据代码

convert(CHAR(6),GETDATE(),12)+ --开单日期

'-'+@Shop\_Code+'-'+ --店铺代码

right('0000'+CONVERT(varchar(5),CNT),4) --增量

FROM Sys\_BillType WHERE Bill\_type=@Bill\_type

UPDATE Sys\_BillType SET cnt= cnt%9999+1 WHERE Bill\_type=@Bill\_type

--Smallint 32767

end

/\*示例:\*/

set statistics time on

set statistics io on

go

declare @billno varchar(25)

exec Get\_BillNo 'POS','G00002',@billno output

select @billno

---原有生成模式

select MAX(RIGHT(billno,4)) from Pos\_Salemaster where BillNo like 'OS071201-G00002%'

--select top 10 \* from dbo.Pos\_Salemaster

--select top 10 \* from dbo.Inv\_MoveMaster

--select top 10 \* from dbo.Inv\_TransMaster

--select top 10 \* from dbo.Sal\_OrderMaster

--drop table Sys\_BillType;

--drop proc Get\_BillNo

--truncate table Sys\_BillType

附件3：某公司联合机房优化效果对比图。

                          （图一）

                                                      （图二）

说明：优化从2010年3月12日开始，

统计时间2009-11-12至2010-05-11，

其中图一  纵向表示：每个月CPU>40% 的小时次数，

横向表示：优化前后日期；

其中图二  纵向表示：每个月CPU>60% 的小时次数，

横向表示：优化前后日期；

数据来源：SCOM监控邮件之：数据库服务器性能日报表

此次优化，系统共修改了86张表的索引，其中包含调整组合索引的顺序，更改索引的类别等；修改表字段类型一个，存储过程两个。

           货品分发系统共修改视图两个，新建触发器两个。