<HSQLDB 优化大作业最终报告>

2015013200 软工51 黄超
2015080121 软工52 李在弦

实验目的

(1) 分析 HSQLDB 源代码，研究 HSQLDB 的查询处理机制
(2) 阅读文献，调研查询优化的相关技术
(3) 实现优化查询方案
(4) 对比优化前后的查询效率和资源开销

实验环境

操作系统: OSX
处理器：2.9 GHz Intel Core i7
内存：16 GB 2133 MHz LPDDR3
IDE: Eclipse
TPC-H Size = 1GB

数据分析

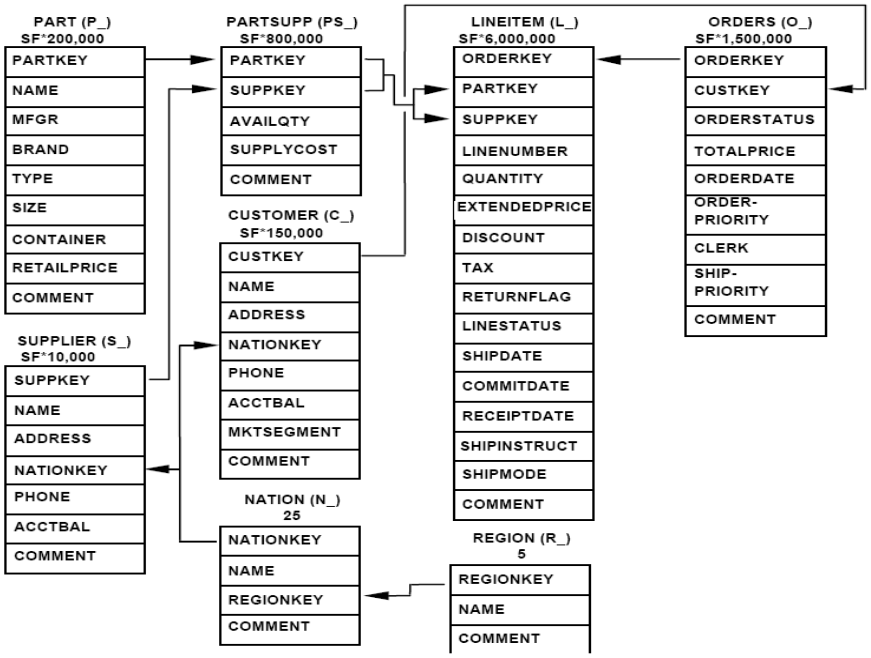
1. TPC-H

TPC-H是由TPC(Transaction Processing Performance Council)
事务处理性能委员会公布的一套针对数据库决策支持能力的测试基准，通过模拟数据库中与业务相关的复杂查询和并行的数据修改操作考察数据库的综合处理能力，获取数据库操作的响应时间和每小时执行的查询数指标

1.1 TPC-H模型概述

(1) 遵照第三范式（3-NF）。
(2) 模拟零售上市场分析，实现了一个数据仓库。
(3) 共包含8个基本关系/表，其中：
表 REGION 和表 NATION 的记录数是固定的（分别为5条记录和25条记录）。
其它6个表的记录数，则随所设定的参数SF而有所不同，其数据量可以设定从1GB~3TB不等，用8个级别供用户选择。
(4) TPC-H基准测试包括22个查询（Q1~Q22）随机组成查询流
(5) 2个更新（带有INSERT和DELETE的程序段）操作组成一个更新流；查询流和更新流并发执行，查询流数目随数据量增加而增加。

数据库模型见下图



· 每个表格后面的括号内为这个表的列名的前缀

· 箭头指出表与表之间的一对多的关系

· 每个表名下方的数字或公式表示表的行数。一些是SF(Scale Factor)
中的因子，用来获得数据库的大小

1.2 TPC-H的目的

TPC- H 主要目的是评价特定查询的决策支持能力，强调服务器在数据挖掘、分析处理方面的能力。查询是决策支持应用的最主要应用之一，数据仓库中的复杂查询可以分成两种类型：一种是预先知道的查询，如定期的业务报表；另一种则是事先未知的查询，称为动态查询（Ad- Hoc Query）。

通俗的讲，TPC-H就是当一家数据库开发商开发了一个新的数据库操作系统，

采用Ｔｐ Ｃ － Ｈ作为测试基准，来测试衡量数据库操作系统查询决策支持方面的能力.

1.3 TPC-H 模型的评价指标

(1) **主要评价指标**：各个查询的响应时间
即从提交查询到结果返回所需时间

(2) **TPC-H基准测试的度量单位**：每小时执行的查询数**（QphH@size)**
H表示每小时系统执行复杂查询的平均次数
size表示数据库规模的大小，它能够反映出系统在处理查询的能力

2. 数据表分析

```
CREATE CACHED TABLE ORDERS (
    O_ORDERKEY INTEGER NOT NULL,          /* 主键，订单固有编号 */
    O_CUSTKEY INTEGER NOT NULL,           /* 外键引用C_CUSTKEY */
    O_ORDERSTATUS CHAR(1) NOT NULL,       /* 订单状态 */
    O_TOTALPRICE DECIMAL(15,2) NOT NULL, /* 订单总价格 */
    O_ORDERDATE DATE NOT NULL,            /* 订单日期 */
    O_ORDERPRIORITY CHAR(15) NOT NULL,    /* 订单优先级 */
    O_CLERK CHAR(15) NOT NULL,            /* 职员编号 */
    O_SHIPPRIORITY INTEGER NOT NULL,      /* 发货优先权 */
    O_COMMENT VARCHAR(79) NOT NULL        /* 评论 */
    );

ALTER TABLE ORDERS ADD PRIMARY KEY (O_ORDERKEY); /* 添加主键 */
ALTER TABLE ORDERS ADD FOREIGN KEY (O_CUSTKEY) references CUSTOMER; /* 添加外键 */

CREATE CACHED TABLE LINEITEM (
    L_ORDERKEY INTEGER NOT NULL,             /* 外键引用 O_ORDERKEY */
    L_PARTKEY INTEGER NOT NULL,              /* 外键应用 P_PARTKEY，和L_SUPPKEY混合引用(PS_PARTKEY, PS_SUPPKEY) */
    L_SUPPKEY INTEGER NOT NULL,              /* 外键引用S_SUPPKEY，和L_PARTKEY混合引用外键(PS_PARTKEY, PS_SUPPKEY) */
    L_LINENUMBER INTEGER NOT NULL,           /* 行号 */
    L_QUANTITY DECIMAL(15,2) NOT NULL,       /* 数量 */
    L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL, /* 总额 */
    L_DISCOUNT DECIMAL(15,2) NOT NULL,       /* 折扣 */
    L_TAX DECIMAL(15,2) NOT NULL,            /* 税收 */
    L_RETURNFLAG CHAR(1) NOT NULL,           /* 返回标志 */
    L_LINESTATUS CHAR(1) NOT NULL,           /* 状态 */
    L_SHIPDATE DATE NOT NULL,                /* 发货日期 */
    L_COMMITDATE DATE NOT NULL,              /* 提交日期 */
    L_RECEIPTDATE DATE NOT NULL,             /* 接收日期 */
    L_SHIPINSTRUCT CHAR(25) NOT NULL,        /* 运输通知 */
    L_SHIPMODE CHAR(10) NOT NULL,            /* 运输模式 */
    L_COMMENT VARCHAR(44) NOT NULL           /* 评论 */
    );

ALTER TABLE LINEITEM ADD PRIMARY KEY (L_ORDERKEY,L_LINENUMBER); /* 添加主键 */
ALTER TABLE LINEITEM ADD FOREIGN KEY (L_ORDERKEY) references ORDERS; /* 添加外键 */
ALTER TABLE LINEITEM ADD FOREIGN KEY (L_PARTKEY,L_SUPPKEY) references PARTSUPP; /* 添加外键 */
```

Integer：意思是必须为整数（比如值的增长为1），取值范围为-2,147,483,646到2,147,483,647。

Char：size N是用来存储一个固定长度为N的字符串类型

Decimal：意思是必须能够描述取值范围从-9,999,999,999.99到+9,999,999,999.99内、数值增长为0.01的所有有理数。

Date：是一个可以被描述为YYYY-MM-DD的值，它所有的字符均为数字。一个日期必须能够描述连续的14年里的每一天，但是对日期的内部描述没有特殊的要求。

2.1 ORDERS表

ORDERS 表示订单的信息，总共有1500000行数据

主键：O_ORDERKEY

外键：O_CUSTKEY（顾客编号）

注释：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”。

2.1.1 属性分析

O_ORDERKEY：订单固有编号

数据范围（1 ≤ O ≤ 6000000）

O_CUSTKEY：顾客编号

数据范围（1 ≤ C ≤ 149999）

O_ORDERSTATUS：订单状态（O,F,P）

O：732044个

F：729413个

P：38543个

O_TOTALPRICE：买单总额

数据范围（857.71 ≤ T ≤ 555285.2）

平均值 151219.5376

O_ORDERDATE：订单日期

数据范围（“1992年1月1号” ≤ O ≤ “1998年8月2号”）

O_ORDERPRIORITY：订单优先级

1-URGENT：300343个

2-HIGH：300091个

3-MEDIUM：298723个

4-NOT SPECIFIED：300254个

5-LOW：300589个

O_CLERK：职员编号

数据范围（“Clerk#000000001” ≤ C ≤ “Clerk#000001000”）

O_SHIPPRIORITY：发货优先权

1500000个数据都为0

O_COMMENT：顾客留的评论

以字符串性质保存数据

2.2 LINEITEM表

LINEITEM 表示在线商品的信息，总共有6001215行数据

主键：混合的主键（L_ORDERKEY，L_LINENUMBER）

外键：L_ORDERKEY（来自哪个订单），L_PARTEKY（来自于哪个零件），L_SUPPKEY（来自于哪个供货商）

2.2.1 属性分析

L_ORDERKEY：订单固有编号

数据范围（1 ≤ O ≤ 6000000）

L_PARTKEY：零件固有编号

数据范围（1 ≤ P ≤ 200000）

L_SUPPKEY：供货商固有编号

数据范围（1 ≤ S ≤ 10000）

L_LINENUMBER：行号（1，2，3，4，5，6，7）

1：1499686个

2：1285828个

3：1071394个

4：857015个

5：643287个

6：429070个

7：214621个

L_QUANTITY：数量

数据范围（1 ≤ Q ≤ 50）

平均值 25.50797

L_EXTENDEDPRICE：总额

数据范围（901 ≤ E ≤ 104949.5）

平均值 38255.13848

L_DISCOUNT：折扣

数据范围（0 ≤ D ≤ 0.1）

平均值 0.04999943

L_TAX：税收

数据范围（0 ≤ T ≤ 0.08）

平均值 0.040013509

L_RETURNFLAG：返回标志（N，R，A）

N：3043852个

R：1478870个

A：1478493个

L_LINESTATUS：状态（O，P）

O：3004998个

P：2996217个

L_SHIPDATE：发货日期

数据范围（“1992年1月2日”≤ S ≤“1998年12月1日”）

L_COMMITDATE：提交日期

数据范围（“1992年1月31日”≤ S ≤“1998年10月31日”）

L_RECEIPTDATE：接收日期

数据范围（“1992年1月4日”≤ S ≤“1998年12月31日”）

L_SHIPINSTRUCT：运输通知信息

DELIVER IN PERSON：1500048个

TAKE BACK RETURN：1499758个

NONE：1500862个

COLLECT COD：1500547个

L_SHIPMODE：运输模式

TRUCK：856998个

MAIL：857401个

REG AIR：856868个

AIR：858104个

FOB：857324个

RAIL：856484个

SHIP：858036个

L_COMMENT： 评论

查询语句分析

订单优先权检查查询

这个查询可以让我们了解订单优先权系统工作得如何，并给出顾客满意度的一个估计值。

```
select
    o_orderpriority, /* 订单优先级 */
    count(*) as order_count /* 订单优先级计数 */
from
    orders /* 单表查询 */
where
    o_orderdate >= date ':1'
    and o_orderdate < date ':1' + interval '3' month /* 指定订单的时间段：某三个月 */
    and exists ( /* 子查询 */
        select
            *
        from
            lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority /* 按订单优先级分组 */
order by
    o_orderpriority; /* 按订单优先级排序 */
```

查询语句得到订单优先级统计值。计算给定的某三个月的订单的数量，在每个订单中至少有一行由顾客在它的提交日期之后收到。

这个查询语句的特点是：带有分组、排序、聚集操作、子查询并存的单表查询操作。子查询是相关子查询。

启发

在整个查询过程中，子查询部分会有重复查询。为了提高效率，我们可以用临时表先保存 selcect * from lineitem where l_commitdate < l_receiptdate 查询语句的结果，然后以后在子查询部分中可以直接对它进行 l_orderkey = o_orderkey 的情况的查询。执行原来的查询语句的话，在子查询部分中每次要对整个lineitem表进行查询，但是如果使用刚才说的方法的话，只对临时表进行查询就可以，这样会节约很长查询时间。

查询结果

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

优化思路与具体实现

优化分析与思路

- 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

具体实现

1. **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。

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

   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

2. **等价转换**

   根据论文《On Optimizing an SQL-like Nested Query》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。

```
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;
```

3. **distinct语句优化**

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。

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

```
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 
```

优化后，数据缓存稳定后，平均6.3s

参考文献

《On Optimizing an SQL-like Nested Query》 (WON KIM,1982)

组员分工和贡献

李在弦：数据分析（数据表分析，查询语句分析），优化实现，讨论优化思路

黄超：优化分析，优化实现，设计导入数据代码，讨论优化思路，搜索优化相关文献