SQL查询之执行顺序解析

2017-03-23 | Database | 178

一直对SQL的理解就处于半生不熟的状态，前段时间阿里面试三面的时候问道让写一个SQL语句，统计每个学生所有成绩平均分大于80分的结果，当时差不多能写出来，但是多少心里没底，索性就想好好温习一下SQL语言，不料发现几篇解释SQL语句执行逻辑顺序的文章，感觉挺有意思，仔细看完觉得困惑减少很多。想来以前我确实没有关注这个点，以前大学学习SQL仿佛也就是镜中花水中月一般，只是知道这么回事压根不清楚用在何处，每次写SQL都是从SELECT开始，都是凭感觉一蹴而就，对错与否也并不自知，这实在是件很可怕的事情。

去年的时候一时兴起注册了阿里天池比赛的账号，当时的数据分析语言就提供了SQL一项，然鹅当时的SQL知识早已归还老师，最后弃，书到用时方恨少啊。学习这件事还真是一件件积累而成，也需要有兴趣不断的去探索才能发现更有意思的东西。

言归正传，本文试图总结一下关于SQL查询中的执行顺序，方便以后温故知新。

SQL查询执行顺序

以下是常见的SQL语句查询逻辑执行顺序，从多个地方考证过，应该没有问题，如果谁能从数据库源码分析的角度给出验证就更好了，序号则为实际执行顺序：

```
(7)     SELECT
(8)     DISTINCT <select_list>
(1)     FROM <left_table>
(3)     <join_type> JOIN <right_table>
(2)     ON <join_condition>
(4)     WHERE <where_condition>
(5)     GROUP BY <group_by_list>
(6)     HAVING <having_condition>
(9)     ORDER BY <order_by_condition>
(10)    LIMIT <limit_number>
```

执行顺序简介

在SQL语句的执行过程中，每一步都会产生一个虚拟表（Virtual Table，简称VT），用来保存SQL语句的执行结果，以下是上述SQL的执行顺序。

执行FROM语句

第一步，执行FROM语句。我们首先需要知道最开始从哪个表开始的，这就是FROM告诉我们的。经过FROM语句对两个表执行笛卡尔积，会得到一个虚拟表，暂且叫VT1。总共有——table1的记录条数 * table2的记录条数——条记录，这就是VT1的结果。

执行ON过滤

执行完笛卡尔积以后，接着就进行ON join_condition条件过滤，比如ON a.customer_id = b.customer_id，根据ON中指定的条件，去掉那些不符合条件的数据，得到VT2表。

添加外部行（外联结）

这一步只有在连接类型为OUTER JOIN时才发生，如LEFT OUTER JOIN（左连接）、RIGHT OUTER JOIN（右连接）和FULL OUTER JOIN（经过测试Mysql不支持该连接方式）。大多数时候会省略OUTER关键字。
添加外部行的工作就是在VT2表的基础上添加保留表中被过滤条件过滤掉的数据，非保留表中的数据被赋予NULL值，最后生成虚拟表VT3。
这个稍微难理解举个例子：

```
+-------------+----------+----------+-------------+
| customer_id | city     | order_id | customer_id |
+-------------+----------+----------+-------------+
| baidu       | hangzhou |     NULL | NULL        |
+-------------+----------+----------+-------------+
```

这一行数据要是在上一步ON过滤：ON a.customer_id = b.customer_id中是绝对会被过滤掉的，但是如果我们的外连接是LEFT JOIN,则需要补充这一行数据，成为我们的VT3。回顾一下左连接：

> LEFT JOIN 关键字会从左表 (table_name1) 那里返回所有的行，即使在右表 (table_name2) 中没有匹配的行。

接下来的操作都会在该VT3表上进行。

执行WHERE过滤

对添加外部行得到的VT3进行WHERE过滤，只有符合 where_condition 的记录才会输出到虚拟表VT4中。

执行GROUP BY分组

上面得到的虚拟表还没有经过聚合分组，**GROU BY子句主要是对使用WHERE子句得到的虚拟表进行分组操作。**得到的内容会存入虚拟表VT5中，此时，我们就得到了一个VT5虚拟表，接下来的操作都会在该表上完成。

执行HAVING过滤

这里需要注意的是到目前为止已经有了三种过滤，ON、WHERE和HAVING，三者在执行时间段上是有严格区别的，HAVING子句主要和GROUP BY子句配合使用，对分组得到的VT5虚拟表进行条件过滤，然后得到虚拟表VT6。

SELECT列表

从虚拟表VT6中选择出我们需要的内容，生成虚拟表VT7。

执行DISTINCT子句

如果在查询中指定了DISTINCT子句，则会创建一张内存临时表（如果内存放不下，就需要存放在硬盘了）。这张临时表的表结构和上一步产生的虚拟表VT7是一样的，不同的是对进行DISTINCT操作的列增加了一个唯一索引，以此来除重复数据。

执行ORDER BY子句

对虚拟表中的内容按照指定的列进行排序，然后返回一个新的虚拟表，上述结果会存储在VT8中。

执行LIMIT子句

LIMIT子句从上一步得到的VT8虚拟表中选出从指定位置开始的指定行数据。mysql的limit语法如下：

```
LIMIT n,m
```

从第n条记录开始，选择m条数据，用于分页场景较多。但是limit的性能在数据量稍微大一点的时候就会急剧下降，我用项目中19W多的数据做了一下实验：

```
[SQL]SELECT * FROM bas_detectionooperationinformation LIMIT 190000,10

受影响的行: 0
时间: 7.825s
```

这是第一次查询这个表所用时长,后续重复执行这条语句速度就很快了，我怀疑是数据库缓存执行结果原因，我再重新更换另外一张表，第一次执行依然是比较慢的。
但是执行下面的语句无论怎样速度都很快，数据量大的时候limit确实性能不好。

```
[SQL]SELECT * FROM bas_detectionooperationinformation LIMIT 0,10

受影响的行: 0
时间: 0.074s
```

在明确sql的执行逻辑基础上，写sql确实会思路清晰很多，本篇文章主要参考了下列文章，原文结合实验数据进行分析确实挺好，本文没有全部摘录，只取了有代表性的语言进行了总结。

参考

SQL逻辑查询语句执行顺序

-EOF-

#sql

浅析Serializable接口与transient关键字

ThreadPoolExecutor线程池源码分析