

专注，勤学，慎思。戒骄戒躁，谦虚谨慎

just do it

导航

博客园

首页

新随笔

订阅

管理

2023年12月							>
日	一	二	三	四	五	六	
26	27	28	29	30	1	2	
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	29	30	
31	1	2	3	4	5	6	

统计

随笔 - 193

文章 - 0

评论 - 349

阅读 - 84万

随笔分类 (211)

Consul(1)

ETL(1)

MongoDB(3)

MySQL 8.0(8)

MySQL DDL(3)

MySQL ErrorLog(3)

MySQL 备份还原(1)

MySQL 高可用(5)

MySQL 管理(12)

MySQL 基础(11)

MySQL 体系结构(7)

MySQL 优化(18)

MySQL 运维(6)

MySQL 主从(5)

PostgreSQL 管理(4)

PostgreSQL 基础(3)

PostgreSQL优化(7)

Python(10)

Redis(25)

SQL Server 管理(16)

SQL Server 基础(10)

SQL Server 权限(2)

SQL Server 优化(23)

SQLServer 复制(1)

T-SQL(12)

Zabbix(1)

其他(1)

数据库设计(6)

死锁(6)

随笔档案 (193)

2023年9月(1)

2023年8月(1)

2023年1月(1)

2022年12月(1)

2022年8月(1)

2022年4月(2)

2021年8月(2)

2020年12月(3)

2020年11月(3)

2020年8月(3)

2020年7月(8)

2020年6月(1)

2020年5月(4)

2020年4月(4)

2020年3月(2)

2020年1月(2)

2019年12月(4)

2019年11月(3)

2019年10月(2)

2019年9月(6)

2019年8月(3)

2019年7月(3)

2019年6月(1)

2019年5月(5)

2019年1月(3)

2018年12月(1)

2018年11月(3)

2018年10月(4)

2018年9月(7)

2018年8月(4)

2018年7月(3)

2018年6月(8)

2018年5月(5)

2018年4月(1)

2018年3月(2)

2018年2月(1)

2018年1月(3)

2017年11月(6)

2017年10月(2)

2017年9月(4)

更多

最新评论

1. Re:SQLServer中的执行计划缓存由于长时间缓存对性能造成的干扰

@catcherx 可以清理对应的执行计划缓存，如果这一块不清楚的话，那就重启吧...

--MSSQL123

2. Re:SQLServer中的执行计划缓存由于长时间缓存对性能造成的干扰

@MSSQL123 @zwei16888 应该是所谓的parameter sniff问题，最简单的做法，找到对应的存储过程中的sql语句，在潜在问题的sql语句上加上option(recompile)...

--catcherx

3. Re:SQL Server数据库的存储过程中定义的临时表，真的有必要显式删除（drop table #tableName）吗？

SQL Server数据库的存储过程中定义的临时表

--流畅的心情

4. Re:MySQL中建表时空（NULL）和非空（NOT NULL）的一些思考

专门登录顶一个，非常不能接受有些人无脑添加的这些限制，都是意淫的道理。

--iamstrong01

5. Re:MySQL执行计划extra中的using index 和 using where using index 的区别

厉害了

--无解_123

MySQL 8.0中的 explain analyze (译)

原文地址：<https://mysqlservteam.com/mysql-explain-analyze/>

MySQL 8.0.18刚刚发布(译者注：原文发表时间为[October 17, 2019](#))，它包含了一个全新的特性来分析和理解查询是如何执行的:explain analyze。

explain analyze是什么

EXPLAIN ANALYZE是一个查询分析工具，它会告诉你MySQL在查询上花了多少时间以及原因。它将计划查询、度量查询并执行查询，同时计算行数并测量在执行计划中不同阶段花费的时间。当执行完成时，EXPLAIN ANALYZE将打印计划和度量结果，而不是查询结果。(译者注：直白地说就是，explain analyze会真是地执行当前的查询，返回的执行计划以及代价信息，但是不会返回查询自身的结果)

这个新特性是在常规的EXPLAIN查询计划检查工具之上构建的，可以看作是先前在MySQL 8.0中添加的explain forat = tree的扩展。除了普通的explain将打印的查询计划和估计成本之外，explain analyze还将输出执行计划中单个迭代器的实际成本。

如何使用explain analyze

作为一个示例，我们将使用来自Sakila Sample数据库的数据和一个查询，该查询列出了每个员工在2005年8月完成的工作总量。这个问题很简单：




```
SELECT first_name, last_name, SUM(amount) AS total
FROM staff INNER JOIN payment
      ON staff.staff_id = payment.staff_id
      AND
      payment_date LIKE '2005-08%'
GROUP BY first_name, last_name;
```




first_name	last_name	total
Mike	Hillyer	11853.65
Jon	Stephens	12218.48

2 rows in set (0.02 sec)


只有两个人，Mike和Jon，我们在2005年8月得到了他们每个人的总数，EXPLAIN FORMAT=TREE 将会显示执行计划和成本信息




```
1 EXPLAIN FORMAT=TREE
2 SELECT first_name, last_name, SUM(amount) AS total
3 FROM staff INNER JOIN payment
4   ON staff.staff_id = payment.staff_id
5   AND
6   payment_date LIKE '2005-08%'
7 GROUP BY first_name, last_name;
8
9 -> Table scan on <temporary>10      -> Aggregate using temporary table
11      -> Nested loop inner join      (cost=1757.30 rows=1787)
12      -> Table scan on staff      (cost=3.20 rows=2)
13      -> Filter: (payment.payment_date like '2005-08%')      (cost=117.43 rows=894)
14      -> Index lookup on payment using idx_fk_staff_id (staff_id=staff.staff_id)      (cost=117.43 rows=8043)
```



但是它没有告诉我们这些估计是否正确，或者查询计划中的哪些操作实际花费了时间。 EXPLAIN ANALYZE可以做到这一点:



```
1 EXPLAIN ANALYZE
2 SELECT first_name, last_name, SUM(amount) AS total
3 FROM staff INNER JOIN payment
4   ON staff.staff_id = payment.staff_id
5   AND
6   payment_date LIKE '2005-08%'
7 GROUP BY first_name, last_name;
8
9 -> Table scan on <temporary>      (actual time=0.001..0.001 rows=2 loops=1)
10      -> Aggregate using temporary table      (actual time=58.104..58.104 rows=2 loops=1)
11      -> Nested loop inner join      (cost=1757.30 rows=1787) (actual time=0.816..46.135 rows=5687 loops=1)
12      -> Table scan on staff      (cost=3.20 rows=2) (actual time=0.047..0.051 rows=2 loops=1)
13      -> Filter: (payment.payment_date like '2005-08%')      (cost=117.43 rows=894) (actual time=0.464..22.767 rows=2844 loops=2)
14      -> Index lookup on payment using idx_fk_staff_id (staff_id=staff.staff_id)      (cost=117.43 rows=8043) (actual time=0.450..19.988 rows=8024 loops=2)
```



这里有一些新的衡量方法:

- 获取第一行的实际时间(毫秒)
- 获取所有行的实际时间(以毫秒为单位)
- 读取的实际行数
- 实际循环次数

让我们看一个具体的例子，筛选迭代器的成本估计和实际度量，筛选迭代器选择了2005年8月的销售(上面的EXPLAIN ANALYZE输出中的第13行)。

```
Filter: (payment.payment_date like '2005-08%')
(cost=117.43 rows=894)
(actual time=0.464..22.767 rows=2844 loops=2)
```

过滤器的估计成本为117.43，估计返回894行，这些估计是查询优化器在执行查询之前根据可用的统计信息做出的。该信息也以EXPLAIN FORMAT=TREE输出的形式出现。

从循环数开，此筛选迭代器的循环次数为2，这是什么意思?要理解这个数字，我们必须查看查询计划中过滤迭代器上面的内容。

在第11行有一个嵌套循环联接，在第12行有一个对Staff表的表扫描。

这意味着我们正在执行一个嵌套循环联接，其中我们扫描Staff表，对于表中的每一行，我们使用索引查找和对付款日期进行筛选来查找付款表中相应的行。因为staff表中有两行(Mike和Jon)，我们对过滤和第14行上的索引查找进行了两次循环迭代。

对于很多人来说，EXPLAIN ANALYZE提供的一个有趣的信息是实际消耗时间，“0.464.22.767”，这意味着读取第一行平均需要0.464 ms，读取所有行平均需要22.767 ms。

是平均值吗?是的，因为循环，我们必须对迭代器计时两次，报告的数字是所有循环迭代的平均值。

这意味着过滤的实际执行时间是这些数字的两倍，因此，如果我们查看在嵌套循环迭代器(第11行)中接收所有行所需的时间，它是46.135 ms，比一次运行过滤迭代器所需的时间多一倍多。

译者注：

这里的时间成本计算规律就是，每一步的执行时间，是包含了其子步骤的执行时间的之和，这几个步骤的时间包含关系是这样的：

Nested loop inner join这一层总的的时间是58.104ms，也就是整各join的时间成本，包含了

“Table scan on staff表”和“payment表上的Filter的时间”

filter的时间又包含了：“index lookup” + “where条件filter条件”的时间，其中最耗时的就是index lookup这一步，也即数据查询的过程。

Index lookup 这一步的时间是19.988*2，乘以2意思是两次循环迭代，因此整个loop join过程的时间大部分都耗费在这个index lookup这个查找上，平均每次（两次）Filter（22.767）= payment_date like '2005-08%'的筛选 + Index lookup on payment 查找（19.988）

实际读取的行数为2844，而估计值为894行。优化器漏掉了一个因子3（译者注：这句话不太明白是什么意思，漏掉了什么）。同样，由于循环的原因，估计的和实际的数字都是所有循环迭代的平均值。

如果我们查看表结构，payment_date列上没有索引或直方图，因此提供给优化器用于计算筛选器选择性的统计信息是有限的。

对于更好的统计信息会产生更准确的估计的示例，我们可以再次查看索引查找迭代器。我们看到索引提供了更精确的统计数据:8043行与8024行实际读取的比较。

这很好，出现这种情况是因为索引附带了额外的统计信息，而非索引则没有。

那么你能利用这些信息做些什么呢?分析查询并理解为什么它们执行得不好需要一些实践。但一些简单的提示，让你开始:

- 如果你想知道为什么花了这么长时间，看看时间，执行的时候时间都花费在哪一步？
- 如果你想知道为什么优化器选择了该计划，请查看行计数器。估算的行数与实际的行数之间有很大的差异（即几个数量级或更多），这表明您应该仔细看一下。优化器根据估计值选择计划，但是查看实际执行情况可能会告诉您另一个计划会更好。

如果您想知道优化器为什么选择该计划，请查看行计数器。巨大的差异。在估计的行数和实际行数之间的几个数量级或更多)是一个标志，表明您应该更仔细地查看它。优化器根据估计值选择计划，但是查看实际执行情况可能会告诉您另一个计划会更好。

就是这样！MySQL查询分析工具箱中的另一个工具：

- 要检查查询计划：EXPLAIN FORMAT=TREE
- 要跟踪查询执行：EXPLAIN ANALYZE
- 要理解计划选择：Optimizer trace

我希望您喜欢这个新特性的快速浏览，解释分析将帮助您分析和理解慢速查询。

译者补充：

关于MySQL执行计划的几种展示方式，explain/explain format=tree/explain format=json/optimizer_trace

其实本质上都是一样的，只是详细程度不一样，对于explain analyze可以同时显示预估的+实际执行的信息，以下是将译文中的示例数据库导入到本地后，展示出来的一些信息，与上文中的信息稍有差异。

1, explain

最简洁或者粗略的执行计划显示方式，可以显式：表的访问方式、表之间的驱动顺序，以及Extra列中的其他信息，包括是否产生排序，使用临时表空间等等。

2, expalin format = tree

与explain analyze类似，同时包含了以预估的每一步的代价信息，仅仅是预估信息，并不包含实际执行信息



```
1 -> Table scan on <temporary>
2      -> Aggregate using temporary table
3      -> Nested loop inner join      (cost=1757.30 rows=1787)
4      -> Table scan on staff      (cost=3.20 rows=2)
5      -> Filter: (payment.payment_date like '2005-08%')      (cost=117.43 rows=894)
6      -> Index lookup on payment using idx_fk_staff_id (staff_id=staff.staff_id)      (cost=117.43 rows=8043)
```

3, explain format = json

以json的格式显式与expalin format = tree的信息类似，说实话，可读性并不入expalin format = tree



```
1 {
2   "query_block": {
3     "select_id": 1,
4     "cost_info": {
5       "query_cost": "1757.30"
```

0

发表评论

其实这些信息，都是跟explain format = json或者说explain analyze中，预估部分是一致的，这些数据都跟expalin format = tree一致，只不过trace中会枚举出来标访问时候每种可能性。

2/2