大型数据库优化设计方案

杨德仁 （银川 宁夏邮电通信软件中心 750001）
马晓燕 （银川供电局 750011）

摘要：本文首先讨论了基于第三范式的数据库表的基本设计，着重论述了建立主键和索引的策略和方案。然后从数据库表的扩展设计和库表对象的放置等角度概述了数据库管理系统的优化方案。

关键词：优化 第三范式（3NF） 冗余数据 索引 数据分割 对象放置（Object Placement）

1 引言

数据库优化的目标无非是避免磁盘I/O瓶颈、减少CPU利用率和减少资源竞争。为了便于读者阅读和理解，笔者参阅了Sybase、Informix和Oracle等大型数据库系统参考资料，基于多年的工程实践经验，从基本表设计、扩展设计和数据库表对象放置等角度进行讨论，着重讨论了如何避免磁盘I/O瓶颈和减少资源竞争，相信读者会一目了然。

2 基于第三范式的基本表设计

在基于表驱动的信息管理系统（MIS）中，基本表的设计规范是第三范式（3NF）。第三范式的基本特征是非主键属性只依赖于主键属性。基于第三范式的数据库表设计具有很多优点：一是消除了冗余数据，节省了磁盘存储空间；二是有良好的数据完整性限制，即基于主外键的参照完整限制和基于主键的实体完整性限制，这使得数据容易维护，也容易移植和更新；三是数据的可逆性好，在做连接（Join）查询或者合并表时不遗漏、也不重复；四是因消除了冗余数据（冗余列），在查询（Select）时每个数据页存的数据行就多，这样就有效地减少了逻辑I/O，每个Cache存的页面就多，也减少物理I/O；五是对大多数事务(Transaction)而言，运行性能好；六是物理设计(Physical Design)的机动性较大，能满足日益增长的用户需求。

在基本表设计中，表的主键、外键、索引设计占有非常重要的地位，但系统设计人员往往只注重于满足用户要求，而没有从系统优化的高度来认识和重视它们。实际上，它们与系统的运行性能密切相关。现从系统数据库优化角度讨论这些基本概念及其重要意义：

(1) 主键(Primary key)：主键被用于复杂的SQL语句时，频繁地在数据访问中被用到。一个表只有一个主键。主键应该有固定值（不能为null或缺省值，要有相对稳定性），不含代码信息，易访问。把常用（众所周知）的列作为主键才有意义。短主键最佳（小于25bytes），主键的长短影响索引的大小，索引的大小影响索引页的大小，从而影响磁盘I/O。主键分为自然主键和人为主键。自然主键由实体的属性构成，自然主键可以是复合性的，在形成复合主键时，主键列不能太多，复合主键使得Join操作复杂化、也增加了外键表的大小。人为主键是，在没有合适的自然属性键、或自然属性复杂或灵敏度高时，人为形成的。人为主键一般是整性值（满足最小化要求），没有实际意义，也略微增加了表的大小；但减少了把它作为外键的表的大小。

(2) 外键（Foreign key）：外键的作用是建立关系型数据库中表之间的关系（参照完整性），主键只能从独立的实体迁移到非独立的实体，成为后者的一个属性，被称为外键。

(3) 索引(Index)：利用索引优化系统性能是显而易见的，对所有常用于查询中的where子句的列和所有用于排序的列创建索引，可以避免整表扫描或访问，在不改变表的物理结构的情况下，直接访问特定的数据列，这样减少数据存取时间；利用索引可以优化或排除耗时的分类操作；把数据分散到不同的页面上，就分散了插入的数据；主键自动建立了唯一索

引，因此唯一索引也能确保数据的唯一性（即实体完整性）；索引码越小，定位就越直接；新建的索引效能最好，因此定期更新索引非常必要。索引也有代价：有空间开销，建立它也要花费时间，在进行insert、delete和update操作时，也有维护代价。索引有两种：聚族索引和非聚族索引。一个表只能有一个聚族索引，可有多个非聚族索引。使用聚族索引查询数据要比使用非聚族索引快。在建索引前，应利用数据库系统函数估算索引的大小。

① 聚族索引（Clustered index）：聚族索引的数据页按物理有序储存，占用空间小。选择策略是，被用于where子句的列：包括范围查询、模糊查询或高度重复的列（连续磁盘扫描）；被用于连接joins操作的列；被用于order by和group by子句的列。聚族索引不利于插入操作，另外没有必要用主键建聚族索引。

② 非聚族索引（Nonclustered index）：与聚族索引相比，占用空间大，而且效率低。选择策略是，被用于where子句的列：包括范围查询、模糊查询（在没有聚族索引时）、主键或外键列、点（指针类）或小范围（返回的结果域小于整表数据的20%）查询；被用于连接Join操作的列、主键列（范围查询）；被用于Order by和Group by子句的列；需要被覆盖的列。对只读表建多个非聚族索引有利。索引也有其弊端，一是创建索引要耗费时间，二是索引要占有大量磁盘空间，三是增加了维护代价（在修改带索引的数据列时索引会减缓修改速度）。那么，在哪种情况下不建索引呢？对于小表（数据小于5页）、小到中表（不直接访问单行数据或结果集不用排序）、单值域（返回值密集）、索引列值太长（大于20bitys）、容易变化的列、高度重复的列、Null值列，对没有被用于Where子语句和Join查询的列不建索引，都不能建索引。另外，对主要用于数据录入的，尽可能少建索引。当然，也要防止建立无效索引，当Where语句中多于5个条件时，维护索引的开销大于索引的效益，这时，建立临时表存储有关数据更有效。

批量导入数据时的注意事项：在实际应用中，大批量的计算（如电信话单计费）用C语言程序做，这种基于主外键关系数据计算而得的批量数据（文本文件），可利用系统的自身功能函数（如Sybase的BCP命令）快速批量导入，在导入数据库表时，可先删除相应库表的索引，这有利于加快导入速度，减少导入时间。在导入后再重建索引以便优化查询。

(4) 锁：锁是并行处理的重要机制，能保持数据并发的一致性，即按事务进行处理；系统利用锁，保证数据完整性。因此，我们避免不了死锁，但在设计时可以充分考虑如何避免长事务，减少排它锁时间，减少事务中与用户的交互，杜绝让用户控制事务的长短；要避免批量数据同时执行，尤其是耗时并用到相同的数据表。锁的征用：一个表同时只能有一个排它锁，一个用户用时，其他用户在等待。若用户数增加，则Server的性能下降，出现“假死”现象。如何避免死锁呢？从页级锁到行级锁，减少了锁征用；给小表增加无效记录，从页级锁到行级锁没有影响，若在同一页内竞争有影响，可选择合适的聚族索引把数据分配到不同的页面；创建冗余表；保持事务简短；同一批处理应该没有网络交互。

(5) 查询优化规则：在访问数据库表的数据(Access Data)时，要尽可能避免排序（Sort）、连接(Join)和相关子查询操作。经验告诉我们，在优化查询时，必须做到：

① 尽可能少的行；

② 避免排序或为尽可能少的行排序，若要做大量数据排序时，最好将相关数据放在临时表中操作；用简的单键（列）排序，如整形或短字符串排序；

③ 避免表内的相关子查询；

④ 避免在where子句中使用复杂的表达式或非起始的子字符串、用长字符串连接。

⑤ 在where子句中多使用“与”（And）连接，少使用“或”(Or)连接。

⑥ 利用临时数据库。在查询多表、有多个连接、查询复杂、数据要过滤时，可以建临时表（索引）以减少I/O。但缺点是增加了空间开销。

除非每个列都有索引支持，否则在有连接的查询时分别找出两个动态索引，放在工作表中重新排序。

3 基本表扩展设计

基于第三范式设计的库表虽然有其优越性（见本文第一部分），然而在实际应用中有时不利于系统运行性能的优化：如需要部分数据时而要扫描整表，许多过程同时竞争同一数据，反复用相同行计算相同的结果，过程从多表获取数据时引发大量的连接操作，当数据来源于多表时的连接操作；这都消耗了磁盘I/O和CPU时间。

尤其在在遇到下列情形时，我们要对基本表进行扩展设计：许多过程要频繁访问一个表、子集数据访问、重复计算和冗余数据，有时用户要求一些过程优先或低的响应时间。

如何避免这些不利因素呢？根

据访问的频繁程度对相关表进行分割处理、存储冗余数据、存储衍生列、合并相关表处理，这是克服这些不利因素和优化系统运行的有效途径。

3.1 分割表或储存冗余数据

分割表分为水平分割表和垂直分割表两种。分割表增加了维护数据完整性的代价。

水平分割表：一种是当多个过程频繁访问数据表的不同行时，水平分割表，并消除新表中的冗余数据列；若个别过程要访问整个数据，则要用连接操作，这也无妨分割表；典型案例是电信话单按月分割存放。另一种是当主要过程要重复访问部分行时，最好将被重复访问的这些行单独形成子集表（冗余储存），这在不考虑磁盘空间开销时显得十分重要；但在分割表以后，增加了维护难度，要用触发器立即更新、或存储过程或应用代码批量更新，这也会增加额外的磁盘I/O开销。

垂直分割表（不破坏第三范式），一种是当多个过程频繁访问表的不同列，可将表垂直分成几个表，减少磁盘I/O（每行的数据列少，每页存的数据行就多，相应占用的页就少），更新时不必考虑锁，没有冗余数据。缺点是要在插入或删除数据时要考虑数据的完整性，用存储过程维护。另一种是当主要过程反复访问部分列，最好将这部分被频繁访问列数据单独存为一个子集表（冗余储存），这在不考虑磁盘空间开销时显得十分重要；但这增加了重叠列的维护难度，要用触发器立即更新、或存储过程或应用代码批量更新，这也会增加额外的磁盘I/O开销。垂直分割表就可以达到最大化利用Cache的目的。

总之，为主要过程分割表的方法适用于：各个过程需要表的不联结的子集；各个过程需要表的子集，访问频率高的主要过程不需要整表。在主要的、频繁访问的主表需要表的子集而其他主要频繁访问的过程需要整表时则产生冗余的子集表。

注意，在分割表以后，要考虑重新建立索引。

3.2 存储衍生数据。

对一些要做大量重复性计算的过程而言，若重复计算过程得到的相同结果（源列数据稳定，因此计算结果也不变），或计算牵扯多行数据需额外的磁盘I/O开销，或计算复杂需要大量的CPU时间，就考虑存储计算结果（冗余储存）。现予以分类说明：

若在一行内重复计算，就在表内增加列存储结果。但若参与计算的列被更新时，必须要用触发器更新这个新列。

若对表按类进行重复计算，就增加新表（一般而言，存放类和结果两列就可以了）存储相关结果。但若参与计算的列被更新时，就必须要用触发器立即更新、或存储过程或应用代码批量更新这个新表。

若对多行进行重复性计算（如排名次），就在表内增加列存储结果。但若参与计算的列被更新时，必须要用触发器或存储过程更新这个新列。

总之，存储冗余数据违反第三范式，这会增加维护数据完整性的代价，必须用触发器立即更新、或存储过程或应用代码批量更新，以维护数据的完整性。

3.3 消除昂贵结合

对于频繁同时访问多表的一些主要过程，考虑在主表内存储冗余数据，即存储冗余列或衍生列（它不依赖于主键），但破坏了第三范式，也增加了维护难度。在源表的相关列发生变化时，必须要用触发器或存储过程更新这个冗余列。当主要过程总同时访问两个表时可以合并表，这样可以减少磁盘I/O操作。但破坏了第三范式，也增加了维护难度。对父子表和1：1关系表合并方法不同，合并父子表后，有冗余表；1：1关系表合后，在表内有冗余数据。

4 数据库对象的放置策略

数据库对象的放置策略是均匀地把数据分布在系统的磁盘中，平衡I/O访问，避免I/O瓶颈。

(1) 把访问分散到不同的磁盘，即使用户数据尽可能跨越多个设备，多个I/O运转，避免I/O竞争，克服访问瓶颈；分别防置随机即访问和连续访问。

(2) 分离系统数据库I/O和应用数据库I/O。把系统审计表和临时库表放在不忙的磁盘上

(3) 把事务日志放在单独的磁盘上，减少磁盘I/O开销，还有利于在障碍后恢复，提高了系统的安全性。

(4) 把频繁访问的“活性”表放在不同的磁盘上；把频繁用的表、频繁做Join操作的表分别放在单独的磁盘上，甚至把频繁访问的表的字段放在不同的磁盘上，把访问分散到不同的磁盘上，避免I/O争夺；

(5) 利用段分离频繁访问的表及其索引（非聚族的）、分离文本和图象数据。段的目的是平衡I/O，避免瓶颈，增加吞吐量，实现并行扫描，提高了并发度，最大化磁盘的吞吐量。利用逻辑段功能，分别放置“活性”表及其非聚族索引以平衡I/O。当然最好利用系统的默认段。另外，利用段可以使备份和恢复数据更加灵活，使系统授权更加灵活。■