SQL查询操作处理流程

SQL查询

这里看到书上讲述SQL的查询处理，在SQL中查询是通过select语句以及一些辅助的子句来实现需要的输出，这里使用的是mysql，首先，要理解物理表和虚拟表的区别，物理表就是存储在文件系统上的一个或者多个文件，按照相应的存储数据结构将每一行的数据存储，虚拟表是我们在物理表的基础上构建出来的，可能是全部的表结构，也可能是表中的部分字段或者部分行，甚至可能是表中某些字段经过某种运算之后的结果。但是SQL语言不像C/C++/JAVA这类语言一样，它们的主要区别在于SQL给你一个需求，不关心它是怎么实现的，真正的实现由数据库完成，当然不同的数据库可能会有差别很大的实现方式，而C/C++这类的语言，我们必须一步步的实现，完成指定的功能。所以SQL的内部实现过程我们不用关心，但是对于查询请求，它的执行过程和执行顺序对我们对结果的认识有很大的影响，但是真正的执行过程需要深入数据库内部才能明白使用哪些方式（例如索引）优化，所以这里我们重点看一下查询请求每个子句的执行顺序。

标准的SQL查询语句的结果如下所示：

```
(8) SELECT x, xx, xxx, xxxx  \
(9)      DISTINCT <x, xx,...> \
(1)      FROM <table1, table2, table3,...>  \
(3)      <join_type>JOIN<join_table>  \
(2)      ON  <join_condition>   \
(4)      WHERE  <where_condition>  \
(5)      GROUP BY <group_by_list>  \
(6)      WITH[CUBE | ROLLUP]   \
(7)      HAVING  <having_condition>  \
(10)    ORDER BY <order_by_list>   \
(11)    LIMIT <limit_number> OFFSET <offset_number> ;
```

每个子句执行的次序已经在前面进行了标记，在一一介绍这些子句之前，我们要先明白，执行这个命令的输入是这些表名、条件（ON、WHERE和HAVING）以及一些字段名（SELECT，DISTINCT、ORDER BY、GROUP BY），其中表名标识的是物理存储的表，字段名是每一个表的字段的名称，条件是针对字段的或者该字段上执行的聚集函数的值得逻辑表达式。这些子句的每一步都是产生一个逻辑关系表。

执行流程

FROM操作

这一步的输入是多个表，在这一步是对多个关系表执行笛卡尔积操作，生成的结果是VT1，假设有3个表，t1、t2和t3，三个表分别有r1、r2、r3行，每个表有c1、c2、c3个字段，那么执行笛卡尔积得到的VT1则有r1*r2*r3行，有c1+c2+c3个字段。

ON子句

这一步是在第一步生成的VT的基础之上执行的判断操作，ON后面的逻辑表达式是一个针对VT1表上的每一行的判断，我们将淘汰不满足ON条件的行，但是一般情况下，我们执行逻辑操作返回的结果总是TRUE或者FALSE，但是在SQL中，可能存在第三个值，那就是NULL，这个一般被认为空（注意，不是空字符串），它既不是TRUE也不是FALSE，我们来看： mysql> select NULL = NULL \G *************************** 1. row *************************** NULL = NULL: NULL

mysql> select 0 = NULL \G *************************** 1. row *************************** 0 = NULL: NULL

mysql> select 1 = NULL \G *************************** 1. row *************************** 1 = NULL: NULL

mysql> select NULL != NULL \G *************************** 1. row *************************** NULL != NULL: NULL

所以，我们可以把NULL视为未知状态，所以两个未知状态也不是相等的。但是在条件判断的时候，我们要分出TRUE或者FALSE，因为我们要根据这个结果判断是否淘汰这一行，在ON子句中，是将NULL视为FALSE的，所以如果某一行的该字段是NULL，那么它将会被淘汰。例如：

接着，我们看一下将这一个表执行完笛卡尔积之后再执行ON操作得到的结果：

可以看出，在执行笛卡尔积之后生成的VT1表应该含有9行，其中t2.b应该包含‘hello’、‘world和’NULL三种，这里只返回了‘world’，所以在判断NULL != 'hello'的时候虽然返回的是NULL，但是ON子句把它当做了FALSE。

所以在ON子句的操作之后，得到了表2，这里面只包含满足ON后面逻辑操作计算得到TRUE的行（淘汰计算结果为FALSE或者NULL的行）。

添加外部行

这一步执行的是添加外部行的操作，这里针对的是外部JOIN的，因为外部JOIN需要保证进行连接的左边的表或者右边的表中每一行都出现在VT2中，但是如果在ON操作的时候这一行被淘汰了呢？那么就进行补充添加，仅仅根据左连接或者右连接添加上左边或者右边表中缺失的行，然后其它的字段补充NULL就可以了，得到了VT3。

可以从上例中看到，在笛卡尔积计算之后，一共产生了9行，然后执行ON操作，只剩下了3行满足条件的，而条件是t2.b!='hello'，这就导致了t2表中只剩下了b='hello'的那一行，如上上一个图所示，但是如果执行了右外连接的操作，那么需要添加t2表（因为t2是连接操作右边的那个表）中缺失的每一行（这里需要添加两行），对于这两行除了t2字段外的其他字段，都补充NULL就可以了。如果将 上例中的t1和t2换一下，然后再讲right join换成left join，可以得到类似的结果，只不过补充的NULL出现在是右边。

WHERE子句

这一步执行的是我们最熟悉的WHERE子句，这一步是在VT3的基础上执行条件过滤，同样，只保留条件计算得到TRUE的行，淘汰结果为FALSE和NULL的行。将返回的结果标记为VT4。

但是，在WHERE子句执行的时候需要注意一些问题：

- 由于WHERE是第四步执行的，而此时并没有执行GROUP BY子句，所以不能再WHERE子句中使用聚集函数运算之后的结果作为条件运算的输入，这将会导致执行出错。 mysql> select * from T1 as t1 right join T1 as t2 on t2.b != 'hello' where count(t1.b) != 1; ERROR 1111 (HY000): Invalid use of group function
- 由于在执行WHERE的时候并没有执行到SELECT操作，所以在SELECT操作中出现的as别名将不能用在WHERE子句中。 mysql> select t1.a as A from T1 as t1 right join T1 as t2 on t2.b != 'hello' A != 10; ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'A != 10' at line 1
- 要明确WHERE是在添加缺失行之后执行的，而ON执行在添加缺失行之前，所以如果在ON和WHERE使用相同的逻辑计算，得到的结果可能不是相同的。

GROUP BY子句

这一步执行的是分组的操作，在VT4的基础上，根据GROUP BY确定的列进行分组，在GROUP BY子句中会认为所有的NULL都是相同的，而不是NULL != NULL，将它们分在一组，例如：

WITH子句

with子句分为ROLLUP和CUBE两种，这两种我们基本上用不到，CUBE在mysql也不支持，ROLLUP只是在得到的结果添加额外的一行。

这里不探讨ROLLUP的作用了，假设它生成的结果为VT6

HAVING子句

执行HAVING子句，这个子句的作用也是将汇聚之后的每一组（在GROUP BY指定的列上相同的行为一组）进行判断，这是有区别与ON和WHERE子句的，它们都是对于上一个操作结果中的每一行进行判断，而HAVING是对每一组。这里可以是对一个或者多个字段的判断，也可以是每一个组中执行汇聚函数的判断。生成的结果是VT7。　　但是，在HAVING执行的时候需要注意，虽然在GROUP BY的时候将执行列为NULL的划分到同一组了，但是对于某一列为NULL的时候，执行这一列的count操作会被认为为0的，例如：

但是在计算count(*)的时候，某一行即使都是NULL，也会被算作一行的，例如：

这一点需要在具体的应用上特别注意，尤其是使用OUTER JOIN的时候会添加一些NULL，可能由于使用count(*)而不是count(字段名)导致不正确的计数。

SELECT子句

这一步执行的是SELECT操作，然后SELECT出现在查询语句的最前面，但是它是到了第八步才被执行，这一步是对VT7上的选择SELECT指定的列。生成VT8。另外，在SELECT中不能对前面字段的别名进行使用，也就是SELECT的别名只有在整个SELECT子句执行完之后才有效。

这一点需要在具体的应用上特别注意，尤其是使用OUTER JOIN的时候会添加一些NULL，可能由于使用count(*)而不是count(字段名)导致不正确的计数。

OFFSET...LIMIT子句

最后一步执行的是OFFSET ... LIMIT子句，这一步就是在上面生成的VT10表中，从指定的OFFSET开始选择指定的行数，OFFSET也可以省略，而写成LIMIT m, n.它的含义和LIMIT n OFFSET m是一样的。

总结

好了，一般的SQL查询语句都是按照这个顺序执行的，当然具体的逻辑查询和物理查询方案这里我们没有涉及，仅仅是介绍每一步的执行顺序和它的作用，当然其中还涉及一些mysql中的规则，以后使用SQL查询语句之前要首先理好这个顺序，然后再去考虑使用什么样的查询语句才能得到希望的结果。