第 11 讲:连接算法

15-445/645 数据库系统(2022 年秋季)

https://15445.courses.cs.cmu.edu/fall2022/

卡内基梅隆大学安迪·帕夫洛

1 连接

一个好的数据库设计的目标是尽量减少信息的重复。这就是为什么表格是基于规范化理论组成的。因此需要连接来重建原始的表。

本类将涵盖用于合并两个表的**内部等值连接**算法。*等值连接*算法连接键相等的表。这些算法可以调整以支持其他连接。

操作符输出

对于在 join 属性上匹配的元组 r∈r 和元组 s∈s, join 操作符将 r 和 s 连接在一起，形成一个新的输出元组。

在现实中，连接操作符生成的输出元组的内容是不同的。它取决于 DBMS 的查询处理模型、存储模型和查询本身。连接操作符输出的内容有多种方法。

- **数据:**这种方法将外部表和内部表中的属性值复制到元组中，并将元组放入该操作符的中间结果表中。这种方法的优点是，查询计划中的未来操作符永远不需要返回到基表来获取更多的数据。缺点是，这需要更多的内存来物化整个元组。这被称为*早期物化*。DBMS 还可以进行额外的计算并省略查询中稍后不需要的属性，以进一步优化此方法。
- **记录 id:**在这种方法中，DBMS 只复制连接键和匹配元组的记录 id。这种方法非常适合列存储，因为 DBMS 不会复制查询不需要的数据。这被称为*晚物化(late materialization)*。

成本分析

这里用于分析不同连接算法的成本指标是用于计算连接的磁盘 I/o 数。这包括从磁盘读取数据以及向磁盘写入任何中间数据所产生的 I/o。

注意，只考虑计算连接产生的 I/o，而不考虑输出结果时产生的 I/O。这是因为输出成本取决于数据，而且，任何连接算法的输出都是相同的，因此在不同的算法之间，成本不会改变。

Variables used in this lecture:
- M pages in table R (Outer Table), m tuples total
- N pages in table S (Inner Table), n tuples total

一般来说，会有很多算法/优化在某些情况下可以减少连接成本，但没有一个算法在每个场景下都能很好地工作。

2 嵌套循环连接

在高层次上，这种类型的连接算法由两个嵌套的 for 循环组成，它们遍历两个表中的元组并成对地比较它们。如果元组与连接谓词匹配，则输出它们。外层 for 循环中的表称为*外层表*，而内层 for 循环中的表称为*内层表*。

DBMS 总是希望使用“较小”的表作为外部表。更小的表可以是元组的数量或页面的数量。DBMS 还希望在内存中缓冲尽可能多的外部表。它也可以尝试利用索引在内部表中查找匹配。

简单嵌套循环联接

对于外部表中的每个元组，将其与内部表中的每个元组进行比较。这是最坏的情况，DBMS 必须在没有任何缓存或访问局部性的情况下，对外部表中的每个元组进行内部表的整个扫描。

Cost: $M + (m \times N)$

块嵌套循环联接

对于外部表中的每个块，从内部表中获取每个块，并比较这两个块中的所有元组。这种算法执行较少的磁盘访问，因为 DBMS 为每个外部表块扫描内部表，而不是为每个元组扫描。

成本:M + (M × N)

如果 DBMS 有 B 缓冲区可用来计算连接，那么它可以使用 B− 2 缓冲区扫描外部表。它将使用一个缓冲区来扫描内部表和一个缓冲区来存储连接的输出。**成本:**M + M × N

B− 2

索引嵌套循环连接

以前的嵌套循环连接算法性能很差，因为 DBMS 必须执行顺序扫描以检查内部表中的匹配。然而，如果数据库已经在连接键上为其中一个表建立了索引，它可以使用该索引来加速比较。DBMS 既可以使用现有索引，也可以为连接操作构建临时索引。

外部表将是没有索引的表。内部表将是有索引的表。

假设每个索引探测的成本是每个元组的某个常量 C。

成本:M + (M × C)

3 Sort-Merge Join

从高层来看，sort-merge join 会对两个表的关联键进行排序。DBMS 可以使用外部归并排序算法来完成这一工作。然后它用游标遍历每个表并发出匹配项(就像在归并排序中一样)。

如果一个或两个表已经按连接属性排序(比如群集索引)，或者输出需要按连接键排序，那么这个算法是有用的。

这个算法最坏的情况是，如果两个表中所有元组的 join 属性包含相同的值，这在真实的数据库中是非常不可能发生的。在这种情况下，合并的成本将是 M • N。不过在大多数情况下，键都是唯一的，所以合并的代价大约是 M + N。

假设 DBMS 有 B 个缓冲区用于算法:

- 表 R 的排序成本:2M × 1 +?logB– 1?M ??

吗?吗?NB? ?

- 表 S 的排序成本:2N × 1 + log
- 合并代价:(M + N)

总代价:排序+合并

4 哈希连接

哈希连接算法的高层思想是使用哈希表将元组根据它们的连接属性分割成更小的块。这减少了 DBMS 在计算连接时需要对每个元组执行的比较次数。散列连接只能用于完全连接键上的等值连接。

如果元组 r∈r 和元组 s∈s 满足连接条件，那么它们对于连接属性具有相同的值。如果该值被哈希为某个值 i，则 R 元组必须在桶 ri 中，S 元组必须在桶 si 中。因此，桶 ri 中的 R 元组只需要与桶 si 中的 S 元组进行比较。

基本的哈希连接

- **阶段#1 -构建:**首先，扫描外部关系并在连接属性上使用哈希函数 h1 填充哈希表。哈希表中的键是连接属性。值取决于实现方式(可以是完整的元组值或元组 id)。
- **阶段#2 -探测:**扫描内部关系，并在每个元组的连接属性上使用哈希函数 h1 跳转到哈希表中的相应位置，并找到匹配的元组。由于哈希表中可能存在冲突，DBMS 将需要检查 join 属性的原始值，以确定元组是否真正匹配。

如果 DBMS 知道外部表的大小，则连接可以使用静态哈希表。如果它不知道大小，那么连接必须使用动态哈希表或允许溢出页。

√

一张有 N 页的表需要 N 个缓冲区。上述方法在阶段#1 中最多创建 B - 1 个大小为 B 块的溢出分区，因此假设哈希函数均匀分布记录，使用这种方法可以哈希的最大表是 B • (B - 1)缓冲区。如果 hash √ 函数不均匀，一个蒙混

因子 f > 1 可以引入，因此这样的表最大的是 B • f • N。

探测阶段的一个优化是使用布隆过滤器。这是一个概率数据结构，可以放入 CPU 缓存，并回答*这个问题: 键* x *在哈希表中吗?要么绝对没有，要么可能有*。这可以通过防止磁盘读取不会导致发出元组来减少磁盘 I/O 的数量。

Grace 散列连接/分区散列连接

当表不适合主内存时，DBMS 必须基本上随机地交换表的进出，这会导致较差的性能。Grace 散列连接是基本散列连接的扩展，它也将内部表散列到写入磁盘的分区中。

- **阶段#1 -构建:**首先，扫描外部表和内部表，并在连接属性上使用哈希函数 h1 填充哈希表。该哈希表的桶根据需要被写入磁盘。如果单个 bucket 不适合内存，DBMS 可以使用*不同哈希函数 h2(其中 h1 = h2)的递归分区*来进一步划分 bucket。这可以递归地继续下去，直到桶装入内存。

- **阶段#2 -探测:**对于每个桶级别，检索外部和内部表的相应页面。然后，对这两个页面中的元组执行嵌套循环联接。页面会放入内存，所以这个连接操作会很快。

分割相位成本:2 ×(M + N)探测相位
成本:(M + N)**总成本:**3×(M + N)

混合哈希连接优化:适应基本哈希连接和 Grace 哈希连接;如果键有偏斜，则将热分区保留在内存中，并立即执行比较，而不是将其溢出到磁盘。难以正确实现。

5 的结论

连接是与关系数据库交互的一个重要部分，因此确保 dbms 具有执行连接的高效算法是至关重要的。

Algorithm	I/O Cost	Example
Simple Nested Loop Join	$M + (m \cdot N)$	1.4 hours
Block Nested Loop Join	$M + (M \cdot N)$	50 seconds
Index Nested Loop Join	$M + (m \cdot C)$	Varies
Sort-Merge Join	$M + N + (\text{sort cost})$	0.75 seconds
Hash Join	$3 \cdot (M + N)$	0.45 seconds

图 1:上表假设如下:M = 1000, M = 100000, N = 500, N = 40000, B = 100，每个?I/O 0.1 ms。这种成本是 R + S = 4000 + 2000 IOs, R = 2 • M • 1 +⌈ log B− 1⌈ M / B⌉ ⌉ = 2000 • 99(1 +⌈ 日志⌈ ⌉ 1000/100)= 4000 和 S = 2 • N • 1 +⌈ 日志 B− 1⌈ N / B?⌉ ⌉ = 1000 • 99(1 +⌈ 日志⌈ ⌉ 500/100)= 2000。

哈希连接几乎总是优于基于排序的连接算法，但在某些情况下，基于排序的连接会是首选。这包括对非均匀数据的查询，当数据已经在连接键上排序，以及当结果需要排序时。好的 dbms 会使用其中一种，或者两种都使用。