第 12 讲:查询处理

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 查询计划

DBMS 将 SQL 语句转换为查询计划。查询计划中的操作符被安排在树中。数据从这棵树的叶子流向根。树中根节点的输出就是查询的结果。典型的操作符是二进制的(1-2 个子)。相同的查询计划可以以多种方式执行。

2 处理模型

DBMS *处理模型*定义了系统如何执行查询计划。它指定了查询计划的执行方向，以及在执行过程中操作符之间传递什么样的数据。处理模型有不同的模型，针对不同的工作负载有不同的权衡。

这些模型也可以实现从上**到下或从下到**上调用操作符。虽然从上到下的方法更常见，但从下到上的方法可以允许更严格地控制*管道*中的缓存/寄存器。

我们考虑的三种执行模型是:

- 迭代器模式
- 实体化模型
- 矢量化/批处理模型

迭代器模式

迭代器模型，也称为 Volcano 或 Pipeline 模型，是最常见的处理模型，几乎每个(基于行的)DBMS 都使用它。

迭代器模型的工作原理是为数据库中的每个操作符实现一个 Next 函数。查询计划中的每个节点在它的子节点上调用 Next，直到到达叶子节点，叶子节点开始发出元组到它们的父节点进行处理。然后，在检索下一个元组之前，每个元组尽可能地向上处理计划。这在基于磁盘的系统中是有用的，因为它允许我们在访问下一个元组或页面之前充分利用内存中的每个元组。迭代器模型的示例图如图 1 所示。

迭代器模型中的查询计划操作符是高度组合的，并且易于理解，因为每个操作符都可以独立于查询计划树中的父或子操作符实现，只要它实现了如下的 Next 函数:

- 在每次调用 Next 时，如果没有更多的元组要发出，操作符要么返回单个元组，要么返回 null 标记。
- 该操作符实现了一个循环，该循环对其子元素调用 Next 来检索它们的元组，然后处理它们。通过这种方式，在父对象上调用 Next 会调用其子对象上的 Next。作为响应，子节点将返回父节点必须处理的下一个元组。

迭代器模型允许*流水线*，DBMS 可以在检索下一个元组之前通过尽可能多的操作符处理元组。在查询计划中为给定元组执行的一系列任务称为*管道*。

有些操作符会阻塞，直到子进程发出他们所有的元组。此类操作符的例子包括联结、子查询和排序(ORDER BY)。这样的操作符被称为*管道断路器*(*pipeline breaker*)。

输出控制很容易使用这种方法(LIMIT)，因为一旦拥有所需的所有元组，操作符就可以停止对其子操作符(或多个子操作符)调用 Next。

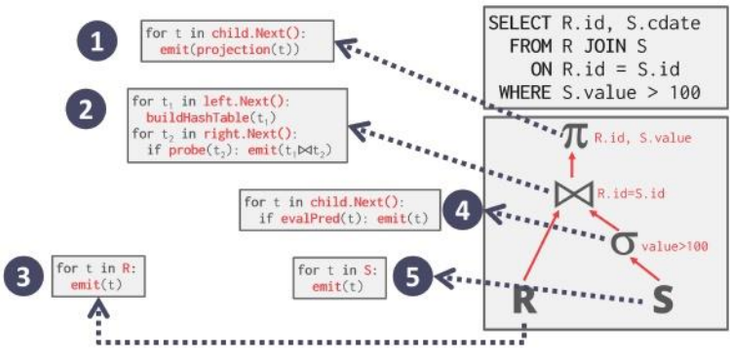

图 1:迭代器模型示例-每个操作符的不同 Next 函数的伪代码。Next 函数本质上是 for 循环，迭代其子操作符的输出。例如，根节点在其子节点上调用 Next，即 join 操作符，这是一种访问方法，遍历关系 R 并发出一个向上的元组，然后对其进行操作。在所有元组处理完毕后，会发送一个 null 指针(或另一个指示符)，让父节点知道继续前进。

实体化的模型

*物化模型*是迭代器模型的特殊化，其中每个操作符一次性处理其所有输入，然后一次性发出其所有输出。而不是有一个返回单个元组的 next 函数，每个运算符在每次到达时返回其所有的元组。为了避免扫描太多元组，DBMS 可以向下传播有关需要多少元组的信息给后续操作符(例如 LIMIT)。操作符将其输出“物化”为单个结果。输出可以是一个完整的元组(NSM)，也可以是一组列的子集(DSM)。物化模型的示意图如图 2 所示。

每个查询计划操作符都实现了一个输出函数:

- 操作符一次性处理来自其子节点的所有元组。
- 此函数的返回结果是 operator 将发出的所有元组。当操作符完成执行后，DBMS 不需要返回操作符来检索更多的数据。

这种方法更适合 OLTP 工作负载，因为一次查询通常只访问少量的元组。因此，检索元组的函数调用更少。物化模型不适合具有大量中间结果的 OLAP 查询，因为 DBMS 可能不得不在操作符之间将这些结果溢出到磁盘上。

向量化模型

和迭代器模型一样，*向量化模型*中的每个运算符都实现了一个 Next 函数。然而，每个运算符发出的是数据的*批处理*(即向量)，而不是单个元组。操作符的内部循环

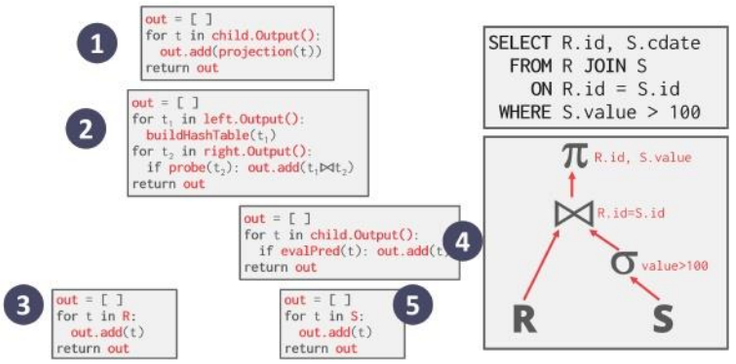

图 2:Materialization 模型示例—从根开始，调用 child.Output()函数，该函数调用下面的操作符，返回所有元组。

实现被优化为处理批量数据，而不是一次只处理一项数据。批量的大小可以根据硬件或查询属性而变化。矢量化模型的例子见图 3。

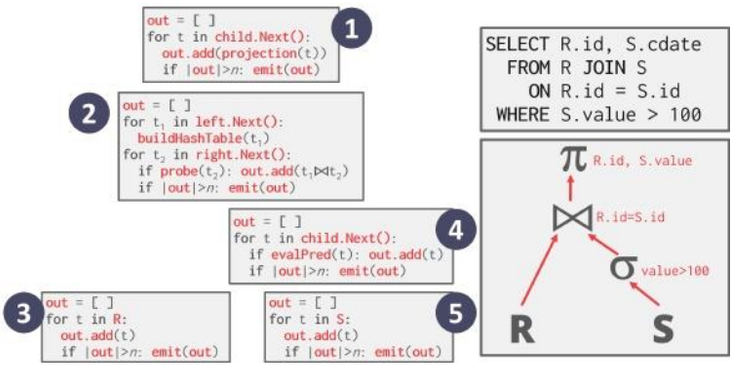

图 3:向量化模型示例—向量化模型与迭代器模型非常相似，除了在每个操作符上，输出缓冲区与期望的发射大小进行比较。如果缓冲区更大，那么一个元组批处理被发送上去。

向量化模型方法非常适合必须扫描大量元组的 OLAP 查询，因为对 Next 函数的调用较少。

向量化模型允许操作符更容易使用向量化(SIMD)指令来处理批量的元组。

处理方向

方法一:从上到下

-从根源开始，将数据从孩子“拉”到父母

-元组总是通过函数调用**传递**

方法 2:从下到上

-从叶子节点开始，将数据从孩子“推送”到父母

-允许更严格地控制操作员管道中的缓存/寄存器

3 .访问方法

*访问方法*是 DBMS 访问表中存储的数据的方式。一般来说，存取模型有两种方法;数据要么从表中读取，要么通过顺序扫描从索引中读取。

顺序扫描

顺序扫描操作符遍历表中的每一页，并从缓冲池中检索它。当扫描遍历每一页上的所有元组时，它会对谓词进行评估，以决定是否将元组发送给下一个操作符。

DBMS 维护一个内部游标，用于跟踪它检查的最后一个页面/槽。

顺序表扫描几乎总是 DBMS 执行查询时效率最低的方法。有一些优化方法可以帮助提高顺序扫描的速度:

- **预取:**提前取下几页，这样 DBMS 在访问每一页时就不必阻塞存储 I/O。
- **缓冲池旁路:**扫描操作符将从磁盘获取的页面存储在本地内存中，而不是缓冲池中，以避免连续的洪泛。
- **并行化:**使用多个线程/进程并行执行扫描。

延迟物化:DSM dbms 可以延迟将元组拼接在一起，直到查询计划的上部。这允许每个操作符将所需的最小信息量传递给下一个操作符(例如，记录 ID，记录在列中的偏移量)。这只在列存储系统中有用。

- **堆聚类:**元组使用由聚类索引指定的顺序存储在堆页面中。
- **近似查询(有损数据跳过):**对整个表的采样子集执行查询以产生近似结果。这通常是在允许低误差产生接近准确答案的场景中计算聚合时进行的。
- **Zone Map(无损数据跳过):**预先计算页面中每个元组属性的聚合。然后，DBMS 可以通过首先检查 Zone Map 来决定是否需要访问某个页面。每个页面的 Zone Map 存储在单独的页面中，并且每个 Zone Map 页面中通常有多个条目。因此，可以减少顺序扫描中检查的总页数。区域图在云数据库系统中特别有价值，因为通过网络进行数据传输需要更大的成本。图 4 是 Zone Map 的一个示例。

图 4:Zone Map 示例—区域地图存储页面中值的预先计算的聚合。在上面的例子中，select 查询从区域映射中实现了原始数据中的最大值只有 400。然后，不必遍历页面中的每个元组，查询可以完全避免访问页面，因为没有一个值将大于 600。

索引扫描

在*索引扫描*中，DBMS 选择一个索引来查找查询所需的元组。

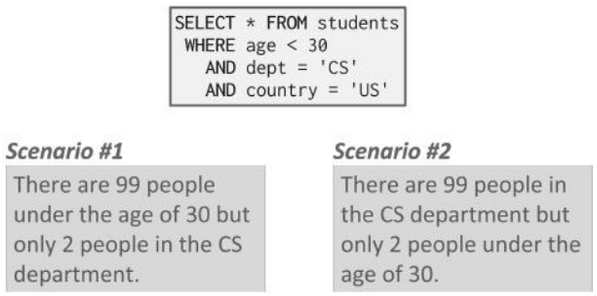

图 5:索引扫描示例—考虑一个包含 100 个元组和两个索引:age 和 department 的表。在第一种情况下，最好在扫描中使用部门索引，因为它只有两个元组要匹配。选择 age 索引并不会比简单的顺序扫描好多少。在第二种情况下，年龄指数会消除更多不必要的扫描，是最优选择。

在数据库管理系统选择索引的过程中，涉及到许多因素，包括:

索引包含什么属性

查询引用哪些属性

•属性的值域

•谓词组合

索引是否有唯一键或非唯一键

图 5 显示了一个索引扫描的简单例子。

更高级的数据库管理系统支持多索引扫描。当为查询使用多个索引时，DBMS 使用每个匹配的索引计算记录 id 集，根据查询的谓词组合这些集，检索记录并应用可能保留的任何谓词。DBMS 可以使用位图、哈希表或 Bloom 过滤器通过集合交集来计算记录 id。图 6 是一个使用多索引扫描的例子。

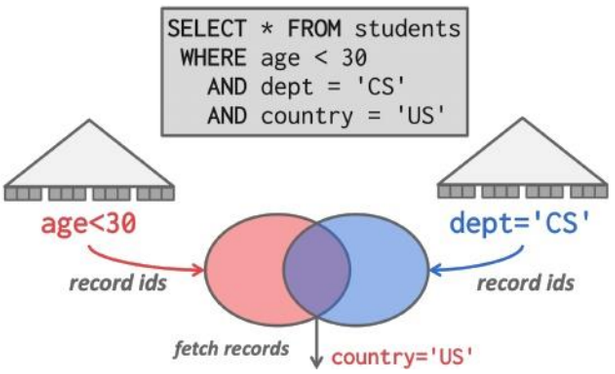

图 6:多索引扫描示例—考虑图 5 中的同一个表。有了多索引扫描支持，我们首先使用相应的索引分别计算满足年龄和深度谓词的记录 id 集。然后我们计算两个集合的交集，获取相应的记录，并应用剩余的谓词 country= ' US '。

4 修改查询

修改数据库的操作符(INSERT、UPDATE、DELETE)负责检查约束和更新索引。对于 UPDATE/DELETE，子操作符传递目标元组的 Record id，并且必须跟踪以前看到的元组。

如何处理 INSERT 操作符有两种实现选择:

- **选择 #1:**在操作符内部物化元组。
- **选择 #2:**Operator 插入任何从子操作符传入的元组。

万圣节难题

万圣节难题是一种异常，其中更新操作更改了元组的物理位置，导致扫描操作符多次访问该元组。这种情况可能发生在集群表或索引扫描上。

这种现象最初是由 IBM 的研究人员在 1976 年万圣节那天构建 **System R** 时发现的。这个问题的解决方案是跟踪每个查询修改的记录 id。

5 表达式求值

DBMS 将 WHERE 子句表示为*表达式树*(参见图 7 中的示例)。树中的节点表示不同的表达式类型。

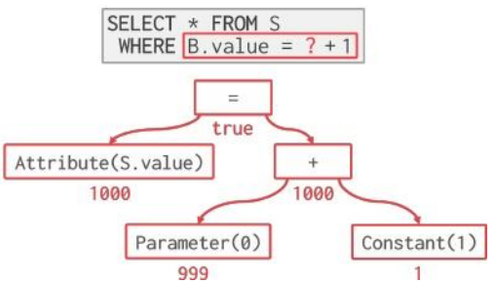

图 7:表达式求值示例—WHERE 子句及其对应表达式的关系图。

可以存储在树节点中的表达式类型的一些例子:

- 比较(=，<，>，!=)

  合取(AND)，析取(OR)

  算术运算符(+，-，*，/，%)

- 常量和参数值

  元组属性引用

为了在运行时对表达式树求值，DBMS 维护一个上下文句柄，其中包含执行的元数据，例如当前元组、参数和表模式。然后，DBMS 遍历树以评估其操作符并生成结果。

以这种方式计算谓词很慢，因为 DBMS 必须遍历整个树，并为每个操作符确定正确的操作。更好的方法是直接求值表达式(想想 JIT 编译)。基于内部成本模型，DBMS 将决定是否采用代码生成来加速查询。