第 16 章

组织索引设计过程

- 从相关人员的职责角度看如何组织索引设计过程
- 使用计算机辅助式的索引设计工具
- 它们的目标及优缺点
- 对于基础评估过程的总结
- 设计出色索引的 9 个步骤

简介

在数据库发展的早期，索引设计在专家组中被认为是一种核心职责。当时的应用非常简单，只需要一位数据库专家就能够熟悉所有的处理需求。而在今天，这几乎不可能了。

现今，每位应用程序开发者在编写一个新的 SQL 语句时都会对现有的索引进行评估，至少会用基础问题来进行评估，理想情况下，他们还应使用 QUBE 来检查最差输入下的性能。若性能不可接受，那么他们将考虑进行必要的改进。如有必要，可将该语句连同建议（譬如向一个索引上加一个列，或创建一个新索引）一起向数据库专家征求意见。最终的决策可能是集中的，至少在公司层面被综合应用。专家是评估并控制整体成本及副作用的最佳人选。

当然，将处理索引相关问题的职责委派给开发人员才能取得最好的效果。然而，许多公司认为所需的培训和指导（约 1 至 2 周）是一笔过于庞大的开销。

在大公司里，一个合理的折中方案是聘用 50/50 的专家，他们会将 50% 的时间用于应用程序开发，另外 50%的时间用于协助同事进行索引评估及其他性能问题的处理（比如根据 EXPLAIN 的输出内容解决某个优化器问题）。 290
经验显示，为每 5 至 10 位应用开发者配一名 50/50 专家的方式效果很好。

索引设计需要同时掌握技术技能以及应用系统知识。相比让数据库专家熟悉应用系统的细节而言，教会开发人员索引技能是更容易的。

计算机辅助式索引设计

令人意外的是，索引设计工具直至 20 世纪 90 年代才在市面上出现。第一批工具是基于规则的，而许多当前的工具则是使用优化器来评估各个可供选择的索引。这些工具的目的是找出能将系统在一个特定负载下运行的平均响应时间降至最低的索引集（见图 16.1）。

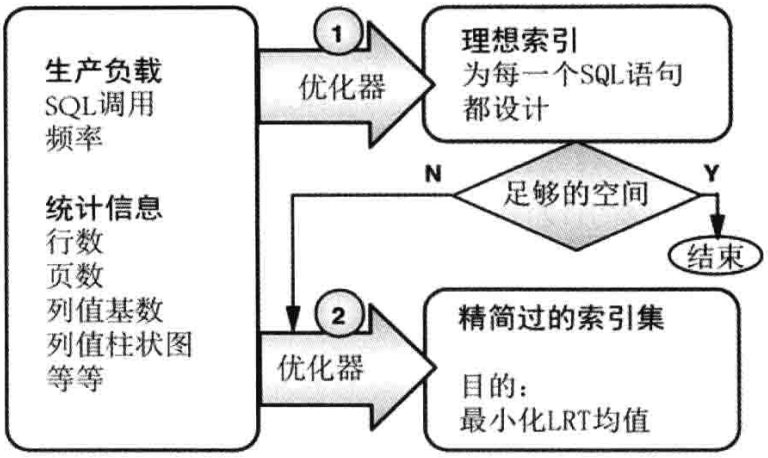
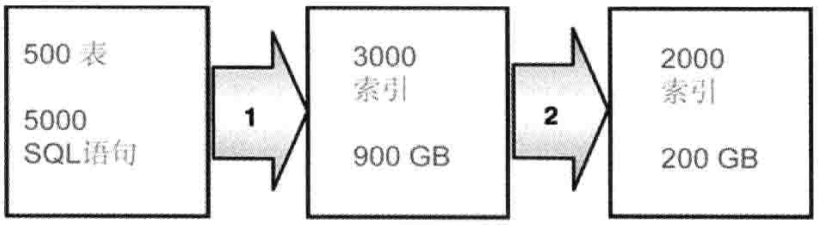


图 16.1　计算机辅助式索引设计

通常情况下，为一个给定的 SELECT 语句找出最佳索引是很快的，但这可能会导致表上有过多的索引。于是，工具必须从中找出价值最低的索引。显然，当有许多 SELECT 语句时，这是一个非常耗时的任务。想象一下，为了评估删除一个索引带来的影响，工具需要进行多少次成本估算。据工具的开发者说，这是一个庞大的搜索范围。有必要使用启发式的快捷方式将执行时间控制在可接受的范围内。当然，这会影响目标的完成质量，如图 16.2 所述。另外，最佳索引集并不一定必须是理想索引集的子集。



方案质量	多半是好的	多变的
执行时间	多半是可接受的	非常长

图 16.2　精简索引集可能是极度耗时的

现今的许多工具都与象棋计算机类似——可以限定计算时间的上限。于是，最终产出的结果即为给定时间内所能得出的最好方案。

目前，一个实际的方法似乎是每次只向索引设计工具提供一个 SELECT 语句（参见图 16.3）。工具于是便会生成出所能达到的最佳索引，即使是对于一个涉及多表连接的查询语句也是如此——只要我们确保使用了最差输入值而非绑定变量。当然，这不是一个可有可无的事情——请不要忘了前文提到过的过滤因子缺陷。我们接下来要做的是基于每张表所能容许的索引数量来做出一个折中的方案，并为每张表挑选所需的索引，可能还会需要对一些索引进行合并。 291

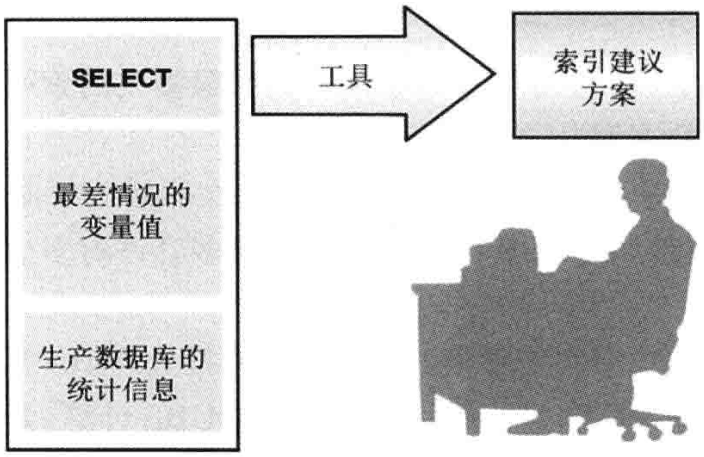


图 16.3　切实的方法

索引设计的基本方法在图 16.4 中进行了小结。

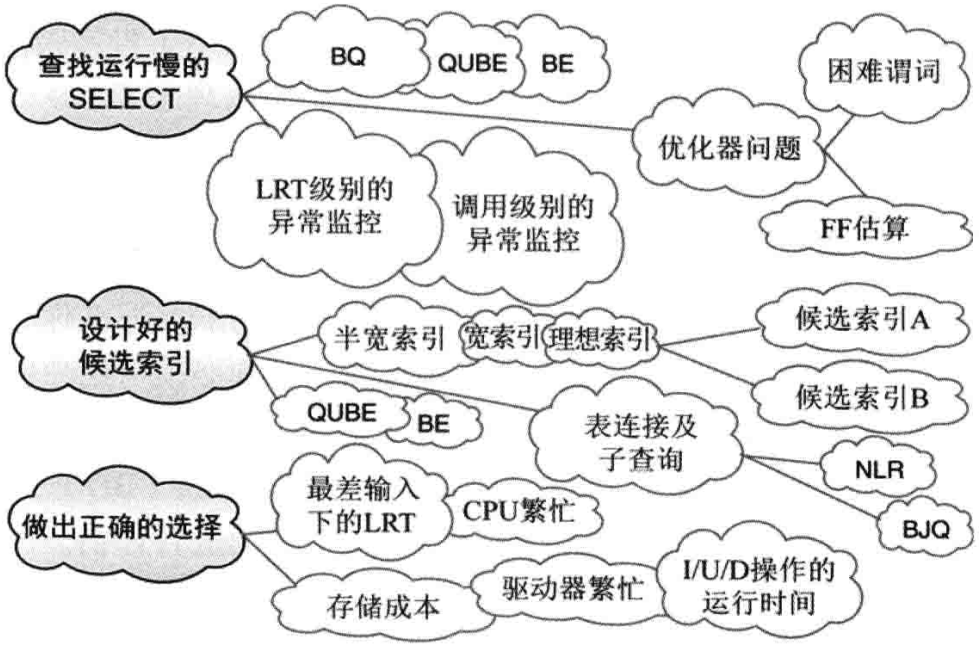


图 16.4　索引设计基本方法总结

292 设计出色索引的 9 个步骤

1. 当表结构第 1 版设计（主键、外键、表行顺序）完成时，就开始创建第 0 版的索引：主键索引、外键索引及候选键索引（如果有的话）。
2. 对第 1 版表结构设计的性能表现进行检查：使用 QUBE 评估一些重负载事务和批处理程序在理想索引下的响应时间。若评估结果无法满足要求，则将那些具有 1：1 或 1：C（1 对 0 或 1）关系的表进行合并，同时将冗余数据添加至有 1：M（一对多）关系的依赖表中。
3. 当表结构基本稳定后，你可以开始添加一些明显需要的索引——基于对应用系统的理解。
4. 若一个表的变化频率很高（如每秒有大于 50 次的插入、更新或删除），那么你应该用 QUBE 评估一下该表最多容许有多少个索引。
5. 当知道一个程序的数据库处理模式（事务型或批处理型）后，就需要用最新的数据库版本进行最坏输入下的 QUBE 计算。若评估出一个事务的本地响应时间超出了应用的警戒值（如 2 s），则
293 表明当前的数据库版本无法满足该程序。对于一个批处理程序而言，对响应延时的接受度必须针对具体情况逐个评估。不过，为了避免长时间锁等待，相邻两个事务提交点之间耗时的告警阈值应该与本地响应时间的告警阈值相同。一旦超出了告警阈值，你就应当对索引进行改进（半宽索引、宽索引或理想索引）。若在使用了理想索引的情况下评估结果（QUBE）仍不令人满意，或者所需的索引数量超出了第 4 步中评估的表上所容许的最大索引数，那么你需要根据第 15 章中所讨论的问题对这些慢查询进行更精确的评估。若评估出的响应时间仍旧过长，那么你必须像第 2 步中那样修改表的设计。最差情况下，你必须与用户协商调整需求，或者与管理人员协商调整硬件配置。
6. SQL 语句被编写后，开发人员就应使用基础问题（BQ），或者如果可行的话，用基础连接问题（BJQ）对其进行评估。
7. 当应用程序发布至生产环境后，有必要进行一次快速的 EXPLAIN 检查：对所有引起全表扫描或全索引扫描的 SQL 调用进行分析。这一检查过程也许能发现不合适的索引或优化器问题。

8. 当生产系统正式投入使用后，需要针对首个高峰时段生成一个 LRT 级别的异常报告（尖刺报告或类似的报告）。若一个长响应时间问题并非由排队或优化器问题引起，那么你应该用第 5 步中的方法进行处理。
9. 至少每周生成一个 LRT 级别的异常报告。

变体 1

若没有时间对所有的程序都进行 QUBE 计算，那么可以仅对那些可能出问题的程序进行评估（例如，浏览类事务及大规模批量处理程序）。

变体 2

若无法实现 LRT 级别的异常报告，那么可以用一个较低的阈值（例如，100 ms 的响应时间）生成一个调用级别的异常报告。另外，若有运行明显较慢的事务未被调用级别的异常报告捕获，那么也应当用 QUBE 对其进行分析。

备注

在第 2 步中，一个表结构设计图就够用了，并不需要完整的 CREATE TABLE 语句。由此，一旦第一个程序被定义完成，索引设计就可以开始了。随后定义的程序可能需要额外追加表和索引列。

备注 2

294

在第 2 步至第 5 步中，对于数据库处理的描述必须包含相应 SQL 语句的相关信息，不过 SQL 的具体结构（如连接或子查询）是不需要的。在这个阶段，我们应当假定优化器会选择（或能被强制指定选择）使用我们所设想的访问路径。

参考文献

1. Peter Gulutzan and Trudy Pelzer, *SQL Performance Tuning*, Addison-Wesley, Reading, MA, 2002.
2. Mark Gurry, *Oracle SQL Tuning Pocket Reference*, O'Reilly, Sebastopol, CA, 2002.
3. Sharon Bjeletch, Greg Mable, Vipul Minocha, and Kevin Viers, *Microsoft SQL Server* 7.0 *Unleashed*, SAMS Publishing, Indianapolis, IN, 1999.
4. C. J. Date with Colin J. White, *A Guide to DB2*, Addison-Wesley, Reading, MA, 1990.
5. Marianne Winslett, Jim Gray Speaks Out, *www.acm.org/sigmod/record/issues/0303/Gray_SIGMOD_Interview_Final.*
6. Amy Anderson and Michael Cain, *www-1.ibm.com/servers/enable/site/bi/strategy/index.html.*
7. Dennis Sasha and Philippe Bonnet, *Database Tuning—Principles, Experiments, and Troubleshooting Techniques*, Morgan Kaufmann Publishers, San Francisco, CA, 2003.
8. Ari Hovi, *Data Warehousing*, Tietovarastotekniikka, Talentum, Helsinki, 1997.
9. Donald K. Burleson, *www.dba-oracle.com/art_dbazine_idx_rebuild.htm.*

术语表

索引设计方法中所涉及的术语

以下是我们在讨论索引设计方法时所使用到的术语。这些术语并不是现今人们正在使用的任何一个数据库管理系统的官方术语。

设计最佳索引的算法 首先设计一个宽索引（第三颗星）以使所需扫描的索引片尽可能地窄（第一颗星）。若该索引能避免排序（第二颗星），那么这个索引即为三星索引。若无法避免排序，那么它将只是二星索引，即牺牲了第二颗星。于是，我们便需要设计另外一个能避免排序的候选索引，即拥有第二颗星，牺牲第一颗星。这两个二星索引中的一个最终将成为相应 SELECT 语句的最佳索引。

辅助式随机读 本书中用于指代自动跳跃式扫描、列表预读及数据块预读的术语。

辅助式顺序读 本书中用于指代将一个游标拆分为多个范围谓词游标，每个游标分别扫描一个索引片。当多个处理器和磁盘驱动器可用时，响应时间将相应地降低。

BJQ，基础连接问题 是否存在或计划设计一个包含所有本地谓词列的索引（包含语句涉及的所有表的本地谓词）？

BQ，基础问题 是否存在或计划设计一个包含所有 WHERE 语句涉及的列的索引（一个半宽索引）？

最佳索引 为一个 SELECT 语句所能设计出的最好的索引。这个索引可能拥有三颗星，即理想索引；也可能只有两颗星，由于 ORDER BY 和范围谓词同时存在而不得不牺牲第一颗星或第二颗星。

调用级别的异常监控 生成异常报告，该报告展示了某监视时段中运行得最慢的 SQL 调用。如果需要从一个有大量不同 SQL 语句的慢程序中找出那些运行得很慢的 SQL 调用，这个报告是非常重要的。

候选索引 A 和 B 最佳索引设计算法中所使用到的两个索引。

CPU 系数 用于计算随机访问、顺序访问、FETCH 和排序的 CPU 时间的值。

CQUBE，CPU 时间的快速上限估算　快速、粗略地估算 SQL 的 CPU 时间需要用到四个变量：TR、TS、F 和 RS。

298 **问题制造者**　一个独占了资源的事务，可能是由不合适的索引导致的。

DB2 for LUW　一个运行在 Linux、UNIX 和 Windows 平台上的 DBMS。

困难谓词　无法参与定义索引片的谓词——即该谓词无法成为匹配谓词；有时我们称此类谓词不可索引。

宽索引　一个包含了 SELECT 语句所涉及的所有字段的索引，在使用这类索引的情况下，不需要进行表访问。

过滤因子缺陷　指这样一种场景：性能最差的情况并非发生在过滤因子值最高时，可能是由于访问路径中没有排序，一旦取得第一屏数据就返回，且所有的谓词列都未参与定义索引片。

第一颗星　SELECT 语句所需的索引行都是相邻的，或至少尽可能地相互靠近。这颗星尽可能地将所需扫描的索引片的厚度降至最低。

理想索引　拥有三颗星的索引。

索引片　索引被扫描的部分。索引片的厚度会影响对表的同步读的数量。

LRT 级别的异常监控　生成异常报告，该报告展示了本地响应时间或 SQL 运行耗时特别长的操作型事务。

NLR，本地行的数量　在最大过滤因子条件下，使用本地谓词筛选后剩余的本地行数。该指标被用于预测连接的最佳表访问顺序——将具有最低 NLR 的表作为最外层表。

QUBE，快速上限估算　一个只使用两个变量（TR 和 TS）的简易的本地响应时间评估方法，通常用于在早期发现可能运行得很慢的访问路径。若估算所使用的最差情况的过滤因子与真实值相近，那么就能够揭示索引及表设计相关的性能问题。从定义可以看出 QUBE 是悲观的，它有时会做出错误的警告。

确实困难的谓词　无法参与索引过滤过程的谓词。我们因此需要访问索引片上的每一个表行来匹配谓词，即便谓词列已经被拷贝至索引上也是如此。

第二颗星　索引行的顺序满足 SELECT 语句中 ORDER BY 的要求。满足这颗星的查询不需要对结果集进行排序。

半宽索引　一个包含 WHERE 子句中所有列的索引，使用半宽索引将使得访问路径仅在必要时才访问表。

尖刺　一个本地响应时间或 SQL 运行时间特别长的操作型事务。

第三颗星　包含 SELECT 语句中所有列的索引。满足这颗星意味着避免了表访问——访问路径只需访问索引，第三颗星通常是最重要的一颗星。

299 **三星索引**　一个针对给定 SELECT 语句的理想索引。

访问 DBMS 读取一个索引行或表行的成本。如果 DBMS 扫描一个索引或表的片段（被读取的表在物理上是相邻的），读取片段的第一行将引起一次随机访问，读取剩余的行将引起每行一次的顺序访问。一次索引访问和一次表访问的成本相同。

调优的潜在空间 计划中的索引改进所能实现的本地响应时间提升上限。

受害者 由于需要等待资源而被问题制造者所影响的事务。

通用术语

访问路径 优化器为了获取某个 SQL 语句的结果集所使用的方法。对于单表 SELECT 而言，这是指以某种方式使用某个索引或者进行全表扫描；对于连接查询而言，这是指在前者的基础上增加表的访问顺序和连接方式。

异步读 在前一组页还在处理中时就开始下一组页的读取，I/O 时间和处理时间之间可能有很大一部分的重叠，理想情况下，这些页的异步 I/O 在被请求处理之前就已经完成了。这一过程被称为预读。

位图索引 用于替代 B 树索引，索引列的每一个值都有一个位向量。适用于基数很小且变更频率极低的列。位图索引能使某类有复杂 WHERE 子句的查询运行得很快，尤其是在数据仓库系统中。有些 DBMS 产品并不支持此类索引。

块 参见术语“页”的解释。

布尔谓词 如果当一个谓词被校验为假时就能排除一行，那么该谓词就被称为布尔谓词（BT），否则它就是非布尔的。非 BT 谓词可能使得 WHERE 子句对于优化器而言太困难了——访问路径不是最优的。若一个 WHERE 子句中没有 OR 操作符，则所有的谓词都是 BT 的。

B 树索引 这是最常见的索引类型。索引上的列（通常）是从单个表中拷贝的。索引的最底层（叶子页）包含了指向每一个表行的指针。叶子页层拥有其自己的索引树，该树的最顶层被称为根页。

缓冲池 计算机内存的一个区域，从磁盘上读取的索引或表页被存放在这里。缓冲池可能被划分为多个子池，并将其分配给单独的索引和表。缓冲池的管理者试图将频繁使用的数据保留在池中以避免从磁盘上进行额外的读取。

聚集索引 在 SQL Server 中是指一个包含表行的索引，在 DB2 中是指任何一个索引行顺序与表行顺序相同或计划相同的索引。

聚簇索引 这类索引使得 DBMS 在向表中添加记录时，将新记录添加至由聚簇索引键所定义的主页（或接近主页的页）上。一张表上只能有一个聚

簇索引。有些 DBMS 产品并不支持聚簇索引，但我们可以通过对表进行重组或重新加载来使表行按照所需的顺序存储。

覆盖索引　在 SQL Server 中是指一个包含了 SELECT 语句所涉及的所有列的索引，它能够避免表访问。这与 SQL Server 中的术语表查询相对，表查询是指访问路径使用了索引但仍需从表中读取记录。

300

CPU 缓存　CPU 芯片的高速内存，用于存储运行最频繁的程序指令和数据。

游标　嵌入式 SQL 中的一个结构。用于通过 FETCH 调用向应用程序传递结果集，每次传递一行。

数据库　逻辑上相关的数据，一个 DBMS 相关的表的集合。

按数据分区的二级索引　将超大索引进行分区，每个表分区对应一个索引分区，从而降低不可用率。

数据块预读　Oracle 中用于表达这样一个过程的术语：从索引片上收集多个指针，从而使得能够发起多重随机 I/O 来并行读取表行。

DBMS，数据库管理系统　用于支持、管理及控制所有数据库的使用行为的软件。现在网络和分层系统已经几乎被关系型系统所取代了。

数据仓库　一个提供持续的时间相关数据的业务环境，用于决策支持。被设计用来支持大量各种类型的不可预测的查询和报告。通常这些数据是从多个操作型系统加载过来并转换成统一格式的。它能够支持趋势分析，比如每月销售情况的对比。

默认值　在没有特别给定值的情况下 DBMS 所使用的假定值。

反范式化　向一个表中添加冗余数据，这在数据仓库类环境中非常常见，在操作型数据库中有时也需要使用这一方式以提升 SELECT 语句的性能。

磁盘驱动器　一个存储设备，该设备能够在同一时间支持一次读取或一次写入操作。

执行计划　DBMS 用于描述所使用的访问路径的输出信息。

事实表　包含详细的事务数据，如包括销售数据、价格或优惠之类的销售或支付条目。这些数据在维度数据的辅助下被汇总或分组。

宽表　一张包含其他表中部分字段或由几张表合并而成的表。

FETCH　在游标操作中用于提取数据的 SQL 调用，每次提取一行记录。

过滤因子　被用来定义谓词的选择性——满足谓词条件的记录数占表行总数的比例。该值取决于列值的分布情况。当评估一个索引是否合适时，最差情况的过滤因子比平均的过滤因子更重要。

外键　一个指向另一张表或本表的主键的列或组合列。外键上通常有引用完整性约束来保证数据的完整性。

空闲空间　为了支持新添加到表及索引中的行，当页被加载或重组时，页中

预留出一定比例的空闲空间。

哈希连接 一种连接方式。这种连接方式首先对一张表上的连接列进行哈希计算，再将其存储在一张临时表中（本地谓词已参与数据过滤）；对于另一张表上满足本地谓词的记录，则通过其哈希值对临时表中的记录进行校验。 301

提示 在 SQL 调用或绑定选项中所指定的规范，用以影响优化器，提示的语法与具体的产品相关。我们应当只在优化器由于不合理的成本估算而无法选择最佳访问路径时才使用这一技术。

绑定变量 WHERE 语句中所使用的程序变量，如 WHERE SALARY>:SALARY 中的:SALARY。

索引匹配 通过谓词限定待扫描的索引片的大小。一个或多个列（有时称为匹配列）确定了索引片的起始位置，并使用 B 树索引的结构找到第一个所需的索引条目。同样地，这些列也确定了扫描的结束位置。匹配谓词有时被称为范围限定谓词.

只需访问索引 一个不需要访问表就可以提供所有被请求的数据的访问路径。

索引前读 SQL Server 的一个术语。用于指代向前读取下一组叶子页。

索引过滤 那些无法参与匹配过程的索引列仍然能够被用来与谓词的输入值相比较，如此一来，表访问将只在必要的时候才发生。

索引跳跃式扫描 一个 Oracle 的术语，用于指代读取多个索引片而不是进行全索引扫描。

完整性 数据库的一个状态，在该状态下，所有数据的约束和规则都是有效的。

I/O 处理器发起的一个从磁盘读取一个页或向磁盘写入一个页的请求，该写入伴随着一次更新。

连接方式 优化器做出的关于如何连接表的决定，通常优化器会选择嵌套循环，虽然其他方式也是可用的。

叶子页 索引最下层的页，该页包含了键值及其对应表行的指针，索引行是按键值的顺序存储的。

最近最少使用算法 一个通常被缓存管理器和磁盘服务器缓冲管理器所使用的算法，用于定位那些为了满足新请求而应当被覆盖的页。

列表预读 DB2 for z/OS 中所使用的一种机制，对指向表行的索引指针按照表行所属页的顺序进行排序，从而使得 DBMS 可以通过跳跃式顺序读的方式访问表行。

本地响应时间 排除工作站及服务器之间的传输及等待时间之后，一个事务的运行时间，即服务器响应时间。

锁　一个针对顺序处理所设计的数据结构，用于保证逻辑一致性，通常关联至一张表、页或行。

物化结果行　执行所需的数据库访问来构建结果集。在最佳情况下，这一过程只需将一行记录从数据库缓冲池传递给应用程序。在最差情况下，DBMS 需要发起大量的磁盘读请求。

302

合并扫描　一种连接方式。在此种连接方式下，一张或多张表按统一的顺序被排序（在使用本地谓词过滤过之后），然后再将表或工作文件中满足条件的行进行合并（Oracle：*排序合并连接*）。

镜像　将所有的页同时写入两个磁盘驱动器。

多块 I/O　Oracle 顺序读。

多索引访问　从多个索引或同一索引的多个索引片上采集指针，对其进行比较，然后再访问所需的行。也被称为*索引与*（索引交集）和*索引或*（索引并集）。

多重顺序前读　SQL Server 顺序读。

多行 FETCH　在游标操作中用一次 SQL 调用同时请求多行数据。

嵌套循环　一种连接方式。在这种连接方式下，DBMS 首先从外层表上找出一行满足本地谓词的数据，然后再在下一个表（即内层表）上查找相关的行，并校验哪些是满足内层表的本地谓词的，以此类推。

非叶子页　叶子页以外的索引页，这类页包含了一个（可能被截断了的）键值，该键值包含了指向下一层级的页的指针及该页上的最大键值。

null　一个空值或未知值。当存储一个表行时，若表行中有一列未指定值，那么 DBMS 会用一个特殊的标识来指代 null。

优化器　关系型数据库管理系统的一个组件，该组件负责为每一个 SQL 语句选择访问路径。它会评估每一个合适的访问路径的成本，通常这是基于 I/O 时间和 CPU 时间的权重之和来得到的。

页　索引行和表行是按页组织在一起的（Oracle 使用“块”这一术语）。通常，页的大小为 4KB，但也有使用其他大小的页的情况。页的大小决定了能在页中存储的索引行和表行的数量。在读取数据时，DBMS 会从磁盘上读取一整页到缓冲池中，因此单个 I/O 能够一次读取多行。

谓词　SQL 语句的 WHERE 子句中的一个搜索参数。

主键　唯一确定一个表行的一个或一组列。

查询　一个以 SQL 的形式表达的数据请求，该请求提供了满足搜索参数的数据行。

RAID 5　一个由廉价的磁盘所组成的 5 级冗余阵列——一个常见的存储数据的方法——逻辑卷被条带至多个构成一个 RAID 组的磁盘驱动器上。例

如第一个 32 KB 的条带被写至磁盘驱动器 1，第二个条带被写至磁盘驱动器 2，以此类推。

RAID 10 适用于频繁随机写入、更新或删除的数据库，实际上它就是 RAID 0 镜像 + RAID 1 条带。与奇偶校验冗余数据不同，一个更新了的页会被同时写入两个磁盘驱动器，一个页也可以从任何一个磁盘驱动器上读取。由随机写导致的 RAID 10 的磁盘负载（磁盘繁忙度）比 RAID 5 低，但前者需要更多的磁盘驱动器。

读缓存 是磁盘服务器的半导体内存（RAM）中的一块区域，用于存储最近从磁盘驱动器上读取过的页。读缓存的目的是为了减少从磁盘驱动器读取页的次数。通常，读缓存比数据库的内存缓冲池要大得多，能保存在最近 20 分钟左右的时长内被随机读取过的页。 303

冗余 出于安全或性能的考虑，在磁盘驱动器或表中存储额外的数据拷贝。RAID 5 冗余是指为一个条带组（如 7 个条带，每个条带 32 KB）上的每一个位都存储一个奇偶校验位。有了这些奇偶校验位后，当磁盘损坏时，这些条带中的任何一个都能够被重建。

关系 在表的关系模型中所使用的术语。

关系型数据库 一个根据关系型模型用关系型 DBMS 构建的数据库。

重组 索引被重组以还原其正确的物理顺序，这对于索引片扫描和全索引扫描很重要。表被重组以还原空闲空间及表行的顺序。

根页 B 树索引结构中最顶端的页。

行 一个表行必须能被单个表页容纳，一个索引行必须能被单个叶子页容纳。

顺序预读 DB2 顺序读。

顺序读 将多个索引或叶子页从磁盘加载到缓冲池中。由于 DBMS 提前就知道需要请求哪些页，所以可以在页被真正请求之前先执行读取操作，这类操作包括顺序预读、多块 I/O 和多重连续预读。

服务时间 通常指 CPU 时间和同步磁盘 I/O 时间的总和。

跳跃式顺序 按一个方向扫描一组不连续的行。

条带 RAID 条带是指将索引或表的第一个条带（如 32 KB）存储在磁盘驱动器 1 上，将第二个条带存储在磁盘驱动器 2 上，以此类推，从而将负载平均分布在一组磁盘上。由于磁盘服务器可能提前从多个磁盘上并行读取数据，因此当请求下一组页时，这些页很可能已经被保存在磁盘服务器的读缓存中了。

汇总表 反范式化的事实表。由于是反范式化的，所以不需要连接，所有所需的列都在一张表里。

同步 I/O 当进行 I/O 操作时，DBMS 无法再进行其他操作，即它必须等待，

直至 I/O 操作完成。

表　关系的实现，由包含列值的行组成。

事务　用户和程序之间的一次与本地响应时间相关的交互，它由一个或多个 SQL 调用构成，有可能只是读取，也有可能涉及更新。

触发器　一个存储在数据库中的程序模块，当一个 SQL 调用访问表时，该程序模块会自动开始执行。不同的 SQL 调用都有各自专门针对插入、删除和更新操作的触发器。它们通常由 SQL 和程式化扩展（例如 SQL Server 中的 T-SQL 或 Oracle 中的 PL/SQL Oracle）编写而成。

视图　一个虚拟表，虽然它本身并不存储任何数据，但它提供了表列的一个子集，由一个包含 SELECT 语句的 create view 语句所定义。

写缓存　磁盘服务器的半导体内存（RAM）中的一个区域，该区域用于存储数据，并由电池电源保护（非易失性存储，NVS）。DBMS 每秒可能会
304 进行多次将被修改的页写至磁盘驱动器上的操作，这些页首先就会被存储在写缓存中。而缓存中的 LRU 页则会被写至磁盘驱动器上。写缓存可能包含了那些在最近几分钟内被更新过的页，若某个页被频繁地更新，那么该页可能会长久地停留在写缓存中。缓存越大，更新引起的磁盘负载（磁盘繁忙度）就越低。

索引

A

access path, 31
  hint, 35, 253, 262
access pattern, 60
alarm limit, 86, 122, 292
AND operator, 92
arithmetic expression, 91
AS/400 system, 56
assisted random read, 16, 275
asynchronous read, 19, 59, 116
audience, 3
automating, 8
auxiliary table, 88

B

B-tree index, 4, 26, 199, 204, 268
background, 3
balanced tree index, 13
basic estimates, 145
basic join question, 159, 173, 175, 293, 297
basic question, 9, 63, 125, 159
batch job, 65, 86, 108, 247, 274
batch program, 65, 86, 219, 292
best index, 54, 64, 79, 83, 84, 99, 253, 254
binary search, 70
binomial distribution, 210
bit vector, 25
bitmap index, 25, 198
block, 22
BQ, 64, 77, 100, 293
  verification, 87
browsing, 7, 247
bubble chart, 112, 131
buffer pool, 4, 13, 64, 69, 70, 260
  hit ratio, 64
  size, 271
  subpool, 270, 274
buffer pool hits, 260

C

cache hits, 260
candidate A, 55, 79, 83, 93, 99, 149, 167
candidate B, 55, 84, 99, 149
candidate key index, 24
cardinality, 39, 41, 246, 252, 255
Cartesian product, 185
Cloudscape, 251
cluster, 26
clustered, 27
  index, 24
  index scan, 68
clustering
  index, 23
  ratio, 162
column
  correlation, 38, 255
  fixed length, 23
  non-key, 238
  restrictive, 4
  variable length, 23, 232
  volatile, 7, 216
comebacks, 273
commit point, 86, 121
comparison
  performance, 80
computer-assisted index design, 290
control information, 13
cost, 21
  access selection, 36
  additional index, 58
  assumptions, 21
  CPU time, 21
  denormalization, 180
  disk servers, 21
  disk space, 21
  maintenance, 80

cost (*continued*)
memory, 21
sort, 84
storage, 21
counting touches, 70
covering index, 33
CPU
assumptions, 48
cache, 281
coefficients, 278
queuing, 122, 267
time, 21, 65, 112, 123, 303
time estimation, 278
CQUBE, 278
culprit, 112, 120, 124
cursor, 42, 92
split, 92, 248

D
data block prefetching, 18, 277
data integrity, 90
data warehouse, 186, 271
data-partitioned secondary indexes, 243
DB2, 7, 22, 23, 25–27, 33, 34, 42, 91, 93, 108, 111, 123, 132, 232, 233, 247, 250, 252, 275
DB2 for LUW, 23
DB2 for z/OS, 17
denormalization, 160, 192, 197, 283, 303
downward, 176
upward, 176
vs join, 180
dimension table, 185
disk
assumptions, 48
cache, 68
load, 7
space, 61
storage, 5
disk drive, 20
utilization, 267
disk read ahead, 20
disk server, 5, 70, 269
cache, 14
disorganization, 6, 68
drive load, 59
drive queuing, 20, 121

E
early materialization, 78, 176
education, 4
elapsed time, 123
exception
monitor, 111
monitoring, 7
exception report, 110, 112
call-level, 123, 293
LRT-level, 293
EXPLAIN, 33, 34, 44, 91, 94, 106, 108, 119
EXPLAIN PLAN, 34
extent, 25

F
fact table, 185, 192, 197
fat index, 51, 54, 61, 64, 190, 251, 272, 297
fault tolerance, 21, 61
FETCH
multi-row, 70, 282
field procedure, 91
files
open, 64
filter factor
pitfall, 94
filter factor, 37, 56, 65, 78, 89, 95, 107, 143, 252
first star, 50, 64, 298
FIRST_ROWS(n), 35
foreign key, 190
free space
distributed, 61
index calculation, 208
frequently used data, 15
full index scan, 87, 92, 106, 119
full table scan, 68, 70, 92, 106, 119
function, 91
fuzzy indexing, 174

G
get page, 123
guidelines, 4

H
hardware, 3
hardware capacity, 8
hashing, 26
histogram, 35, 111, 253

home page, 26, 69
host variable, 107
hot spots, 215, 273

I
IBM iSeries, 271
ideal index, 54, 62, 93, 143, 149, 167, 272, 292, 298
  join, 170
IN, 94
in-storage tables, 271
inadequate indexing, 4, 7, 9, 79
index
  ANDing, 195, 302
  backwards, 44
  block, 243
  both directions, 240
  candidates, 9
  clustering, 69, 74
  columns, 231
  composite, 7
  covering, 251
  creation examples, 234
  creeping, 208
  design, 56
  design method, 8
  design tool, 62
  function-based, 241
  hot spots, 217
  join, 200
  key truncation, 241
  length, 218, 232
  levels, 13
  locking, 232
  maintenance, 5, 58, 62
  multiple slices, 88
  non-clustering, 75
  non-leaf pages, 70
  non-unique, 12
  number, 232
  only, 64, 72, 79
  options, 237
  ORing, 195, 302
  pointer, 117, 195, 302
  resident, 274
  restrictions, 231
  row, 12
  row suppression, 233, 237
  screening, 64, 92, 108
  size, 232
  slice, 39, 54, 67, 68, 72, 297
  suppression, 33
  unique, 12, 240
index design summary, 291
index read-ahead, 18
index skip scan, 18, 90, 242
index-organized table, 24
inner table, 136, 302
insert
  pattern, 61, 208
  random, 208
  rate, 7
integrity check, 121
Intel servers, 21
interleave, 22
intermediate table, 190

J
join, 9
  comparison, 170
  hash, 165
  hash join, 285
  hybrid, 136
  ideal index, 170
  inner, 140
  merge scan, 136, 163, 285
  method, 136
  multiple, 171
  nested loop, 136
  nested loop, 140
  outer, 140

K
key
  descending, 150, 168
  foreign, 8
  modified, 6
  partial, 256
  primary, 8
  sequence, 6
  truncation, 269

L
leaf page, 5, 49, 67, 70, 204, 267
leaf page split, 24, 206, 226
  ratio, 207
LIKE, 91
LIO, 123
list prefetch, 17, 277
local disk drives, 21

local response time, 7, 112
lock wait, 66, 86, 112, 121
lock wait trace, 121
logical I/O, 123
LRT, 65
LRT-level exception monitoring, 126
LRU algorithm, 271

M
mainframe servers, 21
matching
  columns, 31, 72, 79
  index, 64
materialization, 42, 56, 301
  early, 44
matrix, 8
memory, 4, 55
mirroring, 25
misconceptions, 4
monitoring, 108, 271
  software, 9
  splits, 226
multi-block I/O, 16
multi-clustered indexes, 27
multiple index access, 92, 195
multiple serial read-ahead, 16
multitier environment, 65
myths, 4

N
network delays, 65
nine steps, 292
non-BT, 247
nonclustered indexes, 27
nonindexable, 33
nonleaf page, 5, 49, 64, 267
nonsargable, 33
nonSQL time, 120
NULL, 233, 251

O
one screen transaction, 153
optimizer, 30, 92, 148, 246
  cost based, 9, 162, 246, 253
  cost estimates, 107, 252
  cost formulae, 259
  CPU time, 261
  helping, 248, 261
  I/O time, 259
  transformation, 248
OPTIONS (FAST n), 35
OR, 92, 247
Oracle, 6, 18, 22, 23, 25, 26, 33, 35, 40, 90, 108, 123, 126, 131, 224, 233, 238, 247, 277
  cursor sharing, 254
  hints, 262
other waits, 112, 118, 122
outer table, 136, 139, 302

P
page, 12, 21, 302
  adjacency, 24
  number, 12
  request, 123, 125
  size, 22
parallelism, 19, 116
Peoplesoft, 255
perfect order, 70
performance
  monitor, 110
  problems, 2
  tools, 106
pitfall list, 91
pointer
  direct, 61
  length, 61
  symbolic, 61
pool, 273
predicate, 30
  compound, 30, 37, 255
  difficult, 33, 91, 246
  join, 136
  local, 136, 139, 142, 159, 185, 297
  matching, 33, 301
  non-boolean, 92
  nonindexable, 91, 298
  range, 53, 82, 88, 92, 252
  really difficult, 34, 247, 298
  redundant, 264
  simple, 30, 92
  stage 1, 34
  stage 2, 34
  suspicious, 91
prediction formulae, 9
prefetch, 16, 303
production, 4
  cutover, 4, 9
  workload, 8
promising culprits, 125

Q
QUBE, 9, 55, 58, 63, 65, 138, 246, 267, 292
  accuracy, 73
  assumptions, 66, 267
query table, 197
queuing, 121
  disk drive, 66
  theory, 122

R
RAID 10, 25, 59
RAID 5, 25, 59, 68
RAM, 21
random I/O, 69, 107
random read, 20
random touch, 78, 138, 260, 279, 299
  cheap, 67, 260, 275
  unproductive, 125, 179
randomizer, 26
reactive approach, 105
read cache, 5, 60, 70, 78, 88, 115, 272
  neighbors, 267, 270
redundancy, 25
redundant data, 159, 176
relational database, 1
reorganization, 23, 69, 70, 162, 206
  cost, 125
  frequency, 215
  interval, 215
  summary, 228
RISC, 21
root page, 4, 260
row, 21
  extended, 69
  inserted, 205
  length, 268
  long, 272
RUNSTATS, 34

S
SAP, 255
sargable, 33
screening, 32
second star, 50, 298
seek time, 20
selectivity ratio, 39
semifat index, 63, 78, 272
sequential
  prefetch, 16, 69
  read, 24, 116, 272
  touch, 67, 299
server
  cache, 15
service time, 65, 303
SHOW PLAN, 34
single-tier environment, 65
sixty four bit, 21, 269, 271
skewed distribution, 253
skip-sequential, 17, 90, 116, 275, 287
  benefit, 75, 277
  estimation, 276
  read, 70
sort, 55, 72, 96, 106, 145, 267, 282
  unnecessary, 250
spike report, 108, 110, 111, 112, 131, 132
split monitoring, 226
SQL call exception report, 131
SQL optimizer, 2
SQL pool, 255
SQL Server, 18, 22–24, 27, 35, 107, 123, 129, 200, 223, 232, 247
  hints, 263
stage 2 predicates, 247
standards, 6
star join, 185
statistics, 30, 56, 246
  falsifying, 264
stripe, 20
striping, 21, 25
subqueries, 175
  correlated, 176
  non-correlated, 175
summary table, 192, 303
superfluous index, 57, 62
  possibly, 58
  practically, 57
  totally, 57
suspicious access path, 106, 108
synchronous read, 19, 112, 114, 115, 116
system tables, 271
systematic index design, 9

T
table
  access order, 136, 140, 153, 161, 175, 187
  join, 136
  lookup, 33, 300
  resident, 274

table (*continued*)
small, 273
temporary, 163
touches, 70
unnecessary touches, 251
very small, 162
table design
optimistic, 175
unconscious, 180
textbooks, 4
thick slice, 31, 84
thin slice, 31, 53, 72, 79, 88
third star, 51, 75, 298
three star
algorithm, 54
three star index, 9, 49, 51, 79, 80, 100, 151
touch, 67, 69, 268
trace, 108
trigger, 88
tuning potential, 119
two candidate algorithm, 56, 62, 64

U
UNION, 176
UNION ALL, 94, 176, 249
UNIX, 21

V
victim, 112

W
wait for prefetch, 112, 120
Windows servers, 21
workfile, 271
write cache, 60

Broadview
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

通过设计适用于现今硬件的索引来提升关系型数据库的性能

在过去的几年中，硬件和软件的发展速度超出了所有人的想象，因此关系型数据库的性能也不如以前那么受人关注了。然而，事实却是硬件的发展速度并没有跟上日益增长的数据处理量。尽管磁盘压缩密度的大幅增加使得存储的成本很低且顺序读的速度很快，但是随机读的速度却仍旧很慢。于是，许多原有的设计建议已无法适用于现今的情况——索引的最优化点已经有了很大的提升。但许多原有的问题其实并没有消失——它们只是以另一种形式呈现而已。

本书提供了一种简单高效的设计索引和表的方法。作者通过大量的举例及案例研究描述了DB2、Oracle和SQL Server优化器是如何决定以何种方式访问数据库的，同时还阐述了快速估算所选择的访问路径的CPU及响应时间的方法。这使得对比不同设计方案的优劣成为了可能，且能帮助你在众多方案中选出最合适的那一个。

本书的目标读者是那些希望理解SQL性能相关内容，并希望了解如何有效设计表和索引的人。通过本书，拥有多年关系型系统经验的读者能够更好地判断新硬件的引入所可能带来的变化。

Tapio Lahdenmäki，数据库性能顾问，教授通用索引设计课程。他在IBM公司工作了三十多年，是公司全球课程中有关DB2 (for z/OS)性能相关课程的主要作者。

Michael Leach，关系型数据库顾问，已从IBM公司退休，他拥有二十年的应用系统及数据库课程的教授经验。两位作者的文章均被翻译成了多国语言广为传播。他们有关索引设计的方法被成功应用于许多核心系统。

上架建议：数据库

ISBN 978-7-121-26054-4

9 787121 260544

定价：79.00元

WILEY

策划编辑：张春雨
责任编辑：徐津平
封面设计：李　玲