黄超 李在弦

HSQL优化展示

数据分布分析

- Orders
  - ORDERS 表示订单的信息，总共有1,500,000行数据
  - 并不是所有的顾客都会有订单。事实上，数据库中大约1/3的用户不会有任何订单。这些订单随机的分配给2/3的用户。这样做的目的是为了在加入两个或更多的表时，数据库有能力操作“死数据”。
- Lineitem
  - LINEITEM 表示在线商品的信息，总共有6,001,215行数据
  - 数据量要远大于Orders

优化分析

- 对于第一层的查询条件，前两个条件是朴素条件
- 第三个查询需要物化子表， 并在子表上执行查询。
- 每次在外层执行循环都要在内层物化子表
- Exist条件或许有优化空间

尝试思路

- 优化Exist查询
- 修改查询语句，换为其他等价语句

代码分析-EXIST查询优化

- 对HSQL的源码进行阅读可以发现，在代码中，HSQL注册了一个isSingleRow变量，用于标注这次的查询是否只需要单一结果。但是却没有在代码中使用，留下了一个TODO Label。这会导致Exist语句每次都需要把整个子表遍历一遍，而不是发现结果就返回

代码分析-等价语句

- 根据论文《On Optimizing an SQL-like Nested Query》（WON KIM,1982），此类查询语句符合论文中提出的J类优化，可以将子查询转化为等价的多表连接查询
- 我们进行了等价转化，发现语句中有distinct，也许存在新的优化空间
- 对代码分析发现，HSQL在进行count操作时，对于distinct条件，他将查询结果先排序后再线性遍历统计，每次值变化一次则count+1，但是在那之前他已经用hash表处理过了，可以直接通过size获取distinct的值的数量

代码分析-EXIST

- 在org/hsqldb/QuerySpecification.java文件的L1663行加入判断单条查询的语句，计算从开始执行到结束执行的时间间隔

- 未优化前，数据缓存稳定后，平均6.7s
- 优化后，数据缓存稳定后，平均6s

```
1663                if (isSingleRow && rowCount > 0) {
1664                    break;
1665                }
1666            }
1667
```

代码分析-等价语句

- 先将语句进行等价变换，之后再优化src/org/hsqldb/SetFunction.java中L305的代码
- 优化后，数据缓存稳定后，平均6.3s

代码分析-等价语句

```sql
select o_orderpriority, count(*) as order_count
from orders
where
    o_orderdate >= date '1993-06-01'
    and o_orderdate < date '1993-06-01' + interval '3' month
    and exists (
        select *
        from lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority;
================

select o_orderpriority, count(distinct o_orderkey) as order_count
from orders inner join lineitem on (o_orderkey = l_orderkey)
where
    orders.o_orderdate >= date '1993-06-01'
    and orders.o_orderdate < date '1993-06-01' + interval '3' month
    and lineitem.l_commitdate < lineitem.l_receiptdate
group by
    o_orderpriority
order by
    o_orderpriority;
```

代码分析-等价语句

```
302             if (count > 0 && isDistinct && type.isCharacterType()) {
303                 Object[] array = new Object[distinctValues.size()];
304
305                 distinctValues.toArray(array);
306                 // changed by huangchao
307                 /*
308
309                 SortAndSlice sort = new SortAndSlice();
310
311                 sort.prepareSingleColumn(0);
312                 arrayType.sort(session, array, sort);
313
314                 count = arrayType.deDuplicate(session, array, sort);
315                 */
316                 count = distinctValues.size();
317             }
318
```

O_ORDERPRIORITY	ORDER_COUNT
1-URGENT	10640
2-HIGH	10501
3-MEDIUM	10396
4-NOT SPECIFIED	10523
5-LOW	10430