# Talent-Plan Section 3

宋阳

北京航空航天大学

Email: yangsoonlx@gmail.com

**课程目标**：熟悉数据库基础知识

## 1 课程作业

select $t_1.a$, count(*), avg($t_1.b$) from $t_1$ left outer join $t_2$ on $t_1.a = t_2.a$ group by $t_1.a$，请给出所有可能的逻辑执行计划 (画出 Plan 树)，并分析 $t_1$ 的数据分布对各种逻辑执行计划执行性能的影响。查询的初步逻辑查询计划如下图所示:

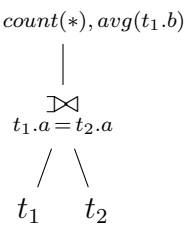


图 1-1 初步逻辑查询图

## 2 执行计划分析

查询编译可以分为 3 个步骤: **1. 建立查询的分析树 2. 查询重写 3. 物理计划生成**。本文的行文思路是先考虑不同表属性情况下，对初始的分析树1-1 进行查询重写，然后针对各个查询树进行物理计划生成分析。

如果优化器想判断哪个查询计划执行最快，就必须估计代价，在本文我们使用了一些描述操作符代价的参数。对于一个关系 $R$, **在本文我们假设关系 $R$ 是聚集的**:

1. $B(R)$ 表示关系 $R$ 所需磁盘块的个数，$M$ 表示关系 $R$ 可以获取到的内存块的个数。
2. $T(R)$ 表示关系 $R$ 中元组的个数。
3. $V(R, a)$ 表示关系 $R$ 中属性 $a$ 中不同值的个数。

对于 sql 的查询分析树中的一些关系操作，我们使用操作符号进行代替，如: $\gamma$(分组操作符)，$\pi$(投影操作符)，$\delta$(消除重复) $\bowtie$(左外连接)。

逻辑优化主要是基于规则的优化，数据库逻辑优化规则包括：列裁剪，最大最小消除，投影消除，谓词下推等等。本次课程作业的 sql 语句包括连接和聚合操作，针对这类 sql 语句可以使用聚合消除，外连接消除等逻辑优化规则。逻辑优化会针对 sql 语句中算子的不同性质进行不同的优化操作。所以接下来开始讨论在下面这几种不同的组合下可以进行的逻辑优化操作。

1. 属性 $t_1.a$ 在表 $t_1$ 中不具有唯一性，属性 $t_2.a$ 在表 $t_2$ 中也不具有唯一性；
2. 属性 $t_1.a$ 在表 $t_1$ 中具有唯一性，属性 $t_2.a$ 在表 $t_2$ 中不具有唯一性；
3. 属性 $t_1.a$ 在表 $t_1$ 中不具有唯一性，属性 $t_2.a$ 在表 $t_2$ 中具有唯一性；
4. 属性 $t_1.a$ 在表 $t_1$ 中具有唯一性，属性 $t_2.a$ 在表 $t_2$ 中具有唯一性；

## 2.1 属性 $t_1.a$ 和属性 $t_2.a$ 均不具有唯一性

**逻辑执行优化**：第一种组合是最普通的组合，即属性 $t_1.a$ 在表 $t_1$ 中不具有唯一性，属性 $t_2.a$ 在表 $t_2$ 中也不具有唯一性。这时候，只能对图1-1中的最初始的执行计划进行优化：从 sql 语句中可以看到，最终的输出结果只涉及到了表 $t_1$ 中的属性 a 和属性 b，表 $t_2$ 只是用来做连接操作的。所以我们使用列裁剪优化规则，裁剪掉用不上的列。优化后的逻辑查询计划如图2-2所示:

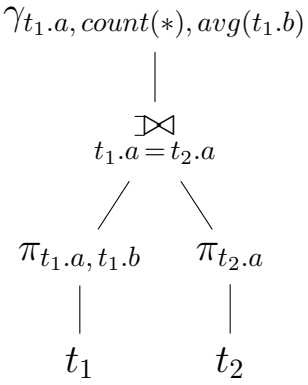


图 2-2 属性 $t_1.a$ 和属性 $t_2.a$ 均不具有唯一性时的逻辑查询图

**物理执行分析**:

1. 通过逻辑执行优化，我们通过列裁剪的方式，只选用上层算子需要的属性，投影不能减少 $T(t_1)$ 值，但是可以使得读取的元组大小被有效的缩减，为了上层的 Left Outer Join 减少内存需求。

2. 当执行到 Left Outer Join 时:

(1) 当 $B(t_2) < M$, 我们可以使用一趟算法, 将关系 $t_1$ 放到 $M-1$ 的内存块中, 依据 $t_2.a$ 构造适当的查询结构, 将 $t_1$ 按块读入剩下的块中, 用 $t_1.a$ 在 $M-1$ 块中查找匹配的值。这个过程需要 $B(t_1)+B(t_2)$ 次磁盘 IO。

(2) 当 $B(t_2) > M$, 我们可以使用 Hash Join，Sort Merge Join，Nested-Loop Join。这里主要讨论 Hash Join 和 Sort Merge Join。

**Hash Join** 因为使用的是 Left Outer Join, 我们只能使用 $t_2$ 作为 Inner 表, 在上面构建哈希表。左表作为驱动表进行哈希匹配, Hash Join 适合在 Join 连接项不是索引的情况下。大约需要 $3*(B(t_1)+B(t_2))$ 次磁盘 IO。

**Sort Merge Join**

**当 $t_1.a$ 和 $t_2.a$ 均不是索引的时候**, 分为 2 阶段: 第一阶段, 我们将关系 $t_1$ 和 $t_2$ 分别划分为 $B(t_1)/M$ 和 $B(t_2)/M$ 个子表，在 $B(R)+B(S) \leq M^2$ 的前提下，对每个子表进行排序，存储到外存上; 第二阶段，把每个子表加载到内存中，因为子表总个数小于 $M$, 我们可以在子表上进行归并排序进行匹配连接，最后输出结果。大约需要 $3*(B(t_1)+B(t_2))$ 次磁盘 IO。

**当考虑到索引，就涉及到基于索引的连接**。

**当 $t_1.a$ 和 $t_2.a$ 均是索引的时候**，就不需要第一阶段的排序过程，可以直接从内存中读取索引，直接比较两个关系的索引值，但因为是 Left Outer Join，左表可能需要去磁盘读取所有的值，当匹配值相等时左表去磁盘取出相应的值；也可能不需要进行磁盘 IO，因为如果 $t_1.a$ 是二级索引，$t1.b$ 是聚簇索引，左表不需要进行磁盘读写，直接从二级索引中就能返回 $t1.b$ 的值。直接内存操作就可以，若不能把索引都加载到内存中，也需要极少的磁盘 IO。

**当 $t_1.a$ 和 $t_2.a$ 中有一个是索引的时候**，这种情况和上面类似，可以避免进行对关系的排序，减少磁盘 IO。

3. 分组聚合

**基于一趟算法的分组聚合**: 对于中间聚合结果 $\underset{t_1.a=t_2.a}{\bowtie}$，如果 $B(\delta(\underset{t_1.a=t_2.a}{\bowtie})) < M$, 表示我们可以把每一种 $t_1.a$ 的项存在内存中用来统计每个分组中的 $sum(t_1.b)$ 和 $count(*)$，然后留出一块内存，用来遍历关系 $\underset{t_1.a=t_2.a}{\bowtie}$，统计结果，直到扫描到最后一个块时，才输出结果。需要 $B(\underset{t_1.a=t_2.a}{\bowtie})$ 磁盘 IO。

**基于 hash 的分组聚合**: 我们先将 $\underset{t_1.a=t_2.a}{\bowtie}$ 的所有元组散列到 $M-1$ 个桶里,只要满足 $B(\underset{t_1.a=t_2.a}{\bowtie}) < M^2$, 就可以在内存中处理每一个桶，计算每个桶中的分组聚合数目，并输出结果，需要 $3*B(\underset{t_1.a=t_2.a}{\bowtie})$ 次的磁盘 IO。

**基于排序的分组聚合**: 如果之前进行 Left Outer Join 时候选择的是 Sort Merge Join，那么 $\underset{t_1.a=t_2.a}{\bowtie}$ 就是按 $t_1.a$ 顺序存储，可以将 $B(\underset{t_1.a=t_2.a}{\bowtie})$ 划分为 $B(\underset{t_1.a=t_2.a}{\bowtie})/M$ 块，每次读入子块，检测每个排序关键字为 $v$ 的元组，进行累计所需聚集，如果内存块空了，就继续读下一个子块。只需要 $B(\underset{t_1.a=t_2.a}{\bowtie})$ 次的磁盘 IO。

## 2.2 属性 $t_1.a$ 具有唯一性，属性 $t_2.a$ 不具有唯一性

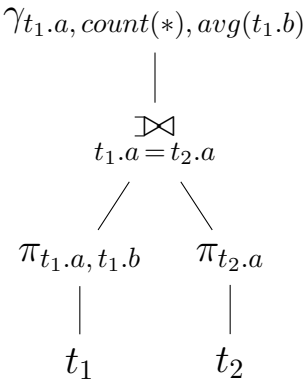


图 2-3 属性 $t_1.a$ 具有唯一性，属性 $t_2.a$ 不具有唯一性时的逻辑查询图

**逻辑执行优化**: 这种情况和 2.1 的情况一样，逻辑执行计划和图2-2一致，但是有一点不同之处，因为 $t_1.a$ 的属性是唯一性的，所以对应的 $t_1.b$ 也是唯一的，其实数据库在进行 avg 运算的时候，可以直接输出 $t1.b$。

**物理执行分析**: 关于物理执行分析和 2.1 的也一致，在此就不再赘述。

## 2.3 属性 $t_1.a$ 不具有唯一性，属性 $t_2.a$ 具有唯一性

**逻辑执行优化**: 当 $t_2.a$ 具有唯一性属性时，sql 语句中的左外连接算子可以进行消除。对于表 $t_1$ 的每一行来说，经过左外连接之后，最终只产生一行连接结果，而且，最终输出的结果只需要表 $t_1$ 中的属性。因此，优化后的逻辑查询结果如图2-4 所示：

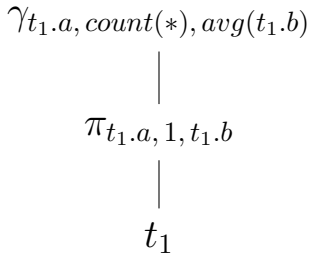


图 2-4 属性 $t_1.a$ 不具有唯一性，属性 $t_2.a$ 具有唯一性时的逻辑查询图

**物理执行分析**: 在本节条件下，我们考虑选择一个合适的分组聚合算法：

**基于一趟算法的分组聚合**: 如果 $B(\delta(t_1)) < M$, 表示我们可以把每一种 $t_1.a$ 的项存在内存中用来统计每个分组中的 $sum(t_1.b)$ 和 $count(*)$，然后留出一块内存，用来遍历关系 $t_1.a$，统计结果，直到扫描到最后一个块时，才输出结果，需要 $B(t_1)$ 次磁盘 IO。

**基于 hash 的分组聚合**: 如果 $B(\delta(t_1)) > M$, 我们先将 $t_1$ 的所有元组散列到 $M-1$ 个桶里，只要满足 $B(t_1) < M^2$, 就可以在内存中处理每一个桶，计算每个桶中的分组聚合数目，并输出结果，需要 $3 * B(t_1)$ 次的磁盘 IO，基于 hash 的分组聚合，适合在 $t_1.a$ 非索引的情况下。

**基于排序的分组聚合**:

当 $B(\delta(t_1)) > M$，$t_1.a$ **不是索引的时候**，同样分为 2 个阶段：首先，我们将 $t_1$ 划分为 $B(t_1)/M$ 个子块，分别对每个子块进行排序；第二阶段，将每个子表的第一块加载到内存中，然后查找第一个最小的值 $v$，并使用 $v$ 进行聚集计算。当某一子块的内存块读取完毕就读取下一块到内存，需要 $3 * B(t_1)$ 次磁盘 IO，而且需要保证 $B(t_1) < M^2$。

当 $B(\delta(t_1)) > M$，$t_1.a$ **是索引的时候**，如果 $t_1.a$ 是聚簇索引，所以 $t_1$ 是按照 $t_1.a$ 顺序排序，之后的处理和 2.1 的**基于排序的分组聚合**处理类似。如果 $t_1.a$ 是二级索引，$t_1.b$ 是聚簇索引，就需要很少的磁盘 IO，在内存中处理就可以。

## 2.4 属性 $t_1.a$ 和属性 $t_2.a$ 均具有唯一性

**逻辑执行优化**: 首先根据 2.3 的结果，我们可以做外连接消除优化。又因为 $t_1$ 也具有唯一性，所以可以继续做聚集消除操作。因此，优化后的逻辑查询结果如图2-5所示：

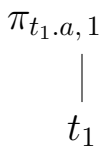


图 2-5 属性 $t_1.a$ 和属性 $t_2.a$ 均具有唯一性时的逻辑查询图

**物理执行分析**: 在这种情况下，如果在没有索引的情况下，就是简单的做扫描表关系就行。如果 $t_1.a$ 是二级索引，$t_1.b$ 是聚簇索引。就可以直接在内存处理，减少磁盘 IO。