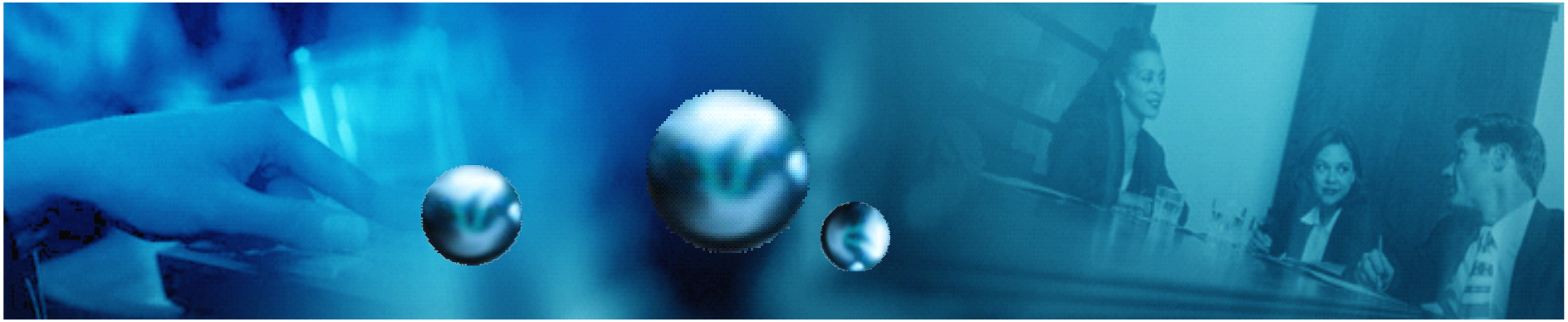
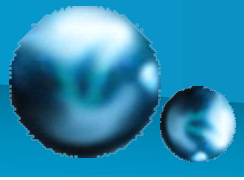
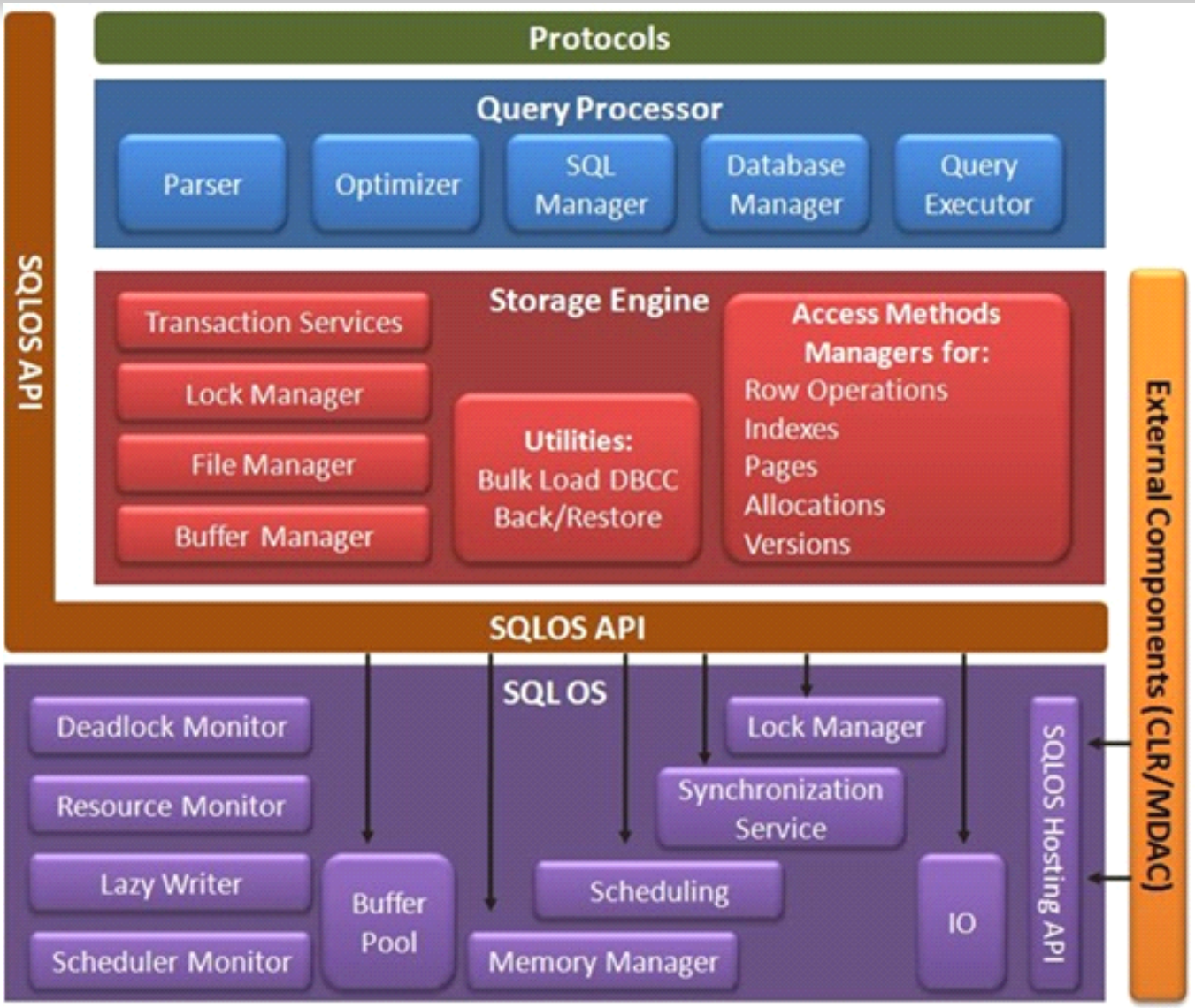
SQL Server 2005性能优化





目录

SQL Server 2005系统架构



四大组件

- **协议组件**：负责接收请求并把它们转换成关系引擎能够识别的形式。它还能够获取任意查询、状态信息、错误信息的最终结果，然后把这些结果转换成客户端能够理解的形式，最后再把它们返回到客户端。

- **查询处理器组件**：负责接受SQL批处理然后决定如何处理它们。对T-SQL查询和编程结构，查询处理器可以解析、编译和优化请求并检查批处理的执行过程。如果批处理被执行时需要数据，它会发送一个数据请求到存储引擎。

- **存储引擎组件**：负责管理所有的数据访问，包括基于事务的命令（Transaction-based command）和大批量操作（Bulk Operation）。这些操作包括备份、批量插入和某些数据库一致性检查（Database Consistency Checker，DBCC）命令。

- **SQLOS组件**：负责处理一些通常被认为是操作系统职责的活动，例如线程管理（调度），同步单元（Synchronization Primitive），死锁检测和包括缓冲池(Buffer Pool)的内存管理。

性能调优方法学

最困难
但最有成效

最简单
但是收效最少

常见的性能问题

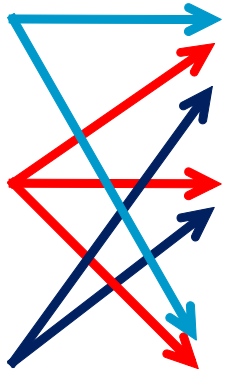
常规情况下

磁盘负载过重 → 架构不良

内存负载过重 → 代码不良

处理器负载过重 → 代码不良

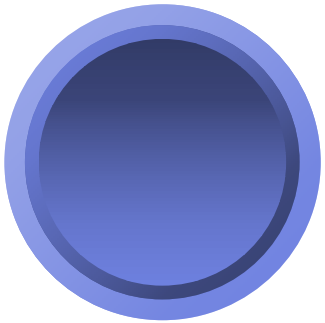
网络负载过重 → 客户端代码不良 → 调整客户端设计



调整架构

调整代码

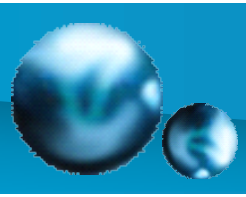
调整存储



数据架构

如何设计良好的关系型数据库架构

- 适度的规范化，适度的冗余
- 对数据热区的判断
  - 根据数据热区定义索引、表分割定义
- 优化SELECT查询
  - 尽量将数据存储在同一张表中
  - 使用索引及索引覆盖策略
- 优化UPDATE事务
  - 尽量将需要更新的数据放在一张较小的表中
- 优化DELETE事务
  - 在大规模删除中评估分区的效果
- 优化INSERT事务
  - 减少对自动编号的依赖

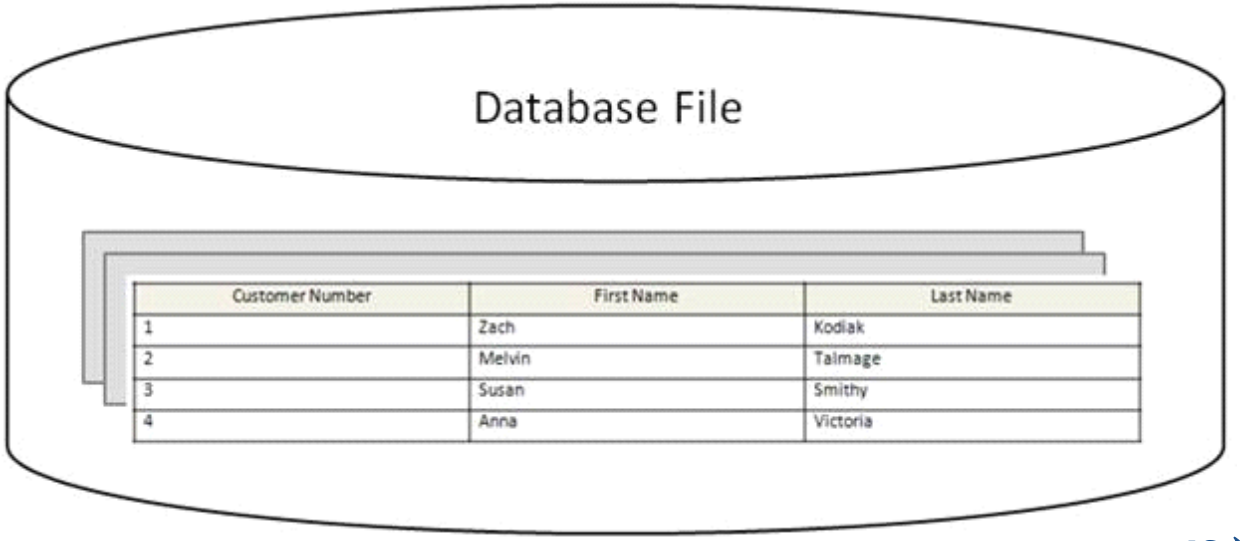


表

Customer Number	First Name	Last Name
1	Zach	Kodiak
2	Melvin	Talmage
3	Susan	Smithy
4	Anna	Victoria

表是如何存储的？

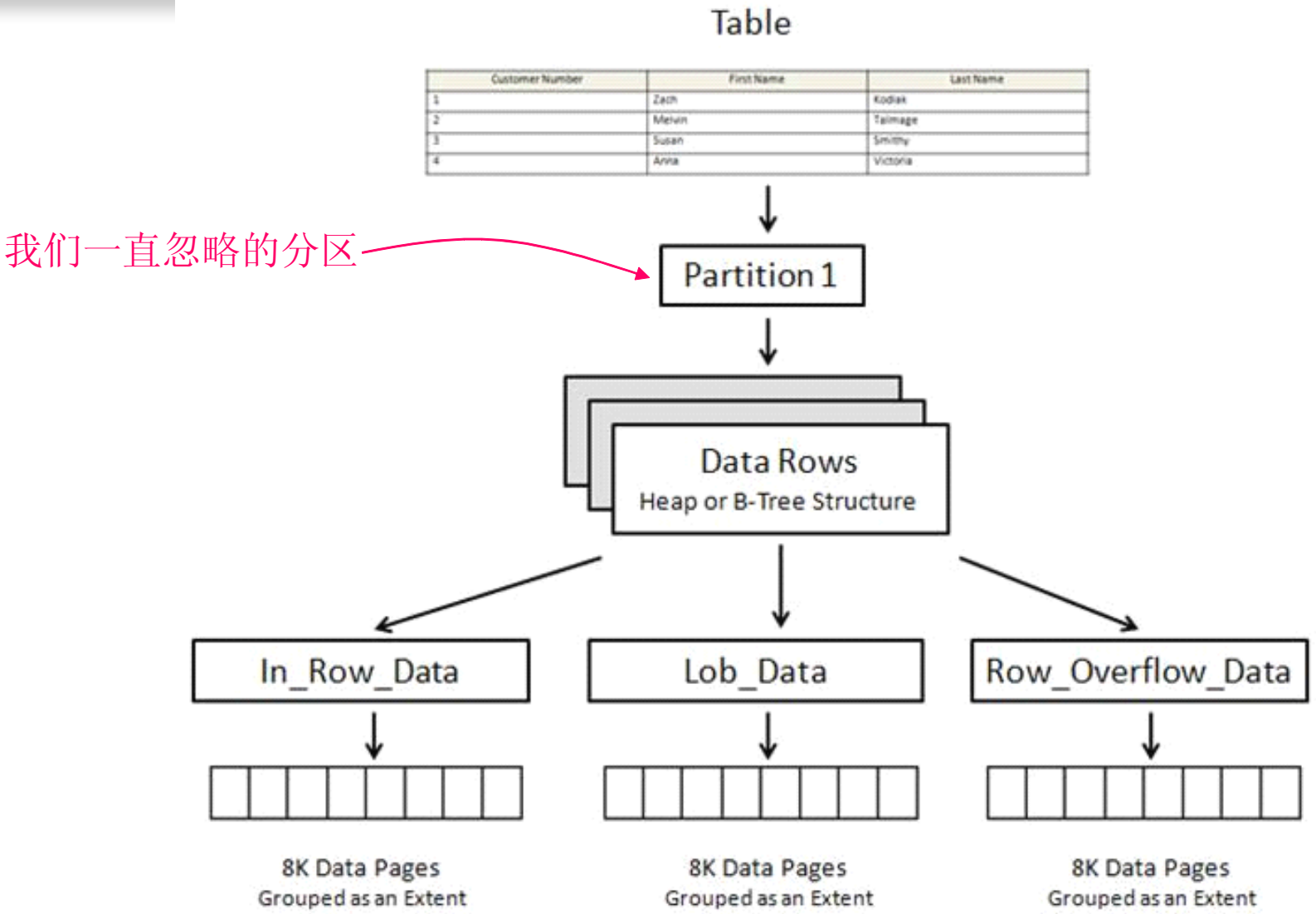
主数据库文件



mdf文件

ldf文件不是用于表存储，在此不讨论。

表的物理存储元素



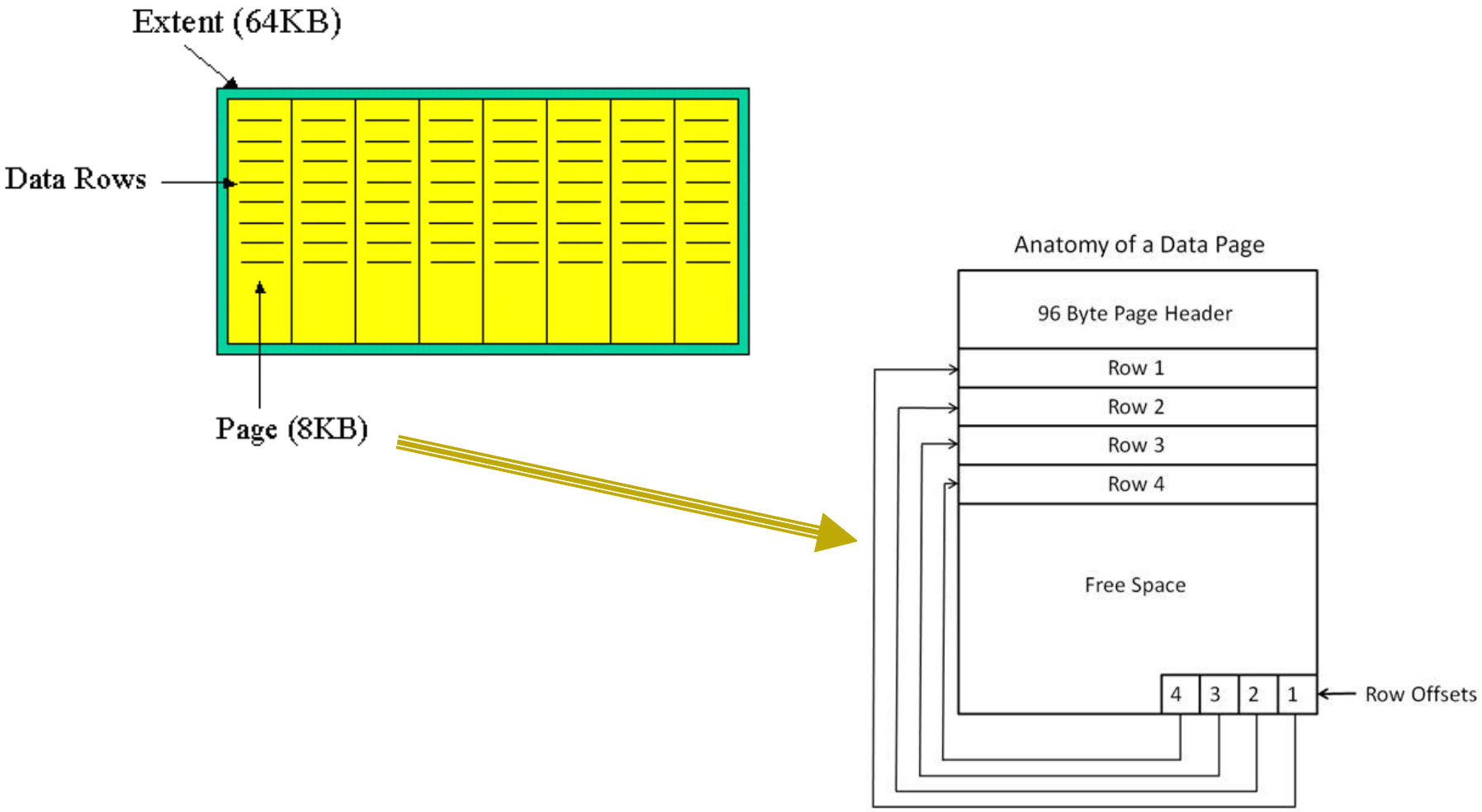
表的物理存储分配单元

- In_Row_Data：最常见的分配单元类型，用于存储数据与索引
- Lob_Data：用于存储text, ntext, xml, image, varchar(max), nvarchar(max), varbinary(max)类型的数据
- Row_Overflow_Data： varchar, nvarchar, varbinary, sql_variant超出8060字节的部分

Page存储的数据种类

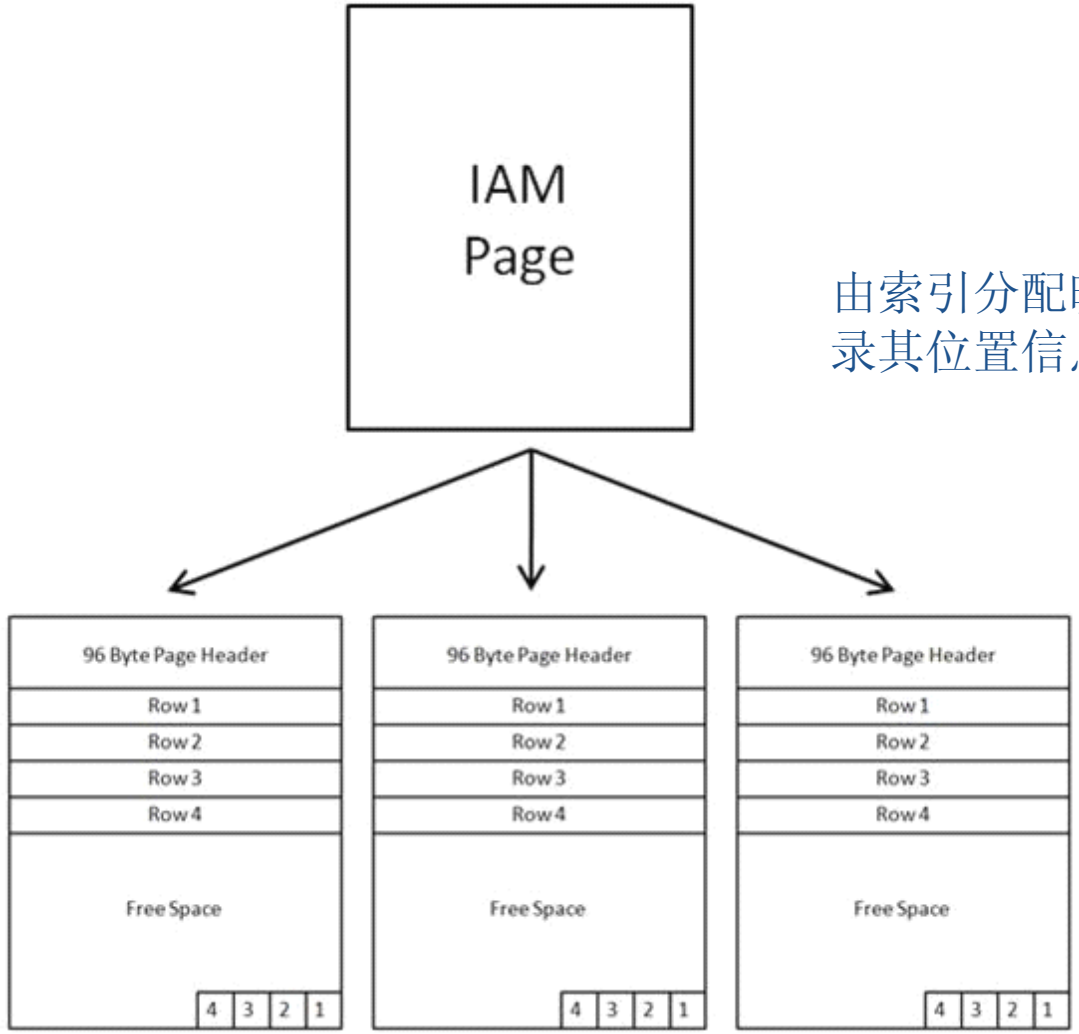
1. 数据
2. 索引
3. Text/Image
4. 全局分配映射 (GAM)/共享全局分配映射 (SGAM): 存储extent的分配信息
5. Page自由空间 (PFS): 存储有关已分配page以及page内自由空间的数量
6. 索引分配映射 (IAM): 存储关于分配给表和索引的extent的相关信息
7. 批量数据变更映射(BCM): 存储自上次事务日志备份以来，由批量操作而改变的extent的相关信息
8. 差异变更映射(DCM):存储自上次数据库备份以来，改变的extent的相关信息

存储分配单元的物理结构



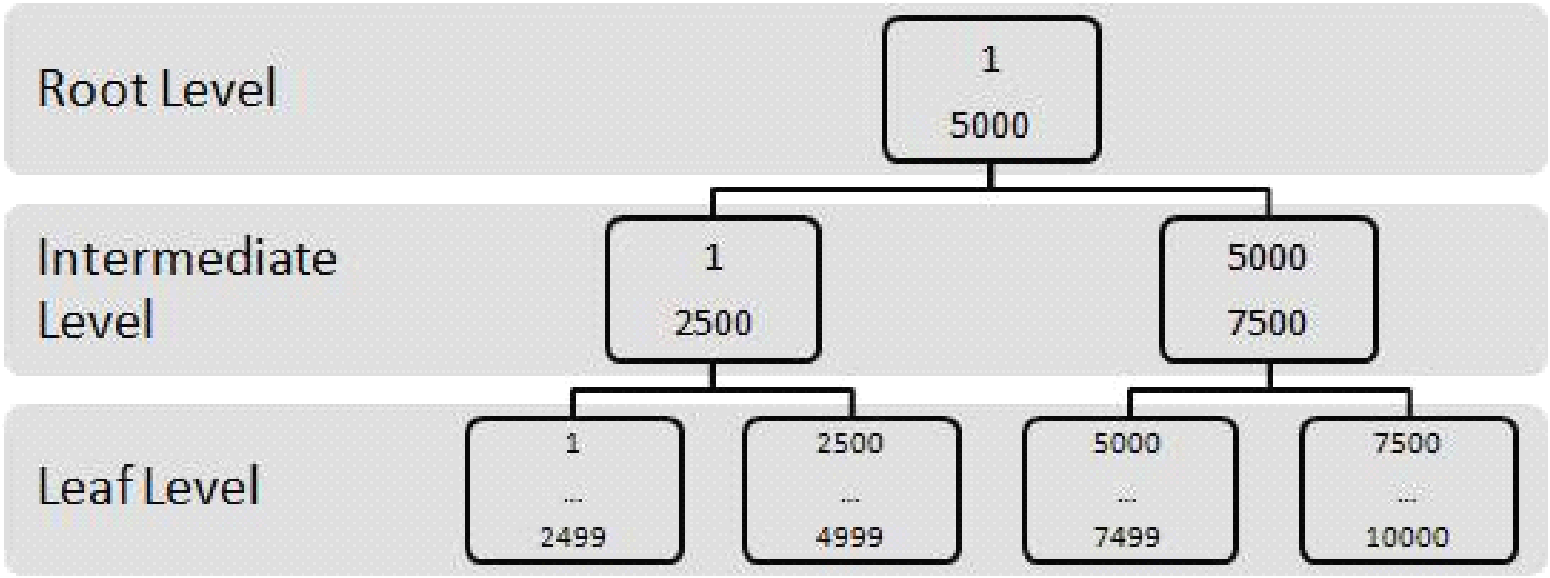
一个page内数据行的数量取决于其的大小

存储在堆中的数据

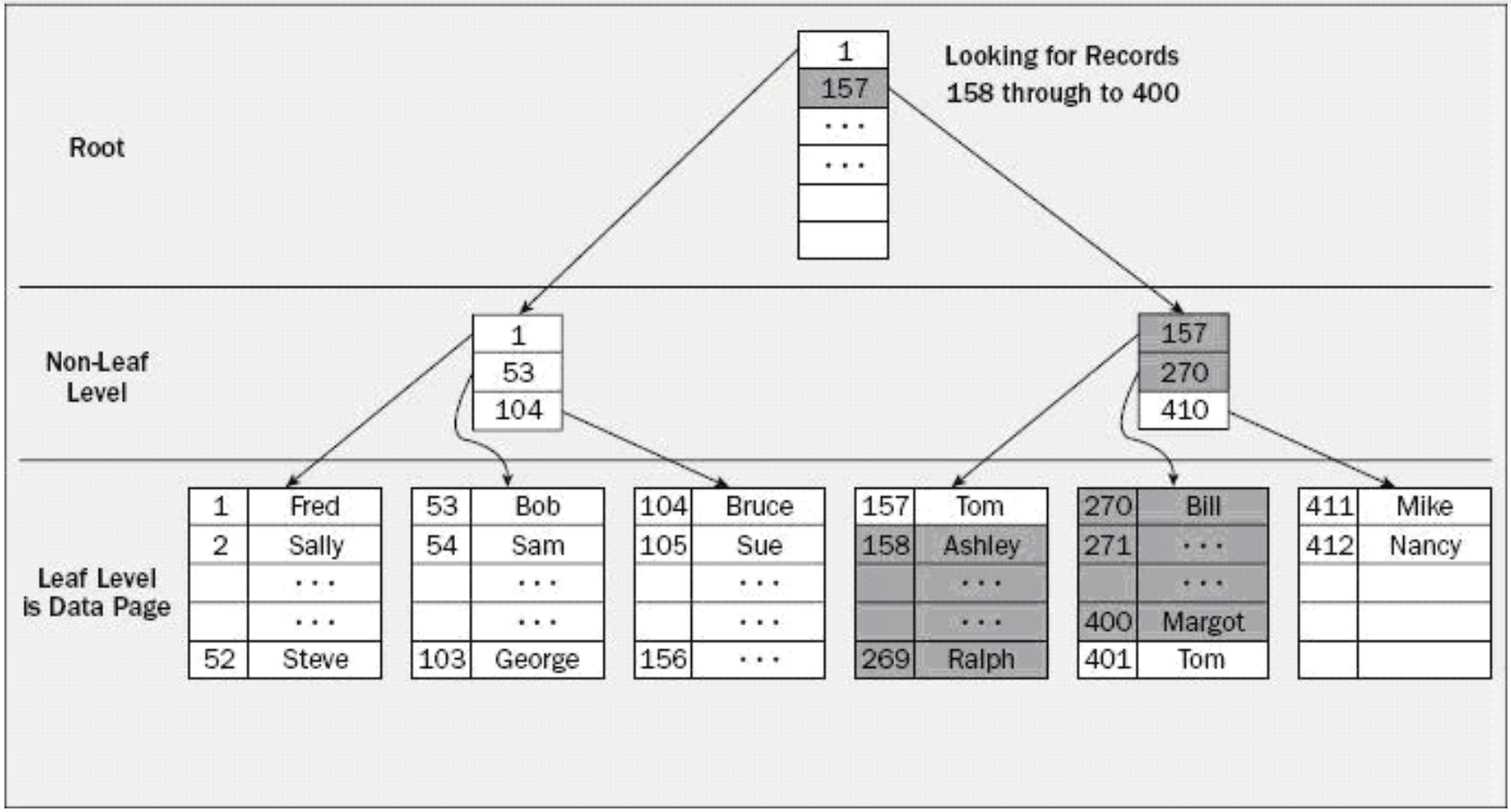


由索引分配映射(IAM)记录其位置信息

B树索引

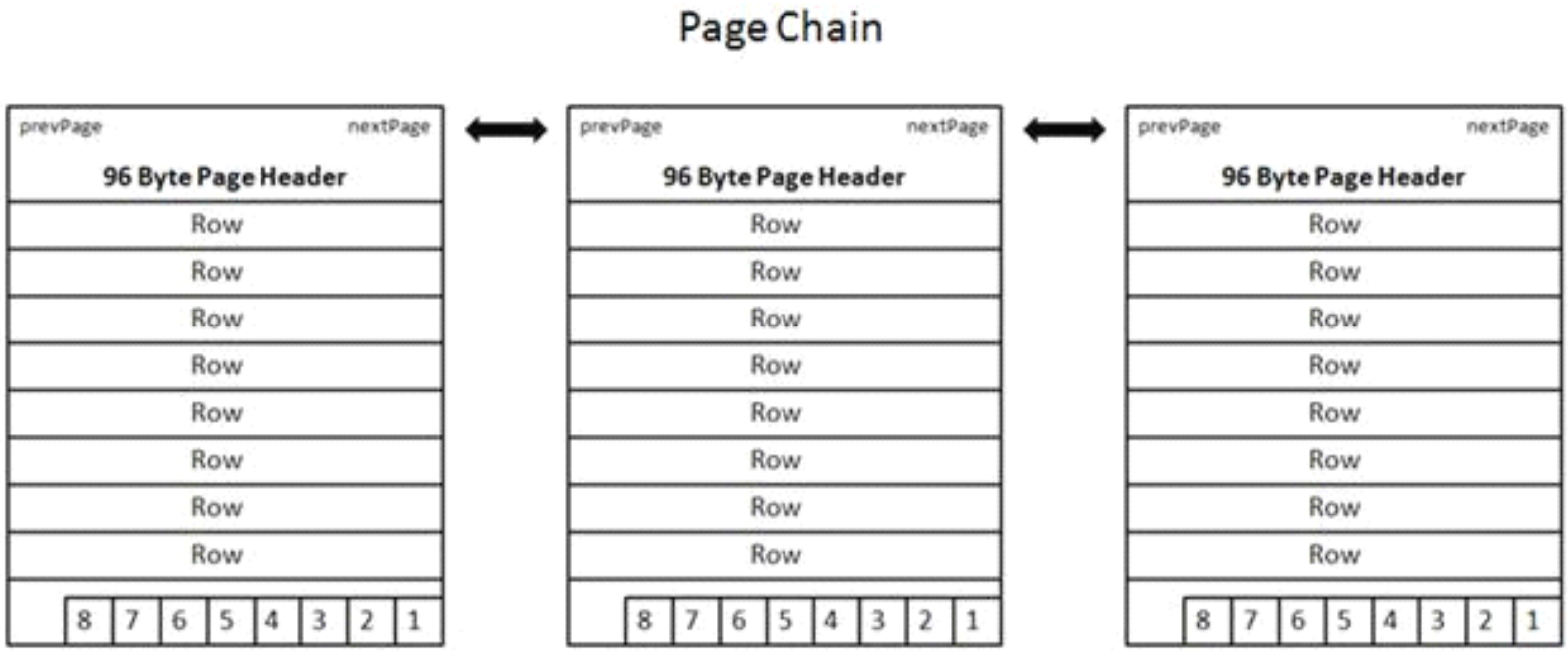


聚集索引



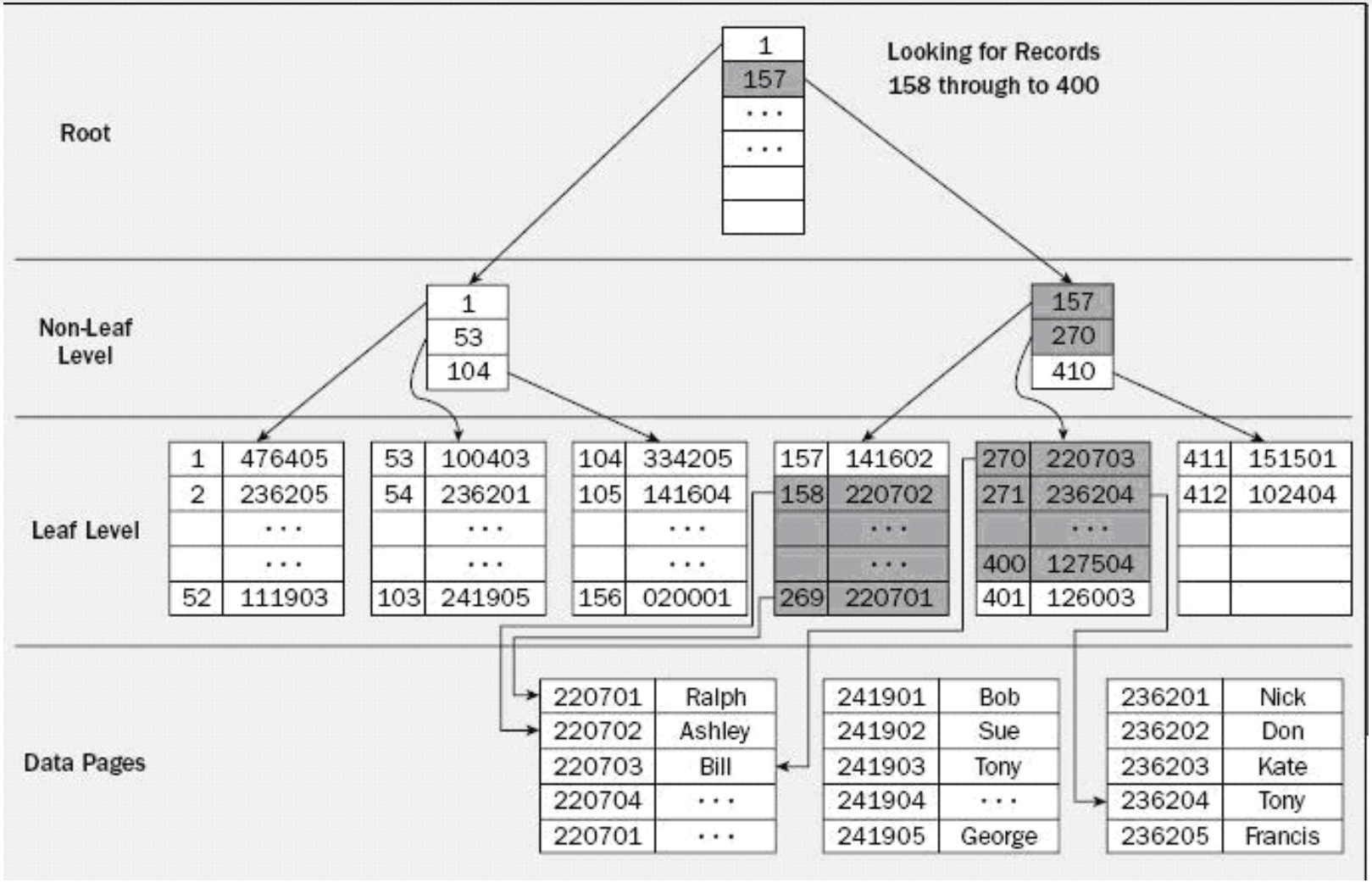
在聚集索引中，数据与索引是相同的，数据就存放在索引的叶子结点上。

聚集索引



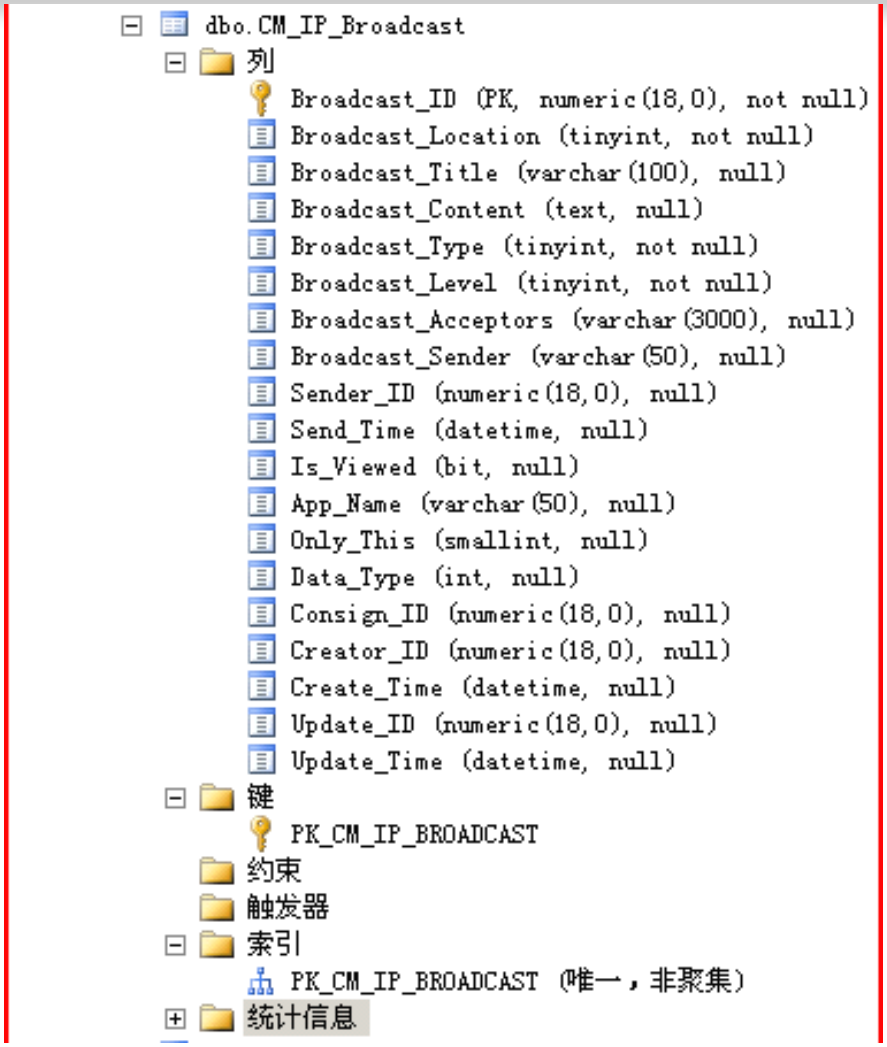
聚集索引的叶子结点是物理连续的(双向链表结构)。

非聚集索引



非聚集索引的叶子结点并不存储数据，而是存放指向数据的指针

站内短信Messages

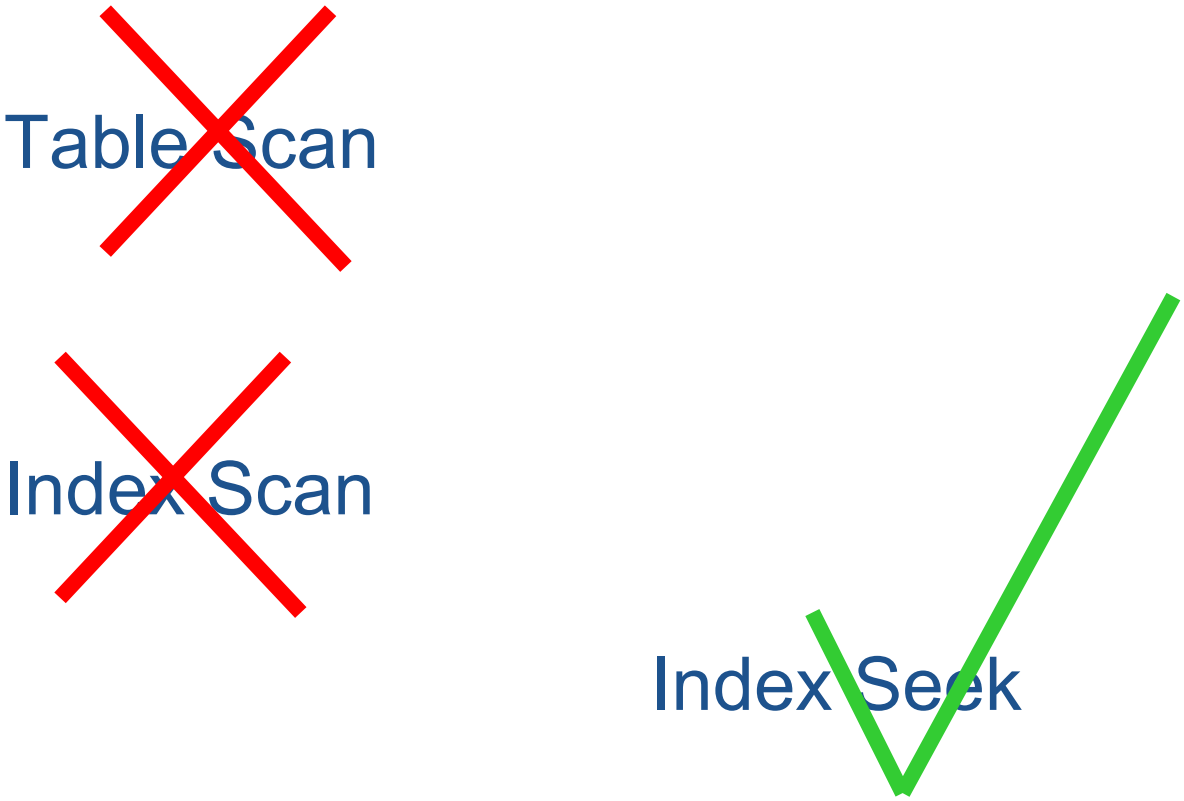


查询示例

- select * from Messages where Message_ID =1000
- select * from Messages where Message_ID >999 and Message_ID <1017
- select * from Messages where Message_ID >1000 and Message_ID <1017
- select * from Messages where Message_ID >1000 and Message_ID <1018

在创建聚集索引前后，为何查询计划不同？

查询计划



索引原则(1)

- 聚集索引：对表建立主键时，就会为主键自动添加了聚集索引，如id字段，而我们没有必要把聚集索引浪费在主键上，除非只按主键查询，所以应该把聚集索引设置在按条件查询频率最高的那个字段或者组合的字段。

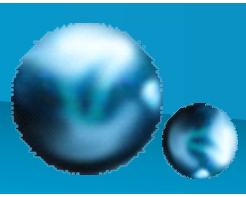
```
select name from users where user_id = 119
select name from users where user_name like '张三%'
```

索引原则(2)

- 聚集索引的建立要根据实际应用的需求来进行，并非是在任何字段上建立索引就能提高查询速度。聚集索引建立遵循下面几个原则：
- 包含大量非重复值的列
- 使用下列运算符返回一个范围值的查询：BETWEEN、>、>=、< 和 <=
- 被连续访问的列
- 返回大型结果集的查询
- 经常被使用join或 group by 子句的查询访问的列;一般来说，这些是外键列。对order by 或group by子句中指定的列进行索引，可以使 SQL Server 不必对数据进行排序，这样可以提高查询性能。

索引原则(3)

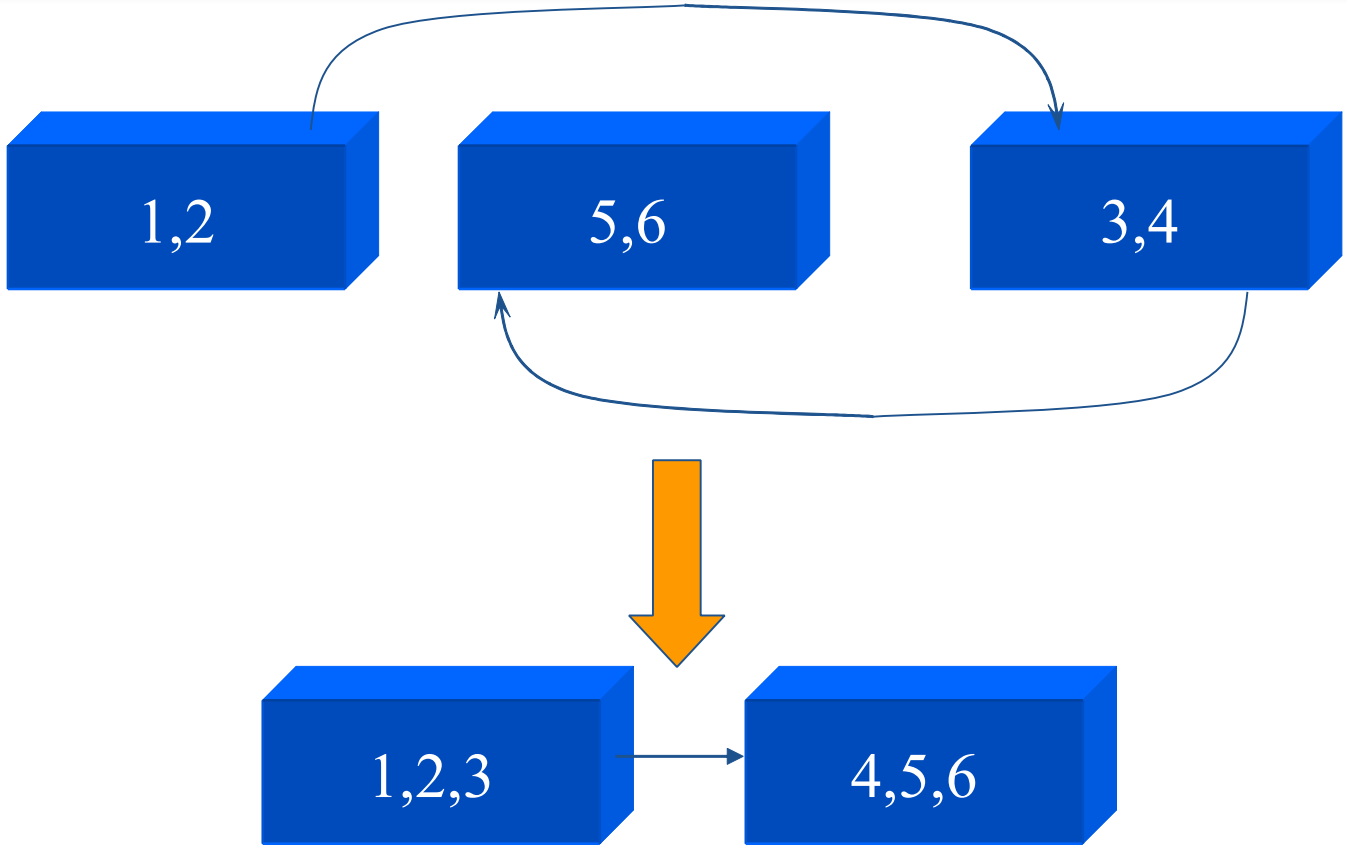
- 复合索引：在聚集索引中按常用的组合字段建立索引，形成复合索引。经常同时存取多个字段，且每个字段都含有重复值可考虑建立复合索引,这样能形成索引覆盖,提高where语句的查询效率。
- 组合索引要尽量使关键查询形成索引覆盖，其前导列一定是使用最频繁的列。



索引原则(4)

- 改变一个表的内容，将会引起索引的变化。频繁的对数据操作如insert,update,delete语句将导致系统花费较大的代价进行索引更新，引起整体性能的下降。
- 一般来讲，在对查询性能的要求高于对数据维护性能要求时，应该尽量使用索引，有时在这种操作数据库比较频繁的某些极端情况下，可先删除索引，再对数据库表更新大量数据，最后再重建索引，新建立的索引总是比较好用。

索引碎片整理



索引重构

DBCC SHOWCONTIG 正在扫描'CM_CONF_Permission' 表...
表: 'CM_CONF_Permission' (377768403)；索引ID: 0，数据库ID: 15
已执行TABLE 级别的扫描。
- 扫描页数.....: 46
- 扫描区数.....: 11
- 区切换次数.....: 10
- 每个区的平均页数.....: 4.2
- 扫描密度[最佳计数:实际计数].....: 54.55% [6:11]
- 区扫描碎片.....: 90.91%
- 每页的平均可用字节数.....: 6559.2
- 平均页密度(满).....: 18.96%

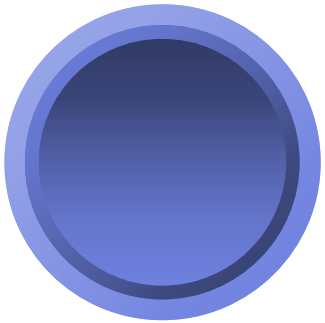
需要重构

DBCC SHOWCONTIG 正在扫描'Messages' 表...
表: 'Messages' (1396252079)；索引ID: 1，数据库ID: 15
已执行TABLE 级别的扫描。
- 扫描页数.....: 58
- 扫描区数.....: 8
- 区切换次数.....: 7
- 每个区的平均页数.....: 7.3
- 扫描密度[最佳计数:实际计数].....: 100.00% [8:8]
- 逻辑扫描碎片.....: 0.00%
- 区扫描碎片.....: 25.00%
- 每页的平均可用字节数.....: 222.6
- 平均页密度(满).....: 97.25%

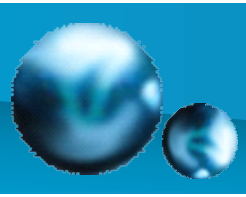
无需重构

索引原则(5)

- 查询中用于计算的字段不要加索引。例如查询select * from Messages where DATEPART(year, Create_Time) =‘2009’，逐条扫描，对于Create_Time字段加索引是没有效果的。



查询代码



SARG

- 在查询分析阶段，查询优化器查看查询的每个阶段并决定限制需要扫描的数据量是否有用。如果一个阶段可以被用作一个扫描参数(SARG)，那么就称之为可优化的，并且可以利用索引快速获得所需数据。

  SARG的定义：用于限制搜索的一个操作，因为它通常是指一个特定的匹配，一个值的范围内的匹配或者两个以上条件的AND连接。形式如下：

  列名 操作符 <常数 或 变量>

  或

  <常数 或 变量> 操作符 列名

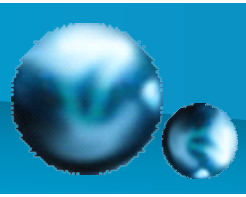
SARG示例

- Name='张三'
- price>5000
- 5000<价格
- Name='张三' and pice>5000

- 如果一个表达式不能满足SARG的形式，那它就无法限制搜索范围，也即SQL Server必须对每一行都判断它是否满足where子句中的所有条件。所以一个索引对于不满足SARG形式的表达式来说是无用的。

SARG，非SARG

- SARG：name like '张三%'
- 非SARG：name like ‘%张三‘
- SARG：Name='张三' and Price>5000
- 非SARG：Name='张三' or Price>5000
- SARG：Price=5000
- 非SARG：abs(Price)=5000



不可优化的Where子句

- select * from Users where user_name like ‘%张三%’
- select * from Messages where DATEPART(year, Create_Time) =‘2009’
- select * from My_Account where amount/30 < 1000
- select count(*) from Users where address in(‘杭州’,’上海’)

与索引协调使用的SQL查询

- 不能利用索引的SQL语句都需要考虑优化

- select * from My_Account where amount/30 < 1000
- select * from My_Account where amount < 1000/30

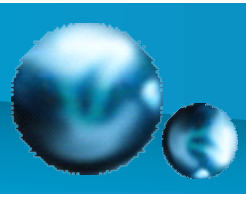
In

- select count(*) from Users where address in(‘杭州’,’上海’)
- create procedure count_user as
  begin
  declare @a int
  declare @b int

  select @a=count(1) from users where address =‘杭州’
  select @b=count(1) from users where address =‘上海'

  select total=@a+@b
  end

尽量不要在where子句中使用IN



示例表

```
color
----------
Black
Blue
Green
Red
Products table:
sku   product_description   color
----  -------------------- ------
1     Ball                 Red
2     Bike                 Blue
3     Tent                 NULL
```

Not in的改进(1)

原写法

```
SELECT C.color
FROM Colors AS C
WHERE C.color NOT IN (SELECT P.color
                      FROM Products AS P)
```

改进写法1

```
SELECT C.color
FROM Colors AS C
WHERE NOT EXISTS(SELECT *
                 FROM Products AS P
                 WHERE C.color = P.color);
```

Not in的改进(2)

改进写法2

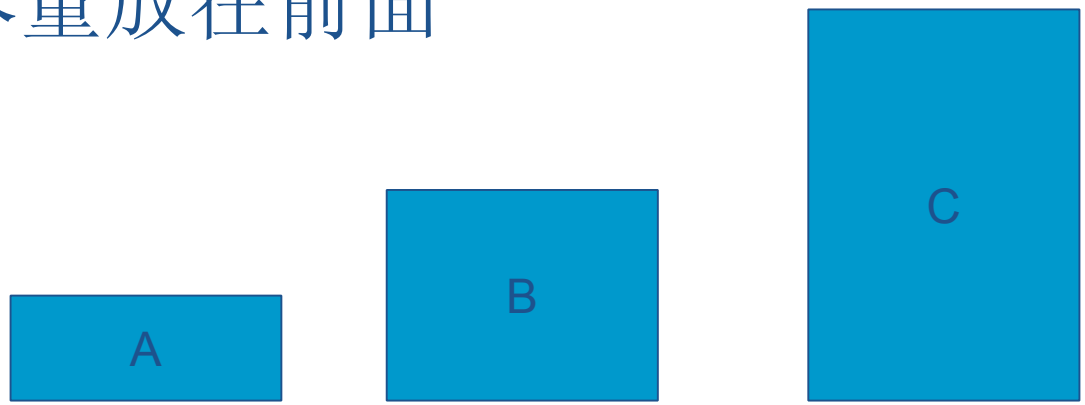
- SELECT color FROM Colors
  EXCEPT
  SELECT color FROM Products

改进写法3

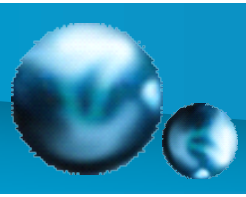
- SELECT C.color
  FROM Colors AS C
  LEFT OUTER JOIN Products AS P
  ON C.color = P.color
  WHERE P.color IS NULL;

join的问题

- 连接条件要充分考虑带有索引的表、行数多的表，并注意优化表顺序，行数少的表尽量放在前面

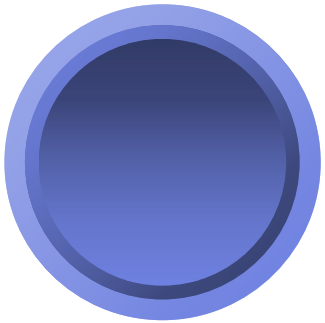



```
select name, address from A
join B on A. id = B.id
join C on C.id = A.id
where name like ‘张三%'
```



其他注意事项

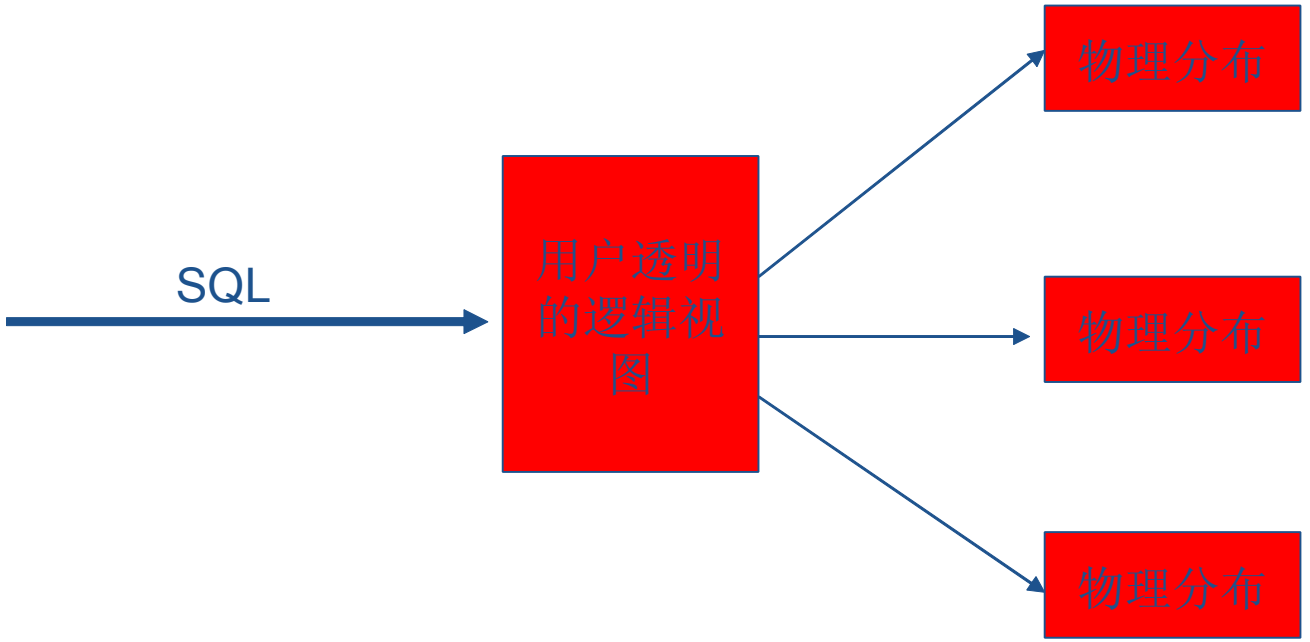
- select id, name, age from users，
  而非select * from users
- 尽量使用基于Set的逻辑，减少使用基于Row的逻辑
- 尽量不要使用cursor
- 对于小数据集使用表变量而不是临时表
- 对于需要反复使用的大数据集使用临时表而不是表变量
- 尽量使用WHERE替代HAVING



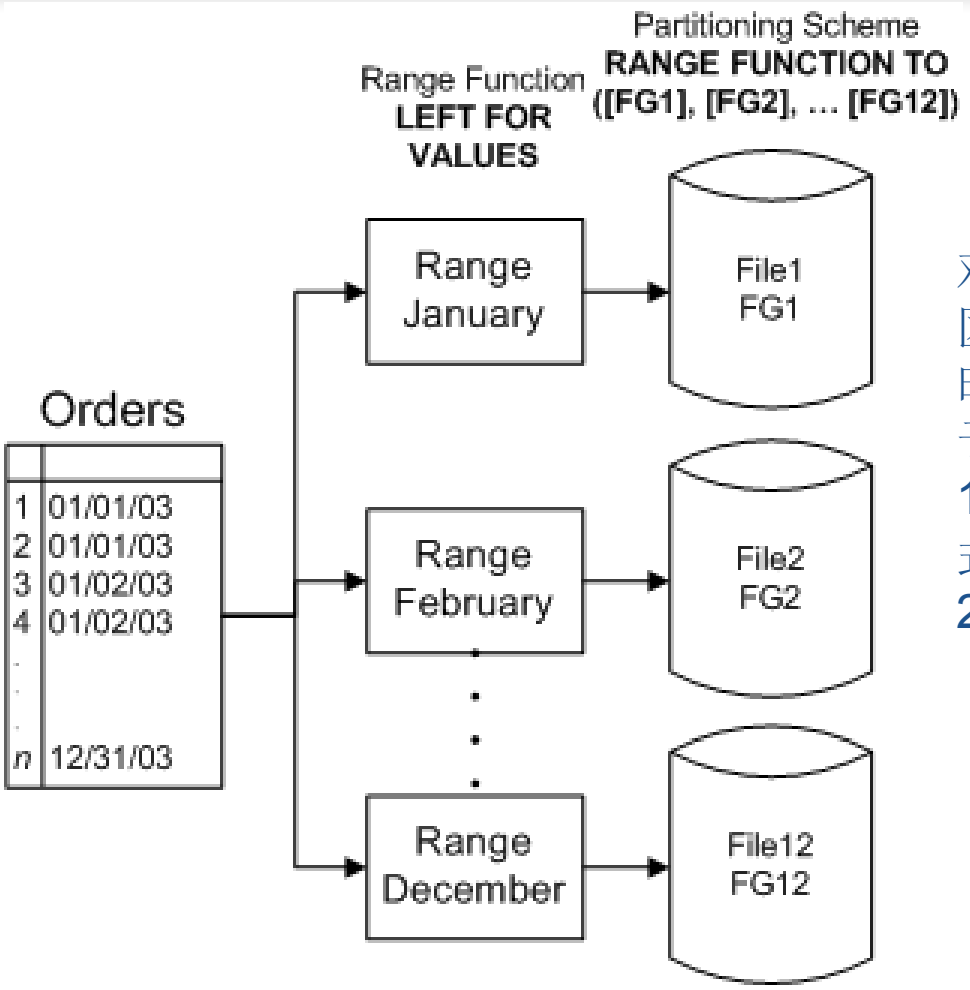
存储设计

Divide and Conquer

- 分区：表分区、索引分区
- 分库
- SQL Server集群



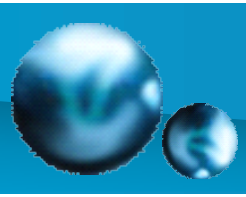
表分区



对数据库中的每一个表都进行分区并不现实。如果某个大型表同时满足下列两个条件，则可能适于进行分区：

1.该表包含或将包含以多种不同方式使用的大量数据

2.维护开销超过了预定义的维护期



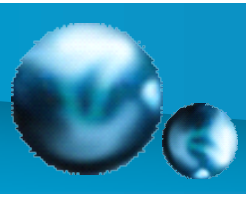
表分区的方式

- 垂直分区：将原始表分成多个只包含较少列的表
- 水平分区：水平分区将表分为多个表。每个表包含的列数相同，但是行更少。

- 通常不建议采用垂直分区

创建表分区的步骤

- 创建文件组
- 创建分区函数：根据一个数据字段中的值把一个表或目录的行映射到预先定义的分区中
- 创建分区方案：将分区连接到指定的文件组
- 对一个表进行分区：表创建指令中添加一个“ON”语句，用来指定分区方案以及应用该架构的字段



索引分区

- 建议使用相同的分区函数和分区方案对表及其索引进行分区，保持存储对齐。

分区的效能

- 在良好的设计的情况下，是可以极大提升系统查询性能的。
- 但分区是一把双刃剑，并不总能提高效率。需要充分考虑：

1. 频繁查询的字段与数据分布的关系
2. 服务器的物理架构

tempdb

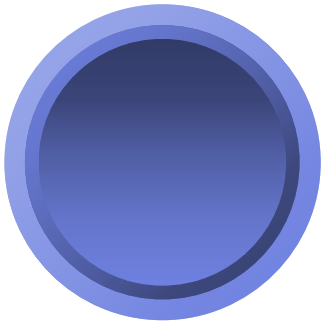
- 查询
- 触发器
- 快照隔离级别以及读写快照（RCSI）
- MARS
- 联机的索引创建
- 临时表，表变量，以及表值函数
- DBCC CHECK
- LOB参数
- 游标
- Service Broker以及事件通知
- XML以及LOB变量
- 查询通知
- 数据库邮件
- 索引创建
- 用户定义函数

用到**tempdb**空间的特性列表

有限的可用空间和过多的DDL/DML会使Tempdb超过负载，导致运行在同一个服务器中的其他无关应用变得运行缓慢或失败。

tempdb配置的考虑

- 1、 允许 tempdb 数据库自动根据需求进行扩展。对于那些生成的中间结果集超出 tempdb 数据库预期的查询操作来说，这样能够保证在执行完成前查询过程不会中断。
- 2、 将 tempdb 数据库文件的原始大小设置为一个合理的数值，避免文件自动扩展过多的空间。如果 tempdb 数据库扩展过于频繁，性能可能会受到影响。这也适用于用户数据库的数据文件。
- 3、 将文件增长百分率设置为一个合理的数值，避免tempdb数据库文件增长过于频繁。如果文件增长相比与写入tempdb数据库的数据量要小，那么 tempdb 可能需要不断扩展，从而影响性能。
- 4、 将tempdb数据库放置在高速I/O子系统中，确保良好的性能。将tempdb数据库分割在多个磁盘中，能够实现更好的性能。并使用文件组，将tempdb数据库和用户数据库放置在不同的磁盘中。



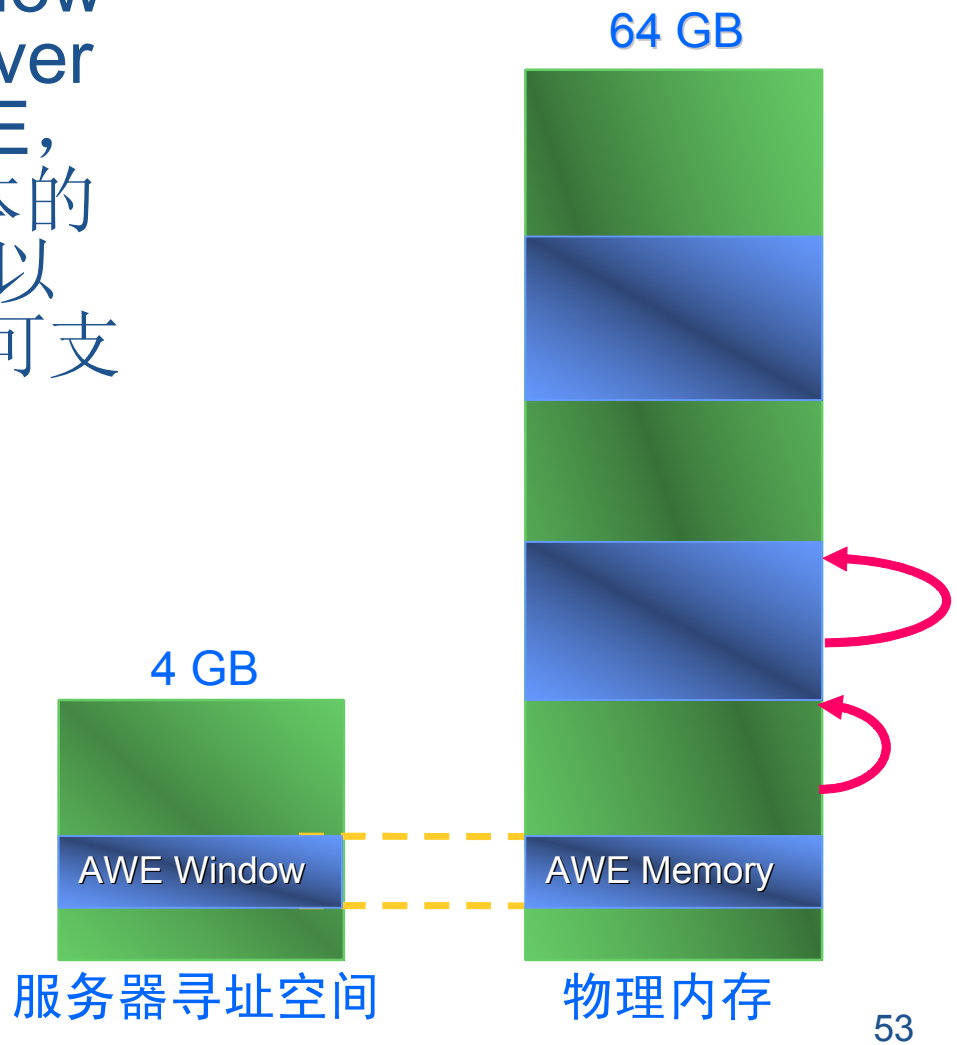
硬件配置

32位环境下启动PAE模式

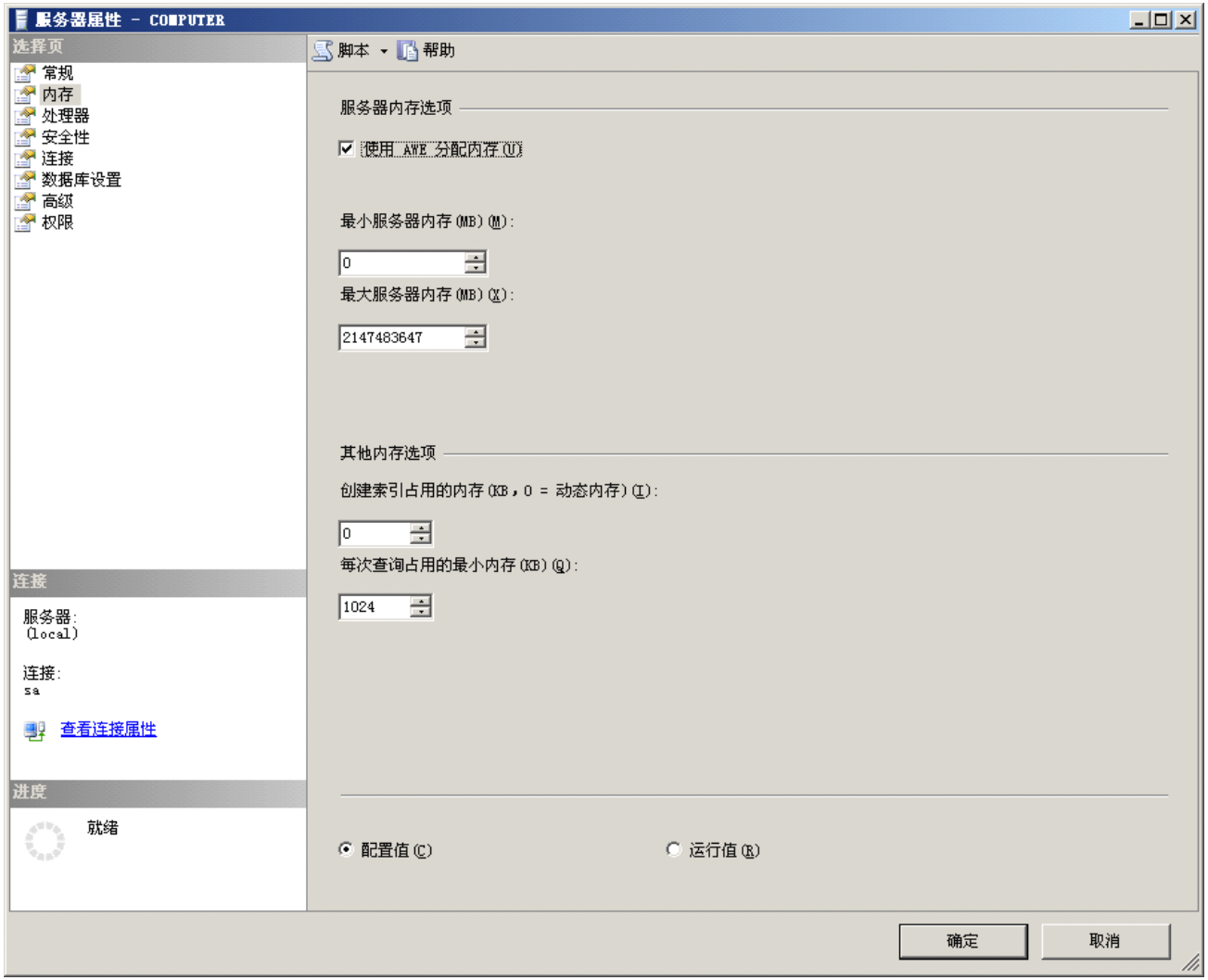
- PAE—Physical Address Extension
- 服务器内存>= 2 GB，<= 4 GB
  - Boot.ini 加上 /3GB，如下：
    - multi(0)disk(0)rdisk(0)partition(1)\%systemroot%="Windows Server 2003, Datacenter Edition" /3GB
- 服务器内存> 4 GB，<= 16 GB
  - Boot.ini 除了加上 /3GB的参数，还必须要再加上 /PAE 的参数
- 服务器安装超过 > 16 GB
  - 此时就不须在Boot.ini加上 /3GB的参数，只要加上 /PAE 的参数就可以

32位环境下启动AWE模式

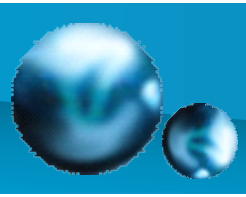
- AWE—Address Window Extension，SQL Server 2005 企业版支持AWE，从而允许在 32 位版本的 Windows 使用 4 GB 以上的物理内存。最多可支持 64 GB 的物理内存



AWE配置



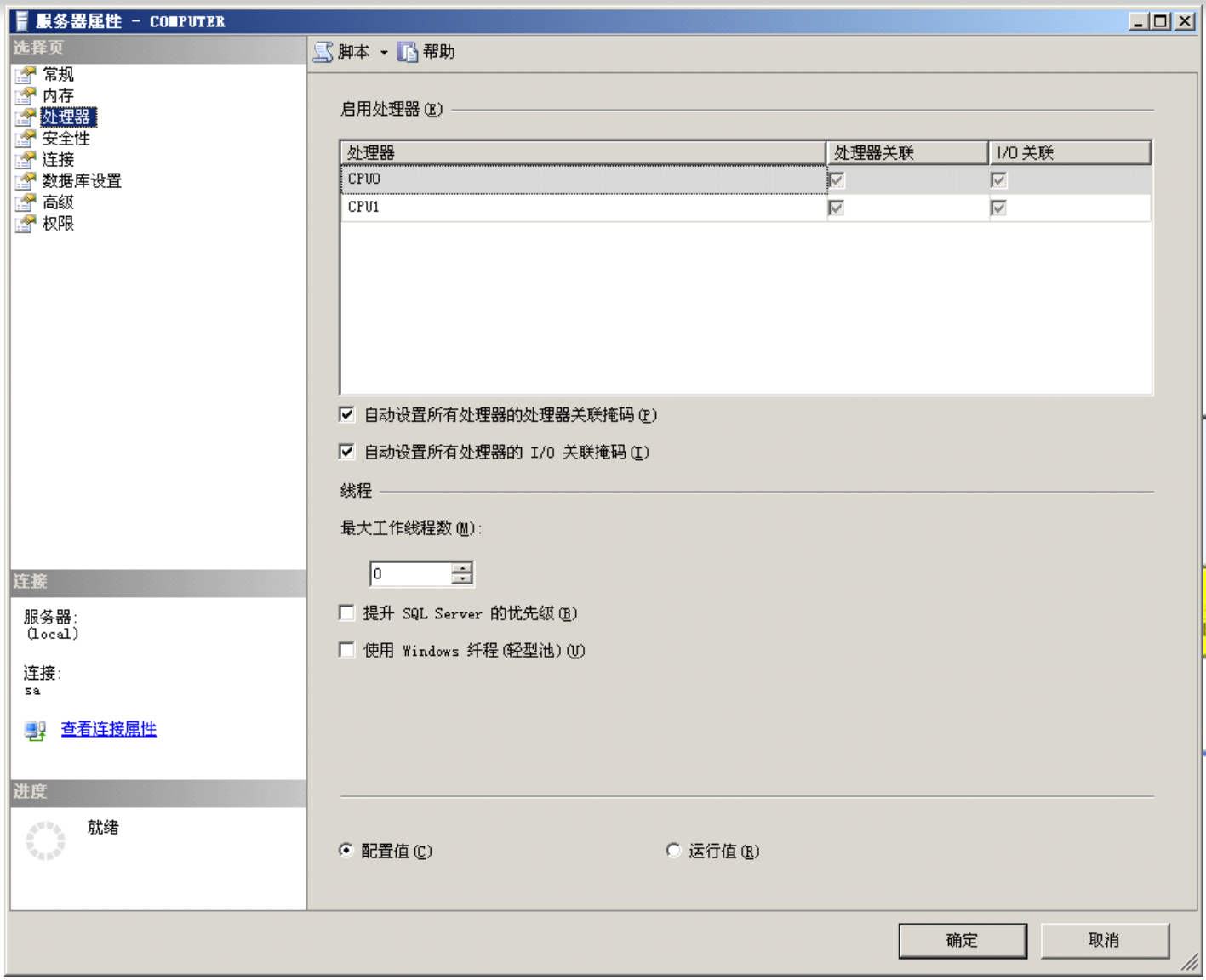
AWE 内存是动态分配的



64位版本

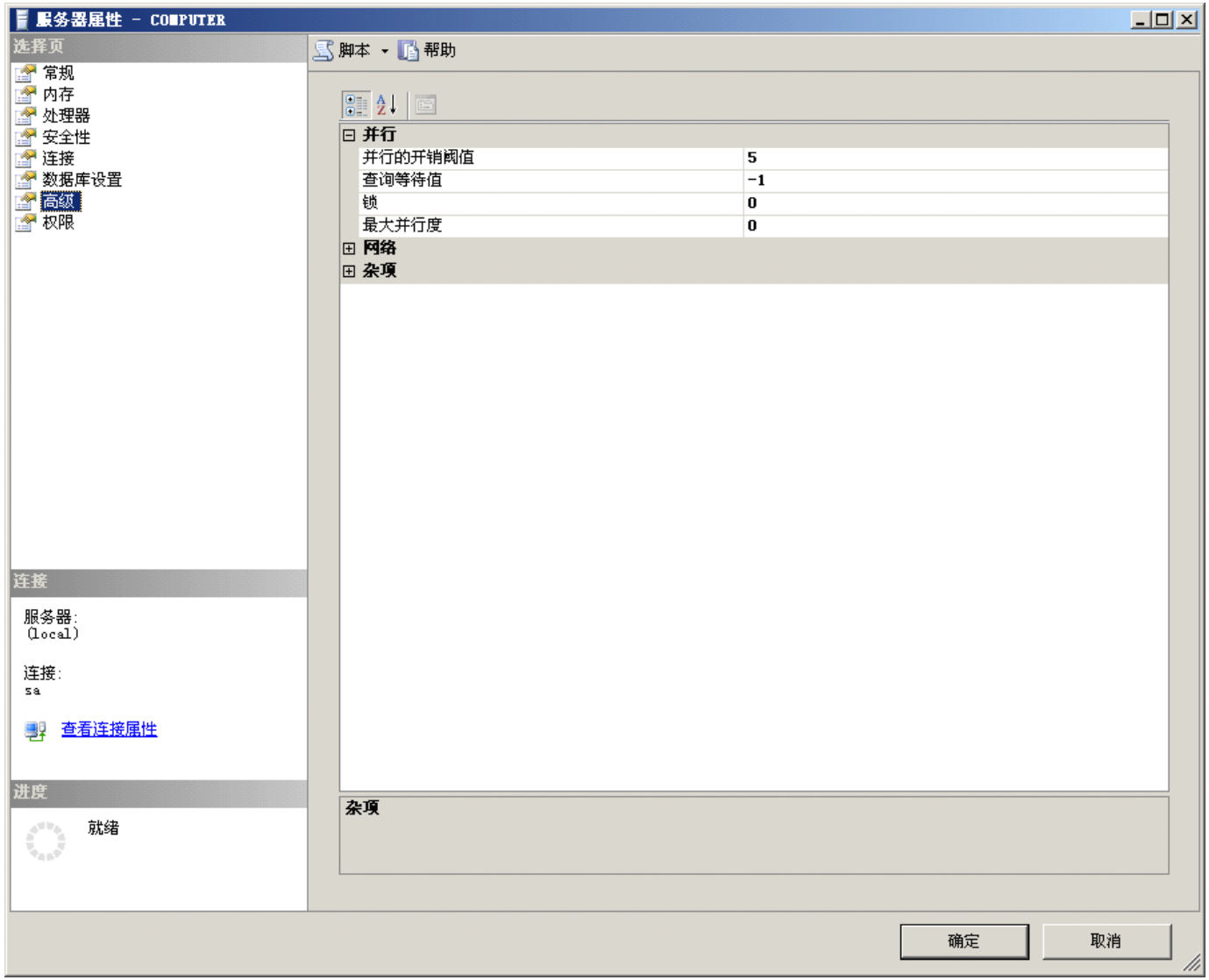
- 当处理器是性能瓶颈时，64位SQL Server版本将有较大的帮助（需64位CPU、64位Windows支持）
- 当磁盘I/O是性能瓶颈时，64位版本对性能提升的帮助则非常有限

CPU并行计算



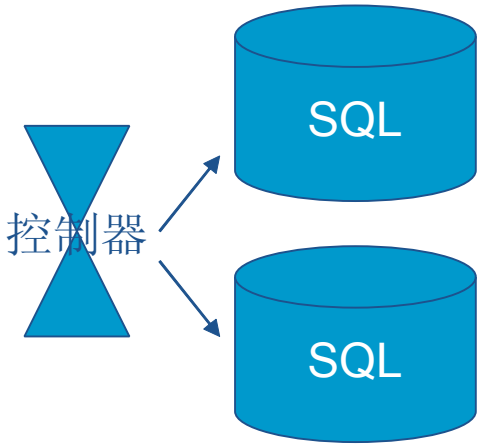
- 指定多CPU进行并行计算
- 启用纤程

配置并行查询选项

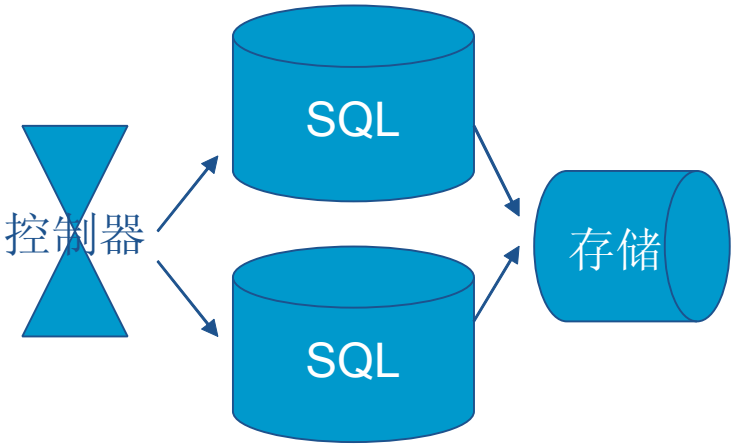


数据库集群

- 负载均衡与故障迁移

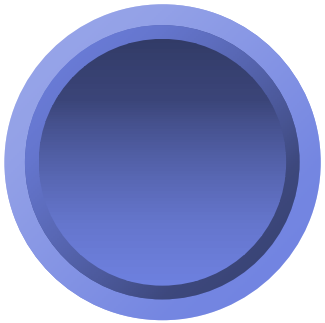


数据独立存储架构



共享数据存储架构

SQL Server本身尚不支持负载均衡，需要借助第三方软件实现。

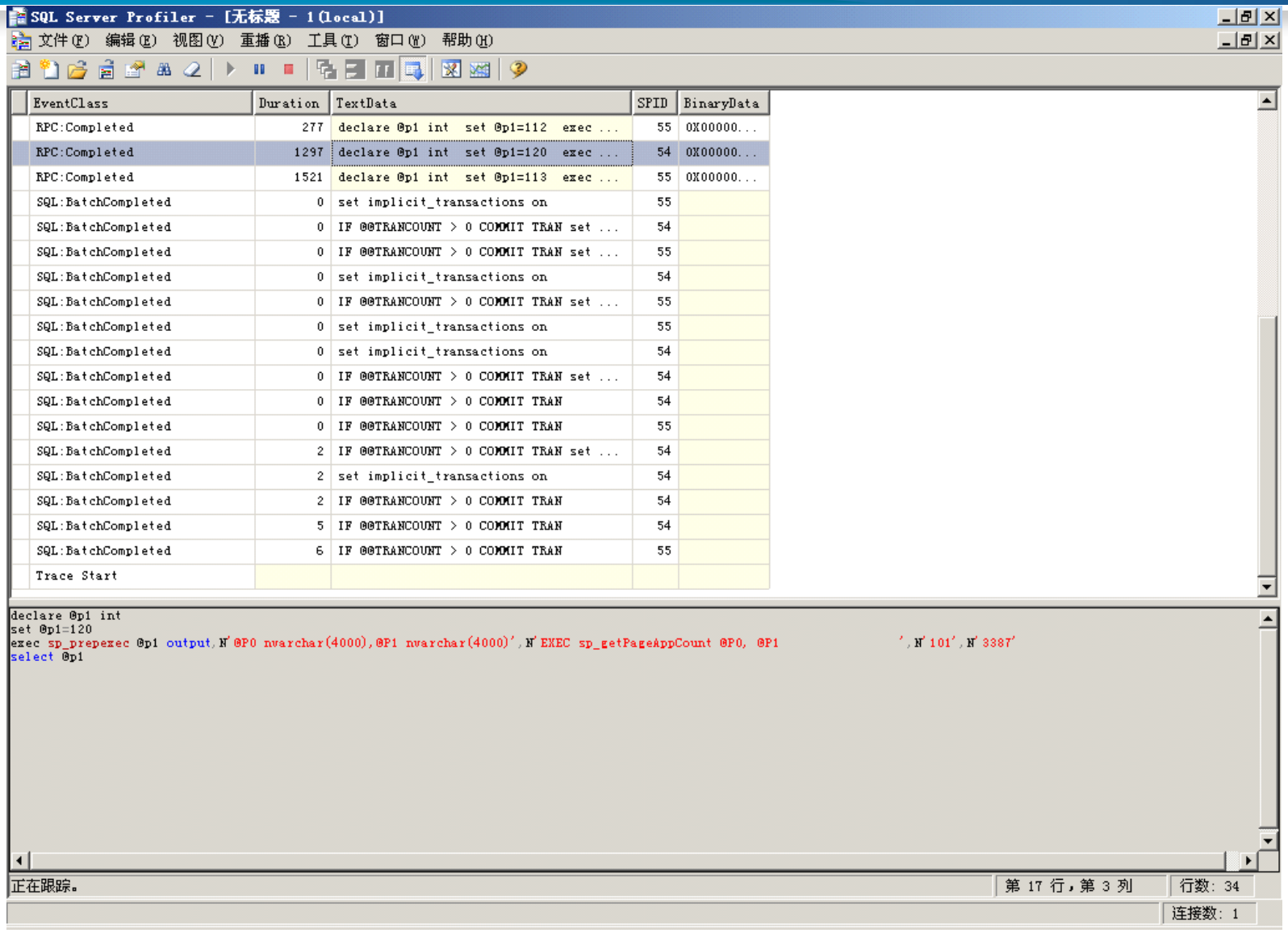


性能监测工具

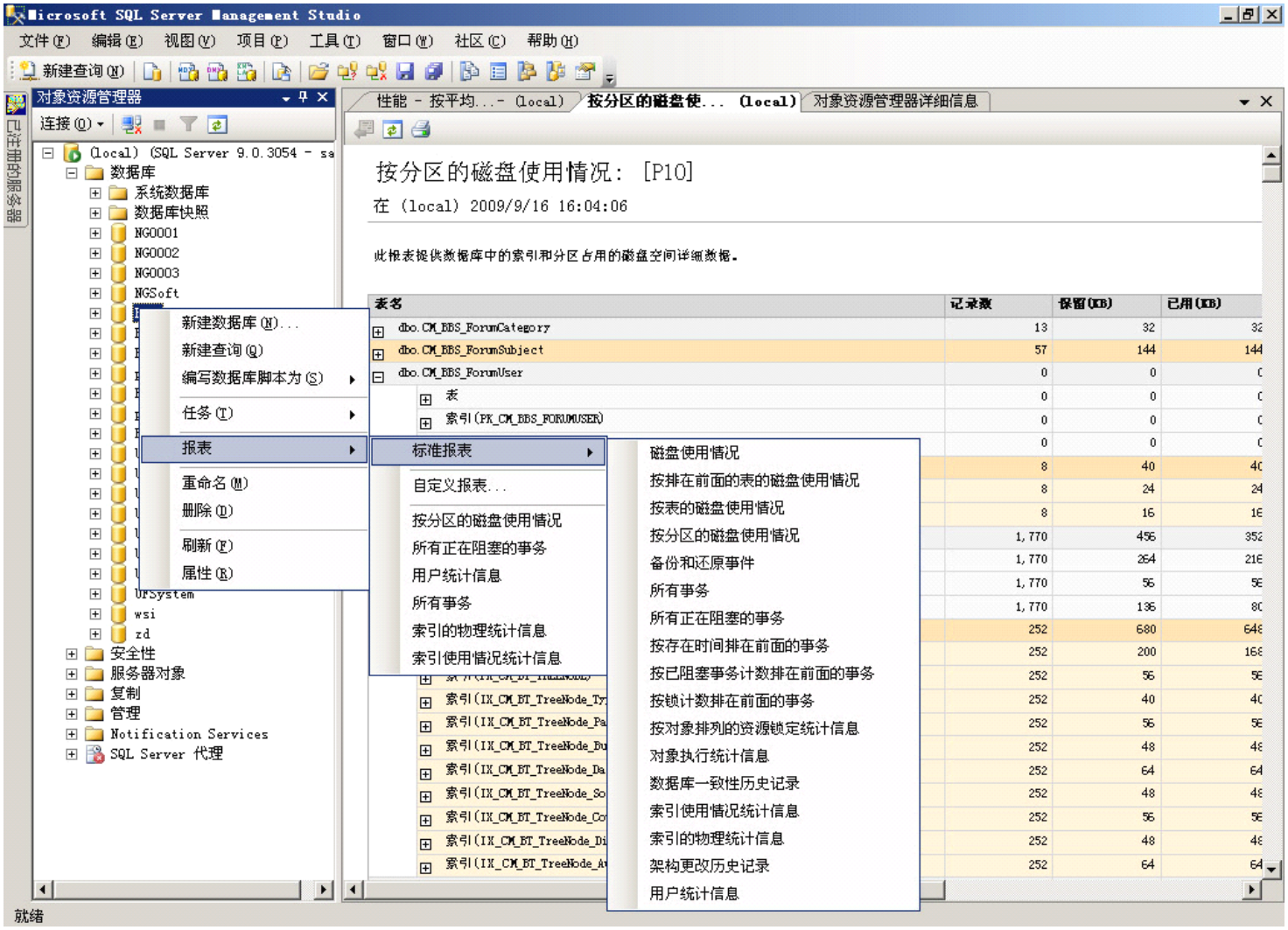
各个层次的监测工具

System/OS	• Windows System Monitor • Alerts (Performance-Based)
SQL Server	• SQL Profiler / SQL Trace • Activity Monitor / SQL Server Agent Alerts • Dynamic Management Views (DMVs)
Query-Level	• Database Engine Tuning Advisor • Query Execution Plans

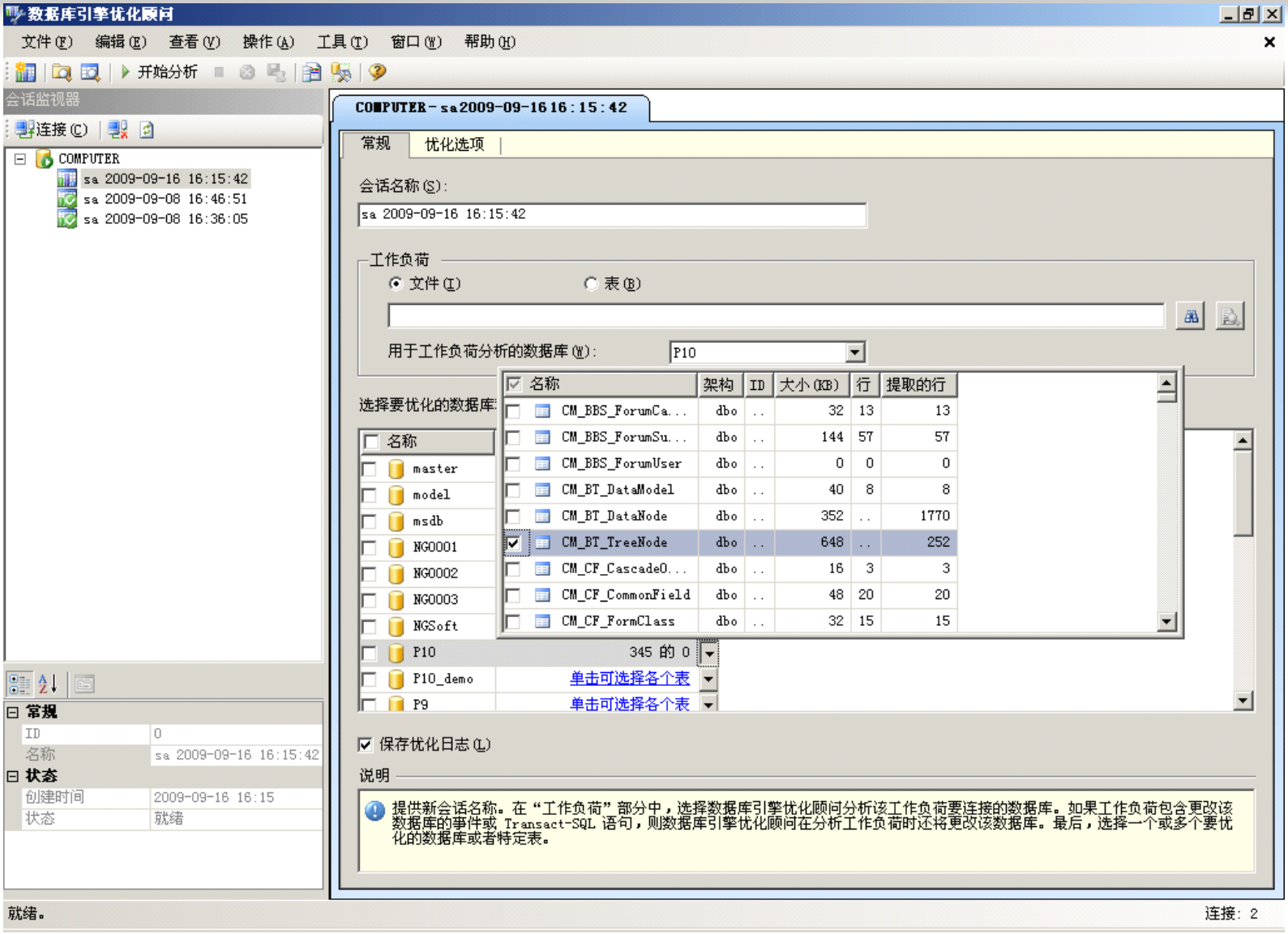
SQL Server Profiler



SQL Server报表

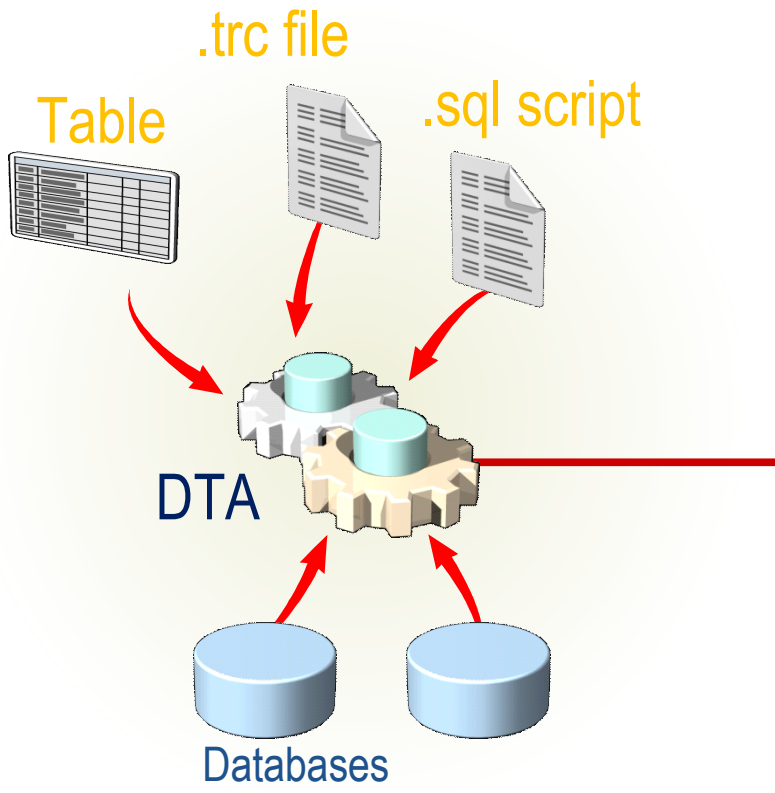


数据库引擎优化顾问(DTA)



DTA示意

分析的来源



分析的结果

Index Recommendations

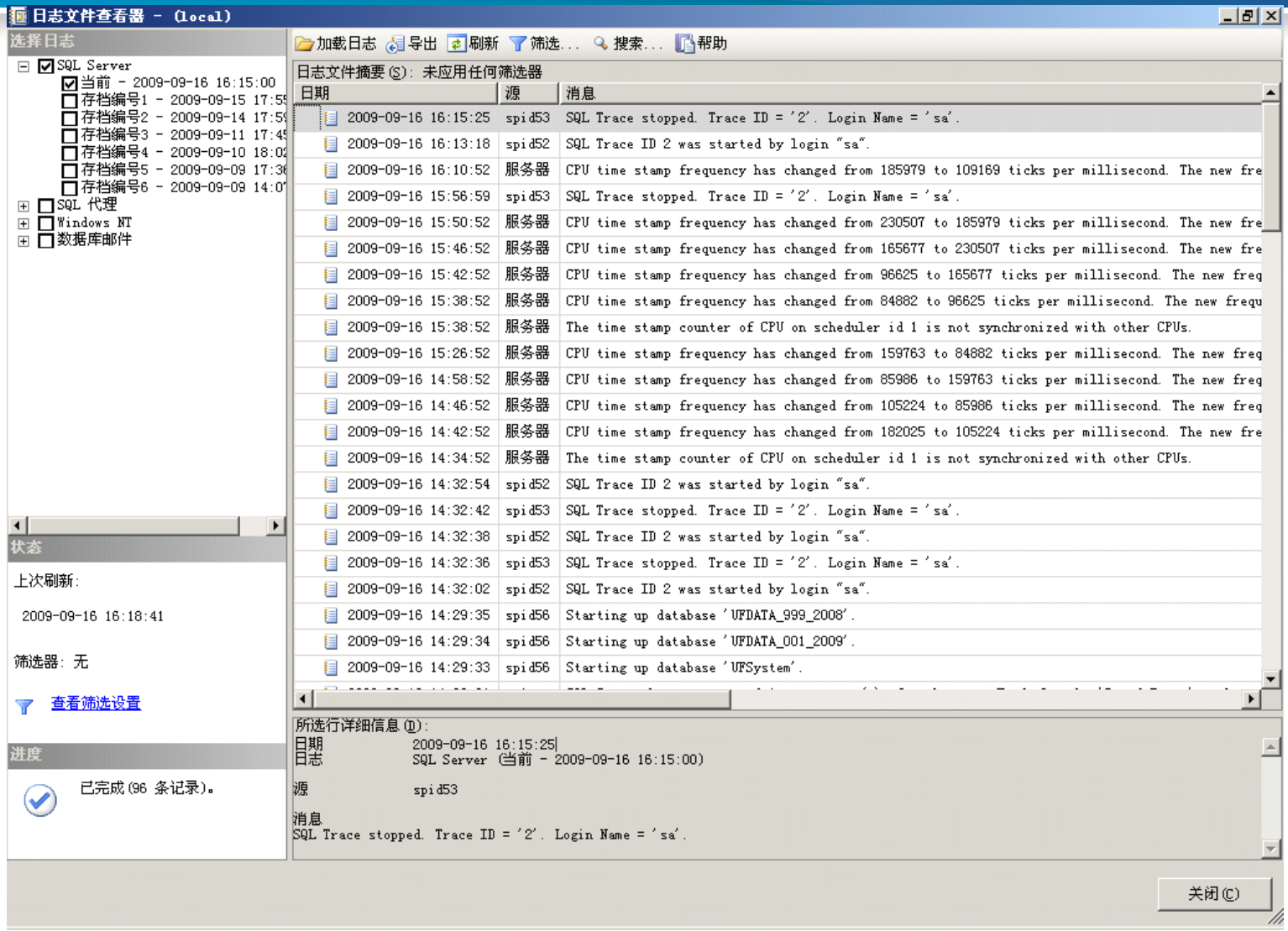
Object Name	Recomme...	Target of Recommendation
[HumanResources].[Department]		PK_Department_DepartmentID
[HumanResources].[Department]		AK_Department_Name
[HumanResources].[Employee]	drop	~~IX_Employee_ShiftID~~
[HumanResources].[Employee]	drop	~~IX_Employee_ManagerID~~
[HumanResources].[Employee]		PK_Employee_EmployeeID
[HumanResources].[Employee]		IX_Employee_AddressID
[HumanResources].[Employee]		AK_Employee_rowguid
[HumanResources].[Employee]		AK_Employee_LoginID
[HumanResources].[Employee]		AK_Employee_NationalIDNumber
[HumanResources].[EmployeeDe...	drop	~~IX_EmployeeDepartmentHistory_Depart...~~
[HumanResources].[EmployeeDe...		PK_EmployeeDepartmentHistory_Emplo...
[HumanResources].[EmployeePay...		PK_EmployeePayHistory_EmployeeID_R...
[HumanResources].[JobCandidate]	drop	~~IX_JobCandidate_EmployeeID~~
[HumanResources].[JobCandidate]		PK_JobCandidate_JobCandidateID

Tuning Reports

Select report: Query detail report

Statement String	Type	Cost of the statement	Cost of the statement with recommended configuration	Weight
BEGIN SELECT * FR...	Select	0.02	0.02	1
SELECT * FROM Hu...	Select	0.07	0.07	1
SELECT Name, Produ...	Select	0.01	0.01	1
UPDATE Production.P...	Insert	0.12	0.12	1
UPDATE Production.P...	Insert	0.12	0.12	1
SELECT Name, ListPr...	Select	0.07	0.07	1
INSERT INTO Sales...	Update	0.05	0.03	1
SELECT P.Name, V.N...	Select	0.08	0.08	1
SELECT * FROM Hu...	Select	0.01	0.01	1
DELETE FROM Sales...	Delete	0.08	0.06	1
SELECT * FROM Pur...	Select	0.13	0.13	1
SELECT Name, SUM(...	Select	0.02	0.02	1
SELECT * FROM Sale...	Select	0.12	0.12	1

SQL Server日志

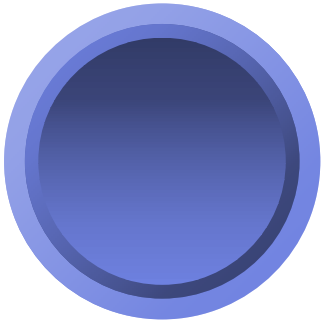


动态管理视图(DMV)
动态管理函数(DMF)

- DMV与DMF是SQL Server 2005的新增功能。它使DBA方便在数据库服务器和数据库实例层面监测系统的运行状况，而不是向以往那样通过系统表来完成这些监测功能。所有的DMV与DMF都存在于master数据中，属于sys schema
- 示例:
  - 数据库引擎
    - Sys.DM_DB_File_Space_Usage
  - 索引
    - Sys.DM_DB_Index_Operational_Stats
    - Sys.DM_DB_Index_Physical_Stats
  - I/O
    - Sys.DM_IO_Pending_IO_Requests
    - Sys.DM_IO_Virtual_File_Stats
  - 数据库镜像
  - 事务

利用DMV观察索引使用情况

```
SELECT
  object_name(a.object_id) AS table_name,
  COALESCE(name, 'object with no clustered index') AS index_name,
  type_desc AS index_type,
  user_seeks,
  user_scans,
  user_lookups,
  user_updates
 FROM sys.dm_db_index_usage_stats a INNER JOIN sys.indexes b
 ON a.index_id = b.index_id
 AND a.object_id = b.object_id
 WHERE database_id = DB_ID('P10')
 AND a.object_id > 1000
```



升级至：

SQL Server 2008

Windows Server 2008

微软的数据平台蓝图

Thank you