# THOR问题思考纪要

## 数据库中使用一些轻量级压缩

在数据库中很少使用重量级压缩，因为目前主要的瓶颈是在CPU，而不是磁盘，所以使用轻量级压缩要更合适一些，比如：数值压缩（Numeric Compression）, 字符串压缩（String Compression），字典压缩（Dictionary-based Compression）

数值压缩：整数类型存储的数据通常远远小于整数边界，数值压缩就是主要是信息熵，压缩数据的宽度，这种简单的压缩方式效率很高，而且解压的负担非常小。通常可以利用一段数据中的最大最小值来计算一个偏移，然后再对数据进行数值压缩

## 表达式运算溢出的问题

表达式运算时涉及到类型升级的问题，如果进行升级，则会加大计算的成本，但如果不升级数据类型，特别是乘法运算时，很容易出现数据溢出问题。

逻辑运算不会存在数据类型升级问题；

对于整型：

加法，减法，乘法：升级一个数据类型，最大支持64位

除法，不需要升级数据类型

对于浮点型：

计算时全部使用DOUBLE类型进行运算

数值类型提升 int8 > int16> int32 > int64 >? double

float > double

int64 和 float，则转换成double

## 常量在向量表达式运算时是否需要优化？

常量一般时和变量进行运算的，否则就可以再编译期进行优化，因此不需要单独针对常量向量做优化

## 是否需要针对NULLMAP做优化？

第一阶段不考虑NULL值，第二阶段考虑

## 表达式计算是否需要针对各种数据类型优化？

实际测试的结果表明，整形，浮点型的算术运算，不同精度类型的速度实际上差别不大（甚至INT和LONG要比SHORT，char的速度要快一些）。

差别主要是在存储空间上，主要是对CACHE更友好，但因为目前的CACHE是足够容纳最大数据类型的SEGMENT的，因此针对不同的数据类型优化算术运算的意义不大。

在SIMD的情况下会好些，但是需要做很多而外的优化，实际优化代码的工作量较大，价值不高。

因此算术运算，整形建议直接提升至BIGINT类型？，浮点直接升级为DOUBLE类型

逻辑运算不涉及类型升级；

## 关于SIMD指令

SIMD指令，目前最新的AUX2支持到256位，未来的AVX 512估计在2015年能够普及，目前主要的使用方式是使用 汇编，intrinsic，以及编译器自动优化等方式，目前用的较多的是使用intrinsic，intrinsic和汇编是一一对应的，但有些功能可能不完整， intrinsic指令使用的要求较高，太过于涉及底层细节，可移植性较差，编译器自动优化的方式，之前有论文说效果较差，后面需要实测一下。

和SIMD类似的技术，还有SIMT，SMT，这两个技术目前都是在GPU上的，目前CPU还不支持，后续CPU+GPU混合计算可能是个趋势，另外，现在内存数据库（内存计算）的技术发展实际上很多事可以参考图形处理的一些相关经验。

如何设计最后一段的数据扫描

通常扫描都是直接整段进行扫描的，这样是不需要任何参数，但是最后一段，因为不会填满一段，所以处理方式就会比较特殊；

有两种方式，一种就是生成两个表达式树，一个是针对SEGMENT满的情况，一个是针对没满的情况，运行时判断是否已经到达最后一段，如果没有则使用前面加速的表达式计算，最后一段则使用动态的计算方式；

另外一种就是直接使用动态方式，先判断段内的大小是否需要用SIMD加速

结论，先使用简单的方式，性能影响应该是很小的；

## 关于表达式的缓冲

一种方式就是在表达式节点中，设置固定的缓冲，比如成员变量，这样用起来比较方便，但问题是空间消耗大，如果是复杂SQL的话，表达式本身就会占用很大的空间；

另外一种就是计算的空间由外部给，表达式本身没有固定的缓冲，这样一个SQL，一个线程仅仅使用一个BUFFER就可以搞定；

但带来的问题是，对于常量来说，会存在反复的取值赋值运算，这个实际上是浪费的；

## 关于执行节点的缓冲

执行节点可能会在不同的线程中执行，因此执行节点的结果的内存是由外部提供的，以避免多余的拷贝

执行节点的缓冲和表达式计算的缓冲不太一样，执行节点设计上考虑了多线程执行的场景，而表达式计算则目前不考虑多线程执行，而考虑多线程会增加缓冲的复杂度，所以两者的缓冲设计上差别较大。

## GROUP BY的中间结果

进行分组聚合的结果最好是行模式，这样性能是最好的，但输出结果如果行方式的，会和其他的表结构模型上有区别，会增加复杂度。

## 关于最大最小值HINT

在使用表达式做范围过滤时，目前不考虑B树索引的，因此，如果全表扫描，性能会比较差，

一种折中的方法就是在数据加载时，设置段的最大最小值的HINT，这样一方面可以比较快捷的实现范围查找，另外一方面可以比较方便数据压缩。

最大最小值的HINT可以统一存放，可以用一个小的辅助表来存放，这样在使用WHERE做表达式过滤时可以先扫描这个表，用最大最小值过滤一遍；只要最大最小中的任一条件满足，及需要扫描实际的数据段，否则可以跳过该段；

## 关于顺序取值和随机取值

表字段的顺序取值和随机取值的区别是比较大的，

如果是顺序取值，特别是正好是一段未压缩的数据时，直接取值就可以，如果是随机取值，则必须提供一个缓冲区，需要先将零散的数据拷贝到缓冲区才能进行后续的操作；

顺序取值的话可以做一些优化，通常随机取值时需要赋值所有的ROWID，而顺序取，则在第一个位置设置一个特殊值，当遇到这个特殊值，就表示是顺序取值；