第 14 讲:查询规划与优化

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 概述

由于 SQL 是声明性的，因此查询只告诉 DBMS 要计算什么，而不告诉它如何计算。因此，DBMS 需要将 SQL 语句转换为可执行的查询计划。但是执行查询计划中的每个操作符有不同的方法(例如，连接算法)，这些计划之间的性能会有差异。DBMS 优化器的工作是为任何给定的查询选择最优计划。

查询优化器的第一个实现是 IBM System R，设计于 20 世纪 70 年代。在此之前，人们不相信 DBMS 能够比人类更好地构建查询计划。System R 优化器中的许多概念和设计决策至今仍在使用。

查询优化有两种高级策略。

第一种方法是使用静态规则，或*启发式方法*。启发式方法将查询的部分内容与已知的模式进行匹配，以组合出一个计划。这些规则转换查询以消除低效。虽然这些规则可能需要咨询目录来了解数据的结构，但它们永远不需要检查数据本身。

另一种替代方法是*使用基于成本的搜索*来读取数据，并估计执行等效计划的成本。成本模型选择成本最低的计划。

查询优化是构建 DBMS 中最困难的部分。一些系统已经尝试应用机器学习来提高优化器的准确性和效率，但是目前没有主要的 DBMS 部署基于这种技术的优化器。

逻辑规划与物理规划

优化器生成*逻辑代数表达式*到最佳等效物理代数表达式的映射。逻辑计划大致等价于查询中的关系代数表达式。

*物理操作符*使用访问路径为查询计划中的不同操作符定义特定的执行策略。物理计划可能取决于所处理的数据的物理格式(即排序、压缩)。

从逻辑计划到物理计划并不总是存在一对一的映射。

2 逻辑查询优化

一些选择优化包括:

尽早执行过滤器(谓词下推)。

•重新排序谓词，以便 DBMS 首先应用最具选择性的谓词。

拆分一个复杂的谓词并将其下推(拆分合取谓词)。

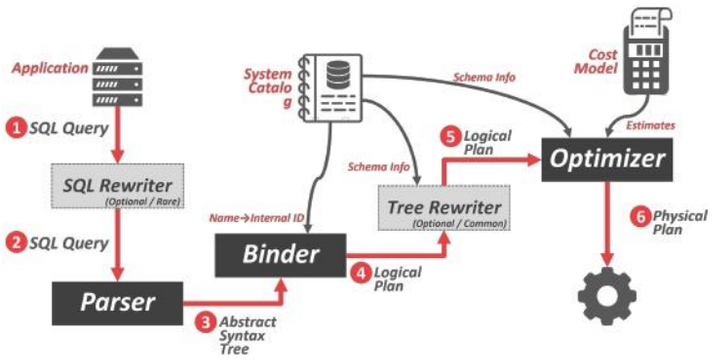

图 1:体系结构概述-连接到数据库系统并发送 SQL 查询的应用程序，该查询可能被重写为不同的格式。SQL 字符串被解析成组成语法树的令牌。活页夹通过查阅系统目录将语法树中的命名对象转换为内部标识符。绑定器发出一个逻辑计划，该计划可以提供给树重写器以获取额外的模式信息。逻辑计划交给优化器，优化器选择最有效的过程来执行该计划。

??中给出了一个谓词下推的例子。

一些投影优化包括:

- 尽早执行投影，以创建更小的元组并减少中间结果(*投影下推*)。
- 投影出除了请求或需要的属性之外的所有属性。

投影下推的一个例子如图 2 所示。

```
SELECT s.name, e.cid
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
   AND e.grade = 'A'
```

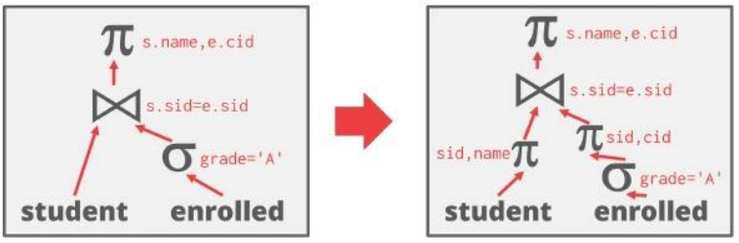

图 2:投影下推—由于查询只请求学生姓名和 ID，DBMS 可以在应用连接之前删除除这两列以外的所有列。

DBMS 可以使用的另一个优化是删除不可能或*不必要的谓词*。在这种优化中，DBMS 省略了对表中每个元组的结果不会改变的谓词的求值。绕过这些谓词可以减少计算成本。图 3 显示了两个不必要谓词的例子。

一个类似的优化是*合并谓词*。图 4 显示了这种优化的一个例子。

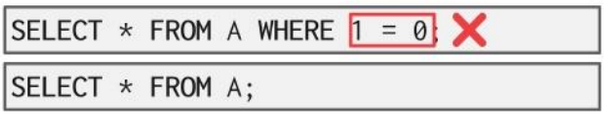

图 3:不必要的谓词—第一个查询中的谓词总是为假，可以忽略。前一个查询可以重写为后一个查询，以产生相同的结果，但节省了计算。

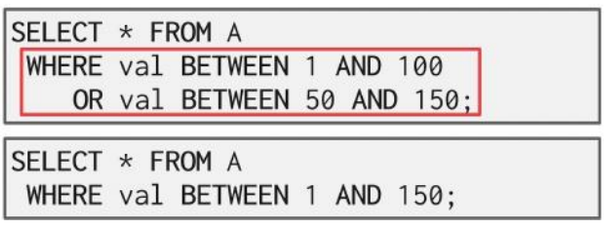

图 4:合并谓词—查询 1 中的 WHERE 谓词具有冗余性，因为它搜索的是 1 到 150 之间的任何值。查询 2 显示了查询 1 中更简洁的表达请求的方式。

JOIN 操作的顺序是查询性能的关键决定因素。穷举枚举所有可能的连接顺序是低效的，因此连接顺序优化需要一个成本模型。然而，我们仍然可以用启发式的方法来优化，消除*不必要的连接*。图 5 展示了一个连接消除的例子。

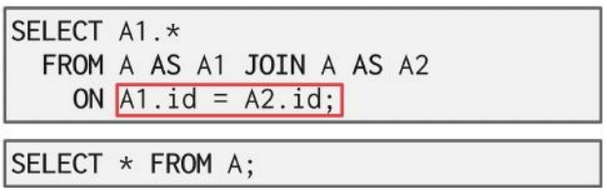

图 5:连接消除-查询 1 中的连接是浪费的，因为 A 中的每个元组都必须存在于 A 中。查询 1 可以改为写成查询 2。

DBMS 还可以在不引用成本模型的情况下优化嵌套子查询。这种类型的优化有两种不同的方法:

- 通过去相关和/或扁平化来重写查询。图 6 显示了一个这样的例子。

分解嵌套查询，并将结果存储到临时表中。图 7 显示了一个这样的例子。

3 成本估算

DBMS 使用成本模型来估计执行计划的成本。这些模型评估查询的等效计划，以帮助 DBMS 选择最优的计划。

查询的成本取决于几个基本的指标，包括:

- **CPU:**成本小，但难以估计。
- **磁盘 I/O:**块传输的数量。

内存:使用 DRAM 的数量。

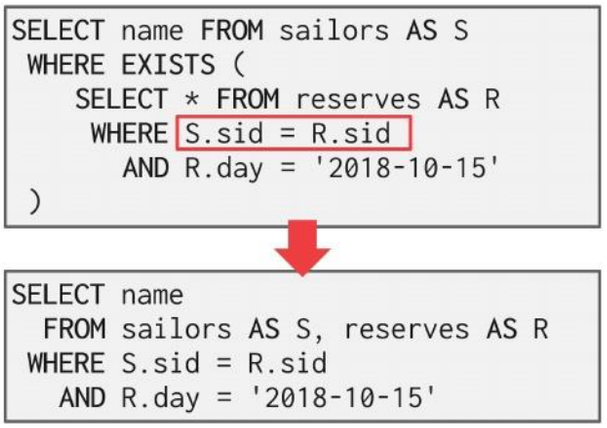

图 6:子查询优化—重写通过将子查询重写为 JOIN，可以将前一个查询重写为后一个查询。以这种方式移除一层嵌套可以有效地*扁平化*查询。

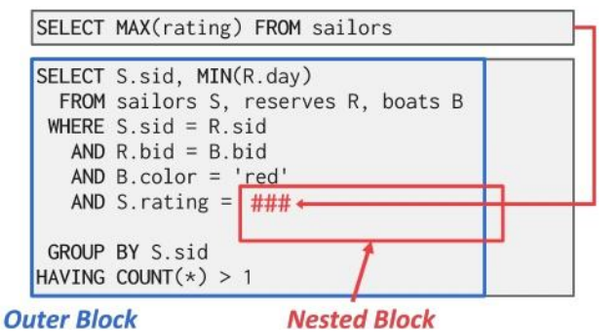

图 7:子查询优化—分解—对于带有子查询的复杂查询，DBMS 优化器可能会将原始查询分解为块，并一次专注于优化每个单独的块。在这个例子中，优化器通过将嵌套的查询拉出到自己的查询中，并随后使用这个结果来实现原始查询的逻辑，从而使用嵌套聚合来分解一个查询。

- **网络 (Network):**发送的消息数量。

穷举查询的所有有效计划对于优化器来说执行太慢了。仅对于联结来说，它是交换性的和结合性的，每个 n 向联结有 4^n 种不同的排序。为了高效地工作，优化器必须限制其搜索空间。

为了估算查询的成本，DBMS 在其内部编目中维护有关表、属性和索引的内部*统计*信息。不同的系统以不同的方式维护这些统计信息。大多数系统试图通过维护一个内部统计表来避免实时计算。然后这些内部表可能会在后台更新。

对于每个关系 R，DBMS 维护以下信息：

- NR:R 中元组的个数
- V (A, R):属性 A 中不同值的个数

有了上面列出的信息，优化器可以推导出*选择基数*SC(A，R)统计量。选择基数是给定 $V(A,R)$ 的属性值的平均记录数。n^R 注意

这假设了数据的一致性。这种假设往往是不正确的，但它简化了优化过程。

选择统计

选择基数可用于确定将为给定输入选择的元组的数量。

唯一键上的相等谓词很容易估计(参见图 8)，图 9 显示了一个更复杂的谓词。

```
SELECT * FROM people
 WHERE id = 123
```

```
CREATE TABLE people (
  id INT PRIMARY KEY,
  val INT NOT NULL,
  age INT NOT NULL,
  status VARCHAR(16)
);
```

图 8:简单谓词示例——在这个示例中，确定使用哪个索引很容易，因为查询包含一个唯一键上的相等谓词。

```
SELECT * FROM people
 WHERE val > 1000
```

```
SELECT * FROM people
 WHERE age = 30
   AND status = 'Lit'
   AND age+id IN (1,2,3)
```

图 9:复杂谓词示例—更复杂的谓词(如范围或连词)更难估计，因为谓词的选择基数必须以非平凡的方式组合。

谓词 P 的*选择性*(sel)是符合条件的元组的比例。用来计算选择性的公式取决于谓词的类型。复杂谓词的选择性很难准确估计，这可能会给某些系统带来问题。选择性计算的一个例子如图 10 所示。

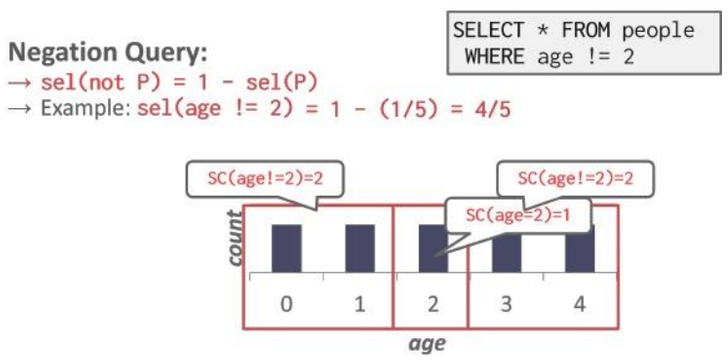

图 10:否定查询的选择性示例—通过从 1 中减去肯定查询的选择性来计算否定查询的选择性。在例子中，答案出来是 5-4，这是准确的。

注意，一个谓词的选择性等同于该谓词的概率。这使得概率规则可以应用在许多选择性计算中。这在处理复杂谓词时特别有用。例如，如果我们假设一个连词中涉及的多个谓词是*独立*的，那么我们可以将该连词的总选择性计算为单个谓词的选择性的乘积。

选择性计算假设

在计算谓词的选择基数时，使用了以下三个假设。

- **均匀数据**:值的分布(除了重命中者)是相同的。
- **独立谓词**:属性上的谓词是独立的。
- **包含原则**:连接键的域重叠，这样内部关系中的每个键也将存在于外部表中。

这些假设往往不被真实的数据所满足。例如，*相关属性*打破了谓词独立性的假设。

4 柱状图

真实数据往往是有偏差的，很难做出假设。然而，存储数据集的每一个单独的值是昂贵的。通过将数据存储在*直方图*中对值进行分组来减少内存使用量的一种方法。图 11 展示了一个带有桶的图的例子。

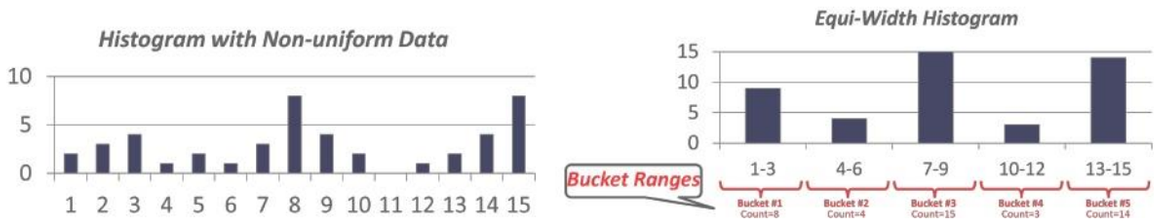

图 11:等宽直方图:第一个图显示了整个数据集的原始频率计数。第二张图是等宽直方图，它将相邻键的计数组合在一起，以减少存储开销。

另一种方法是使用*等深度*直方图，改变桶的宽度，使每个桶的总出现次数大致相同。图 12 展示了一个例子。

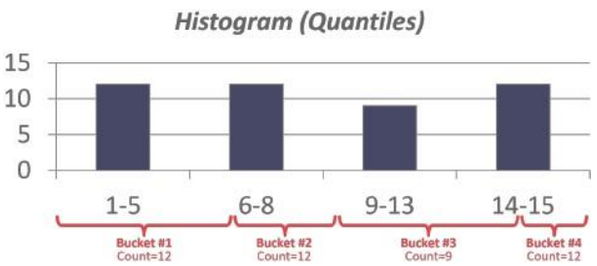

图 12:等深度直方图——为了确保每个桶具有大致相同数量的计数，直方图改变了每个桶的范围。

代替直方图，一些系统可能会使用*草图*来生成一个数据集的近似统计。

5 个抽样

DBMS 可以使用*抽样*将谓词应用到具有类似分布的表的较小副本上(参见图 13)。每当对底层表的更改量超过某个阈值(例如，元组的 10%)时，DBMS 就会更新样本。

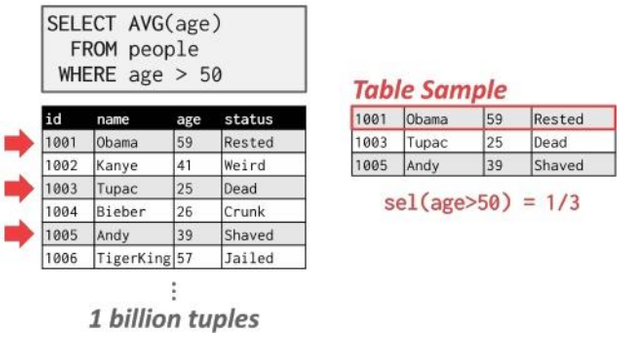

图 13:抽样——DBMS 可以从原始表的一个子集派生出谓词的选择性，而不是使用表中的 10 亿个值来估计选择性。

6 计划枚举

在执行基于规则的重写之后，DBMS 将列举查询的不同计划并估计它们的成本。然后在耗尽所有计划或某些超时后，它将为查询选择最佳计划。

7 单关系查询计划

对于单关系查询计划，最大的障碍是选择最佳的访问方法(即顺序扫描、二分查找、索引扫描等)大多数新的数据库系统只是使用启发式方法，而不是复杂的成本模型来选择访问方法。

对于 OLTP 查询，这尤其容易，因为它们是 *sargable*(Search Argument Able)，这意味着存在一个可以为查询选择的最佳索引。这也可以用简单的启发式方法来实现。

8 多关系查询计划

对于多关系查询计划，随着连接数量的增加，备选计划的数量也会迅速增长。因此，重要的是要限制搜索空间，以便能够在合理的时间内找到最优计划。有两种方法来解决这个搜索问题:

- **自下而上**:从零开始，然后制定计划，达到你想要的结果。例如:IBM System R, DB2, MySQL, Postgres，大多数开源 dbms。
- **自上而下**:从你想要的结果开始，然后沿着树向下工作，找到能让你达到目标的最佳计划。例如:MSSQL, Greenplum, CockroachDB, Volcano

9 自下而上优化示例—System R

使用静态规则执行初始优化。然后使用动态规划，使用分而治之的搜索方法来确定表的最佳连接顺序。

- 将查询分解成块，并为每个块生成逻辑操作符
- 对于每个逻辑运算符，生成一组实现它的物理运算符
- 然后，迭代地构建一个“左深”树，以最小化执行计划的估计工作量

10 自上而下的优化示例-火山

从我们想要查询的逻辑计划开始。通过将逻辑运算符转换为物理运算符，执行分支限界搜索来遍历计划树。

- 在搜索过程中跟踪全局最佳计划。
- 在规划时将数据的物理属性视为一等实体。