基数估计

汇报人：woodyuan(原嘉)

导师：flacroxing(邢家树)

背景

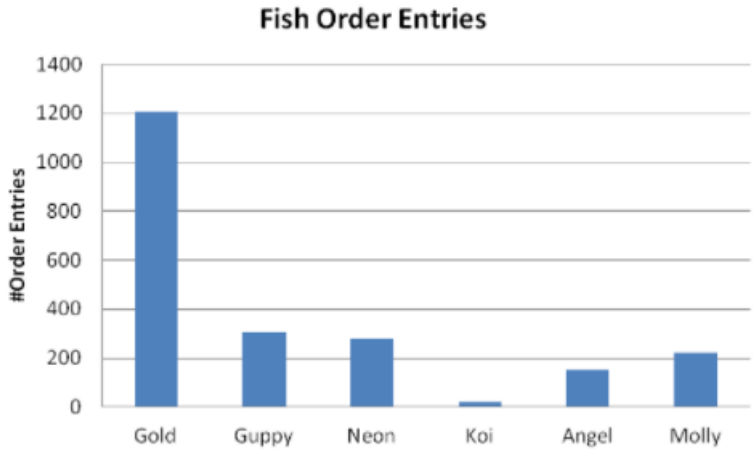
列的基本统计信息能够在一定程度上反映数据的情况：

1. table一共有多少行数据
2. 每一列不同值的个数(NDV)
3. 每一列的最大值，最小值，空值占比，列值的平均长度

但是当数据分布不均匀的时候，基本统计信息是不足够反映数据分布情况的，数据库优化器会默认认为目标列的数据量在其最小值和最大值之间是均匀分布的（最小值最大值不准确会导致谓词越界），并且会按照这个均匀分布原则来计算目标列的where查询条件后的结果集的cardinality,进而据此来计算成本值并选择执行计划，如果目标数据列分布不均匀，那么cardinality的估计就是错误的，优化器产生的执行计划也不会是最优的。这时候需要数据直方图，描述列上的数据分布情况

有两种类型的skew(数据分布不均)：nonuniform value repetition and nonuniformity in range

- nonuniform value repetition：列中至少有一个值数量是明显超过其他值的数量的，并且对该列的查询涉及到equality和equijoin



- nonuniformity in range：对列的查询涉及到range

对于上述两种数据分布不均的情况，在我们有了直方图后，就能够对目标列的where查询条件后的结果集的cardinality进行预估。例如表中的某个列上数据占据了整个表的80%（数据分布倾斜），相关的索引就可能无法帮助减少满足查询所需的I/O数量，此时还不如选择全表扫描。又例如我们有一个五项的表联接，其结果集只有 10 行。 应当以一种使第一个联接的结果集（集合基数）尽可能小的方式将表联接起来，通过在中间结果集中携带更少的负载，查询将会运行得更快。为了使中间结果最小化，优化器尝试在 SQL 执行的分析阶段评估每个结果集的集合基数，拥有直方图将会极大地帮助优化器作出正确的决策

Mysql对统计信息的使用

索引列Cardinality(NDV)

当query涉及到的列有多个索引的时候，通常认为Cardinality大的索引能够过滤掉更多的数据，所以根据索引的Cardinality大小来选择使用哪个索引

但是当数据分布不均匀的时候，可能存在这种情况·，虽然NDV较大，但是列中某个值占比高，假设sql语句where a=10,b=20，虽然列a的NDV大于列b的DNV，但是a=10命中的行数远大于b=20命中的行数，此时走a这一列的索引是不如走b这一列的索引的。

直方图

对于range查询，mysql优化器更偏向于与使用range optimizer 的估计行数而不是直方图的估计行数

如果列上加了索引，那么等值查询优化器会使用index dives而不是直方图

可见Mysql对直方图的使用还没有很广泛，同样Mysql建立直方图的策略不管是从采样还是直方图类型来看都没有做的很好

调研对比

MySQL

索引Cardinality(NDV)

通过采样的方式来计算Cardinality：

默认的InnoDB存储引擎对8个叶子节点Leaf Page进行采样。采样过程如下

取得B+树索引中叶子节点的数量，记为A

随机取得B+树索引中的8个叶子节点，统计每个页不同记录的个数，即为P1，P2....P8

通过采样信息给出Cardinality的预估值:Cardinality=(P1+P2+...+P8)*A/8

直方图数据采样

1. 参数 **histogram_generation_max_mem_size**限制了采样数量：

   rows_in_memory = histogram_generation_max_mem_size / row_size_bytes
2. 采样是整个表上的均匀采样，mysql使用了`SYSTEM` sampling，是一种page-level的采样策略。

等宽直方图(Singleton Histogram)

当数据中不同值的个数小于等于Bucket个数时，建立等宽直方图

```
{
  "buckets": [
    [--第一个桶(bucket)中的统计信息
      1,--桶中的值
```

```
      0.3333333333333333--累计占比
    ],
    [
      2,
      0.6666666666666666
    ],
    [
      3,
      1
    ]
  ],
  "null-values": 0,--数据类型
  "last-updated": "2017-03-24 13:32:40.000000",--上次更新时间
  "sampling-rate": 1,--采样率，1为全表采样
  "histogram-type": "singleton",--直方图类型
  "number-of-buckets-specified": 128,--桶个数
  "data-type": "int",--数据类型
  "collation-id": 8
}
```

selectivity

Cardinality(估计行数)=total rows * selectivity

对于等宽直方图，selectivity的计算直接与累计频率相关，例如给出一个value，要计算它的累计频率，则找到第一个保存着不小于value值的桶，然后返回该桶的累计评率。范围查询同样类似

等高直方图(Equi_HeightHistogram)

当数据中不同值的个数大于Bucket个数时，建立等高直方图

与等宽直方图不同的地方在于，等高直方图尽量保证所有bucket高度一样，一个bucket中会有多个值

所以等高直方图每一个Bucket保存最大值，最小值，不同值的个数，累计频率等信息

```
{
  "buckets": [
    [       --第一个桶(bucket)中的统计信息
      1,    --最小值
      9710, --最大值
      0.009996666666666666, --累积占比，0.99%
      2571  --第一个桶中累积几个值
    ],
...
    [       --第100个桶中的统计信息
      989875,
      999994,
      0.9996666666666667,  --因为该列包含部分NULL值，所以这里不是1.0（100%）
      2580
    ]
  ],
  "data-type": "int",    数据类型
  "null-values": 0.0003333333333333333, --是否包含NULL值，或者NULL值的占比
  "collation-id": 8,
  "last-updated": "2020-04-21 07:21:53.084054",  --直方图最后更新时间
  "sampling-rate": 1.0,  --采样比例 100%
  "histogram-type": "equi-height",  --等高直方图
```

```
  "number-of-buckets-specified": 100  --共有100个桶
}
```

严格遵守“等高”的原则进行分桶会使某些值出落桶的边界上，而导致同一个值出现在两个不同的桶中。所以 MySQL 在实现中对其进行了一定的修改：如果将一个值加入桶中会导致桶中数据频次超过总行数的 1/N，则根据哪种情况更接近 1/N 将该值放入其中该桶或下一个桶中。

selectivity

对于等高直方图

由于一个bucket中可以存在很多个值，考虑进了桶内最大最小值相差大小(桶的宽度)来计算

- 单点查询selectivity

```
selectivity = (bucket_frequency / num_distinct) * value_probability
其中
   bucket_frequency为该桶所有值的占比(该桶的累计频率-前一个桶的累计频率)
   num_distinct为该桶中不同值的个数
   value_probability=num_distinct / (桶中最大值-桶中最小值+1)
简化后
selectivity=bucket_frequency / (桶中最大值-桶中最小值+1)
```

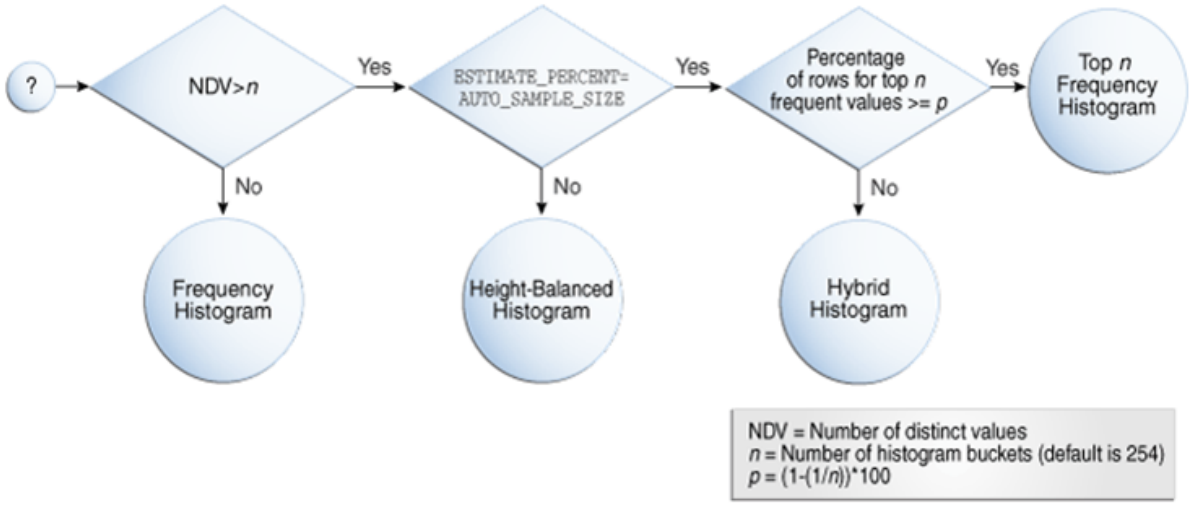
- less_than_equal_selectivity

  计算方式略有不同，具体可以看源码

Oracle

如何选择直方图类型

Oracle通过几条不同的策略选择建立不同的直方图



1. NDV：不同值的个数
2. n: 直方图Bucket的个数，默认值254
3. p=(1-(1/n)) * 100，当n=254，则p=99.6
4. estimate_percent 默认值即为AUTO_SAMPLE_SIZE

Cardinality Algorithm

Endpoint Numbers and Values

- Endpoint Numbers能够唯一标识一个桶，在频率直方图和hyrid直方图中为累计频次，在等高直方图中为0-n的桶编号
- Endpoint Values每个桶中的最大值

Popular and Nonpopular

- Popular Values:首要条件是该value为Endpoint Value

  对于频率直方图，当前桶中Endpoint Value为Popular Value的条件是当前桶的Endpoint Number减去前一个桶的Endpoint Number大于1.第一个桶中不含Popular Value

  对于等高直方图，与频率直方图不同的地方在于第一个桶中含有Popular Value

  对于hybrid直方图，已经保存了其他信息用于判断一个值是不是Popular Value

```
cardinality of popular value =
  (num of rows in table) *
  (num of endpoints spanned by this value / total num of endpoints)
```

cardinality为该值的估计数量

num of endpoints spanned by this value为popular value横跨桶的个数

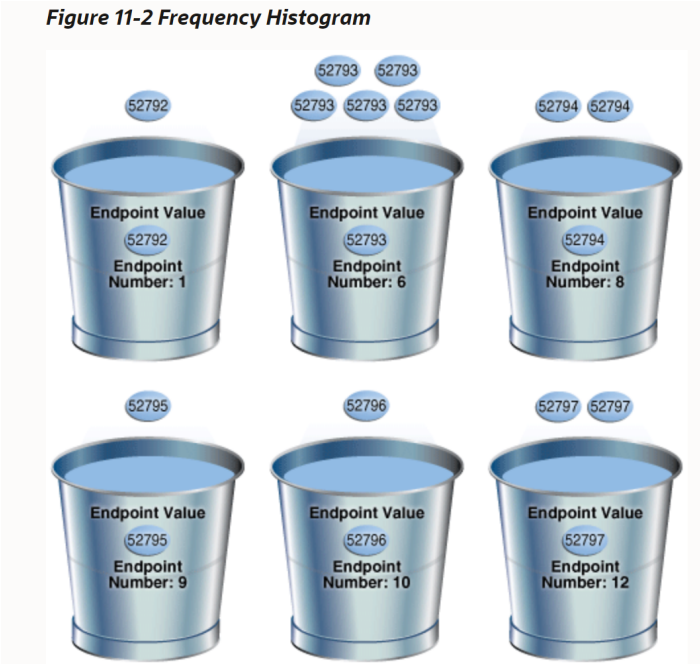
total num of endpoints为桶的个数

- Nonpopular Values：非Popular Values

```
cardinality of nonpopular value =
  (num of rows in table) * density
```

Frequency Histograms

Figure 11-2 Frequency Histogram

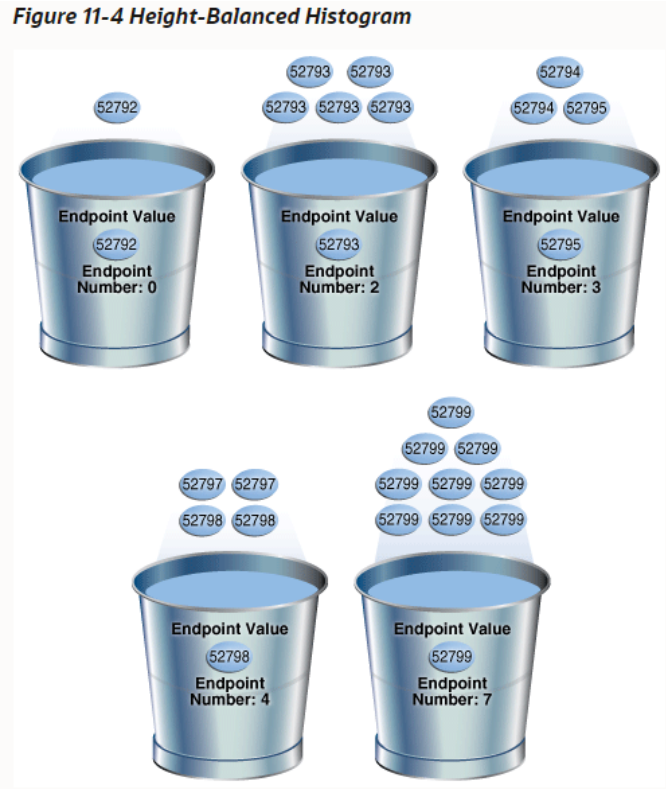


Top Frequency Histograms

忽略掉statistically insignificant的nonpopular values，为其余数据建立Frequency Histograms

Height-Balanced Histograms

Figure 11-4 Height-Balanced Histogram



规定桶的数量后，首先将数据均匀的分到每个桶中，然后进行下面两个步骤

1. 要求最小值为一个桶的endpoint value，最大值为最后一个桶的endpoint value，所以新建0号桶，将52792放入0号桶
2. 将具有相同endpoint value 的桶进行压缩，所以1,2号桶压缩合并为2号桶，5,6,7号桶压缩合并为7号桶

最终3,4号桶中不含有popular value

Hybrid histogram

当estimate_percent 为默认值AUTO_SAMPLE_SIZE并且TOP-N的占比不能超过p值，则建立**Hybrid histogram**

1. 例如列中数据如下

Figure 11-5 Coins

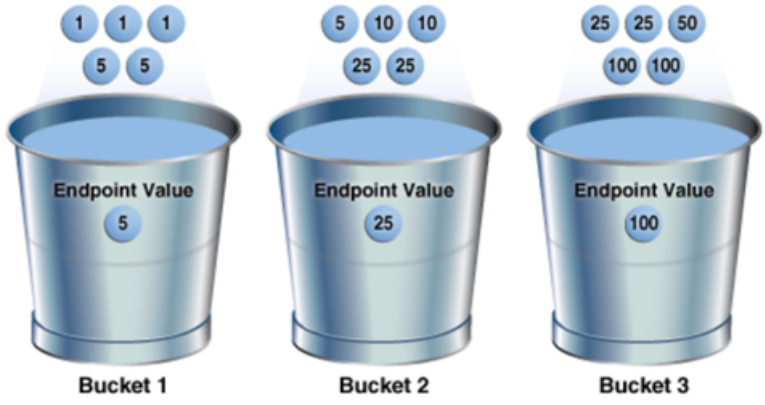
Description of "Figure 11-5 Coins"

You gather statistics for this table, setting the `method_opt` argument of `DBMS_S`

`coins` column into three buckets, as shown in the following figure.

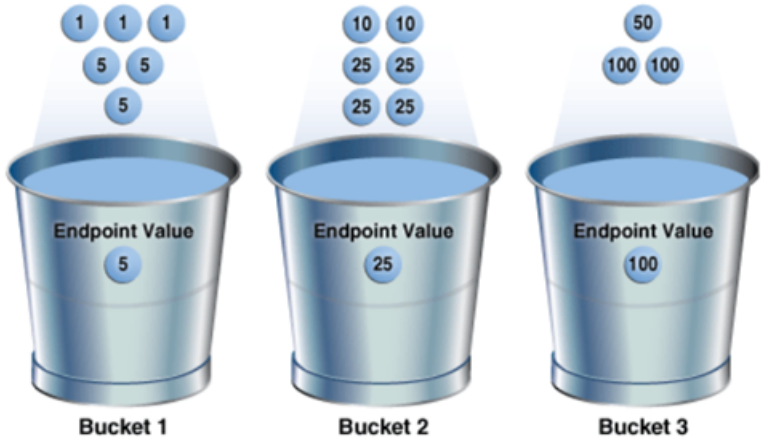
2. 根据Bucket的数量将数据划分到桶中

Figure 11-6 Initial Distribution of Values



Description of "Figure 11-6 Initial Distribution of Values"

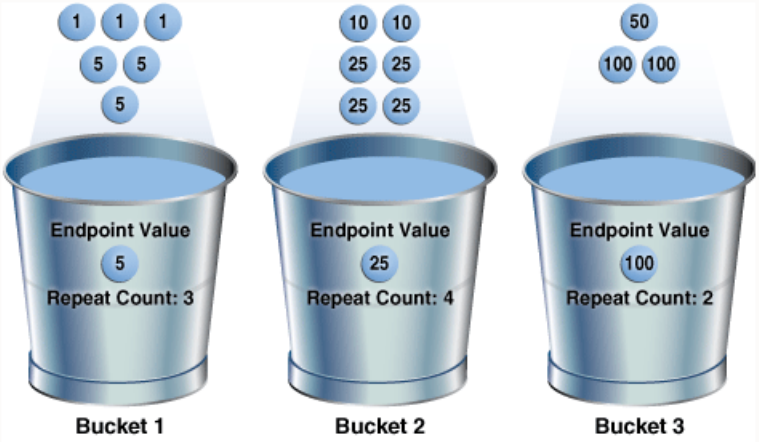
3. 从第一个桶开始，将其他桶中出现的当前桶最大值移入当前桶（具体的建立方法略有吧不同，后面会详细讨论）



Description of "Figure 11-7 Redistribution of Values"

hybrid还保存Endpoint value出现的次数(endpoint_repeat_count)，用来做基数估计

Figure 11-8 Endpoint Repeat Count



AUTO_SAMPLE_SIZE

- auto sample size algorithm使用full table scan来收集基本统计信息(空值的数量，列平均长度，最大值，最小值，总行数，NDV)
- 高效算法用来估算NDV和TOP-N freq，并且Frequency and top frequency histograms会随着基本统计信息的收集同时建立
- Hybrid Histogram使用采样数据的方法来建立，与基本统计信息的收集解耦

WHY choose Hybrid Histogram

由上面的公式可以知道，cardinality of non-pop value

```
cardinality of nonpopular value =
  (num of rows in table) * density
```

其中density为NewDensity

```
NewDensity = (num_rows – popfreq)/((NDV – popCnt)* num_rows)
```

- NDV:不同值的个数
- popfreq: popular value的总频次
- popCnt: popular value的个数
- num_rows: 总行数

根据计算popular value和non-popular value的cardinality的公式来看，它们的结果可能非常不一样，通常non-popular value的估计值远小于popular value的估计值。但是它们实际的数量可能非常相近。考虑下面一种情况:

当一个value的数量刚好比bucket的容量大一个的时候，该值会被设为popular value，而另一个value的数量刚好比bucket的容量小一个的时候，该值会被设为non-popular value。这样它们的cardinality会选择不同的计算方式，尽管实际上它们的数量相差很小，但是估计值会相差很大。而Hybrid Histograms解决了这个问题

How Hybrid Histogram solve this problem

Hybrid Histogram不只是用Endpoint Number的相差值来判断一个值是否是popular value，而是用上面提到的endpoint_repeat_count

如果endpoint_repeat_count>average bucket size，则被认为是popular value

selectivity的计算：

- 如果是popular value :

  ```
  selectivity = endpoint_repeat_count/(not null value row number)
  ```

- 如果是non-popular value并且不是endpoint value

  ```
  selectivity = NewDensity = (num_rows – popfreq)/((NDV – popCnt)* num_rows)
  ```

- 如果是non-popular value并且是endpoint value

  ```
  selectivity = Greatest(NewDensity, endpoint_repeat_count/(not null value
  row number))
  ```

Automatic Histogram Creation

oracle设置了Automatic Histogram Creation为AUTO后，能够通过之前query语句的使用情况和数据的分布情况来综合考虑是否建立直方图

Column Usage

oracle保存历史query predicate or join信息，例如下面STAFF表中列HIRE_DATE用了一次RANGE查询，当一列被query语句用作predicate or join后，就成为candidate column，之后再收集基础统计信息后，可能会被建立直方图

```
DBMS_STATS.REPORT_COL_USAGE(USER,'STAFF')
------------------------------------------------------------------------------
LEGEND:
.......
EQ         : Used in single table EQuality predicate
RANGE      : Used in single table RANGE predicate
LIKE       : Used in single table LIKE predicate
NULL       : Used in single table is (not) NULL predicate
EQ_JOIN    : Used in EQuality JOIN predicate
NONEQ_JOIN : Used in NON EQuality JOIN predicate
FILTER     : Used in single table FILTER predicate
JOIN       : Used in JOIN predicate
GROUP_BY   : Used in GROUP BY expression
.............................................................................
##############################################################################
COLUMN USAGE REPORT FOR ADHOC.STAFF
...................................
1. HIRE_DATE                          : RANGE
##############################################################################
```

在下面这幅图col1，txtcol成为candidate column

```
Query:                                                                  Column
usage:

select sum(amount) from sales where col1 = 10;                          [EQ]
select sum(amount) from sales where col1 != 10;                         [recorded
as EQ]
select sum(amount) from sales where col1 > 10;                          [RANGE]
select sum(amount) from sales s, customers c where s.col1 =  c.col1;
[EQ_JOIN]
select sum(amount) from sales s, customers c where s.col1 != c.col1;    [EQ_JOIN
NONEQ_JOIN]
select sum(amount) from sales where txtcol like 'ALA%';                 [LIKE]
```

Beneficial Histograms

什么情况下适合建立直方图：

- The column has value skew **and** column usage indicates RANGE, LIKE, EQ or EQ_JOIN
- The column has range skew **and** column usage indicates LIKE or RANGE.
- The column has a low number of distinct values (with some repeated values) **and** column usage indicates RANGE, LIKE, EQ or EQ_JOIN

第三条尽管值的分布均匀，但是DNV小，查询不存在的值或者RANGE的范围刚好处于"空洞"处，就会结果集很小，所以仍然适合建立直方图

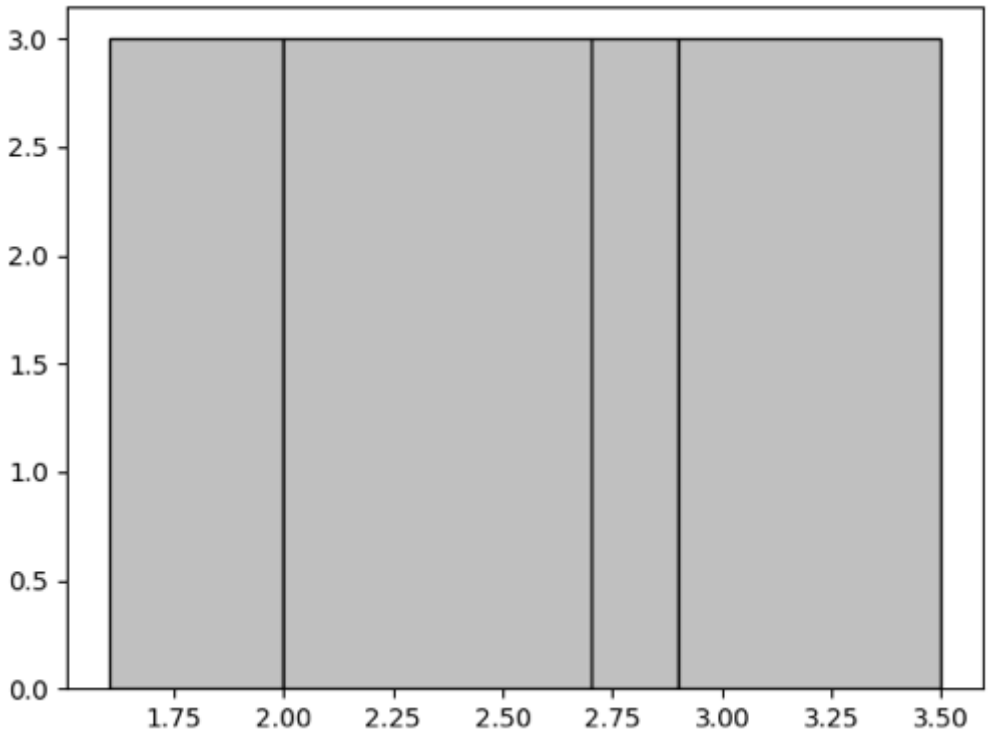
TiDB

当涉及单点查询是使用CM-Sketch，当涉及范围查询时使用Histogram

Introduction to Statistics

Information	Version 1	Version 2
The total number of rows in the table	√	√
Column Count-Min Sketch	√	×
Index Count-Min Sketch	√	×
Column Top-N	√	√ (Maintenance methods and precision are improved)
Index Top-N	√ (Insufficient maintenance precision might cause inaccuracy)	√ (Maintenance methods and precision are improved)
Column histogram	√	√ (The histogram does not include Top-N values.)
Index histogram	√	√ (The histogram buckets record the number of different values in each bucket, and the histogram does not include Top-N values.)
The number of `NULL`s in the column	√	√
The number of `NULL`s in the index	√	√
The average length of columns	√	√
The average length of indexes	√	√

Histogram



建立equal-depth histogram，类似于等高直方图。尽量保证所有值均匀的分配到每个bucket中，同时保证相同值处于处于同一个桶中

TiDB选择等深直方图的原因是相比于等宽直方图，等深直方图在最坏情况下也可以很好的保证误差
Accurate estimation of the number of tuples satisfying a condition

抽样的时候实现了蓄水池抽样算法，用来生成均匀抽样集合。令样本集合的容量为 S，在任一时刻 n，数据流中的元素都以 S/n 的概率被选取到样本集合中去。如果样本集合大小超出 S，则从中随机去除一个样本。举个例子，假如样本池大小为 S = 100 ，从头开始扫描全表，当读到的记录个数 n < 100 时，会把每一条记录都加入采样池，这样保证了在记录总数小于采样池大小时，所有记录都会被选中。而当扫描到的第 n = 101 条时，用概率 P = S/n = 100/101 决定是否把这个新的记录加入采样池，如果加入了采样池，采样池的总数会超过 S 的限制，这时需要随机选择一个旧的采样丢掉，保证采样池大小不会超过限制。

采样完成后，将所有的数据排序，由于知道采样过后总的行数和直方图的桶数，因此就可以知道每个桶的深度。这样就可以顺序遍历每个值 V：

- 如果 V 等于上一个值，那么把 V 放在与上一个值同一个桶里，无论桶是不是已经满，这样可以保证每个值只存在于一个桶中。
- 如果不等于，那么判断当前桶是否已经满，如果不是的话，就直接放入当前桶，否则的话，就放入下一个桶

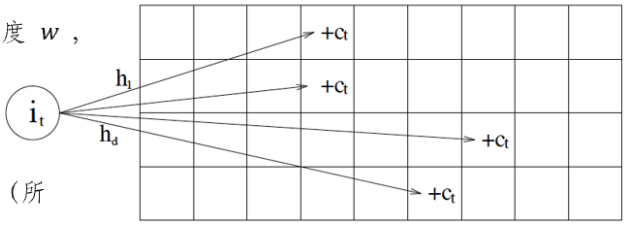
Count-Min Sketch

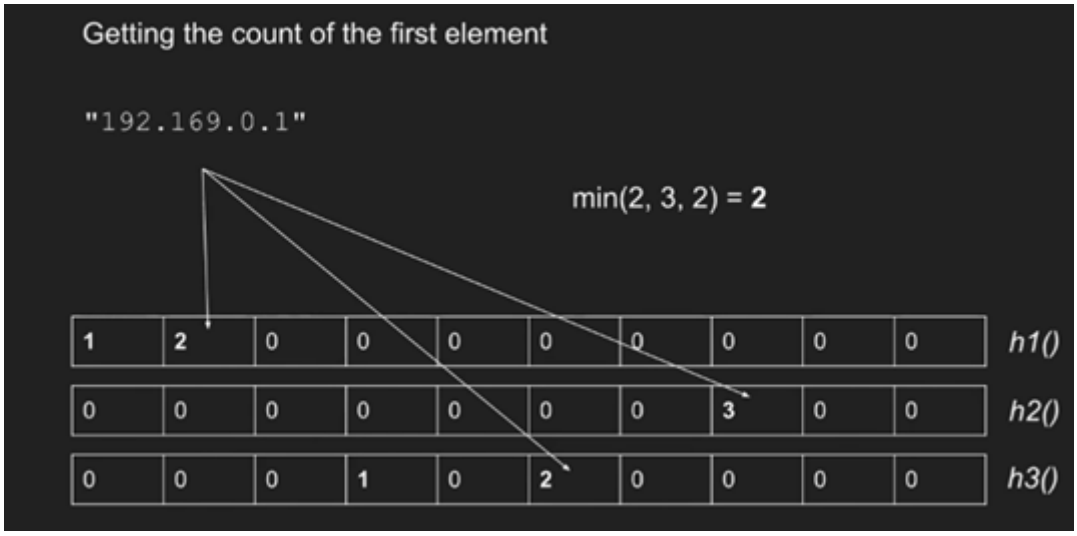
Count-Min Sketch 是一种可以处理等值查询，Join 大小估计等的数据结构，并且可以提供很强的准确性保证

CM-Sketch 的内部数据结构是一个二维数组，宽度 w，深度 d，此外还需要 d 个两两独立的哈希函数 $h_1, \ldots, h_d$

$$w = \left\lceil \frac{e}{\epsilon} \right\rceil, \mathrm{d} = \lceil \ln\left(\frac{1}{\sigma}\right) \rceil$$

两个参数的含义是：在 $1-\sigma$ 的概率下，总误差（所有元素查询误差的之和）小于 ϵ。





如果数据分布不均匀。那么就可以将高频次的value(top N)单独存储，将剩余值通过CM-Sketch存储

小米soar

功能特点

- 跨平台支持（支持 Linux, Mac 环境，Windows 环境理论上也支持，不过未全面测试）
- 目前只支持 MySQL 语法族协议的 SQL 优化
- 支持基于启发式算法的语句优化
- 支持复杂查询的多列索引优化（UPDATE, INSERT, DELETE, SELECT）
- 支持 EXPLAIN 信息丰富解读
- 支持 SQL 指纹、压缩和美化
- 支持同一张表多条 ALTER 请求合并
- 支持自定义规则的 SQL 改写

采样

- 根据论文《Random sampling for histogram construction: how much is enough?》来设计采样率

```
table size n, histogram size k, maximum relative error in bin size f, and
error probability gamma, the minimum random sample size is
      r = 4 * k * ln(2*n/gamma) / f^2
 Taking f = 0.5, gamma = 0.01, n = 10^6 rows, we obtain
      r = 305.82 * k
 Note that because of the log function, the dependence on n is
 quite weak;even at n = 10^12, a 300*k sample gives <= 0.66
* bin size error with probability 0.99
```

具体的做法是采样行数:

wantRowsCount := 300 * common.Config.SamplingStatisticTarget

其中SamplingStatisticTarget默认值为100

再利用tableStatus来获取tableRows的近似值

然后计算factor = wantRowsCount / tableRows

最后利用sql语句；来采样数据

```
where RAND() <= factor LIMIT wantRowsCount
```

采样后的数据重新用insert语句插入到test_table中使用

- 散粒度

  计算每一列的散粒度，用于索引推荐

  - 若列为主键或单列唯一索引，则直接返回1
  - 否则在采样的数据中使用select count(distinct)来获取NDV，在table status中获取总行数
  - 计算散粒度=colNum / float64(rowTotal)

调研结果对比

- Mysql通过控制采样内存大小来控制采样数量，不管是对NDV的统计还是直方图的建立，在数据分布不均匀的情况下会出现很大的误差.
- Oracle对基本列信息(MAX,MIN,总行数,平均列长度),NDV,TOP-N Freq的统计是通过全表采样得到的，达到100%准确率。
- Oracle Frequence Histogram和TOP-N Frequence Histogram的建立是随着NDV和TOP-N Freq的统计同时建立的，同样采用了100%采样，准确率高。TOP-N Frequence Histogram平衡了准确率和建立直方图效率的问题。Hybrid Histogram使用采样方式建立，准确性可能会因为采样比例而浮动
- Mysql等高直方图对点查询的估计会有较大的误差，如果频次高的数据和频次低的数据放到同一个Bucket中，计算是不会区分对待的，会得到相等的Cardinality估计。而Oracle Hybrid直方图不仅区分了popular value(即出现频次高的value)，non-poular value，还把出现频次高的value但是没达到popular value的情况考虑进来了，最后在面对数据分布不均的时候会得到更准确的Cardinality估计
- TiDB范围查询采用了等高直方图，点查询采用了CM-Sketch，相比Oracle的实现，仍然有不足之处

技术选型

- 统计基本统计信息(MAX,MIN,空值数，NDV，列平均长度，总行数)和建立直方图，能够持久化保存这些统计信息
- 参考oracle的实现，根据不同的情况选择建立不同的直方图，舍弃Height-Balanced Histograms
- 参考小米soar的采样数300*k，我们加大采样数量，设定采样数据为10w行，小于10w行全表采样，大于则随机抽样
- 对于大型表，考虑使用并发来加速统计信息的收集和直方图的建立，例如并行排序

难点要点

1. 实现one pass:

   如何设计精妙高效的数据结构和算法来统计NDV和TOP-N Freq，能够从NDV的统计信息中提取中出TOP-N Freq。如果在统计NDV的过程中发现NDV过大，得重新设计数据结构和切换算法来统计TOP-N Freq，那么可以将已有数据传给新数据结构，统计TOP-N Freq就不需要重头扫描。并且在NDV和TOP-N Freq统计结束后就能够从他们的数据结构中得到Frequence Histogram或者TOP-N Frequence Histogram。

2. Histogram Construction in Oracle Database 12c在这篇文章中作者提到了采样比例可能会给Hybrid Histogram的效果带来较大的影响(如不能正确统计到most frequent value)，由于non-popular的计算直接和popular value相关，或许可以用到Top-N Freq的统计信息来尽量包含top-n values

算法汇总

Approximate NDV：

step1.将扫描到的目标统计列的值通过哈希算法，转换为64位的二进制数(也就是8字节的数字)，放入一个叫做概要(synopsis)的数据结构体中。该结构体中包含一个长度为16384的数组和一个level标记(初始化为0)。数组的每个单元用来存放转换后的64为二进制值

step2.扫描下一个列值，转换为哈希64位二进制数，将改二进制数与概要里面已经存在的二进制数做比较。存在相同的则抛弃，否则插入到概要里

step3.新插入的二进制到概要前，如果发现概要已经满了，则丢弃其中部分数据(具体的做法是，第一次丢弃的时候，丢弃概要里面所有首位为 0 的数值；第二次丢弃的是前面2位为 00 的所有数值；以此类推),我们称之为分裂。每分裂以此，level标识自增1

step4.扫描到的新哈希值，如果满足丢弃规则，将不再插入概要，直接丢弃

step5.重复步骤2,3,4的动作，一直到表被扫描完毕

使用 S 表示概要的大小(16384), l 表示级数(level标识，也就是分裂次数), N 表示概要的最终剩余数值数。那么，可以得出一个估算结论：

- 如果l=0 :

$$NDV = N$$

- 如果l>0 :

$$NDV = S * 2^l$$

证明：

假如要计算的是表主键的NDV，那么

- NDV= 表的记录数，步骤1 采取的哈希算法分布均匀的理想情况下，步骤2 不会有被丢弃的情况。
- 概要做第一次分裂的时候，表扫描的记录数为S；这个时候概要会被丢弃 S/2 的记录数（基于哈希算法分布均匀的情况下，首位为 0 和首位为 1 的值数应该是对半开的）。
- 概要做第二次分裂的时候，表扫描的记录数(不包括分裂之前的记录)为 $\frac{1}{2} * S * 2^1$.可以理解为：第一次分裂后一直到第二次分裂时，需要插入$\frac{1}{2}S$ 个哈希数到概要里面，从概率上来说，需要扫描 2^1 倍的数量，才可以填满(因为首位为0的哈希值不会插入)。
- 以此类推，第三次分裂的时候，表扫描的记录数(不包括分裂之前的记录)为$\frac{1}{2} * S * 2^2$。
- 第l次分裂的时候，表扫描的记录数(不包括第一次分裂之前的记录)为$\frac{1}{2} * S * 2^{(l-1)}$
- 第l次分裂完，一直到扫描结束，这个步骤的表扫描记录数的上限应该为$\frac{1}{2} * S * 2^l$

那么，当NDV统计结束的时候，概要的级数为 l，那么表扫描的记录数为

$$(2^0 + 2^1 + 2^2 + \ldots\ldots + 2^l)(\frac{1}{2})S = S * (2^l - \frac{1}{2})$$

假如要计算的是表普通列，那么结果也是和上面的一致，因为列值相同的列哈希计算后，结果也是相同的。相同的列值在 Approximate NDV 算法步骤2，也会被过滤掉。因此结果的计算公式不变

Lossy Counting Algorithm：

参数：

$s \in [0,1]$代表着算法得到的$item$的频次接近$s * N$

ε代表着用户定义的错误率

N代表着数据流的当前长度

f_e代表着$iteme$在数据流中的真实出现频次

D是(e, f, Δ)的合集，用来统计频次，e代表$item$，f代表当前频次，Δ代表f的最大可能误差

f代表着估计的频次

$b_{current}$代表着数据分桶的编号

step1：逻辑上将数据流被分成一个个的bucket，每个bucket的大小为$w = \lceil \frac{1}{\varepsilon} \rceil$,bucket从1开始标号

step2：初始D为空。每次当一个新数据item来了之后，检查D中是否存有这个item，如果有，则将f加1。若没有，则新建$(e, 1, b_{current})$并插入到D中

step3：每次$N \equiv 0 \ mod \ N$（即一个bucket的数据处理完的时候）,将D中(e, f, Δ)满足$f + \Delta \leq b_{current}$的元素删除

当数据流中所有数据处理完成以后，D中所有的元素满足$(s - \varepsilon) * N \leq f \leq s * N$

基于Lossy Counting Algorithm来计算Top-N Freq脚本代码:

```
CREATE GLOBAL TEMPORARY TABLE lossytable (val NUMBER, freq NUMBER, bk NUMBER);

create or replace procedure find_topn(nm_rws  integer,tpk integer)
is
    CURSOR c_chk (cval NUMBER)
    IS
        SELECT   *
          FROM   lossytable
         WHERE   val = cval;

    chk     c_chk%ROWTYPE;
    n       INTEGER := 0;
    delta   NUMBER := 0;
    b       NUMBER := 1;
    --nm_rws  integer:= 1000000; -- Number of rows in the table
    --tpk integer:=254; --top frequent values
    bktsize integer;
BEGIN
    COMMIT;
    Bktsize := nm_rws/tpk;
    FOR c IN (SELECT   * FROM t1)
    LOOP
        n := n + 1;
        OPEN c_chk (c.col);
        FETCH c_chk INTO chk;
        IF c_chk%FOUND
        THEN
            UPDATE   lossytable
               SET   freq = freq + 1
             WHERE   val = c.col;
        ELSE
            INSERT INTO lossytable (val, freq, bk)
              VALUES   (c.col, 1, b-1);
        END IF;
        IF n MOD Bktsize = 0
        THEN
            FOR j IN (SELECT   * FROM lossytable)
```

```
            LOOP
                IF j.freq +j.bk <= b
                THEN
                    DELETE   lossytable
                     WHERE   val = j.val;
                END IF;
            END LOOP;
            b := b + 1;
        END IF;
        CLOSE c_chk;
    END LOOP;
   /*+ deleting infrequent element as delete lossytable where freq=1 */
END;delete lossyTable where val=j.val
        b=b+1
```

Build Hybrid Histogram

1.ROWCNT：初始化为0，每处理一个数据+1

2.BKSIZE：每次循环迭代，为每个bucket计算bucket size：

$popcnt \geq NRB-1$，$BKSIZE = CDN - FREQ/(MNB-1)$

$popcnt < NRB-1$，$BKSIZE = (CDN - POPFREQ - FREQ)/(MNB - POPCNT - 1)$

3.CUMFREQ：$CUMFREQ = CUMFREQ + freq$,值被用来定义`bucket number(endpoint number)`

4.Bkcnt：每创建一个bucket，则+1

5.BKTROWS：统计每一个bucket中数据个数，每次迭代加上`freq`，与`BKSIZE`做比较，决定新的桶要不要建立

6.如果`BKTROWS`比`BKSIZE`大或者未处理的NDV数小于等于未创建的bucket个数或者未处理的数据个数小于等于未创建的bucket个数，那么应当创建新的桶

7.当`Bkcnt=HB-1`那么处理结束，剩下数据全部放到最后一个桶中

下面脚本代码总结了上述过程

```
CREATE GLOBAL TEMPORARY TABLE tmp_hybrid_hist
(
    ep_num     NUMBER, -- endpoint number
    ep_val     NUMBER, -- endpoint value
    ep_rpcnt   NUMBER -- endpoint repeat count
);

/* Note that we can write and execute all of the statements below dynamically,
but I
use the simple SQL statements explicitly to demonstrate the mechanism*/

  CREATE OR REPLACE PROCEDURE CREATE_HH (tnm_rws integer, maxcval number,hbn
integer,smplsize number)
   IS
    val        VARCHAR2 (240);
    freq       NUMBER;
    cumfreq    NUMBER;
```

```
    bktrows    NUMBER;
    bktsize    NUMBER;
    rowcnt     NUMBER;
    cdn        NUMBER;
    ndv        NUMBER;
    popcnt     NUMBER;
    popfreq    NUMBER;
    bktcnt     NUMBER;
    hb         INTEGER := 254; -- we are going to create 254 buckets

    CURSOR cur_hh
    IS
          SELECT   val ep,
                   freq,
                   cdn,
                   ndv,
                   (SUM (pop) OVER ()) popcnt,
                   (SUM (pop * freq) OVER ()) popfreq
            FROM   (SELECT   val,
                             freq,
                             (SUM (freq) OVER ()) cdn,
                             (COUNT ( * ) OVER ()) ndv,
                             (CASE
                                  WHEN freq > ( (SUM (freq) OVER ()) / 254)
                                  THEN
                                      1
                                  ELSE
                                      0
                              END)
                                 pop
                      FROM   (  SELECT
                                        "COL"
                                            val,
                                        COUNT ("COL") freq
                                  FROM   "SYS"."T1" sample (.5500000000) t
                                 WHERE   "COL" IS NOT NULL
                              GROUP BY   "COL"))
        ORDER BY val;
BEGIN
    COMMIT;

    OPEN cur_hh;

    bktrows := 0;
    bktcnt := 0;
    cumfreq := 0;
    rowcnt := 0;
    cdn := 0;
    ndv := 0;

    LOOP
        FETCH cur_hh
        INTO   val,freq,cdn,ndv,popcnt,popfreq;

        EXIT WHEN cur_hh%NOTFOUND;

        rowcnt := rowcnt + 1;
```

```
        IF (rowcnt = 1)
        THEN
            IF (hb - 1 <= popcnt)
            THEN
                bktsize := (cdn - freq) / (hb - 1);
            ELSE
                bktsize := ( (cdn - popfreq - freq) / (hb - popcnt - 1));
            END IF;
        END IF;

        bktrows := bktrows + freq;
        cumfreq := cumfreq + freq;

        IF (   bktrows >= bktsize
            OR (ndv - rowcnt) <= (hb - bktcnt)
            OR rowcnt = 1
            OR rowcnt = ndv)
        THEN

            IF (bktcnt = hb  AND rowcnt <> ndv)
            THEN
                        INSERT INTO tmp_hybrid_hist (ep_num, ep_val, ep_rpcnt)
              VALUES   (tnm_rws /* numrows in the table*/, maxcval /* max col
value*/, 1);

                RETURN;
            END IF;

            bktrows := 0;
            bktcnt := bktcnt + 1;

            INSERT INTO tmp_hybrid_hist (ep_num, ep_val, ep_rpcnt)
              VALUES   (cumfreq, val, freq);
        END IF;
    END LOOP;

    CLOSE cur_hh;


END;
/
SELECT   * FROM tmp_hybrid_hist;
```

整体设计

一期工程：

提供创建直方图，预估基数的API，实现上述技术选型，并且调优

- 创建直方图：用户输入命令 Create Histogram from databaseName.tableName.columnName N要求创建直方图，并指定bucket的个数为N，若不指定则默认值为254，最大值为2048，如果直方图已存在，则不再建立
- 更新直方图：用户输入命令 Update Histogram from databaseName.tableName.columnName N

- 删除直方图：用户输入命令 Delete Histogram from databaseName.tableName.columnName
- 点查询基数估计：用户输入命令 get_equal_cardinality value databaseName.tableName.columnName
- range查询基数估计：用户输入命令 get_range__cardinality value_low contain_low value_high contain_high databaseName.tableName.columnName

  其中contain_low=1，则表示包含下界。contain_low=0，则表示不包含下界。contain_low=-1，则表示下界无穷小

  其中contain_high=1，则表示包含上界。contain_low=0，则表示不包含上界。contain_low=-1，则表示上界无穷大
- 提供show_histogram(databaseName.tableName.columnName)来形象化展示直方图和基础统计信息

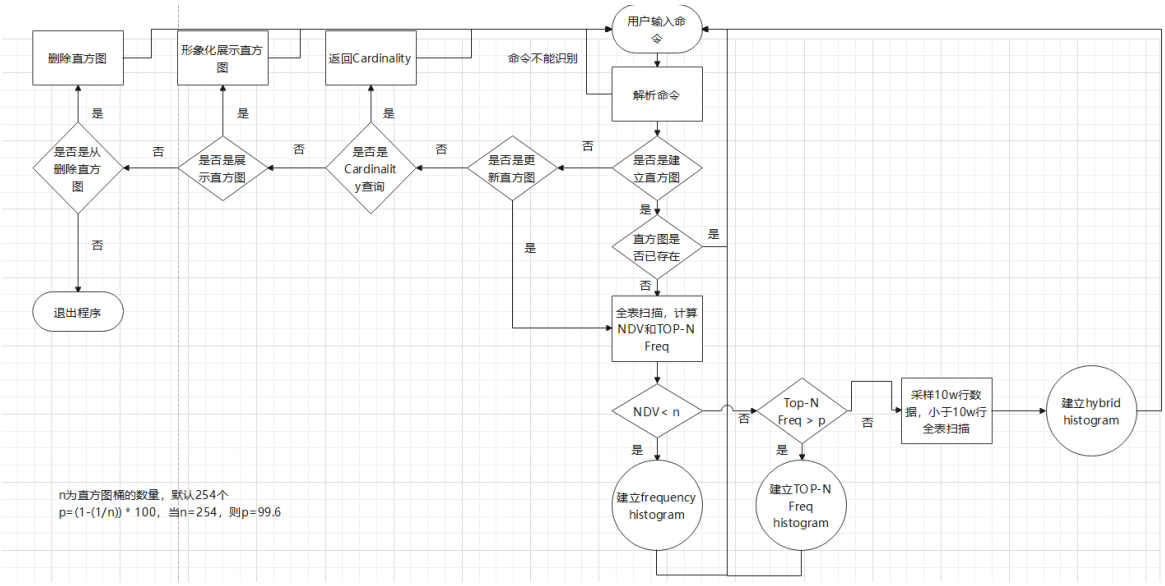
二期工程：

实现上述Automatic Histogram Creation，解析每条sql语句，记录column使用信息，并且根据上述算法自动建立直方图

三期工程：

根据基本统计信息给出索引优化建议，是否建立索引，如果一个查询中有多个索引可以使用，使用哪个索引等

流程图



模块

cmd

与用户交互，读取用户命令并解析上面一期工程中所述的API，然后执行相应的操作，返回相应的结果

sampling

使用github.com/go-sql-driver/mysql包来实现与mysql server的交互，读取数据库，分为全表扫描和部分采样

Gather Statistics

全表扫描，用Approximate NDV，The Lossy Counting Algorithm等算法来估计NDV和TOP-N Freq，同时得到最大值，最小值，空值个数，总行数，列平均长度等信息。

```
type BasicStatistics struct{
   NumOfNULL int
   TotalRows int
   MAX       int
   MIN        int
   AverageLength int
   NDV       int
}
type TotalBasicStatistics struct{
   TotalBS map[string]*BasicStatistics
}
```

用结构体BasicStatistics来存储基本统计信息，用结构体TotalBasicStatistics来存储所有column的基本统计信息，内部是map，key为stringle类型,格式为数据库名.表名.列名—databaseName.tableName.columnName,以便于使用的时候能够快速查找

Build Histogram

- Frequence Histogram AND Top-N Frequence Histogram:

```
type FrequenceHistogram struct{
    BS *BasicStatistics
    distinct_values []interface{}
    Buckets map[interface{}]int
    LastUpdateTime time.Time
    BktSize int
}
type TotalFrequenceHistogram struct{
    TotalFH map[string]*FrequenceHistogram
}
```

利用map来保存Frequence Histogram，bucket的个数可以让用户通过参数设置，默认值为254，最大值为2048，map的key为endpoint value，value为endpoint number

由于Frequence Histogram的NDV<254，则用distinct_values保存下不同的endpoint value,并且排序

利用估计NDV和Top-N Freq的数据结构来建立Frequence Histogram，不必再次遍历数据

同时还要记录下基础统计信息BasicStatistics和Histogram的上次更新时间

BktSize记录bucket的个数

用TotalFrequenceHistogram记录下所有的Frequence Histogram，内部是map，key为stringle类型,格式为数据库名.表名.列名　　databaseName.tableName.columnName,以便于使用的时候能够快速查找

- Hybrid Histogram

```
type Hybridbucket struct{
   EndpointValue interface{}
   EndpointNumber int
   RepeatCount int
```

```
}
type HybridHistogram struct{
   BS *BasicStatistics
   Buckets []Hybridbucket
   LastUpdateTime time.Time
   BktSize int
}
type TotalHybridHistogram struct{
    TotalFH map[string]*HybridHistogram
}
```

由于Hybrid Histogram NDV大于254，则不保存不同的值

利用Hybridbucket来记录EndpointValue，EndpointNumber，RepeatCount

采用上述算法来采样建立Hybrid Histogram，采样默认10w行，小于10w行数据则全表采样

持久化

将直方图以一定格式持久化保存到文件中，下次服务启动的时候可以从文件中读取

计算Cardinality

对于Frequence Histogram和Top-N Frequence Histogram:

- 点查询估计
  - 若value未命中，则直接返回0
  - 若value命中
    - ```
      cardinality of popular value =
        (num of rows in table) *
        (num of endpoints spanned by this value / total num of endpoints)
      ```
    - ```
      cardinality of nonpopular value =
        (num of rows in table) * density

      其中density
      NewDensity = (num_rows – popfreq)/((NDV – popCnt)* num_rows)
      ```
- 范围查询估计：注意边界值包不包含，如果包含，注意边界值是popular value还是nonpopular value

对于Hybrid Histogram:

如果value不命中，则当做是non-popular value并且不是endpoint value处理

- 如果是popular value :

  ```
   cardinality =  (num of rows in table) * (endpoint_repeat_count/(not null value row number))
  ```

- 如果是non-popular value并且不是endpoint value

  ```
  cardinality =  (num of rows in table) *  ((num_rows – popfreq)/((NDV – popCnt)* num_rows))
  ```

- 如果是non-popular value并且是endpoint value

```
 cardinality =(num of rows in table) * Greatest(NewDensity,
endpoint_repeat_count/(not null value row number))
```

测试计划

- 从数据统计的准确度，统计时间，资源消耗三个方面来与Mysql已有实现，oracle已有实现，TiDB已有实现进行对比，预期比Mysql好
- 网上收集各种类型的数据库表来对直方图进行测试，看直方图能否对实际生产过程中各种类型的数据都有较好的估计结果
- 自己创建极端数据来测试，针对TOP-N Frequence Histogram和Hybrid Histogram的特点来进行测试

人时安排

首先实现一期工程

3周时间完成程序代码的编写，2周不断测试调优