CMP

Microsoft SQL Server Planning and Building a High Performance Database

计算机网络基础与应用系列丛书

由经验丰富的专家撰写

内容丰富、通俗易懂

一个可伸缩的、高性能数据库管理系统

(美) Robert D. Schneider 著
李小坚 王涛 马惠 译

规划与建立高性能 SQL Server 6.5 数据库

机械工业出版社
西蒙与舒斯特国际出版公司

计算机网络基础与应用系列丛书

规划与建立高性能 SQL Server 6.5 数据库

R. D. 斯科奈德
（美） Robert D. Schneider 著
李小坚 王 涛 马 惠 译
李旭东 刘炳兴 审校

机 械 工 业 出 版 社
西蒙与舒斯特国际出版公司

Microsoft SQL Server 6.5 版是一个可伸缩的、高性能数据库管理系统，它专为分布式客户/服务器环境而设计。它的内置数据复制功能、强大的管理工具和开放式的系统体系结构为发布物有所值的信息解决方案提供了一个卓越的平台。

本书提供了数据库系统必备的基础和工具性的资料，适用于应用程序开发人员、数据库管理人员、系统分析员、系统管理员，以及对数据库有兴趣的大学生和计算机工程技术人员。对于使用最新 SQL Server 6.5 及以上版本来设计、开发、维护、管理、规划与建立高性能数据库应用系统有很大的参考价值。

本书版权登记号：图字：01-97-0932

图书在版编目(CIP)数据

规划与建立高性能 SQL Server 6.5 数据库/(美)斯科奈德(Schneider, R. D.)著；李小坚等译. -北京：机械工业出版社，1997.11

(计算机网络基础与应用系列丛书)

书名原文：Microsoft SQL Server Planning and Building a High Performance Database

ISBN 7-111-05889-5

Ⅰ.规… Ⅱ.①斯… ②李… Ⅲ.数据库管理系统，SQL Server 6.5 Ⅳ.TP311.13

中国版本图书馆 CIP 数据核字(97)第 22195 号

出 版 人：马九荣(北京市百万庄南街 1 号 邮政编码 100037)

责任编辑：江 颖

北京大地印刷厂印刷·新华书店北京发行所发行

1997 年 11 月第 1 版第 1 次印刷

787mm×1092mm 1/16· 26 印张·622 千字

印 数：0001—6000 册

定 价：43.00 元

译者的话

本书分为16章3大部分。第1章首先介绍了如何建立优化的测试环境，这是建立一个高性能数据库的基准。第2章～第4章为第1部分，介绍了范式、检索、优化器等内容，重点将放在数据库的逻辑设计。第5章～第8章为第2部分，介绍了SQL语言要点、存储过程和触发器等的详细信息。第9章～第16章为第3部分，讨论了高效的数据库管理和数据库设计的应用细节。

本书提供了数据库系统必备的基础和工具性的资料，适用于应用程序开发人员、数据库管理人员、系统分析员、系统管理员，以及对数据库有兴趣的大学生和计算机工程技术人员。对于使用最新SQL Server 6.5及以上版本来设计、开发、维护、管理、开发和建立高性能数据库应用系统有很大的参考价值。本书还提供了大量的例题和程序，为更好地掌握和使用SQL Server有很大的帮助。

全书由李小坚、王涛、马惠三人共同翻译，审校由李小坚、李旭东、刘炳兴完成。其中第1章和第11章～第14章由李小坚翻译，第2章～第8章由王涛翻译，第9章、第10章和第15章、第16章及附录A、附录B由马惠翻译，参加翻译的其他人员还有尺雪丰、蒋琪、陈佩。

译者由于水平有限，很难表达出原著语言的细微之处；加之时间仓促，错误在所难免，望读者批评指正。

译　者

1997年10月

前　言

使用 SQL Server 能快速建造功能强大和高可靠性的数据库应用系统。但仅此一点是不够的，用户还需要应用具有高性能。建立这样高质量的系统需要有一定的经验和技巧，但许多的信息系统专家并没有时间来了解错综复杂的 SQL Server 引擎，通过这本书的学习你将会对建立快速的数据库应用系统有一个透彻的了解。

适用的读者

本书为应用程序开发人员、数据库管理人员、系统分析员、系统管理员使用 SQL Server 6.0 或以上版本设计、开发、维护和管理一个高效的应用程序提供了一个优秀的工具。

要想获益最大，程序员、系统分析员应该熟息以下内容：

- 基本的 SQL 语言

数据库管理员应该了解：

- 关系数据库设计概念
- 中级或高级 SQL 语言
- Microsoft SQL Server 引擎结构

系统管理员应该懂得：

- Microsoft SQL Server 引擎结构
- Microsoft NT 操作系统中级水平

使用本书的益处

- 提高生产力。对于用户来说，如果查询要花费 3min 的时间才能返回数据，是很难提高生产率的。同样要花费 36h 才能建立一个周末报表也不能达到基准线的要求。在这本书中，我们引用了许多特殊的变化可以对你的系统和应用程序进行升级进而提高机构的生产率。
- 增加应用可靠性。通常，一个低效的应用程序也是一个不可靠的应用程序。至少，快速系统要比慢速系统稳定。当提高了系统速度的时候也就意味着提高了系统的可靠性。
- 降低硬件开支。一些机构能够通过连续的硬件升级来获得更好的性能，然而，他们更加希望 IS 成员能利用较少的硬件来做更多的工作。这本书中提供了许多好的建议可以使你的系统利用现存的硬件来获得额外的速度。
- 简化应用。在很多的情况下，最简单的解决方案就是最可靠的解决方案。这本书中的很多暗示会帮助你提高系统的速度同时降低系统的复杂性。
- 更好地使用户满意。在过去，许多的用户只关心系统是否在工作，性能并不是主要的考虑因素，今天的用户则把系统的速度看成是最关键的因素。许多组织是在痛苦的教训中才认识到这一点的。无论设计如何优良，用户对于一个慢的系统是不会满意的。

如何使用本书

这本书分成如下几个主要部分：

1)第 1 章，在这章当中，我们讨论如何设置最有效的优化测试环境。

2)第 1 部分——创建一个优良的数据库。整个的应用都依靠的是有效的数据库设计，在这一部分当中，我们回顾在关系型数据库设计当中一些最为重要的方面。

3)第 2 部分——改善数据库访问。不管是通过直接的 SQL 还是通过存储过程，重要的是能够尽可能有效地访问数据库，在这部分当中，我们描述如何书写高效的 SQL、存储过程和触发器。另外，我们还包括其他一些重要的主题，如事务、并发性和游标。

4)第 3 部分——优化 SQL Server 引擎。如果低效数据库不能正确地调整，即使是最高效的应用也能够产生延迟。在这部分当中，我们将回顾许多的技术，你可以使用它来调整你的数据库使其尽可能的快速运行。

5)附录 A——事例分析。这个附录包括了许多的事例分析，每个事例都具有许多相关的问题。

6)附录 B——SQL Replay 1.10。包括一个强大的工具——SQL Replay 来帮助收集和分析 SQL Server 性能统计信息。SQL Replay 通过对系统信息进行定期的快照来收集数据，信息是从系统表、系统过程、系统锁和 SQL Server 的全局变量中导出并允许我们浏览 SQL Server 过去的活动，找出性能的问题和瓶颈，查看锁的竞争和大量使用的时间段。

不需要一章章地阅读这本书，为了从书中获得最大的益处，仅仅浏览那些与你的特殊条件有关的部分，然而，不管你的条件如何这里仍有一些章节是值得注意的，包括：

- 设置优化测试环境。在没有首先发现一些重要的细节之前就进入提高你的系统性能的工作是个最大的错误，这章覆盖了这些内容的详细资料。
- 关系数据库设计的有关概念。一个好的数据库设计是系统能够良好运行的基础。在这节当中会教会我们在数据库设计时如何实现关系模式，它同样也覆盖了关系模式不适合的特殊形式。
- 检索策略。错误的索引策略能够破坏一个完美的数据库设计。列出了一些直截了当的索引技术，可以使用它来动态的提高系统的吞吐量。
- SQL 技巧。没有 SQL 就不能访问数据，在这章当中我们引用了一些可以改善 SQL 语句的办法。
- 一般技巧。在这章当中包括了许多建议，不管什么样的引擎和使用什么样的工具，它都可以帮助你开发更快更好的应用程序。
- 事例分析。这个附录包括了很多方面的事例分析，花一些时间来了解它：你将得到一个增长经验的好机会。

如何使用本书例题

在本书的每章中都会找到很多的例子，它将帮助你快速地实现我们的建议，每章中的例子并不相互依靠，这将帮助我们集中在特殊的问题和解决方案上。请记住在这本书中的例子可能不完全符合你的数据库设计、应用程序软件或者引配置，但在特殊的问题上它仍可以对你提供帮助。

目　录

第 2 部分 改善数据库访问

第1章　建立优化测试环境

当应用程序运行速度缓慢或者系统似乎失调时，大多数管理员和程序员总是急急忙忙去做优化调试工作，准备甩开膀子大干一场。不幸的是在做实实在在的调试工作之前，必须先做一些可能是繁琐的但又非常重要的工作。如若不建立一个稳定的、准确的测试环境，就很可能面临这样的危险：对造成现行系统运行性能不好的根本原因做出不正确的判断，并且再运用这些错误假设作为整个行动计划的基础，直到后来才发现问题时已为时晚矣。

在这一章里，我们从三方面阐述以下重要问题。

首先介绍测试计划的安排。这是在开始测试前的前题。这个测试计划安排包括配置适当的硬件、操作系统、数据库和应用软件。另外会讨论一些组织化问题，这些是在优化工作刚一开始时应着重注意的。

然后，提出在测试时会用到的一些方法。虽然，在这过程中大多成功的系统测试员和调试者都非常依赖测试计划来指导他们，但在仿效一种新的有前途的技术方法时，应折中运用原则性和创造性作出正确处理。

最后阐明怎样才能把在测试过程中获得的信息转化为现实的性能改进。

虽然这章的主要目的是针对那些已有系统和应用程序的调试人员，但在新开发时，仍可以采用这些建议。事实上，将系统性能视为最重要的开发因素是保证产品系统满足要求的一种途径。

1.1　测试前应遵循的步骤

1.1.1　硬件因素

硬件环境事实上包括几个不同的部件，其中每一部分在整体性能上都起很大的作用，下面分别加以详细说明。

- CPU　理想情况下，产品环境和测试环境应有相同的CPU速度和性能。如果不是这样，究竟该相信哪种数据：是从产品系统中获得的数据，还是从测试系统中获得的数据？多处理系统中经常出现这种问题，因为测试系统只有一个CPU而产品系统可能有多个CPU。如果你打算调试这个平台，那这种区别确实成了一个难题。
- 内存　应保证测试平台与产品系统有相同的内存，这一点很关键。一个运行速度很慢的应用程序，如果增加几兆内存，可能会“奇迹般地”解决问题。如果测试系统和产品系统在内存配置上有很大区别，那么，对要研究的问题应慎重下结论。
- 硬盘驱动器　如果产品系统中使用过时的、慢的硬盘驱动器，而测试系统由于使用最新一代数据存储设备而在速度方面有优势，那么肯定会对机器性能和反应速度得出错误的结论。如果打算测试SQL Server(平台)反应速度，那么在建立测试系统时应与产品系统的磁盘配置相匹配。磁盘大小和分区也对系统整体性能有很大影响。如果使用一个与实际系统很相似的磁盘，那么会得到更加理想的结论。
- 连接(本地和网络)　如果产品应用系统中使用了分布式数据库或在客户/服务器环境中工作，那么在配置测试平台时应考虑这些连接、网络和客户/服务器因素。总的说来网络和客户/服务器体系是很重要的，如果测试时不考虑这些因素，那么很可能得不到

完全准确的产品性能信息。

理想情形是测试系统和产品系统完全相同，但实际上管理员和开发者大都很幸运，有他们的测试机器，并且很少有企业能让一台有生产能力的机器闲置着而不做事。如果你的企业碰到这种情形，尽可能用自己的产品机器来测试，这意味着只有在下班时间或周末才有权使用设备。关于怎样才能获得权力使用产品系统用来测试，请参阅这章的"操作因素"那部分。

1.1.2 软件因素

一旦建立起一个测试系统，下一步是必须获取合适的软件并准确配置它，在这一部分中我们把软件分为下列组成部分。

- 操作系统　优化平台必须和产品系统有相同的操作系统，在 Windows NT 操作系统中，参数的微小差别会对性能产生很大影响，这也许会错误引导你的结论。如果你不知如何控制这些操作系统参数，就要求你的系统管理员去做，尽可能使操作系统参数相同。
- 数据库及工具　尽量保证测试环境中安装与产品系统相同版本的服务器及工具，这一点对测试客户/服务器系统特别重要。即使产品没有客户/服务器特点，也要记住：Microsoft 公司经常改进查询优化功能，能改变数据库驱动而改变系统性能参数。
- 应用　为了保证优化顺利进行应避免使用不同版本的应用程序。有时应用程序中某一行的变化会对性能有很大影响，一定要测试用户埋怨的那个版本，否则可能得出应用软件没有性能问题的结论：你的版本运行得非常好。

1.1.3 数据因素

建立了硬件和软件环境后，下一个应予以考虑的是在测试时要用到的数据，理想情况下测试数据库应与正在开发的或产品数据库相匹配，这就要求有相同数量的表格、索引、视图、存储过程和触发器。

管理人员和开发人员在建立测试数据库时易犯的一个常见错误是使用一个过小的数据集合。如果测试数据库不够大，那么很可能对实际系统性能作出错误的估计，不管基本数据库设计是好是坏，在一个小型数据库上，即使一个非常糟糕的查询执行速度都很快。

如果有可能，从 SQL Enterprise Manager 中使用 Database/Object Transfer 菜单选项，拷贝一份完全一样的产品数据库到测试平台。如果产品数据库太大或者测试系统没有足够大的存储空间，测试数据库仍必须带有与产品数据库相同多的表格、索引、视图、触发器和存储过程。使用产品数据库中 SCHEMA CREATION UTILITY 能得到 SQL，它们对建立一个相同的测试数据库是必要的。如果要测试一个使用两个表格的查询，那么只须从产品系统中拷贝这两个表格到测试系统。

一旦建立了测试数据库并且输入了合适量的数据别忘了运行像在产品数据库中一样要 UPDATE STATISTICS。

1.1.4 制定测试计划

在开始实验之前制定测试计划并不是绝对必要的，当然一个精心规划好的测试计划由于提供了清晰的方向使得工作更加容易；对于一个多人组成的测试小组这一点特别重要，没有一个测试计划，小组成员可能做重复工作，也有可能破坏测试结果的整体性，一旦完成测试，测试计划也可作为总结报告的基础。

没有必要太多时间花在计划上，但在高层必须是明确的；在评价性能时除了其他用户或开发人员提供的帮助外，希望有计划可循。

例 1.1　测试计划：

……

……

……

测试#：　17

日期：　03/09/97

起始时间：　08:00

结束时间：　11:00

平台：　产品平台

操作：　在 call_tracking 表格上增加/改变索引

特殊需要：　在增加/改变索引时需暂时给表格上锁

小组成员：　Nicole——负责人

Jed——助手

Li——助手

测试状态：　(未完成)

测试结果：　(未完成)

测试#：　18

日期：　03/09/97

起始时间：　12:15

结束时间：　16:30

平台：　产品平台

操作：　增加存储缓冲器数目并测试 SQL Server 系统性能，
在其余时间内观察系统性能统计量

特殊需要：　需要重新启动系统——这会导致数据库系统停机两分钟左右

小组成员：　Lynn——负责人

Karl——助手

测试状态：　(未完成)

测试结果：　(未完成)

测试#：　19

日期：　03/10/97

起始时间：　08:00

结束时间：　17:30

平台：　开发平台

操作：　将程序 cc01288 和 cc03432 从现行的可重复读写层次改为
未交付隔离层次，并在修改前后监视应用系统的响应时间

特殊需要：　没有

小组成员：　Miguel——负责人
　　Lynn——助手
测试状态：　（未完成）
测试结果：　（未完成）

测试#：　20
日期：　03/11/97
起始时间：　8:00
结束时间：　17:30
平台：　产品平台
操作：　如果测试19提高了产品性能，就把程序cc01288和cc03432增加到系统中
特殊需要：　需要用户输入相应响应时间
小组成员：　Miguel——负责人
　　Meyer——助手
　　Sandra——助手
测试状态：　（未完成）
测试结果：　（未完成）
……
……
……

甚至可以把测试计划存入数据库中，然后用SQL获得有用的查询。

1.1.5 操作因素

建立了硬件、软件、数据库，并且制定了好的测试计划后最大的困难在于所要做的调试工作怎样才能得到操作上的支持？

操作上的问题一般是优化应用系统中最容易让人沮丧的，因为管理部门不愿意让你得到所需要的全部或部分资料，甚至连用户也不愿意帮你查出运行问题，虽然他们从你的发现中获利最多，而应用软件的设计人员和开发人员会对你的测试可能会发现的一些问题有点紧张不安。不幸的是，只有要求管理部门，用户和开发人员的合作才能对系统有正确的估计。在开始测试前你有权询问下面每一个问题：

- 系统可用率　如果没有专门的测试平台而必须在产品系统上做实验，这也许意味着在深夜或周末工作，但这也比根本没有系统好。如果管理部门不愿授予你权力，就可以指出：如果你能使用那些反应迟缓的平台，就可以更快解决他们的运行问题。如果产品系统一天24小时，一星期7天连续运转，那么不幸的是很难挤出一点“安全”的工作时间。
- 实际系统负载　我们在前面已经明白，最好是在产品系统环境中测试，但是只有在下班时间才有权力使用产品系统一般是不够的，因为无论你多么努力，如果你使用的是一个相对空闲的系统，就不可能再现运行问题。例如，网络工作负载在一天之内变化很大，在凌晨两点时测试，运行良好，而12小时后速度也许会很慢，这点强调了尽可能在现实条

件下测试的重要性。

- 用户输入和支持　要尽可能使协调测试科学性，如果用户抱怨响应太慢，很明智的做法是要求他们举出特定的例子，当然如果在他们工作时坐在他们旁边那是更理想的了。如果是当场发现一个问题而不是三天之后再听说起，那有更好的机会来分析它。一旦着手并且(理想情况下)解决了这个问题，仍需从用户那里得到反馈信息和帮助，否则可能永远不清楚修改是否起作用。
- 信息系统员工的支持　除非信息部门只有你一人，否则在优化新开发的或已有的应用系统时很有可能要与你的伙计一块合作。作为一个管理员必须与设计系统的程序员，分析员保持良好关系；作为一个程序员或分析员需要数据库管理员的合作。不幸的是很大一部分协调工作由于政策失调或大家指手划脚而陷入困境。因此不管你的工作责任如何，重要的是要敏感地、有策略地做调查研究、提建议。(几乎)没有人有意设计运行缓慢的应用系统或数据库系统。另外，在负荷加重和参数变化的情况下，一个设计得很好的系统也会慢下来。记住有很多办法达到设计要求：在设计过程中你的伙计做出的决定也许在当时是能满足要求的正确决定。没有人会认为非建设性批评行得通，在工作过程中要牢记这一点。

1.2　测试过程中应遵循的步骤

如果在测试过程中发现一种完全不同的、可能是很好的途径，并且你认为它会导致更进一步的研究，那么应该怎么办?完全抛弃精心规划的计划?还是只写下你的想法而回到原计划上?如果意识到计划不能完全达到要求那又怎么办?

所有上述问题的实用答案是在做测试实验时灵活掌握。别害怕屈从于一时冲动，有时会突然找到最有希望的优化方案。如果由于太忙而没有记下这种方案而让它溜走，那是不应该的。

另一方面，别因为测试计划部分过时而抛弃它，相反为什么不把新意见吸收到已有的方案中去? 例如，仍使用前面讲到的那个例子，假设刚开始实验 19。

例 1.2

测试#：	19
日期：	03/10/97
起始时间：	08:00
结束时间：	17:30
平台：	开发平台
操作：	将程序 cc01288 和 cc03432 从现行的可重复读写层次改为未交付隔离层次，并在修改前后监视应用系统的响应时间
特殊需要：	没有
小组成员：	Miguel——负责人
	Lynn——助手
测试状态：	完成
测试结果：	性能没有提高，但发现了一个与索引有关的问题，见测试 19.1。

你知道,像刚开始设想的一样,有两个程序的问题不是交易隔离层次。相反你意识到每个程序中有几个 SELECT 语句使用了 ORDER BY 子句,通过检查 SHOWPLAN 输出会发现 ORDER BY 子句中所给出的列上没有索引。一旦加上索引每个程序运行时间只需原来的十五分之一。怎样才能记住这个信息呢? 一种方法是增加一个新的测试来反应附加实验的结果。

例 1.3

测试#:	19.1
日期:	03/10/97
起始时间:	08:00
结束时间:	17:30
平台:	开发平台
操作:	给 LEADS 表格在域 TERRITORY_NUMBER 和域 ZIP_CODE 增加复合索引,并测试对程序 cc01288 和 cc03432 的影响
特殊需要:	没有
小组成员:	Miguel——负责人
	Lynn——助手
测试状态:	完成
测试结果:	程序 cc01288 和 cc03432 运行得比原来快

这种方法使你不用该变 DISCIPLINED 方法而又采纳了创造性建议。

1.3 测试后应遵循的步骤

1.3.1 正式评估

一旦完成测试并且(希望如此)证实了你的假设,下一步就应该把发现结果用文件表示出来;即使知道每个假设是完全错了,这步也是必要的。由于下面几个原因把结论写成报告显得很重要:

- 更好的交流　如果在优化过程得到其他人的帮助,那么给他们提供你的测试结论是一种公平交易。可以口头上给他们讲,但写下结果可以减少混淆、误解的可能性。如果建议包括数据库、系统配置和修改应用程序等各方面,而且又希望别人认真对待这些意见并且相应作出改变,那么一定要把它们写下来。即使你不需要任何用户和开发者的帮助,也不提出任何改进意见,你的管理人员有可能想知道你了解些什么。
- 将来的测试　除非这轮测试解决了所有运行问题,否则仍有可能在将来某个时刻需要再来一次测试,协调应用系统和环境。事实上,即使解决了每个问题,定时地监视系统的响应是很明智的,在后来的评价中将会发现早期的测试计划和结论的书写文档是非常有用的工具。
- 帮助新来的员工　现在人员调整的速度实际上使得你必须在将来某一天对新来的员工解释测试方法和步骤,现在记下结果会减少将来必须花费的时间,并且提高准确性。

1.3.2 确定下一步骤

即使你认为找出了每个运行瓶颈所在,在完成测试实验后不要急于立刻做大幅度的修改,